鉄道模型を楽しむ
PIC16F マイコン電子工作とプログラミング

角川 裕次

2021 年 9 月 17 日

本書の内容の利用に関する注意事項

- 本書に記載された一部の技術・手法は，特許権により保護されています．
- 本書に掲載された電子回路は，個人による学習・研究目的での製作・利用を想定しています．商業目的での利用をご検討の場合は、事前に著者までご相談ください。
- 本書に掲載されたソフトウェアおよびソースコードは，以下の 2 つのライセンスのもとで提供されます．
  1. GNU General Public License v2 (GPLv2)
     個人による非営利目的 (直接・間接を問わず経済的利益を得ることを目的としない利用) の利用には，GPLv2 ライセンスが適用されます．GPLv2 ライセンスのもとでは，本ソフトウェアに実装された特許技術を含め，学習・研究・個人的な実践における使用・改変・再配布を事前の許諾なく行うことができます．
  2. 商業ライセンス
     営利目的での利用 (製品への組み込み・販売・受託業務・サービス提供等) には，商業ライセンスが適用されます．商業ライセンスのもとでは，GPLv2 のソース公開義務を負うことなく本ソフトウェアおよび特許技術を利用することができます．商業ライセンスの取得については著者までお問い合わせください．

安全上の注意

- 本書の内容は善意に基づき十分に注意を払って作成していますが, 正しさや安全性に関する保証は一切いたしかねます. 組み立てや使用法を誤れば財産・身体・生命等を損なう事故となる場合があります. いかなる結果についても著者は責任を負いませんので, 本書の内容を十分に理解した上で, 各自の判断と責任のもとでご利用下さい.
- 電子工作に際しては, ハンダゴテやニッパーなどの各種工具で火傷や怪我をしないよう十分にご注意下さい.
- ハンダ付け作業の途中で席を離れるときは, どんな僅かな時間であってもハンダゴテの電源プラグを抜いて, 必ず電源を切るようにして下さい.
- 電子工作の作業台は常に整理整頓を心がけ, ハンダゴテの周囲には燃えやすいものを置かないようにして下さい.
- 5 ボルトおよび 12 ボルトの電圧でもショートすれば発火し火災に至る可能性があります. 工作中だけでなく使用中においても目を離さず, 事故が起きないよう十分にご注意下さい.

はじめに

本書は鉄道模型を楽しむためのマイコン電子小工作集です. 基本となる LED 点灯装置や PWM 式パワーパックの作成から始めて, 最後には自動運転システムの作成までを行います. アナログ回路では実現が困難なものでも, マイコンを用いてファームウエアを工夫すれば実に多彩な動作が可能となります.

特にマイクロコントローラ (microcontroller; 以下「マイコン」と略記) のファームウエア (ソフトウエア) プログラミングを学習します. 使用するマイコンは Microchip Technology 社 PIC 16F シリーズの中でも, 本書執筆時点で最新でメモリ量や内蔵周辺モジュール (ペリフェラル) の充実した 16F18877, 16F18857, 16F18346, 16F18326, 16F18313 を使用します. ファームウエアは簡単なものから高度なものへと, 少しずつ学習を高めてゆけるよう順序と内容に工夫をしています.

本書で想定する読者

- 鉄道模型メーカ既製の商品では物足りない方.
- 欲しいものは自分で作り上げたい自作派の方.
- 鉄道模型の自動運転システムに興味のある方.
- 初級レベルの電子工作とコンピュータプログラミングの技術を持ち, 中級レベルを目指したい方.
- OS もなにもない素のマイコンを対象として, 組み込みリアルタムシステム開発を勉強したい方.
- PIC マイコンのサンプルプログラムを探している方.

たとえ予備知識に自信がなくても, 本書の内容を理解して製作できるようになりたいという熱意と努力の継続こそが一番の条件です.

本書の基本方針: 30 年以上は使い続けられるものを

本書で紹介する各種工作は 30 年間に渡って使い続けられることを念頭に, 十分に枯れていて, しかも今後も当分は使い続けられると思われる技術を用いて設計を行います. 製作の時にはマイコンへのファームウエアの書き込みを行うためにパーソナルコンピュータを使用しますが, ひとたび製作が完了した後にはパーソナルコンピュータはまったく使用しません.

近年ではパーソナルコンピュータやスマートホンで操作できる鉄道模型のコントローラがいくつも発表されており, 大変便利そうでしかも見た目のきらびやかさについ心が奪われてしまいがちです. しかしそのようなシステムは, すぐに使えなくなるのではないかと心配します. 10 年後も今のパーソナルコンピュータやスマートホンを使い続けることは一体可能なのでしょうか. 特にスマートホンの内蔵バッテリは数年で劣化して使えなくなり, そして交換バッテリーが販売終了になる時は必ずやってきます. このようにパーソナルコンピュータやスマートホンを数年で使い捨てをせざるを得ない残念な現実があります. 一方, 鉄道模型は 30 年あるいはそれ以上の永きに渡って楽しむ物であり, パーソナルコンピュータやスマートホンとは寿命の観点からあまり相容れないものと考えています.

当然のことながら, 製作した装置が壊れる時は必ず来ますし, マイコンや半導体などの部品が生産中止となる時が来るのも避けることはできません. しかし使用する部品はできるだけ汎用性の高いものを選ぶようにし, ほとんどの部分をデジタル回路として設計していますので, 代替部品を用いても回路定数の再計算等が生じるこ

とはありません. ごく一部にアナログ回路がありますが, 回路自体が単純なので回路定数の再計算は容易に行えます. そして本書で一番中心となるファームウエアでのアルゴリズムは, たとえマイコンが変わっても基本的な論理自体は不変ですから, 本書で紹介したアルゴリズムを新たな別のマイコンでも使い続けることができます.

本書は鉄道模型を永きに渡って楽しめることを意図した小工作集です. しかし鉄道模型も紹介する装置も決して永遠に不変のものではなく, 一切は絶えず変化し続けてゆきます. 今あるものを大切にしつつも固執せず, 新たなものを受け入れつつも熱狂せず, みなさまが何事にもとらわれない自由で穏やかな境地であられますことを願っています.

目次

第 1 章　オリエンテーション 30
1.1　本書の利用法 30
1.2　鉄道模型を楽しむ 4 つのモデルコース 31
1.3　ハイライト: 本書で製作する装置の代表例 33
1.4　本書で取り扱わないこと・取り扱うこと 38
1.5　小規模から大規模まで自由自在なシステム構築 40

第 I 部　鉄道模型とマイコン電子工作の基本 45

第 2 章　電気式鉄道模型工学概論 46
2.1　制御方式 46
2.2　給電方式 51
2.3　ポイントの電気的特性 52
2.4　ポイントマシン 53
2.5　フィーダとギャップ 56

第 3 章　電子部品の基礎知識 61
3.1　受動部品 61
3.2　コネクタ部品 69
3.3　半導体部品 70
3.4　PIC マイクロコントローラ 81

第 4 章　マイコン電子工作の基礎練習 86
4.1　回路図 86
4.2　工具の準備 88
4.3　回路の組み立て 90
4.4　MPLAB X IDE を用いたファームウエアの開発と書き込み 92
4.5　MPLAB X IPE を用いたインテル HEX 形式の配布ファームウエアの書き込み 100

第 5 章　マイコンプログラミングの基礎学習 104
5.1　PORT: デジタル I/O ポートモジュール 105
5.2　ADC: AD 変換モジュール 106
5.3　TMR2/4/6: 8 ビットタイマモジュール 108
5.4　CCP: キャプチャ/比較/PWM モジュール 109
5.5　PPS: ペリフェラルピンセレクトモジュール 110

5.6 PIC マイコンにおける PWM 制御 112
5.7 ファームウエア 113

第 II 部 PWM 制御によるジオラマ用 LED 照明と基本パワーパック 123

第 6 章 ブレゼンハムアルゴリズムの基礎理論 124
6.1 線形増加アルゴリズム 126
6.2 指数増加アルゴリズム 132

第 7 章 LED の点灯 – 基礎編 136
7.1 LED 風点滅 137
7.2 電球風点灯 139
7.3 電球風点滅 142

第 8 章 LED の点灯 – 発展編 146
8.1 ネオン管風点灯 146
8.2 固定小数点実数 149
8.3 蝋燭風点灯 150
8.4 蛍光灯風点灯 153

第 9 章 パワーパック: 基本機能型 160
9.1 使用方法 161
9.2 回路図 162
9.3 CWG: 相補波形ジェネレータモジュール 163
9.4 モータドライバ IC TB6643KQ 165
9.5 ファームウエア 169

第 10 章 パワーパック: 自動加減速・低速ブースト機能型 180
10.1 使用方法 181
10.2 自動加減速機能の実現 184
10.3 割り込みの概念 184
10.4 低速ブースト機能の実現 186
10.5 ファームウエア 187

第 11 章 パワーパック: ワンハンドルマスコン型 201
11.1 使用方法 202
11.2 回路図 202
11.3 ファームウエア 203

第 III 部 ポイントの制御 209

第 12 章 マイコン制御ポイント切替器: ソレノイド方式ポイントマシン用 210

12.1 使用方法 . . . 212
12.2 回路図 . . . 214
12.3 3 線ソレノイド方式ポイントマシン用アダプタ . . . 217
12.4 電源投入時の動作 . . . 218
12.5 WDT: ウォッチドッグタイマモジュール . . . 219
12.6 ファームウエア . . . 221

第 13 章 マイコン制御ポイント切替器: DC モータ方式用 230
13.1 使用方法 . . . 230
13.2 ファームウエア . . . 231

第 14 章 リレー制御型ポイント切替器 236
14.1 使用方法 . . . 236
14.2 2 線ソレノイド方式ポイント切替器 . . . 237
14.3 3 線ソレノイド方式ポイント切替器 . . . 239
14.4 DC モータ方式用ポイント切替器 . . . 240

第 15 章 サーボモータ制御器 242
15.1 使用方法 . . . 243
15.2 回路図 . . . 245
15.3 サーボモータの概要 . . . 246
15.4 TMR1/3/5: 16 ビットタイマモジュール . . . 246
15.5 ファームウエア . . . 248

第 IV 部 鉄道模型を楽しむアクセサリ 257

第 16 章 車両センサ 258
16.1 車両センサインターフェイス仕様 . . . 259
16.2 変調赤外線反射検出型車両センサ . . . 260
16.3 変調赤外線遮断型車両センサ . . . 262
16.4 磁気リードスイッチ型車両センサ . . . 263
16.5 ホール IC 型車両センサ . . . 264
16.6 センサテスタ . . . 265

第 17 章 走行速度測定器 267
17.1 使用方法 . . . 268
17.2 回路図 . . . 270
17.3 7 セグメント LED による文字の表示 . . . 271
17.4 7 セグメント LED のダイナミック点灯 . . . 271
17.5 ファームウエア . . . 273

第 18 章 色灯式信号機 291

18.1 使用方法 . . . 293
18.2 回路図 . . . 295
18.3 動作仕様のデータ記述化 . . . 297
18.4 ファームウエア . . . 299

第 19 章 線路電圧極性検出器 320
19.1 使用方法 . . . 320
19.2 回路図 . . . 322
19.3 FVR: 固定参照電圧モジュール . . . 324
19.4 CM: コンパレータモジュール . . . 325
19.5 線路電圧極性の検出 . . . 326
19.6 ファームウエア . . . 327

第 20 章 DCC デコーダとの接続 334
20.1 使用方法 . . . 334
20.2 回路図 . . . 335

第 21 章 装置の相互結合 337
21.1 アクセサリ制御入出力 . . . 338
21.2 相互接続の回路 . . . 340

第 V 部 テレビリモコン式のパワーパック 345

第 22 章 パワーパック: テレビリモコン式 346
22.1 使用方法 . . . 347
22.2 回路図 . . . 349
22.3 赤外線テレビリモコン規格 . . . 350
22.4 赤外線リモコン受信モジュール . . . 351
22.5 ファームウエア . . . 353

第 23 章 パワーパック: ポイント制御機能・テレビリモコン式 379
23.1 使用方法 . . . 380
23.2 回路図 . . . 380
23.3 同期 2 線式シリアル通信 I^2C . . . 382
23.4 MSSP: 同期シリアル通信モジュール . . . 384
23.5 I^2C I/O エクスパンダ IC MCP23017 . . . 386
23.6 ファームウエア . . . 389

第 VI 部 EEPROM を用いた高度なカスタマイズ技法 402

第 24 章 インテル HEX 形式への変換 403
24.1 インテル HEX 形式 . . . 404

24.2 インテル HEX 形式変換ソフトウエア bin2hex 406
24.3 ファイル 16 進ダンプソフトウエア bindump 409
24.4 各種オペレーティングシステムでのビルト方法 411

第 25 章 RS-232C 接続 EEPROM 書き込み器 413
25.1 使用方法 414
25.2 回路図 419
25.3 非同期シリアル通信 RS-232C 420
25.4 EUSART: 非同期シリアル通信モジュール 422
25.5 巡回バッファ構造による入出力バッファリング 424
25.6 I^2C EEPROM 24FC64 と 24FC1025 426
25.7 ファームウエア 427

第 26 章 サウンド EEPROM データの生成 464
26.1 使用方法 465
26.2 音源ファイルの編集 468
26.3 サウンド EEPROM データ形式 474
26.4 WAV ファイル形式 475
26.5 サウンド EEPROM データファイル生成プログラム wav2img 478
26.6 各種オペレーティングシステムでのビルト方法 483

第 27 章 PCM サウンド再生器 486
27.1 使用方法 488
27.2 回路図 489
27.3 PWM 波出力による PCM サウンド再生 490
27.4 TMR0: 8 ビットタイマモジュール 492
27.5 チャタリングの除去 493
27.6 ファームウエア 493

第 28 章 ポイント一括切替器 509
28.1 使用方法 510
28.2 回路図 514
28.3 ロータリエンコーダ 515
28.4 経路設定データの書式 516
28.5 ファームウエア 518
28.6 経路設定データ書式チェッカ chkroute 536

第 VII 部 鉄道模型の自動運転システムの製作 545

第 29 章 自動往復運転システム 546
29.1 使用方法 551
29.2 回路図 557

29.3 減速度パラメータの自動学習 . . . 563
29.4 有限状態機械による自動運転の実現 . . . 567
29.5 ファームウエア . . . 572

第 30 章 スクリプトによる自動運転システム 620
30.1 使用方法 . . . 621
30.2 回路図 . . . 623
30.3 自動運転スクリプト言語の構文 . . . 623
30.4 自動運転スクリプトの例 . . . 628
30.5 ファームウエア . . . 632

第 31 章 自動運転スクリプトの作成技法 649
31.1 単線退避駅トラックプラン . . . 649
31.2 エンドレス交換駅トラックプラン . . . 651
31.3 二重リバーストラックプラン . . . 653
31.4 エンドレススイッチバック駅トラックプラン . . . 654

第 32 章 自動運転スクリプトのエミュレーション技法 658
32.1 使用方法 . . . 659
32.2 自動運転エミュレータプログラム emuscript . . . 663
32.3 各種オペレーティングシステムでのコンパイル方法 . . . 671

参考文献 674

索引 676

図目次

1.1 スイッチバックのあるレイアウトでの自動運転システム 30
1.2 (イメージ写真) 本書の装置は様々なスケール・ゲージで利用可能 31
1.3 4 つのモデルコース 32
1.4 本書で紹介する自動運転スクリプトのレイアウト例 33
1.5 各章の相関図 34
1.6 低速ブースト機能の効果 36
1.7 自動運転用スクリプトの例 37
1.8 エンドレスレイアウトでの自動運転システムの例 37
1.9 自動運転における停止のための最適な減速度の自動学習 38
1.10 PIC マイコンの例: 16F18877, 16F18326, 16F18313 40
1.11 拡張可能な自動運転システム 43
1.12 アクセサリ出力への様々な機器の接続 43

2.1 各種走行制御方式での電圧波形 47
2.2 PWM デューティ比と波形 48
2.3 PWM 波 (デューティー比を 0% から 100% に変化) 49
2.4 DCC 電圧波形の例 50
2.5 DCC 機器の例 50
2.6 模型車両への給電方式 51
2.7 ポイントに関する用語 52
2.8 選択式・非選択式ポイントの電気的特性 52
2.9 ポイントの例 54
2.10 外付けポイントマシンの例 54
2.11 ソレノイド式ポイントマシン内部の模式図 55
2.12 フィーダとギャップの例 (KATO N ゲージ ユニトラック) 56
2.13 通常区間用とリバース区間用の逆転スイッチ 56
2.14 選択式ポイントにおけるギャップの必要性 57
2.15 リバース区間におけるギャップの必要性 57
2.16 レイアウトプランで使用する記号 58
2.17 ギャップとフィーダの付け方 58

3.1 抵抗器 62
3.2 カラーコードによる抵抗値の表示 62
3.3 可変抵抗器のカーブ三種 63
3.4 可変抵抗器 10kΩ の端子間の抵抗値 63

3.5 セラミックコンデンサ . . . 63
3.6 電解コンデンサ . . . 63
3.7 セラミックコンデンサの容量の表示 . . . 64
3.8 電解コンデンサの容量と極性の表示 . . . 64
3.9 スイッチ . . . 65
3.10 DIP スイッチ . . . 66
3.11 ロータリースイッチ . . . 67
3.12 リードスイッチ . . . 67
3.13 リレー . . . 67
3.14 ポリスイッチ . . . 68
3.15 ロータリエンコーダ . . . 69
3.16 DC ジャック . . . 69
3.17 ターミナルブロック . . . 69
3.18 基板対電線型コネクタ . . . 70
3.19 RS-232C 用 D サブコネクタ . . . 70
3.20 ダイオード . . . 71
3.21 発光ダイオード LED . . . 72
3.22 LED と電流制限抵抗 . . . 72
3.23 7 セグメント LED . . . 73
3.24 定電流ダイオード CRD . . . 74
3.25 電界効果トランジスタ FET 2N7000 . . . 74
3.26 トランジスタアレイ TBD62083APG . . . 75
3.27 定電圧レギュレータ 7805 . . . 75
3.28 モータドライバ TB6643KQ . . . 76
3.29 EEPROM 24FC64 . . . 77
3.30 I/O エクスパンダ MCP23017 . . . 78
3.31 光変調型フォト IC と赤外線 LED . . . 78
3.32 赤外線リモコン受信モジュール各種 . . . 79
3.33 ホール IC SK8552G . . . 79
3.34 フォトカプラ TLP785 . . . 80
3.35 オーディオアンプ IC NJM386BD . . . 80
3.36 RS-232C インターフェース IC ADM3202AN . . . 81
3.37 PIC マイコン各種 (16F18xxx シリーズ) . . . 82
3.38 PIC マイコンのピン配置 . . . 83

4.1 試作例: LED 点灯装置 . . . 86
4.2 【回路図】 LED 点灯 . . . 87
4.3 作業机の様子 . . . 89
4.4 太陽電機産業 PX-60RT-1.5C . . . 89
4.5 電源関係 (+5V と GND) の結線 . . . 90
4.6 操作に関係する部品レイアウトと部品名称 . . . 90

4.7 基板を安定させる足の仮付け . . . 90
4.8 回路図への作業経過の記入 . . . 90
4.9 LED 点灯装置の配線面 . . . 90
4.10 結束バンドによる電線の整理 . . . 92
4.11 ファームウエアの書き込みツール PICkit3, PICkit4, 書き込みアダプタ . . . 92
4.12 PICkit4 とアダプタを接続し, PIC マイコンをゼロプレッシャーソケットに搭載した様子 . . . 92
4.13 MPLAB X IDE での新規プロジェクト作成 . . . 95
4.14 MPLAB X IDE でのプロジェクトへのプログラム入力 (1/2) . . . 96
4.14 MPLAB X IDE でのプロジェクトへのプログラム入力 (2/2) . . . 97
4.15 MPLAB X IDE でのビルド (コンパイル) . . . 98
4.15 MPLAB X IDE での PIC マイコンへのファームウエアの書き込み . . . 99
4.16 MPLAB X IPE の起動/設定/書き込み (1/2) . . . 101
4.16 MPLAB X IPE の起動/設定/書き込み (2/2) . . . 102

5.1 LED 点灯装置 . . . 104
5.2 ADC の内部構造の概略 (16F18313) . . . 107
5.3 TMR2 の内部構造の概略 . . . 108
5.4 CCP1 の PWM モードにおける PWM 波形出力 . . . 109
5.5 PWM 波の生成 . . . 110
5.6 PPS の内部構造の概略 . . . 111
5.7 PWM 制御による LED 点灯 . . . 112
5.8 状態遷移図 . . . 121

6.1 ブレゼンハムアルゴリズムによる一次関数の計算 . . . 124
6.2 本書で紹介するブレゼンハムアルゴリズム 2 種 . . . 126
6.3 線分, ピクセル, モータ回転速度 . . . 126
6.4 連続直線と離散直線 . . . 127
6.5 直線による平面の二分 . . . 128
6.6 ピクセル (x, y) に隣接する 3 つの候補ピクセル . . . 129
6.7 CR 回路 . . . 133

7.1 LED 風点滅の輝度変化 . . . 136
7.2 LED 風点滅の状態遷移図 . . . 138
7.3 状態遷移の記述方法 . . . 139
7.4 電球風点灯の輝度変化 . . . 140
7.5 電球風点灯の状態遷移図 . . . 142
7.6 電球風点滅の輝度変化 . . . 142
7.7 電球風点滅の状態遷移図 . . . 142

8.1 ネオン管風点灯の輝度変化 . . . 146
8.2 固定小数実数 Q0.16 形式 . . . 149
8.3 蝋燭風点灯の輝度変化の例 . . . 151

8.4 蛍光灯風点灯の輝度変化期間 . . . 154
8.5 安定度による違いによる蛍光灯風点灯の輝度変化 . . . 155
8.6 蝋燭風点灯の状態遷移図 . . . 156
8.7 蝋燭の灯 (碇型 2 匁和蝋燭) . . . 158

9.1 試作例: パワーパック . . . 160
9.2 接続方法 . . . 160
9.3 ファームウエア構造 (概要) . . . 161
9.4 操作表示部品の配置例と名称 . . . 161
9.5 【回路図】 パワーパック (基本機能型) 本体 . . . 164
9.6 CWG1 の内部構造の概略 (出力ステアリングモード) . . . 164
9.7 H ブリッジモータドライバの模式図と入力ごとの動作 . . . 166
9.8 PWM 制御におけるアクティブ・ロウとアクティブ・ハイ . . . 166
9.9 ヘッドライトとテールライトが点灯する原理 . . . 168
9.10 走行中にヘッドライトとテールライトがちらつく原理 (デューティ・オフ期間) . . . 168
9.11 ファームウエア構造 (詳細) . . . 170

10.1 低速ブースト機能が超低速走行を可能とする原理 . . . 180
10.2 自動加減速機能 . . . 180
10.3 ファームウエア構造 (概要) . . . 182
10.4 操作表示部品の配置例と名称 . . . 183
10.5 接続方法 . . . 183
10.6 割り込み処理 . . . 185
10.7 低速ブースト機能による PWM デューティ比の時間変化 . . . 186
10.8 低速ブースト機能による高 PWM デューティ比期間を挿入した PWM 波形 . . . 186
10.9 PWM 波形における低速ブーストパルス幅と低速ブースト周波数 . . . 186
10.10 低速ブースト機能の実現方法 (概要) . . . 188
10.11 通常期間と低速ブースト期間での状態遷移 . . . 196
10.12 通常期間と低速ブースト期間の状態遷移の実現方法と PWM 波出力 . . . 197

11.1 マスコンでの力行・惰行・制動による速度の制御 . . . 201
11.2 接続方法 . . . 201
11.3 ワンハンドルマスコンの コントローラ部の製作例 . . . 201
11.4 ファームウエア構造 (概要) . . . 202
11.5 【回路図】 ワンハンドルマスコン . . . 203

12.1 試作例: ポイント切替器 . . . 210
12.2 操作表示部品の配置例と名称 . . . 210
12.3 ファームウエア構造 (概要) . . . 211
12.4 ポイントマシンとの接続 . . . 212
12.5 出力波形の模式図 . . . 212
12.6 トグルスイッチによるポイント切り替え . . . 214

12.7 テスタ棒によるタッチ式のポイント切り替え . . . 214
12.8 押しボタンスイッチによるポイント切り替え . . . 214
12.9 【回路図】 ポイント切替器 (ソレノイド方式/モータ方式共通) . . . 215
12.10 動作確認用テスタ (2 線ソレノイド方式用/モータ方式用共通) . . . 217
12.11 3 線ソレノイド方式用ポイントマシン用アダプタ . . . 217
12.12 出力波形の模式図 (3 線式アダプタの出力) . . . 218
12.13 3 線ソレノイド方式用動作確認用テスタ . . . 218
12.14 ウオッチドッグタイマモジュール WDT の内部構造の概略 . . . 219
12.15 WDT を用いたファームウエアの構造 . . . 220
12.16 ファームウエア構造 (詳細) . . . 222
12.17 有限状態機械の状態遷移図 . . . 222
12.18 ポイント制御入力に対する状態遷移図 . . . 224

13.1 ファームウエア構造 (概要) . . . 231
13.2 出力波形の模式図 . . . 231
13.3 ファームウエア構造 (詳細) . . . 232

14.1 試作例: ポイント切替器 . . . 236
14.2 接続方法 . . . 237
14.3 【回路図】 2 線ソレノイド方式用リレー制御型ポイント切替器 . . . 238
14.4 【回路図】 3 線ソレノイド方式用リレー制御型ポイント切替器 . . . 239
14.5 【回路図】 DC モータ方式用リレー制御型ポイント切替器 . . . 240

15.1 サーボモータ制御器 . . . 242
15.2 入力信号とサーボモータ出力の模式図 . . . 243
15.3 操作表示部品の配置例と名称 . . . 243
15.4 接続例 . . . 244
15.5 【回路図】 サーボモータコントローラ . . . 245
15.6 サーボモータの制御信号 . . . 247
15.7 TMR1 の内部構造の概略 . . . 247
15.8 ファームウエア構造 . . . 249

16.1 本書で紹介する車両センサ 4 種の検出方式 . . . 258
16.2 車両センサインターフェース仕様の概念図 . . . 259
16.3 【回路図】 車両センサ (変調赤外線反射検出型) . . . 260
16.4 試作例: 変調赤外線反射検出型車両センサ (S7136/S6809) . . . 261
16.5 【回路図】 車両センサ (変調赤外線遮断検出型) . . . 262
16.6 試作例: 赤外線遮断検出車両センサ (S6809) . . . 263
16.7 車両センサ (磁気リードスイッチ型) . . . 263
16.8 磁気リードスイッチ型車両センサ . . . 264
16.9 【回路図】 車両センサ (磁気ホール IC 型) . . . 264
16.10 試作例: 磁気ホール IC 型車両センサ . . . 265

16.11 【回路図】 車両センサ用テスタ 265

17.1 走行速度測定器 267
17.2 走行速度の測定・算出方法の原理 267
17.3 ファームウエア構造 (概要) 268
17.4 操作表示部品の配置例と名称 268
17.5 接続方法 268
17.6 設定モード時の表示 269
17.7 【回路図】 走行速度測定器 272
17.8 セグメントの名称 273
17.9 7 セグメント LED による文字の表示 273
17.10 7 セグメント LED のダイナミック点灯 273

18.1 試作例: 色灯式信号機 291
18.2 信号の動作の例 (三灯式信号機の例) 291
18.3 現示パターン (1/2) 292
18.3 現示パターン (2/2) 293
18.4 操作表示部品の配置例と名称 294
18.5 接続方法 294
18.6 【回路図】 色灯式信号器制御部 295
18.7 【回路図】 色灯式信号器表示部 297
18.8 現示パターンのデータ表現と現示の進行の仕組み 300
18.9 ファームウエア構造 (概要) 301
18.10 電球風点灯を実現する LED 輝度制御 (3 灯式信号機での例) 302
18.11 CWG と GPIO を組み合わせた LED の輝度の制御の概要 302

19.1 線路電圧極性検出器 321
19.2 操作表示部品の配置例と名称 321
19.3 接続方法 321
19.4 ファームウエア構造 (概要) 321
19.5 線路電圧と検出結果 322
19.6 【回路図】 線路電圧極性検出器 323
19.7 固定参照電圧モジュール FVR の内部構造の概略 324
19.8 コンパレータモジュール CMx の内部構造の概略 325
19.9 コンパレータモジュール CM と固定参照電圧モジュール FVR による電圧極性の検出 327

20.1 試作例: DCC デコーダ接続インターフェース 334
20.2 DCC システム 335
20.3 DCC デコーダとの接続 335
20.4 【回路図】 DCC デコーダ接続インターフェース装置 336

21.1 装置間の相互接続 337

21.2 アクセサリ制御出力の論理和と分岐 . . . 339
21.3 試作例: 相互接続用装置 . . . 340
21.4 【回路図】 論理和装置 (3 入力 1 出力) . . . 341
21.5 【回路図】 出力分岐装置 (1 入力 3 出力) . . . 341
21.6 【回路図】 論理否定装置 . . . 342
21.7 【回路図】 アクセサリ制御出力モニタ . . . 343

22.1 試作例: テレビリモコン式パワーパック . . . 346
22.2 汎用テレビリモコンの例 . . . 346
22.3 接続方法 . . . 346
22.4 ファームウエア構造 (概要) . . . 348
22.5 部品の配置例と名称 . . . 348
22.6 【回路図】 パワーパック (テレビリモコン式) 拡張ボード . . . 351
22.7 パワーパック拡張ケーブル . . . 351
22.8 赤外線テレビリモコン NEC フォーマット . . . 352
22.9 赤外線リモコン受信モジュール OSRB38C9AA . . . 352
22.10 赤外線リモコン受信モジュール OSRB38C9AA の出力 . . . 352
22.11 赤外線信号の変化からコマンドの実行までの関数呼出しの関係 . . . 358
22.12 テレビリモコンのボタン押下からパワーパックの動作までのデータの流れ . . . 363
22.13 赤外線フレームデコーダの構造 (概要) . . . 364
22.14 赤外線フレームデコーダの有限状態遷移図 . . . 364

23.1 試作例: ポイント制御機能付テレビリモコン式パワーパック . . . 379
23.2 接続方法 . . . 379
23.3 【回路図】 ポイントインターフェースボード . . . 381
23.4 I^2C バスケーブル . . . 382
23.5 I/O エクスパンダ IC MCP23017 の内部構造の概略 . . . 386
23.6 MCP23017 の制御バイト . . . 388
23.7 MCP23017 のアドレスシーケンス . . . 388
23.8 MCP23017 の I^2C 通信プロトコル (一部) . . . 389
23.9 ポイント切替動作キューによるポイントの順次切り替えの仕組み . . . 391

24.1 作業手順: インテル HEX ファイルによる EEPROM 使用機器の利用 . . . 403
24.2 bin2hex 使用中の様子 (Ubuntu 上) . . . 404
24.3 インテル HEX 形式のファイル全体の例 . . . 404
24.4 インテル HEX 形式のレコードの例 . . . 405

25.1 EEPROM 書き込み器 . . . 413
25.2 接続方法 . . . 413
25.3 端末ソフトウエア gtkterm を通して使用中の様子 (Ubuntu 上) . . . 413
25.4 操作表示部品の配置例と名称 . . . 414
25.5 RS-232C 接続で使用する品 . . . 415

25.6 本装置の一般的な接続方法 . . . 415
25.7 【回路図】 EEPROM メモリ書き込み器 . . . 421
25.8 D サブ 9 ピンコネクタの配列 (正面より) . . . 422
25.9 【回路図】 RS-232C ケーブル (D サブ 9 ピンコネクタ) . . . 423
25.10 巡回バッファによるバッファリング . . . 425
25.11 I^2C EEPROM 24FC64 の制御バイトとアドレスシーケンス . . . 427
25.12 I^2C EEPROM 24FC1025 の制御バイトとアドレスシーケンス . . . 427
25.13 I^2C EEPROM 24FC64 と 24FC1025 の I^2C 通信プロトコル . . . 428
25.14 ファームウエアにおける主要ソースファイル間の関係 . . . 430
25.15 ファームウエア構造 (概要) . . . 430

26.1 作業手順: PCM サウンド再生器で使用するサウンド EEPROM データの生成 . . . 464
26.2 wav2img 使用中の様子 (Ubuntu 上) . . . 465
26.3 サウンド EEPROM データ作成の作業手順の詳細 . . . 466
26.4 サウンド再生の 2 つの種類: ワンショット型とリピート型 . . . 466
26.5 サウンド EEPROM データのファイル構造の模式図 . . . 467
26.6 音源ファイルの切り出しとその繰り返し . . . 469
26.7 Audacity 起動時の画面 . . . 469
26.8 切り分けの箇所 (時間軸の拡大による表示) . . . 469
26.9 前後の不要部分を切り捨てて必要部分だけを残す . . . 470
26.10 編集箇所 (切り出す場所の目星をつける) . . . 470
26.11 開始部と繰返部の切り出し (上: 元, 中: 開始部, 下: 繰返部) . . . 471
26.12 終了部の切り出し . . . 472
26.13 WAV ファイルの内部構造 (概略) . . . 475
26.14 WAV ファイルの例 (16 進ダンプ) . . . 477

27.1 試作例: EEPROM メモリ型 PCM サウンド再生器 . . . 486
27.2 作業手順: PCM サウンド再生器の利用 . . . 487
27.3 ファームウエア構造 (概要) . . . 487
27.4 EEPROM メモリ型 PCM サウンド器の接続表示部品の配置例と名称 . . . 488
27.5 【回路図】 EEPROM メモリ型 PCM サウンド器 . . . 489
27.6 データと信号の流れ . . . 491
27.7 PWM 波出力を利用した PCM サウンド再生の原理 . . . 491
27.8 TMR0 の内部構造の概略 . . . 492
27.9 チャタリングの除去 . . . 493
27.10 サウンドデータの読み出しから PWM 波の出力までの流れ . . . 494
27.11 ファームウエア構造 (詳細) . . . 495

28.1 試作例: ポイント一括切替器 . . . 509
28.2 接続方法 . . . 509
28.3 ファームウエア構造 (概要) . . . 510

28.4 作業手順: 経路一括切替器の利用 . . . 510
28.5 操作表示部品の配置例と名称 . . . 511
28.6 7 セグメント LED によるポイント切替内容の表示例 . . . 512
28.7 【回路図】 プログラマブル式ポイント一括切替器/メインボード . . . 513
28.8 インクリメンタル式ロータリエンコーダ . . . 515
28.9 ある架空の駅の線路配線図 (数字: ポイント番号, ローマ字: 場所) . . . 516
28.10 EEPROM に書き込む経路設定データの例 . . . 516
28.11 経路設定データの書式 . . . 517
28.12 ファームウエアにおける主要ソースファイル間の関係 . . . 519
28.13 動作の流れの概要 . . . 529
28.14 作業手順: 経路設定データ書式チェッカ chkroute を使用した経路一括切替器の利用 . . . 537
28.15 誤りなしと誤りありの経路データの例 . . . 537
28.16 経路設定データ書式チェッカ chkroute における主要ソースファイル間の関係 . . . 539

29.1 事前組み込み済みレイアウト 3 種 . . . 546
29.2 試作例: 自動往復運転システム . . . 546
29.3 事前組み込み済みのトラックプラン 3 種 . . . 547
29.4 エンドレストラックプランの実施例 (KATO ユニトラック) . . . 548
29.5 リバーストラックプランの実施例 (KATO ユニトラック) . . . 548
29.6 ファームウエア構造 (概要) . . . 549
29.7 接続方法 . . . 550
29.8 自動往復運転システムの操作表示部品の配置例と名称 . . . 551
29.9 トラックプランの記号: フィーダ . . . 553
29.10 VSLOW と最低維持速度での走行 . . . 556
29.11 徐行速度の設定の重要性 . . . 556
29.12 【回路図】 自動往復運転システム / メインボード (1/2) . . . 558
29.13 【回路図】 自動往復運転システム / メインボード (2/2) . . . 559
29.14 【回路図】 自動往復運転システム / センサ・ポイントインターフェースボード . . . 560
29.15 制動の開始から停車までの走行速度の変化 . . . 563
29.16 制動センサと停車センサの間の距離 L と走行距離の関係 . . . 564
29.17 走行距離の近似算出法 . . . 565
29.18 単線トラックプランでの自動運転を実現する基本操作の列 . . . 568
29.19 基本操作に対する状態遷移図の断片と動作 . . . 569
29.20 自動運転における基本状態 . . . 571
29.21 単線トラックプランに対する自動運転有限状態機械の状態遷移図 . . . 572
29.22 エンドレストラックプランに対する自動運転有限状態機械の状態遷移図 . . . 572
29.23 リバーストラックプランに対する自動運転有限状態機械の状態遷移図 . . . 573
29.24 ファームウエアにおける主要ソースファイル間の関係 . . . 575
29.25 自動運転パターン実行の抽象化状態遷移図 . . . 587
29.26 関数 `auto_run_start()` の構造 (概要) . . . 601
29.27 減速度パラメータの改善プロセス . . . 605

29.28 センサ停止走行 RUN-UNTIL(v,i,Sb,Ss) 608
29.29 センサ通過走行 RUN-TO(v,Sp) 609

30.1 作業手順: スクリプトによる自動運転システムの利用 620
30.2 試作例: アクセサリインターフェースボード 621
30.3 アクセサリインターフェースボード部品の配置例と名称 622
30.4 【回路図】 自動往復運転システム / アクセサリインターフェースボード 624
30.5 自動運転スクリプト言語の構文 (1/3) 625
30.5 自動運転スクリプト言語の構文 (2/3) 626
30.5 自動運転スクリプト言語の構文 (3/3) 627
30.6 エンドレス退避駅トラックプラン 630
30.7 ファームウエアにおける主要ソースファイル間の関係 632

31.1 単線退避駅トラックプラン 649
31.2 エンドレス交換駅トラックプラン 651
31.3 エンドレス交換駅トラックプランの実施例 (KATO ユニトラック) 651
31.4 二重リバーストラックプラン 653
31.5 二重リバーストラックプランの実施例 (KATO ユニトラック) 653
31.6 エンドレススイッチバック駅トラックプラン 655
31.7 課題用レイアウト 5 題 657

32.1 emuscript 使用中の様子 (Ubuntu 上) 658
32.2 作業手順: エミュレータを使用した自動運転スクリプト開発 659
32.3 自動運転スクリプト ato-shuttle.txt のエミュレーション実行時の出力 660
32.4 自動運転エミュレータ emuscript における主要ソースファイル間の関係 662
32.5 ソースコード関係図 664
32.6 エミュレーション実行機構の概要 664

表目次

1.1 習得するプログラミング技法 41
1.2 使用する PIC と内蔵ペリフェラルモジュール 42

3.1 カラーコード 63
3.2 LED 電流制限抵抗の計算例 73
3.3 TB6643KQ 入出力ファンクション表 76
3.4 本書で使用する主な EEPROM 77
3.5 本書で使用する PIC マイコン 81
3.6 PIC マイコン内蔵周辺モジュール装備の一覧 (抜粋) 85

4.1 【部品表】 LED 点灯装置 87

5.1 【ソースファイル一覧】 PWM 制御 LED 点灯 113

7.1 【ソースファイル一覧】 LED 風点滅 137
7.2 【ソースファイル一覧】 電球風点灯 140
7.3 【ソースファイル一覧】 電球風点滅 143

8.1 【ソースファイル一覧】 ネオン管風点灯 147
8.2 【ソースファイル一覧】 蝋燭風点灯 151
8.3 【ソースファイル一覧】 蛍光灯風点灯 154

9.1 【部品表】 パワーパック (基本機能型) 162
9.2 【ソースファイル一覧】 パワーパック: 基本機能型 169

10.1 【ソースファイル一覧】 パワーパック: 自動加減速・低速ブースト機能型 187

11.1 【部品表】 ワンハンドルマスコン 203
11.2 【ソースファイル一覧】 パワーパック: ワンハンドルマスコン型 203

12.1 代表的なポイントマシンへの接続 213
12.2 ポイント切替器の仕様 213
12.3 【部品表】 ポイント切替器 215
12.4 開通状態出力の仕様 216
12.5 【部品表】 動作確認用テスタ (2 線ソレノイド方式用/モータ方式用共通) 217
12.6 【部品表】 3 線ソレノイド方式ポイントマシン用アダプタ 218
12.7 【部品表】 3 線ソレノイド方式用動作確認用テスタ 218
12.8 本書のポイント切替器の電源投入時の動作 219

12.9 【ソースファイル一覧】 ポイント切替器: ソレノイド方式用 221

13.1 【ソースファイル一覧】 ポイント切替器: DC モータ方式用 231

14.1 ポイント切替器の仕様 237
14.2 ポイント切替の仕様 238
14.3 【部品表】 2 線ソレノイド方式用リレー制御型ポイント切替器 238
14.4 【部品表】 3 線ソレノイド方式用リレー制御型ポイント切替器 239
14.5 【部品表】 DC モータ方式用リレー制御型ポイント切替器 240

15.1 【部品表】 サーボモータコントローラ 245
15.2 【ソースファイル一覧】 サーボモータ制御器 248

16.1 【部品表】 車両センサ (変調赤外線反射検出型) 261
16.2 【部品表】 車両センサ (変調赤外線遮断検出型) 262
16.3 【部品表】 車両センサ (磁気リードスイッチ型) 263
16.4 【部品表】 車両センサ (磁気ホール IC 型) 264
16.5 【部品表】 車両センサ用テスタ 265

17.1 DIP スイッチによる模型縮尺の設定 270
17.2 【部品表】 走行速度測定器 270
17.3 【ソースファイル一覧】 走行速度測定器 274

18.1 DIP スイッチによる現示パターンの設定 294
18.2 【部品表】 色灯式信号機制御部 295
18.3 【部品表】 色灯式信号機表示部 296
18.4 【ソースファイル一覧】 色灯式信号機 299

19.1 DIP スイッチによるの設定 322
19.2 【部品表】 線路電圧極性検出器 323
19.3 【ソースファイル一覧】 線路電圧極性検出器 327

20.1 DCC デコーダのリード線 335
20.2 【部品表】 DCC デコーダ接続インターフェース装置 336

21.1 【部品表】 論理和装置 (3 入力 1 出力) 341
21.2 【部品表】 出力分岐装置 (1 入力 3 出力) 341
21.3 【部品表】 論理否定装置 342
21.4 【部品表】 アクセサリ制御出力モニタ 342

22.1 テレビリモコンのボタン機能一覧 347
22.2 【部品表】 パワーパック (テレビリモコン式) 拡張ボード 349
22.3 【ソースファイル一覧】 パワーパック: テレビリモコン式 353

23.1 【部品表】 ポイントインターフェースボード 382
23.2 MCP23017 のレジスタ表の抜粋 (BANK=0) 387
23.3 【ソースファイル一覧】 パワーパック: ポイント制御機能付テレビリモコン式 389

25.1 DIP スイッチによる通信速度の設定 416
25.2 EEPROM 書き込み器コマンド一覧 416
25.3 サポート済み EEPROM 416
25.4 【部品表】 EEPROM メモリ書き込み器 419
25.5 RS-232C での一般的なピン番号と信号名 422
25.6 ボーレート決定の計算式 424
25.7 【ソースファイル一覧】 EEPROM 書き込み器 428

26.1 PCM サウンドデータ仕様 465
26.2 ワンショット型サウンド EEPROM データ形式 474
26.3 リピート型サウンド EEPROM データ形式 474
26.4 RIFF チャンク 475
26.5 fmt チャンク 476
26.6 data チャンク 476
26.7 【ソースファイル一覧】 サウンド EEPROM データ生成 wav2img 478

27.1 【部品表】 EEPROM メモリ型 PCM サウンド器 490
27.2 【ソースファイル一覧】 EEPROM メモリ型 PCM サウンド器 494

28.1 ポイントインターフェースボードの DIP スイッチ設定 512
28.2 エラーコード表 512
28.3 【部品表】 プログラマブル式ポイント一括切替器/メインボード 514
28.4 図 28.9 に対する経路表の例 516
28.5 【ソースファイル一覧】 EEPROM プログラム型ポイント一括切替器 518
28.6 【ソースファイル一覧】 経路設定データ書式チェッカ chkroute 539

29.1 メインボード DIP スイッチ OPRPAT 設定 552
29.2 インターフェースボード DIP スイッチ設定 552
29.3 パラメータの初期設定の参考値 554
29.4 【部品表】 自動往復運転システム / メインボード 561
29.5 【部品表】 自動往復運転システム / センサ・ポイントインターフェースボード 562
29.6 主な記号 565
29.7 【ソースファイル一覧】 自動往復運転システム 573
29.8 致命エラー時の LED の点滅パターン (本章の範囲のみ) 580

30.1 致命エラー時の LED の点滅パターン 622
30.2 【部品表】 自動往復運転システム / アクセサリインターフェースボード 623
30.3 【ソースファイル一覧】 EEPROM プログラム型自動運転システム 632

32.1 エラーメッセージ表 661
32.2 【ソースファイル一覧】 自動運転スクリプトエミュレータ emuscript 663

ソースファイル一覧

4.1 LED-Test.c . . . 93
5.1 main.c . . . 114
5.2 io.h . . . 114
5.3 mcr.h . . . 114
5.4 mcr.c . . . 116
5.5 adc.h . . . 117
5.6 adc.c . . . 118
5.7 LED-Light-LED.c . . . 119
6.1 incr-lin.h . . . 129
6.2 incr-lin.c . . . 130
6.3 incr-exp.h . . . 134
6.4 incr-exp.c . . . 134
7.1 LED-Light-Winker-LED.c . . . 137
7.2 LED-Light-Bulb.c . . . 140
7.3 LED-Light-Winker-Bulb.c . . . 143
8.1 LED-Light-Neon.c . . . 147
8.2 random.h . . . 148
8.3 random.c . . . 149
8.4 LED-Light-Candle.c . . . 151
8.5 LED-Light-FluoLamp.c . . . 155
9.1 io.h . . . 170
9.2 pp-const.h . . . 172
9.3 PowerPack-Basic.c . . . 172
9.4 mcr.h . . . 173
9.5 mcr.c . . . 174
9.6 mcr-motor.h . . . 175
9.7 mcr-motor.c . . . 177
10.1 pp.h . . . 189
10.2 PowerPack-SlowBoost.c . . . 189
10.3 pp-misc.c . . . 191
10.4 pp-control-vr.c . . . 193
10.5 pp-control.c . . . 194
10.6 pp-sb.h . . . 196
10.7 pp-sb.c . . . 198
11.1 pp-control-mc.c . . . 204

11.2 mcr.h . . . 206
11.3 mcr.c . . . 206
12.1 io.h . . . 221
12.2 turnout.h . . . 222
12.3 instate.c . . . 223
12.4 TurnoutCtr-Solenoid.c . . . 226
12.5 mcr.h . . . 228
12.6 mcr.c . . . 229
13.1 TurnoutCtr-Motor.c . . . 232
15.1 io.h . . . 248
15.2 servo.h . . . 249
15.3 ServoMotorCtr.c . . . 250
15.4 servo-pulse.c . . . 253
15.5 mcr.h . . . 254
15.6 mcr.c . . . 255
17.1 io.h . . . 274
17.2 defs.h . . . 275
17.3 Speedometer.c . . . 275
17.4 config.h . . . 279
17.5 config.c . . . 279
17.6 speed-timer.h . . . 281
17.7 speed-timer.c . . . 281
17.8 led7seg.h . . . 282
17.9 led7seg.c . . . 283
17.10 mcr.h . . . 288
17.11 mcr.c . . . 288
18.1 io.h . . . 303
18.2 led.h . . . 304
18.3 led.c . . . 304
18.4 aspect.h . . . 306
18.5 aspect.c . . . 307
18.6 Signal-TimerBased.c . . . 309
18.7 light-fsm.h . . . 312
18.8 light-fsm.c . . . 313
18.9 mcr.h . . . 316
18.10 mcr.c . . . 316
19.1 io.h . . . 328
19.2 DirDetect.c . . . 329
19.3 mcr.h . . . 331
19.4 mcr.c . . . 331

22.1 io.h . . . 354
22.2 PowerPack-IR.c . . . 355
22.3 pp-control-ir.c . . . 356
22.4 led-ir-mon.h . . . 359
22.5 led-ir-mon.c . . . 359
22.6 keysym.h . . . 360
22.7 cmd.h . . . 360
22.8 cmd.c . . . 361
22.9 ir-frame.h . . . 363
22.10 ir-frame.c . . . 365
22.11 rc.h . . . 367
22.12 rc.c . . . 367
22.13 rc-data.c . . . 368
22.14 auto-sel.c . . . 373
22.15 if.h . . . 374
22.16 if-toq.h . . . 374
22.17 if-toq-dummy.c . . . 374
22.18 mcr.h . . . 375
22.19 mcr.c . . . 375
23.1 if.h . . . 390
23.2 if-toq.c . . . 392
23.3 i2c.h . . . 394
23.4 i2c.c . . . 394
23.5 i2c-mcp23017.h . . . 396
23.6 i2c-mcp23017.c . . . 397
23.7 if-turnout.c . . . 399
24.1 bin2hex.c . . . 407
24.2 bindump.c . . . 409
24.3 Makefile . . . 412
25.1 io.h . . . 429
25.2 Writer-EEPROM.c . . . 431
25.3 devinfo.h . . . 434
25.4 devinfo.c . . . 434
25.5 cmd.h . . . 436
25.6 cmd.c . . . 436
25.7 ihex.h . . . 442
25.8 ihex.c . . . 443
25.9 cons.h . . . 446
25.10 cons.c . . . 446
25.11 com.h . . . 448

25.12 com.c . . . 450
25.13 f-24fc64.h . . . 453
25.14 f-24fc64.c . . . 453
25.15 f-24fc1025.h . . . 454
25.16 f-24fc1025.c . . . 455
25.17 i2c-24fc64.h . . . 456
25.18 i2c-24fc64.c . . . 457
25.19 i2c-24fc1025.h . . . 459
25.20 i2c-24fc1025.c . . . 459
25.21 mcr.h . . . 461
25.22 mcr.c . . . 461
26.1 defs.h . . . 478
26.2 snd-eeprom.h . . . 478
26.3 misc.c . . . 479
26.4 wav2img.c . . . 480
26.5 Makefile . . . 484
27.1 io.h . . . 494
27.2 SoundPlayer-EEPROM.c . . . 496
27.3 sw.h . . . 498
27.4 sw.c . . . 498
27.5 snd-eeprom.c . . . 499
27.6 buff.h . . . 501
27.7 buff.c . . . 501
27.8 producer.h . . . 503
27.9 producer.c . . . 503
27.10 consumer.h . . . 504
27.11 consumer.c . . . 504
27.12 mcr.h . . . 505
27.13 mcr.c . . . 505
28.1 SampleRoute.txt . . . 516
28.2 io.h . . . 518
28.3 defs.h . . . 519
28.4 RouteCtr.c . . . 521
28.5 fatal.h . . . 523
28.6 fatal.c . . . 523
28.7 rotenc.h . . . 524
28.8 rotenc.c . . . 525
28.9 mem.h . . . 525
28.10 mem.c . . . 526
28.11 route.h . . . 528

28.12 route.c . . . 529
28.13 action.h . . . 532
28.14 action.c . . . 533
28.15 mcr.h . . . 534
28.16 mcr.c . . . 535
28.17 test-route-ok.txt . . . 537
28.18 test-route-e3.txt . . . 537
28.19 test-route-e4.txt . . . 537
28.20 test-route-e5.txt . . . 537
28.21 test-route-e6.txt . . . 537
28.22 test-route-e8.txt . . . 537
28.23 chkroute.c . . . 540
28.24 action-stub.c . . . 541
28.25 fatal-stub.c . . . 541
28.26 mem-stub.c . . . 542
28.27 xc.h . . . 543
28.28 Makefile . . . 543
29.1 io.h . . . 574
29.2 AutoTrainOp.c . . . 577
29.3 fatal.h . . . 580
29.4 fatal.c . . . 581
29.5 manual-run.c . . . 582
29.6 fsm.h . . . 584
29.7 fsm.c . . . 584
29.8 fsm-shuttle.c . . . 585
29.9 fsm-endless.c . . . 587
29.10 fsm-reverse.c . . . 588
29.11 script-dummy.c . . . 590
29.12 if.h . . . 591
29.13 if.c . . . 592
29.14 if-sensor.c . . . 594
29.15 if-turnout.c . . . 596
29.16 if-accessary.c . . . 597
29.17 ar.h . . . 599
29.18 ar.c . . . 600
29.19 ar-decp.h . . . 605
29.20 ar-decp.c . . . 605
29.21 ar-motor.h . . . 606
29.22 ar-motor.c . . . 609
29.23 ar-est.h . . . 613

29.24 ar-est.c 614
29.25 ar-stay.h 615
29.26 ar-stay.c 616
29.27 mcr.h 617
29.28 mcr.c 617
30.1 ato-shuttle.txt 628
30.2 ato-endless.txt 629
30.3 ato-reverse.txt 629
30.4 ato-sta1-endless1.txt 631
30.5 script.h 636
30.6 script.c 638
30.7 script-read.c 640
30.8 script-exec.c 645
31.1 ato-sta1-shuttle.txt 649
31.2 ato-sta1-endless2.txt 651
31.3 ato-wreverse.txt 653
31.4 ato-switchback.txt 654
32.1 emuscript.c 665
32.2 usage.c 667
32.3 fatal-stub.c 668
32.4 exec-stub.c 669
32.5 Makefile 671

第 1 章

オリエンテーション

本章ではこの書籍全体を俯瞰し, 最終的な目標を説明します.

1.1 本書の利用法

主として対象とする鉄道模型は以下のもので, 特に日本の鉄道模型の多数派である N ゲージや 16 番で本書の装置が利用可能です.

- 直流 12 ボルト・アナログ制御方式の鉄道模型を対象とします. ただし, 本書の装置の一部は DCC デコーダを介して利用できます.
- 特定のスケールやゲージの鉄道模型に限定していません. さまざまなスケールやゲージに適用できますが, おおよそ Z, N, TT, HO, 16 番, S, O のスケール・ゲージの範囲を想定しています.

本書の利用方法には以下の 5 つのステップがあります. 一番最初のステップ 1 だけでも鉄道模型を十分に楽しめて, マイコン電子工作の技術も身につきますから, まずは気楽に取り組んみてはいかがでしょうか.

1. 装置を製作して, 鉄道模型を楽しむ.
2. 回路を理解する.
3. ファームウエアを理解する.
4. 回路やファームウエアを改造して新しい機能を追加する.
5. 独自の回路とファームウエアを新たに設計して, オリジナルの装置を開発する.

本書ではマイコンを使用した鉄道模型に関係する様々な装置を紹介します. マイコンを使わないアナログ電子回路による電子工作例がこれまで多数が紹介されてきていますが, 本書ではアナログ電子では実現困難な複雑な動作をマイコンで実現します. 電子回路は得意でもコンピュータ・プログラミングが苦手な方にとっては, これは逆に欠点と感じられるかも知れません. そこで単にマイコンを使用することだけを目的にするのではな

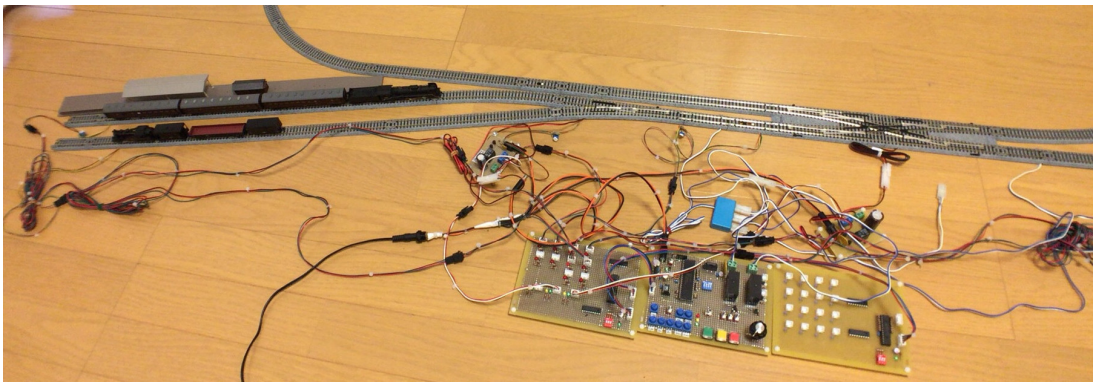


図 1.1: スイッチバックのあるレイアウトでの自動運転システム



図 1.2: (イメージ写真) 本書の装置は様々なスケール・ゲージで利用可能. 写真手前より, N ゲージ キハ 07 (ワールド工芸, 筆者組立), N ゲージ C11 とスハ 43 系客車 (KATO), OJ ゲージ C59 と 10 系寝台車 (ムサシノモデル, モデロセッテ), OJ ゲージ DE10 と 2 軸貨車 (クマタ, ムサシノモデル)

く, マイコンを使わなければできない便利で高度で魅力的なものを目指します. マイコンのファームウエアで極力処理をすることで電子回路は簡潔になり, ファームウエアの改良で機能の向上が容易にできるという利点もあります.

本書には多くの内容があるため, すべてを理解する必要はありません. 技術的な興味が出てきてから先のステップへ進めば十分です. 本書の内容を深く理解すれば, DCC デコーダの完全自作で必要となる基礎技術も身につけることができますので, ぜひとも挑戦して下さい.

1.2　鉄道模型を楽しむ 4 つのモデルコース

本書で紹介する装置の組み合わせ次第で, さまざまな鉄道模型の楽しみかたができます. ここでは 4 つのモデルコースを紹介します (図 1.3). 本書で紹介する装置はごく限られた機能しか持たないものが多いですが, それらを相互に接続することで複雑なシステムを構成できるように意図しています.

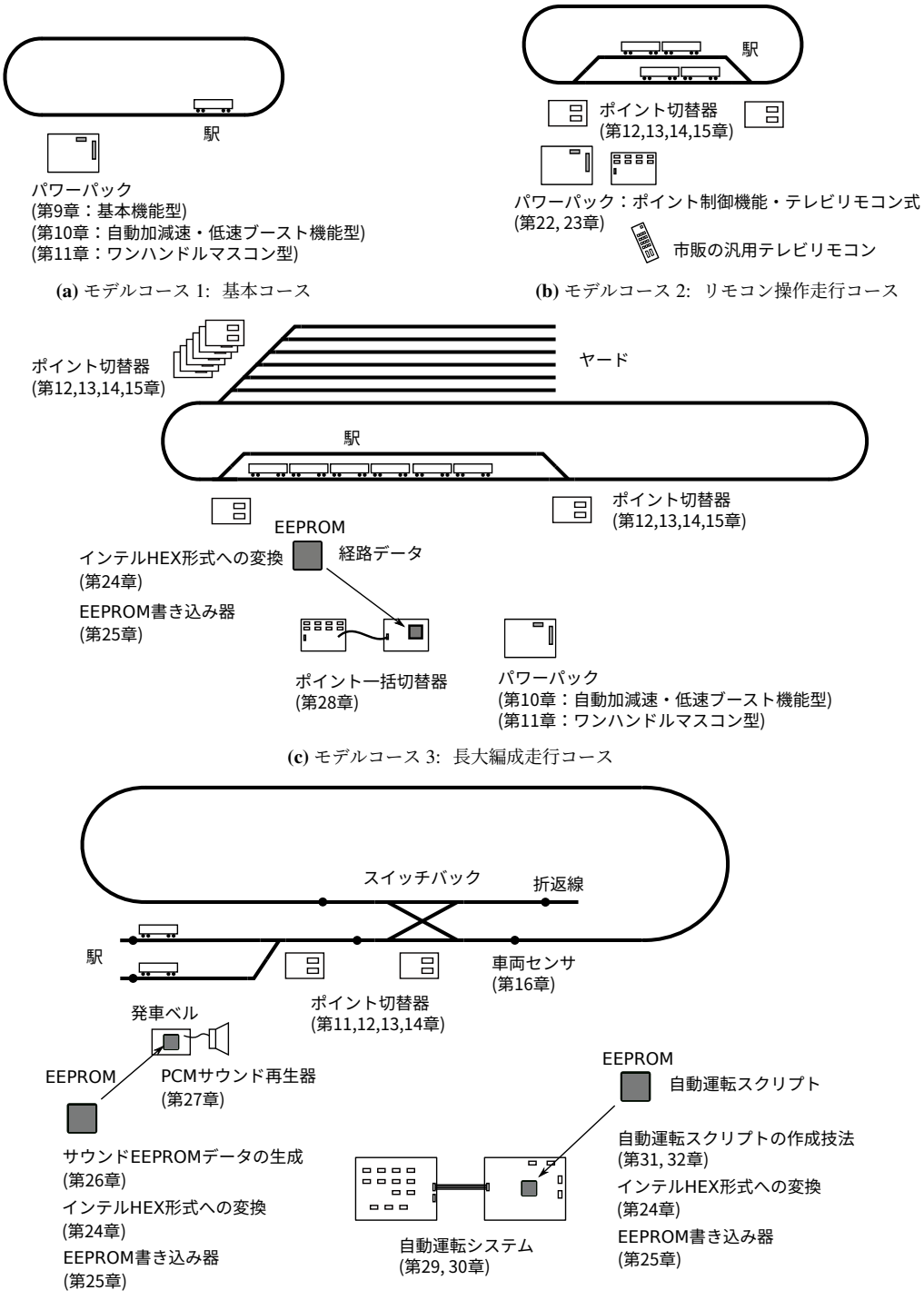
モデルコース 1: 基本コース

図 1.3a に示すように, 手軽・気楽に楽しむエンドレスのレイアウトで, パワーパックひとつの最小構成です. パワーパックにもさまざまな機能を持ったものを紹介しています. いずれもコアレスモータ搭載の模型車両でもスムーズに加減速できます. (ただし一部の模型車両では未対応の場合があります.)

走行を楽しむには, トランジスタコントローラのような自動加減速機能や, 低速ブースト機能 (超低速走行を可能にする機能) を持つものをおすすめします. 基本機能型は学習を主眼にしたものですから, 実用にはあまりおすすめしません. ポイントを設置して交換可能駅のレイアウトなどにも発展できます. オプションとして, PCM サウンド再生器で発車ベルを鳴らしたり, 信号機や走行速度測定器を導入しても楽しいでしょう.

モデルコース 2: リモコン操作走行コース

図 1.3b に示すように, 列車交換可能な駅を持つレイアウトで, 速度やポイントの切り替えはリモコン操作により離れたところから走行を楽しむ構成です. パワーパックには, リモコン機能とポイント制御機能を持つものを使用します.



(a) モデルコース 1: 基本コース

(b) モデルコース 2: リモコン操作走行コース

(c) モデルコース 3: 長大編成走行コース

(d) モデルコース 4: 自動運転コース

図 1.3: 4 つのモデルコース

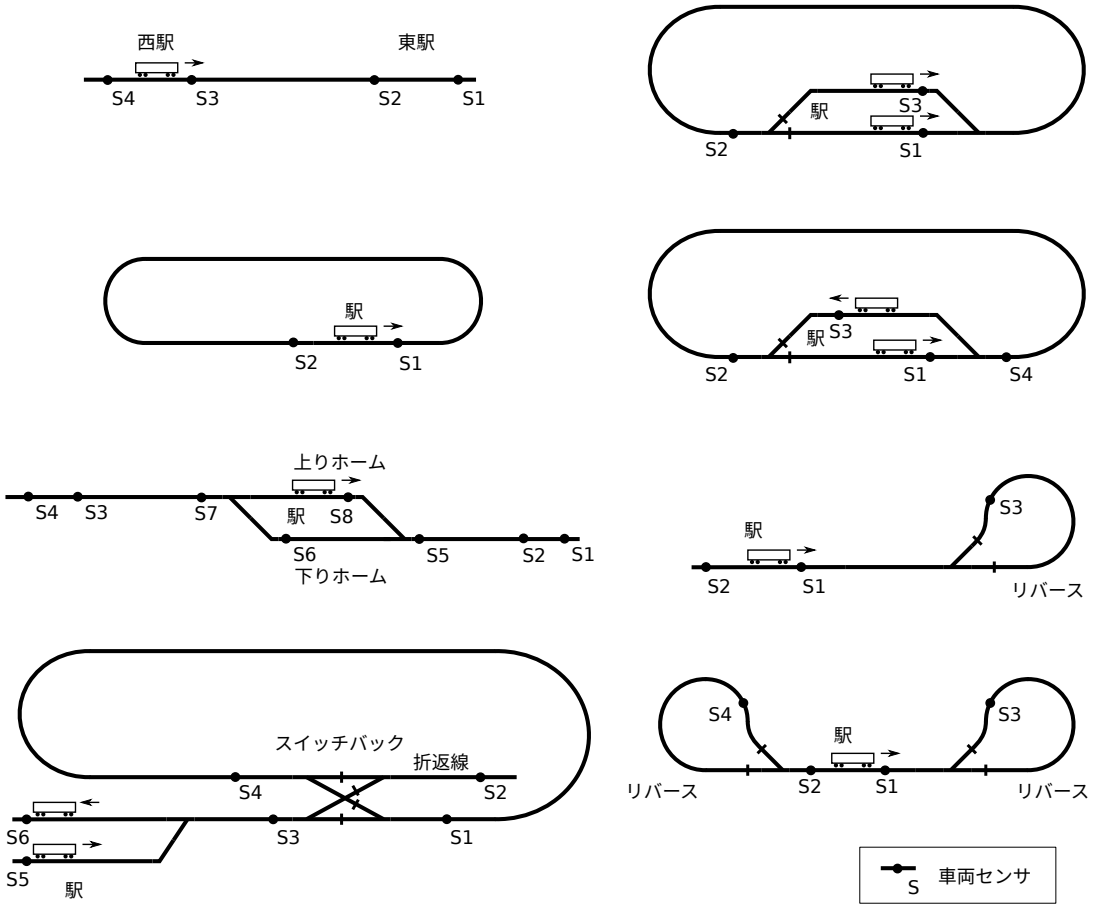


図 1.4: 本書で紹介する自動運転スクリプトのレイアウト例

モデルコース 3: 長大編成走行コース

図 1.3c に示すように, 様々な編成が控えるヤードを持つレイアウトで, 長大編成の列車のロングランを運転を楽しむ構成です. ポイント一括切替器により, 入換作業でのポイントの切り替えを簡単にします.

モデルコース 4: 自動運転コース

図 1.3d に示すように, 自動走行を行い, 走行の様子を眺めて楽しむ構成です. 単線を往復するだけの小さなレイアウトでの手動運転は単調で退屈ですが, 自動運転システムがあれば十分に楽しめます. 従来の自動運転システムでは, 駅に列車を停止線ぴったりに停車させるために面倒で微妙なブレーキのパラメータ調整が必要でした. 本書の自動運転システムでは, 自動学習機能によってそのような面倒な調整は不要です. 線路を敷き, センサを装置に接続して走行開始ボタンを押せば, あとは自動運転システムがブレーキパラメータを自動的に学習します.

本書では図 1.4 に示すレイアウトに対する自動運転パターンを紹介します. ここに示すものだけでなく, 運転スクリプトを書くことでさまざまなレイアウトに合わせた独自の自動運転が実現可能です.

1.3　ハイライト: 本書で製作する装置の代表例

本書で特に興味深い内容をいくつか挙げてみました. なお本書における各章の相関図は図 1.5 に示す通りです.

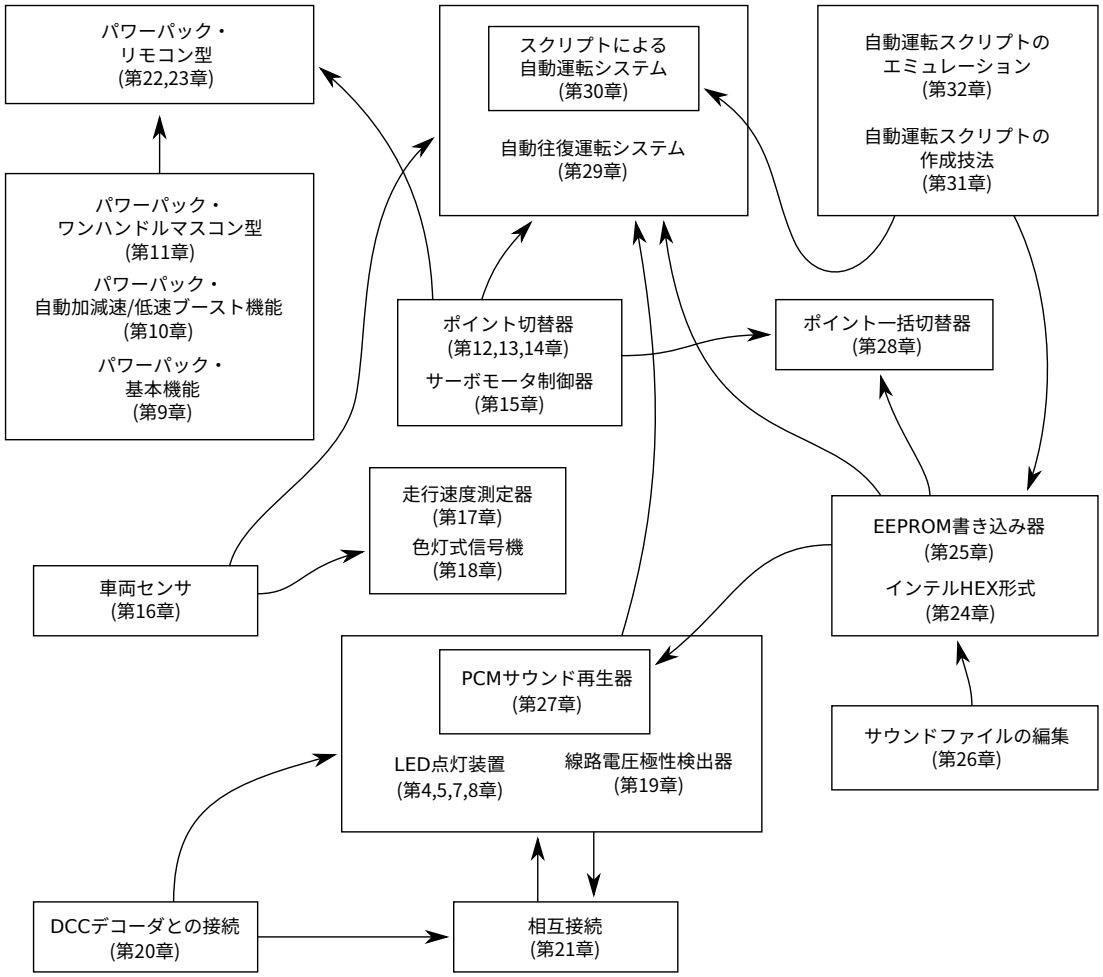


図 1.5: 各章の相関図

PCM サウンド再生器

IC レコーダなどを使って自分で録音した音源や動画 Web サイトからダウンロードした音源などを PCM サウンドデータにして, EEPROM に書き込んで再生する装置です. 例えば駅の発車メロディや踏切の音など, 録音された実際の音の再生が可能となります. (第 27 章で紹介)

DCC デコーダとの接続

DCC デコーダのファンクション出力に, 本書で紹介する各種アクセサリ機器を接続して制御する方法を紹介します. これにより DCC スロットルから, 例えば LED 点灯装置や PCM サウンド再生器の操作が可能となります. (第 20 章で紹介)

走行速度測定器

車両の走行速度を測定し, 7 セグメント LED で表示します. 本装置は車両が 2 つの車両センサ間を移動する実際の時間と車両センサ間の物理的な距離を元に, km/h 単位の走行速度に換算します. (第 17 章で紹介)

従来の装置に見られるような, 線路電圧を測定して比例計算により速度に換算する擬似的な手法とは異なります. 模型のスケール (例えば 1/150 や 1/80 など) と車両センサ間の距離は, DIP スイッチとボリュームを使っ

て設定変更が可能にしてあり, さまざまなスケールの模型や設置環境にて使用可能です.

色灯式信号機

2 灯式から 6 灯式までの色灯式信号機を, ひとつのファームウエアで 11 通りの現示パターンを実現します. これには毎分 80 回点滅するフリッカー信号や高速現示も含まれています. また, ハードウエア的には LED を光源に使用しますが, 電球を模倣する点滅の輝度制御を行い, 電球を使用した信号機の点灯イメージを実現する機能も DIP スイッチで選択できます. (第 18 章で紹介)

11 通りの現示パターンの実現には, パターン個別にプログラムコードを書くのではありません. 点灯パターンをデータとして表現し, 現示の切り替えはデータを参照して実行してゆく方式のプログラム構造です. 現示パターンの選択は, データの選択により行います. これはデータの形で動作仕様を表現する技法であり, 自動運転における自動運転スクリプトの実行方式の源流です. 自動運転スクリプトは, データの形で運転動作の仕様を表現しているものだからです.

パワーパック: 自動加減速・低速ブースト機能型

PWM 周波数 31.25kHz でモータを駆動する PWM 式パワーパック装置です. トランジスタコントローラのように加減速を自動で行う機能と常点灯機能を有しています. モータドライバ IC を利用していますので, 大変簡単に組み立てが行えます. またコアレスモータ搭載の模型車両でもスムーズに加減速できます. (第 10 章で紹介)

そしてファームウエアにおける大きな特徴は, 低速ブースト機能を実現している点です. 低速ブースト機能とは, 出力電圧が小さい低速走行時にモータのトルクをブースト (増強) する機能で, 超低速走行が可能になります. 特に自動運転の際には模型の走行に驚くほどの実感性があり, 模型の世界についつい没入してしまうほどです.

PWM 周波数が高いとモータの特性上, PWM デューティ比が低いときのトルクが低くなります. それゆえに低速ブースト機能のない普通のパワーパックでは, 走行の開始時には模型車両はなかなか走り出しません. パワーパックのスピードボリュームを上げてゆくと急発進 (いわゆるラピッドスタート) となり, 走行はあまり実感的ではありません. そこで本装置では, PWM デューティ比が低いときには 1 秒間に数十回, 高 PWM デューティ比の信号をわずかな時間だけ加えて, モータの静摩擦力を越えたトルクを与えて超低速走行を実現します. この機能の効果を図 1.6 に示します.

テレビリモコン式パワーパック

汎用の赤外線テレビリモコンを使用して, 遠隔制御を行うパワーパック装置です. (第 23 章で紹介)

本装置の特徴はポイント切替制御機能も有している点です. 大きなレイアウトで, あるいはくつろぎながら楽しむにちょうどよいものです. テレビリモコン信号のパケットを受信・解析するソフトウエア技法もすべて解説します. この技法を理解すれば, DCC デコーダのパケット受信部分の自作もできるようになります.

自動運転システム

制御シーケンスを独自のスクリプト言語で記述し EEPROM に格納することで, さまざまなレイアウトや走行パターンに対応可能です. (第 29 章と第 30 章で紹介)

自動運転スクリプトの例を図 1.7 に示します. 自動運転パターンの記述に, 高度な専門知識を必要とする C 言語, Java, Python などの汎用的なプログラミング言語は使用しません. 本書では独自のスクリプト言語を新たに設計・定義しますが, これは鉄道模型の自動運転に特化している言語なので, 専門知識は不要で, 容易に自動運転パターンの記述が行えます. このように特定の応用タスク向けに設計されたコンピュータ言語はドメイン固有

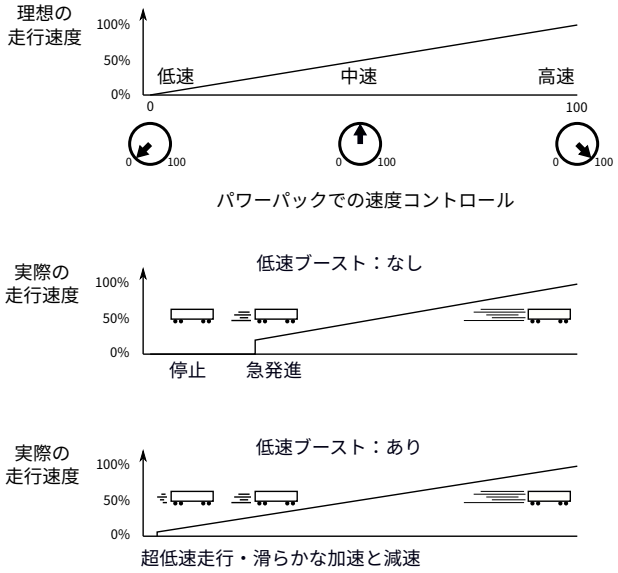


図 1.6: 低速ブースト機能の効果

言語 (domain-specific language; DSL) と一般に呼ばれるもので, 本書ではその設計と実装の一例を紹介します.

図 1.7 の例では, コメントを除けば自動運転スクリプトは実質的にわずか 8 行で, 駅を発車, レイアウトを走行, 駅に停車 (減速度は自動調整), の繰り返しの動作を記述しています. スクリプト言語の機能には確率的な動作を可能とするコマンドがありますので, 例えば機関区で機関車がランダムに行き来するシーンなど, ずっと見ていても飽きない自動運転が実現できます.

本装置の特徴は, センサ位置への的確な停車にあります. 目的とする停止センサ位置へぴったりとしかも滑らかに停車するように, 減速度パラメータを自動学習する機能を有しています (特許取得済み [18]). この機能により, 模型車両ごとに手作業による試行錯誤しながらの絶妙なパラメータ調整から開放されます.

自動運転スクリプトのベリフィケーション機能

自動走行の途中で「予期しないエラー」が発生しないように, 自動運転スクリプトのエラーを予め検出する機能を持たせます. (第 30 章で紹介)

自動走行の開始前には自動運転スクリプトを解析 (静的解析) して, もしエラーが検出されれば自動走行の実行は行いません. そのため自動走行の途中での強制終了はまったく発生しません.

自動運転スクリプトのエミュレーション実行

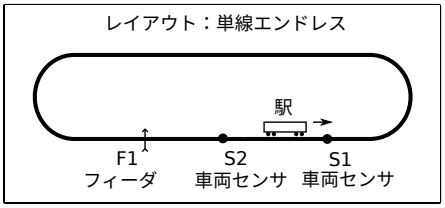
本書では, パーソナルコンピュータ上で自動運転スクリプトを擬似的に実行して動作の確認（動的解析) を可能とするエミュレータを用意しました. (第 32 章で紹介)

最終的に完成する自動運転スクリプトがどんなに簡潔であったとしても, もしエミュレータがなければ, 完成するまでのデバッグ作業は極めて不便です. 自動運転スクリプトの開発の手順は, (1) パーソナルコンピュータ上で自動運転スクリプトを設計・修正をし, (2) EEPROM に書き込み, (3) EEPROM を自動運転システムに装着し, (4) 実際に自動運転を行って動作確認し, (5) もし不具合があれば (1) に戻る, という流れです. 単なる 1 文字のタイプミスであっても不具合が見つかる度に, この手順を繰り返さないといけません. そのため自動運転スクリプト言語に不慣れなうちは, ずいぶんと時間と労力と気力を失ってしまいます.

エミュレータには上記のベリフィケーション (検証) の機能も持たせており, これを用いることでほとんどの

自動走行スクリプト：単線エンドレス用

```
#----------------------------
# 単線エンドレス
#----------------------------
#   減速度番号割当
#     1: 駅
#   センサ番号割当
#      S1: 駅, 停止
#      S2: 駅, 制動
#----------------------------
# 駅, 停車(東向)
# 初期化
  D 1 1;         # 東行 (トラック1: 順方向)
  M 1;           # モーター&常点灯On
# ■ 繰返点
:1:
  # 駅を出発, 走行
  R 32 1 2 1;  # 速度32(100%), 減速度番号 1, 制動:S2, 停止:S1
# ■ 定位置 (駅)
:0:
  # 駅, 停車
  S 32;          # 待機時間 32(=100%)
  # 繰返し
  J 1;
# データ終了
/
#----------------------------
```



自動走行スクリプト内のコマンド（一部）

```
#
    コメント(行末まで)
:l:
    ラベルlを定義
D f x;
    フィーダfの極性をx(前進/後進)に設定
M x;
    常点灯出力をx(オン/オフ)に設定
R p d sb ss;
    速度pへ(加速)走行し、
    指定センサに到着したら停車
    センサsbで制動開始、センサssで停車
    自動学習パラメータ番号はd
S t;
    時間tの経過を待機
J l;
    指定ラベルlに実行を移す
? l p;
    確率p/10で指定ラベルlに実行を移す
```

図 1.7: 自動運転用スクリプトの例

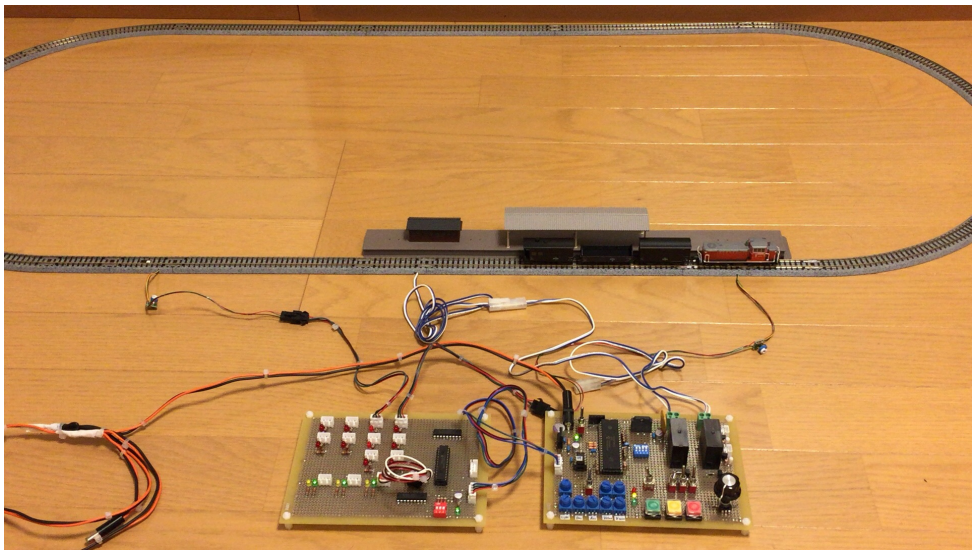


図 1.8: エンドレスレイアウトでの自動運転システムの例

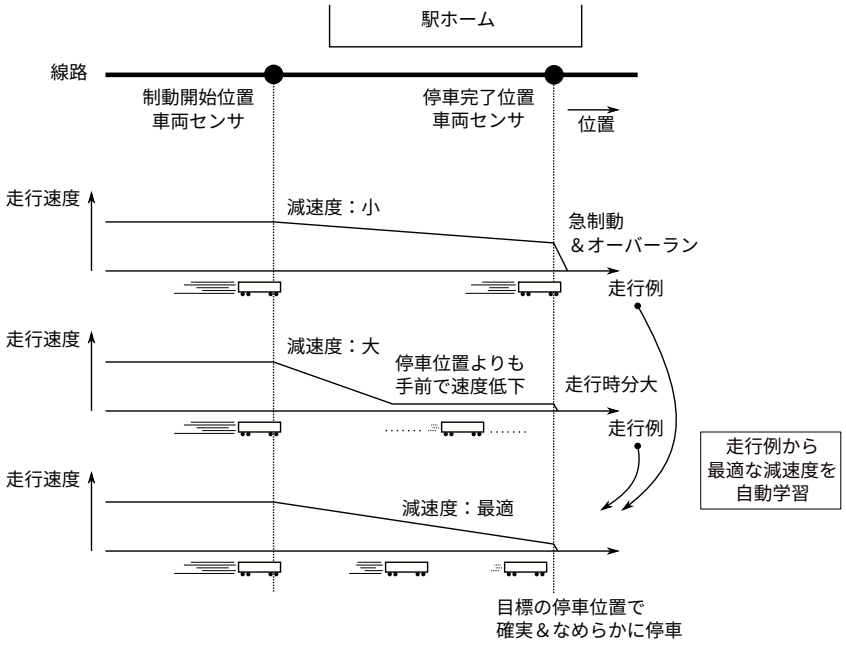


図 1.9: 自動運転における停止のための最適な減速度の自動学習

不具合をパーソナルコンピュータ上で発見・修正できますから, 大幅に時間と労力が削減できます. エミュレータを用いた自動運転スクリプトの開発は, (1) パーソナルコンピュータ上で自動運転スクリプトを設計・修正をし, (2) パーソナルコンピュータ上でのベリフィケーションとエミュレーションにより動作確認をし, 不具合があれば (1) に戻り, そして (3) EEPROM に書き込み, (4) EEPROM を自動運転システムに装着し, (5) 実際に自動運転を行って動作の最終確認をする, という流れです.

ベリフィケーションとエミュレーションを実現するプログラムコードは意外と簡単に実現できていますが, これも簡明な独自のスクリプト言語を設計することの大きな利点です.

1.4　本書で取り扱わないこと・取り扱うこと

本書はマイコンのファームウエアを中心に解説します. すでに電子工作の入門書や, マイコンの機能を詳細に解説した書籍は多数出版されておりますが, 鉄道模型向けのマイコンのファームウエアを細かに解説した書籍はまだ出版がされていないようです. そこで本書がその役割となることを意図しています. 多くの分量の題材があり, また紙面も限られていることから, 本書で取り扱わないことと取り扱うことを以下の通りとします.

取り扱わないこと

- 電子工作の基本
  近年は初心者向けで優れた電子工作の書籍が多数出版されていますので, 例えば工具の説明やハンダ付けの技法などの詳細は他書を参照して下さい.
- 電子・電気工学の基本
  本書では高度な電子・電気工学は使用しないようにして, できるだけマイコンのファームウエアで処理をします. 電子工作の入門書を読んでおくと十分でしょう.
- 使用する部品の詳細な解説
  最小限のことは本書で説明しますが, それを超えた内容は各自でデータシートを読むようにして下さい.
- AC100 ボルト

高性能で低価格な市販の AC アダプタを利用して, 感電事故の元は極力廃するとともに工作の手間を削減します.

- PIC マイコンの詳細な機能ならびにソフトウエア開発環境の使用法の解説
  本書では Microchip Technology 社のマイコン PIC 16F シリーズを用い, 統合開発環境 MPLAB X IDE 上でファームウエア開発を行いますが, これらの詳細については優れた他書が多数ありますので必要に応じて各自で参照して下さい. 本書では PIC マイコンの包括的な解説は行わず, 本書で紹介する装置の理解に必要最小限の解説に留めておきます.
- ケース加工, 模型車両やレイアウトへの組み込み
  ケース加工は好みが大きく分かれるところですし, 模型のスケールも Z スケールのような小さなものから O スケールのように大きなものまで様々で, 一概には言えないことがたくさんあります. 各自で工夫をして下さい.
- C 言語の解説
  近年では初心者向けの優れた解説書が多数出版されていますで, C 言語に不慣れな方はそちらを参照して自習して下さい.
- 複数の模型列車の同時走行制御
  本書では小規模なレイアウトを対象として, 1 編成の模型列車を走行させることに限定します. 大規模なレイアウトで複数の模型列車を同時に走行させることは目指しません. ギャップ切りや線路への給電の切り替えが面倒なので, アナログ方式は止めて DCC 方式に移行するのが最終的には得策だと思います.

取り扱うこと

- アルゴリズムの解説
  ファームウエアを構築する際に有用な設計技法や各種アルゴリズムを紹介します. 具体的には, 状態遷移システム設計法, 割り込み処理, 入出力バッファリング, スクリプト構文解析, スクリプト実行機構などです. またモータ回転数や LED 輝度の算出で有用なブレゼンハムアルゴリズムを紹介します. 表 1.1 に, 本書で学ぶプログラミング技法を示します.
  自動運転システムでは C 言語のソースコードが 4000 行を越えますが, ソフトウエア機能ごとに細かに分割をして実装してあります. 設計容易性, 理解可能性ならびに保守可能性を高めるには, 抽象化が大きな鍵となります. 抽象化の技法についても学んでゆきます.
- 全ソースコードの掲載と解説
  上記の目的を達成するためにソースコードはすべて掲載します. ソースコードは全体を合わせると 1 万 4 千行を越えますが, ほとんどのソースファイルでは 200 行程度以下となるように, しかも論理的な機能単位となるように注意深く分割をして, 理解しやすい工夫をしています. またソースファイルはできるだけ再利用するように構成しています. 参考として本書のプログラム規模の統計を以下に示します.

全プロジェクトのソースファイル延べ行数	27892 行	(= A)
実際に開発したソースファイル総行数	14216 行	(= B)
再利用されたソースファイル延べ行数	13676 行	(= $A - B$)

- 本書執筆時点で最新の PIC マイコン 16F シリーズ
  本書では, マイクロチップ・テクノロジー社 PIC マイコン 16F18xxx(5 桁) シリーズの 16F18877, 16F18857, 16F18346, 16F18326, 16F18313 を使用します. 図 1.10 にそれらの一部を示します. 以前の

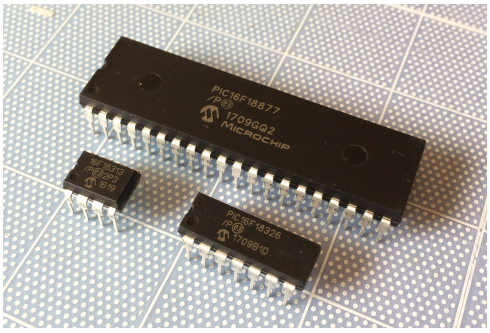


図 1.10: PIC マイコンの例 (上から右回りに 16F18877, 16F18326, 16F18313)

16F18xx(4 桁) シリーズと比べて高度な内蔵周辺モジュールと十分なメモリ容量が搭載され, 加えて低価格なので, 多彩な機能を余裕を持って実現可能です. 本書で製作する装置で使用する PIC および内蔵周辺モジュールを表 1.2 に示します.

統合開発環境の MPLAB X IDE や C コンパイラ XC8 は, マイクロチップ・テクノロジー社から Windows 版, MacOS 版, Linux 版いずれも無料でダウンロードできます. ファームウエアをマイコンチップに書き込むために, 安価で購入可能な PICkit4 プログラマを使用します. 開発環境一式は極めて低価格で揃います. ただし開発環境を動作させるためのパーソナルコンピュータは必須ですので, 各自で用意をして下さい.

入門的なこと一切を他書に譲るのは, 限られたページ数で上記を達成するためです. そのため入門者にとって不親切な点が多々ある点はご容赦下さい. 他書では見られないことに本書は注力します.

1.5　小規模から大規模まで自由自在なシステム構築

本書では, 鉄道模型を楽しむ各種の電子工作を小さな装置に分解して, 装置ごとに作成する方式で進めてゆきます. そして各装置は相互に結合して連携した動作が可能な設計にします. LED 点灯装置や PCM サウンド再生器などの装置は, インターフェース部を少し工夫をしてあり, 手動によるスイッチや自動運転装置などの装置類のどちらからでも操作を可能にしていますので, ひとたび製作した装置は無駄になることなく広く永く利用が可能です.

この方式の対極にあるのが単一の巨大システムです. ハードウエアコストに無駄がないといえば確かにそうなのですが, あらかじめ厳密に仕様を定めて作成する必要があり, 巨大であるが故に設計・製作および動作検証には大変な手間を要し, さらに機能拡張や変更が困難という欠点があります. ただし固定的な機能を持つ装置を大量に生産する場合にはこの方式となるでしょう.

本書の方式の利点を詳しく説明します. まず, 機能ごとに独立して各装置の設計ができる点が利点として挙げられます. つまり個々の機器は, 同一のはたらきを持つ別の装置に容易に置き換えができます. ある装置の性能が気に入らなければより性能の良い装置に置き換えれば良いですし, 一部が故障した場合でもその装置の交換だけで済みます.

システムをその時々の事情に合わせて順次拡大・縮小が容易にできるのも利点です. 例として, 図 1.11 に自動運転システムの最小構成と最大構成を示しています. 接続する車両センサ, ポイント切替器, アクセサリ類もそれぞれ独立に設計・製作ができ, 自動運転システム本体は無改造のままで, 構成を様々に変更できます. つまり, 手持ちの装置を自由に組み替えることで, 様々なレイアウトに対応が可能です.

はたらき自体は同じでも, 制御対象ごとの仕様に合わせた装置を作ることも可能です. この典型的な例がポ

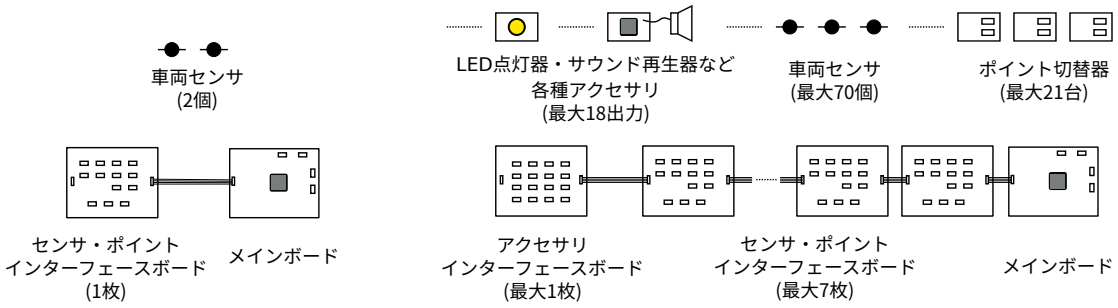
表 1.1: 習得するプログラミング技法

章	内容	開発環境の使い方	LED 輝度の PWM 制御	DC モータの PWM 制御	状態遷移手法	プレゼンハムアルゴリズム	固定小数点演算	疑似乱数	タイマによる計時	タイマ割り込み	入出力割り込み	入出力バッファリング	7セグメント LED	LED ダイナミック点灯	チャタリング除去	ロータリエンコーダ	RS-232C 通信	I²C EEPROM IC	I²C I/O エクスパンダ IC	インテル HEX ファイル	WAV オーディオファイル	PCM サウンド	赤外線テレビリモコン受信	データによる動作仕様記述	ドメイン固有言語 (DSL)
4	電子工作の基礎 (LED 点灯)	✓																							
5	プログラミングの基礎 (LED 点灯)		✓		✓																				
7	LED の点灯 (基礎)		✓		✓	✓																			
8	LED の点灯 (発展)		✓		✓	✓	✓	✓																	
9	パワーパック (基本)			✓																					
12, 13	ポイント切替器				✓																				
15	サーボモータ制御器				✓	✓			✓																
10, 11	パワーパック (低速ブースト)			✓	✓	✓				✓															
17	走行速度測定器								✓	✓			✓	✓											
18	色灯式信号機		✓		✓	✓			✓															✓	
19	線路電圧極性検出器														✓										
22	パワーパック (リモコン)			✓	✓	✓				✓													✓		
23	パワーパック (リモコン＋ポイント制御)			✓	✓	✓													✓						
24	インテル HEX 形式への変換																		✓						
25	EEPROM 書き込み器										✓	✓					✓	✓		✓					
26	サウンド EEPROM データ																		✓	✓	✓				
27	PCM サウンド再生器									✓		✓			✓			✓				✓			
28	ポイント一括切替器									✓			✓	✓	✓	✓		✓	✓	✓				✓	✓
29	自動往復運転システム			✓	✓	✓				✓									✓						
30	スクリプトによる自動運転			✓	✓	✓		✓		✓								✓	✓					✓	✓
31	自動運転スクリプトの作成技法																		✓					✓	✓
32	自動運転スクリプトのエミュレーション																							✓	✓

表 1.2: 使用する PIC と内蔵ペリフェラルモジュール

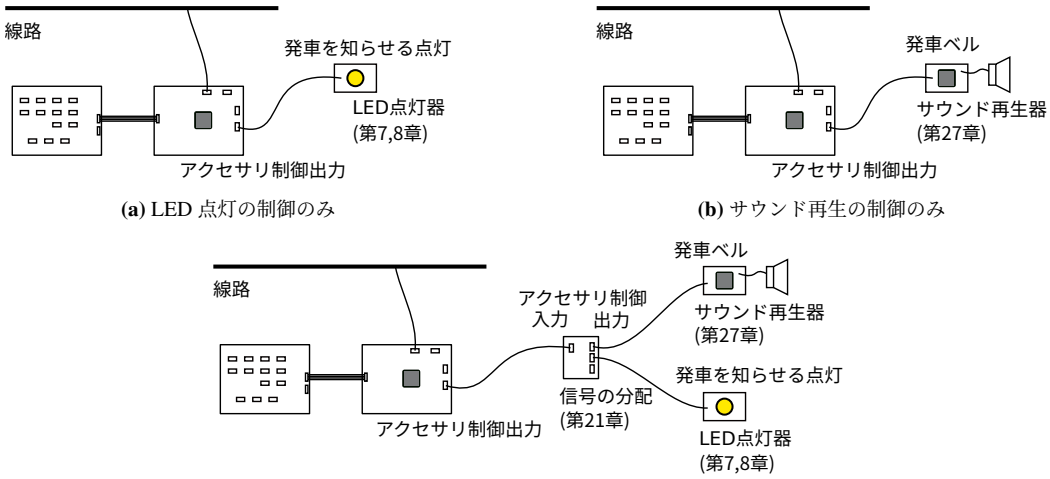
章	内容	PIC	ADC	CCP	CWG	CM	EUSART	FVR	IOC	MSSP	PORT	PPS	TMR0	TMR1/3/5	TMR2/4/6	WDT	備考
4	電子工作の基礎 (LED 点灯)	16F81313									✓						
5	プログラミングの基礎 (LED 点灯)	16F81313	✓	✓							✓	✓			✓	✓	
7	LED の点灯 (基礎)	16F81313	✓	✓							✓	✓			✓	✓	
8	LED の点灯 (発展)	16F81313	✓	✓							✓	✓			✓	✓	
9	パワーパック (基本)	16F18857	✓	✓	✓						✓	✓			✓	✓	モータ Dr
10, 11	パワーパック (低速ブースト)	16F18857	✓	✓	✓						✓	✓		✓	✓	✓	モータ Dr
12, 13	ポイント切替器	16F18313									✓					✓	モータ Dr
15	サーボモータ制御器	16F18326	✓								✓			✓		✓	
17	走行速度測定器	16F18857	✓								✓			✓	✓	✓	
18	色灯式信号機	16F18346	✓	✓	✓				✓		✓	✓			✓	✓	
19	線路電圧極性検出器	16F18326				✓		✓			✓					✓	
22	パワーパック (リモコン)	16F18857	✓	✓	✓				✓		✓	✓		✓	✓	✓	モータ Dr
23	パワーパック (リモコン, ポイント制御)	16F18857	✓	✓	✓					✓	✓	✓		✓	✓	✓	モータ Dr, I^2C, I/O Exp
25	EEPROM 書き込み器	16F18326	✓				✓			✓	✓	✓				✓	RS-232C, I^2C, EEPROM
27	PCM サウンド再生器	16F18313		✓							✓	✓	✓	✓	✓	✓	I^2C, EEPROM
28	ポイント一括切替器	16F18857								✓	✓	✓		✓		✓	I^2C, I/O Exp, EEPROM
29	自動往復運転システム	16F18877	✓	✓	✓					✓	✓	✓		✓	✓	✓	モータ Dr, I^2C, I/O Exp
30	スクリプトによる自動運転	16F18877	✓	✓	✓					✓	✓	✓		✓	✓	✓	モータ Dr, I^2C, I/O Exp, EEPROM

ADC: AD 変換, CCP: キャプチャ/比較/PWM, CWG: 相補波形ジェネレータ, CM: コンパレータ, EUSART: 非同期通信, FVR: 固定参照電圧, IOC: 入力変化割り込み, MSSP: 同期シリアル通信, PORT: デジタル I/O ポート, PPS: ペリフェラルピンセレクト, TMR0, TMR1/3/5, TMR2/4/6: タイマ, WDT: ウオッチドッグタイマモータ Dr: モータドライバ IC TB6643KQ, I/O Exp: I/O エクスパンダ IC MCP23017



(a) 最小構成　　**(b)** 最大構成

図 1.11: 拡張可能な自動運転システム



(a) LED 点灯の制御のみ

(b) サウンド再生の制御のみ

(c) LED 点灯とサウンド再生の同時制御

図 1.12: アクセサリ出力への様々な機器の接続

イント切替器です. ポイントマシンの駆動には, 2 線式ソレノイド (KATO や TOMIX など), 3 線式ソレノイド (PECO など), DC モータ (Fulugrex など), サーボモータ等, 様々な電気的特性や機構を持ったものが使われています. そこで本書ではポイント切替器の制御信号の共通インターフェース (ポイント制御入力) を定めて, ポイントマシンごとの差異を個々のポイント切替器で吸収します. つまり利用する側は, ポイントマシンの電気的特性や機構を気にすることなく, 共通インターフェースに従った信号を送るだけでポイントの切り替え制御が可能です. ポイント一括切替器 (第 28 章）や自動運転システム (第 29 章, 第 30 章) は共通インターフェースに従ってポイントの切替を行いますから, ひとつのレイアウトに様々な種類のポイントマシンの混在が可能です. 使用するポイントマシンの種類に合わせて, ポイント一括切替器や自動運転システムの回路の手直しは一切不要です.

もうひとつの例が, PCM サウンド再生器 (第 27 章) や DCC デコーダとの接続 (第 20 章) などの各種アクセサリで採用している制御方式です. こちらについても共通インターフェース (アクセサリ制御入力およびアクセサリ制御出力) を定めて, 外部からの制御を電気的に統一した方法で行えるようにしています. 制御信号の分配/集約は, 接続ケーブルのコネクタを挿すだけで行える相互結合用の装置 (第 21 章) も紹介していますから, 単に接続ケーブルのコネクタの抜き差しで本書の装置を結合して, 自由自在にシステムを構築することが可能です.

この利便性を図 1.12 に示しています. 自動運転システムで列車の出発を知らせるアクセサリ類を追加したい

場合を考えます. LED 点灯器だけの接続 (図 1.12a), PCM サウンド再生器だけの接続 (図 1.12b), 両方の同時接続 (図 1.12c) いずれでも, 自動運転システム本体に変更は一切不要です. もちろん押しボタンによる手動操作も可能です.

本書で紹介する装置はどれも単機能に徹していますので, 設計は単純, 理解が容易, 間違いが入り込みにくいという長所があり, ひとつひとつ順に学んでゆくことが容易です. みなさまには是非とも理解を確実なものにし, 更に優れたものを生み出されることを期待いたします.

第Ⅰ部

鉄道模型とマイコン電子工作の基本

第 2 章

電気式鉄道模型工学概論

本章では鉄道模型の電気工学的側面を概説します. ここでは以下の内容を学びますが, すでに熟知されている方はこの章を読み飛ばして構いません.

- 鉄道模型の走行制御 (アナログ/DCC, など)
- 鉄道模型への電気の供給 (2 線式/3 線式, など)
- ポイントの切り替えと線路の極性 (選択式/非選択式, など)
- ポイントマシンの制御 (ソレノイド/DC モータ, など)
- フィーダとギャップ (線路への電気の供給, およびレール間の絶縁)

鉄道模型での電気工学に不慣れな方は, 熟読して内容の理解につとめて下さい. この章の内容の理解が不十分だと, ショートを起こして模型車両が走行できないだけでなく, 場合によっては部品の焼損や模型車両の破損となる場合がありますから, 大変重要な内容です.

本書で特に対象する模型の走行方式は, 電気エネルギーでモータを回転させて車輪を駆動して走行をする方式です. 現在の主流はこの方式の鉄道模型ですが, 他の方式も使用されています. その代表的な例がライブスチームで, 大型鉄道模型での一大分野を成しています. これは石炭やアルコールの燃焼による熱をエネルギー源にして蒸気を発生させて, 昔の蒸気機関車と同様な機構で走行する方式です.

最初に走行の制御方式と模型車両への給電方式の 2 つの観点より模型車両の走行の実現方法を概説します. 次にポイント (分岐器) の電気的な特性を概説します.

2.1 制御方式

模型車両の走行を制御する方式を大別すると以下の分類ができます.

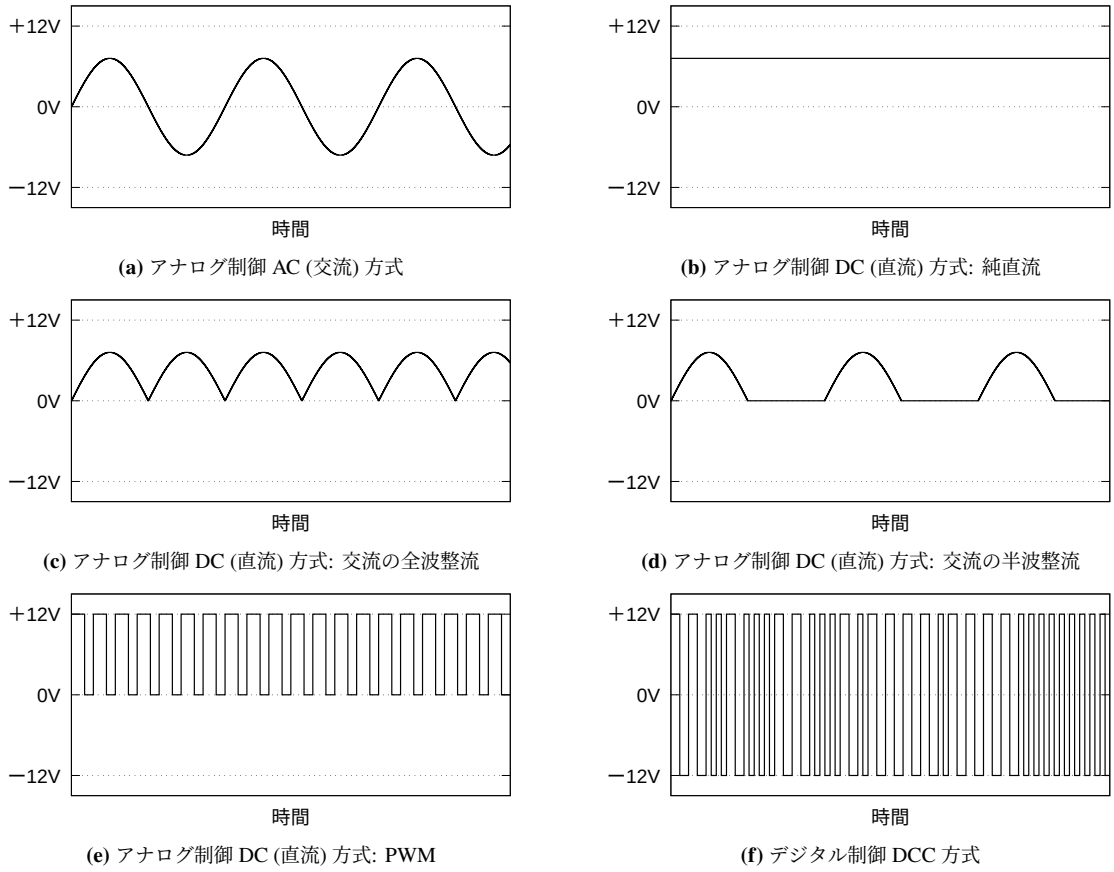
走行制御方式の分類

アナログ制御方式
- AC (交流) 方式
- DC (直流) 方式

デジタル制御方式
- DCC (Digital Command Control) 方式
- Märklin Digital 方式
- Selectrix 方式

2.1.1 アナログ制御方式

アナログ制御方式は, 模型車両に給電する電圧の大小を制御することにより模型車両の走行速度の高低を変化させます. 模型車両が受けた電圧はモータへ直接接続されていますので, 電圧の大小に従ってモータの回転速



(a) アナログ制御 AC (交流) 方式

(b) アナログ制御 DC (直流) 方式: 純直流

(c) アナログ制御 DC (直流) 方式: 交流の全波整流

(d) アナログ制御 DC (直流) 方式: 交流の半波整流

(e) アナログ制御 DC (直流) 方式: PWM

(f) デジタル制御 DCC 方式

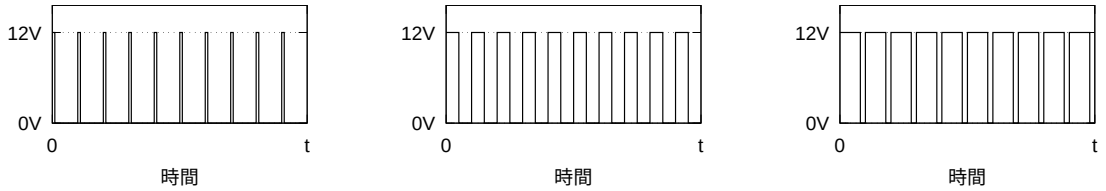
図 2.1: 各種走行制御方式での電圧波形

度が変化します. 給電する電圧の高低を制御する装置はパワーパックと呼ばれており, 模型車両を運転する人はこれを操作します. アナログ制御方式の特徴は単純明解さにあります. 模型車両とパワーパックのどちらも簡単な電気回路で済むため, いずれも簡単に自作が可能です.

アナログ制御方式では, 線路を使用して給電する場合がほとんどです. 制御しているのは線路へ印加する電圧であり, 模型車両を個別に制御をしていない点に注意が必要です. すなわち複数の模型車両が同じ線路上にあれば, 同じ電圧を受けて同時に走行します. 機関車の重連や 1 編成に複数の動力車を組み込む場合にはこの方式は簡単で良いのですが, 複数の模型列車がある場合には少し複雑になります. 停止しておきたい模型列車に給電がされないよう, ギャップを切ってセクション分けしたり, 選択式ポイントの切り替えの利用が必要になるからです. 従って大規模なレイアウトや複数の模型車両の配置を行う場合には, 電気配線を入念に設計しなくてはなりません.

AC 方式は, 模型車両に給電する電圧に交流 (alternating current; AC) を使用する方式です. 図 2.1a にこの方式の電圧波形を示します. 現在この方式は主に Märklin 社のアナログ式 HO スケールの模型で採用されています.

DC 方式は, 模型車両に給電する電圧に直流 (direct current; DC) を使用する方式です. 現在この方式がアナログ制御方式の主流を占めており, KATO 社や TOMIX 社等の N ゲージや 16 番の模型で採用されています. DC 方式の多くでは線路の左右の 2 本のレールを使用して模型車両に電力の供給を行います. 線路に印加する電圧の標準は –12 ボルトから +12 ボルトの範囲です. 電圧値の絶対値の大小でモータの回転速度の高低が決まり, 電圧値の極性でモータの回転方向が決まります. 電圧値がゼロの場合にはモータは停止します. 進行方向と電



(a) 10%　**(b)** 50%　**(c)** 80%

図 2.2: PWM デューティ比と波形

圧極性の関係は, 進行方向の右側のレールをプラス極性とするのを標準としています.

DC 方式において印加する電圧は必ずしも純直流 (図 2.1b) である必要はなく, パルス的な電圧であっても構いません. 商用電源の交流 100V 60Hz/50Hz をトランスで降圧してダイオードで整流して給電する場合 (図 2.1c, 図 2.1d) や, PWM 方式の場合 (図 2.1e) が挙げられます. ただしいずれも走行時には電圧極性が反転することはありません.

PWM 方式パワーパック

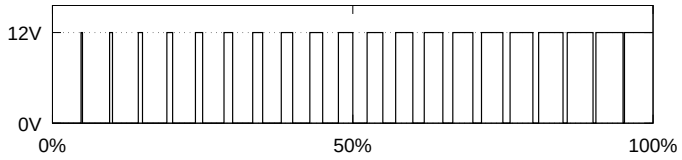
PWM 方式のパワーパックは 0 ボルトと 12 ボルトのオン・オフを高速に切り替えて, 1 周期中の 12 ボルトの期間の比率の大小を制御することでモータの回転速度の大小を変化させます. 1 周期中の 12 ボルトの期間の比率をデューティ比 (duty ratio) と呼び, 1 秒あたりの周期回数を PWM 周波数 (PWM frequency) と呼びます. 図 2.2 にデューティー比の違いによる波形の違いを示します.

PWM 方式では, モータに印加する電圧のデューティ比の大小を制御することで, 図 2.3 に示す通り印加される平均電圧を変化させて, モータの回転速度の大小を変化させます.

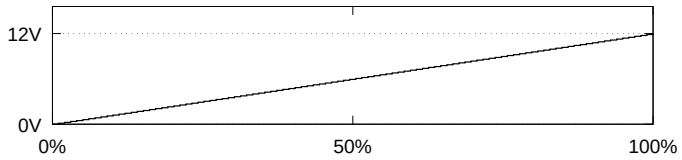
- デューティ比が 0% の場合
  常に 0 ボルトが印加されるので, モータはまったく回転しません.
- デューティ比が 100% の場合
  常に 12 ボルトが印加されるので, モータは最高速度で回転します.
- 一般にデューティ比が d% ($0 \leq d \leq 100$) の場合
  印加される平均電圧は $12d/100$ ボルトであり, この電圧に応じてモータの回転速度が決まります.

PWM 周波数は数 10Hz 程度から数 10kHz 程度の間の値をとるのが一般的です. 数 100Hz 以下の場合は人間の可聴域の周波数なので, モータの振動音が聞こえます. 20kHz 以上の場合は人間の可聴域外の周波数なので, モータの振動音は聞こえません. 平均電圧は PWM 周波数の高低には関係なくデューティ比だけで決まります. そのため PWM 周波数が高くてもデューティ比に従ってモータが回転すると思うかもしれませんが, 実際にはそのようにはならず, 低速でのトルクが弱くなり低速走行が思うようにできないことが起きます. PWM 周波数とモータの実際の回転の関係はモータのさまざまな特性に関係するために一概には言えないのですが, おおよそ以下の原理があります. PWM 周波数が高いと 1 秒あたりに多くの回数のパルスが与えられますが, 逆に 1 パルスあたりのエネルギーは弱くなります. このときにはモータが回転を始めるための静摩擦力を上回らないので, モータは回転しない結果となります. 例えばダルマ落としで遊ぶ際に, 木槌で強く叩く場合には一発で積木は動きますが, 羽毛で弱く叩く場合にはどんなに高頻度でも積木はまったく動かないのと似ています.

PWM 方式はアナログ制御方式に限らず, DCC 等のデジタル制御方式でもモータの制御の部分で使われています. PWM 方式ではオン・オフを高速にスイッチングを行うだけなので, 駆動回路のモータドライバの発熱を少なくできます. 多くの半導体メーカから小型のモータドライバ IC が各種開発されており, DCC デコーダの小



(a) パルス電圧の波形



(b) 平均電圧の値

図 2.3: PWM 波 (デューティー比を 0% から 100% に変化)

型化の実現に寄与しています.

常点灯機能

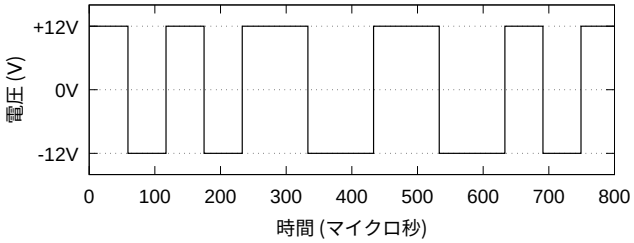
常点灯機能とは, 模型車両が停車しているときでもヘッドライトや室内灯を点灯させる仕組みのことです. 印加する電圧のパルス幅を細くすると, モータがその静摩擦のために回転は生じない一方, 応答の早い LED は点灯します. この原理により停車と点灯の両方を実現しています.

2.1.2　デジタル制御方式

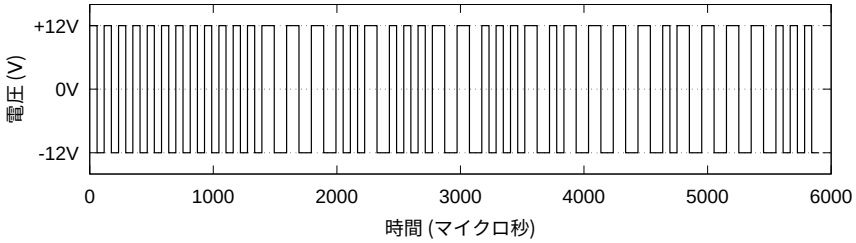
デジタル制御方式は, 模型車両の走行の方向と速度をデジタル信号による指令を与えて制御します. この方式の大きな特徴は, 同一の線路上にあっても模型車両ごとに異なる指令を与えて異なる動作を可能としている点にあります. 例えば同じ線路上で隣り合う 2 台の機関車のうちの 1 台だけを走らせることが可能です. アナログ制御方式でこれを行うには, 線路にギャップを入れて絶妙な位置に 2 台の機関車を停車するしかありませんでした. しかしデジタル制御方式ではギャップを入れることなく可能となります. 走行速度のほかにも, 走行中であるか否かに関わらずヘッドライト, テールライト, 室内灯などを個別に点灯・消灯が可能です.

デジタル制御方式の別の特徴は, 車輌にサウンド機能を搭載して様々なサウンド発生を操作できることも挙げられます. アナログ制御方式でもサウンド機能はありますが, 同一の線路上の 2 台の機関車から別々の音を出すことは困難でした. 一方, デジタル制御方式では, 三重連の蒸気機関車それぞれから別々のサウンドを同時に発生させたり, 寝台列車の電源車から発電機のサウンドを発生させたり, あるいは機関区のたくさんの機関車が賑やかにいろいろなサウンドを発生させることが, いとも簡単に実現できます.

本書執筆時でのデジタル制御方式の主流は, DCC 方式, Märklin Digital 方式, Selectrix 方式の 3 通りです. DCC 方式は仕様が公開されており, NMRA (National Model Railroad Association; 全米鉄道模型協会) ならびに NEM (European standards for model railroads; 欧州鉄道模型標準規格) にて規格として制定されており, 多くのメーカが製品を販売しています. Märklin Digital 方式はドイツの Märklin 社独自の規格で, 残念ながら仕様は公開されていない模様です. Märklin Digital 方式の基本的な考えは DCC 方式と類似しているようですが技術的な詳細が異なり, 直接的な互換性はないようです. ただしデコーダには DCC 方式と Märklin Digital 方式の両方に対応したものが ESU 社より販売されています. Selectrix 方式は当初は Trix 社の鉄道模型のデジタル制御方式として考案され, 現在ではその規格は公開されて一部は NEM 規格として制定されています.



(a) ビット 0 と 1 (長パルスは 0, 短パルスは 1)

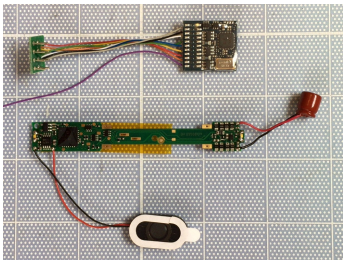


(b) ビット列 11111111111100011011100111010000100001111 に対する波形

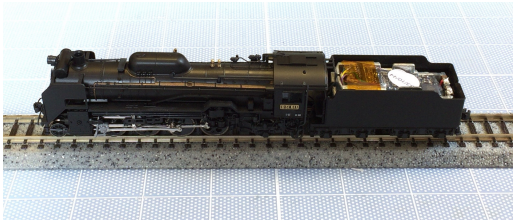
図 2.4: DCC 電圧波形の例



(a) DCC 制御機器の例.
左から, KATO DCS50K, Digitrax DCS200, 同 DT402

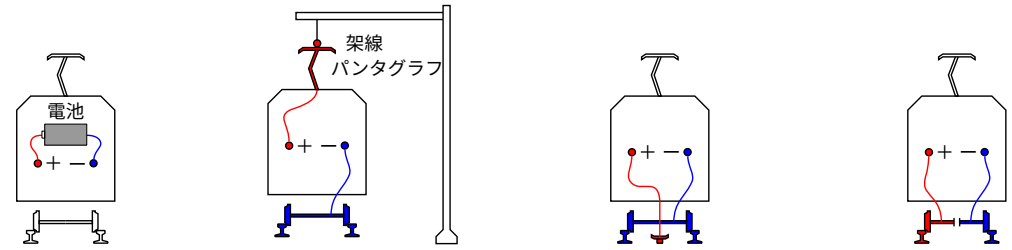


(b) DCC デコーダの例. 手前から, Digitrax SDN144A0, ESU LokPilotFX V3.0



(c) 模型車両への DCC デコーダ搭載の例.
(KATO N ゲージ D51 標準形 + ESU LokSound V4.0 サウンドデコーダ)

図 2.5: DCC 機器の例



(a) バッテリ式　**(b)** 架線式　**(c)** 3 線式　**(d)** 2 線式

図 2.6: 模型車両への給電方式

DCC 方式では基本的に線路を電力の供給線として使用すると同時に, 指令のデジタル信号の通信線としても使用します. 図 2.4 に DCC での電圧波形の例を示し, 図 2.5 に DCC システム機器の例を示します. 指令は 0 と 1 のビット列のデジタル信号で表現され, デジタル信号のビットの値に基づいて線路に印加する電圧波形を変調して, 線路上の各模型車両に指令を伝えます. 模型車両には DCC デコーダ (DCC decoder) と呼ばれている小さなマイコン装置を搭載しておき, DCC デコーダは受けた指令を元に模型車両の走行等を制御します. あらかじめ DCC デコーダそれぞれに DCC アドレスと呼ばれる固有の番号を付与しておきます. そして各指令には対象の DCC アドレスを含めて送信します. すると同じ線路上の複数の模型車両が存在していても, DCC デコーダは DCC アドレスの判別により自身への指令であるか否かを区別できます. これが模型車両ごとに独自の動作が可能となる仕組みです.

2.2　給電方式

模型車両に電力を供給する方式を大別すると以下の分類ができます.

模型車両への給電方式の分類

バッテリ式
架線式
3 線式
2 線式

バッテリ式は図 2.6a に示す通り, 外部から電力供給を受ずに模型車両に搭載した電池から受電する方式です. 乗用可能な大型の鉄道模型や安価な玩具でこの方式が採用されています. 架線式は図 2.6b に示す通り架線と線路を使用して, 模型車両は 2 箇所から受電する方式です. この方式は実際の鉄道での電化方式と同様な方式であることから, 導入例は少ないながらも現在でも使われています. 3 線式は図 2.6c に示す通り線路の左右の 2 本のレールに加えてセンターレールを配して, 模型車両は 3 箇所から受電する方式です. Märklin 社の HO スケール模型ではこの方式が採用されています. 2 線式よりも安定した集電が可能となることや, 線路の電気極性を気にすることなくリバース区間の設置が可能であることが, この方式の長所です.

2 線式は現在の主流で, 図 2.6d に示す通り 2 本のレールを使用して, 模型車両は 2 箇所から受電する方式です. この方式では走行に最低限必要な 2 本のレールを同時に給電にも使用するため, 設備は最低限で済むのが長所です. その一方, リバース区間の設置には線路の電気極性に注意が必要であり, 集電性能は 3 線方式よりも若干劣ることが, この方式の短所です.

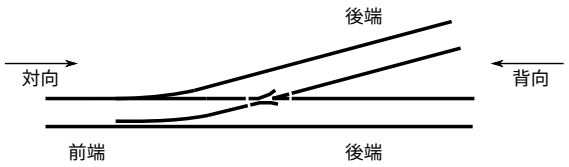
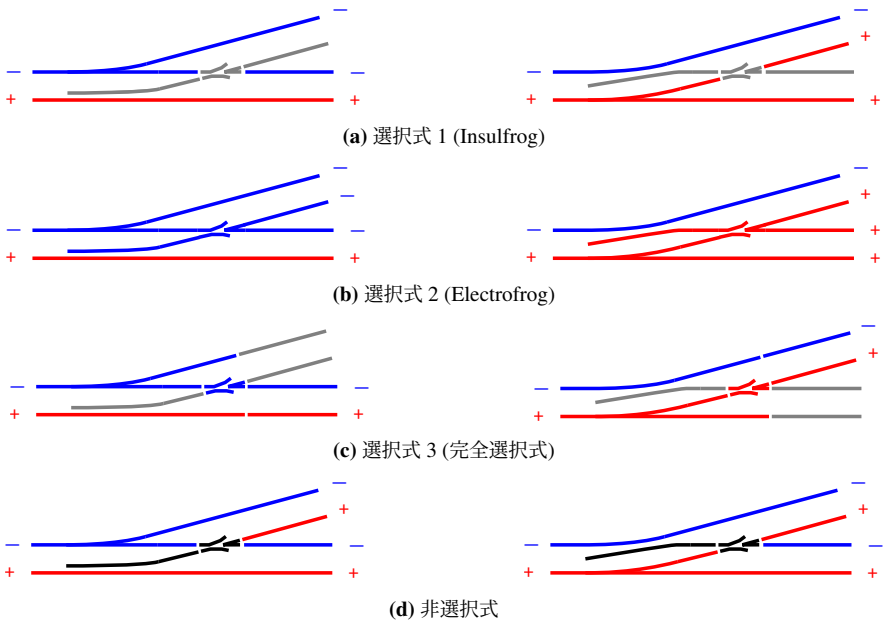


図 2.7: ポイントに関する用語



(a) 選択式 1 (Insulfrog)

(b) 選択式 2 (Electrofrog)

(c) 選択式 3 (完全選択式)

(d) 非選択式

図 2.8: 選択式・非選択式ポイントの電気的特性

2.3　ポイントの電気的特性

ポイント (分岐器) の電気極性に関する方式を大別すると以下の分類ができます. なお日本の鉄道の実車においては「分岐器 (turnout)」が正式な用語として日本工業規格 JIS E 1311:2002(鉄道–分岐類用語) で規定されています. 日本の鉄道模型においては「ポイント (point)」という用語が伝統的に使用されていますので, 本書ではポイントという用語を使用します.

ポイントの電気極性方式の分類

非選択式
選択式

ポイントの非選択式・選択式の分類は, ポイントの前端[*1] に給電されている電圧がそのままポイントの後端[*2] に給電されるか否かの違いによりなされます. ポイントの電気的特性の代表的なものを図 2.8 に示します. メーカや製品シリーズで様々なバリエーションがあるためすべてを詳細に網羅することは難しく, ここでは基本的な事項のみを説明します. 詳細は各製品の説明書を各自でご参照下さい.

非選択式ポイント

[*1] JIS E 1311:2002, 番号 730. 「分岐器の分岐する手前の基本レール端の位置。」図 2.7 参照のこと.
[*2] JIS E 1311:2002, 番号 731. 「分岐器の分岐した後の基準線及び分岐線の, 最遠のクロッシング端又はレール端の位置。」

非選択式ポイント とは, 開通方向に関わらずポイントの前端と両方の後端を導通する仕組みのポイントをいいます.

この方式はデジタル制御方式において重用されます. なぜなら, 室内灯を点灯し続けたり, あるいはサウンドを発し続けたりするには, 線路上の模型車両すべてに給電する必要があるからです. アナログ制御方式においてスプリングポイントとして使用する際には, 模型車両は非開通方向の背向[*3] からでも受電できてポイントへ進入できるため重用されます.

選択式ポイント

選択式ポイント とは, ポイントの前端と開通方向の後端を導通しますが, 非開通方向の後端とは導通しない仕組みのポイントをいいます. 図 2.8 に示す通り, 選択式ポイントはさらに細かく分類されます. 具体的には, 非開通方向の線路は同極, あるいは絶縁されているなどの違いがあります.

この方式はアナログ制御方式において重用されます. なぜなら非開通方向へは給電されないため, 非開通方向の線路上に模型車両の留置ができるからです. スプリングポイントでない限り, ポイントの非開通方向の背向にある模型車両を走らせないことにすれば, 進路の構成の制御と線路への給電の制御とを同時に行えますので, アナログ制御方式において合理的な方法と言えます. 一方, デジタル制御方式においては選択式ポイントは基本的には使われません. なぜなら非開通方向の線路上の模型車両へは制御信号が届かないため, DCC の最大の特徴である車両個別の制御ができなくなるためです. しかし多数の模型車両を留置している大レイアウトで, 電流消費量の削減やサウンドを停止させるために使うことも考えられます.

2.4　ポイントマシン

ポイントマシンの機構方式を大別すると以下の分類ができます.

ポイントマシンの機構方式の分類

手動方式
電動方式
　ソレノイド方式
　　2 線式
　　3 線式
　DC モータ方式
　サーボモータ方式

手動方式は, 人の手によって切り替えを操作する方式です. ポイントマシンの出費が不要なぶん安価で済むのは長所ですが, ポイントへ直接手を伸ばして操作する必要があり少々面倒なのが短所です. 現在では電動方式が主流です. 図 2.9 にポイントの例を示し, 図 2.10 に外付けのポイントマシンの例を示します.

電動方式は切り替え動作時間の緩急の観点に基づくと, 以下の分類ができます.

切り替え動作時間に基づく電動方式ポイントマシンの分類

瞬時アクション方式
　ソレノイド方式
　　2 線式

[*3] JIS E 1311:2002, 番号 760. 「分岐器の後端側から前端側への向き。」

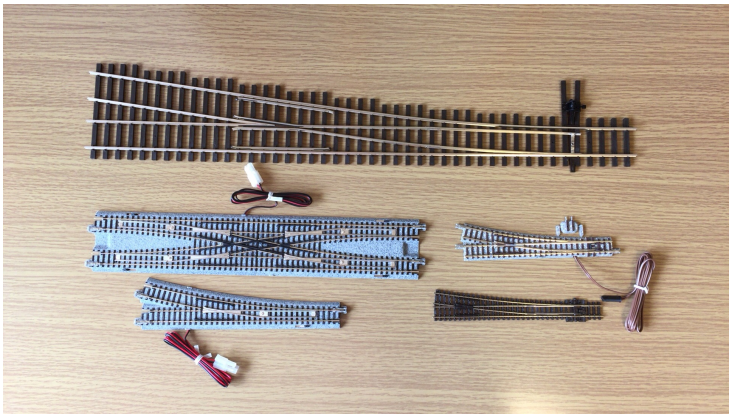
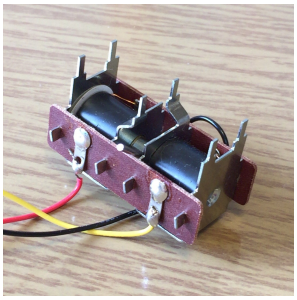
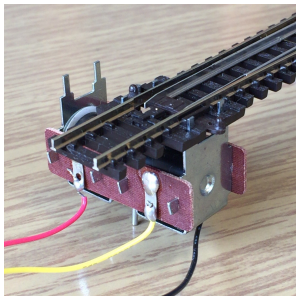


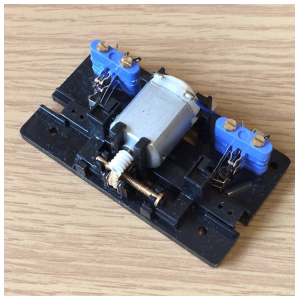
図 2.9: ポイントの例. (手前左から時計回りに, KATO N ゲージ ユニトラック 6 番右, 同複線両渡り, カツミ OJ ゲージ 8 番右, TOMIX N ゲージ ファイントラック N-PR-541-15(F), PECO SL-388)



(a) PECO PL10 (ソレノイド方式)



(b) PECO PL10 の取り付け



(c) Fulgrex (DC モータ方式)

図 2.10: 外付けポイントマシンの例

3 線式
スローアクション方式
DC モータ方式
サーボモータ方式

瞬時アクション方式は, 切り替え信号を受けるとトングレールを高速に動かして瞬時にポイントの切り替え動作を完了するものです. ソレノイド方式では瞬間的に電磁石で磁力を発生させて切り替え動作を行うため, この方式に分類されます. スローアクション方式は, 切り替え信号を受けるとトングレールをゆっくりと動かして切り替え動作を行うものです. DC モータ方式やサーボモータ方式ではモータの回転により時間をかけて切り替え動作を行うため, この方式に分類されます.

ソレノイド方式

ソレノイド方式は, 内蔵するソレノイド (電磁石) に電流を流すこと生じる磁力によりポイントの切り替えを行います.

2 線式ポイントマシンには 2 本の電線があり, 内部には 1 つの電磁石があります. 図 2.11a に内部の模式図を示します. ポイントの開通方向は, 電磁石の極性, すなわち電流の極性によって決まります. この機構上, ポイントマシンに流す電流は直流です. この方式を採用している例として KATO ユニトラックや TOMIX ファイントラックなどの電動ポイントが挙げられます.



(a) 2 線式　　**(b)** 3 線式

図 2.11: ソレノイド式ポイントマシン内部の模式図

3 線式ポイントマシンには 3 本の電線があり, 内部には 2 つの電磁石があります. 図 2.11b に内部の模式図を示します. 3 本の電線のうち 1 本は共通線で, 2 つの電磁石の両方に接続されています. 残りの 2 本は 2 つの電磁石それぞれに接続されています. ポイントの開通方向はどちらの電磁石に電流を流すかで決まります. この機構上, ポイントマシンに流す電流は直流でも交流でも構いません. この方式を採用している例として PECO PL-10 Turnour Motor が挙げられます.

これらソレノイド方式のポイントマシンを使用する際の重要な注意事項は, 電流を流す時間を数 100 ミリ秒程度に抑えなければならないということです. 内部の電磁石は電気回路的にはコイルですので, 電流のオン・オフ時にはコイルのインダクタンス成分により抵抗値は高くて流れる電流は少ないのですが, 電流を継続的に流し続けると抵抗値は低くなり, 大電流が流れて発熱・焼損に至ります.

この方式は機構が簡単で小型化しやすく, KATO ユニトラックや TOMIX ファイントラック等のように, 道床つき線路の内部に組み込まれて簡単に使用できる製品があるのが大きな長所です. 一方, 切り替え音が大きく, また一瞬で切り替わってしまう点が実感的ではないところが短所です.

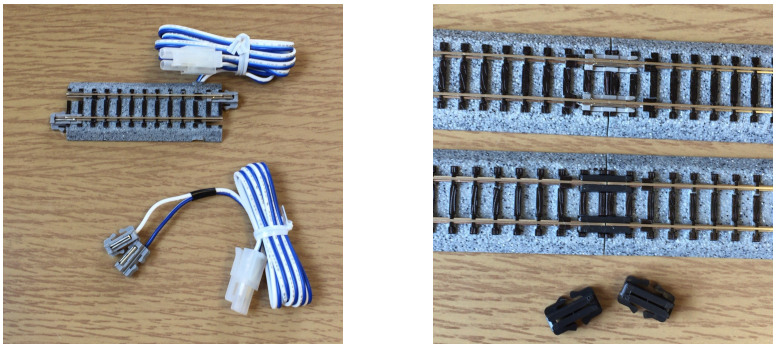
DC モータ方式

DC モータ方式のポイントマシンは DC モータを回転させてポイントの切り替えを行います. ポイントの開通方向は, DC モータに印加する電圧の極性で決まります. DC モータから歯車を通じ, 回転数を落としてポイントのトングレールをゆっくりと動かしますので, 非常に実感的な動作が長所です. 一方, 装置は大きめであり, 設置・調整作業が若干面倒である点が短所です. この方式を採用している例としては, Fulgrex ACC001 Slow Action Point Motor や Tortoise #6000 Slow Motion Switch Machine が挙げられます.

この方式のポイントマシンはいずれも, 電圧を常時かけつづけても大丈夫な構造になっています. 切り替えが完了すると, 自動的に DC モータへの給電を遮断するスイッチが内部に用意されているためです. そのためソレノイド方式のように焼損する危険性がないぶん, 安心して使えます. 上で挙げた製品例ではいずれもそのようになっていますが, 他の製品でも同じかどうかは分かりませんので各自で取扱説明書を熟読して下さい.

サーボモータ方式

サーボモータ方式のポイントマシンは, ラジコンの模型飛行機や模型自動車にも使われているホビー用のサーボモータを使用してポイントの切り替えを行います. サーボモータへの入力はパルス波であり, そのデューティ比によりモータ軸の回転角が指定されます. サーボモータは DC モータとは異なり, モータ軸の回転角を細やかに指定できる点が特徴です. この性質から, 鉄道模型ではポイントマシンの他に, 腕木式信号機の腕木や踏切遮断機の遮断竿の駆動にも使用されます. この方式を採用している例として PECO PLS-120 SmartSwitch が挙げられます.



(a) フィーダー線路 (上) とターミナルユニジョイナー (下)　**(b)** 絶縁ジョイナー (下) によるギャップ (中) およびギャップなしの接続 (上)

図 2.12: フィーダとギャップの例　(KATO N ゲージ ユニトラック)

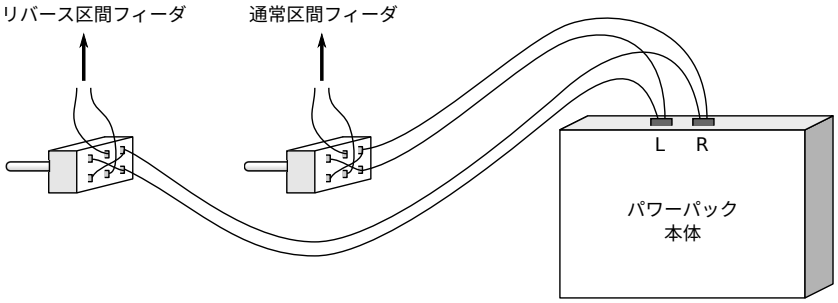


図 2.13: 通常区間用とリバース区間用の逆転スイッチ

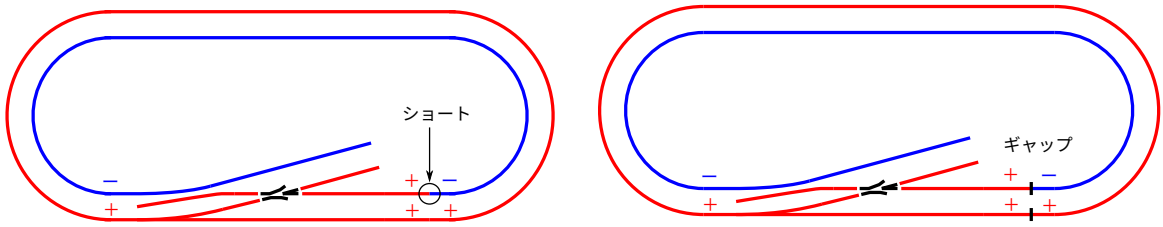
2.5　フィーダとギャップ

フィーダ (feeder wire) とは, 給電のためにパワーパックと線路を結ぶ電線のことです. ギャップ (gap) とは, 電気的な絶縁を目的とする線路に入れる切れ目のことです. 図 2.12 にフィーダとギャップの例を示します.

入門用のパワーパックにはフィーダ出力系統は 1 つだけの場合が多いですが, 中級以上のパワーパックでは図 2.13 に示すようにフィーダ用の出力系統が 2 つあり, 一方は通常区間用の出力で他方はリバース区間用の出力としての使用を想定しています. それぞれの逆転スイッチの操作により線路への給電の極性を独立して切り替えることができます.

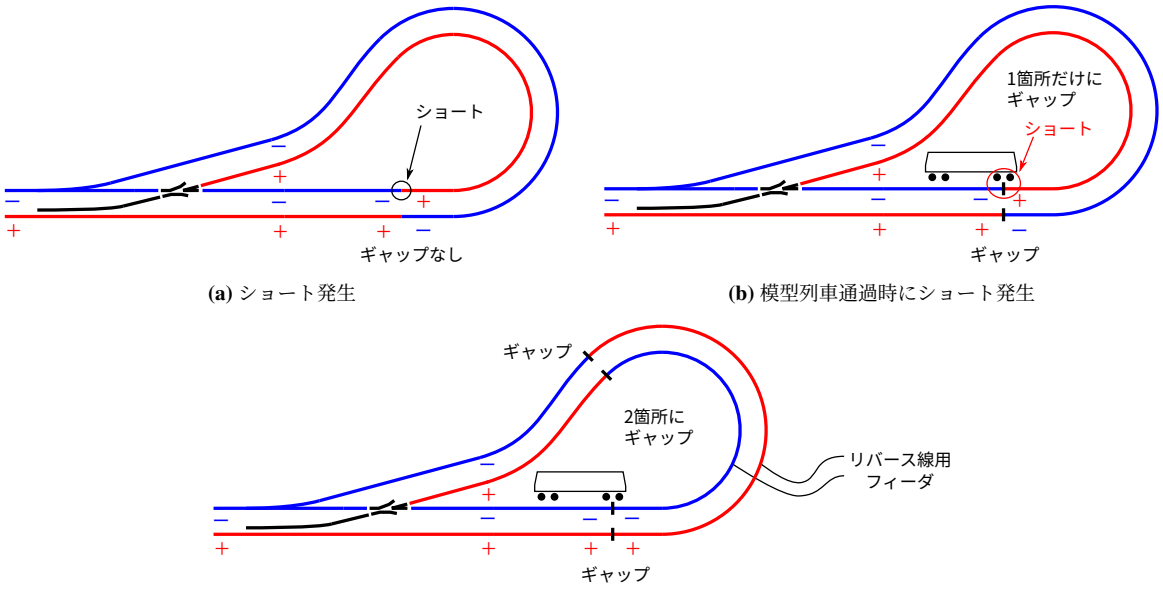
選択式ポイントを含んだレイアウトやリバース区間のあるレイアウトでは, 各所に給電がされるように必要に応じてフィーダを付け, ショートが起きないように必要に応じてギャップを入れます. 選択式ポイントにおけるギャップの必要性を図 2.14 を例にして説明します. 選択式ポイントの中でも図 2.8b で示したものでは, ポイントの非開通方向では左右のレールが同極性になります. もしポイントの後端が前端に接続されていれば, 開通方向によりショートが発生してしまいます. そこでギャップを入れてショートの解決を図ります.

リバース区間におけるギャップの必要性を図 2.15 を例にして説明します. 非選択式のポイントの場合, ギャップを入れなければ図 2.15a に示すようにショートが発生しますから, ギャップを入れてショートの解決を図ります. しかしギャップが一ヶ所だけでは図 2.15b に示すように, 模型列車がギャップを通過する時にショートが発生します. そこでギャップを図 2.15c に示すように二ヶ所に入れたリバース区間を作ります. このときリバース区間の長さは模型列車の編成全体が収まるようにします. そうしなければ編成の先頭と末尾で電源極性が反対となるからです. そして走行中にはショートが起きないように逆転スイッチを操作を行います. 以上は非選



(a) ショート発生　　**(b)** ギャップによるショート回避

図 2.14: 選択式ポイントにおけるギャップの必要性



(a) ショート発生

(b) 模型列車通過時にショート発生

(c) 2 つのギャップによるリバース区間の構成

図 2.15: リバース区間におけるギャップの必要性

択式ポイントの場合で説明しましたが, 選択式ポイントで図 2.8b に示したものも同様です.

DCC システムにおいても給電が 2 線式の場合にはギャップを切る必要があることは図 2.15 を見ると分かると思います. また図 2.15c より, リバース区間がある場合には線路への給電極性の逆転の必要性が分かります. 給電極性の逆転を手動で行うのは面倒ですので, これを自動化するリバーサ (reverser) と呼ばれている装置が使用されます.

TOMIX ファイントラックは完全選択式の選択式ポイントですので, リバース区間がない限りはギャップは不要です. KATO ユニトラックの選択式ポイントは完全選択式ではありませんからギャップは必要です. なお KATO ユニトラックのポイントには, 選択式と非選択式を設定により変更できるものがありますので適宜使い分けると良いでしょう.

ギャップには 2 種類あり, 2 本のレールのうちの片方だけに入れる片ギャップと, 2 本のレールの両方に入れる両ギャップがあります. フィーダにも 2 種類あり, 2 本のレールのうちの片方だけに接続する片フィーダと, 2 本のレールの両方に接続する両フィーダがあります. 片ギャップと片フィーダを多用すれば配線の手間は半減しますが, どの場合に可能であるかは選択式ポイントの電気的性質にも依存します. そこで以降ではアナログ 2 線式の場合において動作が一番確実な両ギャップ・両フィーダの場合について説明します.

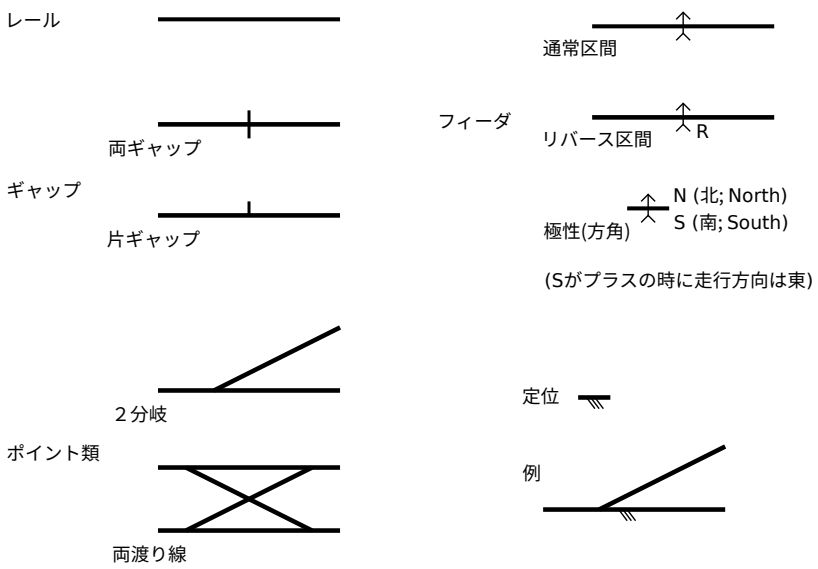
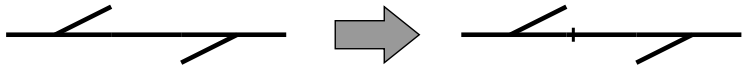
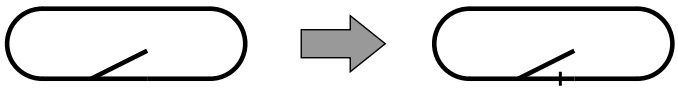


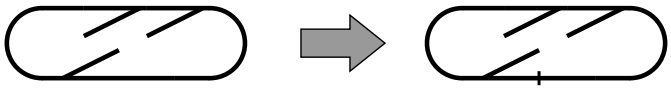
図 2.16: レイアウトプランで使用する記号



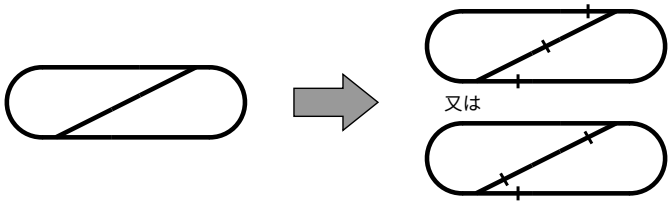
(a) 2 つのポイントの後端が互いに接続



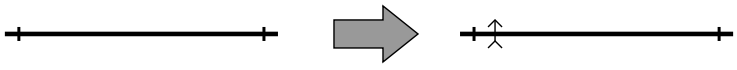
(b) ポイントの前端と後端が接続



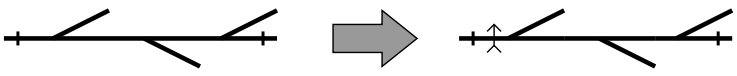
(c) ポイントの前端と後端が複数ポイントを通じて接続



(d) リバース区間



(e) ポイントのない区間へのフィーダ



(f) ポイントのある区間へのフィーダ

図 2.17: ギャップとフィーダの付け方

フィーダとギャップが満たすべき条件を直感的に述べると以下の通りです.

- 線路のどの場所へも給電が可能でなければなりません. 給電が可能であるとは, フィーダによる直接的な通電と, ポイントの前端から後端への間接的な通電の二通りがあります.
- 模型列車を線路上に置いていないとき, ポイントや逆転スイッチをどのように切り替えてもショートが起きないようにしなくてはなりません.
- 模型列車を線路上に置いたとき, どの位置に模型列車を置いてもショートが起きないようにしなくてはなりません.

上記の条件に違反しないようなフィーダとギャップの場所の具体的な決定方法を以下に説明します. ただし運転の都合で更にギャップを入れる場合がありますが本書では省略します. なおレイアウトプラン中で使用する記号の説明を図 2.16 に示します. また図 2.17 に例を示しますが, ポイントの種類によりギャップを省略できる場合もありますが詳しくは省略します.

- 図 2.17a: 2 つのポイントの後端が互いに接続されている場合はギャップを入れます. そうしなければポイントの切り替えによりショートが起き得るためです.
- 図 2.17b: ポイントの前端と後端が接続されている場合はギャップを入れます. そうしなければポイントの切り替えによりショートが起き得るためです.
- 図 2.17c: あるポイントの前端から出発して, 1 つ以上の異なるポイントの後端・前端と辿り続けると元のポイントの後端に接続されている場合はギャップを入れます. これは上の場合の一般的な場合です. そうしなければポイントの切り替えによりショートが起き得るためです.
- 図 2.17d: リバース区間がある場合は, 模型列車の編成長を考慮して, ショートが起きないようにギャップを入れてリバース区間を作ります. そうしなければ模型列車の先頭と最後尾で逆の極性が印加されてショートが起き得るためです.
- 図 2.17e: ポイントが全くない区間にはどこか 1 箇所にフィーダを付けます. そうしなければその区間はまったく給電されないためです. ただしその区間がリバース区間の場合は, リバース区間用のフィーダ出力系統に接続します. そうしなければリバース区間と通常区間の境界を模型列車が跨いだときにショートが起き得るためです. なおレールが長い場合に生じる電圧降下の対策として, 同じ区間の複数箇所にフィーダを付ける場合もあります.
- 図 2.17f: ポイントがある区間では以下の要領でフィーダを付けます.
  - ポイントの前端でどのポイントの後端にはなっていない場合には前端にフィーダを付けます. そうしなければ前端のみならず後端も給電されないからです.
  - ポイントの後端にはフィーダは付けません. もしそうしてしまうとポイントの切り替えによりショートが起き得るためです.

課題

1. 鉄道模型の黎明期から現在までどのような動力源が使用されてきたかを調査して下さい.
2. 鉄道模型の動力源として電気を使用することの利点を他の動力源と比較を行いがなら説明して下さい.
3. アナログ制御方式とデジタル制御方式それぞれの長所と短所を理由とともに述べてください.
4. アナログ DC 制御方式における純直流パワーパックと PWM パワーパックそれぞれの長所と短所を理由とともに述べてください.
5. NMRA 規格においてアナログ DC 制御方式のパワーパックが満たすべき電気的な条件は何かを調査し

て述べて下さい.

6. NMRA 規格においてアナログ DC 制御方式の模型車両が満たすべき電気的な条件は何かを調査して述べて下さい.
7. アナログ制御方式において選択式ポイントが重用される理由を述べて下さい.
8. アナログ制御方式において非選択式ポイントが有用となるのはどのような場面か述べて下さい.
9. デジタル制御方式において非選択式ポイントが重用される理由を述べて下さい.

第 3 章

電子部品の基礎知識

本章では本書で使用する主要な電子部品を紹介します. ここでは以下の内容を学び, 後の章で紹介する機器で使用する電子部品の基礎知識の会得を目的とします. すでに電子工作に慣れ親しんでいる方は, この章を読み飛ばして構いません.

- 本書で使用する電子部品の機能
- 外見やピン配置
- 安全上の観点での注意事項
- 本書の回路図での記号

詳細な仕様については各自でデータシートを参照して下さい. ただしデータシートは専門的な内容なので, 初心者には読みづらいと思います. 近年では電子工作で使用する部品を紹介する書籍がいくつか出版されていますので, そちらを参考にするのもよいでしょう.

特に注意が必要なのは, 各部品には守らなければ部品が破損してしまう絶対最大定格と呼ばれている使用条件です. これを守らないと部品の破損や事故につながりますから, 絶対最大定格は必ず守って下さい. 本書で紹介する装置ではいずれも絶対最大定格の範囲内で部品を使用しています. しかし使用する部品を変更するしないに関わらず, データシートを読むなど安全については各自で必ず確認をしたうえで製作に取り掛かるようにして下さい.

3.1 受動部品

抵抗器

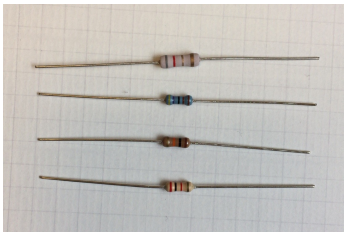
抵抗器 (registor) は, 電気抵抗を得るための部品です. 電流の制限を行ったり, 所定の電圧を得るなどの目的に使用します. 抵抗器の例と回路図での記号を図 3.1 に示します.

抵抗器の重要なパラメータは以下の 2 つです.

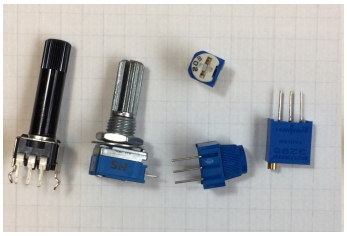
- 抵抗値: 抵抗器が持つ電気抵抗の値です. 電気抵抗の単位はオームで, 記号は Ω を使用します. 記号 kΩ はキロオーム, すなわち 1000Ω を表します.
- 定格電力: 壊れることなく抵抗器で消費されうる電力の最大値を表したものです. 定格電力の単位はワットで, 記号は W を使用します.

抵抗器は固定抵抗器 と可変抵抗器 の 2 種類に大別されます. 固定抵抗器はあらかじめ抵抗値が決められていて変更はできません. 可変抵抗器は指やドライバなどで回転あるいはスライドを行い, 抵抗値の変更ができます.

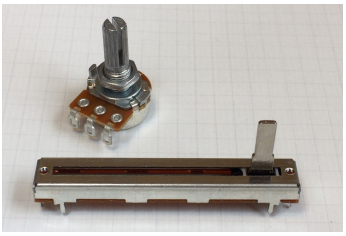
小型の固定抵抗器では, 抵抗器にカラーコードを印刷して抵抗値を表示します. 主に炭素皮膜抵抗と金属皮膜抵抗の 2 種類があり, 前者の誤差は大きく例えば 5% 程度で後者の誤差は小さく例えば 1% 程度です. 通常の電子工作では正確な抵抗値は不要ですから, 炭素皮膜抵抗を使用することがほとんどです. 炭素皮膜抵抗では, 図



(a) 外観: 固定抵抗器



(b) 外観: 可変抵抗器 (基板取付型)



(c) 外観: 可変抵抗器 (パネル取付型)

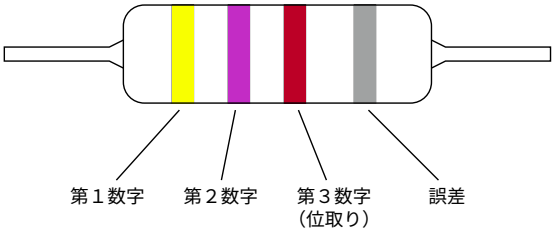


(d) 回路図記号: 固定抵抗器 (10kΩ)



(e) 回路図記号: 可変抵抗器 (10kΩ)

図 3.1: 抵抗器



例1: 470Ω = 47×10^1
　第1数字＝4（黄）　第2数字＝7（紫）　第3数字＝1（茶）（位取り＝10^1）
例2: 1kΩ = 10×10^2
　第1数字＝1（茶）　第2数字＝0（黒）　第3数字＝2（赤）（位取り＝10^2）
例3: 10kΩ = 10×10^3
　第1数字＝1（茶）　第2数字＝0（黒）　第3数字＝3（橙）（位取り＝10^3）

図 3.2: カラーコードによる抵抗値の表示

3.2 に示すように, 3 つの色で抵抗値を表し, もうひとつの色で誤差を表します. 色と数の対応は表 3.1 に示す通りです.

可変抵抗器は, 頻繁に人が操作することを目的とした普段「ボリューム」と呼んでいるものと, 頻繁な操作は目的としない半固定的なものに分類されます. 半固定可変抵抗の中でも特に精密な抵抗値の設定が可能なものは多回転式の構造をしています. 可変抵抗器の軸の回転角度と抵抗値の対応関係の違いにより, 図 3.3 に示す通り A カーブ, B カーブ, C カーブの 3 種があります. B カーブは回転角度と抵抗値が線形の関係にあり, 各種の調整の際によく使用されます. A カーブは音声のボリュームの調整などに使用します.

多くの可変抵抗器では図 3.4 に示す通り 3 つの端子があります. ひとつは共通端子で, この共通端子と他の端子の間で抵抗値が変化します. 共通端子を 2 番端子とし, 他の端子を 1 番および 3 番端子とします. 例えば 10kΩ の可変抵抗器では, 1 番端子と 2 番端子の間の抵抗値は 0Ω から 10kΩ の間で変化し, それに合わせて 3 番端子と 2 番端子の間の抵抗値はその反対の 10kΩ から 0Ω で変化します. そして 1 番端子と 3 番端子の間の抵抗値は常に 10kΩ となります.

コンデサ

コンデンサ (capacitor) とは電荷を蓄えるはたらきを持つ部品です. 電圧の変動を吸収したり, 直流は流さずに交流を流すなどの目的で使用します. コンデンサは構造によって様々なものがありますが, 本書で使用するのはセラミックコンデンサ と電解コンデンサ です. コンデンサの例と回路図での記号を図 3.5 と図 3.6 に示します.

表 3.1: カラーコード

色	数	位取り	誤差
黒	0	10^0	±1%
茶	1	10^1	±2%
赤	2	10^2	
橙	3	10^3	
黄	4	10^4	
緑	5	10^5	±0.5%
青	6	10^6	±0.25%
紫	7	10^7	±0.10%
灰	8	10^8	±0.05%
白	9	10^9	±1%
金		10^{-1}	±5%
銀		10^{-2}	±10%

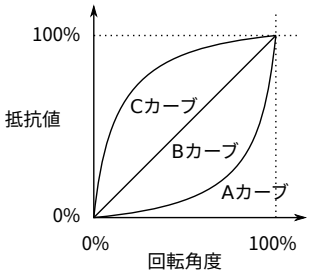


図 3.3: 可変抵抗器のカーブ三種

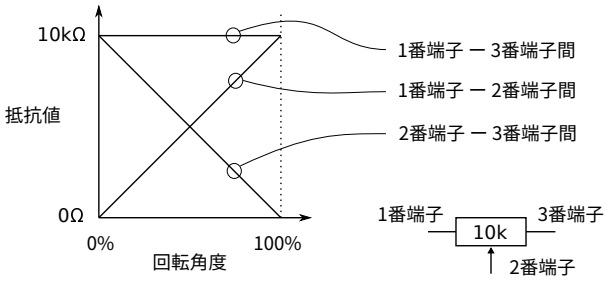
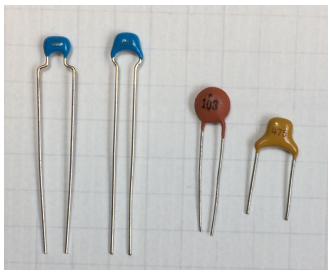


図 3.4: 可変抵抗器 10kΩ の端子間の抵抗値

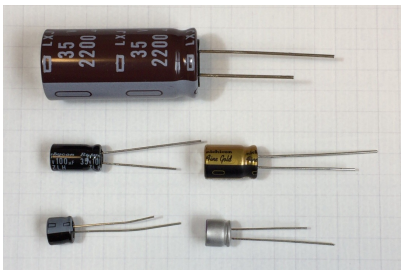


(a) 外観



(b) 回路図記号: 0.1μF

図 3.5: セラミックコンデンサ



(a) 外観



(b) 回路図記号: 100μF

図 3.6: 電解コンデンサ

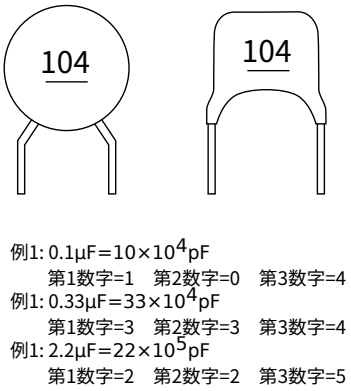


図 3.7: セラミックコンデンサの容量の表示

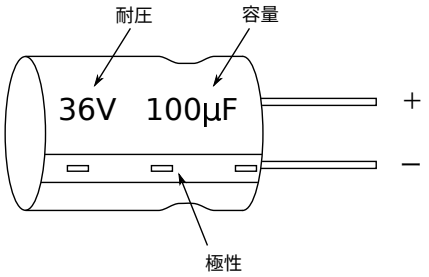


図 3.8: 電解コンデンサの容量と極性の表示

コンデンサの重要なパラメータは以下の 2 つです.

- 容量: コンデンサの静電容量で, 蓄えられる電荷量を表します. 静電容量の単位はファラッドで, 記号は F を使用します. 記号 μF はマイクロファラッド, すなわち 10^{-6}F を表します.
- 耐圧: 壊れることなく印加できる電圧の最大値を表します. 耐圧の単位はボルトで, 記号は V を使用します.

通常の電解コンデンサには極性があり, プラス端子はプラスに, マイナス端子はマイナスになるように接続しなければなりません. もし極性を間違えると最悪の場合には電解コンデンサが爆発して, 内部の電解液が周囲に飛び散り大変危険です. 現在の多くの製品では爆発が起きにくい対策が施されてはいても安全が保証されているわけではありませんから, 爆発の可能性を常に心に留めて極性を決して間違えないようにして下さい. なお電解コンデンサの中には極性のない無極性電解コンデンサと呼ばれるものもあり, オーディオ回路などで使用されます.

電解コンデンサの寿命は長くないという点には十分な注意が必要です. ドライアップ現象と呼ばれているもので, 内部の電解液が少しずつ蒸発して性能が劣化してゆきます. 劣化の進行は気温が高いほど急速に進むことが知られています. 電解コンデンサの仕様には耐久性の項目があり, 例えば 1000 時間 105°C などとデータシートに記載してあります. これは周囲温度 105°C において, 一定範囲内の性能を保持できるのは 1000 時間までということです. 電解コンデンサは信頼性のあるメーカの長寿命の品を購入することをおすすめします.

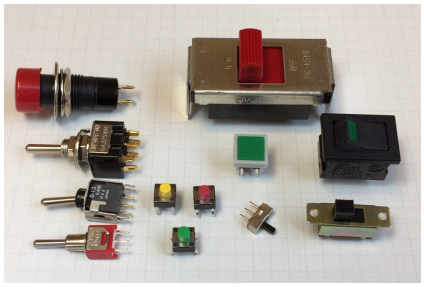
セラミックコンデンサの容量は図 3.7 に示す通り, 3 つの数字を用いて表示がされています. セラミックコンデンサには極性がないため, 極性の表示はありません. 電解コンデンサの容量は図 3.8 に示す通りに表示されています. 電解コンデンサには極性があるため, マイナス側の端子を示す表示もされています.

スイッチ

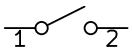
スイッチ (switch) とは電路の開閉を行う部品で, 電源の入切やなどの目的で使用します. スイッチの例と回路図での記号を図 3.9 に示します.

スイッチの重要なパラメータは以下の 2 つです.

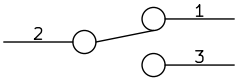
- 定格電流: 壊れることなく流すことができる電流の最大値を表します. 定格電流の単位はアンペアで, 記号は A を使用します. 記号 mA はミリアンペア, すなわち 0.001A を表します.
- 定格電圧: 壊れることなく印加できる電圧の最大値を表します. 定格電圧の単位はボルトで, 記号は V を使用します.



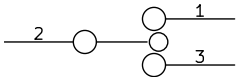
(a) 外観



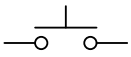
(b) 回路図記号:
1 回路 1 接点型



(c) 回路図記号:
1 回路 2 接点 On-On 型



(d) 回路図記号:
1 回路 2 接点 On-Off-On 型



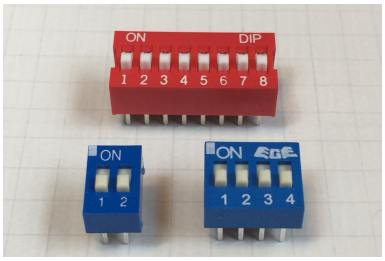
(e) 回路図記号: 押しボタン型

図 3.9: スイッチ

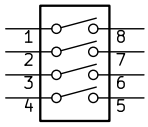
スイッチは多種多様なものがあり, いくつかを挙げると以下の通りです.

- トグルスイッチ
  レバーを前後に倒すことでオン・オフを切り替えます. トグルスイッチには, 接点数や回路数によりいくつかのバリエーションがあります.
  - 1 回路 1 接点 — SPST (Single-Pole Single-Throw; 単極単投)
    2 つの端子あり, ひとつの部品に 1 つのスイッチ (回路) が内蔵されています. レバー操作により 2 つの端子の間でオンとオフが切り替えられます.
  - 1 回路 2 接点 On-On 型 — SPDT (Single-Pole Double-Throw; 単極双投)
    3 つの端子があり, ひとつの部品に 1 つのスイッチ (回路) が内蔵されています. レバーを倒す方向の前後によって, 2 つの接点のうちのどちらか一方と共通端子がオンとなります. このため 2 ポジション型とも呼ばれます. On-On 型とは, 2 つの端子のうち必ずどちらか一方とオンになる性質を言います.
  - 1 回路 2 接点 On-Off-On 型 — SPDT センターオフ
    1 回路 2 接点 On-On 型と同様ですが, On-Off-On 型にはレバーに中点があり, 前でも後ろでもなく, 中央にレバーを位置できる点が違いです. このため 3 ポジション型とも呼ばれます. レバーの倒す方向は前または後のどちらかで, 2 つの接点のうちのどちらか一方のみがオンになります. 中点にレバーが位置しているときは, 2 つの接点のどちらともオフになります.
  - 2 回路 2 接点 On-On 型 — DPDT (Double-Pole Double-Throw; 双極双投)
    6 つの端子があり, ひとつの部品に 2 つのスイッチ (回路) が内蔵されています. 3 つの端子で 1 つの On-On 型スイッチをなし, 2 つのスイッチがその間でオンとオフが切り替えられます. アナログ式のパワーパックの逆転スイッチとして, この型のスイッチがよく用いられています.
  - 2 回路 2 接点 On-Off-On 型 — DPDT センターオフ
    2 回路 2 接点 On-On 型に中点を持たせたものです.

  レバー位置とオン・オフの関係の観点からは, 2 通りの分類ができます.
  - オルタネイト型
    レバーを倒すと, 手を離してもレバーはそのままで, 倒れたままの位置を保ちます.
  - モーメンタリ型
    レバーを倒している間はオンになりますが, レバーを離すとレバーは元の位置に戻りオフになります.



(a) 外観



(b) 回路図記号: 4 回路

図 3.10: DIP スイッチ

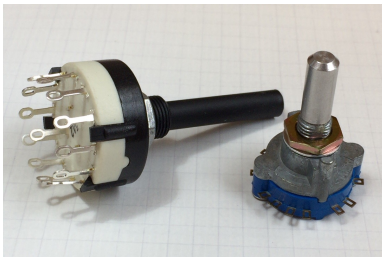
- 押しボタンスイッチ
  ボタンを押すことでオン・オフを切り替えます. 非常ベルのスイッチとして使用されているのをよく目にします. ボタン操作とオン・オフの観点からは, 2 通りの分類ができます.
  - オルタネイト型
    ボタンを押すたびにオンとオフが交互に入れ替わります. つまり, ボタンを押すとオンとなり, 再びボタンを押すと今度はオフとなります. テレビ本体の電源スイッチなどで使用されています.
  - モーメンタリ型
    ボタンを押している間はオンになりますが, 離すとオフに戻ります. 玄関のチャイムのボタンなどで使用されています.
- スライドスイッチ
  つまみを前後にスライドさせてオン・オフを切り替えます. ワイヤレスマイクのスイッチで使用されているのをよく目にします.
- DIP スイッチ
  プリント基板上に取り付けられる小型のスイッチです. 機器の設定の目的で使用されているのをよく目にします. 頻繁な操作の用途では使用しません. ひとつの部品に複数のスイッチが搭載されているのが普通で, スイッチ数が 2 個から 8 個までのものがよく使用されます. DIP スイッチの例と回路図での記号を図 3.10 に示します.
- ロータリスイッチ
  つまみを回転するとその角度に応じた端子と共通端子がオンになります. 電車ゲームのワンハンドルマスコンのレバーなどで使われています. 端子の数は 4 個, 6 個, あるいは 12 個などであり, ひとつの部品に内蔵されている回路数も 1 つあるいは複数の場合があります. 1 回路 12 接点あるいは 2 回路 6 接点などのものがよく使用されます. ロータリースイッチの例と回路図での記号を図 3.11 に示します.

リードスイッチ

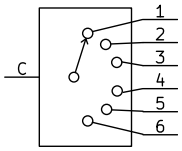
リードスイッチ (reed switch) とは外部からの磁力を受けてオンとなる機械的なスイッチ部品です. 鉄道模型において車両検出のために昔からよく使われています. またドアの開閉を検出する簡単なセンサとして使用されているのをよく目にします. リードスイッチの外観の例とピン配置を図 3.12 に示します.

リレー

リレー (relay) とは外部から制御される電圧でスイッチのオン・オフを制御する部品です. リレー内部には電磁石があり, 電流を流すことで発生する磁力によりスイッチを駆動します. リレーの例と回路図での記号を図

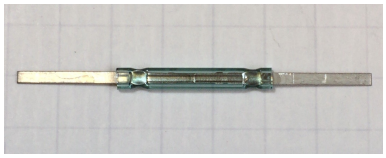


(a) 外観



(b) 回路図記号: 6 接点

図 3.11: ロータリースイッチ

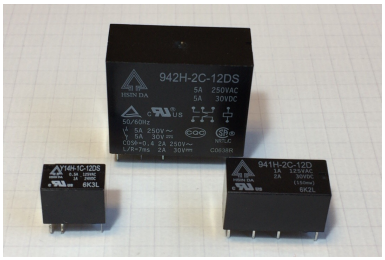


(a) 外観

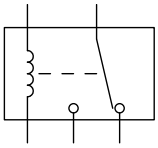


(b) 回路図記号

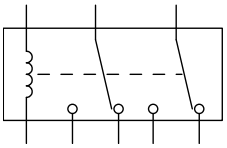
図 3.12: リードスイッチ



(a) 外観



(b) 回路図記号: 1 回路 C 接点



(c) 回路図記号: 2 回路 C 接点

図 3.13: リレー

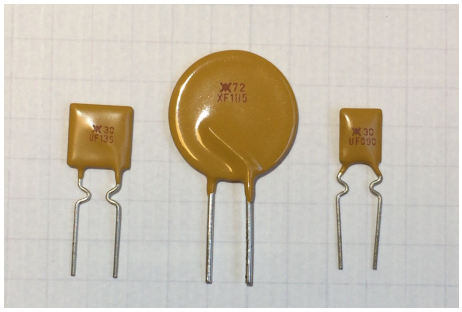
3.13 に示します.

リレーの重要なパラメータは以下の 3 つです.

- コイル電圧: コイルに印加すべき電圧値で, リレー内のスイッチのオン・オフの制御はこの電圧で行います.
- 接点容量: 壊れることなくスイッチに流すことができる電流の最大値です.
- 回路構成: リレー内部のスイッチの回路数と接点構成です.

リレーの接点構成には以下の 3 通りがあります.

- A 接点 (normaly open; NO 接点)
  端子数は 2 つで, スイッチの 1 接点型と同様です. コイルに電流が流れていないときには接点はオフで, コイルに電流が流れると接点がオンとなります.
- B 接点 (normally closed; NC 接点)
  端子数は 2 つで, スイッチの 1 接点型と同様です. コイルに電流が流れていないときには接点はオンで,



(a) 外観



(b) 回路図記号

図 3.14: ポリスイッチ

コイルに電流が流れると接点がオフとなります.

- C 接点 (トランスファー接点)
  A 接点と B 接点の両方を持ちます. 端子数は 3 つで, スイッチの 2 接点型と同様です. コイルに電流が流れていないときには A 接点がオフ, B 接点がオンで, コイルに電流が流れると A 接点がオン, B 接点がオフとなります.

例えば 2 回路 C 接点のリレーは On-On 型のスイッチが 2 つ入っているとみなせますので, 電気的な制御が可能なパワーパックの逆転スイッチとして使用できます.

上記で説明したリレーは電磁継電器と呼ばれるもので, 電磁石を用いて機械的な駆動をするリレーです. 半導体で実現したリレーであるソリッドステートリレーは機械的な機構を持たず, 故障が非常に少ないという長所があります.

ポリスイッチ

ポリスイッチ (PolySwicth) あるいはリセッタブルヒューズ (Resettable Fuse; PTC) とは, 回路に過電流が流れたときに電流を遮断するための部品です. ポリスイッチの例と回路図での記号を図 3.14 に示します.

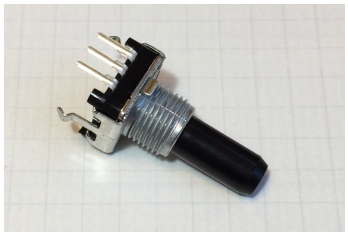
ポリスイッチの重要なパラメータは次のものです.

- 定格電流: 遮断をする電流値を表します.

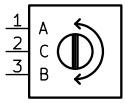
ポリスイッチのしくみは, 電流の増大によりポリスイッチ内の導電性ポリマーが加熱され, 一定温度を越えると急激に抵抗値が増大して電流を遮断するというものです. つまり抵抗値の増大により回路に流れる電流を抑制して過電流から回路を守ります. ポリスイッチの長所は, ポリスイッチが冷えると自動的に導通が復帰して何度でも利用できる点にあります. しかし定格電流を越えるとすぐに遮断されるわけでなく, 少し時間がかかる場合があるのがポリスイッチの短所です. なぜなら遮断されるに十分な熱をポリスイッチが持つまでに時間がかかることがあるためです.

ロータリエンコーダ

ロータリエンコーダ (rotary encoder) とは軸の回転を検出する部品です. 取り付けたつまみを指で回転させて, 値の入力・設定を行うのによく使用されています. 内部には 2 つのスイッチがあり, 軸の回転によって 2 つのスイッチのオンとオフが交互に入れ替わります. また 2 つのスイッチのオンとオフの相互の関係から軸の回転方向を検出できます. ロータリエンコーダの例と回路図での記号を図 3.1 に示します.

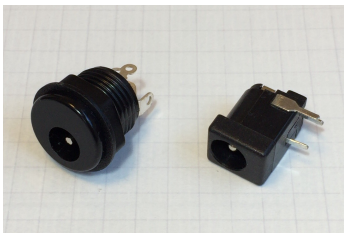


(a) 外観

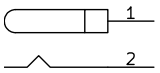


(b) 回路図記号

図 3.15: ロータリエンコーダ

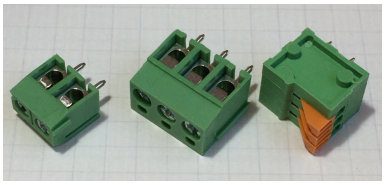


(a) 外観



(b) 回路図記号

図 3.16: DC ジャック



(a) 外観



(b) 回路図記号: 2 ピン



(c) 回路図記号: 3 ピン

図 3.17: ターミナルブロック

3.2　コネクタ部品

DC ジャック

DC ジャック (DC jack) とは AC アダプタなどから直流電流の供給受ける際に使用するコネクタ部品です. AC アダプタ側には DC プラグを, 機器側には DC ジャックを使用します. DC ジャックの例と回路図での記号を図 3.1 に示します.

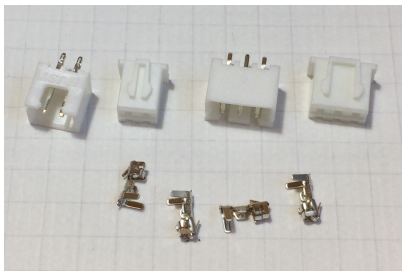
DC プラグと DC ジャックには 2 つの接点があります. 多くの場合はセンタープラス, すなわち中央の端子がプラスです. しかし AC アダプタによっては違う可能性もありますから, AC アダプタの記載を必ず確認して下さい. 接点の極性のほかに, 太さなどの物理形状が DC プラグと DC ジャックで一致しないと差し込めませんので注意して下さい.

ターミナルブロック

ターミナルブロック (terminal block) とは基板と外部とを電気的に接続するためのコネクタ部品です. 電線の接続は, 電線 1 本ごとにドライバーを使って行います. ターミナルブロックの例と回路図での記号を図 3.17 に示します.

コネクタ

コネクタ (connector) とは複数の電線をまとめて電気的に接続するための部品です. 複数の電線をまとめて接続ができるようになっていますので, 一度に多数の電線をしかも間違えないように接続できるので便利です. 製作した装置の基板をケースに入れて使用するときや, ケースに取り付けたボリュームや LED と基板を接続するのに使用すると便利です. また複数の基板の間で信号のやりとりを行う際でも使用します. なおコネクタの接



(a) 外観: JST (日本圧着端子製造) XH コネクタ



(b) 回路図記号:
ハウジング (2 極)



(c) 回路図記号:
コネクタ (2 極)



(d) 回路図記号:
ハウジング (3 極)



(e) 回路図記号:
コネクタ (3 極)

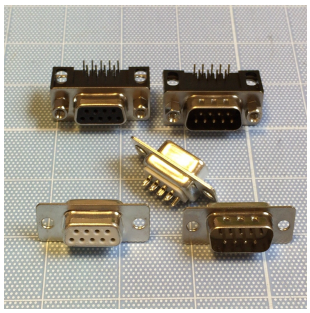


(f) 回路図記号:
ハウジング (4 極)

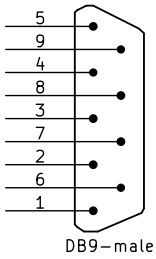


(g) 回路図記号:
コネクタ (4 極)

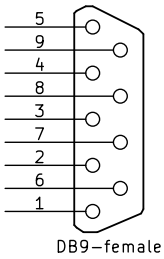
図 3.18: 基板対電線型コネクタ



(a) 外観: D サブコネクタ



(b) 回路図記号: オス



(c) 回路図記号: メス

図 3.19: RS-232C 用 D サブコネクタ

点の数のことを極数と呼んで表します. コネクタの例と回路図での記号を図 3.18 に示します.

コネクタには多種多様なものがありますが, 本書で使うものを 2 種類に分類すると以下の通りです.

- 基板対電線型:　基板と電線の接続を目的としたコネクタです. 基板側にコネクタを取り付けます.
- 電線対電線型:　電線と電線の接続を目的としたコネクタです. コネクタにはオスとメスの違いがあります.

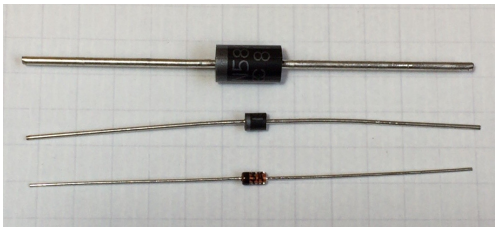
図 3.18 で示した例では, 金属端子のコンタクトに電線を圧着してハウジングに差し込みます. なおコネクタによってはコンタクトを使用せず, 電線を直接ハンダ付けするものやドライバーでネジ止めするものなど様々な形態があります.

RS-232C では図 3.19 に示す D サブコネクタが一般に使用され, オスとメスの違いがありますので注意が必要です.

3.3　半導体部品

整流・スイッチングダイオード

ダイオード (diode) とは一方向のみに電流を流す整流作用を持つ半導体部品です. 家庭用電源の交流 100V あるいはそれを越える電流を整流する目的のものや, 小電流・小電圧でのスイッチングを目的としたものなどがあります. なおここでいう整流ダイオードとは, 接合型ダイオードあるいはショットキーバリアダイオードなどを指します.　なお昔は真空管によるダイオードが使用されていましたが現在では半導体のものが主流ですので, ダイオードといえば半導体ダイオードを指すものとします. ダイオードの例と回路図での記号を図 3.20 に示し



(a) 外観



(b) 回路図記号: 整流・スイッチング



(c) 回路図記号: ショットキバリア

図 3.20: ダイオード

ます.

ダイオードには 2 つの端子があり, それぞれアノード (anode; 陽極) およびカソード (kathode; 陰極) と呼ばれます. 略記の際はそれぞれ A および K と記されます. 電流はアノードからカソードの向きに流れます. すなわち印加電圧の極性が, アノードがプラスでカソードがマイナスの場合に電流が流れます. 装置の組み立ての際には極性を間違わないように取り付けをする必要があります.

ダイオードの重要なパラメータは以下の 3 つです. ここでは大まかな説明だけですので, 詳細はデータシートを参照して下さい.

- 平均整流電流: 壊れることなく順方向に流すことができる電流の最大値を表します. なおこれは 60Hz の交流が印加された場合の値です.
- 降伏電圧 (逆方向電圧): 壊れることなく印加できる逆方向の電圧の最大値を表します. 降伏電圧は無限大であるのが理想ですが, 実際のダイオードでは大きなマイナスの電圧がかかると壊れてしまいます.
- 順方向電圧: ある一定値の電流が流れるために必要な, ダイオードに印加すべき電圧の値を表します. この電圧以下の場合はほとんど電流は流れません. 順方向電圧はゼロであるのが理想ですが, 実際のダイオードでは若干の電圧をかけないと電流は流れません. ダイオードの種類により異なり, 接合型ダイオードでは 0.6V 程度, ショットキバリアダイオードでは 0.3V 程度です.

ダイオードに電流を流すダイオードには電圧が生じます. これが電圧降下となるので, この値を考慮に入れて回路設計をする必要があります. 生じる電圧の値は電流値にもよりますが, データシートに記載の順方向電圧の値が目安となります.

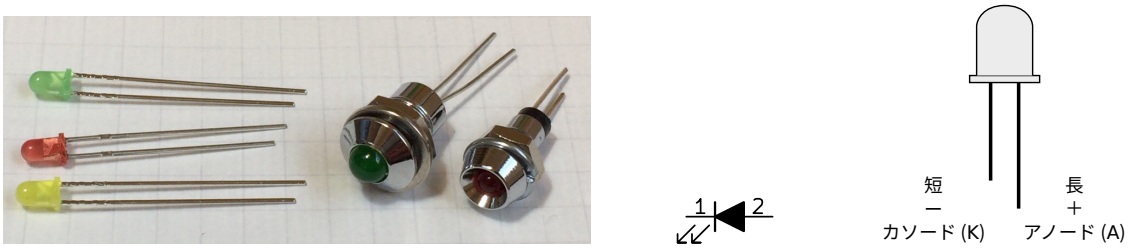
発光ダイオード LED

LED (light emitting diode; 発光ダイオード) とは電流を流すと発光する半導体部品です. LED はダイオードの一種ですが, 整流作用よりも光源としての目的で使用されます. LED の例と回路図での記号を図 3.21 に示します.

LED には整流ダイオードと同様に, 2 つの端子はそれぞれアノード (anode, A; 陽極) およびカソード (kathode, K; 陰極) と呼ばれています. アノード A からカソード K に電流が流れるときに LED は発光します. これら 2 つの端子の区別は長さで区別でき, 長い方がアノード A で + 側, 短い方がカソード K で – 側と決められています. 装置の組み立ての際には極性を間違わないように取り付けをする必要があります.

LED の重要なパラメータは以下の 2 つです.

- 順方向電流: 壊れることなく順方向に流すことができる電流の最大値を表します. この値は LED にもよりますが, 電子機器でインジケータ用に使う小型の LED の多くの場合で, 30mA 程度の値です.



(a) 外観　**(b)** 回路図記号　**(c)** 極性の区別

図 3.21: 発光ダイオード LED

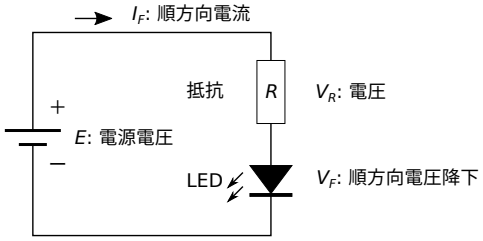


図 3.22: LED と電流制限抵抗

- 逆電圧:　壊れることなく逆方向に印加できる電圧の最大値を表します. この値は LED にもよりますが 5V 程度の値です. 逆電圧の値が低いために整流用の目的では LED を使用しません.

昔は光源には電球を用いていましたが, 電球は多くの電流を必要とし, フィラメントが切れやすいという短所がありました. LED はわずかな電流で発光し, しかも長寿命である点は大きな長所です. 最近の超高輝度 LED では 5mA も流せば十分な明るさに発光します.

LED に接続する抵抗値の決め方について簡単に説明します. 図 3.22 に示すように, 電源 E (ボルト) に抵抗 R (オーム) と LED を接続した場合を考えます. 適切な抵抗値 R は以下のようにして求めます.

- まず LED に流す電流値 I_F を決めます. ただし絶対最大規格を超えてはなりません. 通常の LED なら 1mA から 20mA 程度の範囲です. この I_F を順方向電流と呼びます.
- LED のデータシートから, 順方向電圧降下値 V_F を調べます.
- オームの法則に基づき, 抵抗値 R の計算式は以下の通りになります.

$$\begin{aligned} E &= V_R + V_F \\ &= I_F R + V_F \\ \therefore R &= (E - V_F)/I_F \end{aligned}$$

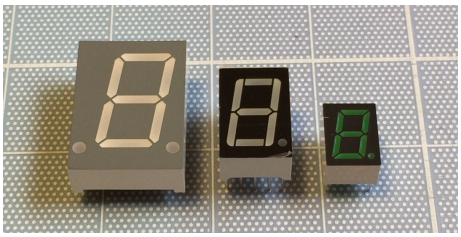
表 3.2 に, 順方向電圧降下値が $V_F = 2.1$V の OptoSupply 社赤色 LED OSR5JA3Z74A および緑色 LED OSG8HA3Z74A での計算例を示します. なおデータシート記載の V_F のは個体ごとにばらつきがありますので, 計算結果に基づいて厳密に抵抗値を選定しなくてもまず大丈夫です.

7 セグメント LED

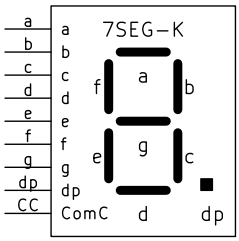
7 セグメント LED (seven-segment LED) とは数字 1 桁と小数点を表示できるように複数の LED を配置した半導体部品です. 数字 1 桁の表示に 7 つの LED を用いることから, 7 セグメント LED と呼ばれています. 7 セグメントの例と回路図での記号を図 3.23 に示します.

表 3.2: LED 電流制限抵抗の計算例

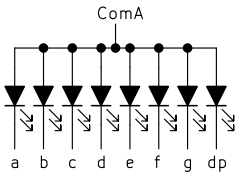
E	I_F	V_F	R
12V	2mA	2.1V	4950Ω
12V	5mA	2.1V	1980Ω
12V	10mA	2.1V	990Ω
5V	2mA	2.1V	1450Ω
5V	5mA	2.1V	580Ω
5V	10mA	2.1V	290Ω



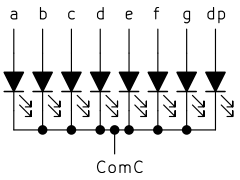
(a) 外観



(b) 回路図記号: カソードコモン



(c) 内部回路: アノードコモン



(d) 内部回路: カソードコモン

図 3.23: 7 セグメント LED

数字の表示は 7 つのセグメント a から g それぞれに対応する LED の発光の組み合わせで行います. 小数点の表示はセグメント dp に対応する LED の発光で行います. 7 セグメント LED 内の 8 つの LED は内部の接続によって 2 通りに分類されます.

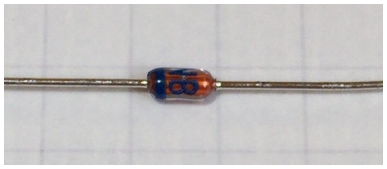
- カソードコモン:　8 つの LED のカソードすべてをひとつの共通端子に結線しています. 8 つの LED のアノードはそれぞれ独立した端子に結線しています.
- アノードコモン:　8 つの LED のアノードすべてをひとつの共通端子に結線しています. 8 つの LED のカソードはそれぞれ独立した端子に結線しています.

定電流ダイオード CRD

CRD (current regulative diode; 定電流ダイオード) とは印加する電圧に関わらずに流れる電流を一定に保つはたらきを持つ半導体部品です. CRD の例と回路図での記号を図 3.24 に示します. CRD には極性があるため, 装置の組み立ての際には極性を間違わないように取り付けをする必要があります. ダイオードという言葉が名前に使われていますが, 実際には整流作用はありませんので注意が必要です.

CRD の重要なパラメータは以下の 2 つです.

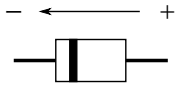
- ピンチオフ電流:　ある一定電圧を印加したとき CRD に流れる電流値を表します. ピンチオフ電流の値



(a) 外観

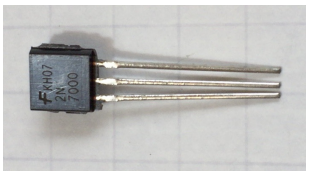


(b) 回路図記号

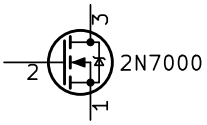


(c) 極性の区別

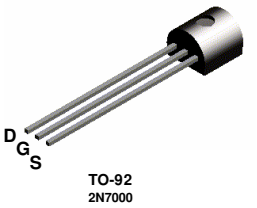
図 3.24: 定電流ダイオード CRD



(a) 外観



(b) 回路図記号: 1: ソース (S), 2: ゲート (G), 3: ドレイン (D)



(c) ピン配置 (FAIRCHILD 社発行のデータシートより引用)

図 3.25: 電界効果トランジスタ FET 2N7000

は 1mA 程度から 20mA 程度の範囲です.

- 定格電圧: 壊れることなく印加できる電圧の最大値を表します.
- 逆方向許容電流: 壊れることなく逆方向に流すことができる電流の最大値を表します.

抵抗器を使って LED に所望の電流値を流す場合, 電源電圧の値に依存して抵抗値が決まります. 従って電源電圧が変化すれば LED に流れる電流値も変化して LED の明るさが変わるだけでなく, 場合によっては LED の絶対最大定格を越えてしまいます. しかし CRD を使えば電源電圧が変化しても LED に流れる電流値を一定に保つことができて便利です.

電界効果トランジスタ FET

FET (field effect transister; 電界効果トランジスタ) とは増幅やスイッチングなどの用途に使用される半導体部品です. FET にはゲート (G), ドレイン (D), ソース (S) の 3 つの端子があり, ゲート電圧によりソース・ドレイン間の電流を制御します. FET の例と回路図での記号を図 3.25 に示します.

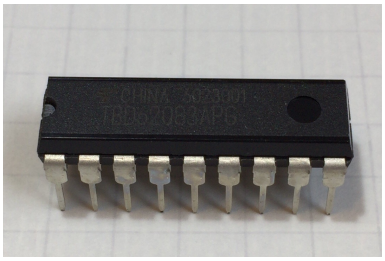
本書で使用する FET は多くのメーカで製造されている 2N7000 で, その特性は以下の通りです.

- 最大ドレイン電流: 200mA
- 最大ドレイン・ソース電圧: 60V
- 最大ドレイン・ゲート電圧: 60V
- 最大ゲート・ソース電圧: ±20V

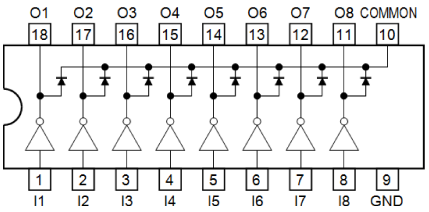
トランジスタアレイ

トランジスタアレイとはひとつの IC パッケージに複数のトランジスタ (あるいは FET) が含まれている半導体部品です. 本書で使用するトランジスタアレイは東芝 TBD62083APG で, 回路図での記号を図 3.26 に示します. 特性は以下の通りです.

- チャンネル (回路) 数: 8

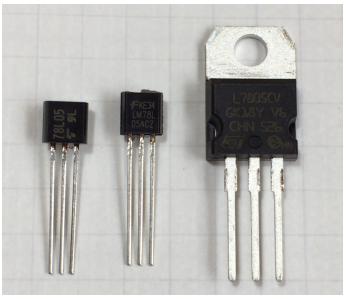


(a) 外観

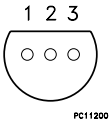


(b) ピン配置 (東芝発行のデータシートより引用)

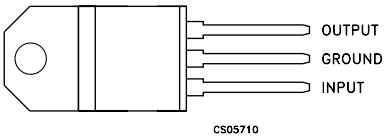
図 3.26: トランジスタアレイ TBD62083APG



(a) 外観



(b) ピン配置 L78L05AC-Z
(STmicroelectronics 社発行の
データシートより引用)



(c) ピン配置 L7805CV-DG
(STmicroelectronics 社発行の
データシートより引用)

図 3.27: 定電圧レギュレータ 7805

- 出力電流: 1 チャンネルあたり最大 500mA
- 出力耐圧: 最大 50V
- 出力オン: 入力電圧が 2.5V 以上の場合
- 出力オフ: 入力電圧が 0.6V 以下の場合

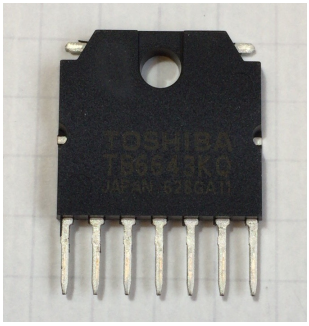
TBD62083APG はシンクタイプと呼ばれるトランジスタアレイです. つまり出力がオンになると出力端子は GND と導通状態となり, 電流は IC 外部から内部へと吸い込まれます.

定電圧レギュレータ

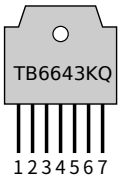
定電圧レギュレータ (voltage-regulator) とは入力電圧に関わらず一定の電圧を出力する半導体部品です. 消費電力の変化に関わらず一定電圧を回路に供給する目的で使用されます. 3 つの端子を持つ定電圧レギュレータは三端子レギュレータとも呼ばれています. 出力電圧が 5V の定電圧レギュレータの代表的なものが 7805 と一般的に呼ばれている三端子レギュレータで, 本書ではこれを使用します. 7805 は多くのメーカが製造しており, 出力電流, 入力電圧, 最低入出力間電圧差などのパラメータで製品ごとに特色があります. 本書で紹介する装置では, 7805 の中でも出力電流が 100mA と 1.5A の 2 種類を使用しています. 図 3.27 に回路図での記号とピン配置を示します.

定電圧レギュレータの重要なパラメータは以下の 3 つです.

- 出力電圧: 出力に一定に保つ電圧値を表します.
- 出力電流: 壊れることなく出力できる電流の最大値を表します.
- 入力電圧: 壊れることなく入力に印加できる入力電圧の最大値を表します.



(a) 外観



(b) ピン配置

番号	名称	説明
1	IN1	制御信号入力 1 ピン
2	IN2	制御信号入力 2 ピン
3	OUT1	出力ピン 1
4	GND	GND ピン
5	OUT2	出力ピン 2
6	N.C.	接続なし
7	VM	電源電圧印加ピン

(c) ピン説明

図 3.28: モータドライバ TB6643KQ

- 最低入出力間電圧差:　出力電圧を一定に保つことが保証されている, 入力と出力の電圧差の最小値を表します.

モータドライバ TB6643KQ

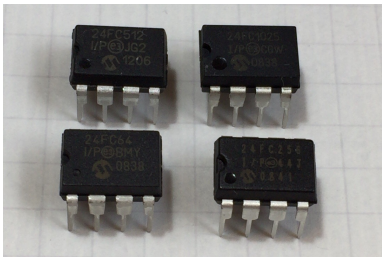
モータドライバ (motor driver) とはモータの回転を制御するための半導体部品です. マイコンからはモータの回転に十分な電力を直接には供給できないため, マイコンとモータの間にモータドライバを入れることでマイコンからモータの回転制御が可能となります. 本書で使用するモータドライバは東芝 TB6643KQ で, 模型車両に搭載されている DC モータの駆動のほか, ソレノイド方式のポイントマシンの駆動も可能です. TB6643KQ の回路図での記号は図 3.28 に示す通りで, その特性は以下の通りです.

- 電源電圧: 10V ~ 45V
- 最大出力電流: 4.0A
- PWM 制御可能: 100kHz (最大)
- 保護機能: 過電流検出回路 (ISD), 過電圧検出回路 (VSD), 熱遮断回路 (TSD), 低下電圧検出回路 (UVLO), 貫通電流防止時間内蔵
- H ブリッジ

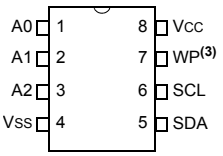
TB6643KQ には, マイコンからの入力ピン IN1 と IN2 の 2 つがあり, モータへの出力ピン OUT1 と OUT2 の 2 つがあります. これら入力と出力の関係を表 3.3 に示します. 入力ピン IN1 と IN2 がともに H (High) の場合は出力ピン OUT1 と OUT2 はショートモードになります. このときには出力ピンへの電流供給を断った上で OUT1 と OUT2 は短絡状態になります.

表 3.3: TB6643KQ 入出力ファンクション表

入力		出力		
IN1	IN2	OUT1	OUT2	モード
H	H	L	L	ショートモード
L	H	L	H	正転/逆転
H	L	H	L	逆転/正転
L	L	ハイインピーダンス		ストップ (TSD, ISD 解除)



(a) 外観



(b) ピン配置
(Microchip Technology 社発行の
データシートより引用)

図 3.29: EEPROM 24FC64

表 3.4: 本書で使用する主な EEPROM

	メモリ容量	
24FC64	64K ビット	(8K バイト)
24FC256	256K ビット	(32K バイト)
24FC512	512K ビット	(64K バイト)
24FC1025	1024K ビット	(128K バイト)

I^{2}C EEPROM

EEPROM とはデータを記憶する半導体部品です. 本書では表 3.4 に示す Microchip Technology 社の I^{2}C 接続の EEPROM をマイコン外部のデータ記憶に使用します. EEPROM の例と回路図での記号を図 3.29 に示します. なお, 表 3.4 に掲載の部品のピン配置はいずれも同じで, 図 3.29 で示す通りです.

I/O エクスパンダ

I/O エクスパンダとは多数のデジタル入出力ポートを持ち, マイコンのピン数の不足を補うための半導体部品です. 本書では図 3.1 に示す Microchip Technology 社の MCP23017 を使用します. その特性は以下の通りです.

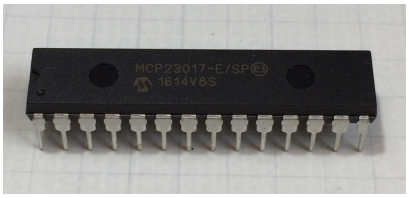
- 16 本の双方向デジタル I/O ピン (それぞれ 8 本のポート A とポート B で構成)
- IOC (Interrupt-On-Change; 入力変化割り込み) 機能
- I^{2}C 接続
- ハードウエアでのアドレス設定で最大 8 つまで同時に接続可能

光変調型フォト IC

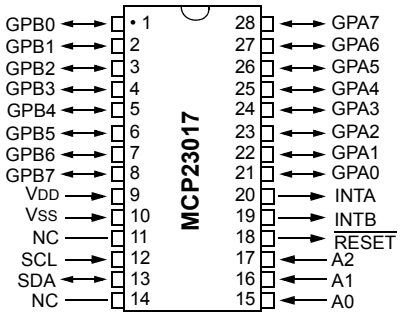
浜松ホトニクス製の光変調型フォト IC S7136 および S6809 は, 別途接続した赤外線 LED からの受光の有無を検出します. 本書では模型車両の検出に光変調型フォト IC を使用した方法を紹介します. 模型列車の存在の検出は, 模型列車による光線の遮断, あるいは, 模型列車による光線の反射により行います. 光変調型フォト IC S7136 および S6809 と回路図での記号を図 3.31 に示します.

光変調型フォト IC S7136 および S6809 の特性は以下の通りです.

- 電源電圧: 4.5V ~ 16V
- 検出出力: オープンコレクタ
- LED カソード駆動出力: オープンコレクタ

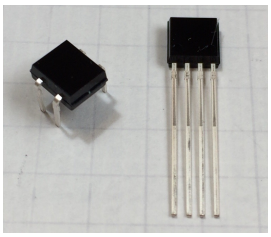


(a) 外観

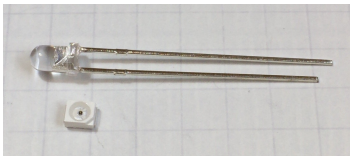


(b) ピン配置 (Microchip Technology 社発行のデータシートより引用)

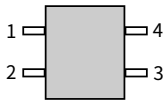
図 3.30: I/O エクスパンダ MCP23017



(a) 外観: S6809(左) と S7136(右)



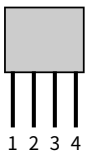
(b) 外観: 赤外線 LED 2 種



(c) ピン配置: S7136

番号	名称	説明
1	カソード (LED)	LED 駆動ピン
2	VCC	電源ピン
3	VOUT	検出出力ピン
4	GND	GND ピン

(d) ピン説明: S7136

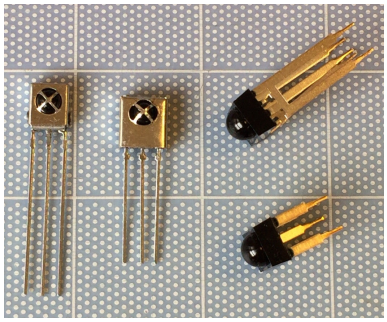


(e) ピン配置: S6809

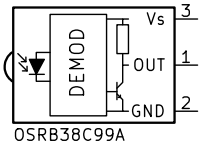
番号	名称	説明
1	VOUT	検出出力ピン
2	GND	GND ピン
3	カソード (LED)	LED 駆動ピン
4	VCC	電源ピン

(f) ピン説明: S6809

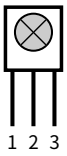
図 3.31: 光変調型フォト IC と赤外線 LED



(a) 外観

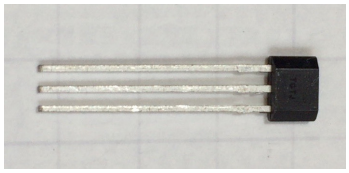


(b) 回路図記号
OSRB38C9AA

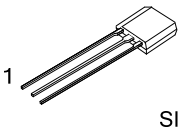


(c) OSRB38C9AA ピン配置

図 3.32: 赤外線リモコン受信モジュール各種



(a) 外観



(b) ピン配置 (ユニソック発行のデータシートより引用)

番号	名称	説明
1	VCC	電源ピン
2	GND	GND ピン
3	VOUT	検出出力ピン

(c) ピン説明

図 3.33: ホール IC SK8552G

- 可視光カットフィルタ内蔵

光変調型フォト IC の特徴は, 外乱光の下でも誤動作が少ない点にあります. 変調をかけて赤外線 LED を点灯して, 光変調型フォト IC での受光がその変調に一致するかを検査することで誤動作を少なくしています. 光変調型フォト IC の価格が若干高めなのは短所ですが, 外乱光に強くて検出が確実なのが長所です.

赤外線リモコン受信モジュール

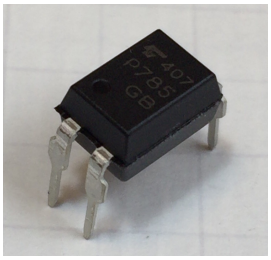
赤外線リモコン受信モジュールとは, テレビのリモコンで使用されている赤外線信号を受信するための半導体部品です. 38kHz の副搬送波で変調した赤外線を論理信号に変換する機能を有します. 赤外線リモコン受信モジュールの外観の例と回路図での記号を図 3.32 に示します.

ホール IC

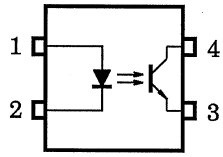
ホール IC (hall IC) とは磁界の有無を検出する半導体部品です. リードスイッチは磁力により接点を機械的に動かしてオン・オフを切り替えますが, ホール IC はホール効果を利用して電子的にオン・オフを切り替えます. 本書で使用するホール IC はユニソック SK8552G で, 外観とピン配置を図 3.33 に示します. その特性は以下の通りです.

- 電源電圧: 3V ~ 20V
- 無極性

このホール IC は無極性のため磁界の向きはどちらの場合でも検出されます. 従って磁石の S 極/N 極の向きをまったく気にせずに模型車両に磁石を取り付けられる点が長所です. 極性のあるホール IC では磁石の S 極/N 極の向きを考えて取り付けないといけません.



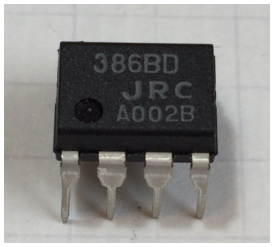
(a) 外観



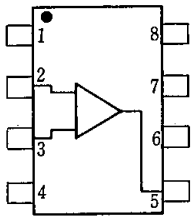
1：アノード
2：カソード
3：エミッタ
4：コレクタ

(b) ピン配置: TLP785
(東芝発行のデータシートより引用)

図 3.34: フォトカプラ TLP785



(a) 外観



ピン配置
1．GAIN
2．−INPUT
3．+INPUT
4．GND
5．OUTPUT
6．V^+
7．BY PASS
8．GAIN

(b) ピン配置 (新日本無線発行の
データシートより引用)

図 3.35: オーディオアンプ IC NJM386BD

フォトカプラ

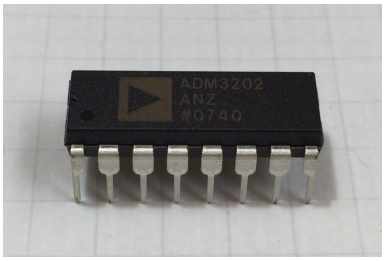
フォトカプラ (photocoupler) とは電気的に絶縁をして信号を伝える半導体部品です. 入力側に発光素子が, 出力側に受光素子があります. 入力信号の有無を発光素子が光に変換し, 受光素子の受光の有無を出力信号とします. 入力と出力を光で結合することで電気的な絶縁が実現されています. 本書で使用するフォトカプラは東芝 TLP785-GB で, 外観とピン配置を図 3.34 に示します. その特性は以下の通りです.

- 変換効率: 100 ~ 300 (GB ランク)
- 発光側最大順電流: 60mA
- 発光側最大逆電圧: 5V
- 受光側最大コレクタ電流: 50mA
- 受光側最大コレクタ・エミッタ間電圧: 80V
- 受光側最大エミッタ・コレクタ間電圧: 7V

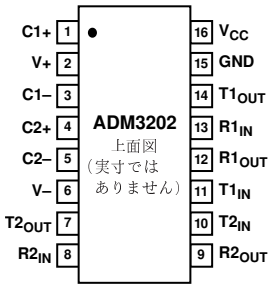
オーディオアンプ IC

オーディオアンプ IC は音声信号を増幅してスピーカで鳴らすための半導体部品です. 本書で使用するオーディオアンプ IC は新日本無線による NJM386BD です. これは 386 と総称されるオーディオアンプ IC の製品のひとつで, 幾つもの半導体メーカがコンパチブルな IC を製造しています. NJM386BD の外観とピン配置を図 3.35 に示します.

NJM386BD の特性は以下の通りです.



(a) 外観



(b) ピン配置 (アナログデバイセズ発行のデータシートより引用)

図 3.36: RS-232C インターフェース IC ADM3202AN

表 3.5: 本書で使用する PIC マイコン

	ピン (端子) 数	プログラムメモリ	データメモリ
16F18877-I/P	40	32K ワード	4096 バイト
16F18857-I/SP	28	32K ワード	4096 バイト
16F18346-I/P	20	16K ワード	1024 バイト
16F18326-I/P	14	16K ワード	1024 バイト
16F18313-I/P	8	2K ワード	256 バイト

- 出力電力: 1W
- 電源電圧: 4V ~ 18V
- 電流増幅度: 20 ~ 200
- 単電源動作

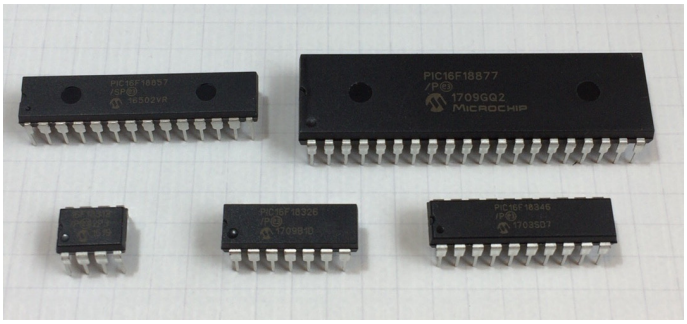
RS-232C インターフェース IC

RS-232C (EIA-232) インターフェース IC とは, マイコンでの 5V あるいは 3.3V でのロジック電圧レベルと RS-232C で規定されている電圧レベルとの間で電圧変換を行う半導体部品です. 本書ではアナログデバイセズ ADM3203AN を使用し, その外観とピン配置を図 3.36 に示します. 特性は以下の通りです.

- 送信 2ch, 受信 2ch
- 動作電圧範囲: 3.0V ~ 5.5V
- RS-232C レシーバ入力電圧範囲: -30V ~ +30V
- RS-232C トランスミッタ出力電圧振幅: ±5.2V
- 最大データレート: 460kbps

3.4　PIC マイクロコントローラ

本書で用いるマイコン (マイクロコントローラ) は, Microchip Technology 社の 8 ビットマイコン PIC 16F シリーズです. PIC マイコンは豊富な周辺 (ペリフェラル) モジュールを内蔵しており, マイコンひとつで様々な外部装置の制御が可能です. 本書で使用する PIC マイコンの一覧を表 3.5 に示します. PIC マイコンの例を図 3.37 に示し, ピン配置を図 3.38 に示ます.

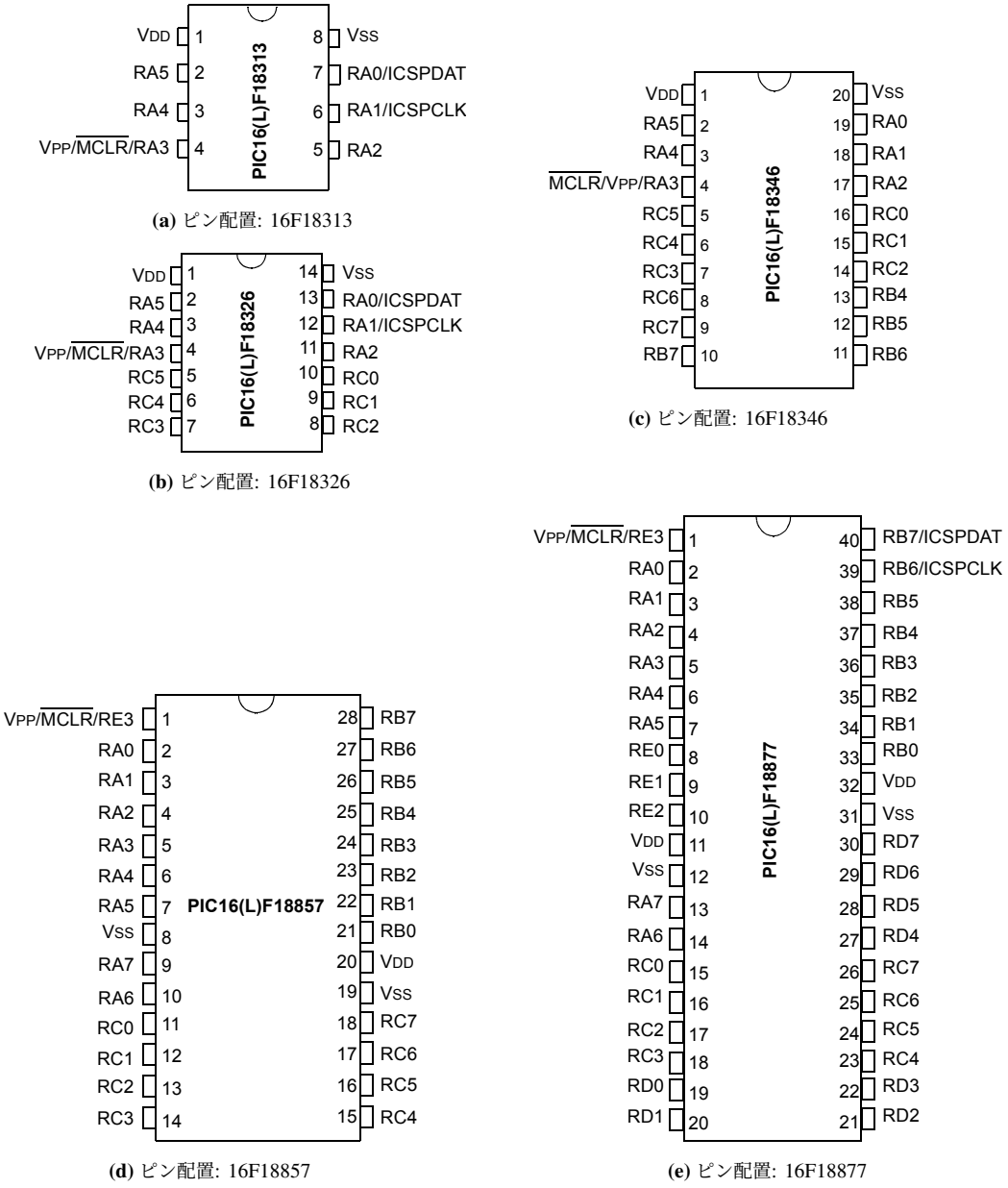


(a) 外観: 右上から時計回りに, 16F18877,
16F18346, 16F18326, 16F18313, 16F18857

図 3.37: PIC マイコン各種 (16F18xxx シリーズ)

PIC マイコンに内蔵されているモジュールのうち, 本書で使用するものを簡単に紹介すると以下の通りです. 同種のモジュールを複数個搭載している場合もあり, その数は PIC マイコンごとに異なります. PIC マイコンごとの内蔵周辺モジュール装備の一覧表を表 3.6 に示します.

- WDT (Watchdog Timer)
  ウオッチドッグタイマです. プログラムのバグで無限ループが生じたときにマイコンをリセットして復帰させます.
- デジタル I/O ポート (Digital Input/Output Port)
  1 ビット単位でデジタル信号を入出力します. 信号の入力または出力の選択はソフトウエアで行います.
- IOC (Interrupt-On-Change)
  入力変化割り込みです. デジタル入力ピンの入力値の変化をハードウエア的に捕捉します.
- PPS (Peripheral Pin Select)
  内蔵周辺モジュールのピン割り当ての設定です. マイコン内部の内蔵周辺モジュールの入出力端子とマイコン外部のピンとの対応関係をソフトウエアにより変更します.
- ADC (Analog-to-Digital Converter)
  アナログ-デジタル変換です. ピンに印加されているアナログ電圧値を 10 ビットのデジタル値に変換します.
- TMR (Timer)
  タイマです. 時間の経過の計測や事象の発生回数のカウントのほか, PWM 波形の生成にも使用します. PIC マイコンのタイマは以下の通り 3 種類に大別されます.
  - TMR0
    8 ビットタイマです. PIC マイコンによっては 16 ビットモードが用意されており, 設定により 16 ビットタイマとして使用可能です.
  - TMR1, TMR3, TMR5
    16 ビットタイマです. これらは互いに同一機能を持ちますが, 実際の搭載数は PIC マイコンにより異なります.
  - TMR2, TMR4, TMR6
    8 ビットタイマです. これらは互いに同一機能を持ちますが, 実際の搭載数は PIC マイコンにより異なります. 各タイマには周期レジスタ PR が付随しており, タイマの値と PR の値が一致するとタイ



(a) ピン配置: 16F18313

(b) ピン配置: 16F18326

(c) ピン配置: 16F18346

(d) ピン配置: 16F18857

(e) ピン配置: 16F18877

図 3.38: PIC マイコンのピン配置
(Microchip Technology 発行のデータシートより引用)

マの値をリセットする機能があります.

- CCP (Capture/Compare/PWM)
  キャプチャ/コンペア/PWM の 3 つの機能を持ったモジュールです. 多彩な使用方法がありますが, 本書ではタイマ出力を入力として PWM 波を生成する目的で使用します.
- CWG (Complementary Waveform Generator)
  相補的波形生成器です. 多彩な使用方法がありますが, 本書では CCP 出力を入力としてモータドライバへ供給する PWM 波を出力する目的で使用します.
- CM (Comparator)

コンパレータです. マイコンの端子に印加されているアナログ電圧値を他の電圧値と比較して, 大小の比較結果を 0 または 1 の値で得ます.

- FVR (Fixed Voltage Reference)
  固定参照電圧器です. 一定の電圧値を安定的に発生させて, コンパレータでの比較の基準電圧などに使用します.
- MSSP (Master Synchronous Serial Port)
  同期 2 線式シリアル通信 I^2C や同期 3 線式シリアル通信 SPI の同期式通信を行います. 本書では I^2C 方式を使用します. I^2C での通信はクロック信号線 SCL と双方向データ線 SDA の 2 つを用いて行います. データの送受信タイミングは主となる機器が制御する SCL に従い, 同期的に行われます.
- EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transmitter)
  本書では RS-232C 非同期シリアル通信で使用します. 通信は送信データ線 TX と受信データ線 RX の 2 つを用いて行います. クロック信号はありませんので, データの送受信タイミングはあらかじめ送受信双方の機器で合意し設定された通信速度に従い, 非同期的に行われます.

課題

1. 以下のカラーコードで示される抵抗値を示して下さい. (ただし誤差については省略します.) 緑・茶・金, 茶・緑・黒, 茶・灰・茶, 茶・緑・赤, 黄・紫・橙.
2. 以下に示す抵抗値に対応するカラーコードを示して下さい. (ただし誤差については省略します.) 8Ω, 240Ω, 2kΩ, 2.7kΩ, 100kΩ.
3. 電解コンデンサの寿命について, コンデンサメーカの公開している技術情報等から調査して説明して下さい.
4. 表 3.2 で示した LED 電流制限抵抗の計算例の数値が正しいか確かめて下さい.
5. モータドライバ IC には本章で紹介したもの以外にどのような製品があるかいくつか調査してそれぞれの特徴を説明して下さい.
6. 小型のオーディオアンプ IC には本章で紹介したもの以外にどのような製品があるかいくつか調査してそれぞれの特徴を説明して下さい.
7. PIC 以外にはどのようなマイクロコントローラがあるか調査してそれぞれの特徴を説明して下さい.

表 3.6: PIC マイコン内蔵周辺モジュール装備の一覧 (抜粋)

	16F18877	16F18857	16F18346	16F18326	16F18313
AD 変換	○	○	○	○	○
FVR	○	○	○	○	○
ESUART					
ESUART1	○	○	○	○	○
CCP					
CCP1	○	○	○	○	○
CCP2	○	○	○	○	○
CCP3	○	○	○	○	
CCP4	○	○	○	○	
CCP5	○	○			
コンパレータ					
C1	○	○	○	○	○
C2	○	○	○	○	
CWG					
CWG1	○	○	○	○	○
CWG2	○	○	○	○	
CWG3	○	○			
MSSP					
MSSP1	○	○	○	○	○
MSSP2	○	○	○		
タイマ					
TMR0	○	○	○	○	○
TMR1	○	○	○	○	○
TMR2	○	○	○	○	○
TMR3	○	○	○	○	
TMR4	○	○	○	○	
TMR5	○	○	○	○	
TMR6	○	○	○	○	

第 4 章

マイコン電子工作の基礎練習

本章では簡単な LED 点灯装置の作成を通じてマイコン電子工作の基礎練習を行います. ここでは以下の内容を学び, 後の章で紹介する装置の製作方法の習得を目的とします. すでに PIC マイコンで電子工作に慣れ親しんでいる方は, 本章は軽く目を通す程度で結構です. しかし電子工作や PIC マイコンでのプログラム開発に不慣れな方は, ここでしっかりと基礎的な練習して下さい. 本章で紹介する回路の組み立てができなければ, さらに高度な後の章の装置の組み立てはとてもできないからです.

- 本書での回路図と部品表の見方
- 電子回路の組み立て方法
- 統合開発環境開発環境 MPLAB X IDE を用いた C 言語によるファームウエア (プログラム) の開発
- PICkit4 を用いた PIC マイコンへのファームウエア書き込み方法

本装置の試作例の写真を図 4.1 に示します.

本書で製作する装置の多くでマイコンを使っています. アナログ制御の鉄道模型にマイコンは不要だと思うかもしれませんが, 複雑な動作や高度な機能もマイコンのファームウエアのプログラミングを工夫することで簡単に実現できます. 本書では, 電子回路の設計の工夫をして複雑な動作や高度な機能を実現するやりかたとは異なる方向性で進めてゆきます. このことから, ファームウエアの設計と理解は電子回路以上に重要ですので, 十分に理解するように努めて下さい.

後の章では, 本章のファームウエアを様々に拡張します. 第 5 章では LED の点灯と消灯を電子的に外部制御可能な機能を実現し, 第 7 章と第 8 章では様々な点灯パターンを実現します.

4.1 回路図

回路図を図 4.2 に, 部品表を表 4.1 に, それぞれ示します. **この回路の電源には 5V を使用しますので, 必ず 5V 出力の AC アダプタを使用してください. 間違って 12V 出力の AC アダプタを使用すると回路が破損する場合がありますので, 十分に注意して下さい.**

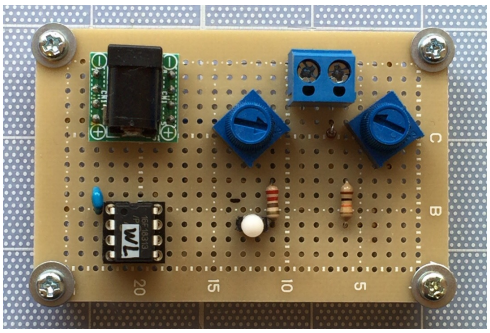


図 4.1: LED 点灯装置の試作例

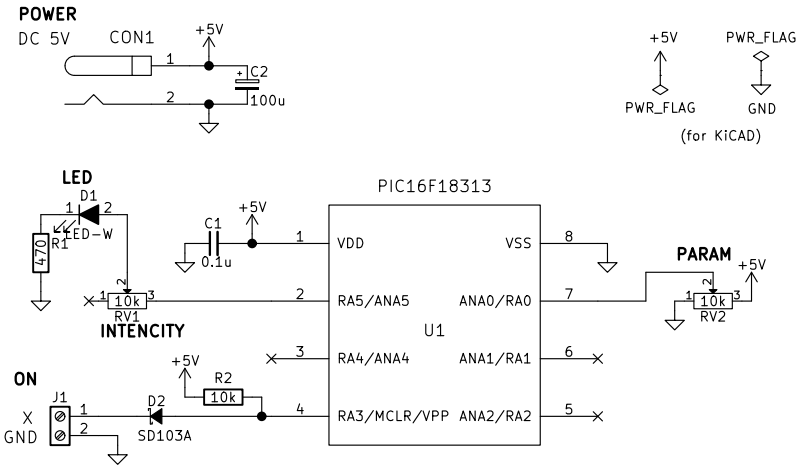


図 4.2: 【回路図】 LED 点灯

図 4.2 の右上に「(for KiCAD)」と書いてある部分は無視して下さい. これは電子回路 CAD ツール KiCAD において回路チェックを行うためのもので, 回路そのものではないからです.

部品表の備考欄における [ ] 内の記号は秋月電子通商における商品番号 (本書執筆時点) ですので, 購入の際の参考にして下さい. もし販売終了となった場合には同等の他の部品で代用して下さい.

本装置には制御入力のためのターミナルを用意しています. このターミナルは点灯のオン・オフを外部から制御可能にするためのもので, 第 5 章と第 7 章のファームウエアで活用します.

表 4.1: 【部品表】 LED 点灯装置

記号	数量	部品名
U1	1	PIC 16F18313-I/P [I-10889]
D2	1	ショットキーバリアダイオード SD103A [I-04271]
D1	1	LED　色は適宜選択
C2	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C1	1	セラミックコンデンサ 0.1μF [P-00090]
R1	1	抵抗 470Ω　1/4W [R-25471]
R2	1	抵抗 10kΩ　1/4W [R-25103]
RV1, RV2	2	可変抵抗器 10kΩ　B カーブ, 半固定型 [P-08012]
	1	IC ソケット 8 ピン [P-00035]
	1	オルタネイト型押しボタンスイッチ　スライドスイッチでも可
J1	1	ターミナルブロック 2 ピン [P-02333]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか基板・配線材等

本装置で使用する PIC マイコンは 16F18313 で, このマイコンでの入出力ピンは以下の通りに使用します.

- RA5 ピン: デジタル出力

この出力を H (High; 5V) または L (Low; 0V) に変化させ, LED に電流を流して点灯させます. LED に流れる電流は可変抵抗器 INTENCITY (RV1) を通じてハードウエア的に加減できるようにしてあります.

- RA3 ピン: デジタル入力
  ターミナルブロック ON (J1) を介して LED 点灯の制御を外部から入力します. ターミナルブロックの X 端子が GND に導通して電圧レベルが L の場合に点灯を行い, 開放 (HiZ) の場合は点灯を行いません. なお本章のファームウエアではこの機能の実現は行わず, 第 5 章と第 7 章のファームウエアで実現します.
- RA0 (ANA0) ピン: アナログ入力
  可変抵抗器 PARAM (RV2) を使用して入力電圧を変化させてアナログ値をマイコンに入力します. 例えば LED が点滅する速度などのパラメータの設定などに使用します. なお本章のファームウエアではこの機能の実現は行わず, 第 7 章のファームウエアで実現します.

4.2　工具の準備

回路の組み立てで使用する工具類は以下の通りです.

- ハンダゴテ
  容量 30W から 40W 程度の電子回路用のものを用意します. 容量の選択に加えてコテ先の選択も重要です. 真鍮の模型車両工作で使用するハンダゴテは電子工作用には向きません.
- ヤニ入り糸ハンダ
  電子工作ではヤニ入り糸ハンダを使用します. 真鍮の模型車両工作で使用するハンダとフラックスは電子工作には向きません.
- ハンダゴテ台とコテ先クリーナ
  高温のハンダゴテを置く場所と, 熱で酸化したコテ先を綺麗にするものです.
- ハンダ吸い取り線
  ハンダ付けを間違ったとき, ハンダを吸い取って部品を取り外すのに使用します.
- ニッパ
  電線の切断に使用します.
- ワイヤストリッパ
  電線の被覆を剥くのに使用します. ニッパでもできるため必ずしも必要ではありませんが, 不慣れな人はワイヤストリッパを使用すると良いでしょう.
- ピンセット
  ハンダ付けの際に熱くなる部品を持つのに使用します. 反作用ピンセットもあると便利です.
- 配線用の電線および錫めっき線
  基板上で配線に使う電線には耐熱電子ワイヤーと呼ばれているものを使用します. これは中の電線が単線で, 被覆が耐熱性のものです. 色は各種用意しておくと良いでしょう. 信号の種類で電線の色を使い分けると配線間違いを見つける際に便利だからです. なお撚り線のものは使い勝手が悪いので基板上での配線には向きません. 錫めっき線は 2 種類用意して, 太めのものを電源関係に, 普通の太さのもを通常の配線に, それぞれ使用すると良いでしょう.
- 結束バンド
  電線の束をまとめるのに役立ちます. 本章のような小規模な回路ではあまり出番はありませんが, 後の章で紹介する複雑な回路を組み立てるときに役立ちます.
- マスキングテープ



図 4.3: 作業机の様子

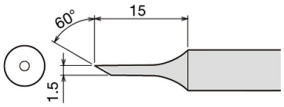


図 4.4: 太陽電機産業 PX-60RT-1.5C (太陽電機産業ホームページより引用)

ハンダ付けの作業は基板を裏返しにして行いますが, このときに部品が落ちてしまわないようにマスキングテープで仮止めをすると作業が楽に行えます. 鉄道模型の塗装の際に使用しているものを流用すれば充分です.

- テスタ
  電気の計測器で, 抵抗値や直流電圧を測定する機能があれば充分です. 正しく結線されているか, ショートはないか, 正しい電圧がきているか, などのチェックで使用します. 本書の電子回路の設計製作にはロジックアナライザやオシロスコープなどの高度な測定機器は不要で, 小型のテスタだけで充分です. 実際に小型のテスタだけで本書のすべての装置を作成しました.
- 作業台
  ハンダ付け作業をしやすい高さで, 熱が加わっても大丈夫な台を用意します. 図 4.3 に作業机の様子を示します.

ハンダ付けについて簡単に説明します. 以下の 3 つの要素が満たされることがハンダ付けを確実に行う秘訣です.

1. ハンダゴテを適切な温度に設定する
   これは人間の技能でコントロールできるものではありませんから良いハンダゴテを買いましょう. 電子部品屋で普通に売っている名の通ったメーカものなら, あまり高価なものでなくてもまず大丈夫です. ハンダゴテの熱量を設定できるパワーコントローラを用意するのも良いでしょう.
2. ハンダゴテの熱を十分に部品に伝える
   コテ先の選定が重要になりますが, これはブラス車両工作のハンダ付けでコテ先を工夫するのと同様です. ハンダゴテに標準的に添付されているコテ先は先端が円錐状のものが多いのですが, これはあまり上手くハンダゴテの熱が部品に伝わらないようです. 図 4.4 に示すような円錐を斜めにカットした形状だとコテ先が大きな面積で部品に接するため熱が確実に伝わります. ただしこれは個人によって好みが分かれるでしょうから, 各自で工夫して下さい.
3. 適切な時間に渡って部品と溶けたハンダの温度を保つ
   これは人間の技能でコントロールできることですので, しっかり研鑽を積んで下さい. 接合されるものとハンダの両方が適切に加熱されることで合金層が形成されて, しっかり接合されます. ハンダを塗り付け

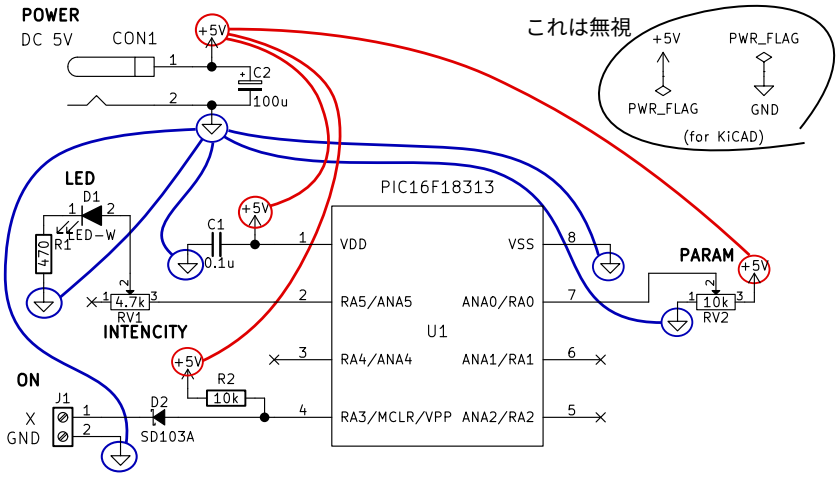


図 4.5: 電源関係 (+5V と GND) の結線

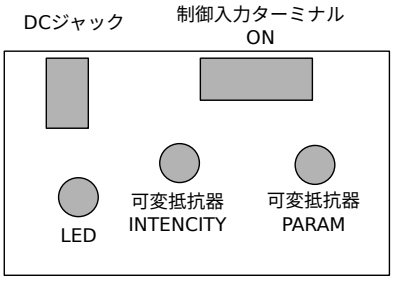


図 4.6: 操作に関係する部品レイアウトと部品名称

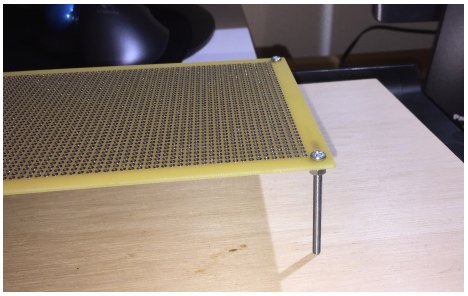


図 4.7: 基板を安定させる足の仮付け

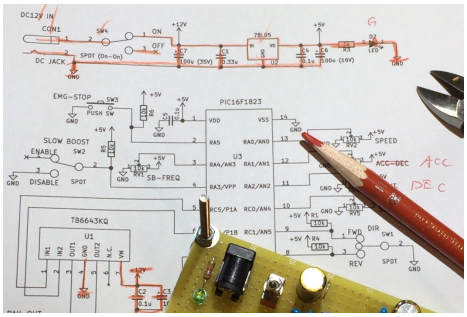


図 4.8: 回路図への作業経過の記入 (第 9 章の回路の組み立て途中での例)

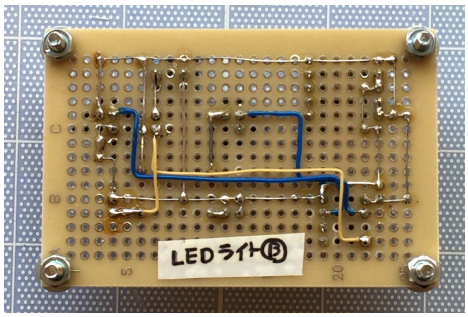


図 4.9: LED 点灯装置の配線面

るような方法では十分に接合はされません.

4.3　回路の組み立て

図 4.2 で示した回路図を実際に組み立てゆく手順の一例を簡単に説明します. 人それぞれやりやすい方法は異なるでしょうから各自で工夫をして下さい. 本装置で操作に関係する部品配置と部品名称を図 4.6 に示しますので, 組み立ての際の参考にして下さい.

1. 工具を揃える
2. 部品を揃える

- 回路図に記載の電子部品
- ユニバーサル基板

  2.54mm (0.1 インチ) ピッチのユニバーサル基板を使います. 片面のものでも両面のものでも, どちらでも構いません. 基板の適切な大きさは回路の複雑さにもよります. 不慣れなうちは少し大きめの基板を使って, ゆったり配線ができるようにすると良いでしょう. 本章の装置では 72mm×47.5mm の程度の大きさの基板で十分でしょう.
- AC アダプタ (5V 出力)

  この DC プラグと DC ジャックの大きさが合うようにします. 一般的なセンタープラス型の AC アダプタを想定して回路図を書いています.

3. 基板の四方に足を仮付けする

   基板がグラグラするとハンダ付け作業が進みません. ハンダ付けをする側である基板の裏側が上になるよう, 図 4.7 のように足を取り付けて安定を図ると作業が非常に楽になります.
4. 部品の配置を決める

   今後の配線作業を考えながら, 基板上での部品の配置をおおままに決めます. 上手な配置だと配線量が少なくて作業が楽になります. 特に最初に決めると良いものは, DC ジャック, ターミナル, コネクタ, 可変抵抗器, LED などといった, 電源, 入力・出力・操作・表示に関係するものです. これらの配置が悪いと後々使い勝手が悪いことになります. 特に電源関係は配線全体に関わりますので一番最初に行います.
5. 電源関係の配線を行う

   最初に電源関係の配線を行います. DC ジャックとそのそばのコンデンサをハンダ付けします. GND 線と +5V 線を可能な限り錫メッキ線で配線します. 回路図の中の記号 

は +5V を,  は GND を, それぞれ表しています. 図 4.5 で示す絵の通り, 具体的な配線は以下の規則で行います.
   - 

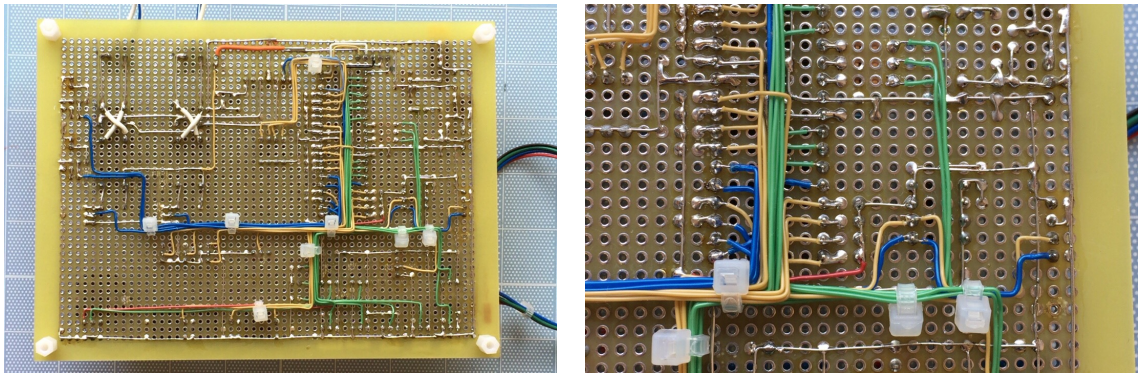
記号の場所をすべて互いに結線します.
   -  記号の場所をすべて互いに結線します.

   この回路では +5V を電源とする部品だけですが, 回路によっては +5V と +12V が混在するものもあります. その場合は, 回路図の電源記号で +5V と描いてあるもの同士だけを結線し, +12V と描いてあるもの同士だけを結線します.
6. 基板への部品のハンダ付けと配線を行う

   決めておいた部品の配置に従って, 部品をハンダ付けをして配線を進めてゆきます. できるだけ錫めっき線を使うようにすると電線の被覆を向く手間が省けて楽ができます. 未配線の部分と配線済の部分とを混乱しないようにするには, 図 4.8 に示すように回路図のコピーを用意しておき, 配線が済むごとにその部分を赤鉛筆で印を付けてゆくと良いでしょう. 図 4.9 は基板裏側の配線面です. これは余りにも下手くそでお手本ではありませんので, これよりも上手に配線するよう心懸けて下さい.
7. テスタによる配線確認を行う

   DC ジャックの +5V と GND の間の電気抵抗をテスタで測定して, ショートしていないか確認します. 抵抗値が 0Ω あるいはそれに近い値であれば電源関係の配線でショートを起こしています. そのほかにも, 回路図通りに配線されているかを部品間の結線をテスタで確認します. 抵抗値が 0Ω であれば部品間で結線されています. もし配線が間違っているようであれば修正をします.
8. 通電を行う

   まだマイコンにファームウエアを書き込んでいないのでなにも動作はしないのですが, とりあえずは AC



(a) 全体図　　**(b)** 拡大図

図 4.10: 結束バンドによる電線の整理 (第 29 章の回路での例)

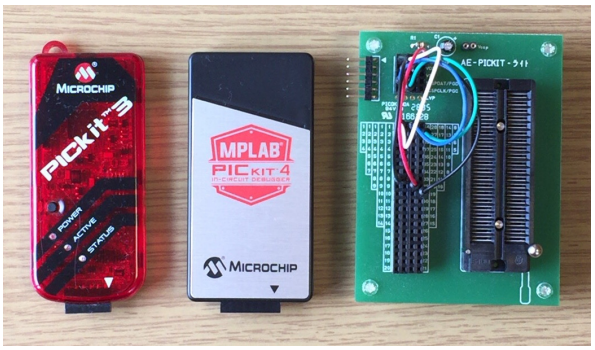


図 4.11: ファームウエアの書き込みツール (左より, Microchip Technology 社 PICkit3, PICkit4, 秋月電子通商 PIC 書き込みアダプタ)

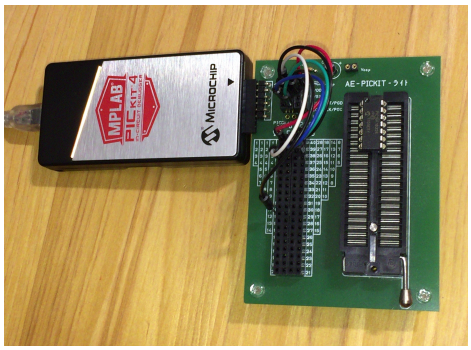


図 4.12: PICkit4 とアダプタを接続し, PIC マイコンをゼロプレッシャーソケットに搭載した様子

アダプタから回路に電源を供給してみます. 発煙や加熱などの異常があればすぐに電気の供給を止めます. 発煙すると部品はもう壊れてしまっていて回復の見込みはほとんどありませんから, 壊れた部品を見つけて取り替えるしかなく大変面倒です. そうならないようにテスタによる配線確認は念入りに行いましょう.

図 4.9 は配線量が非常に少ない場合ですが, 配線量の多い回路を組むときは図 4.10 の例のように結束バンドを使用して電線の整理すると良いでしょう.

4.4　MPLAB X IDE を用いたファームウエアの開発と書き込み

本書掲載のファームウエアを PIC マイコンに書き込むために以下の用意が必要です. 当然ながら回路を組み立ててもマイコンにファームウエアを書き込まないことにはまったく動作はしませんから下記の用意は必須です.

- パーソナルコンピュータ
  下記の統合開発環境の利用のため, Windows, MacOS, Linux のいずれかのオペレーティングシステムが動作するパーソナルコンピュータが必要です.
- 統合開発環境 MPLAB X IDE と C コンパイラ XC8

これらはいずれも Microchip Technology 社のウェブページにて無料で公開されています. 各自でダウンロードして, パーソナルコンピュータにインストールしてください.
なお本章で紹介するファームウエアおよび例は MPLAB X IDE バージョン 5.25, XC8 バージョン 2.10 を使用しています.

- PIC プログラマ PICkit3 または PICkit4
  これは PIC マイコンにファームウエアを書き込む装置です. Microchip Technology 社より数種類が発売されていますが, その中でも安価で入手しやすい PICkit3/4 を使用するのが良いでしょう. 図 4.11 にその写真を示します.
- PIC 書き込みアダプタ
  PICkit3/4 には IC ソケットがないために単体ではマイコンチップと接続ができません. ゼロプレッシャソケットのついた書き込みアダプタが別途必要です. 簡単に自作ができますが, 秋月電子通商の「PICkit 対応 ICSP 書き込みアダプターキット」は必要な部品が全て揃っていますので便利です. 図 4.11 にその写真を示します. 図 4.12 では, PICkit4 とアダプタを接続して, PIC マイコンをゼロプレッシャーソケットに搭載した様子を示します.

作成した LED 点灯装置の簡単な動作確認を次に示すソースコードを使って行います. このソースコードでは LED を 1 秒間隔で点滅します. これはいわゆる「L チカ」と呼ばれているもので, マイコン電子工作界での「HELLO WORLD」です. プログラムとしてはごく僅かなものですが, LED の点滅によりデジタル I/O 出力を確認し, 点滅時間によりクロックの発振周波数を確認するという, マイコン動作の基本を確認できるので非常に有用です.

ソースファイル番号 4.1: LED-Test.c

```
1  /*************************************************
2   * LED点滅テスト / PIC16F18313
3   * by Hirotsugu Kakugawa
4   *************************************************/
5  #include <xc.h>
6  #define _XTAL_FREQ 32000000 /* 32MHz */
7
8  /* コンフィグレーションビット */
9  #pragma config RSTOSC   = HFINT32
10 #pragma config FEXTOSC  = OFF
11 #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
12 #pragma config WDTE     = OFF /* WDT無効 */
13 #pragma config LVP      = OFF /* 低電圧書込不可 */
14
15 void main(void)
16 {
17   /* I/Oピン設定 */
18   TRISA = 0x1F; /* 入力: RA4,RA3,RA2,RA1,RA0 */
19   /* LED点滅 */
20   for (;;) {
21     __delay_ms(500);
22     LATA5 = 1;
23     __delay_ms(500);
24     LATA5 = 0;
25   }
26 }
```

このソースコードを簡単に説明します.

- `#include <xc.h>`
  XC8 コンパイラでの PIC マイコン用の各種定義をインクルードします. これは XC8 コンパイラ特有の

ものです.

- 記号 _XTAL_FREQ
  このマイコンが動作するクロック周波数を定義します. 関数__delay_ms() を使用するにはこの記号の定義が必須です.
- コンフィグレーションビットの定義
  マイコンを動作させるための各種の定義を行います. 詳しく述べるには膨大な説明が必要となりますので本書では割愛します. データシートを参照して下さい.
- 関数 main()
  マイコンが起動するとこの関数が呼び出される仕組みになっています. 関数 main() はその実行を終えることなく, 永遠に動き続けます.
- レジスタ TRISA の設定
  ピンに対応するレジスタのビットを 1 にすると, そのピンが入力と設定されます. 本回路では RA5 ピンは出力とし, その他は入力とします.
- メインループ
  __delay_ms(500) により 500 ミリ秒の間, 待機をします. LATA5 に書き込んだ値がポート RA5 に出力されます. 1 を書き込めば H (5V) が, 0 を書き込めば L (0V) が RA5 にそれぞれ出力されます. その結果として LED が点灯, 消灯します. このプログラム中の一連の動作で LED は 1 秒間隔で点滅します.

このプログラムコードをファームウエアとして PIC マイコンに書き込む作業のおおまかな点順は以下の通りです.

1. 統合開発環境 MPLAB X IDE で新規のプロジェクトを作成します.
2. 作製したプロジェクトでプログラムを入力します.
3. 作製したプロジェクトをビルド (コンパイル・リンク) します. コンパイルエラーがあればプログラムを修正します.
4. PICkit3/4 をパーソナルコンピュータに接続して, PIC マイコンにファームウエアを書き込みます.
5. 開発した装置に PIC を装着して動作確認をします.

一度で正しい動作が得られるとは限りません. 正しい動作になるまで, 再びプログラムの修正, ビルド, 書き込みを行います. ただし装置の組み立ての際に配線間違いの可能性もあるので, よく注意して原因を探り, 手直しを行います.

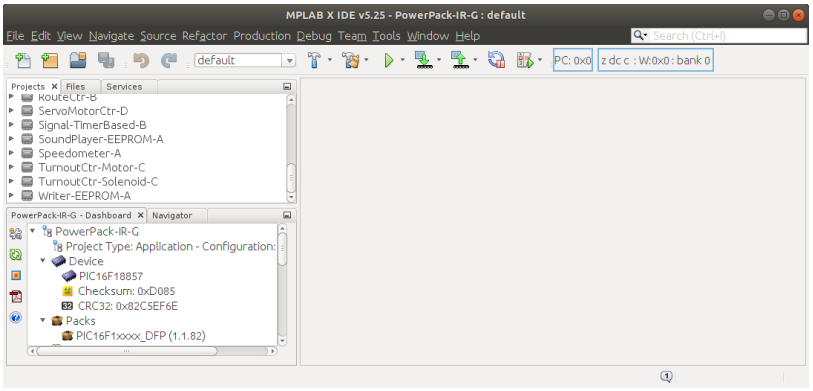
以下にそれぞれの点順を詳しく説明してゆきます.

新規のプロジェクトの作成

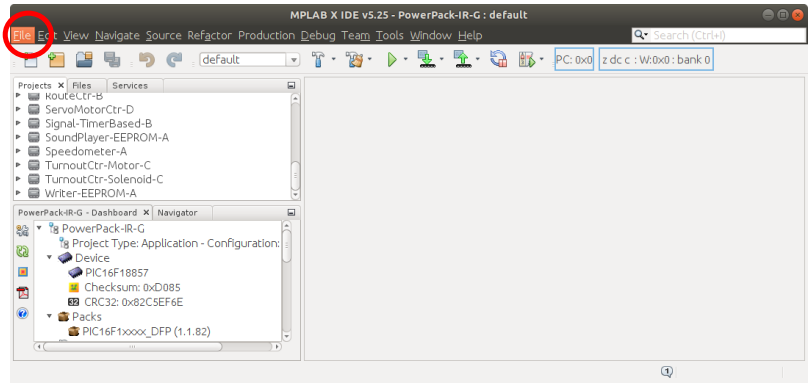
MPLAB X IDE を起動すると図 4.13a のメインウインドウが表示されます. ただし画面構成は実行環境により若干違う場合があります. この例ではすでにいくつもプロジェクトが登録されているのがウインドウ左端に見えます. 初めて起動をする場合にはまだ何もプロジェクトは登録されていません.

次にプロジェクトの新規作成をします. 図 4.13b に示すように, メインウインドウのメニュー「FILE」の中の「New Project」をクリックし, あとは表示に従ってプロジェクトの初期設定を行います.

- カテゴリ (Categories): 「Microchip Embedded」を選択.
- プロジェクト (Project): 「Standard Project」を選択.
- デバイス (Device) : 使用する PIC マイコンを選択 (例えば PIC16F18313)



(a) MPLAB X IDE 起動時のメインウインドウ



(b) FILE メニューの中の「New Project」で新規プロジェクト作成

図 4.13: MPLAB X IDE での新規プロジェクト作成

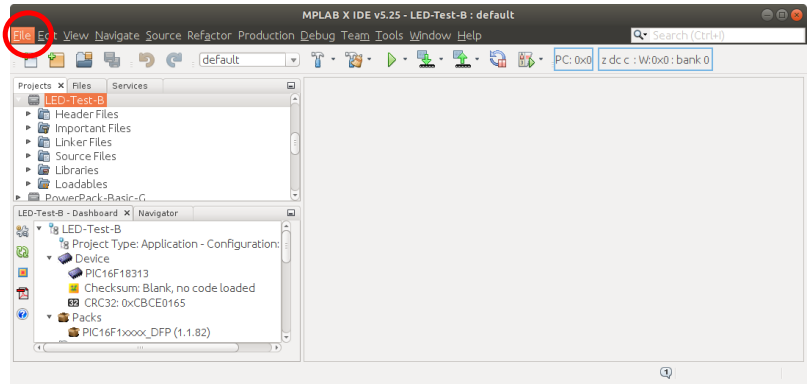
- ツール (Tool) : 使用する書き込みを選択 (例えば PICkit4)
- コンパイラ (Compiler Toolchains) : 「XC8」の最新版を選択.
- プロジェクト名 (Project Name): 適切なプロジェクト名を入力
- プロジェクト位置 (Project Location): プロジェクトファイルを作成するフォルダを選択
- エンコーディング (Encoding): ソースファイルの文字コードを選択. Windows の場合には「Shift_JIS」を, Linux の場合には「UTF-8」を選択すると良い.

プログラムの入力

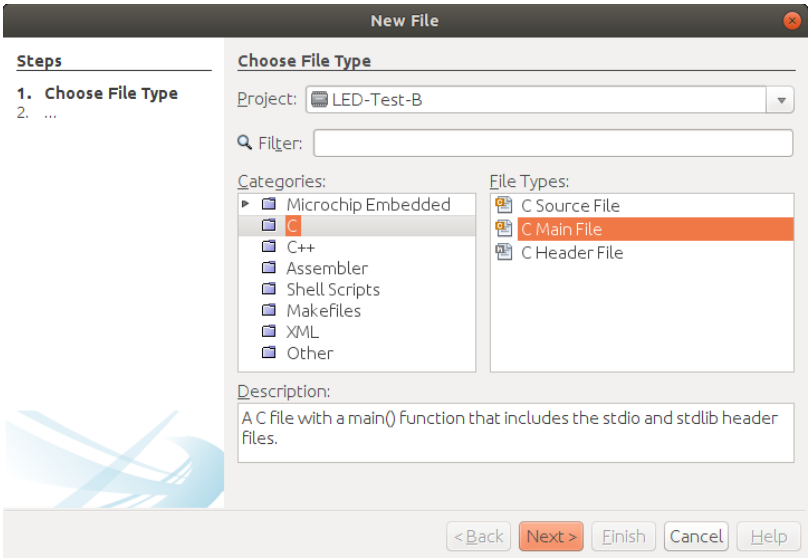
プロジェクト中に C 言語ソースファイル `LED-Test.c` を新規作成します. 図 4.14a に示すように, メインウインドウのメニュー「FILE」の中の「New File」をクリックします. 新規作成する C 言語でのファイルは, main 関数を含むソースファイル (C Main File), それ以外のソースファイル (C Source File), ヘッダファイル (C Header File) の 3 種類から選びます. いまの例では main 関数を含むソースファイルですので, 図 4.14b に示す通り「C Main File」を選択します. そして図 4.14c に示す通り, 新規ファイルのファイル名を指定します.

以上を終えて「Finish」ボタンを押すと, 内容が空のファイルが作成されます. あとはそのソースファイルに上記で示した `LED-Test.c` の内容を図 4.14d のように入力します.

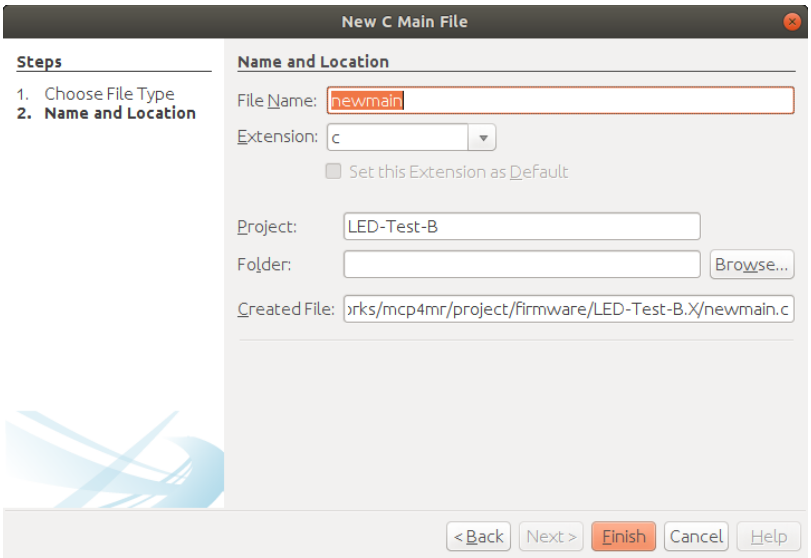
本章のファームウエアは 1 つのソースファイルだけで構成されていますが, 後の章のファームウエアは複数のソースファイルで構成されています. プロジェクトにソースファイルを追加する際は, 上で説明したようにメ



(a) FILE メニューの中の「New File」で新規ファイルの作成を開始

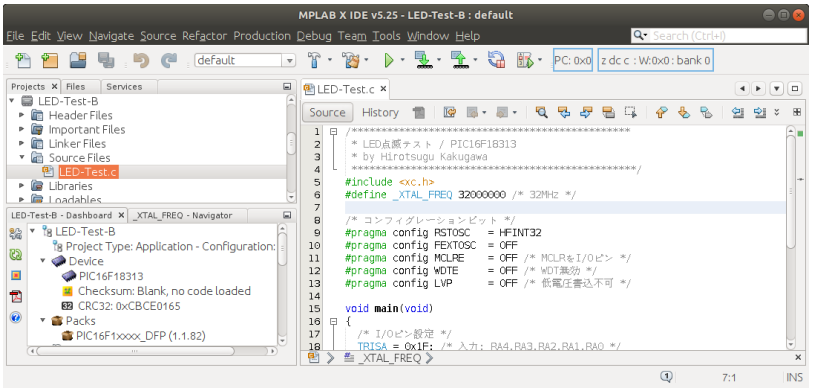


(b) 作成する新規ファイルとして C 言語 main ファイルを選択



(c) 作成する新規ファイルのファイル名を入力 (「newmain」の部分を変更)

図 4.14: MPLAB X IDE でのプロジェクトへのプログラム入力 (1/2)



(d) ファイルにプログラム内容を入力

図 4.14: MPLAB X IDE でのプロジェクトへのプログラム入力 (2/2)

インウインドウのメニュー「FILE」の中の「New File」をクリックして, 新規ファイルを追加してゆきます.

プロジェクトのビルド

マイコンにファームウエアを書き込むには, 入力したソースファイルをビルドする必要があります. ここでいうビルドとは, ソースファイルから PIC マイコンのネイティブコード (機械語) にコンパイル・リンクして, インテル HEX 形式のファイルに変換する作業を言います. ボタンをマウスでクリックすれば MPLAB X IDE が一連の作業を自動的に行うので, コンパイル手順・ネイティブコード・インテル HEX 形式のファイルのことなどを気にすることはまったく不要です.

図 4.15a に示す「Build Project」ボタンをクリックすると, ビルドが実行され, 図 4.15b に示すように「BUILD SUCCESSFUL」の表示が出ればビルド成功です. もしエラーがあればソースファイルを修正します.

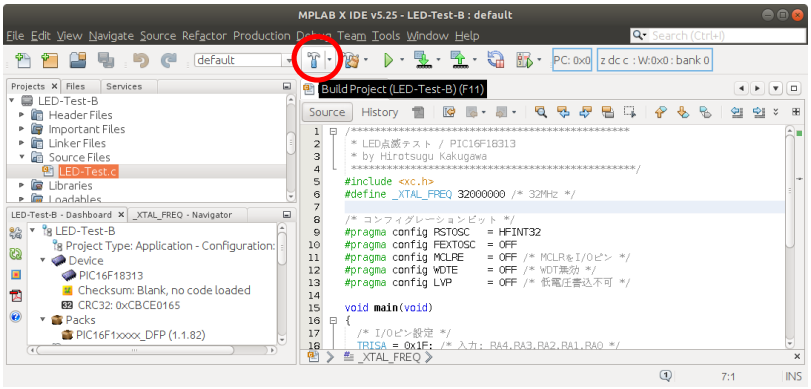
PIC マイコンへのファームウエアの書き込み

PIC マイコンにファームウエアを書き込む手順は以下の通りです. 以下では PICkit4 を使用した場合を例にして説明します.

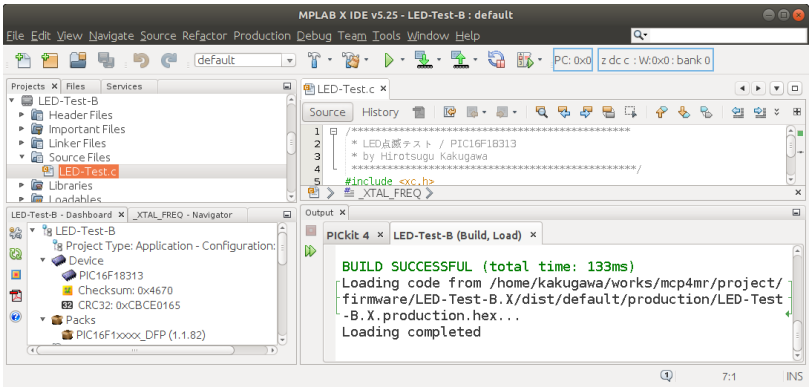
1. PICkit4 とパーソナルコンピュータを USB ケーブルを介して接続します.
2. PICkit4 に書き込みアダプタを接続します.
3. PIC マイコンを書き込みアダプタに載せて, ゼロプレッシャソケットのレバーを操作して PIC マイコンを固定します.
4. 書き込みの際に PICkit4 が PIC マイコンに 5.0V を供給する設定をします. これは, メニュー「FILE」→「Project Properties」→ Categories 内 Conf:「PICkit4」→ Option categories: 「Power」を選択します. そして図 4.15c に示す通り, チェックを入れて Voltage Level に 5.0 を選択します.
5.「Make and Porgram Device」ボタン (図 4.15d) をクリックすると PIC マイコンへファームウエアの書き込みを行います.
6. 書き込みに成功すれば, 図 4.15e に示すように,「Programming/Verify Complete」と表示されます.

書き込みに失敗する場合の原因の可能性には以下のものが考えられます.

- パーソナルコンピュータが PICkit4 を認識しない. 接続や USB デバイスの設定を確認します. MPLAB X IDE インストール直後の場合にはパーソナルコンピュータの再起動により解決するかも知れません.



(a)「Build Project」ボタンをクリックしてビルド開始



(b) ビルド完了 (「BUILD SUCCESSFULL」の表示に注意)

図 4.15: MPLAB X IDE でのビルド (コンパイル)

- 書き込みアダプタの配線設定が間違えている. 書き込みの際に使用する PIC マイコンのピンに正しく結線されているか確認します.
- 書き込みアダプタに載せる PIC マイコンが間違えている. ファームウエアで指定している PIC マイコンと同じがどうか確認します.

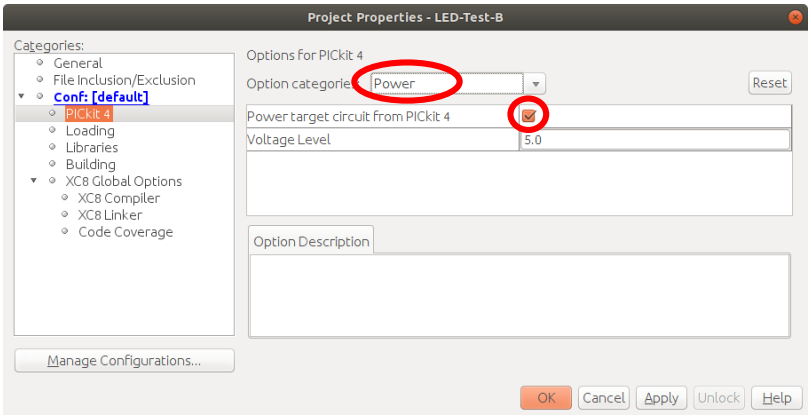
PIC マイコンを装置に装着して動作を確認

最後に動作確認を行います. PIC マイコンをゼロプレッシャソケットから外して装置に装着し, AC アダプタを接続して電源供給を行い, LED がチカチカ点滅すれば成功です.

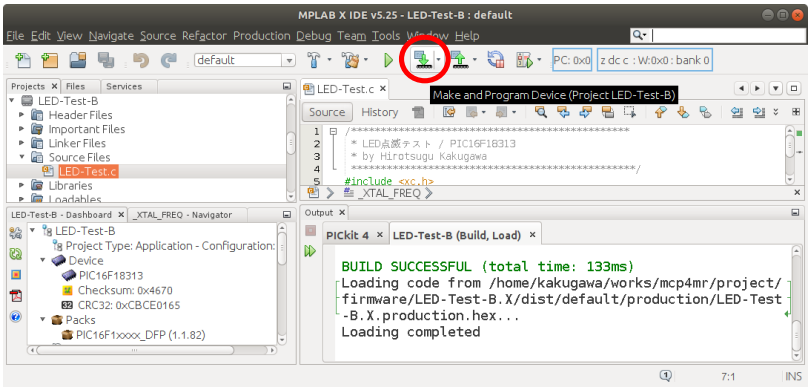
動作しない場合, 可能性として以下のものが考えられますので修正を行います.

- ハードウエア: 配線の間違い
- ソフトウエア: ソースファイルの入力の間違い

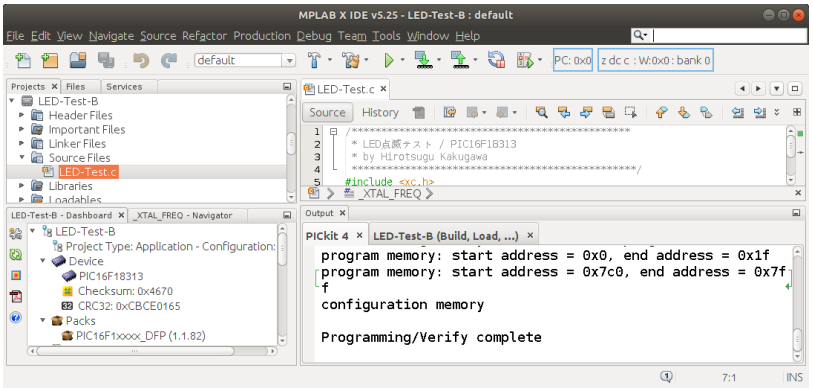
作業の手順は以上ですが, 細かなことはたくさんあってすべて書ききれません. MPLAB X IDE の操作方法や PICkit3/4 の使用方法は, マニュアル, 各種参考書籍, あるいは紹介ウェブページなどを参考に各自で習得して下さい.



(c) PICkit4 から PIC マイコンに 5.0V を供給する設定



(d)「Make and Program Device」ボタンをクリックして PIC へ書き込み開始



(e) ファームウエア書き込み完了

図 4.15: MPLAB X IDE での PIC マイコンへのファームウエアの書き込み

4.5　MPLAB X IPE を用いたインテル HEX 形式の配布ファームウエアの書き込み

本書のファームウエアをすべて手作業での入力やビルドは面倒ですから, 読者への便宜を図ってインテル HEX 形式のファイルでファームウエアを配布する予定です. 以下ではインテル HEX 形式のファームウエアを PIC マイコンに書き込む方法を説明します. ソースファイルの入力やビルドなどは行いませんから, 統合開発環境である MPLAB X IDE の代わりに統合プログラミング環境 MPLAB X IPE を使用します. なお「プログラミング (programming)」とは日本語での「書き込み」を意味する英語表現であり, 開発作業のことではありませんので注意して下さい.

書き込みの作業手順は大まかに以下の通りです.

1. MPLAB X IPE の起動と設定.
2. PIC マイコンにファームウエア HEX ファイルを書き込みます.
3. 開発した装置に PIC を装着して動作確認をします.

以下ではそれぞれの手順を PICkit4 を使用した場合を例にして説明します.

MPLAB X IPE の起動と設定

MPLAB X IPE の起動に先立ち, 以下のように PICkit4 を接続します.

1. PICkit4 とパーソナルコンピュータを USB ケーブルを介して接続します.
2. PICkit4 に書き込みアダプタを接続します.
3. PIC マイコンを書き込みアダプタに載せて, ゼロプレッシャソケットのレバーを操作して PIC マイコンを固定します.

次に MPLAB X IPE を起動します.

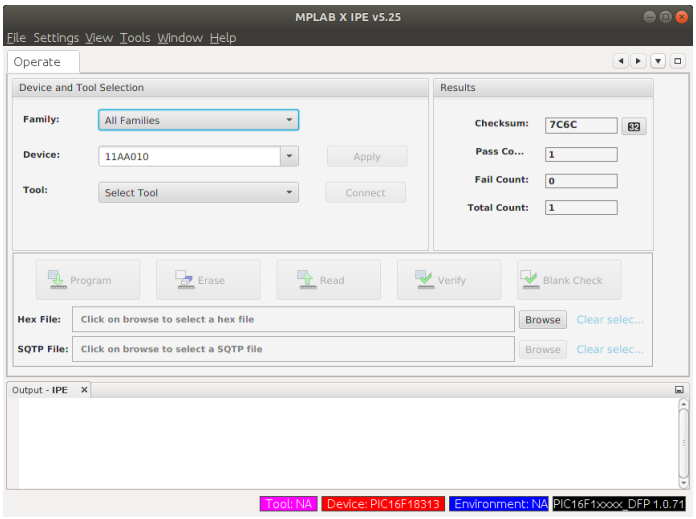
1. 起動後の画面は図 4.16a に示す通りで, 必要な設定を行います. メニュー「Settings」→「Advenced Mode」より, 詳細な設定画面へ移ります. この際にパスワードの入力が要求される場合があり戸惑いますが, 画面をよく観れば親切にも microchip だと教えてくれていますので安心して下さい.
2. PIC マイコンへの電源供給の設定をします. 左の「Power」オプションをクリックし, 図 4.16b に示す通り, 電源供給にチェックを入れ, Voltage Level を 5.0 に設定します.
3. 書き込みを行う PIC マイコンを選択します. 左の「Operate」オプションをクリックし, 図 4.16c に示す通りに設定します.
   - デバイス (Device) : 使用する PIC マイコンを選択 (例えば PIC16F18313)
   - ツール (Tool) : 使用する書き込みを選択 (例えば PICkit4)

   選択を終えたら,「Apply」ボタン,「Connect」ボタンの順でクリックします. PICkit4 および PIC マイコンと正常に通信できれば, 図 4.16d に示す通り,「Target device PIC16F18313 found」と表示されます.

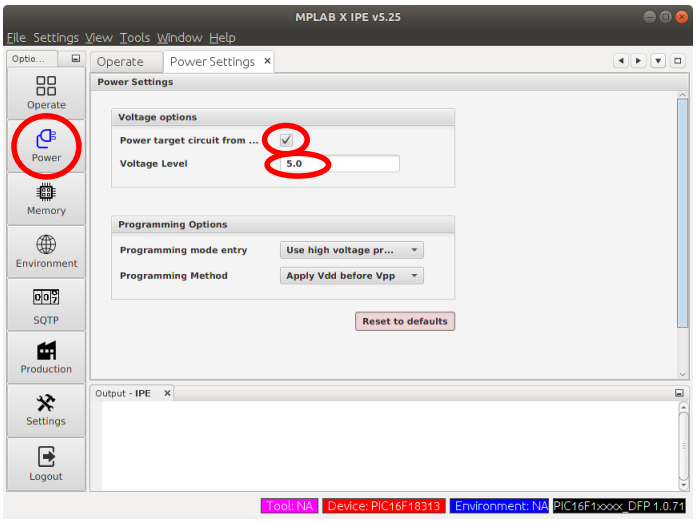
ファームウエア HEX ファイルの書き込み

次は実際に PIC マイコンにファームウエアを書き込みます. 手順は以下の通りです.

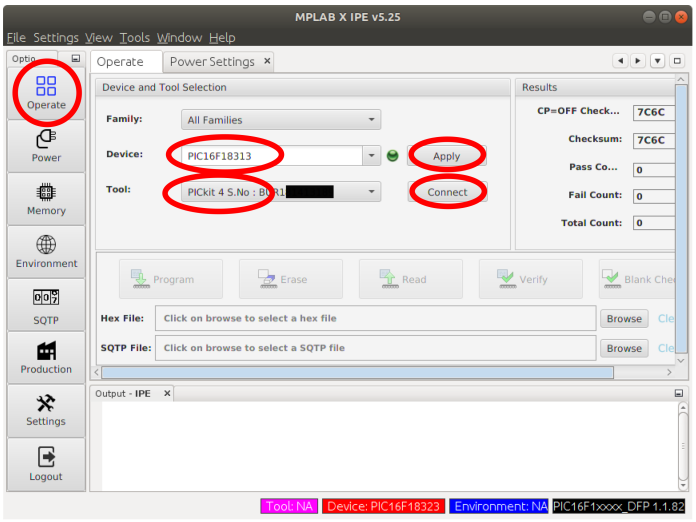
1. 書き込むファームウエア HEX ファイルを指定します. 図 4.16e に示す通り,「HEX File」の「Browse」ボタンをクリックしてファイルを選択します. 次に「Program」ボタンをクリックして PIC マイコンに書き込みます.
2. 書き込みに成功すれば図 4.16f に示す通り「Programming complete」と表示されます.



(a) MPLAB X IPE 起動後の画面

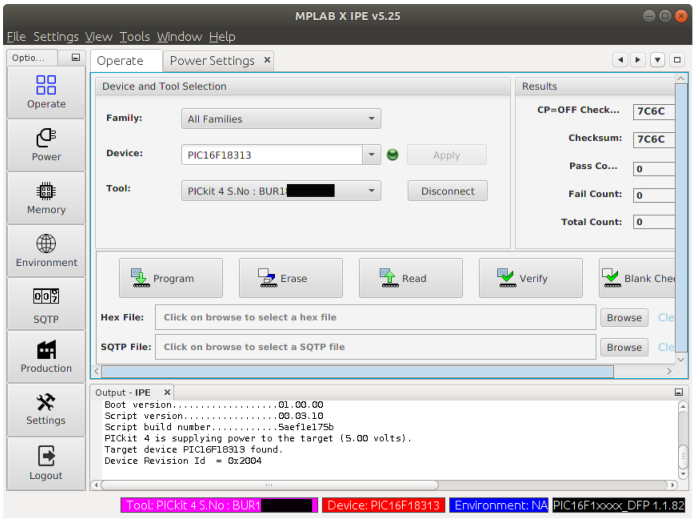


(b) PICkit4 から PIC マイコンへ電源を供給する設定

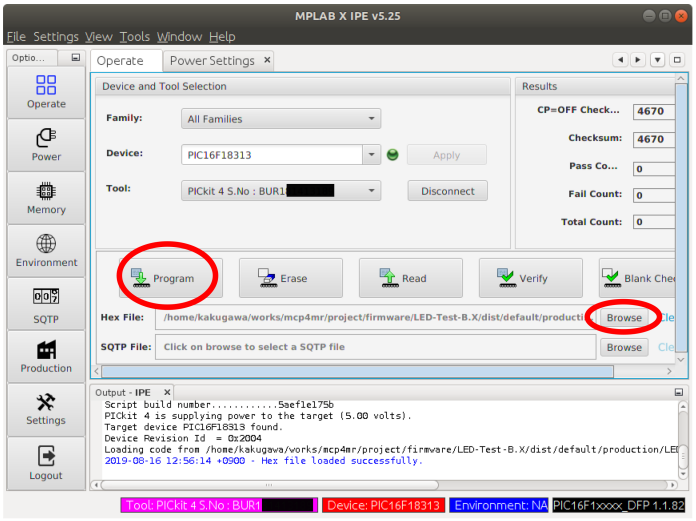


(c) PIC マイコンの選択

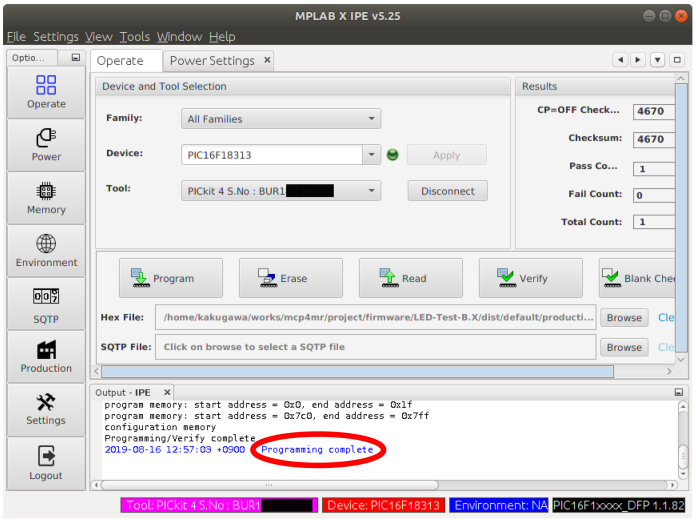
図 4.16: MPLAB X IPE の起動/設定/書き込み (1/2)



(d) PICkit4 と PIC マイコンとの接続の完了



(e) ファームウエア HEX ファイルの選択と書き込み



(f) 書き込みの完了

図 4.16: MPLAB X IPE の起動/設定/書き込み (2/2)

書き込みに失敗するときの主な原因は MPLAB X IDE の場合と同様ですので, 先述の説明を参考にして問題を解決して下さい.

装置の動作確認

最後に動作確認を行います. PIC マイコンをゼロプレッシャソケットから外して装置に装着し, AC アダプタを接続して電源供給を行い, LED がチカチカ点滅すれば成功です.

課題

1. LED の点滅間隔を 0.2 秒にして下さい.
2. 本装置では 1 つの LED を接続していますが, 2 つの LED を接続した回路とファームウエアの設計・製作を行って下さい. LED の点滅パターンは踏切警報灯の点滅に合わせて下さい.

第 5 章

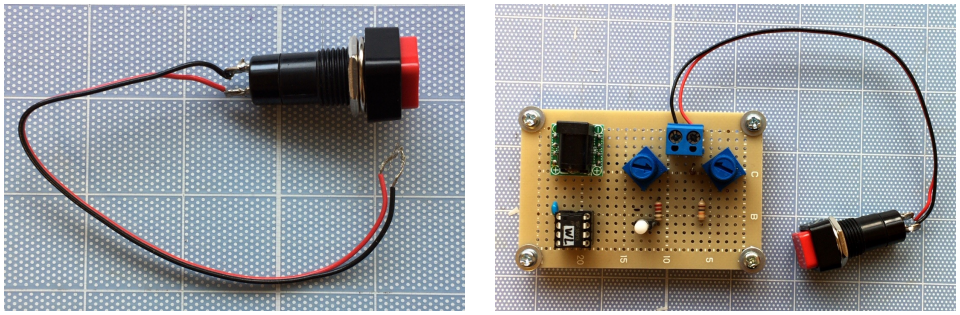
マイコンプログラミングの基礎学習

本章では第 4 章で作成した LED 点灯装置を使用して, マイコンプログラミングの基礎学習を行います. ここで学ぶ内容は以下の通りです. 既に PIC マイコンのファームウエアプログラミングに慣れている方は, 本章を読み飛ばして構いません.

- PIC マイコンの内蔵周辺モジュール
- PWM 制御による LED 輝度制御
- 状態遷移手法によるソフトウエア設計技法
- 複数のソースファイルで構成されるファームウエア開発方法

特に以下の PIC マイコンの内蔵周辺モジュールをここで学びます. これらは本書全体を通して使用する基本的な知識ですので, 十分な理解に努めて下さい.

- PIC マイコンのデジタル I/O ポートモジュール PORT
  スイッチのオン・オフの読み込み (デジタル入力) や, LED の点灯 (デジタル出力) に使います.
- PIC マイコンの AD 変換モジュール ADC
  パワーパックの速度ボリュームの読み込み (アナログ入力) などに使います.
- PIC マイコンのタイマモジュール TMR2/4/6
  時間の測定やモータの PWM 制御での PWM 周波数の設定に使用します.
- PIC マイコンのキャプチャ/比較/PWM モジュール CCP
  モータの回転速度や LED の輝度をなめらかに変化させる PWM 制御などで使います.
- PIC マイコンのペリフェラルピンセレクトモジュール PPS
  PIC マイコンにおける, 内部の内蔵周辺モジュールと外部のピンとの間の割付を設定する仕組みで, 限られたピンの有効利用のために使用します.



(a) 外部制御用スイッチ　　**(b)** 本装置とスイッチの接続

図 5.1: LED 点灯装置

なお内蔵周辺モジュールの機能すべてを詳細に紹介することはせず, 本書の装置での利用目的の観点で必要となる機能に限って紹介をします. 具体的な使用方法を想定しながら読めば理解は早いと思います. 他の機能など深く知りたい方は, 詳細を解説した書籍がいくつも出版されていますので, そちらを参照して下さい.

本章ではスイッチにより LED 点灯装置の点灯のオン・オフを制御するファームウエアを紹介します. このファームウエアが持つ機能は以下の通りです.

- 外部接続のスイッチをオンにすると LED が点灯し, オフにすると消灯する.
- 可変抵抗器 PARAM を回転すると LED の輝度が変化する.
- LED の輝度は PWM 制御で実現する.

実に当たり前の機能に見えますが, これらは全てソフトウエアで実現していることが重要です. スイッチの状態や可変抵抗器のつまみの位置をマイコンで読みとり, LED の輝度をマイコンが PWM 制御します. この仕組みは, PWM 式パワーパックの理解の基本となるものです. なぜなら, この PWM 制御の対象を LED からモータに変更すれば PWM 式パワーパックになるからです. 従って, 本章のプログラムを当たり前でつまらないものと決して思わずに, しっかりと勉強して下さい.

本章ではスイッチを本装置のターミナルに接続して点灯のオン・オフを制御可能にします. なおここでいうターミナルとは, 図 4.2 でのターミナル ON (J1) を指します. 図 5.1 に示すようにスイッチと接続用の電線 2 本を用意して下さい. スイッチの 2 つ端子に電線をハンダ付けし, ターミナルの 2 つの端子 X と GND に 2 本の電線をそれぞれ接続して下さい.

5.1 PORT: デジタル I/O ポートモジュール

PIC マイコンでは内蔵周辺モジュールデジタル I/O ポート PORT を使用してデジタル信号の入出力を行います. I/O ポートは 1 ビットの入出力で, 複数の I/O ポートが PIC マイコンには装備されています. 動作モードには入力モードと出力モードの 2 通りがあり, ファームウエアからモードの選択が可能です. なおほとんどの I/O ポートは入力モードと出力モードのどちらにでも設定が可能ですが, 入力モードに限定されているものもありますので注意して下さい.

- 入力モード
  1 ビットのデジタル信号を入力します. マイコンの I/O ピンに印加されている電圧の L または H レベルを論理値の 0 または 1 として読み取ります.
- 出力モード
  1 ビットのデジタル信号を出力します. 論理値の 0 または 1 を書き込むことで, マイコンの I/O ピンに L または H の電圧を出力します.

I/O ポートはそれぞれ, A0, A1, ... B0, B1, ... という名称で区別します. I/O ポートは 8 本の単位でグループ化されており, PORTA, PORTB, PORTC, ... という名称でグループを区別し, それぞれに対応する 8 ビットのレジスタを用いて各種の設定を行います.

I/O ポートには詳細な動作モードを設定する各種レジスタを説明します. 以下での **x** は I/O ポートのグループを表しており, 具体的には PORTA の場合には `A`, PORTB の場合には `B` となります.

- `PORTx`
  ピンの電圧レベルの読み取りを行うレジスタです. ピンの電圧レベルが H の場合は 1, L の場合は 0 となります. PORTx レジスタ内のビットとピンの対応関係は, 例えば PORTA の 0 ビット目はピン A0, 1

ビット目はピン A1, ... となります. また, 記号 RA0 は PORTA の 0 ビット目, RA1 は PORTA の 1 ビット目, ... が定義されています.

- TRISx
  ピンの入出力モードを設定するレジスタです. 入力モードの場合は 1, 出力モードの場合は 0 に設定します. TRISx レジスタ内のビットとピンの対応関係は, 例えば TRISA の 0 ビット目はピン A0, 1 ビット目はピン A1, ... となります. また記号 TRISA0 は TRISA の 0 ビット目, TRISA1 は TRISA の 1 ビット目, ... が定義されています. 入力モード専用の I/O ポートでは設定できませんので注意して下さい. PIC マイコンによっては RA3 が入力モード専用の場合があります.
- LATx
  データラッチレジスタです. 値を書き込むとその値に対応する電圧レベルがピンに出力されます. また書き込んだ値を保持し, 後から値の読み出しが可能です. 入力モード専用の I/O ポートでは書き込みができませんので注意して下さい. LATx レジスタ内のビットとピンの対応関係は, 例えば LATA の 0 ビット目はピン A0, 1 ビット目はピン A1, ... となります.
- ANSELx
  アナログモード・デジタルモードの選択を設定するレジスタです. アナログモードの場合は 1, デジタルモードの場合は 0 に設定します. AD 変換やコンパレータなどアナログ電圧を入力とする内蔵周辺モジュールにピンの電圧を供給する場合はアナログモードに設定します. デジタル I/O の場合はデジタルモードに設定します. ANSELx レジスタ内のビットとピンの対応関係は, 例えば ANSELA の 0 ビット目はピン A0, 1 ビット目はピン A1, ... となります.
- WPUx
  弱プルアップの設定をするレジスタです. ピンのプルアップを有効にする場合は 1, 無効にする場合は 0 に設定します. WPUx レジスタ内のビットとピンの対応関係は, 例えば WPUA の 0 ビット目はピン A0, 1 ビット目はピン A1, ... となります.
- ODCONx
  オープンドレインモードの設定をするレジスタです. ODCONx レジスタ内のビットとピンの対応関係は, 例えば ODCONA の 0 ビット目はピン A0, 1 ビット目はピン A1, ... となります. 入力モード専用の I/O ポートでは設定できませんので注意して下さい.

5.2　ADC: AD 変換モジュール

PIC マイコンでは内蔵周辺モジュール ADC (Analog-Digital Converter; アナログ・デジタル変換器) を使用してアナログ電圧値をデジタル値に変換を行います. 図 5.2 に 16F18313 の ADC の内部構造の概略を示します. 本節では 16F18313 の ADC の概説します. PIC マイコンによって機能は大きく異なり, ファームウエアも若干異なりますから, 詳しくはデータシートを参照して下さい.

ADC はアナログ電圧値をデジタル値に変換するモジュールで, PIC マイコンには 10 ビットの精度を有する ADC が装備されています. アナログ電圧の入力源にはマイコン外部の電圧を入力するためのピンや内部モジュールの電圧出力など複数の入力チャンネルが用意されており, それらの中からひとつを選択して AD 変換の対象とします.

ADC の使用方法の概略は以下の通りです.

1. マイコンのピンをアナログ入力モードに設定します. これはマイコン外部の電圧を入力として AD 変換する場合です.

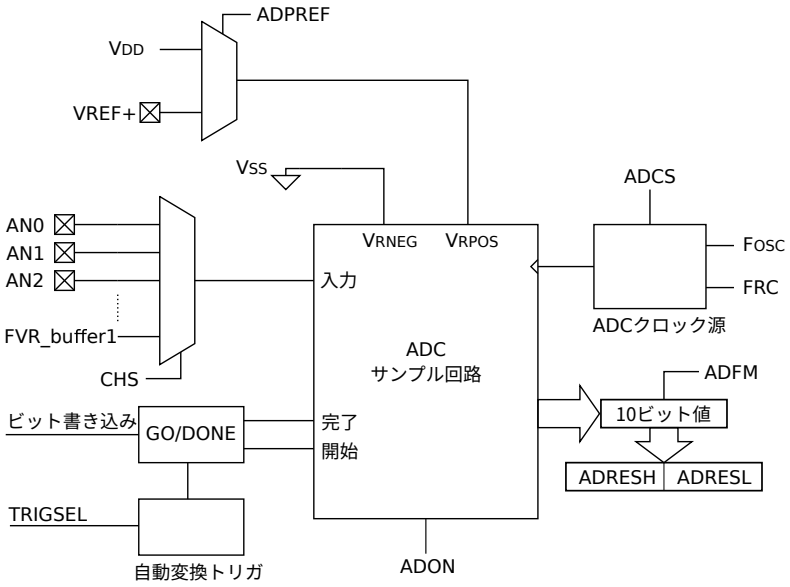


図 5.2: ADC の内部構造の概略 (16F18313)

2. ADC クロック源を選択します. その他, 基準電圧や結果フォーマットなどの設定を行います.
3. ADC をオンにします.
4. 以下を繰り返して随時 AD 変換を行い結果を得ます.
   （a）入力チャンネルを選択します.
   （b）ADC 内部にアナログ電圧値の取り込みが完了するまで待機します.
   （c）AD 変換を開始し, 終了を待機します.
   （d）AD 変換の結果を読み出します.

ADC で使用する主なレジスタやビットを以下に概説します.

- `ADCON0.CHS`
  入力チャンネルを指定します. マイコンのピン, FVR (固定参照電圧モジュール), DAC (デジタルアナログ変換モジュール) などの中からひとつを選択します.
- `ADCON0.GO_nDONE`
  変換状況を表すビットです. 1 の場合は変換途中で, 0 の場合は変換完了です. このビットに 1 を書き込めば変換が開始されます.
- `ADCON0.ADON`
  ADC のオン・オフを選択するビットです. 1 の場合はオンで, 0 の場合はオフです.
- `ADCON1.ADFM`
  ADC の結果をビット表現を選択するビットです. 1 の場合は右寄せフォーマットで, 0 の場合は左寄せです.
- `ADCON0.ADCS`
  クロック源および分周率を指定します.
- `ADREF.ADNREF`
  負の基準電圧源を指定します.

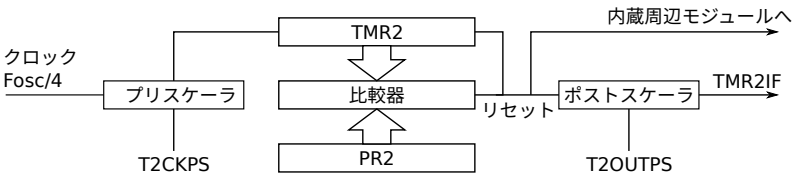


図 5.3: TMR2 の内部構造の概略

- `ADPREF`
  正の基準電圧源を指定します.
- `ADRESH`
  AD 変換の結果の上位ビットを保持するレジスタです.
- `ADRESL`
  AD 変換の結果の下位ビットを保持するレジスタです.

5.3　TMR2/4/6: 8 ビットタイマモジュール

PIC マイコンの内蔵周辺モジュール TMR2 (Timer2; タイマ 2) を使用して時間の計測を行います. 特に一定周期で事象の発生を行い, 様々な周期的動作の実現に利用します. TMR2 と同じ構造を持つタイマに TMR4, TMR6 があり, 各 PIC マイコンでの装備の有無は表 3.6 を参照して下さい. 以下では TMR2, TMR4, TMR6 を総称して TMR2 系タイマと呼ぶことにします.

図 5.3 に TMR2 の内部構造の概略を示します. TMR2 は 8 ビットのタイマで, PIC マイコン内部のクロックなどによりカウントアップを行います. TMR2 モジュールはクロックから受けた信号を分周するプリスケーラを通り, 一定周期で TMR2 レジスタのカウントアップを行います. TMR2 モジュール内部には, タイマ周期を規定する PR2 レジスタがあります. TMR2 レジスタの値と PR2 レジスタの値を比較して, もし一致すれば TMR2 レジスタの値を 0 にリセットします. つまり TMR2 レジスタの値の範囲は, 0 以上 PR2 未満に限定されます. ただし PR2 が 0 の場合には, TMR2 レジスタの値は 0 から 255 までの範囲となります.

TMR2 系タイマで使用する主なレジスタやビットを以下に概説します. なお以下での表記 TMRx は, TMR2, TMR4, TMR6 のいずれかを表すものとします.

- `TMRx`
  8 ビットのレジスタで, タイマの値を保持します.
- `PRx`
  8 ビットの周期レジスタで, 周期の値を保持します.
- `TxCON.TxOUTPS`
  出力ポストスケーラを指定します. 1:1 から 1:16 までの比率を指定可能で, 指定した比率にタイマ出力を分周します.
- `TxCON.TMRxON`
  TMRx のオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.
- `TxCON.TxOUTPS`
  プリスケーラを指定します. 1/1, 1/4, 1/16, 1/64 の比率を指定可能で, 指定した比率にクロック周波数を分周します.
- `PIR1.TMR2IF`

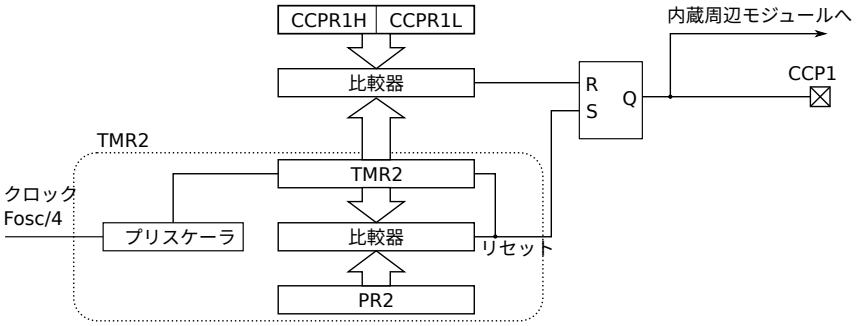


図 5.4: CCP1 の PWM モードにおける PWM 波形出力

割り込み発生フラグです. オーバーフローが発生してタイマの値が 0 に戻ったときにこのビットが 1 になります. 他の TMR2 系タイマでも同等なフラグが装備されています. 0 を書き込むことで割り込み発生フラグはクリアできます.

- `PIE1.TMR2IE`
  TMR2 割り込み許可ビットです. 他の TMR2 系タイマでも同等なビットが装備されています.

5.4 CCP: キャプチャ/比較/PWM モジュール

PIC マイコンの内蔵周辺モジュール CCP (Capture/Compare/PWM; キャプチャ/比較/PWM) は, 外部信号をトリガにしたタイマ値を記憶するキャプチャモード, タイマの値が指定の値に一致すれば信号を発生するコンペアモード, 指定したデューティ比の PWM 波の信号を発生する PWM モードの 3 通りの使用ができるモジュールです. 以下では PWM 波の発生を行う PWM モードの使用方法に限って説明します. また CCP モジュールのひとつである CCP1 の場合について説明します.

PWM モードでは図 5.4 に示すように CCP1 と TMR2 を組み合わせて PWM 波を発生します. CCP1 モジュールでは PWM デューティ比を規定するために CCPR1H レジスタと CCPR1L レジスタの 2 つを使用します. これら 2 つのレジスタはそれぞれ 8 ビットですが, 2 つを連結して 16 ビットレジスタとして使用します. ただし実際には CCPR1H レジスタは 2 ビットだけを使用して, 合計 10 ビットしか使用しません. すなわちデューティ比は 0 から 1023 の整数値で指定します. なお以下では, 表記 CCPR1 により CCPR1H レジスタと CCPR1L レジスタの連結を表すものとします.

CCP1 モジュールの PWM モードは以下の動作により PWM 波の出力を実現します. TMR2 レジスタ, PR2 レジスタ, CCPR1 レジスタ, PWM 出力の 4 者の関係を図 5.5 に示します.

- TMR2 の値はクロックに従って増加します.
- CCP1 の出力は以下の通りに決定されます.
  - TMR2 ≤ CCPR1 の場合: 1 を出力
  - TMR2 > CCPR1 の場合: 0 を出力

  つまり PWM 波のデューティ比は CCPR1 で決まります.
- TMR2 の値が PR2 に一致する TMR2 の値は 0 にリセットされます.
- PWM 波の 1 周期の長さはクロック周波数, プリスケーラ, PR2 の 3 者で決まります.

CCP1 で使用する主なレジスタやビットを以下に概説します. なお以下での表記 CCPx は, CCP1, CCP2, ... のいずれかを表すものとします.

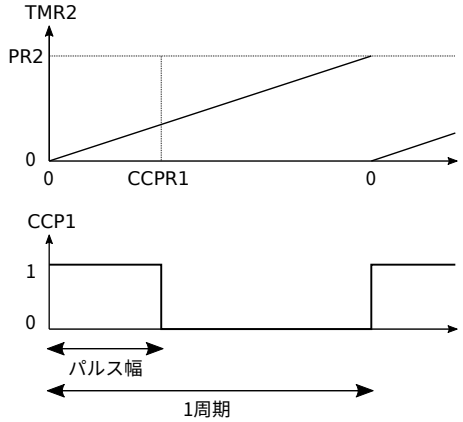


図 5.5: PWM 波の生成 (1 周期はクロック周波数と PR2 で決まり, デューティ比は CCPR1 で決まる)

- `CCPxCON.CCPxEN`
  CCP のオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.
- `CCPxCON.CCPxFMT`
  パルス幅フォーマット設定ビットで, PWM モードのときのみ有効です. 1 の場合は右寄せフォーマットで, 0 の場合は左寄せフォーマットです. `CCPRxH` と `CCPRxL` を連結した 16 ビットのうち, 右側 10 ビットと左側 10 ビットのいずれをデューティ比を指定する 10 ビットとして使用するかを定めます.
- `CCPxCON.CCPxMODE`
  CCP モジュールの動作モードを設定します.
- `CCPRxH`, `CCPRxL`
  PWM モード時にデューティ比を指定します. `CCPRxH` が上位ビットで, `CCPRxL` が下位ビットです.

PWM 周波数の計算方法を説明します. 以下の設定をした場合を考えます.

- クロック周波数 Fosc: 32MHz
  この 1/4 の 8MHz が TMR2 に供給されます.
- TMR2 プレスケーラ: 1/1
  周波数 8MHz で TMR2 はカウントアップします.
- PR2 レジスタ: 0xFF (10 進数では 255)
  TMR2 レジスタの周期が最大となります.

この場合での PWM 周波数は以下の計算式により 31.25kHz と求められます.

$$
\begin{aligned}
F_{PWM} &= (F_{OSC}/4)/(PR2+1) \\
&= (32\mathrm{MHz}/4)/(255+1) \\
&= 8000000/256\ \mathrm{Hz} \\
&= 31250\mathrm{Hz} \\
&= 31.25\mathrm{kHz}
\end{aligned}
$$

5.5　PPS: ペリフェラルピンセレクトモジュール

PIC マイコンでは PPS (Peripheral Pin Select; ペリフェラルピンセレクト) モジュールを使用してマイコンデバイスのピンと内蔵周辺モジュールのピンの間の結合関係を決定します. 図 5.6 に PPS モジュールの内部構造

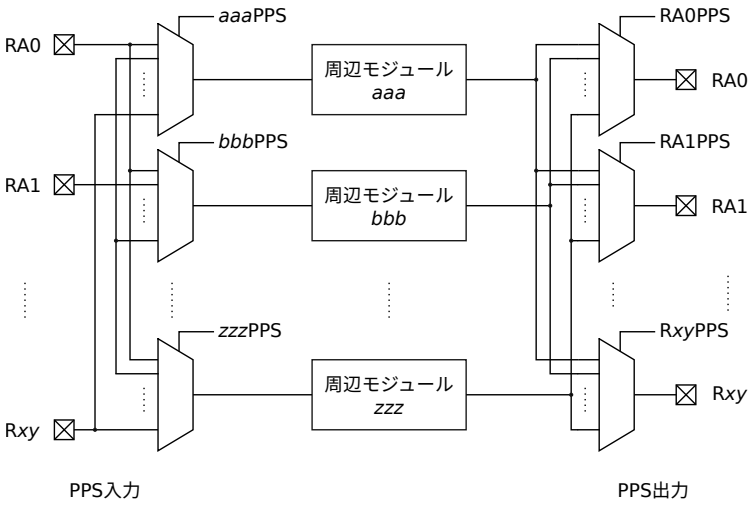


図 5.6: PPS の内部構造の概略

の概略を示します. 本書で紹介する PIC マイコンはいずれも PPS モジュールを有していますが, PPS モジュールを有していない PIC マイコンもありますので詳しくはデータシートを参照して下さい.

PPS モジュールはマイコンデバイスのピンと内蔵周辺モジュールのピンとの結合関係を設定するモジュールで, ピン数の少ないマイコンデバイスでもピンを有効に使用可能になります. なおマイコンによっては設定可能な組み合わせが限定されていますので, 詳しくはデータシートを参照して下さい.

PPS に関連するレジスタで主要なものを以下に概説します.

- xxxPPS
  内蔵周辺モジュールの入力ピン xxx とマイコンデバイスのピンの結合を設定するレジスタです. 例えば, 内蔵周辺モジュールの入力ピン INT に対するレジスタは INTPPS となります.
- RxyPPS
  内蔵周辺モジュールの出力ピン xy とマイコンデバイスのピンの結合を設定するレジスタです. 例えば, マイコンデバイスのピン RA5 に対するレジスタは RA5PPS です. なお周辺モジュールのピンには入力と出力の双方向のものがあります. その場合には入力と出力の両方の設定が必要になりますので注意して下さい.
- PPSLOCK
  PPS 設定の変更禁止の設定を行うレジスタです. ソフトウエアのバグにより, もし PPS 設定が意図しない値に変更されると異常なハードウエア制御がされて重大な事故に繋がります. 一定のアクセス手順を踏まないと PPS 設定の変更ができない機構をこのレジスタは提供します.
- PPSLOCKED
  レジスタ PPSLOCK 中のビットで, PPS 設定の変更禁止の設定を行うビットです. このビットのクリアにより PPS 設定を変更可能にし, セットにより変更禁止にします.
- PPS1WAY
  PPS 設定の変更回数を 1 度だけに限定するためのコンフィギュレーションビットです. このビットが 0 の場合, 何度でも PPSLOCKED ビットのクリアとセットが行えて, 何度でも PPS 設定を行えます. このビットが 1 の場合, マイコンのリセット後は PPSLOCKED ビットのクリアとセットは一度だけしか行えません. つまりリセット後にひとたび正常に PPS 設定を行えば, その後はどのように PPSLOCK や

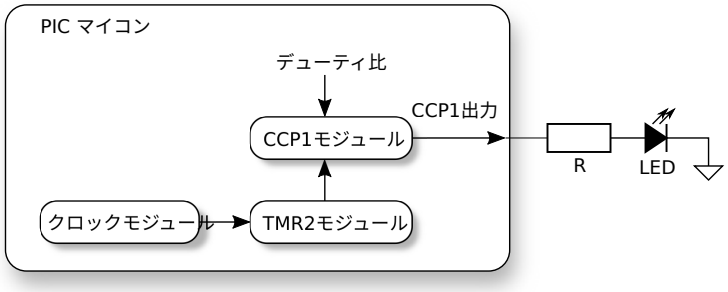


図 5.7: PWM 制御による LED 点灯

PPSLOCKED に値を書き込んでも PPS 設定は変更できません. バグにより異常な PPS 設定となることをこの機構で防ぎます.

PPS 設定の変更を行うためのアクセス手順を以下に説明します. 以下がプログラムコードの骨子です.

```
1   int8_t ogie = GIE;
2   GIE = 0;
3   PPSLOCK = 0x55;
4   PPSLOCK = 0xAA;
5   PPSLOCKbits.PPSLOCKED = 0;
6     ...
7     ... PPS設定
8     ...
9   PPSLOCK = 0x55;
10  PPSLOCK = 0xAA;
11  PPSLOCKbits.PPSLOCKED = 1;
12  GIE = ogie;
```

PPSLOCK に値 0x55 と 0xAA をこの順番で書き込むことにより, PPSLOCKED に値の書き込みが可能になる内部構造になっています. ビット PPSLOCKED をクリアすることで PPS 設定が可能な状態に設定できます. PPS 設定が終わった後は, PPSLOCK に値 0x55 と 0xAA をこの順番で書き込み, ビット PPSLOCKED をセットして, PPS 設定は不可能な状態に設定します. 以上の一連の動作を行う間は割り込み禁止として, 割り込みハンドラが呼び出されないようにします. なぜなら割り込みハンドラ中では内蔵周辺モジュールの操作が行われる可能性があり, 誤動作の原因になり得るからです.

5.6　PIC マイコンにおける PWM 制御

PIC マイコンでの PWM 制御による LED 点灯の仕組みを説明します. 図 5.7 に示す通り, クロック, タイマモジュール, CCP モジュールの 3 つを組み合わせることで実現しています. 本章のファームウエアでは TMR2 と CCP1 を使用していることから, 以下では分かりやすさのために TMR2 と CCP1 を具体例として説明に使用します. PIC マイコンはタイマモジュールや CCP モジュールを複数内蔵しており, PWM 制御の実現は TMR2 と CCP1 に限られてはいなことに注意して下さい.

- TMR2 モジュール
  クロックモジュールよりクロック信号を受けて, その内部のカウンタを 1 つづカウントアップしていきます.
- CCP1 モジュール

TMR2 のカウンタの値を参照して PWM 出力のオン・オフをマイコン外部に出力します. デューティ比の設定によりオン・オフの時間の比率が決まります.

- LED
  PWM 出力に応じて点灯と消灯を行います. PWM 出力がオンの場合はマイコンの出力ピンは 5V になり, LED に電流が流れて点灯します. PWM 出力がオフの場合はマイコンの出力ピンは 0V になり, LED に電流は流れず消灯します.

LED のオン・オフの繰り返しを電光石火の早業で行うと, LED が点滅していると人間は気づくことなくデューティ比に応じた輝度で LED が光っているように見えます. すなわちデューティ比を変化させることで LED の輝度の変化が行えるということです. LED 駆動回路をモータドライバ回路に替えると, PWM 式パワーパックとなります.

5.7　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します. 1 つのファームウエアを 7 つのファイルで実現します.

表 5.1: 【ソースファイル一覧】 PWM 制御 LED 点灯

初	頁	番号	ファイル名	内容	行数
✓	114	5.1	`main.c`	メイン関数	23
✓	119	5.7	`LED-Light-LED.c`	LED 点灯	56
✓	114	5.2	`io.h`	I/O ピン割付: ヘッダ	13
✓	114	5.3	`mcr.h`	MCR・周辺モジュール: ヘッダ	11
✓	116	5.4	`mcr.c`	MCR・周辺モジュール: 実装	99
✓	117	5.5	`adc.h`	ADC モジュール: ヘッダ	9
✓	118	5.6	`adc.c`	ADC モジュール: 実装	76
				計 287 行 (うち初出 287 行, 既出 0 行)	

ソースファイル一覧表の読み方を説明します. 左から順に, そのファイルが初出か否かを示す印, 掲載ページ, ソースファイル番号, ソースファイル名, 内容の概略, ソースファイルの行数となっています. 本書ではソースファイルはなるべく再利用可能な形で開発し, ひとつのソースファイルを複数の章のファームウエアでも利用するように努めています. またソースファイルの掲載と説明は初出のものだけにして紙面の節約を行います. しかし本当の目的は, ソフトウエア開発と保守の労力の削減とソースファイル理解に要する労力の削減にあります. 本章と後の章では LED 点灯装置用のファームウエアを各種作成します. つまりハードウエアは共用して様々な点灯パターンの入れ替えをファームウエアを入れ替えることで実現します. 上記のソースファイル一覧表に掲載のソースファイルのうちのいくつかは共用するソースファイルで, 残りは各ファームウエア固有のものです.

- 共用ソースファイル: `main.c`, `io.h`, `mcr.h`, `mcr.c`, `adc.h`, `adc.c`
- 点灯パターン固有のソースファイル: `LED-Light-LED.c` など

5.7.1　LED 点灯装置での共用ソースファイル

最初に LED 点灯装置の各種ファームウエア間で共用するソースファイルを説明します.

ソースファイル main.c (メイン関数)

これがメイン関数で, 外部の関数 Light() を呼び出して LED の点灯制御を行います.

ソースファイル番号 5.1: main.c

```
/*********************************************
 * LEDライト / PIC16F18313
 * by Hirotsugu Kakugawa
 *********************************************/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"

extern void Light(void);

/*-----------------------------------------------------
 * MAIN
 *-----------------------------------------------------*/
void main(void)
{
  /* システム初期化 */
  System_Init();
  /* ウオッチドッグタイマ 2 [Sec] typ. */
  WDTCONbits.WDTPS = 0x0B; /* 0b01011 */
  CLRWDT();
  /* 発光 */
  Light();
}
```

ソースファイル io.h (I/O ピン割付: ヘッダ)

マイコンのピンや内蔵周辺モジュールを利用する関数に対して, アプリケーションの観点からの意味を持つ記号名 (マクロ名) を与えます. また各種定数などにも記号名を与えます. なおこれは C プリプロセッサのマクロ機能定義#define を用いて行います. プログラミングの際にはマイコンでの物理的な名称は使用せずに, 応用上の観点での論理的な名称を使用して, ソースコード上でアプリケーションロジックを分かりやすくするのが目的です. マクロ SW_ON() の定義では, その名称から SW_ON() が真の場合にスイッチがオンであるとソースコード上で読み取れます. 物理的にはデジタル I/O ポート RA3 が 0 (Low) の場合がそうなのですが, ポート名や, On となるのは 0 の場合なのか 1 の場合なのか, などを覚えておく必要がありません.

ソースファイル番号 5.2: io.h

```
#if !defined(__IO_H__)
#define __IO_H__

/* パラメーター設定ボリュームADCチャンネル: ANA0 */
#define ADC_CH_PARAM            0x00
/* 発光On/Off入力(L:ON, H:OFF): RA3 */
#define SW_ON()                 (RA3 == 0)
/* LED出力ポート(PWM): P1A (RA5) */
#define LED_INTENSITY(D)        CCP1_SetDutyCycle(D);
/* 最大輝度値 */
#define INTENSITY_MAX           1023

#endif /*__IO_H__*/
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンそのものや内蔵周辺モジュールの初期化を行う関数のプロトタイプ宣言を行うヘッダファイルです. これはマイコンならびにファームウエアごとに違う場合がほとんどです.

ソースファイル番号 5.3: mcr.h

```
1  #if !defined(__MCR_H__)
2  #define __MCR_H__
3
4  #define _XTAL_FREQ 32000000 /* 32MHz */
5
6  extern void System_Init(void);
7  extern void CCP1_Init(void);
8  extern void CCP1_SetDutyCycle(uint16_t duty);
9  extern void TMR2_Init(void);
10
11 #endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンそのものや内蔵周辺モジュールの初期化を行う関数の実装を行います. これはマイコンならびにファームウエアごとに違う場合がほとんどです.

- 関数 System_Init()
  クロックを生成する発振器の設定や, 以下に示すような I/O ピンの設定を行います.
  - LATx: デジタル出力での初期値を設定.
  - TRISx: 入力か出力かを設定.
  - ANSELx: アナログ入力かデジタル入力かを設定.
  - WPUx: 弱プルアップをするか否かを設定.
  - ODCONx: オープンドレイン出力とするか否かを設定.

  なお上記の x はポート名 A, B, C などで, マイコンが内蔵している I/O ポートに合わせて設定します. I/O ピンについては第 5.1 節で説明した通りです. また RA5 ピンが CCP1 モジュールの出力となるように PPS の設定を行います. PPS については第 5.5 節で説明した通りです. そして最後に内蔵周辺モジュールを制御する各種関数群の初期化を行います.
- 関数 TMR2_Init()
  内蔵周辺モジュールのタイマ TMR2 を周波数を 31.25kHz でカウントが 1 巡するように設定します. これは後述の PWM 制御において使用します. TMR2 については第 5.3 節で説明した通りです.
- 関数 CCP1_Init() および関数 CCP1_SetDutyCycle()
  CCP1 モジュールを使って PWM 制御出力を生成します.
  - PWM 周波数の決定
    TMR2 から入力を受けるように設定しているので, PWM 周波数は 31.25kHz となります.
  - PWM デューティ比の決定
    関数 CCP1_SetDutyCycle() で設定を行います. 具体的には CCPPR1H と CCPPR1L に PWM デューティ比を表す 0 (0%) から 1023 (100%) の範囲の 10 ビットの値を設定します.
  - PWM 波の出力先
    マイコン内部の CCP1 の PWM 波は, 上述の通り PPS 機能によってマイコン外部の RA5 ピンに出力されるよう関数 System_Init() 内で設定をしています.

  CCP については第 5.4 節で説明した通りです.
- #pragma
  マイコンの各種設定を行います. これら設定は PIC マイコンにファームウエアを書き込む時に行われます. つまりファームウエアを実行するときに設定しているのではありません. この設定の一部を簡単に説明すると以下の通りです. 詳細はデータシートをご参照下さい.

– マイコンはクロック 32MHz で動作させる.
– MCLR ピンを I/O ピンとして使用する.
– ウオッチドックタイマ機能を有効にする.
– 低電圧でのファームウエアの書き込みは行わない.

ソースファイル番号 5.4: mcr.c

```
1  /*-------------------------------------------------------
2   * MCR設定&周辺モジュール (PIC16F18313)
3   *-------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8
9  /*-----------------------------------------------*/
10 /* MCR初期化 (FOSC=32MHz) */
11 void System_Init(void)
12 {
13   /* 発振器 */
14   OSCCON1 = 0x00; /* HFINTOSC with 2x PLL */
15   OSCCON3 = 0x00;
16   OSCEN   = 0x00;
17   OSCFRQ  = 0x06; /* 32MHz */
18   OSCTUNE = 0x00;
19   while (PLLR == 0) {
20     ;
21   }
22   /* I/Oピン */
23   LATA   = 0x00;
24   TRISA  = 0x1F; /* (RA4),RA3,(RA2,RA1),RA0 */
25   ANSELA = 0x01; /* ANA0 */
26   WPUA   = 0x16; /* (RA4),    (RA2,RA1)     */
27   ODCONA = 0x00;
28   int8_t ogie = GIE;
29   GIE = 0;
30   PPSLOCK = 0x55;
31   PPSLOCK = 0xAA;
32   PPSLOCKbits.PPSLOCKED = 0x00;
33   RA5PPS = 0x0C; /* RA5 <- CCP1 */
34   PPSLOCK = 0x55;
35   PPSLOCK = 0xAA;
36   PPSLOCKbits.PPSLOCKED = 0x01;
37   GIE = ogie;
38   /* 周辺モジュール */
39   ADC_Init();
40   TMR2_Init();
41   CCP1_Init();
42 }
43
44 /*-------------------------------------------*/
45 /* TMR2 */
46 /*   PWM周波数 = 31.25kHz
47  *      (32MHz/(31.25kHz*4*1)) - 1 = 255 = PR2
48  */
49 void TMR2_Init(void)
50 {
51   T2CONbits.TMR2ON  = 0; /* Disabled */
52   T2CONbits.T2OUTPS = 0; /* Postscale 1:1 */
53   T2CONbits.T2CKPS  = 0; /* Prescale: 1:1 */
54   PR2  = 0xFF;
55   TMR2 = 0;
56   TMR2IF = 1;
57   T2CONbits.TMR2ON  = 1; /* Enabled */
58 }
59
60 /*-------------------------------------------*/
61 /* CCP1 */
```

```
62 void CCP1_Init(void)
63 {
64   CCP1CONbits.CCP1EN   = 1;   /* Enabled */
65   CCP1CONbits.CCP1FMT  = 0;   /* 右寄せ形式 */
66   CCP1CONbits.CCP1MODE = 0xF; /* PWMモード */
67   CCPR1L = 0x00;
68   CCPR1H = 0x00;
69 }
70
71 void CCP1_SetDutyCycle(uint16_t duty)
72 {
73   CCPR1H = duty >> 8; /* 2 MSBs */
74   CCPR1L = duty;      /* 8 LSBs */
75 }
76
77 /*-------------------------------------------*/
78 /* CONFIG1 */
79 #pragma config FEXTOSC  = OFF
80 #pragma config RSTOSC   = HFINT32
81 #pragma config CLKOUTEN = OFF
82 #pragma config CSWEN    = ON
83 #pragma config FCMEN    = ON
84 /* CONFIG2 */
85 #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
86 #pragma config PWRTE    = OFF
87 #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
88 #pragma config LPBOREN  = OFF
89 #pragma config BOREN    = ON
90 #pragma config BORV     = LOW
91 #pragma config PPS1WAY  = ON
92 #pragma config STVREN   = ON
93 #pragma config DEBUG    = OFF
94 /* CONFIG3 */
95 #pragma config WRT      = OFF
96 #pragma config LVP      = OFF /* 低電圧書込不可 */
97 /* CONFIG4 */
98 #pragma config CP       = OFF
99 #pragma config CPD      = OFF
```

ソースファイル adc.h (ADC モジュール: ヘッダ)

内蔵周辺モジュールのひとつ, A/D 変換モジュールを利用する関数群のプロトタイプ宣言とデータ型の定義を行うヘッダファイルです.

ソースファイル番号 5.5: adc.h

```
1 #if !defined(__ADC_H__)
2 #define __ADC_H__
3
4 typedef uint16_t  adc_val_t;
5 typedef uint8_t   adc_ch_t;
6 extern void      ADC_Init(void);
7 extern adc_val_t ADC_Read(adc_ch_t ch);
8
9 #endif /*__ADC_H__*/
```

ソースファイル adc.c (ADC モジュール: 実装)

ADC(アナログ/デジタル変換) モジュールを利用する関数群の実装を行います.

- 関数 ADC_Init()

  ADC モジュールの初期化を行います.
  - 10 ビットの変換結果は右寄せ形式で得ます.
  - ADC クロックの設定を行います.

- 関数 ADC_Read()

  引数で指定されたチャンネル (ピンに対応) にかかる電圧値を, 10 ビットのデジタル値に変換します. 変換結果をこの関数の返り値とします. この関数の内部での動作は以下の通りです.

  – ADC をオンにします.
  – ADC チャンネルを設定して, ADC への入力ピンを選択します.
  – ADC 内のコンデンサに入力ピンから充電を行い, コンデンサの電圧を入力ピンと同じにします. 充電時間として 5 マイクロ秒待機します.
  – 変換を開始します.
  – 変換の完了を待ちます.

ADC モジュールは PIC マイコンでも型番ごとに使い方が異なるので, C プリプロセッサの条件コンパイル機能を使ってその使い分けを行います.

ソースファイル番号 5.6: adc.c

```
1  /*-----------------------------------------------------
2   * AD変換
3   * 16F18313, 16F18323, 16F18324, 16F18344,
4   * 16F18325, 16F18345, 16F18326, 16F18346,
5   * 16F18854, 16F18855, 16F18856, 16F18857,
6   * 16F18875, 16F18876, 16F18877
7   *-----------------------------------------------------*/
8  #include <xc.h>
9  #include <stdint.h>
10 #include "mcr.h"
11 #include "adc.h"
12
13 #if   defined(_16F18313) || defined(_16F18323) || \
14       defined(_16F18324) || defined(_16F18344) || \
15       defined(_16F18325) || defined(_16F18345) || \
16       defined(_16F18326) || defined(_16F18346)
17
18 /* ADCモジュール初期化 */
19 void ADC_Init(void)
20 {
21   ADCON0bits.ADON   = 1; /* Enabled */
22   ADCON1bits.ADFM   = 1; /* Right Justified Format */
23   ADCON1bits.ADCS   = 2; /* FOSC/32 */
24   ADCON1bits.ADNREF = 0; /* VSS */
25   ADCON1bits.ADPREF = 0; /* VDD */
26 }
27
28 /* ADC実行 */
29 adc_val_t ADC_Read(adc_ch_t ch)
30 {
31   ADCON0bits.CHS = ch;
32   ADCON0bits.ADON = 1;
33   __delay_us(5);
34   ADCON0bits.GO_nDONE = 1;
35   while (ADCON0bits.GO_nDONE != 0) {
36     ;
37   }
38   return ((adc_val_t)ADRESH << 8) | (adc_val_t)ADRESL;
39 }
40
41 #elif defined(_16F18854) || defined(_16F18855) || \
42       defined(_16F18856) || defined(_16F18857) || \
43       defined(_16F18875) || defined(_16F18876) || \
44       defined(_16F18877)
45
46 /* ADCモジュール初期化 */
47 void ADC_Init(void)
48 {
49   ADCON0bits.ADON   = 1; /* ADC Enabled */
```

```
50    ADCON0bits.ADFRM0 = 1; /* Right Justified Format */
51    ADCON1 = 0x00;
52    ADCON2 = 0x00;
53    ADCON3 = 0x00;
54    ADCLK  = 0x0F; /* FOSC/32 */
55    ADREF  = 0x00; /* ADNREF=VSS, ADPREF=VDD */
56  }
57
58  /* ADC実行 */
59  adc_val_t ADC_Read(adc_ch_t ch)
60  {
61    ADPCH = ch;
62    ADCON0bits.ADON   = 1;
63    ADCON0bits.ADCONT = 0;
64    __delay_us(5);
65    ADCON0bits.ADGO = 1;
66    while (ADCON0bits.ADGO != 0) {
67      ;
68    }
69    return ((adc_val_t)ADRESH << 8) | (adc_val_t)ADRESL;
70  }
71
72  #else
73
74  # error "YOUR DEVICE IS NOT SUPPORTED"
75
76  #endif
```

5.7.2　LED 点灯ファームウエア固有のソースコード

次に LED 点灯モジュール用ファームウエア固有のアプリケーションロジックを実現するソースファイルを説明します. アプリケーションロジックを変更すれば同じハードウエアを用いて LED の点滅パターンの変更ができます.

ソースファイル LED-Light-LED.c (LED 点灯)

LED の点灯の制御を行う中心的な動作を以下のソースファイルで行います. まず最初に大まかな説明をします.

- ウオッチドッグタイマ
  CLRWDT() の呼び出しが一定時間の間になければマイコンをリセットする機能で, 詳しくは第 12.5 節で説明します. これはバグによる無限ループから抜け出すためのもので, バグによる影響が深刻となる組み込みシステムにおいてシステムの信頼性を高める方法のひとつです. リセットがかかるまでの時間はソフトウエアで設定可能で, このプログラムでは 2 秒としています. CLRWDT() の呼び出しのことを, ウオッチドッグタイマをクリアすると言います.
- 状態遷移手法による LED 点灯制御
  状態遷移手法とは, 有限状態機械とその状態遷移という形で対象のモデル化を行うソフトウエア設計技法です. 本章では LED 点灯制御の例を通じて状態遷移手法の基礎を学びます. 詳しくはソースコードの後に説明します.

ソースファイル番号 5.7: LED-Light-LED.c

```
1   /*********************************************
2    * 単純On/Off
3    *********************************************/
4   #include <xc.h>
5   #include <stdint.h>
6   #include "mcr.h"
7   #include "io.h"
```

```
#include "adc.h"

/*
 * SW ON:  瞬時に明るくなる
 * SW OFF: 瞬時に暗くなる
 */

/* 有限状態機械 */
enum fsm_state_e {
  STATE_Q0 = 0, /* 消灯 */
  STATE_Q1,     /* 点灯 */
};

void Light(void)
{
  /* 状態 */
  enum fsm_state_e fsm_state = STATE_Q0;
  /* 現在輝度 */
  uint16_t intensity_current = 0;
  /* 主ループ */
  for (;;) {
    __delay_ms(10);
    CLRWDT();
    /* 制御入力を読む */
    int8_t sw_on = SW_ON();
    /* パラメータボリュームを読む (10ビット: 0..1023) */
    adc_val_t adc_val_param = ADC_Read(ADC_CH_PARAM);
    /* 有限状態機械 */
    switch (fsm_state) {
    case STATE_Q0: /* 消灯状態 */
      intensity_current = 0;
      /* 状態遷移 */
      if (sw_on) {
        fsm_state = STATE_Q1;
      }
      break;
    case STATE_Q1: /* 点灯状態 */
      /* 輝度計算 */
      intensity_current = adc_val_param;
      /* 状態遷移 */
      if (!sw_on) {
        fsm_state = STATE_Q0;
      }
      break;
    }
    /* 輝度出力 */
    LED_INTENSITY(intensity_current);
  }
}
```

なおディレイ関数__delay_ms() は, 引数に指定された値をミリ秒単位の時間として, その時間が経つまで何もしないという動作をします. その間ウオッチドッグタイマはクリアされませんので注意が必要です.

上記プログラムでの関数 Light() の骨子は以下の通りとなります.

```
fsm_state = 状態Q0;
for (;;) {
  一定時間待機 __delay_ms(10);
  sw_on = スイッチ読み取り;
  adc_val_param = 可変抵抗器によるパラメータ読み取り;
  switch (fsm_state) {
  case 状態Q0: /* 消灯状態 */
    intensity_current = 0;
    if (sw_on) {
      fsm_state = 状態Q1;
```

```
11      }
12      break;
13    case 状態Q1: /* 点灯状態 */
14      intensity_current = 最大;
15      if (!sw_on) {
16        fsm_state = 状態Q0;
17      }
18      break;
19    }
20    LED輝度をintensity_currentに設定;
21  }
```

この骨子から次のことが分かります.

- プログラムは無限のループを構成
  このループをメインループと呼びます. リセットが起きないように, ウオッチドッグタイマのクリアを行います. また, 10 ミリ秒間隔でメインループ本体部分を実行しています.
- 状態 Q0
  LED の輝度を 0 にすることで消灯します. スイッチ入力がオンになれば状態は Q1 に遷移します. この状態を初期状態として動作が開始します.
- 状態 Q1
  LED の輝度を最大にすることで点灯します. スイッチ入力がオフになれば状態は Q0 に遷移します.

この状態遷移の関係は図 5.8 に示す通りに表現できます. この例のように, 状態とそれらの間の遷移を図で表したものを状態遷移図 (state transition diagram) と呼びます. また, 状態群と状態遷移をプログラムで表現したものを有限状態機械 (finite state machine) あるいは単に状態機械と呼びます. 本章の有限状態機械は非常に単純なために有限状態機械を用いる有用性はあまり感じられないかもしれません. しかし複雑な動作をする場合にはプログラム構造が単純明解になり, 有用な設計手法であることが後ほど分かってきます.

メインループ本体部分ではディレイ関数を用いて一定時間待機をしていますが, これを行う理由を説明します. 後の章のファームウエアでは, LED を徐々に明るくする, あるいはモータを徐々に高速にするなどの制御を行います. もしメインループ本体の実行に一定時間の間隔をとらないと, 一瞬で輝度が最大になったり, 一瞬でモータが最高速度になってしまいます. これを避ける方法の中で一番簡単なのがディレイ関数を用いた方法なのです. 本章のファームエアでディレイ関数を用いる理由はありませんが, 後の章のファームウエアでの書き方に合わせています.

本書のファームウエアを構成するソースファイルは以下の共通した方針で行います.

- `io.h`
  回路におけるマイコンのピンや内蔵周辺モジュールにアプリケーションの観点から記号名を与えます.
- `mcr.h`

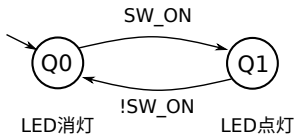


図 5.8: 状態遷移図

マイコンそのものや内蔵周辺モジュールの初期化を行う関数のプロトタイプ宣言を行うヘッダファイルです.

- `mcr.c`
  マイコンそのものや内蔵周辺モジュールの初期化を行う関数の実装を行います.
- 他の内蔵周辺モジュールを利用するためのソースファイル群
  上記の例では `adc.h` や `adc.c` です.
- アプリケーション本体
  装置を動作させるアプリケーションロジックを実現します. 本章の例では `LED-Light-LED.c` がそれに該当します. アプリケーション本体も複数のファイルで構成されるファームウエアも多数あります.

課題

1. ソースファイル `mcr.c` におけるレジスタ `LATA`, `TRIS`, `ANSELA` の設定値が, 87 ページで示した図 4.2 の回路図およびデータシート記載の仕様と合致していることを説明して下さい.
2. PPS 設定はリセット後に 1 度しか行えないようにコンフィギュレーションビット `PPS1WAY` の設定を行って下さい. この設定が正常に行えていることを確認する方法を考えて, 実際に確かめて下さい.

第 II 部

PWM 制御による
ジオラマ用 LED 照明とパワーパック

第 6 章

ブレゼンハムアルゴリズムの基礎理論

本章では, LED の輝度の変化やモータ加減速の制御に便利な, ブレゼンハムアルゴリズムを紹介します. ここでの内容はブレゼンハムアルゴリズムの動作原理を完全に理解するための理論的な話が中心なので, 本章を読み飛ばしてもまったく構いません. ブレゼンハムアルゴリズムの目的だけが分かれば十分です.

オリジナルのブレゼンハムアルゴリズムは一次関数 $y = at + b$ の計算を行うもので, 本書では特にマイコンのファームウエアにおいて一定時間の間隔で LED の輝度やモータの回転速度の増減に使用します. ブレゼンハムアルゴリズムの特に重要な点はわずかな整数演算で実現できるところにあり, PIC 16F などの 8 ビットの整数加算回路しか持たないマイコンでは特に有用です.

本章で紹介するブレゼンハムアルゴリズムを実装した関数を使用した場合, 次の骨子のプログラムコードにより時々刻々と y の値を増加させることができます. なお一次関数の傾きは $a = dy/dx$ で, y の値が目標値に達したらそれ以降は変化させません.

```
1  uint16_t b = 0;
2  incr_lin_init(b); /* y=bの設定 */
3  for (;;) {
4    /* y=ax+bの次のyの値を求める (a=dy/dx) */
5    uint16_t y = incr_lin_next(dx, dy, 目標値);
6    yの値を使用(LED輝度やモータ回転数などの指定);
7  }
```

一定時間間隔 (例えば 50 ミリ秒間隔) で一次関数 $y = at + b$ の計算を行ういくつかの方法を比較してみます.

- 実数演算による実現
  例えば, この場合での C 言語のプログラムは次のように書くことになるでしょう. このように何気なく書いてしまいそうですが, 実数演算の乗算と加算が, ハードウエアやソフトウエアによりどう実現されているかを考えて, 計算の手間を想像してみて下さい.

```
1  static double a = 3.75;
2  static double b = 1.00;
3  double motor_speed_at_t(double t) {
```

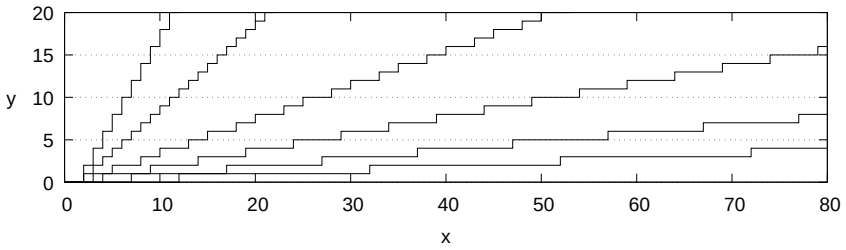


図 6.1: ブレゼンハムアルゴリズムによる一次関数の計算 (様々な傾きに対して)

```
4     double y = a * t + b;
5     return y;
6   }
```

乗算回路を含む実数演算回路を内蔵したマイコンならハードウエアによる高速演算ができます. しかしそうでなければ, ソフトウエアで多くの整数演算を組み合わせて実数演算を実現するしかなく, 高速な動作はできません. リアルタイム性が要求される組込みシステムには不向きです.

- 係数 a の加算による実現
  8 ビットマイコンでの整数の加減算を用いて, 一定時間ごとに係数 a を y の値に加えてゆく方法です. 例えば, この場合での C 言語のプログラムは, 整数演算を用いて次のように書くことになるでしょう.

```
1   static uint16_t a = 4;
2   static uint16_t y = 1;
3   int motor_speed_at_each_t(void) {
4     y = y + a;
5     return y;
6   }
```

この通り簡単に一次関数 $y = at + b$ の計算が行えるように見えるのですが, 落とし穴は a の値は整数でしかも 1 以上でなければならない点です. これでは細やかな増加の度合いの設定は不可能ですし, a の値が 1 未満のゆるやかな増加も不可能です. このように, 使い方にはいろいろと不便なところがあります.

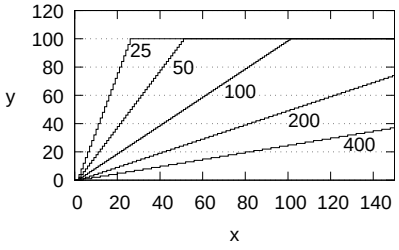
- ブレゼンハムアルゴリズムによる実現
  整数の加減算だけで高速に動作します. 図 6.1 に示すように, 細やかな増加の度合いを設定や, a の値が 1 未満のゆるやかな増加が可能です.

ブレゼンハムアルゴリズムは, 2 次元コンピュータグラフィックスで高速な線分の描画を実現するために, ブレゼンハム (Bresenham) 氏が 1962 年に考案したものです (文献 [2]). ブレゼンハムアルゴリズムには様々な実装がありますが, 本書では組み込みリアルタイムシステムでの利用に適した実装を行います. 後の章では LED 輝度やモータ回転速度の制御に応用して, LED 点灯装置やパワーパックのファームウエアで使用します. 本章では本来のブレゼンハムアルゴリズムである線形的増減の計算を行うものと, それを応用した指数的増減の計算を行うアルゴリズム 2 種を紹介します.

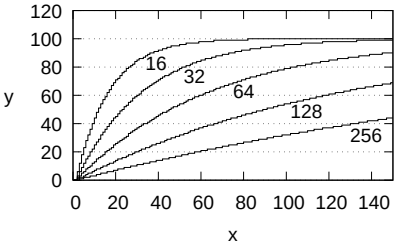
- ブレゼンハム線形増加アルゴリズム
  値を線形的に増加します. 例を図 6.2a で示す通り, 増加の度合いを表すグラフは直線です. 最初の値を y_0 とし, 値が y_1 になるまで傾き D_Y/D_X で増加をします. y の値が y_1 になるとその後は一定値をとります. 用いるパラメータは y_0, y_1, D_Y, D_X の 4 つです.
- ブレゼンハム指数増加アルゴリズム
  値を指数的に増加します. 例を図 6.2b で示す通り, 増加の度合いを表すグラフは CR 回路におけるコンデンサ充電曲線の形をしています. 最初の値を y_0 とし, 値が y_1 になるまで時定数を T_C で増加をします. y の値は y_1 にゆっくりと近づいてゆきます. 用いるパラメータは y_0, y_1, T_C の 3 つです.

いずれも値は y_0 から y_1 へ増加させていきますが, もし $y_1 < y_0$ の場合は減少となります.

コンピュータグラフィックスでの線分の描画とモータ回転速度や LED 輝度の制御との関係を説明します. 図 6.3 にこれらの関係を示します. 2 次元平面上の 2 つの点 $P_0 = (x_0, y_0)$ と $P_1 = (x_1, y_1)$ の間をまっすぐに結んだものが線分です. するとこの線分は以下の 1 次関数で表されます.

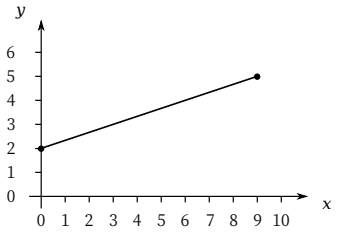


(a) 各 D_X の値に対する線形増加 ($D_Y = 100$)

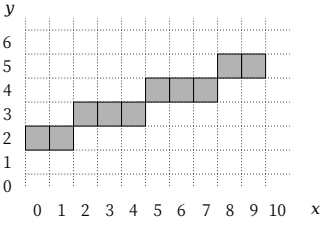


(b) 各 T_C の値に対する指数増加

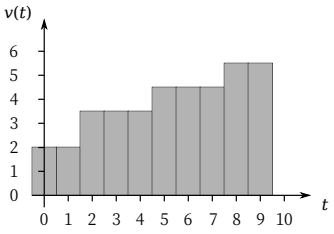
図 6.2: 本書で紹介するブレゼンハムアルゴリズム 2 種 ($y_0 = 0, y_1 = 100$)



(a) 関数 f による線分 (連続値)



(b) 2 次元グラフィックス
におけるピクセル (離散値)



(c) モータ回転速度の
デジタル制御 (離散値)

図 6.3: 線分, ピクセル, モータ回転速度

$$
\begin{aligned}
y = f(x) &= ax + b, \text{ ただし } x_0 \le x \le x_1 \\
a &= (y_1 - y_0)/(x_1 - x_0) \\
b &= y_1 - x_1(y_1 - y_0)/(x_1 - x_0)
\end{aligned}
$$

2 次元コンピュータグラフィックスでの画面はピクセルの集合であり, 各ピクセルは 2 次元座標値を持っています. 背景はあらかじめ白色に塗られているものとします. そして $y = f(x)$ の関係を満たす点 (x, y) に対応するピクセルを黒色に塗ることで, 画面上に線分を描くことができます. モータの回転速度の制御の場合では, 横軸の時間軸 t に対して, 縦軸は時刻 t におけるモータへ印加する電圧 $v(t)$ に対応します. LED の輝度制御の場合では, 横軸の時間軸 t に対して, 縦軸は時刻 t における LED 輝度 $v(t)$ に対応します.

ブレゼンハムアルゴリズムの大きな特徴はその高速性にあります. 本書では 8 ビットの PIC マイコンを使っていますので, 実数演算には多大な時間がかかり, また整数演算であっても乗算や除算には大きな時間を要します. しかしブレゼンハムアルゴリズムではこれらの演算はまったく行わず, 関数 $f(x) = ax + b$ の計算をわずかな回数の整数の加減算で行います.

6.1　線形増加アルゴリズム

6.1.1　典型的な使用例

想定している典型的な使用方法のひとつがゆっくりと加減速を行うパワーパックで, トランジスタコントローラに似た加減速が実現できます. プログラムコードの骨子は以下の通りです.

```
1  uint16_t y0 = 0;
2  incr_lin_init(y0);
3  モータ速度を y0 に設定;
4  for (;;) {
```

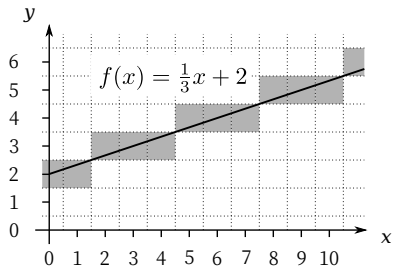


図 6.4: 連続直線と離散直線

```
5     __delay_ms(10);
6     uint16_t y1 = 目標速度;
7     uint16_t y = incr_lin_next(dx, dy, y1);
8     モータ制御のPWMデューティ比をyに設定;
9   }
```

このプログラムコードの動作は以下の通りです.

- 最初はモータは停止しておきます.
- 1 周期 10 ミリ秒で実行を繰り返します. 各周期での動作は以下の通りです.
- 目標速度を読み取ります. これは可変抵抗器などで操作可能にしておきます.
- その目標速度に向けて, モータ速度は徐々に上昇あるいは下降してゆきます.
- 速度の増加あるいは減少の度合いは 2 つのパラメータ dx と dy で指定します. これらは可変抵抗器などで設定可能にしておきます.

上記のプログラムコードの骨子で使われている関数それぞれの動作は以下の通りです.

- `incr_lin_init(y0)`
  線分の開始点の y の値を `y0` で指定します.
- `incr_lin_next(dx, dy, y1)`
  線分の次の y の値を返します. 線分の傾きの 2 つのパラメータ D_X と D_Y はそれぞれ, dx と dy で指定します. 線分の終了点の y の値を y_1 で指定します.

6.1.2　原理

値を線形的に増加するブレゼンハム線形増加アルゴリズムの基本原理を説明します. ブレゼンハムアルゴリズムには様々な変形がありますが, その中でも文献 [17] を元にした説明をします. 描く線分は図 6.4 に示すように, 2 点 (x_0, y_0) と (x_1, y_1) の間とします. この線分を表す 1 次関数は $f(x) = ax + b$ の形をしており, a, b の値の決め方は上述の通りです. ただし一般性を失うことなく $x_0 < x_1$ とし, x_0, y_0, x_1, y_1 は全て整数とします. また説明の簡単のため $y_0 \leq y_1$ と仮定しますが, $y_0 \geq y_1$ の場合には値の増加と減少を入れ替えて考えれば以下の説明がそのまま当てはまります.

計算を整数のみで行うことが目的ですので, 各整数値 X に対して $f(X)$ の値に一番近い整数値 Y を求めることが必要となります. つまり連続値をとる関数 $y = f(x)$ から, 離散値で近似する関数 $Y = F(X)$ を決めることになります. 応用の観点から図 6.4 を見ると, 横軸が時間で縦軸がモータ制御の PWM デューティ比と思えば良いでしょう. ソフトウエアでは処理データはすべて離散化されていて, 時々刻々と進む時間もモータ制御の PWM デューティ比もともに整数値です.

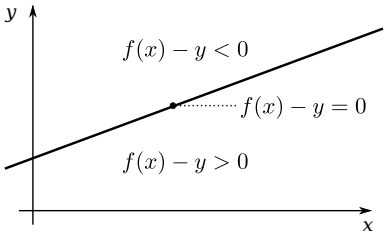


図 6.5: 直線による平面の二分

ブレゼンハムアルゴリズムでは, 整数値 X に対して求めた計算結果を再利用して $X+1$ に対する計算結果を得ます. つまり X の値を 1 つづつ増加しながら計算結果 Y の値を順次求め, 画面上のピクセル (X,Y) を黒色に塗ります. このとき最初 $X = x_0$ に対する計算結果は $Y = y_0$ であり, これは計算するまでもなく与えられたパラメータそのものです. そして $X = x_0+1, x_0+2, ..., x_1$ それぞれに対する計算結果を, 前回の計算結果を利用して順次求めます. 整数値 X に対して $y = f(X)$ の値は整数とは限りませんが, ブレゼンハムアルゴリズムでは y に一番近い整数値 Y を求めて計算結果とします. このとき $y = Y$ とは限らないために誤差が生じることになります. ブレゼンハムアルゴリズムではこの誤差を計算・蓄積して, 常に累積誤差が最小となるように次の計算出力を決定します.

誤差

誤差の定め方を説明します. 図 6.5 に 2 次元平面上の直線 $y = f(x)$ を示しています. するとこの直線を構成する点 (x,y) はいずれも関係式 $f(x) - y = 0$ を満たします. すると 2 次元平面はこの直線で 2 つの領域に分割されます. 一方の領域は直線の上側で, この領域にある点 (x,y) はいずれも関係式 $f(x) - y < 0$ を満たします. 他方の領域は直線の下側で, この領域にある点 (x,y) はいずれも関係式 $f(x) - y > 0$ を満たします. 2 次元平面上の各点 (x,y) における誤差 $\varepsilon_{(x,y)}$ を次の式で定義します.

$$\varepsilon_{(x,y)} = y - f(x)$$

この誤差 $\varepsilon_{(x,y)}$ の値により, 点 (x,y) が直線からどれだけ離れているかを表すとともに, その値の正負によって直線の上か下かを表します. 以下の通り D_Y と D_X を定義します.

$$\begin{aligned} D_Y &= y_1 - y_0 \\ D_X &= x_1 - x_0 \\ y = f(x) &= ax + b \\ &= (D_Y/D_X)x + b \end{aligned}$$

すると点 (x,y) における誤差 $\varepsilon_{(x,y)}$ は以下の通りとなります.

$$\varepsilon_{(x,y)} = y - (D_Y/D_X)x - b$$

この式の両辺に D_X を乗じて除算をなくします.

$$\varepsilon_{(x,y)}D_X = yD_X - xD_Y - bD_X$$

簡単化のため $E_{(x,y)} = D_X\varepsilon_{(x,y)}$ とおき, 以降ではこの $E_{(x,y)}$ を新たな誤差の定義として用います.

$$E_{(x,y)} = yD_X - xD_Y - bD_X$$

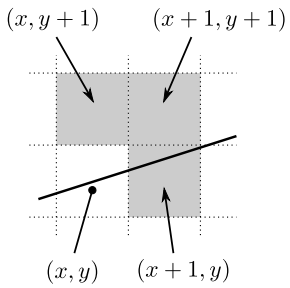


図 6.6: ピクセル (x, y) に隣接する 3 つの候補ピクセル

3 つの候補ピクセル

以上の通りに定義された誤差を元に, ブレゼンハムアルゴリズムが次に黒色に塗るピクセルを求める方法を説明します. 最初に黒色に塗るピクセルは線分の出発点 (x_0, y_0) で, このときピクセルの座標は関数値に一致するので誤差は 0 です.

$$E_{(x_0,y_0)} = 0$$

次に一般的な場合を考えます. 最後に黒く塗ったピクセルを (x, y) としたとき, 次に黒く塗るピクセルの候補は図 6.6 に示す通り, 上 $(x, y+1)$, 右上 $(x+1, y+1)$, 右 $(x+1, y)$ の 3 つです. これは仮定 $y_0 \le y_1$ に基づくもので, 下方のピクセルを塗ることはないからです. 候補ピクセルそれぞれの誤差値は式 6.1 の通りとなります.

$$\begin{aligned} E_{(x+0,y+1)} &= (y+1)D_X - xD_Y - bD_X \\ &= E_{(x,y)} + D_X \\ E_{(x+1,y+1)} &= (y+1)D_X - (x+1)D_Y - bD_X \\ &= E_{(x,y)} + D_X - D_Y \\ E_{(x+1,y+0)} &= yD_X - (x+1)D_Y - bD_X \\ &= E_{(x,y)} - D_Y \end{aligned} \tag{6.1}$$

3 つのピクセルの中で $f(x)$ に一番近いものは, 誤差値が 0 に一番近くなります. 従って次に黒色に塗るピクセルには 3 つの中で誤差値が 0 に一番近いものを選びます. 式 6.1 から分かる通り, 次に選ぶピクセルの誤差値は現在のピクセル (x, y) の誤差値 $E_{(x,y)}$ に D_X や D_Y を加減するだけで求まります. つまり時間のかかる $f(x+1)$ の計算を行うことなく, 時間のかからない誤差値の更新だけで次のピクセルを選んでいます. これがブレゼンハムアルゴリズムの大きな特徴である高速性の鍵です.

誤差値に関して $D_X \ge 0$ かつ $D_Y \ge 0$ であることから, 以下の不等式 6.2 が常に成立します. なお各誤差値は正, 負, ゼロのいずれの場合にもなることに注意して下さい. この不等式は後の説明で使います.

$$E_{(x+0,y+1)} \ge E_{(x+1,y+1)} \ge E_{(x+1,y+0)} \tag{6.2}$$

6.1.3　実装

C 言語によるブレゼンハム線形増加アルゴリズムの実装を示します. 本書での実装では時間軸が X 軸であるとしており, 陽に x の値は現れません. 時刻の進行により x の値は次の値になると考えて, その x の値に対応する y の値を求めます.

ブレゼンハム線形増加アルゴリズムのヘッダファイルを以下に示します. 関数 `incr_lin_reset_error()` は内部で記憶している誤差値を 0 にリセットします.

ソースファイル番号 6.1: `incr-lin.h`

```
#if !defined(__INCR_LIN_H__)
#define __INCR_LIN_H__

extern void     incr_lin_init(uint16_t y0);
extern void     incr_lin_reset_error(void);
extern uint16_t incr_lin_next(uint16_t dx, uint16_t dy,
                              uint16_t y1);

#endif /*__INCR_LIN_H__*/
```

プログラムコードコードを以下に示します. 説明は後ほど行います.

ソースファイル番号 6.2: incr-lin.c

```
/*-----------------------------------------------------------
 * ブレゼンハム線形増加アルゴリズム
 *-----------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "incr-lin.h"

/* y値 */
static uint16_t incr_y = 0;

/* 誤差値 */
static int16_t  incr_e = 0;

/* 初期化 */
void incr_lin_init(uint16_t y0)
{
  incr_y = y0;
  incr_lin_reset_error();
}

/* 誤差値のリセット */
void incr_lin_reset_error(void)
{
  incr_e = 0;
}

/* 次のy値を得る */
uint16_t incr_lin_next(uint16_t dx, uint16_t dy,
                       uint16_t y1)
{
  /* 今のy値 */
  uint16_t y = incr_y;
  /* dx=0の時は即座に目標値へ到達させる */
  if (dx == 0) {
    incr_y = y1;
    incr_e = 0;
    return incr_y;
  }
  /* すでにy値は目標値に到達済 */
  if (incr_y == y1) {
    incr_e = 0;
    return incr_y;
  }
  /* 増加直線の傾き */
  int16_t sdx = (int16_t) dx;
  int16_t sdy = (int16_t) dy;
  /* x値が1増加するまでy値を増加させてゆく */
  for (;;) {
    /* y値が目標値に到達 - 終了 */
    if (incr_y == y1) {
      incr_e = 0;
      break;
    }
    /* 誤差値の計算 (不変式 e_01 >= e_11 >= e_10 に注意) */
    int16_t e_01 = incr_e + sdx;       /* 点 (x,y+1)   */
```

```
56      int16_t e_11 = incr_e + sdx - sdy; /* 点 (x+1,y+1) */
57      int16_t e_10 = incr_e - sdy;       /* 点 (x+1,y)   */
58      /* y値の1増加で誤差値が減少するか否かを検査 */
59      if (e_10 + e_11 < 0) { /* 誤差値は減少 */
60        /* 誤差値を更新 */
61        incr_e = e_01;
62        /* y値を1増加 */
63        if (incr_y < y1) {
64          incr_y++;
65        } else {
66          incr_y--;
67        }
68      }
69      /* x値の1増加で誤差値が減少するか否かを検査 */
70      else if (e_01 + e_11 > 0) { /* 誤差値は減少 */
71        /* 誤差値を更新 */
72        incr_e = e_10;
73        /* x値を1増加(本関数終了) */
74        break;
75      }
76    }
77    return y;
78  }
```

関数 incr_lin_next() の返り値には若干注意が必要です．返り値はその関数が呼ばれた時点での変数 incr_y の値としています．そして関数 incr_lin_next() で更新した変数 incr_y の値は，次の呼び出しのときの返り値としています．これは関数 incr_lin_next() が最初に呼ばれたときの返り値を，関数 incr_lin_init() で設定した値にするためです．

関数 incr_lin_next() を説明します．関数 incr_lin_next() のプログラムコードを見ると 3 つの誤差値 $E_{(x+0,y+1)}$ (= e_01), $E_{(x+1,y+1)}$ (= e_11), $E_{(x+1,y+0)}$ (= e_10) の中で最も 0 に近いものを直接的には選ぶのではなく，実際には以下の形となっています．

```
1   /* y値の1増加で誤差値が減少するか否かを検査 */
2   if (e_10 + e_11 < 0) { /* 誤差値は減少 */
3       誤差値を更新;
4       y値を1増加;
5   }
6   /* x値の1増加で誤差値が減少するか否かを検査 */
7   else if (e_01 + e_11 > 0) { /* 誤差値は減少 */
8     誤差値を更新;
9     関数終了; /* x値の1増加に相当 */
10  }
```

実はこのプログラムコードは先述の 3 つの誤差値の間の不等式 6.2 を利用して，3 つの誤差値の中で最も 0 に近いものを選ぶのと同等なことが保証されています．この詳しい説明は長くなりますので省略します．

次にこのプログラムコードで無限ループが起きないことを簡単に説明します．この観点からのプログラムコードの骨子が以下のものです．もし仮に無限ループが起きるとすれば，その可能性はひとつの for 文に限られています．

```
1   /* dx=0の時は即座に目標値へ到達させる */
2   if (dx == 0) {
3     関数終了;
4   }
5   /* x値が1増加するまでy値を増加させてゆく */
6   for (;;) {
7     /* y値が目標値に到達 - 終了 */
```

```
8     if (incr_y == y1) {
9       関数終了;
10    }
11    /* y値の1増加で誤差値が減少するか否かを検査 */
12    if (e_10 + e_11 < 0) { /* 誤差値は減少 */
13      incr_y++;
14    }
15    /* x値の1増加で誤差値が減少するか否かを検査 */
16    else if (e_01 + e_11 > 0) { /* 誤差値は減少 */
17      関数終了; /* x値の1増加に相当 */
18    }
19  }
```

まず初期状態では $y = y_0$ であり, また $y_0 \leq y_1$ としています. 無限ループにならずに関数を終了するのは, y 値の増加が続いたとき, あるいは x 値が 1 増加したときです. 従って仮に無限ループが起きるとすれば, 2 つ目と 3 つ目の if 文の条件が同時に偽である場合だけです. では本当にそれが起きるのかどうか調べてみます. 以下の条件式はそれらの条件が同時に偽であることを表します.

$$
\begin{aligned}
&(\mathtt{e_10} + \mathtt{e_11} >= 0) \wedge (\mathtt{e_01} + \mathtt{e_11} <= 0) \\
&\quad = \quad (E_{(x+1,y+0)} + E_{(x+1,y+1)} \geq 0) \wedge (E_{(x+0,y+1)} + E_{(x+1,y+1)} \leq 0) \\
&\quad = \quad E_{(x+1,y+0)} + E_{(x+1,y+1)} \geq 0 \geq E_{(x+0,y+1)} + E_{(x+1,y+1)} \\
&\therefore \quad E_{(x+1,y+0)} \geq E_{(x+0,y+1)}
\end{aligned}
$$

ここで 3 つの誤差値の間の不等式 6.2 を用いると以下の式を得ます.

$$E_{(x+1,y+0)} \quad = \quad E_{(x+0,y+1)}$$

これら 2 つの誤差値の定義は $E_{(x+1,y+0)} = E_{(x,y)} - D_Y$, $E_{(x+0,y+1)} = E_{(x,y)} + D_X$ であることから以下の式を得ます.

$$-D_Y = D_X$$

$D_Y \geq 0$ かつ $D_X \geq 0$ を仮定しているのでした. $D_X = 0$ の場合には関数の冒頭部分で終了しますから, for 文に実行が到達するとは決して起きません. $D_X > 0$ の場合には $-D_Y = D_X$ が成立することは決して起きません. つまり無限ループが生じる条件は成立し得ないということです. 従って関数の実行は必ず終了します.

6.2　指数増加アルゴリズム

6.2.1　典型的な使用例

想定している典型的な使用方法のひとつが LED の輝度の増加で, 電球に似た点灯をこの指数増加アルゴリズムで実現できます. プログラムコードの骨子は以下の通りです.

```
1   uint16_t y0 = 0;
2   incr_exp_init(y0);
3   LED輝度をy0に設定;
4   for (;;) {
5     __delay_ms(10);
6     uint16_t y1 = 目標輝度;
7     uint16_t y = incr_exp_next(tc, y1);
8     LED輝度をyに設定;
9   }
```

上記のプログラムコードの骨子で使われている関数それぞれの動作は以下の通りです.

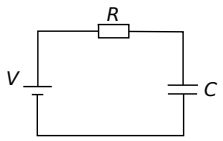


図 6.7: CR 回路

- `incr_exp_init(y0)`
  出力の開始点の y の値を `y0` で指定します.
- `incr_exp_next(tc, y1)`
  出力の次の y の値を返します. 出力の増加度合いを表す時定数パラメータを `tc` で指定します. 出力の最終目標の y の値を y_1 で指定します.

6.2.2　原理

指数関数的に増加するブレゼンハム指数増加アルゴリズムの基本原理を説明します. このアルゴリズムにおいて $y_0 = 0, y_1 = V$ とおけば, これは時定数 T_C の CR 回路におけるコンデンサ充電曲線として知られている次の式に一致します.

$$y(t) = V(1 - \exp(-t/T_C)) \tag{6.3}$$

これは, コンデンサの電荷の蓄積によりコンデンサの電圧が上昇し, それに伴い抵抗に流れる電流値が減少し, コンデンサの電荷の蓄積が緩やかになることで電圧値の増加が次第に緩やかになる現象です. また電球の輝度の場合では, フィラメントの加熱に伴い抵抗値が増えて, 加熱の進み具合が緩やかになることで電球の輝度の増加が次第に緩やかになる現象です.

式 6.3 の時間微分は以下の式となります.

$$\begin{aligned} y'(t) &= \exp(-t/T_C)V/T_C \\ &= (V - y(t))/T_C \end{aligned}$$

関数 $y(t)$ の微分係数 $y'(t)$ は, 時刻 t の瞬間における曲線 $y(t)$ の傾きに相当します. その傾きを D_Y/D_X とおきます.

$$\begin{aligned} D_Y/D_X &= y'(t) \\ &= (V - y(t))/T_C \\ (T_C D_Y)/D_X &= V - y(t) \end{aligned}$$

$D_X = T_C$ と決めれば以下の式を得ます.

$$D_Y = V - y(t)$$

ブレゼンハム指数増加アルゴリズムではこのように時々刻々と傾き D_Y/D_X を変化させて出力値を計算します. 傾きを決める 2 つのパラメータ D_Y と D_X の値をまとめると以下の通りです.

$$\begin{aligned} D_Y &= V - y(t) \\ D_X &= T_C \end{aligned}$$

ブレゼンハム指数増加アルゴリズムはブレゼンハム線形増加アルゴリズムとまったく同じ論理で出力値を計算します. その違いは, ブレゼンハム線形増加アルゴリズムでは D_Y と D_X が一定値だったのが, ブレゼンハム指数増加アルゴリズムでは D_Y と D_X が時間とともに変化しうるという点だけです. ブレゼンハム指数増加アルゴリズムはブレゼンハム線形増加アルゴリズムの一般化であり, 先に説明した原理がそのまま当てはまり

ます.

6.2.3　実装

C 言語によるブレゼンハム指数増加アルゴリズムの実装を示します.

ブレゼンハム指数増加アルゴリズムのヘッダファイルを以下に示します. なお関数 incr_exp_reset_error() は目標値を変更した際に呼ぶもので, 内部で記憶している誤差値を 0 にします.

ソースファイル番号 6.3: incr-exp.h

```
#if !defined(__INCR_EXP_H__)
#define __INCR_EXP_H__

extern void     incr_exp_init(uint16_t y0);
extern void     incr_exp_reset_error(void);
extern uint16_t incr_exp_next(uint16_t tc, uint16_t y1);

#endif /*__INCR_EXP_H__*/
```

プログラムコードコードを以下に示します. これはブレゼンハム線形増加アルゴリズムとほぼ同じで, その違いは D_Y の値が出力値に依存して決まる点だけです.

ソースファイル番号 6.4: incr-exp.c

```
/*-------------------------------------------------------
 * ブレゼンハム指数増加アルゴリズム
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "incr-exp.h"

/* y値 */
static uint16_t incr_y = 0;

/* 誤差値 */
static int16_t  incr_e = 0;

/* 初期化 */
void incr_exp_init(uint16_t y0)
{
  incr_y = y0;
  incr_exp_reset_error();
}

/* 誤差値のリセット */
void incr_exp_reset_error(void)
{
  incr_e = 0;
}

/* 次のy値を得る */
uint16_t incr_exp_next(uint16_t tc, uint16_t y1)
{
  /* 今のy値 */
  uint16_t y = incr_y;
  /* すでにy値は目標値に到達済 */
  if (incr_y == y1) {
    incr_e = 0;
    return incr_y;
  }
  /* xの増加値 */
  int16_t sdx = (int16_t)tc;
  /* yの増加値 */
  int16_t sdy;
  if (incr_y < y1) {
    sdy = y1 - incr_y;
  } else {
```

```
44      sdy = incr_y - y1;
45    }
46    for (;;) {
47      /* y値が目標値に到達 - 終了 */
48      if (incr_y == y1) {
49        incr_e = 0;
50        break;
51      }
52      /* 誤差値の計算 (不変式 e_01 >= e_11 >= e_10 に注意) */
53      int16_t e_01 = incr_e + sdx;        /* 点 (x,y+1)   */
54      int16_t e_11 = incr_e + sdx - sdy; /* 点 (x+1,y+1) */
55      int16_t e_10 = incr_e - sdy;        /* 点 (x+1,y)   */
56      /* y値の1増加で誤差値が減少するか否かを検査*/
57      if (e_10 + e_11 < 0) { /* 誤差値は減少 */
58        /* 誤差値を更新 */
59        incr_e = e_01;
60        /* y値を1増加 */
61        if (incr_y < y1) {
62          incr_y++;
63        } else {
64          incr_y--;
65        }
66      }
67      /* x値の1増加で誤差値が減少するか否かを検査 */
68      else if (e_01 + e_11 > 0) { /* 誤差値は減少 */
69        /* 誤差値を更新 */
70        incr_e = e_10;
71        /* x値を1増加(本関数終了) */
72        break;
73      }
74
75    }
76    return y;
77  }
```

課題

1. ブレゼンハム線形増加アルゴリズムに従って図 6.4 を手計算により描いて下さい.
2. 本章で紹介したブレゼンハム線形増加アルゴリズムをパーソナルコンピュータ上で動作するようにして下さい. $y_0 = 0, y_1 = 100, D_Y = 100$ とし, D_X の値には 10 と 500 それぞれに対してプログラムを実行し, 図 6.2a のグラフと同様なグラフを描いて下さい.
3. 本章で紹介したブレゼンハム指数増加アルゴリズムをパーソナルコンピュータ上で動作するようにして下さい. $y_0 = 0, y_1 = 100, D_Y = 100$ とし, D_X の値には 4 と 512 それぞれに対してプログラムを実行し, 図 6.2a のグラフと同様なグラフを描いて下さい.

第 7 章

LED の点灯 – 基礎編

本章では第 4 章で作成した LED 点灯装置を利用して, 様々な点灯パターンを実現するファームウエアを紹介します. そして後の第 8 章ではさらに高度で複雑な点灯を行うファームウエアを紹介します.
本章では以下の LED 点灯ファームウエアを紹介します.

- LED 風点滅
  点灯のオンとオフを交互に繰り替えして, 点滅する点灯パターンを作ります. 点灯と消灯は LED のごとく一瞬で輝度が変化します. 有限状態機械に基づく設計と実装の骨格を実例を通して学びます.
- 電球風点灯
  LED を光源に使うのですが, 電球を使用しているかのように点灯の際はじわっと点灯を, 消灯の際はゆっくり消灯をする輝度の制御を行います. 輝度の変化にはブレゼンハム指数アルゴリズムを使用します. 特に LED には電球色のものを使うと実感的です.
- 電球風点滅
  点灯のオンとオフを交互に繰り返して点滅する点灯パターンを作ります. 電球を使用しているかのように点灯の際はじわっと点灯, 消灯の際はゆっくり消灯します. 輝度の変化にはブレゼンハム指数アルゴリズムを使用します.

本章で紹介するファームウエアの特徴は以下の通りです.

- 有限状態機械に基づく設計例
  動作が明解で理解が容易です.

本章で学ぶ内容は以下の通りです.

- 簡単な状態遷移を行う有限状態機械の設計
- ブレゼンハムアルゴリズムの応用

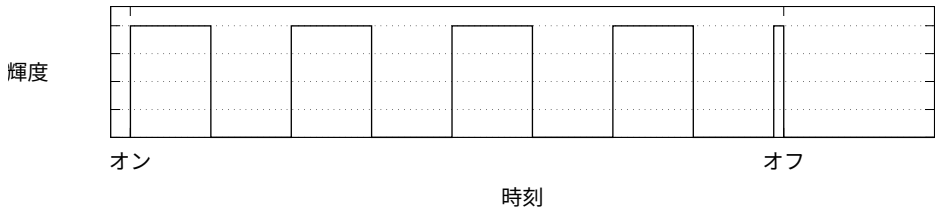


図 7.1: LED 風点滅の輝度変化

7.1　LED 風点滅

このファームウエアでは図 7.1 に示すように一定の時間間隔で LED を点滅します. グラフの横軸は時間を, 縦軸は輝度を, それぞれ表します. 点滅の間隔はパラメータ用の可変抵抗器 PARAM で設定します.

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 7.1: 【ソースファイル一覧】 LED 風点滅

初	頁	番号	ファイル名	内容	行数
	114	5.1	main.c	メイン関数	23
✓	137	7.1	LED-Light-Winker-LED.c	LED 点滅	77
	114	5.2	io.h	I/O ピン割付: ヘッダ	13
	114	5.3	mcr.h	MCR・周辺モジュール: ヘッダ	11
	116	5.4	mcr.c	MCR・周辺モジュール: 実装	99
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76
				計 308 行 (うち初出 77 行, 既出 231 行)	

初出のソースファイル LED-Light-Winker-LED.c を以下に示し, 説明は後ほど行います. 既出のソースファイルは以前の章を参照して下さい.

ソースファイル LED-Light-Winker-LED.c (LED 点滅)

ソースファイル番号 7.1: LED-Light-Winker-LED.c

```
1  /*********************************************
2   * LEDウインカー風
3   *********************************************/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8  #include "io.h"
9
10 /*
11  * 実現方法:
12  *   一定時間間隔で点灯/消灯を切り替える
13  */
14
15 /* 状態機械 */
16 enum fsm_state_e {
17   STATE_Q0 = 0, /* 停止期間 */
18   STATE_Q1,     /* 点灯期間 */
19   STATE_Q2,     /* 消灯期間 */
20 };
21
22 void Light(void)
23 {
24   /* 状態 */
25   enum fsm_state_e fsm_state = STATE_Q0;
26   /* 点滅時間計測用 */
27   uint16_t tmr = 0;
28   /* 主ループ */
29   for (;;) {
30     __delay_ms(10);
31     CLRWDT();
32     /* On/Offスイッチ*/
```

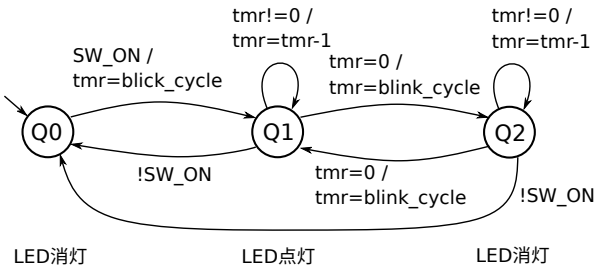


図 7.2: LED 風点滅の状態遷移図

```
33      int8_t sw_on = SW_ON();
34      /* 点滅周期をパラメータボリュームから読む */
35      uint16_t blink_cycle = 5 + (ADC_Read(ADC_CH_PARAM) >> 2);
36      /* 現在輝度 */
37      uint16_t intensity_current = 0;
38      /* 有限状態機械 */
39      switch (fsm_state) {
40      case STATE_Q0: /* 停止期間 */
41        if (sw_on) {
42          fsm_state = STATE_Q1;
43          tmr = blink_cycle;
44        }
45        intensity_current = 0;
46        break;
47      case STATE_Q1: /* 点灯期間 */
48        if (sw_on) {
49          if (tmr == 0) {
50            fsm_state = STATE_Q2;
51            tmr = blink_cycle;
52          } else {
53            tmr = tmr - 1;
54          }
55        } else {
56          fsm_state = STATE_Q0;
57        }
58        intensity_current = INTENSITY_MAX;
59        break;
60      case STATE_Q2: /* 消灯期間 */
61        if (sw_on) {
62          if (tmr == 0) {
63            fsm_state = STATE_Q1;
64            tmr = blink_cycle;
65          } else {
66            tmr = tmr - 1;
67          }
68        } else {
69          fsm_state = STATE_Q0;
70        }
71        intensity_current = 0;
72        break;
73      }
74      /* 輝度出力 */
75      LED_INTENSITY(intensity_current);
76    }
77  }
```

このソースファイル内では図 7.2 に示す状態遷移図の有限状態機械が実現されています. 状態間の矢印にはラベルが添えられてあり, 「/」の左側には状態遷移の条件を, 右側には状態遷移時の動作を, それぞれ記しています. この描き方を図 7.3 で示します.

点滅時間間隔を決めるパラメータは可変抵抗器から読み取り, 点滅時間の経過を変数 tmr で計測します. 実際にはこの変数でループ本体の繰り返し回数を数えるだけですが, ディレイ関数により繰り返し 1 回を 10 ミリ



(a) 状態遷移 (条件部と動作部あり)　**(b)** 状態遷移 (条件部のみ)

(c) 状態遷移 (動作部のみ)　**(d)** 状態遷移 (条件部・動作部なし)

図 7.3: 状態遷移の記述方法

秒としています.

- 状態 Q0 (初期状態) — 停止
  輝度がちょうど 0 の状態で, この状態にある限り輝度は 0 であり続けます. もしアクセサリ制御入力がオンになると状態 Q1 に遷移します.
- 状態 Q1 — 点灯期間
  LED 点滅中の点灯期間にこの状態をとります. 輝度を最大にして LED を点灯します.
  - アクセサリ制御入力がオンの場合
    変数 `tmr` を用いた点灯期間の終了判定を行います. もし点灯期間が終了すれば状態 Q2 に遷移して消灯期間に入ります. そうでなければ時間の経過を勘定します.
  - アクセサリ制御入力がオフの場合
    状態 Q0 に遷移して消灯します.
- 状態 Q2 — 消灯期間
  LED 点滅中の消灯期間にこの状態をとります. 輝度を 0 にして LED を消灯します.
  - アクセサリ制御入力がオンの場合
    変数 `tmr` を用いた消灯期間の終了判定を行います. もし消灯期間が終了すれば状態 Q1 に遷移して点灯期間に入ります. そうでなければ時間の経過を勘定します.
  - アクセサリ制御入力がオフの場合
    状態 Q0 に遷移して消灯します.

7.2　電球風点灯

このファームウエアではブレゼンハム指数アルゴリズムの使用により, 電球のような点灯・消灯を LED で実現します. レトロなレイアウトの夜景に組み込めば実感的な夜景となるでしょう.

輝度の制御は以下の通りです.

- アクセサリ制御入力がオン
  輝度を 0 から徐々に上げていきます.
- アクセサリ制御入力がオフ
  輝度を徐々に 0 へと下げていきます.

輝度の増減の度合いはパラメータ用の可変抵抗器 PARAM で設定します.

- パラメータを 0 にすると一瞬で点灯・消灯となります. これは時定数が 0 の場合です.
- パラメータを大きくしてゆくほど時定数は大きくなり, 点灯や消灯にかかる時間が長くなります.

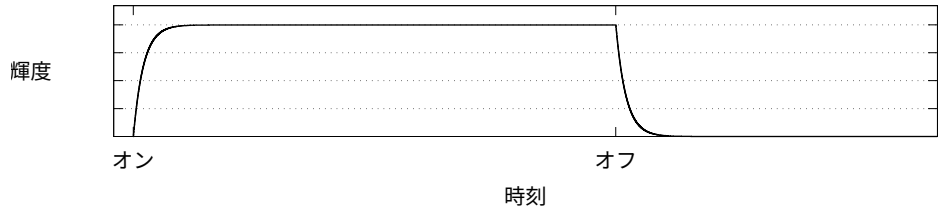


図 7.4: 電球風点灯の輝度変化

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 7.2: 【ソースファイル一覧】 電球風点灯

初	頁	番号	ファイル名	内容	行数
	114	5.1	main.c	メイン関数	23
✓	140	7.2	LED-Light-Bulb.c	電球風点灯	89
	134	6.3	incr-exp.h	指数増加アルゴリズム: ヘッダ	8
	134	6.4	incr-exp.c	指数増加アルゴリズム: 実装	77
	114	5.2	io.h	I/O ピン割付: ヘッダ	13
	114	5.3	mcr.h	MCR・周辺モジュール: ヘッダ	11
	116	5.4	mcr.c	MCR・周辺モジュール: 実装	99
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76
				計 405 行 (うち初出 89 行, 既出 316 行)	

初出のソースファイル LED-Light-Bulb.c は以下の通りで, 後ほど説明を行います. 既出のソースファイルは以前の章を参照して下さい.

ソースファイル LED-Light-Bulb.c (電球風点灯)

ソースファイル番号 7.2: LED-Light-Bulb.c

```
1  /*********************************************
2   * 電球風 On/Off
3   *********************************************/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8  #include "io.h"
9  #include "incr-exp.h"
10
11 /*
12  * 実現方法: 指数関数 (CR充電曲線の形) で輝度を変化
13  * 時定数 0..255
14  *   輝度63%になるまでの最大単位時間:
15  *     tconst [Ticks]
16  *   輝度63%になるまでの最大実時間:
17  *     tconst [Ticks] * 10 [mSec / Tick]
18  *       = 2550 [mSec] = 2.55 [Sec]
19  */
20
21 /* 時定数: オンのとき */
22 #define TCONST_ON(v)    ((v) >> 3)
```

```
23 /* 時定数: オフのとき */
24 #define TCONST_OFF(v)   ((v) >> 5) /* 消灯時の時定数は小さく */
25 /* 計算した輝度値がこれ以下のときは輝度を0にする */
26 #define INTENSITY_EPS   2
27
28 /* 有限状態機械 */
29 enum fsm_state_e {
30   STATE_Q0 = 0, /* 消灯 */
31   STATE_Q1,     /* 点灯 */
32 };
33
34 void Light(void)
35 {
36   /* 状態 */
37   enum fsm_state_e fsm_state = STATE_Q0;
38   /* 現在輝度 */
39   uint16_t intensity_current = 0;
40   incr_exp_init(0);
41   /* 主ループ */
42   for (;;) {
43     __delay_ms(10);
44     CLRWDT();
45     /* On/Offスイッチを読む */
46     int8_t sw_on = SW_ON();
47     /* パラメータボリュームを読む (10ビット: 0..1023) */
48     adc_val_t adc_val_param = ADC_Read(ADC_CH_PARAM);
49     uint16_t tconst_on  = TCONST_ON(adc_val_param);
50     uint16_t tconst_off = TCONST_OFF(adc_val_param);
51     /* 有限状態機械 */
52     switch (fsm_state) {
53     case STATE_Q0: /* 消灯状態 */
54       /* 状態遷移 */
55       if (sw_on) {
56         fsm_state = STATE_Q1;
57         incr_exp_init(intensity_current);
58       }
59       /* 輝度計算 */
60       if (tconst_off == 0) {
61         intensity_current = 0;
62       } else {
63         intensity_current
64           = incr_exp_next(tconst_off, 0);
65       }
66       break;
67     case STATE_Q1: /* 点灯状態 */
68       /* 状態遷移 */
69       if (!sw_on) {
70         fsm_state = STATE_Q0;
71         incr_exp_init(intensity_current);
72       }
73       /* 輝度計算 */
74       if (tconst_on == 0) {
75         intensity_current = INTENSITY_MAX;
76       } else {
77         intensity_current
78           = incr_exp_next(tconst_on, INTENSITY_MAX);
79       }
80       break;
81     }
82     /* 輝度出力 */
83     if (intensity_current <= INTENSITY_EPS) {
84       /* 輝度値が十分に小さいと0とみなす */
85       intensity_current = 0;
86     }
87     LED_INTENSITY(intensity_current);
88   }
89 }
```

このソースファイル内では図 7.5 に示す状態遷移図の有限状態機械が実現されています．実はこの状態遷移

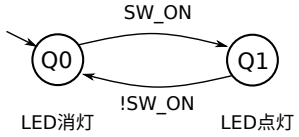


図 7.5: 電球風点灯の状態遷移図

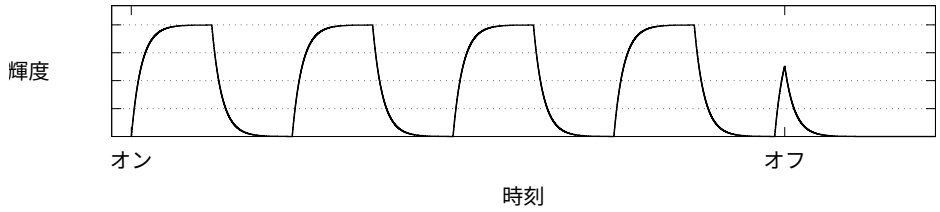


図 7.6: 電球風点滅の輝度変化

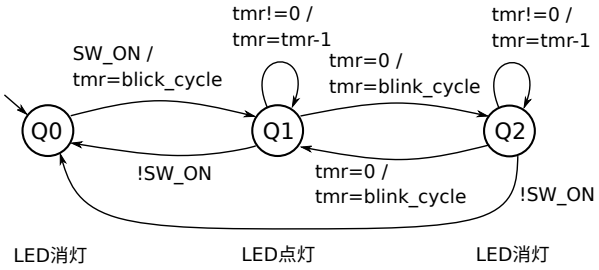


図 7.7: 電球風点滅の状態遷移図

図は図 5.8 と同一です. 2 つの違いは各状態で輝度の増加や減少を行うかどうかであり, それらの詳細は状態遷移図には描いていません. 状態遷移図に描くものを理解に必要なものに限定することで動作の骨格の把握は容易となり, 設計の全体像の理解に有用です.

- 状態 Q0 (初期状態) — 消灯
  輝度がちょうど 0 または輝度が減少中の状態です. アクセサリ制御入力がオンになると状態 Q1 に遷移します. アクセサリ制御入力がオフのままなら輝度を更新します. 輝度減少により輝度が 0 になれば, この状態にある限り輝度は 0 であり続けます.
- 状態 Q1 — 点灯
  輝度が最大または輝度が増加中の状態です. アクセサリ制御入力がオフになると状態 Q0 に遷移します. アクセサリ制御入力がオフのままなら輝度を更新します. 輝度増加により輝度が最大になれば, この状態にある限り輝度は最大であり続けます.

7.3　電球風点滅

このファームウエアでは電球風の点灯・消灯に加えて点滅の機能も持たせています. 点滅の間隔はパラメータ用の可変抵抗器 PARAM で設定しますが, 輝度の増減の度合いはプログラム内部で固定しています.

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 7.3: 【ソースファイル一覧】 電球風点滅

初	頁	番号	ファイル名	内容	行数
	114	5.1	main.c	メイン関数	23
✓	143	7.3	LED-Light-Winker-Bulb.c	電球風点滅	90
	134	6.3	incr-exp.h	指数増加アルゴリズム: ヘッダ	8
	134	6.4	incr-exp.c	指数増加アルゴリズム: 実装	77
	114	5.2	io.h	I/O ピン割付: ヘッダ	13
	114	5.3	mcr.h	MCR・周辺モジュール: ヘッダ	11
	116	5.4	mcr.c	MCR・周辺モジュール: 実装	99
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76
				計 406 行 (うち初出 90 行, 既出 316 行)	

初出のソースファイル LED-Light-Winker-Bulb.c は以下の通りで, 後ほど説明を行います. 既出のソースファイルは以前の章を参照して下さい.

ソースファイル LED-Light-Winker-Bulb.c (電球風点滅)

ソースファイル番号 7.3: LED-Light-Winker-Bulb.c

```
1  /*********************************************
2   * 電球ウインカー風
3   *********************************************/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8  #include "io.h"
9  #include "incr-exp.h"
10
11 /*
12  * 指数関数（CR充電曲線の形）で輝度が変化
13  */
14
15 /* 時定数 0..255 */
16 #define TCONST_ON    6
17 #define TCONST_OFF   4
18
19 /* 有限状態機械 */
20 enum fsm_state_e {
21   STATE_Q0 = 0, /* 停止期間 */
22   STATE_Q1,     /* 点灯期間 */
23   STATE_Q2,     /* 消灯期間 */
24 };
25
26 void Light(void)
27 {
28   /* 状態 */
29   enum fsm_state_e fsm_state = STATE_Q0;
30   /* 現在輝度 */
31   uint16_t intensity_current = 0;
32   incr_exp_init(intensity_current);
33   /* 点滅時間計測用 */
34   uint16_t tmr = 0;
35   /* 主ループ */
36   for (;;) {
37     CLRWDT();
```

```
38      __delay_ms(10); /* 1サイクル 10 [mSec] */
39      /* On/Offスイッチ*/
40      int8_t sw_on = SW_ON();
41      /* 点滅周期をパラメータボリュームから読む */
42      uint16_t blink_cycle = 5 + (ADC_Read(ADC_CH_PARAM) >> 2);
43      /* 有限状態機械 */
44      switch (fsm_state) {
45      case STATE_Q0: /* 停止期間 */
46        if (sw_on) {
47          fsm_state = STATE_Q1;
48          tmr = blink_cycle;
49          incr_exp_init(intensity_current);
50        } else {
51          intensity_current
52            = incr_exp_next(TCONST_ON, 0);
53        }
54        break;
55      case STATE_Q1: /* 点灯期間 */
56        if (sw_on) {
57          if (tmr == 0) {
58            fsm_state = STATE_Q2;
59            tmr = blink_cycle;
60            incr_exp_init(intensity_current);
61          } else {
62            tmr = tmr - 1;
63          }
64        } else {
65          incr_exp_init(intensity_current);
66          fsm_state = STATE_Q0;
67        }
68        intensity_current
69          = incr_exp_next(TCONST_ON, INTENSITY_MAX);
70        break;
71      case STATE_Q2: /* 消灯期間 */
72        if (sw_on) {
73          if (tmr == 0) {
74            fsm_state = STATE_Q1;
75            tmr = blink_cycle;
76            incr_exp_init(intensity_current);
77          } else {
78            tmr = tmr - 1;
79          }
80        } else {
81          fsm_state = STATE_Q0;
82        }
83        intensity_current
84          = incr_exp_next(TCONST_OFF, 0);
85        break;
86      }
87      /* 輝度出力 */
88      LED_INTENSITY(intensity_current);
89    }
90  }
```

この有限状態帰機械の状態遷移図を図 7.7 に示します. この状態遷移図と図 7.2 は同一で, 2 つのファームウエアの根本的な構造は同じであると分かります.

課題

1. 電球風点灯のファームウエアでは, ブレゼンハムの指数アルゴリズムを使用して輝度を増加しました. この代わりに線形アルゴリズムを使用したファームウエアを開発して下さい. 実感的な点灯でしょうか. 指数アルゴリズムと線形アルゴリズムとの場合で比較して下さい.
2. 電球風点灯のファームウエアでは LED の輝度変化を電球の輝度変化に似せることで, LED を使用しながらも電球らしい点灯を実現しました. しかし, よく観察すると微妙に違和感が残ります. 実際の電球では, 電気の輝度が低いときはフィラメントの温度が低く赤みを帯びた色をしており, 輝度が高いときは

フィラメントの温度が高く白みを帯びた色をしています. しかし本章の回路とファームウエアでは単一色 LED を用いているためにこれが実現できていません. そこで RGB LED を使用することで色の変化も模倣して, より実感的な電球風点灯を実現する回路とファームウエアを開発して下さい.

第 8 章

LED の点灯 – 発展編

本章では第 4 章で作成した LED 点灯装置を利用して, 第 7 章よりもさらに高度で複雑な点灯制御を行うファームウエアを紹介します. 特に輝度の変化にランダム性を持たせて, レイアウトの夜景の演出で活躍する以下の点灯パターンを作成します.

- ネオン管風点灯
  夜の街の裏路地のネオン管の灯りに似た LED 点灯を行います. ランダム性を持つ点灯を実現します.
- 蝋燭(ろうそく)風点灯
  蝋燭風の炎のゆらめきに似た LED 点灯を行います. 蒸気機関車の火室やキャンプファイヤーの炎としても使用できます.
- 蛍光灯風点灯
  蛍光灯の点灯の際のチラつきを実現した LED 点灯を行います.

本章のファームウエアの特徴は以下の通りです.

- 有限状態機械に基づく設計
  複雑な動作内容でも明解で理解が容易で, 改造の出発点として流用可能です.

本章で学ぶ内容は以下の通りです.

- 擬似乱数生成器 (xorshift 法)
- 固定小数点数
- より複雑な有限状態機械の設計例

8.1 ネオン管風点灯

ネオン管風の点灯を実現します. 高輝度の期間と低輝度の期間を繰り返しますが, 期間の長さにランダム性を入れます. 低輝度の期間の輝度の値は実際に点灯する様子を見ながら, ネオン管の感じに最も近くなるように決めました. 図 8.1 に示すグラフが輝度の変化の例です.

本ファームウエアを構成するソースファイル一覧を以下に示します.

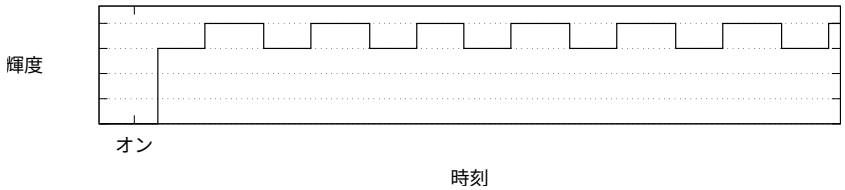


図 8.1: ネオン管風点灯の輝度変化

表 8.1: 【ソースファイル一覧】 ネオン管風点灯

初	頁	番号	ファイル名	内容	行数
	114	5.1	main.c	メイン関数	23
✓	147	8.1	LED-Light-Neon.c	ネオン管風点灯	79
✓	148	8.2	random.h	疑似乱数: ヘッダ	7
✓	149	8.3	random.c	疑似乱数: 実装	26
	114	5.2	io.h	I/O ピン割付: ヘッダ	13
	114	5.3	mcr.h	MCR・周辺モジュール: ヘッダ	11
	116	5.4	mcr.c	MCR・周辺モジュール: 実装	99
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76
				計 343 行 (うち初出 112 行, 既出 231 行)	

ソースファイル LED-Light-Neon.c (ネオン管風点灯: 実装)

高輝度の期間と低輝度の期間を繰り返しますが, 高輝度の期間の長さにのみランダム性を入れています. わずかこれだけのことですが, ネオン管のような印象を受ける点灯になります.

ソースファイル番号 8.1: LED-Light-Neon.c

```
1  /*****************************************************
2   * ネオン管風
3   *****************************************************/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8  #include "io.h"
9  #include "incr-lin.h"
10 #include "random.h"
11
12 /*
13  *   高速で輝度を変化させる.
14  *   輝度維持時間はランダムにして, ネオン風に微妙にチラつきを行う
15  */
16
17 /* 有限状態機械 */
18 enum fsm_state_e {
19   STATE_Q0 = 0, /* 消灯 */
20   STATE_Q1,     /* 低輝度点灯 */
21   STATE_Q2,     /* 高輝度点灯 */
22 };
23
24 #define INTENSITY_LOW           768
25 #define T_LOW       3
26 #define T_HIGH      3
27 #define T_RAND      0x01
28
29 void Light(void)
30 {
31   /* 状態 */
32   enum fsm_state_e fsm_state = STATE_Q0;
33   /* 現在輝度 */
34   uint16_t intensity_current = 0;
35   /* 輝度維持時間 */
36   int16_t tmr = 0;
37   /* 主ループ */
```

```
38    for (;;) {
39      CLRWDT();
40      __delay_ms(10);
41      /* On/Offスイッチ */
42      int8_t sw_on = SW_ON();
43      /* 有限状態機械 */
44      switch (fsm_state) {
45      case STATE_Q0: /* 消灯 */
46        intensity_current = 0;
47        tmr = 0;
48        if (sw_on) {
49          fsm_state = STATE_Q1;
50          tmr = T_LOW;
51        }
52        break;
53      case STATE_Q1: /* 低輝度点灯 */
54        intensity_current = INTENSITY_LOW;
55        if (!sw_on) {
56          fsm_state = STATE_Q0;
57        } else if (tmr == 0) {
58          fsm_state = STATE_Q2;
59          tmr = T_HIGH + (random_get16() & T_RAND);
60        } else {
61          tmr--;
62        }
63        break;
64      case STATE_Q2: /* 高輝度点灯 */
65        intensity_current = INTENSITY_MAX;
66        if (!sw_on) {
67          fsm_state = STATE_Q0;
68        } else if (tmr == 0) {
69          fsm_state = STATE_Q1;
70          tmr = T_LOW;
71        } else {
72          tmr--;
73        }
74        break;
75      }
76      /* 輝度出力 */
77      LED_INTENSITY(intensity_current);
78    }
79  }
```

ソースファイル random.h (擬似乱数: ヘッダ)

擬似乱数に関する関数のプロトタイプ宣言を行います.

ソースファイル番号 8.2: random.h

```
1  #if !defined(__RANDOM_H__)
2  #define __RANDOM_H__
3
4  extern uint16_t random_get16(void);
5  extern uint8_t  random_get8(void);
6
7  #endif /*__RANDOM_H__*/
```

ソースファイル random.c (擬似乱数: 実装)

xorshift 法による擬似乱数生成を行います. xorshift 法の大きな特徴は使用する演算が排他的論理和とシフトだけである点にあります. よく使われる擬似乱数生成法のひとつに乗算を用いた線形合同法があげられます. しかし PIC マイコン 16F シリーズのように乗算命令を持たないマイコンでは線形合同法は高速に実行できません. 一方 xorshift 法で使用する演算は PIC マイコン 16F シリーズをはじめとしてマイコンのほとんどすべてが備えている排他的論理和とシフト命令ですので, 高速に実行できる点が大きな特徴です.

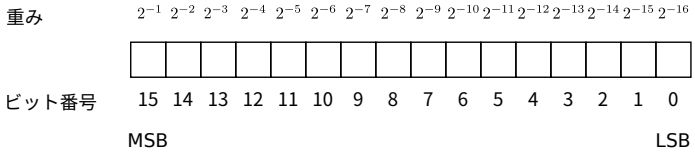


図 8.2: 固定小数実数 Q0.16 形式

ソースファイル番号 8.3: random.c

```
1  /*-----------------------------------------------------------
2   * xorshift法による疑似乱数生成
3   * Reference: http://www.arklyffe.com/main/2010/08/29/xorshift-pseudorandom-
        number-generator/
4   *-----------------------------------------------------------*/
5  #include <xc.h>
6  #include <stdint.h>
7  #include "random.h"
8
9  uint16_t random_get16(void)
10 {
11   static uint16_t y = 1;
12   y = y ^ (y << 1);
13   y = y ^ (y >> 5);
14   y = y ^ (y << 2);
15   return y;
16 }
17
18 uint8_t random_get8(void)
19 {
20   static uint8_t y = 1;
21   y = (uint8_t) (y ^ (y << 1));
22   y = (uint8_t) (y ^ (y >> 1));
23   y = (uint8_t) (y ^ (y << 2));
24   return y;
25
26 }
```

8.2　固定小数点実数

本章で紹介するファームウエアでは実数演算を行いますが, 整数演算を流用して実現できる実数の表現方法である固定小数点数を紹介します. 特に本章では Q0.16 形式の固定小数点実数を用いますので, この場合に限って説明します.

データ表現

Q0.16 形式とは合計で 16 のビットにより固定小数実数を表現する形式で, 整数部分に 0 ビット, 小数部分に 16 ビットを使用します. 整数部分にはまったくビットを使用しないので, 表現される数値は 0 以上 1 未満の範囲の値です. 一般に Qm.n 形式は, 整数部分に m ビット, 小数部分に n ビットの合計 $m+n$ ビットを使用して実数を表現する形式です.

固定小数実数 Q0.16 形式で使用する 16 のビットを図 8.2 に示す通り $x_{15}, x_{14}, x_{13}, ..., x_0$ とします. ただし x_{15} が最上位ビット (most significant bit; MSB) でその重みは $2^{-1} = 0.5$ であり, x_0 が最下位ビット (least significant bit; LSB) でその重みは 2^{-16} です. するとこれら 16 のビットが表現する実数値 x は以下の式で与えられます.

$$\begin{aligned} x &= 2^{-1}x_{15} + 2^{-2}x_{14} + 2^{-3}x_{13} + \cdots + 2^{-16}x_0 \\ &= (2^{15}x_{15} + 2^{14}x_{14} + 2^{13}x_{13} + \cdots + 2^0 x_0)/2^{16} \\ &= (\textstyle\sum_{i=0}^{15} 2^i x_i)/2^{16} \end{aligned}$$

この式から Q0.16 形式を用いて表現可能な実数値は, 0 以上 1 未満の範囲であることが分かります.

固定小数点実数 x の表現で使う 16 のビットを整数と見做したときの値 X は以下の式で与えられます.

$$\begin{aligned} X &= 2^{15}x_{15} + 2^{14}x_{14} + 2^{13}x_{13} + \cdots + 2^{0}x_{0} \\ &= \sum_{i=0}^{15} 2^{i}x_{i} \end{aligned}$$

以上のことから, 16 のビット $x_{15}, x_{14}, x_{13}, ..., x_0$ を整数と見做したときの値 X と固定小数点実数と見做したときの値 x には $x = X/2^{16}$ の関係が成立することが分かります. この関係を利用してプログラム中では整数の定数値を用いて固定小数点実数の定数値を表現できます. たとえば整数の定数 0x8000 は固定小数点実数の定数 0.5 を表すことになります. なぜなら整数の値より $0x8000 = 2^{15} = \sum_{i=0}^{15} 2^{i}x_{i}$ とすれば, $x_{15} = 1$ で $x_i = 0$ $(i \neq 15)$ なので, 固定小数点実数の値は $x = (\sum_{i=0}^{15} 2^{i}x_{i})/2^{16} = 2^{15}/2^{16} = 2^{-1} = 0.5$ となるからです.

加算

2 つの Q0.16 固定小数実数 x と y の加算 $x + y$ は, 整数の加算 $X + Y$ により計算できます. ここで x および y の表現で使用する 16 のビットを $x_{15}, ..., x_0$ および $y_{15}, ..., y_0$ とし, $X = \sum_{i=0}^{15} 2^{i}x_{i}$ および $Y = \sum_{i=0}^{15} 2^{i}y_{i}$ とします. すると以下の関係式から固定小数実数の加算 $x + y$ は整数の加算 $X + Y$ で計算ができることが分かります.

$$\begin{aligned} (x + y) &= (\textstyle\sum_{i=0}^{15} 2^{i}x_{i})/2^{16} + (\sum_{i=0}^{15} 2^{i}y_{i})/2^{16} \\ &= ((\textstyle\sum_{i=0}^{15} 2^{i}x_{i}) + (\sum_{i=0}^{15} 2^{i}y_{i}))/2^{16} \\ &= (X + Y)/2^{16} \end{aligned}$$

乗算

2 つの Q0.16 固定小数実数の乗算 $x \cdot y$ は, 整数の乗算を用いて $(X \cdot Y)/2^{16}$ の式で計算できます.

$$\begin{aligned} (x \cdot y) &= ((\textstyle\sum_{i=0}^{15} 2^{i}x_{i})/2^{16}) \cdot ((\sum_{i=0}^{15} 2^{i}y_{i})/2^{16}) \\ &= (X/2^{16}) \cdot (Y/2^{16}) \\ &= (X \cdot Y)/2^{16} \end{aligned}$$

ただしこの計算の途中では 16 ビットと 16 ビットの整数演算を行いますから, オーバーフローが生じないようにいったん 32 ビットに拡張して乗算を行う必要がある点には十分注意して下さい.

8.3　蝋燭風点灯

本ファームウエアは蝋燭の炎のゆらめきに似た LED 点灯を行います. 灯りのゆらめき度合いの強弱はパラメータ用の可変抵抗器で設定します. 灯りのゆらめきの計算には, 1/f ゆらぎを実現する手法のひとつとして知られている間欠カオス法を使用します. 間欠カオス法とは実数変数 x (ただし $0 \leq x \leq 1$) に対する以下の計算の繰り返しのことであり, これにより得られる x の値の系列は 1/f ゆらぎとなることが知られています.

$$x = \begin{cases} x + 2(x^2) & x < 0.5 \text{ の場合} \\ x - 2(x-1)^2 & \text{それ以外の場合} \end{cases}$$

この式で注意すべきは, x の値がちょうど 0 または 1 の場合には次の x の値も同じ値になってしまい, 永遠に同じ値に固着してしまう点です. 自然界の 1/f ゆらぎでは同じ値への固着は発生しませんが, コンピュータによる上の式では発生します. そこで本ファームウエアでは x の値に毎回わずかな擬似乱数を加えることで値の固着を防ぎます.

間欠カオス法そのままで LED 輝度を変化させると蝋燭のような感じは出ません. そこでこの点を本ファームウエアでは工夫を行い, 灯りのゆらぎの強弱の度合いを可変抵抗器で調整できる機能を持たせます. 以下ではゆらぎの強弱の度合いを安定度と呼ぶことにします. 安定度が低いと灯りのゆらぎは強くて風に吹かれて蝋燭の炎が揺らめいている様に似ていますし, 安定度が高いと灯りのゆらぎは弱くて蝋燭の炎が静かに室内を照らし

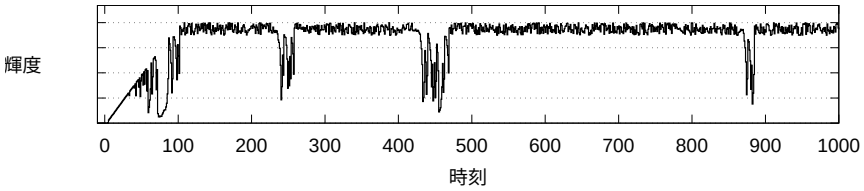


図 8.3: 蝋燭風点灯の輝度変化の例

ている様に似ています. 図 8.3 に示すグラフが輝度の変化の例です.

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 8.2: 【ソースファイル一覧】 蝋燭風点灯

初	頁	番号	ファイル名	内容	行数
	114	5.1	`main.c`	メイン関数	23
✓	151	8.4	`LED-Light-Candle.c`	蝋燭風点灯	132
	148	8.2	`random.h`	疑似乱数: ヘッダ	7
	149	8.3	`random.c`	疑似乱数: 実装	26
	129	6.1	`incr-lin.h`	線形増加アルゴリズム: ヘッダ	9
	130	6.2	`incr-lin.c`	線形増加アルゴリズム: 実装	78
	114	5.2	`io.h`	I/O ピン割付: ヘッダ	13
	114	5.3	`mcr.h`	MCR・周辺モジュール: ヘッダ	11
	116	5.4	`mcr.c`	MCR・周辺モジュール: 実装	99
	117	5.5	`adc.h`	ADC モジュール: ヘッダ	9
	118	5.6	`adc.c`	ADC モジュール: 実装	76
				計 483 行 (うち初出 132 行, 既出 351 行)	

ソースファイル LED-Light-Candle.c (蝋燭風点灯)

本ソースファイルで蝋燭風の点灯を実現します. 間欠カオス法の変数 x は値の範囲が 0 以上 1 以下の実数ですが, C 言語の浮動小数点実数は用いずに固定小数点実数 Q0.16 で表現します. ただし C 言語には固定小数点実数の機能はないので, 16 ビット整数による表現と演算を行います.

ソースファイル番号 8.4: LED-Light-Candle.c

```
1  /*********************************************
2   * 蝋燭風
3   *********************************************/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8  #include "io.h"
9  #include "incr-lin.h"
10 #include "random.h"
11
12 /*
13  * 間欠カオス法による1/fゆらぎを用いて蝋燭の揺らぎを模倣
14  */
15
16 /* 有限状態機械 */
```

```
17  enum fsm_state_e {
18    STATE_Q0 = 0, /* 消灯 */
19    STATE_Q1,     /* 点灯 */
20  };
21
22  #define CANDLE_ENV_SPEED_DX    50
23  #define CANDLE_ENV_SPEED_DY   512
24
25  static void     candle_init(void);
26  static uint16_t candle_next(uint16_t stability);
27  static void     intermittent_chaos_init(void);
28  static uint16_t intermittent_chaos_get(void);
29
30  void Light(void)
31  {
32    /* 状態 */
33    enum fsm_state_e fsm_state = STATE_Q0;
34    /* 現在輝度 */
35    uint16_t intensity_current = 0;
36    /* 蝋燭の輝度 */
37    candle_init();
38    /* 主ループ */
39    for (;;) {
40      __delay_ms(20);
41      CLRWDT();
42      /* On/Offスイッチを読む */
43      int8_t sw_on = SW_ON();
44      /* パラメータボリュームを読む (10ビット: 0..1023) */
45      adc_val_t adc_val_param = ADC_Read(ADC_CH_PARAM);
46      /* 蝋燭炎安定パラメータ (0..255) */
47      uint16_t stability = adc_val_param >> 2;
48      /* 輝度包絡線 */
49      uint16_t intensity_enverope = 0;
50      /* 現在輝度 */
51      intensity_current = candle_next(stability) >> 6;
52      /* 有限状態機械 */
53      switch (fsm_state) {
54      case STATE_Q0: /* 消灯状態 */
55        /* 消灯 */
56        intensity_enverope = 0;
57        /* 状態遷移 */
58        if (sw_on) {
59          fsm_state = STATE_Q1;
60          /* 輝度包絡線 */
61          incr_lin_init(0);
62        }
63        break;
64      case STATE_Q1: /* 点灯状態 */
65        /* 状態遷移 */
66        if (!sw_on) {
67          fsm_state = STATE_Q0;
68        }
69        /* 輝度包絡線計算: すこしづつ明るく */
70        intensity_enverope
71          = incr_lin_next(CANDLE_ENV_SPEED_DX, CANDLE_ENV_SPEED_DY,
72                          INTENSITY_MAX);
73        break;
74      }
75      /* 輝度出力 */
76      if (intensity_current > intensity_enverope) {
77        /* 輝度を包絡線で制限する */
78        intensity_current = intensity_enverope;
79      }
80      LED_INTENSITY(intensity_current);
81    }
82  }
83
84  static uint16_t candle_stability_t = 0;
85
86  static void candle_init(void)
```

```
87  {
88    intermittent_chaos_init();
89    candle_stability_t = 0;
90  }
91
92  static uint16_t candle_next(uint16_t stability)
93  {
94    uint16_t v = 0;  /* 固定小数点数(Q0.16形式) */
95    if (candle_stability_t > 0) {
96      /* 現在の状態にとどまる */
97      candle_stability_t--;
98      /* 僅かにランダム性を入れる */
99      v = 0xE000 ^ (random_get16() >> 3);
100   } else {
101     /* 次の状態に移る */
102     v = intermittent_chaos_get();
103     if (v >= 0xF000) {
104       candle_stability_t = stability;
105     }
106   }
107   return v;
108 }
109
110 /* 間欠カオス - カオス状態はQ0.16形式の固定小数点数で表現 */
111
112 static uint16_t  intermittent_chaos_x = 0;
113
114 static void intermittent_chaos_init(void)
115 {
116   intermittent_chaos_x = 0;
117 }
118
119 static uint16_t intermittent_chaos_get(void)
120 {
121   uint32_t x = (uint32_t)intermittent_chaos_x;
122   uint16_t v = 0;
123   if (x < 0x8000) {
124     x = x + ((x * x) >> (16 - 1));
125   } else {
126     x = x - ((0x00010000 - x) * (0x00010000 - x) >> (16 - 1));
127   }
128   v = (uint16_t) x & 0xFFFF;
129   v = v ^ (random_get16() >> 4);  /* ランダム性を入れる */
130   intermittent_chaos_x = v;
131   return intermittent_chaos_x;
132 }
```

このプログラムを簡単に説明します. 基本的な原理は間欠カオス法の変数 x を計算し, その値に応じた輝度で LED を点灯することの繰り返しですが, 以下の 2 つの理由でプログラムは少し複雑になっています. ひとつめの理由は点灯開始の動作に工夫をしているためです. 点灯開始時にいきなり最高輝度で照らすのではなく, 蝋燭が点火されて少しずつ炎が大きくなってゆく様を実現しています. 炎が大きくなってゆく度合いはブレゼンハムアルゴリズムを用いて計算します. ふたつめの理由は灯りの安定度を可変抵抗器で変更可能としているためです. 安定度が低いときは間欠カオス法を頻繁に適用して次々と輝度を更新します. 安定度が高いときは輝度の高い状態を長く続けるようにします.

8.4　蛍光灯風点灯

グロー管を使用した蛍光灯風の点灯を実現します. 昔ながらのチラつきがある蛍光灯です. スイッチを入れてから少し時間が経った後にすこしちらついて点灯します. また可変抵抗器で蛍光灯の安定度を調節できる機能を持たせています. 安定度が高い場合は新品ですぐに点灯する蛍光灯の様子になりますし, 安定度が低い場合は古ぼけてなかなか点灯しない蛍光灯の様子になります.

本ファームウエアを構成するソースファイル一覧を以下に示します.

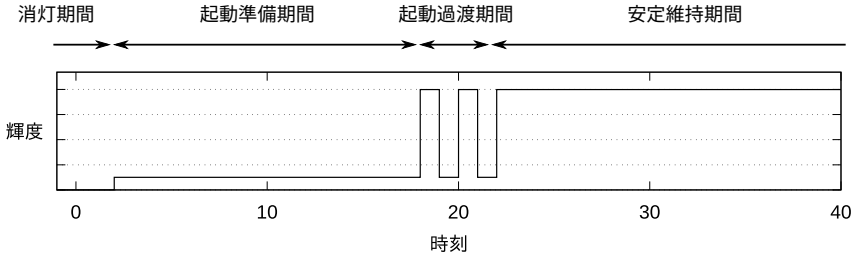


図 8.4: 蛍光灯風点灯の輝度変化期間

表 8.3: 【ソースファイル一覧】 蛍光灯風点灯

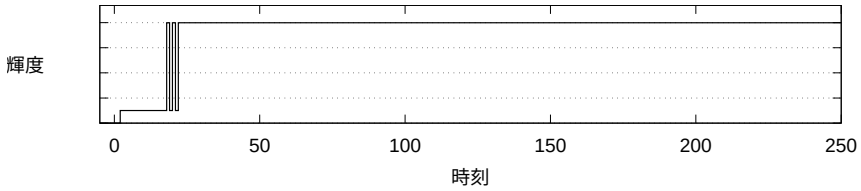
初	頁	番号	ファイル名	内容	行数
	114	5.1	`main.c`	メイン関数	23
✓	155	8.5	`LED-Light-FluoLamp.c`	蛍光灯風点灯	176
	148	8.2	`random.h`	疑似乱数: ヘッダ	7
	149	8.3	`random.c`	疑似乱数: 実装	26
	114	5.2	`io.h`	I/O ピン割付: ヘッダ	13
	114	5.3	`mcr.h`	MCR・周辺モジュール: ヘッダ	11
	116	5.4	`mcr.c`	MCR・周辺モジュール: 実装	99
	117	5.5	`adc.h`	ADC モジュール: ヘッダ	9
	118	5.6	`adc.c`	ADC モジュール: 実装	76
				計 440 行 (うち初出 176 行, 既出 264 行)	

ソースファイル `LED-Light-FluoLamp.c` (蛍光灯風点灯: 実装)

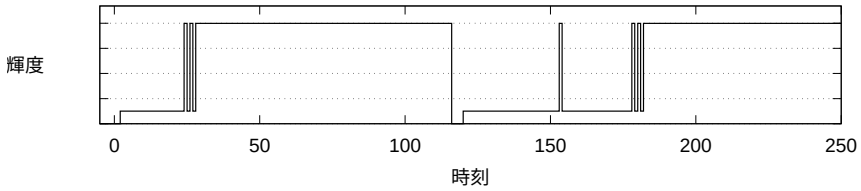
蛍光灯の輝度変化は若干複雑なうえ安定度の調整ができるようにしていますので, 少々複雑なプログラムとなっています. しかし有限状態機械による設計なので状態遷移図を見れば意外と簡単であることが分かります. 言葉での表現は実に困難なのですが, 蛍光灯の点灯の時の輝度の変化を言葉で表現すれば「ジー・パッパッ・ピカー」という感じかと思います. 有限状態機械はこの輝度の変化をモデル化し, 図 8.4 にこの輝度変化の様子を示します.

- 消灯期間
  スイッチが入れられていない期間で, 全く光っていません.
- 起動準備期間 (「ジー」)
  スイッチが入ってすぐの低輝度で点灯している期間です.
- 起動過渡期間 (「パッパッ」)
  高輝度と低輝度の点灯が数回に渡って交互に交代する期間です.
- 安定維持期間 (「ピカー」)
  高輝度で安定して点灯し続けている期間です.
- 安定崩壊期間
  古い蛍光灯が動作の不安定さのために消灯してしまう期間です.

おおよそ以上の変化を繰り返しながら点灯して蛍光灯の模倣をします. また可変抵抗器の設定により安定度



(a) 安定度が高い場合



(b) 安定度が低い場合

図 8.5: 安定度による違いによる蛍光灯風点灯の輝度変化

の高低を変化させることで, 新しい蛍光灯と古くなった蛍光灯の点灯の模倣も行います. 図 8.5a は新しい蛍光灯に対応した輝度の変化のグラフで, すぐに点灯して安定維持期間がずっと続きます. 図 8.5b は古い蛍光灯に対応した輝度の変化のグラフで, なかなか点灯せず, しかも安定維持期間は短くて点灯は不安定です. この実現方法は後で説明します.

ソースファイル番号 8.5: LED-Light-FluoLamp.c

```
1  /*************************************************
2   * グロー管式蛍光灯風
3   *************************************************/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8  #include "io.h"
9  #include "random.h"
10
11 /*
12  * 点灯のチラつきを状態遷移で実現.
13  * チラつき具合(「安定度」)はボリュームで設定
14  */
15
16 /* 有限状態機械 */
17 enum fsm_state_e {
18   STATE_Q0   = 0,  /* 消灯期間     */
19   STATE_Q1A,       /* 起動準備期間 */
20   STATE_Q1B,       /* 起動準備期間 */
21   STATE_Q2A,       /* 起動過渡期間 */
22   STATE_Q2B,       /* 起動過渡期間 */
23   STATE_Q3,        /* 安定維持期間 */
24   STATE_Q4,        /* 安定崩壊期間 */
25 };
26
27 uint8_t prob_p_1a = 0;
28 uint8_t prob_p_1b = 0;
29 uint8_t prob_p_2  = 0;
30 uint8_t prob_p_3  = 0;
31 uint8_t prob_p_4  = 0;
32
33 #define THR_X1    15
34 #define THR_X2A    1
35 #define THR_X2B    3
36 #define THR_X3     6
```

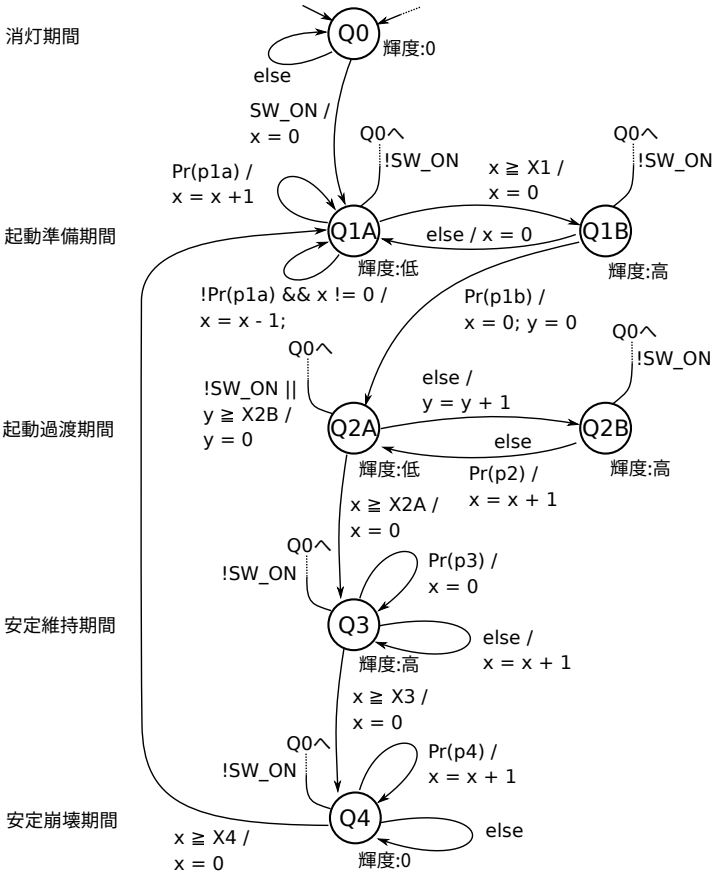


図 8.6: 蝋燭風点灯の状態遷移図. 記号 Pr(p) は確率 p で生起する事象を表す.

```
37  #define THR_X4      3
38
39  #define INTENSITY_0         0
40  #define INTENSITY_LOW       (INTENSITY_MAX >> 6)
41  #define INTENSITY_HIGH      (INTENSITY_MAX)
42
43  static void   read_params(void);
44  static int8_t with_prob(uint8_t ip);
45
46  void Light(void)
47  {
48    /* 状態 */
49    uint8_t fsm_state = STATE_Q0;
50    /* 現在輝度 */
51    uint16_t intensity_current = 0;
52    /* 遷移タイミング決定用カウンタ */
53    int8_t x = 0;
54    int8_t y = 0;
55    /* 主ループ */
56    for (;;) {
57      __delay_ms(50);
58      CLRWDT();
59      /* On/Offスイッチ */
60      int8_t sw_on = SW_ON();
61      /* ボリュームを読んでパラメータを設定 */
62      read_params();
63      /* 乱数空打ち */
64      random_get16();
65      /* 状態機械 */
66      switch (fsm_state) {
```

```
67      case STATE_Q0: /* 状態 Q0 - 消灯期間 */
68        if (sw_on) {
69          fsm_state = STATE_Q1A;
70          x = 0;
71        }
72        intensity_current = INTENSITY_0;
73        break;
74      case STATE_Q1A: /* 状態 Q1A - 起動準備期間(1) */
75        if (!sw_on) {
76          fsm_state = STATE_Q0;
77        } else if (x >= THR_X1) {
78          fsm_state = STATE_Q1B;
79          x = 0;
80        } else if (with_prob(prob_p_1a)) {
81          x = x + 1;
82        } else if (x != 0) {
83          x = x - 1;
84        }
85        intensity_current = INTENSITY_LOW;
86        break;
87      case STATE_Q1B: /* 状態 Q1B - 起動準備期間(2) */
88        if (!sw_on) {
89          fsm_state = STATE_Q0;
90        } else if (with_prob(prob_p_1b)) {
91          fsm_state = STATE_Q2A;
92          x = 0;
93          y = 0;
94        } else {
95          fsm_state = STATE_Q1A;
96          x = 0;
97        }
98        intensity_current = INTENSITY_HIGH;
99        break;
100     case STATE_Q2A: /* 状態 Q2A - 起動過渡期間(1) */
101       if (!sw_on) {
102         fsm_state = STATE_Q0;
103       } else if (x >= THR_X2A) {
104         fsm_state = STATE_Q3;
105         x = 0;
106       } else if (y >= THR_X2B) {
107         fsm_state = STATE_Q0;
108         y = 0;
109       } else {
110         fsm_state = STATE_Q2B;
111         y = y + 1;
112       }
113       intensity_current = INTENSITY_LOW;
114       break;
115     case STATE_Q2B: /* 状態 Q2B - 起動過渡期間(2) */
116       if (!sw_on) {
117         fsm_state = STATE_Q0;
118       } else if (with_prob(prob_p_2)) {
119         fsm_state = STATE_Q2A;
120         x = x + 1;
121       } else {
122         fsm_state = STATE_Q2A;
123       }
124       intensity_current = INTENSITY_HIGH;
125       break;
126     case STATE_Q3: /* 状態 Q3 - 安定維持期間 */
127       if (!sw_on) {
128         fsm_state = STATE_Q0;
129       } else if (x >= THR_X3) {
130         fsm_state = STATE_Q4;
131         x = 0;
132       } else if (with_prob(prob_p_3)) {
133         x = 0;
134       } else {
135         x = x + 1;
136       }
137       intensity_current = INTENSITY_HIGH;
```



図 8.7: 蝋燭の灯 (碇型 2 匁和蝋燭)

```
138         break;
139       case STATE_Q4: /* 状態 Q4 - 安定崩壊期間 */
140         if (!sw_on) {
141           fsm_state = STATE_Q0;
142         } else if (x >= THR_X4) {
143           fsm_state = STATE_Q1A;
144           x = 0;
145         } else if (with_prob(prob_p_4)) {
146           x = x + 1;
147         }
148         intensity_current = INTENSITY_0;
149         break;
150       }
151       /* 輝度出力 */
152       LED_INTENSITY(intensity_current);
153     }
154   }
155
156   /* 安定度パラメータ設定 */
157   static void read_params(void)
158   {
159     /* s: 安定度 --- 0 (不安定) ... 255 (安定) */
160     uint16_t s = ADC_Read(ADC_CH_PARAM) >> 2;
161     prob_p_1a = 127 + (s >> 1);
162     prob_p_1b = s;
163     prob_p_2  = 127 + (s >> 1);
164     prob_p_3  = s;
165     prob_p_4  = 127 + (s >> 1);
166   }
167
168   /* 疑似乱数を用いて確率 ip/255 で真を返す */
169   static int8_t with_prob(uint8_t ip)
170   {
171     uint16_t r = (random_get16() >> 8);
172     if (ip == 0) {
173       return 0;
174     }
175     return (r <= ip);
176   }
```

蛍光灯の点灯の模倣をする仕組みを説明します. 図 8.6 に上記のソースコードで用いられている有限状態機械の状態遷移図を示します. 基本は「ジー・パッパッ・ピカー」のように輝度を変化させますが, 古くなった蛍光灯の不安定な点灯を模倣を行います. 点灯に失敗したり, いったんは点灯しても消えてしまい再度点灯したりと, 可変抵抗器によるパラメータ設定に従って確率的に状態遷移を行います. 言葉によるこれ以上の説明は難しいので, 状態遷移図を見て各自で理解をして下さい.

課題

1. ソースファイル `LED-Light-Candle.c` 中の関数 `intermittent_chaos_get()` の変数 `x` に対する計算が, 式 8.1 を計算して間欠カオスを実現していることを説明して下さい.
2. 実数演算には上で述べた固定小数点数による方法と浮動小数点数による方法の 2 通りがあります. 浮動小数点数におけるビット表現の規格である IEEE 745 を調査して説明して下さい.
3. 固定小数点数と浮動小数点数それぞれの長短を説明して下さい.
4. MARS ライトの点灯を模倣するファームウエアを作成して下さい.
5. Gyralight の点灯を模倣するファームウエアを作成して下さい.
6. 本章で紹介した蛍光灯風点灯のファームウエアは, 実感的な点灯を少し犠牲にして簡潔なものとしています. より実感的な点灯になるような状態遷移図を設計して, ファームウエアを実装して下さい.
7. LED を 2 つ装備した LED 点灯装置を作成し, 踏切警報器の点灯を実現して下さい.
8. 本章で紹介した蝋燭風の点灯は, 実物の蝋燭の炎と比較するとどうしても見劣りがします. LED は一点で明かりの強弱が変化するだけですが, 実物の蝋燭の炎は図 8.7 に示すように空間的な広がりがあり, 空間的に炎が大小して炎は揺らめきます. そこで複数のチップ LED を使用して, 実感ある空間的な揺らめきを実現する方式を考案して下さい.

第 9 章

基本機能型
パワーパック

本章では基本的な機能を実現した PWM 制御式パワーパックを紹介します. 主な仕様は, スロットルはツマミ操作によるボリュームコントロール式, 周波数 31.25kHz の PWM 制御, 最大出力 3A としています. 本章で紹介する回路とファームウエアの特徴は以下の通りです.

- 保護回路内蔵のモータドライバ IC の使用
  自作では実現が大変な短絡防止機能・加熱防止機能などが IC に内蔵されており, 手軽にしかも安価に安全性の高い装置が実現できます.
- 高周波 PWM によるノイズのない走行
  PWM 周波数は可聴域を越えていますから耳障りな音は聞こえません.

本章で学ぶ内容は以下の通りです.

- PIC マイコンの相補波形ジェネレータモジュール CWG
- PWM 制御によるモータ回転制御
- モータドライバ TB6643KQ の使用方法

本章の回路は第 10 章, 第 22 章, 第 23 章で紹介するパワーパックと共通で, 各章で異なるファームウエアを載せて異なる機能を持つパワーパックを実現します. そのために本章の回路図には本ファームウエアでは使用しない回路も含まれていますが, それらは後の章に備えてのものですから本章では無視して下さい.

この装置のファームウエアの大まかな構造を図 9.3 に示します. スピードボリュームに従って PWM デューティ比をモータドライバへ出力するという, 極めて単純なものです.

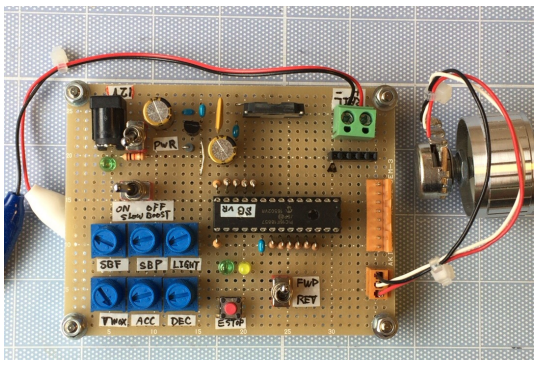


図 9.1: パワーパックの試作例

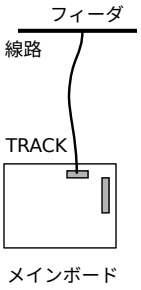


図 9.2: 接続方法

```
1  main()
2  {
3    for(;;) {
4      スピードボリュームを読む;
5      PWMデューティ比を計算;
6      モータドライバに出力;
7    }
8  }
```

図 9.3: ファームウエア構造 (概要)

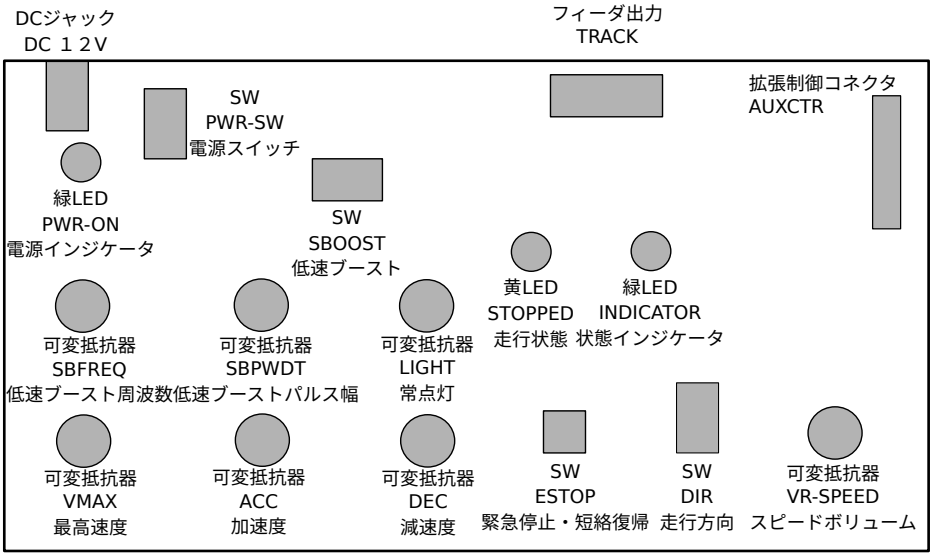


図 9.4: 操作表示部品の配置例と名称

9.1　使用方法

本装置の使用方法を以下に説明します. 以下の説明で使用する部品名称を図 9.4 に配置例とともに示しています. 基本的な接続を図 9.2 に示し, 手順は以下の通りです.

1. DC ジャックに 12V の AC アダプタを接続します.
2. フィーダ出力はターミナル TRACK ですので, これを線路に接続します.

操作部分と表示部分の説明は以下の通りです.

- 電源スイッチ PWR-SW
  回路全体の電源スイッチです. 電源を入れるとパイロットランプ PWR-ON が点灯します.
- 緑色 LED INDICATOR
  正常動作しているとき点灯します.
- 黄色 LED STOPPED
  停車時に点灯し, 走行時に消灯します.
- 可変抵抗器 VR-SPEED: スピードボリューム
  走行速度を決めます.

- 走行方向スイッチ DIR
  走行方向を決めます.
- 押しボタンスイッチ ESTOP
  モータドライバの過電流保護の解除をして短絡復帰を行います.

操作方法は以下の通りです.

1. 速度制御はスピードボリューム VR-SPEED で行います. スピードボリュームを左一杯に回して 0 の位置にしておきます. スピードボリュームが 0 でないまま電源スイッチを入れると模型車両は急発進してモータを痛めてしまいます. (第 10 章ではこの点の改善を行います.)
2. 電源スイッチ PWR-SW を入れます. するとパイロットランプ PWR-ON が点灯します.
3. スピードボリューム VR-SPEED を回して速度を加減し, 模型車両を走行させます.
4. 走行方向を切り替えるにはスイッチ DIR を切り替えます. 走行方向の切り替えは停車してから行って下さい. 停車しないまま走行方向を切り替えるとモータを痛めてしまいます. (第 10 章ではこの点の改善を行います.)

9.2　回路図

図 9.5 にパワーパック本体の回路図を, 表 9.1 にパワーパック本体の部品表を, それぞれ示します.

この回路の電源には 5V と 12V の 2 種類がありますので配線作業の際には注意して下さい. 回路図の電源記号で +5V と描いてあるもの同士だけを結線し, +12V と描いてあるもの同士だけを結線します. 部品によって 5V を電源とするものと 12V を電源とするものとがありますので, 配線を間違えると部品は壊れてしまいます. 本格的に使用する場合はケースへの組み込みを行うことになりますが, その場合はスイッチや LED などはすべてコネクタ等を用いてケースと基板との間の接続を行うと良いでしょう.

表 9.1: 【部品表】 パワーパック (基本機能型)

記号	数量	部品名
U1	1	PIC 16F18857-I/SP [I-11888]
U2	1	3 端子レギュレータ L7805CV-DG　1.5A (又は同等品) [I-07903]
U3	1	モータドライバ TB6643KQ [I-07688]
D2	1	ダイオード 1S3　30V 1A [I-01707]
D1, D4	2	LED 緑 [I-06413]
D3	1	LED 黄 [I-11639]
F1	1	ポリスイッチ MF-R185　1.85A [P-12629]
C1, C6	2	電解コンデンサ 100μF　耐圧 25V 以上 [P-07760]
C3, C4, C5	3	セラミックコンデンサ 0.1μF [P-00090]
C2	1	セラミックコンデンサ 0.33μF [P-08147]
R5, R6, R7	3	抵抗 1kΩ　1/4W [R-25102]
R1, R2, R3, R4, R8	5	抵抗 10kΩ　1/4W [R-25103]
RV1	1	可変抵抗器 10kΩ　B カーブ, スピード制御用 [P-12576]
RV2, RV3, RV4, RV5, RV6, RV7	6	可変抵抗器 10kΩ　B カーブ, 半固定型 [P-08012]

(つづく)

(つづき)

記号	数量	部品名
		可変抵抗器用ツマミ　使用する可変抵抗器に合わせて適宜
SW2	1	単極双投トグル SW・中点付　On-Off-On 型 [P-02400]
SW3, SW4	2	単極双投トグル SW　On-On 型 [P-02399]
SW1	1	プッシュ SW　[P-03646]
J3	1	ターミナルブロック 2 ピン　[P-02333]
J1	1	基板用コネクタ 8 ピン　JST XH コネクタ B8B-XH-A(LF) [C-12253]
	1	IC ソケット 28 ピン　[P-01339]
J2	1	ピンソケット 6 ピン　ICSP 用 (無くても可) [C-03784]
CON1	1	DC ジャック　[C-09408]
	1	AC アダプタ 12V　出力 3A [M-10661]
		ほか基板・配線材等

回路図から分かる通りフィーダ出力は 1 系統のみです. リバース区間のあるレイアウトに使用する場合にはフィーダ出力を 2 つに分岐してそれぞれに逆転スイッチを取り付けて, パワーパック本体の走行方向スイッチ DIR は使用せずに取り付けた逆転スイッチを使用して下さい.

9.3　CWG: 相補波形ジェネレータモジュール

PIC マイコンの内蔵周辺モジュール CWG (Complementary Waveform Generator; 相補波形ジェネレータ) を使用して各種の波形の生成が行えます. CWG には実に多彩な使用方法がありますが, そのすべてはとても紹介しきれません. 本章では DC モータの正転・逆転を PWM 制御 で行う際に便利な CWG の出力ステアリングモードの使用方法を紹介します. PIC マイコンによっては複数の CWG モジュールを内蔵しており, CWG1, CWG2, ... と名前が付けられています. ここでは本章のファームウエアで使用している CWG1 を具体例にした使用方法を説明します.

DC モータの PWM 制御での CWG1 の使用の基本方針は, TMR2 モジュールと CCP1 モジュールが PWM 波形を生成し, CWG1 モジュールがそれを受けてモータドライバの 2 つの入力ピンへ出力する出力ステアリングモードでの使用です. TMR2 モジュールと CCP1 モジュールを使った PWM 周波数 31.25kHz の PWM 波形の生成は第 5 章で紹介した方法と同じでプログラムコードも全く同じですから, ここでは説明は繰り返しません. 第 5 章の説明を適宜復習して下さい.

図 9.6 に CWG1 モジュールを出力ステアリングモード (output steering mode) と呼ばれる使い方をした場合の内部構造の概略を示します. 図には 2 つの出力 CWG1A と CWG1B が記されていますがこの他に図には記していない 2 つの出力 CWG1C と CWG1D があり, CWG1 モジュールには合計 4 つの出力があります. 本書では CWG1C と CWG1D を使用しませんので説明は省きます. 図 9.6 から分かる通り CWG1 モジュールへの入力は CCP1 からの PWM 波であり, CWG1 モジュール内部にある 3 種類の設定ビット POL, STR, OVR の値に従って出力を決定します. これらの設定ビットは図からも分かる通り CWG1A と CWG1B それぞれにあります. 3 種類の設定ビットの説明は以下の通りです.

- POL
  出力極性ビット (output polarity bit) と呼ばれています. CWG1 モジュールへの入力と POL の値の排他

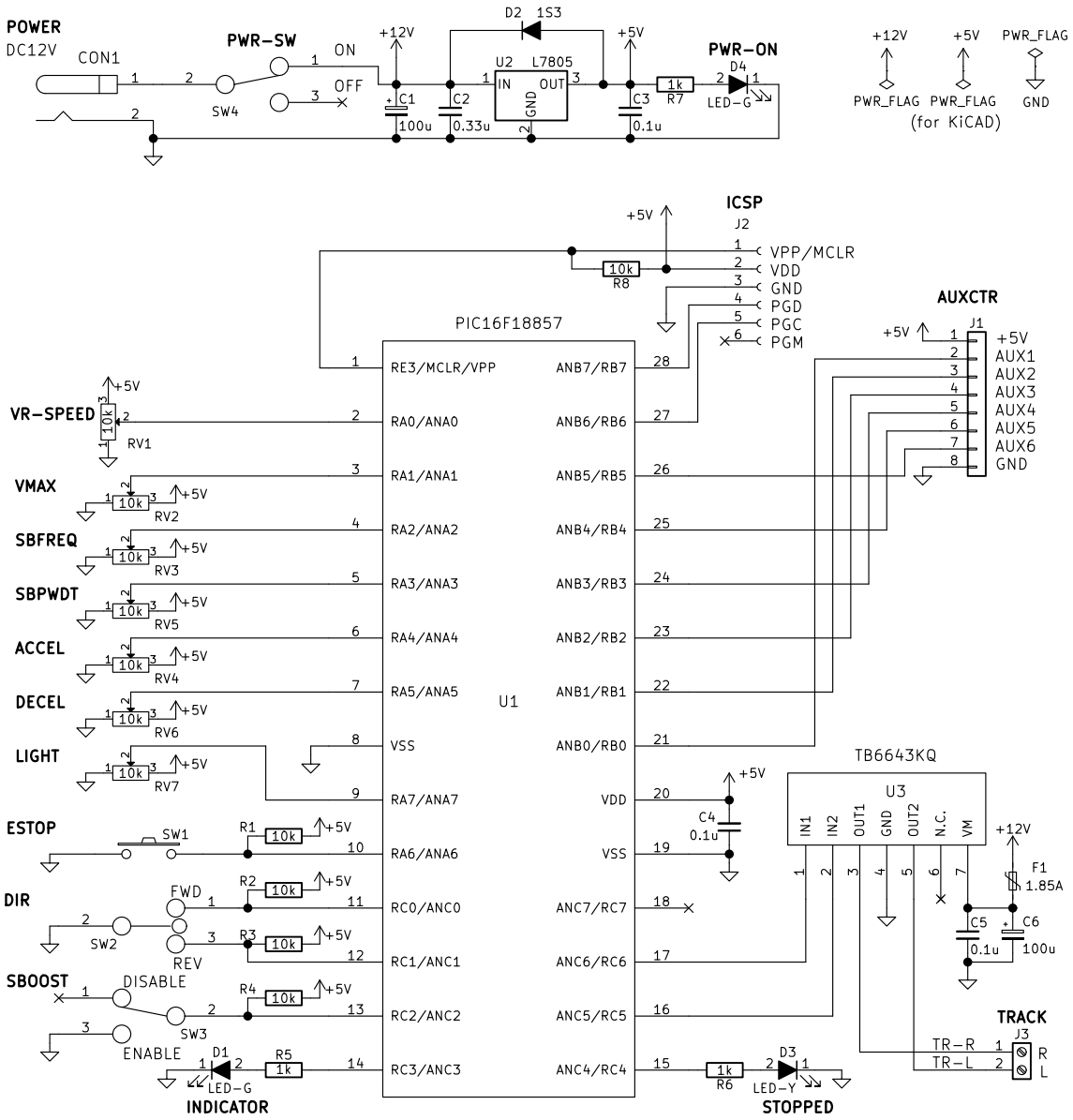


図 9.5: 【回路図】 パワーパック (基本機能型) 本体

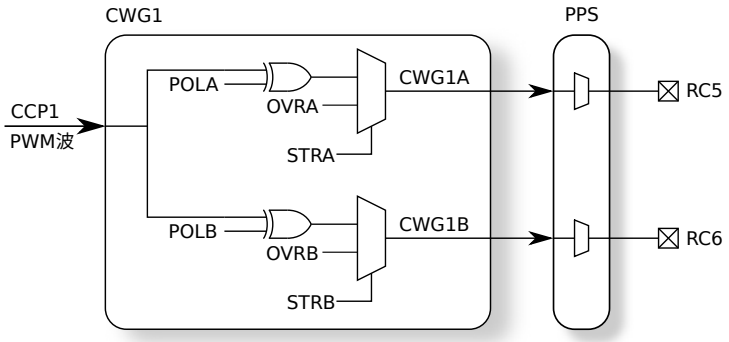


図 9.6: CWG1 の内部構造の概略 (出力ステアリングモード)

的論理和を計算して次段に送っている点に注意して下さい.

- POL=0 の場合
  CWG1 モジュールへの入力をそのまま次段に送ります.
- POL=1 の場合
  CWG1 モジュールへの入力を反転して次段に送ります.

• STR
ステアリングイネーブルビット (steering enable bit) と呼ばれています.

- STR=0 の場合
  ステアリングデータビット OVR に設定されている値を出力とします. この場合, CWG1 モジュールへ入力される CCP1 からの信号は出力されません.
- STR=1 の場合
  POL の設定値に従って, CWG1 モジュールへ入力される CCP1 からの信号を出力します.

• OVR
ステアリングデータビット (steering data bit) と呼ばれています. ステアリングイネーブルビット STR=0 の場合には上述の通りこのステアリングデータビットの値が出力されます.

9.4　モータドライバ IC TB6643KQ

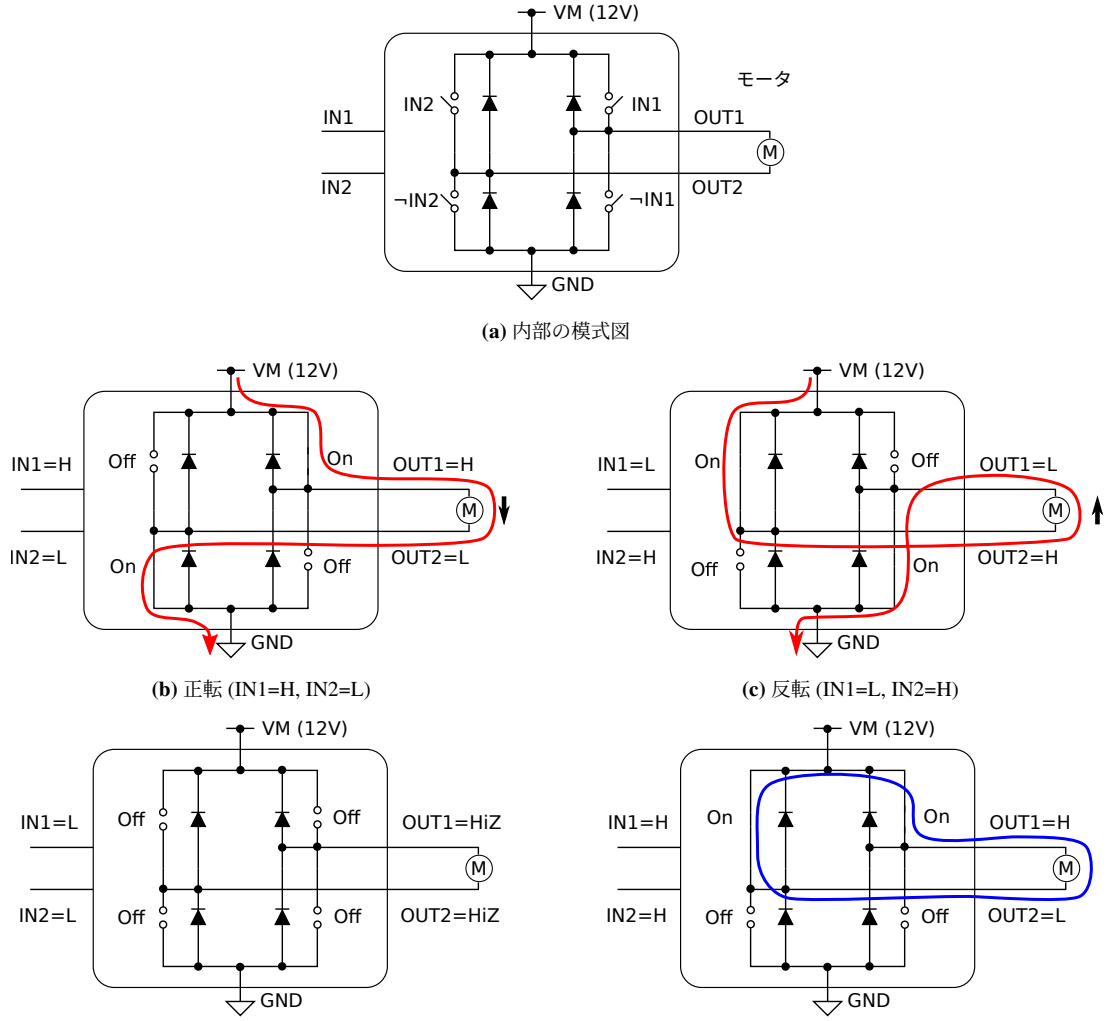
モータドライバ IC TB6643KQ 使用して模型車両を走行させる方法を説明します. 第 3 章 76 ページに掲載の入出力ファンクション表 3.3 で示した通り, モータドライバ IC TB6643KQ には 2 つの入力ピン IN1 と IN2 があります. 入力ピンに与える信号を制御することで 2 つの出力ピン OUT1 と OUT2 の出力電圧を変化させることができ, その結果として線路に印加する電圧を制御できます. 以下では, 出力ピン OUT1 を右レールに, 出力ピン OUT2 を左レールに, それぞれ接続したものとして説明します.

モータドライバ IC TB6643KQ は H ブリッジ型モータドライバで, その内部回路の概略を図 9.7a に示します. なおこの図にはスイッチが 4 つ描いてありますが, 実際には機械的なスイッチではなく FET が使われています. この図のように 4 つのスイッチによりモータの回転方向を変化できる回路を H ブリッジと呼び, 機械的なスイッチは使用せず半導体だけでモータの逆転が行えます.

走行方向の制御の原理は以下の通りです.

• 前進
ピン IN1 を H (High) に, ピン IN2 を L (L) に, それぞれ設定します. 入出力ファンクション表 3.3 より, OUT1 には H (12V), OUT2 には L (0V) がそれぞれ出力されます. 右レールがプラス, 左レールがマイナスとなり, 模型車両の走行方向は前進になります. この場合のモータドライバ内での電流の流れを図 9.7b に示しています. なお OUT1 と OUT2 の実際の出力電圧は VM ピンに印加される電圧から電圧降下分を差し引いた値で必ずしも 12V とは限りませんが, 以降では説明の便宜上 12V として説明します.

• 後進
ピン IN1 を L に, ピン IN2 を H に, それぞれ設定します. 入出力ファンクション表 3.3 より, OUT1 には L (0V), OUT2 には H (12V) がそれぞれ出力されます. 線路の電圧極性が先ほどの場合とは逆なので, 模型車両の走行方向は後進になります. この場合のモータドライバ内での電流の流れを図 9.7c に示しています.



(a) 内部の模式図

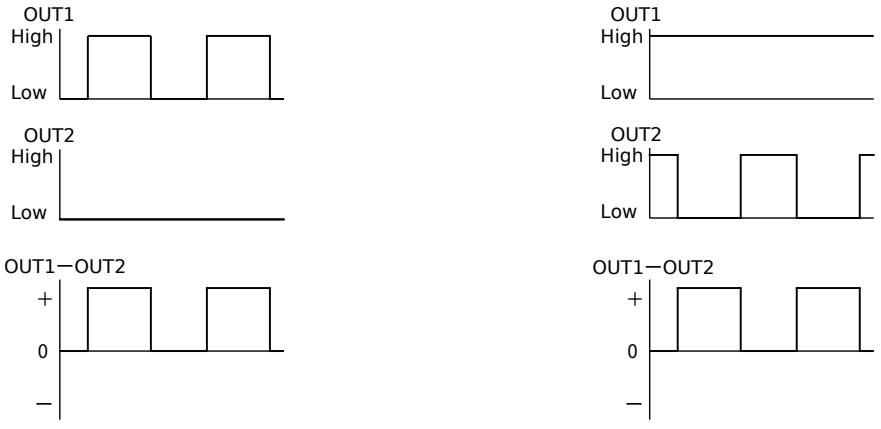
(b) 正転 (IN1=H, IN2=L)

(c) 反転 (IN1=L, IN2=H)

(d) ハイインピーダンスモード (IN1=L, IN2=L)

(e) ショートモード (IN1=H, IN2=H)

図 9.7: H ブリッジモータドライバの模式図と入力ごとの動作



(a) アクティブ・ハイ

(b) アクティブ・ロウ

図 9.8: PWM 制御におけるアクティブ・ロウとアクティブ・ハイ

9.4.1　アクティブ・ハイとアクティブ・ロウ

PWM 制御はアクティブ・ハイとアクティブ・ロウの 2 通りに分類できます. 電流を流すときに信号を H にするか L にするかの違いがアクティブ・ハイとアクティブ・ロウの違いです.

- アクティブ・ハイ (active-high) 方式
  図 9.8a に示す通り, 通常は OUT1 と OUT2 をともに L にしておきます. OUT1 が H のときには, OUT2 を基準とする OUT1 の電圧 (OUT1-OUT2) は H になります.
- アクティブ・ロウ (active-low) 方式
  図 9.8b に示す通り, 通常は OUT1 と OUT2 をともに H にしておきます. OUT2 が L のときには, OUT2 を基準とする OUT1 の電圧 (OUT1-OUT2) は H になります.

本パワーパックでの走行速度の PWM 制御はアクティブ・ロウと呼ばれる方式で行います.

- 前進
  IN1 を常に H とし, IN2 に PWM 波を与えます. IN2 が L となる期間の比率を走行速度の比率に設定します. つまり 30% の速度で走行させるときは IN2 が L の期間の比率を 30% にします.
- 後進
  IN1 に PWM 波を与え, IN2 を常に H とします. IN1 が L となる期間の比率を走行速度の比率に設定します. つまり 30% の速度で走行させるときは IN1 が L の期間の比率を 30% にします.

アクティブ・ハイとアクティブ・ロウのどちらを用いても PWM 制御としては同じですが, モータドライバ TB6643KQ の入力端子を駆動する場合ではアクティブ・ハイかアクティブ・ロウにより TB6643KQ の動作に大きな違いがあるため注意が必要です. なお本ファームウエアではアクティブ・ロウ方式を採用してモータの PWM 制御を行いますが, その理由は模型車両のヘッドライトとテールライトの点灯を適切にするためであり, これについては次節で詳しく説明します. アクティブ・ロウ方式でモータドライバ TB6643KQ の入力ピンを制御する場合は, 入出力ファンクション表 3.3 より, 出力は以下の通りとなります.

- モータに電力を供給しない期間
  IN1 と IN2 はともに H なので, OUT1 と OUT2 はショートモードです.
- モータに電力を供給する期間
  前進時は IN1 を H に, IN2 を L にし, 後進時は IN1 を L に, IN2 を H にすることで, 模型車両のモータは適切な方向に回転します.

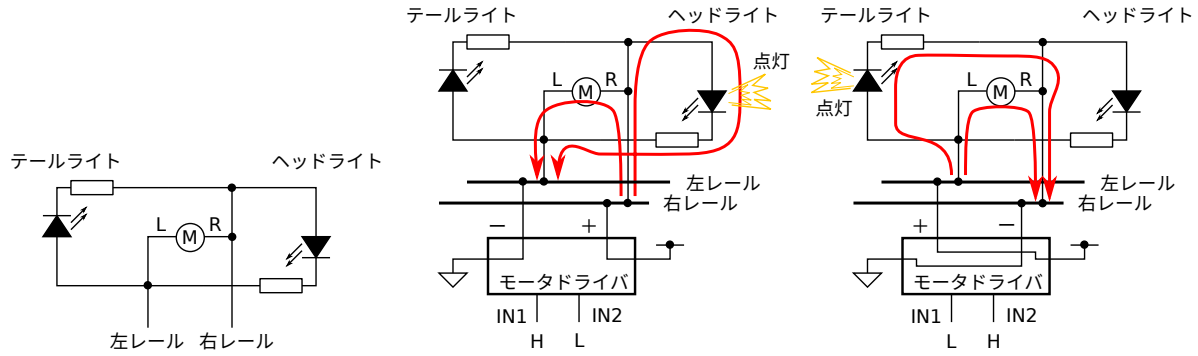
アクティブ・ロウ方式でモータドライバを駆動した場合には, モータに電力を供給しない期間のモータドライバ出力 OUT1 と OUT2 はショート状態になります.

9.4.2　ヘッドライトとテールライト: 点灯とちらつきの原理

最初にヘッドライトとテールライトの点灯の原理を説明します. 図 9.9a が典型的なヘッドライト・テールライト回路です. この図では LED を使った場合を示していますが, 電球を使った場合でも原理は同じです.

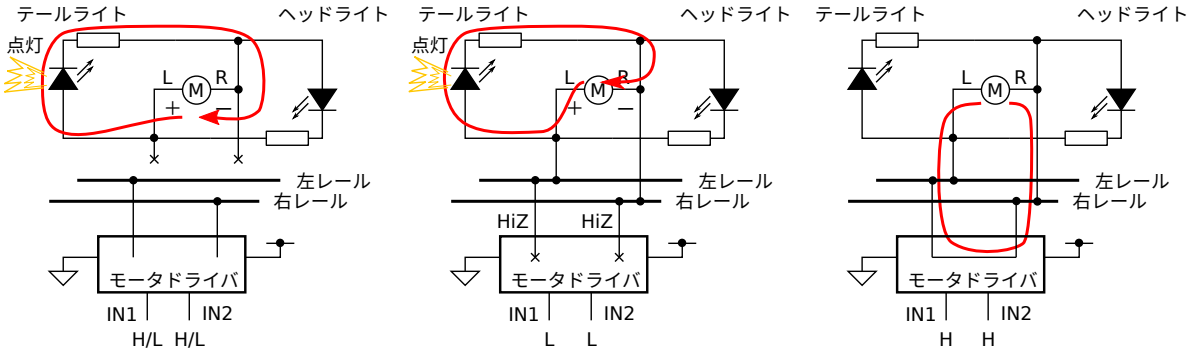
線路に電圧が印加されているとき, ヘッドライトとテールライトの点灯の原理は以下の通り説明できます.

- 右レールの極性がプラスの場合
  図 9.9b に示すように電流が流れて, ヘッドライトが点灯してモータが回転して前進します. この時にはテールライトは点灯しません.



(a) 典型的な回路　**(b)** 前進時 (ヘッドライト点灯)　**(c)** 後進時 (テールライト点灯)

図 9.9: ヘッドライトとテールライトが点灯する原理



(a) 離線時　**(b)** ハイインピーダンスモード時　**(c)** ショートモード時

図 9.10: 走行中にヘッドライトとテールライトがちらつく原理 (デューティ・オフ期間)

- 左レールの極性がプラスの場合
  図 9.9c に示すように電流が流れて, テールライトが点灯してモータは反転して後進します. この時にはヘッドライトは点灯しません.

次に走行中にヘッドライトとテールライトがちらつくことがしばしば有りますが, その原理を説明します. 模型車両が走行中に振動等で離線することがしばしばあります. その瞬間はモータへの電流は途絶えますが, 慣性によりモータの回転が続くために発電機としてはたらき, 図 9.10a に示す状態となります. つまりわずかにテールライトが光る結果となってしまいます. 電球の場合には全く気にならなかったのですが, LED の場合には顕著です. その理由は以下の通りです.

- 電球の場合: フィラメントの温度は時間をかけて徐々に上昇し, ある程度の温度になってから光り始める特性を持つため, 瞬間的な電流では実際には光りません. フィラメントの温度は瞬間的には上昇しないからです.
- LED の場合: 電流が流れた瞬間に光る特性を持つため, 瞬間的な電流でも光る結果となってしまいます.

このように電球の場合には気にならなかったものが, LED の場合には顕著となっています. しかし模型列車の離線はパワーパックから完全に切り離されている状態なので, アナログ制御方式を用いる限りはこれ以上のパワーパック側でのチラつきの防止対策は不可能で, 模型列車にノイズ吸収回路を搭載するなどの工夫が必要

です.

最後に, 本パワーパックがデューティ・オフ期間 (電圧を供給しない期間) にモータドライバ IC のショートモードにしている理由を説明します. これはヘッドライトとテールライトを考慮した結果です.

アクティブ・ハイ方式でモータドライバを駆動した場合を考えます. デューティ・オフ期間のモータドライバ出力 OUT1 と OUT2 はハイインピーダンス状態になることが, 入出力ファンクション表 3.3 と図 9.7d から分かります. しかしデューティ・オフ期間にも模型車両の中のモータは慣性で回転が続くため, モータは発電機としてはたらいて, 図 9.10b に示す通りモータで発生した電圧がテールライトを点灯させてしまいます. PWM 周波数の早さで電圧印加とハイインピーダンス状態が交互に起きますから, ヘッドライトとテールライトの点灯と消灯が高速に生じて, 人間の目には同時に点灯するように見える結果となります. たとえ列車の編成中で先頭車両, 動力車両, 後尾車両の 3 つが別々でも, 動力車両で発生した電圧は線路を伝って先頭車両と後尾車両に届いてヘッドライトとテールライトが同時に点灯します.

この同時点灯の問題は, PWM 制御をアクティブ・ロウ方式で行うことで回避できます. 図 9.10c に示す通りにモータドライバ出力をショート状態にすれば, ヘッドライト・テールライトに印加される電圧はゼロになり点灯しません. 以上がモータドライバ IC のショートモードにしている理由です.

9.4.3　過電流保護

列車が脱線したり線路の上に金属を落としてショートが起きると過電流により火災になりかねませんから, 回路での過電流対策は必須です. そこで本回路では以下の 3 段階の過電流対策を行っています.

- 過電流保護機能を内蔵した AC アダプタの使用
- ポリスイッチによる保護
- 過電流保護機能を内蔵したモータドライバの使用

本回路で使用しているモータドライバ TB6643KQ には第 3 章で説明した通り過電流保護機能が内蔵されています. TB6643KQ の過電流保護機能が作動すると解除しない限りは再び出力をしませんから, 走行はできないことになります. 過電流保護機能を解除する方法は, 入力ピン IN1 と IN2 を 5.1 マイクロ秒以上の期間ともに L にすることです. 本ファームウエアでは, 押しボタンスイッチ ESTOP が押されると入力ピン IN1 と IN2 を一定時間 L にして過電流保護機能を解除します.

9.4.4　他のモータドライバの使用

IN1 と IN2 による制御入力形式は TB6643KQ 固有のものではなく, 例えばローム BD6231F も同じ入力形式でモータ制御を行います. そのためファームウエアは変更せずにモータードライバだけを置き換えることも場合によっては可能です. 特に TB6643KQ は電源電圧が最低 10V でないと正しい動作が保証されていませんので, 10V 以下の低電圧のモータを使用している模型車両では本回路はそのままでは使えません. その場合はローム BD6231F のような低電圧でも動作するモータドライバに置き換えて下さい.

9.5　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 9.2: 【ソースファイル一覧】 パワーパック: 基本機能型

初	頁	番号	ファイル名	内容	行数
✓	172	9.3	`PowerPack-Basic.c`	パワーパックメイン関数	69

(つづく)

```
1  void main(void)
2  {
3    初期化;
4    for(;;) {
5      __delay_ms();
6      走行方向SWを読み，モータドライバ出力をそれに合わせる;
7      vr_speed = スピードボリュームの値;
8      speed = 速度値(vr_speedより計算);
9      md = モータPWMデューティ比(speedより計算);
10     PWMデューティ比出力をmdに設定;
11   }
12 }
```

図 9.11: ファームウエア構造 (詳細)

(つづき)

初	頁	番号	ファイル名	内容	行数
✓	172	9.2	pp-const.h	パワーパック定数定義: ヘッダ	17
✓	170	9.1	io.h	I/O ピン割付: ヘッダ	50
✓	173	9.4	mcr.h	MCR・周辺モジュール: ヘッダ	8
✓	174	9.5	mcr.c	MCR・周辺モジュール: 実装	84
✓	175	9.6	mcr-motor.h	モーター制御周辺モジュール: ヘッダ	25
✓	177	9.7	mcr-motor.c	モーター制御周辺モジュール: 実装	98
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76

計 436 行 (うち初出 351 行, 既出 85 行)

初出のソースファイルが多数ありますが, 本章のファームウエアでは不要な定義やプログラムコードもあります. これは後の章で紹介するパワーパックのファームウエアと共用するためです. 以降では本章の内容の理解に必要な部分だけを説明します.

ファームウエアの構造をもう少し詳しく説明したのが図 9.11 です. これと照らし合わせながら実際のソースファイルを熟読して理解して下さい.

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います. 本ファームウエアで関係するのは以下のものだけです.

- ADC チャンネル ADC_CH_SPEED
- 状態インジケータ LED 関係 LED_INTICATOR_*xxx*
- 停車中表示 LED 関係 LED_STOPPED_*xxx*
- 進行方向スイッチ関係 SW_DIR1F, SW_DIR1R

ソースファイル番号 9.1: io.h

```
1  #if !defined(__IO_H__)
2  #define __IO_H__
3
```

```
4  /* 設定VR ADC 走行速度: ANA0 */
5  #define ADC_CH_SPEED              0x00
6  /* 設定VR ADC 最高速度: ANA1 */
7  #define ADC_CH_VMAX               0x01
8  /* 設定VR ADC 低速ブースト周波数設定: ANA2 */
9  #define ADC_CH_SBFREQ             0x02
10 /* 設定VR ADC 低速ブーストパルス幅設定: ANA3 */
11 #define ADC_CH_SBPWDT             0x03
12 /* 設定VR ADC 加速度設定: ANA4 */
13 #define ADC_CH_ACCEL              0x04
14 /* 設定VR ADC 減速度設定: ANA5 */
15 #define ADC_CH_DECEL              0x05
16 /* 設定VR ADC - 常点灯設定: ANA7 */
17 #define ADC_CH_LIGHT              0x07
18
19 /* 状態インジケータLED: RC3 */
20 #define LED_INDICATOR_ON()        (LATC3 = 1)
21 #define LED_INDICATOR_OFF()       (LATC3 = 0)
22 #define LED_INDICATOR_TOGGLE()    (LATC3 = ~LATC3)
23 /* 停車中表示 LED: RC4 */
24 #define LED_STOPPED_ON()          (LATC4 = 1)
25 #define LED_STOPPED_OFF()         (LATC4 = 0)
26 #define LED_STOPPED_TOGGLE()      (LATC4 = ~LATC4)
27
28 /* 進行方向スイッチFWD: RC0 */
29 #define SW_DIR1F()                (RC0 == 0)
30 /* 進行方向スイッチREV: RC1 */
31 #define SW_DIR1R()                (RC1 == 0)
32 /* 緊急停止ボタン: RA6 */
33 #define SW_EMG_STOP()             (RA6 == 0)
34 /* 低速ブースト有効スイッチ: RC2 */
35 #define SW_SBEN()                 (RC2 == 0)
36
37 /* 主幹制御器 制動3: RB0 */
38 #define SW_MC_BRK3()              (RB0 == 0)
39 /* 主幹制御器 制動2: RB1 */
40 #define SW_MC_BRK2()              (RB1 == 0)
41 /* 主幹制御器 制動1: RB2 */
42 #define SW_MC_BRK1()              (RB2 == 0)
43 /* 主幹制御器 緩メ:  RB3 */
44 #define SW_MC_NWT()               (RB3 == 0)
45 /* 主幹制御器 力行1: RB4 */
46 #define SW_MC_ACC1()              (RB4 == 0)
47 /* 主幹制御器 力行2: RB5 */
48 #define SW_MC_ACC2()              (RB5 == 0)
49
50 #endif /*__IO_H__*/
```

ソースファイル pp-const.h (パワーパック定数定義: ヘッダ)

パワーパックのファームウエアで使用する定数類を定義します.

- 最高速度 SPEED_MAX
  走行速度の最大値で, 1023 と定義します. 走行速度は PWM デューティ比の値で表し, 0 から 1023 までの範囲の値とします. 10 ビットで表現される整数値なので, C 言語の 16 ビット整数型 uint16_t を変数型に使用します.
- 速度増加減少定数 SPEED_INC_DY
  ブレゼンハムの線形増加アルゴリズムを使用した加減速を行う際の D_Y の値で, 固定値を用います. 加減速度の変更は D_X の値の変化で行います. なお本ファームウエアではこの定数は使用せず, 後の章で使用します.
- 実行サイクル間隔 RUN_INTERVAL

プログラムの実行の間隔を定義します. 1 サイクル 50 ミリ秒と定義します.

- ADC 値マージン ADC_MARGIN
  可変抵抗器によっては左一杯に回し切っても抵抗値がちょうど 0Ω にならない品もあります. もし抵抗値がちょうど 0Ω にならなければ模型列車の速度を 0 にできず大変に困ります. そこで ADC 値からこの ADC 値マージン値を差し引くことで, この問題を解決します. 例えば ADC 値が 18 の場合には 8 を引いて 10 とし, ADC 値が 8 未満の場合には 0 とします.

ソースファイル番号 9.2: pp-const.h

```
#if !defined(__PP_CONST_H__)
#define __PP_CONST_H__

/* 最高速度 */
#define SPEED_MAX       1023
/* 速度増加減少定数 */
#define SPEED_INC_DY    2048
/* 実行1サイクルの時間 [mSec] */
#define RUN_INTERVAL      50
/* 1秒あたりの実行回数 1000/RUN_INTERVAL [Hz] */
#define NRUNS_PER_SEC     20
/* LEDによる警告点灯/消灯時間 [mSec] */
#define T_LED_WARNING    100
/* ADC値マージン */
#define ADC_MARGIN         8

#endif /*__PP_CONST_H__*/
```

ソースファイル PowerPack-Basic.c (パワーパックメイン関数)

パワーパックを実現する核心部分で, 概要は以下の通りです.

- 走行方向スイッチの通りに走行方向を設定します.
- スピードボリュームの通りにモータ PWM デューティ比を出力します.
- 停車中のときは LED を点灯します.

モータドライバの制御のおいて以下の関数を使用します. 詳しくは後のソースファイルにて説明します.

- MOTOR_DIR_FWD()
  走行方向を前進 (forward) とします.
- MOTOR_DIR_REV()
  走行方向を後進 (reverse) とします.
- MOTOR_DIR_NONE()
  走行方向をニュートラルにします.
- MOTOR_PWM_OUT(md)
  引数に指定された値をモータ PWM デューティ比として出力します.

ソースファイル番号 9.3: PowerPack-Basic.c

```
/*============================================================
 * パワーパック(基本機能PWM型) / PIC16F18857
 * by Hirotsugu Kakugawa
 *============================================================*/
#include <xc.h>
#include <stdint.h>
```

```
7  #include "mcr.h"
8  #include "mcr-motor.h"
9  #include "adc.h"
10 #include "io.h"
11 #include "pp-const.h"
12
13 /*-------------------------------------------------------
14  * MAIN
15  *-------------------------------------------------------*/
16 void main(void)
17 {
18   /* システム初期化 */
19   System_Init();
20   /* ウオッチドッグタイマ: 256 [mSec] */
21   WDTCON0bits.WDTPS = 0x08; /* 0b01000 */
22   CLRWDT();
23   /* モータ出力初期化 */
24   MOTOR_DIR_NONE();
25   MOTOR_PWM_START();
26   /* 状態インジケータLED */
27   LED_INDICATOR_OFF();
28   /* 停車中表示 LED */
29   LED_STOPPED_ON();
30   /* 主ループ */
31   for (;;) {
32     __delay_ms(RUN_INTERVAL);
33     CLRWDT();
34     /* 走行方向を出力 */
35     if (SW_DIR1F()) {
36       MOTOR_DIR_FWD();
37       LED_INDICATOR_ON();
38     } else if (SW_DIR1R()) {
39       MOTOR_DIR_REV();
40       LED_INDICATOR_ON();
41     } else {
42       MOTOR_DIR_NONE();
43       LED_INDICATOR_OFF();
44     }
45     /* スピードボリューム読込 */
46     adc_val_t vr_speed = ADC_Read(ADC_CH_SPEED);
47     /* 目標走行速度 */
48     /* (可変抵抗器のADC値がちょうど0にならない場合への対策) */
49     uint16_t speed = 0;
50     if (vr_speed < ADC_MARGIN) {
51       speed = 0;
52     } else {
53       speed = vr_speed - ADC_MARGIN;
54     }
55     /* モーターPWMデューティー比を出力 */
56     uint16_t md = speed;
57     MOTOR_PWM_OUT(md);
58     /* 停車中表示 LED */
59     if (speed == 0) {
60       LED_STOPPED_ON();
61     } else {
62       LED_STOPPED_OFF();
63     }
64     /* モータードライバ LSI TB6643KQ 制御 (過電流検出の解除) */
65     if (SW_EMG_STOP()) {
66       MotorDriver_TB6643KQ_Reset();
67     }
68   }
69 }
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います. PWM 制御のための初期化や関数群は別のソースファイル mcr-motor.h と mcr-motor.c で取扱います.

ソースファイル番号 9.4: mcr.h

```
#if !defined(__MCR_H__)
#define __MCR_H__

#define _XTAL_FREQ 32000000 /* 32MHz */

extern void System_Init(void);

#endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います. I/O ピンのデジタル入出力やアナログ入力の設定に加えて, PWM 出力の I/O ピンの設定も行います. 本ファームウエアでは内蔵周辺モジュール CWG1 を使用し, マイコンの 2 本のピンを通じてマイコン外部のモータドライバを駆動します. 具体的には, CWG1 モジュール出力の CWG1A と CWG1B を PPS 機能を用いてマイコンの RC5 ピンと RC6 ピンに結合します. そしてマイコン外部のハードウエア的に配線により, RC5 ピンはモータドライバ TB6643KQ の IN1 入力ピンに, RC6 ピンは IN2 入力ピンに, それぞれ接続されています.

ソースファイル番号 9.5: mcr.c

```
/*-----------------------------------------------------
 * MCR設定&周辺モジュール (PIC16F18857)
 *-----------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "mcr-motor.h"
#include "adc.h"

/*---------------------------------------------*/
/* MCR初期化 */
void System_Init(void)
{
  /* 発振器 (FOSC=32MHz) */
  OSCCON1 = 0x10; /* HFINTOSC with 2x PLL */
  OSCCON3 = 0x00;
  OSCEN   = 0x00;
  OSCFRQ  = 0x05; /* 16MHz */
  OSCTUNE = 0x00;
  while (PLLR == 0) {
    ;
  }
  /* I/Oピン */
  LATA   = 0x00;
  LATB   = 0x00;
  LATC   = 0x00;
  ANSELA = 0xBF; /* ANA7,ANA5,ANA4,ANA3,ANA2,ANA1,ANA0 */
  ANSELB = 0x00; /* - */
  ANSELC = 0x80; /* - */
  TRISA  = 0xFF; /* RA7,RA6,RA5,RA4,RA3,RA2,RA1,RA0 */
  TRISB  = 0xFF; /* (RB7,RB6),RB5,RB4,RB3,RB2,RB1,RB0 */
  TRISC  = 0x87; /* (RC7),RC2,RC1,RC0 */
  WPUA   = 0x00;
  WPUB   = 0xFF; /* RB7,RB6,RB5,RB4,RB3,RB2,RB1,RB0 */
  WPUC   = 0x80; /* RC7 */
  WPUE   = 0x00;
  ODCONA = 0x00;
  ODCONB = 0x00;
  ODCONC = 0x00;
  int8_t ogie = GIE;
  GIE = 0;
  PPSLOCK = 0x55;
  PPSLOCK = 0xAA;
  PPSLOCKbits.PPSLOCKED = 0;
  RC5PPS = 0x05;   /* RC5 <- CWG1A */
```

```
46    RC6PPS = 0x06;   /* RC6 <- CWG1B */
47    PPSLOCK = 0x55;
48    PPSLOCK = 0xAA;
49    PPSLOCKbits.PPSLOCKED = 1;
50    GIE = ogie;
51    /* 周辺モジュール */
52    ADC_Init();
53    MCR_Motor_Init();
54  }
55
56  /*-------------------------------------------*/
57  /* CONFIG1 */
58  #pragma config FEXTOSC  = OFF
59  #pragma config RSTOSC   = HFINTPLL
60  #pragma config CLKOUTEN = OFF
61  #pragma config CSWEN    = ON
62  #pragma config FCMEN    = ON
63  /* CONFIG2 */
64  #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
65  #pragma config PWRTE    = OFF
66  #pragma config LPBOREN  = OFF
67  #pragma config BOREN    = ON
68  #pragma config BORV     = LO
69  #pragma config ZCD      = OFF
70  #pragma config PPS1WAY  = ON
71  #pragma config STVREN   = ON
72  #pragma config DEBUG    = OFF
73  /* CONFIG3 */
74  #pragma config WDTCPS   = WDTCPS_31
75  #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
76  #pragma config WDTCWS   = WDTCWS_7
77  #pragma config WDTCCS   = SC
78  /* CONFIG4 */
79  #pragma config WRT      = OFF
80  #pragma config SCANE    = available
81  #pragma config LVP      = OFF /* 低電圧書込不可 */
82  /* CONFIG5 */
83  #pragma config CP       = OFF
84  #pragma config CPD      = OFF
```

ソースファイル mcr-motor.h (モーター制御周辺モジュール: ヘッダ)

マイコンの PWM 出力に関係する関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 9.6: mcr-motor.h

```
1   #if !defined(__MCR_MOTOR_H__)
2   #define __MCR_MOTOR_H__
3
4   /* 走行方向 (PWM出力先の切替): CWG1A, CWG1B */
5   /*  (Inverted Polarity 設定に注意) */
6   #define MOTOR_DIR_FWD() \
7     { CWG1STRbits.STRA = 1; CWG1STRbits.STRB = 0; }
8   #define MOTOR_DIR_REV() \
9     { CWG1STRbits.STRA = 0; CWG1STRbits.STRB = 1; }
10  #define MOTOR_DIR_NONE()  \
11    { CWG1STRbits.STRA = 0; CWG1STRbits.STRB = 0; }
12  /* モーターPWM出力: CCP1 */
13  #define MOTOR_PWM_OUT(D)          CCP1_SetDutyCycle(D)
14  /* PWM用クロック動作開始 */
15  #define MOTOR_PWM_START()         TMR2_Start()
16
17  extern void MCR_Motor_Init(void);
18  extern void TMR2_Init(void);
19  extern void TMR2_Start(void);
20  extern void CCP1_Init(void);
21  extern void CCP1_SetDutyCycle(uint16_t duty);
22  extern void CWG1_Init(void);
23  extern void MotorDriver_TB6643KQ_Reset(void);
24
```

```
25 #endif /*__MCR_MOTOR_H__*/
```

ソースファイル mcr-motor.c（モーター制御周辺モジュール: 実装）

マイコンの PWM 出力に関係する関数等の実装を行います. TMR2 モジュールと CCP1 モジュールにより, LED 輝度制御の時と同様に PWM 周波数 31.25kHz の PWM 波形を生成します. 本パワーパックでは機械的な逆転スイッチは用いずに, ソフトウエアによる制御で前進・後進の切り替えを行います. これを実現するためにはモータドライバの 2 つの入力に供給する PWM 波形を切り替えないといけません. これを行うために CWG1 モジュールの出力ステアリングモードを使用します.

- 前進時
  以下の設定が行われます.
  - STRA=1, STRB=0
    ソースファイル mcr-motor.h で定義したマクロ MOTOR_DIR_FWD() でこれが行われています.
  - POLA=POLB=1 および OVRA=OVRB=1
    ソースファイル mcr-motor.c 内の関数 CWG1_Init() においてこれが行われています.
  - CWG1A を RC5 ピンに結合し, CWG1B を RC6 ピンに結合
    ソースファイル mcr.c 内の関数 System_Init() においてこれが行われています.
  - RC6 ピンをモータドライバ TB6643KQ の IN1 入力ピンに接続, RC5 ピンを IN2 入力ピンに接続.
    回路図においてこの接続が行われています.

  以上の設定と接続から, マイコンのピンの出力は以下の通りになります.
  - IN1 = RC6 = H
    STRB=0 なので CWG1A の出力は OVRB の値となります. OVRB=1 と設定しているため, CWG1A の出力は 1 (H) となります.
  - IN2 = RC5 = CCP1 の出力の反転
    STRA=1 なので OVRA の値は無視されます. POLA=1 なので CWG1A の出力は CCP1 出力の反転となります.

  これは図 9.8b に示すアクティブ・ロウ方式の PWM 制御であることが分かります. OUT1 は右レールに, OUT2 は左レールにそれぞれ接続されますので, モータドライバ TB6643KQ のモータ出力電圧は OUT1 ≥ OUT2 なので模型車両は前進します.
- 後進時
  前進時とまったく逆でマイコンのピンの出力は以下の通りになりますが, 詳しい説明は省略します.
  - IN1 = RC6 = CCP1 の出力の反転
  - IN2 = RC5 = H

  これも図 9.8b に示すアクティブ・ロウ方式の PWM 制御であり, 前進の場合と極性が逆になっていることが分かります. OUT1 は右レールに, OUT2 は左レールにそれぞれ接続されますので, モータドライバ TB6643KQ のモータ出力電圧は OUT1 ≤ OUT2 なので模型車両は後進します.
- ニュートラル時
  STRA=0, STRB=0 に設定されますから, マイコンのピンの出力は以下の通りになります.
  - IN1 = RC6 = H
  - IN2 = RC5 = H

  モータドライバ TB6643KQ のモータ出力電圧は常に OUT1 = OUT2 なので, 模型車両は走行を停止し

ます.

ソースファイル番号 9.7: mcr-motor.c

```
/*-----------------------------------------------------------
 * モータ駆動関係の周辺モジュール設定 (PIC16F18877/18857)
 * 使用周辺モジュール: TMR2, CCP1, CWG1
 * PWM周波数: 31.25kHz
 * 出力ピン:  CWG1A, CWG1B
 * 出力形式:  Active Low
 *-----------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "mcr-motor.h"

/*-------------------------------------------*/
/* MCR_Motor */
void MCR_Motor_Init(void)
{
  TMR2_Init();
  CCP1_Init();
  CWG1_Init();
}

/*-------------------------------------------*/
/* TMR2 */
/*   PWM周波数 = 31.25kHz
 *      (32MHz/(31.25kHz*4*1)) - 1 = 255 = PR2
 */
void TMR2_Init(void)
{
  T2CONbits.TMR2ON = 0; /* Disabled */
  T2CONbits.T2OUTPS = 0;  /* Postscale: 1:1 */
  T2CONbits.T2CKPS  = 0;  /* Prescale:  1:1 */
  T2CLKCONbits.CS = 0x01; /* Clock: FOSC/4 */
  T2HLTbits.MODE  = 0x00; /* Software Control */
  T2PR  = 0xFF;           /* PWM周波数: 31.25kHz */
  T2TMR = 0;
  TMR2IF = 0;
}

void TMR2_Start(void)
{
  T2CONbits.TMR2ON = 1; /* Enabled */
}

/*-------------------------------------------*/
/* CCP1 */
void CCP1_Init(void)
{
  CCP1CONbits.CCP1EN   = 1;   /* Enabled */
  CCP1CONbits.CCP1FMT  = 0;   /* 右寄せ形式 */
  CCP1CONbits.CCP1MODE = 0xF; /* PWMモード */
  CCPTMRS0bits.C1TSEL  = 0x1; /* TMR2 */
  CCPR1L = 0x00;
  CCPR1H = 0x00;
}

void CCP1_SetDutyCycle(uint16_t duty)
{
  CCPR1H = duty >> 8; /* 2 MSBs */
  CCPR1L = duty;      /* 8 LSBs */
}

/*-------------------------------------------*/
/* CWG1 */
void CWG1_Init(void)
{
  CWG1CON0bits.CWG1EN = 0; /* Disabled */
```

```
67    CWG1CON0bits.CWG1MODE = 0; /* ステアリングモード */
68    CWG1CON1bits.POLA = 1; /* Inverted Polarity */
69    CWG1CON1bits.POLB = 1; /* Inverted Polarity */
70    CWG1STRbits.OVRA = 1; /* Active Low */
71    CWG1STRbits.OVRB = 1; /* Active Low */
72    CWG1STRbits.STRA = 0; /* Steering A (0:OVRA, 1:Inv-Data) */
73    CWG1STRbits.STRB = 0; /* Steering B (0:OVRB, 1:Inv-Data) */
74    CWG1CLKCON = 0; /* Clock Source: FOSC */
75    CWG1ISM = 0x01; /* Data Source: CCP1 */
76    CWG1CON0bits.CWG1EN = 1; /* Enabled */
77  }
78
79  void MotorDriver_TB6643KQ_Reset(void)
80  {
81    /* 割り込み禁止/許可状態を保存 */
82    int8_t ogie  = GIE;
83    int8_t opeie = PEIE;
84    /* 割り込み禁止 */
85    GIE  = 0;
86    PEIE = 0;
87    /* IN1,IN2:=LOW,LOW */
88    CWG1STRbits.OVRA = 0;
89    CWG1STRbits.OVRB = 0;
90    CWG1STRbits.STRA = 0;
91    CWG1STRbits.STRB = 0;
92    __delay_us(5); /* 1uS以上 */
93    CWG1STRbits.OVRA = 1;
94    CWG1STRbits.OVRB = 1;
95    /* 割り込み禁止/許可状態を復元 */
96    GIE  = ogie;
97    PEIE = opeie;
98  }
```

－＊－＊－＊－

本章では基本的なパワーパックを作成しました. しかし以下のように改良の余地はまだまだあります. 後の章では改良方法の実装を示しますので, 本章の内容の理解を完全にした上で引き続き勉強に精進して下さい.

- ボリュームを 0 にせずに電源スイッチを入れると模型車両は急発進してしまいます. ボリュームが 0 でないまま電源を入れても急発進しない機能を第 10 章のファームウエアで実装します.
- 模型車両の走行中に走行方向スイッチを切り替えると模型車両は急逆走してしまいます. 走行中に走行方向スイッチを切り替えても急逆走をしない機能を第 10 章のファームウエアで実装します.
- 本章のファームウエアではボリュームを回すと直ちにそれに応じた走行速度で模型車両が走行します. 実車のようにゆっくりと加減速をする機能を第 10 章のファームウエアで実装します.
- 本ファームウエアでは PWM 周波数 31.25kHz の PWM 波形を生成します. これは可聴域を越えた周波数なので耳障りな音はモータからは聞こえません. しかし周波数が高いがためにモータによっては低速域でのトルクが小さく, 思うように低速走行できない模型車両が多数あります. 低速時のトルクを強力にする機能を第 10 章のファームウエアで実装します.

課題

1. 先頭車両, 動力車両, 後尾車両の 3 両編成の列車を考えます. モータドライバをアクティブ・ハイで制御した場合, PWM オフ期間に線路を伝って先頭車両と後尾車両に届き, ヘッドライトとテールライトが点灯すると本文で述べました. なぜこれが起きるのでしょうか. 編成全体での回路図を描き, 電流の流れを示すことでその理由を説明して下さい.

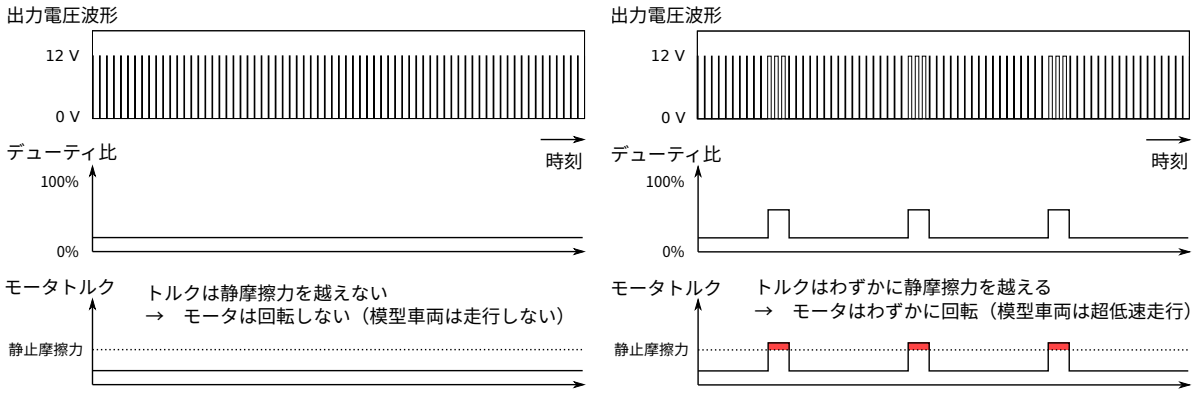
2. PWM 周波数を 100Hz とするようにファームウエアを変更して下さい. ソースファイルの変更内容を示すとともに, PWM 周波数が 100Hz となることを説明して下さい.

第 10 章

自動加減速・低速ブースト機能型
パワーパック

本章では自動加減速・低速ブースト機能を有するパワーパックを紹介します. ハードウエアは第 9 章で紹介したものをそのまま使用して, ファームウエアを高機能なものに入れ替えます. 本章で紹介するファームウエアで新たに追加される特徴は以下の通りです.

- 低速ブースト機能
  超低速走行を実現します. 図 10.1 に示すうに, 低速走行時には強パルスを 1 秒間に数十回の頻度で加えてモータのトルクが静摩擦力を越えるようにするとモータはわずかに回転して, 超低速走行が可能になります. なおここで強パルスとは, 高 PWM デューティ比を短時間出力することを指しています.
- 自動加減速機能



(a) 低速ブースト機能なし　　**(b)** 低速ブースト機能あり

図 10.1: 低速ブースト機能が超低速走行を可能とする原理

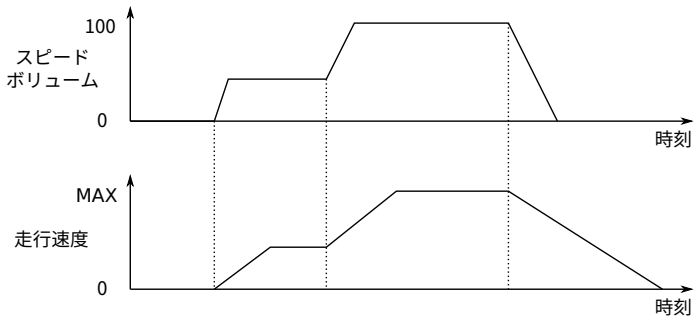


図 10.2: 自動加減速機能

急にスピードボリュームを大きく操作してもゆっくりと加速・減速をする, いわゆるトランジスタコントローラに似た機能を実現します. 図 10.2 に加減速の様子を示します. 加速度と減速度は可変抵抗器で設定します.

- ラッシュ防止機能
  電源を入れた瞬間に模型車両が急発進するのを防止する機能です. パワーパックの電源を入れた時にスピードボリュームがゼロの位置に戻るまで模型車両の走行をしません.
- 走行中逆転防止機能
  走行方向のスイッチを模型車両が走行中に切り替えても, 走行方向は切り替えない機能です. つまり走行方向の実際の切り替えを停車時だけに限定する機能です.
- 緊急停止機能
  走行中の模型車両へ供給する電圧を即座にゼロにして緊急停止する機能です. 自動加減速機能によるゆるやかな減速をしている場合には必須の機能です.
- 最高速度設定機能
  模型車両の走行する最高速度を 12V の 100% から 50% の範囲で設定できる機能です. 入れ替えやローカル線での低速走行の際の細かな速度調整に便利です.
- 常点灯機能
  模型車両が停車している間もヘッドライトや室内灯を点灯させる機能です.

本章で学ぶ内容は以下の通りです.

- 割り込みの基礎概念
- タイマ割り込み

注意

常点灯機能には共通規格が存在せず, 本パワーパックですべての模型車両で必ず常点灯ができるとは限りません. 電球式のヘッドライト等では加熱による模型車両の破損や火災も起き得ます. 電球式・LED 式に関わらず必ず各自で安全性を確認した上で使用して下さい.

本章のファームウエアの大まかな構造を図 10.3 に示します. 自動加減速の機能はブレゼンハムアルゴリズムで実現します. つまり, スピードボリュームで決まる目標走行速度へ向けて, 力行時には現在の走行速度をゆっくりと増加し, 制動時にはゆっくりと減少します.

このように実際に PWM デューティ比をモータドライバに出力するのはタイマ割り込みハンドラ内で行い, 定期的に高 PWM デューティ比を出力してトルク増強の実現を行います.

10.1　使用方法

本装置の使用方法を説明します. 図 10.4 が本パワーパックの部品配置例で, 図 9.4 (161 ページ) の再掲です. AC アダプタと線路への接続の接続を図 10.5 に示します. これは第 9 章と同様な接続です.

操作部分と表示部分の説明をします. ハードウエアは第 9 章と全く同一ですが, そこでは使用しなかったスイッチや可変抵抗器を本ファームウエアで新たに使用します.

- 電源スイッチ PWR-SW
  回路全体の電源スイッチです. 電源を入れるとパイロットランプ PWR-ON が点灯します.
- 緑色 LED INDICATOR

```
void main(void)
{
  for(;;) {
    スピードボリュームを読む;
    speed_target = 目標速度を決定;
    speed = ブレゼンハムアルゴリズムで次の現在速度を決定;
    md = PWMデューティ比を計算;
  }
}

タイマ割り込みハンドラ:（低速ブーストの実現）
{
  switch (sb_state) { /*次のモータ出力*/
  case 通常期間状態:
    sb_state = 低速ブースト状態;
    PWMデューティ比mdをモータドライバに出力;
    次のタイマ割り込み発生時間を設定;
    break;
  case 低速ブースト状態:
    sb_state = 通常期間状態
    PWMデューティ比50%をモータドライバに出力; /*トルクの増強*/
    次のタイマ割り込み発生時間を設定;
    break;
  }
}
```

図 10.3: ファームウエア構造 (概要)

正常動作しているとき点灯します. ラッシュ防止機能および走行中逆転防止機能が作動している時には点滅します.

- 黄色 LED STOPPED
  停車時に点灯し, 走行時に消灯します.
- 可変抵抗器 VR-SPEED: スピードボリューム
  走行速度を決める可変抵抗器です.
- 走行方向スイッチ DIR
  走行方向を決めるスイッチで, 前進, 後進, ニュートラルの 3 つを切り替えて選択します.
- 押しボタンスイッチ ESTOP
  緊急停止をします. またショートによりモータドライバの過電流保護機能が動作していれば, その解除をします.
- 可変抵抗器 VMAX
  最大速度, すなわちスピードボリューム VR-SPEED をいっぱい回したときの速度を 50%~100% の間で設定します. この設定を小さくするとスロー走行を細やかに制御できます.
- 可変抵抗器 ACC
  加速度を設定します. この設定がゼロの場合には即座にスピードボリュームで指示された速度になります. この設定が大きくなるほど加速度は小さく, ゆっくりと加速します.
- 可変抵抗器 DEC

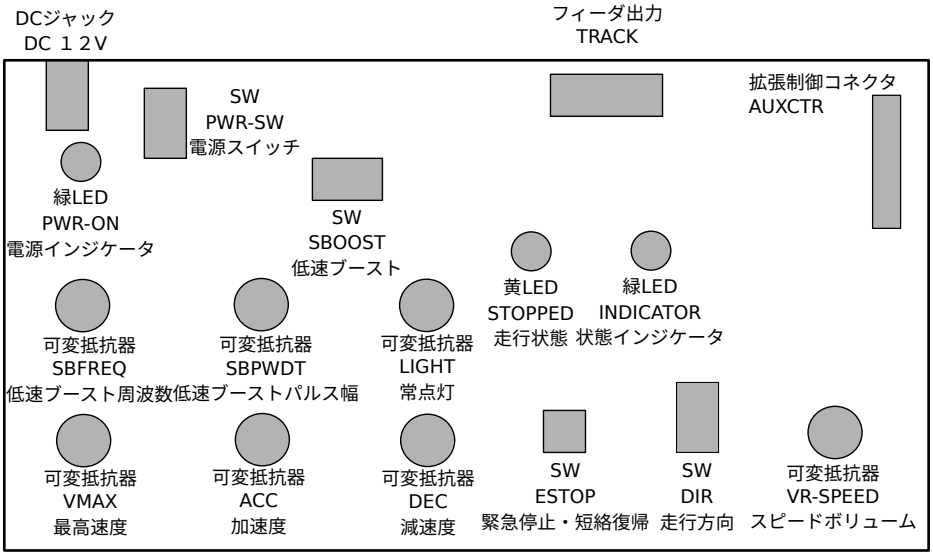


図 10.4: 操作表示部品の配置例と名称

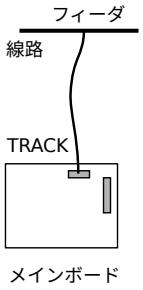


図 10.5: 接続方法

減速度を設定します. この設定がゼロの場合には即座にスピードボリュームで指示された速度になります. この設定が大きくなるほど減速度は小さく, ゆっくりと減速します.

- スイッチ SBOOST
  低速ブースト機能のオン・オフを切り替えます.
- 可変抵抗器 SBFREQ
  低速ブースト周波数を設定します. 0% の場合はパルスの発生回数が少なくてブーストは最も弱く, 100% の場合はブーストは最も強くなります. 詳しくは後述します.
- 可変抵抗器 SBPWDT
  低速ブーストパルス幅を設定します. 0% の場合はパルス幅が小さくてブーストは最も弱く, 100% の場合はブーストは最も強くなります. 詳しくは後述します.
- 可変抵抗器 LIGHT
  常点灯機能での輝度を設定します. 0% の場合は輝度は 0 で消灯状態で, 100% の場合は輝度は最大になります. 設定が大きいと模型車両が走り出しますし, 発熱量の大きな電球の場合は加熱により模型車両を痛める場合がありますから, 設定は各自で適切に行って下さい.

基本的な操作方法は以下の通りです.

1. 電源を入れる前に各種パラメータを標準的な位置に設定します. 走行方向スイッチ DIR は中点のニュートラルに, 最高速度 VMAX は最大に, 加速度 ACC と減速度 DEC はともにゼロに, スイッチ SBOOST はオフに, 低速ブースト周波数 SBFREQ と低速ブーストパルス幅 SBPWDT はともにゼロに, それぞれ設定しておきます. これらの設定は模型車両の様子を見ながら微調整して下さい.
2. スピードボリュームをゼロの位置にして電源 PWR-SW を入れます. もし位置がゼロでなければラッシュ防止機能により走行できず, その間は緑 LED INDICATOR が点滅します.
3. 走行方向スイッチ DIR を前進あるいは後進に切り替え, スピードボリューム VR-SPEED を回して模型車両を走行させます.
4. 脱線等でショートが発生した場合にはモータドライバの過電流検出回路が作動して出力が停止します. 過電流検出の解除には, ショートの原因を取り除いて緊急停止スイッチ ESTOP を押して下さい.

各種の設定方法を説明します.

- 加減速 ACC, DEC の調整
  設定はゼロから開始し, 徐々に大きくしてゆきます. 減速度が大きい場合にはすぐには停車できませんから, 緊急停止には押しボタンスイッチ ESTOP を押します.
- 低速ブースト SBFREQ, SBPWDT の調整
  スイッチ SBOOST をオンにしてスピードボリュームをごくわずかに回します. コアレスモータ搭載の模型車両などではこの時点で動き出すものもありますので, 低速ブースト機能は使わなくても十分に低速走行が可能です. しかしスピードボリュームを回してもなかなか走り出さない模型車両では低速ブースト機能が効果的となる可能性がありますので, 以下のように設定をします. まずスピードボリュームを模型車両が走り出さない位置まで回します. 次に可変抵抗器 SBFREQ と SBPWDT の 2 つの設定値を少しずつ大きくしてゆき, 模型車両がわずかに走り出すようにします. 2 つの値の組み合わせで微妙な違いがありますので, 後述の低速ブースト機能の説明を参考にして各自の模型車両に合った設定をして下さい.
- 常点灯 LIGHT の調整
  まずスピードボリュームをゼロにしておきます. LIGHT の設定はゼロから開始して, ヘッドライトが点灯し, しかも模型車両が動き出さない位置に設定します.

10.2　自動加減速機能の実現

自動加減速機能は, スピードボリュームを一気に変化させても模型車両の走行速度は徐々に変化してゆく機能です. 従来のパワーパックでトランジスタ・コントローラあるいは TAT (True Action Throttle) と呼ばれているものはコンデンサの充放電に基づくハードウエアでこの機能を実現していました. 本パワーパックではソフトウエアによりこの機能を実現します. 具体的には可変抵抗器による設定をパラメータとして, ブレゼンハムの線形増加アルゴリズムを用いて速度を少しづつ変化させます.

10.3　割り込みの概念

割り込み (interrupt) とはプログラムの実行とは別に非同期的に発生する事象の処理を可能とする機構です. 本書の多くのファームウエアでは割り込みと呼ばれている機構を使用しています. 割り込みの概念は組み込み機器においては必須といってよいもので, 外界とのインターフェースや実時間性の実現に使用します.

割り込みが発生すると図 10.6 に示すようにプログラムの実行を一時的に中断して割り込みハンドラと呼ばれるプログラムコードが自動的に呼び出されます. この機構により, 主となる処理の実行途中で生じるハードウエア的な信号の送受信や一定時間間隔での処理を, 自動的かつ即座に実行できます. ただし主となる処理と割り

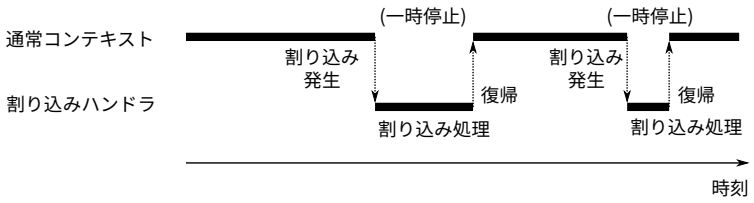


図 10.6: 割り込み処理

込みハンドラとの間での共用データへのアクセスは, データの一貫性が保たれるように注意深くプログラミングする必要があります.

次に PIC マイコンでの割り込みを説明します. 以下の形で割り込み処理を実現します.

- 割り込み許可ビットの設定
  周辺モジュールごとに割り込みを発生させるかどうかを設定するビットが用意されています. TMR1 モジュールでは TMR1IE ビットの値の設定により, TMR1 モジュールによる割り込み発生の許可・禁止を設定します. なお TMR1 モジュールでは, タイマの値がオーバーフローした時に割り込みが発生します.
- 割り込み発生フラグビット
  割り込みが発生すると, 割り込み要因に対応するレジスタのフラグビットは自動的に 1 に設定されます. そして明示的に 0 をこのビットに代入するまでは値は 1 であり続けます. TMR1 モジュールでの割り込み発生フラグビットは TMR1IF です.
- 割り込みハンドラ (interrupt handler)
  割り込みハンドラとは割り込み処理を行うプログラムコードです. PIC 16F シリーズでは割り込み要因に関わらず割り込みハンドラはひとつだけで, 割り込み要因を割り込みハンドラ内で分析をして適切な割り込み処理を行います.
  XC8 コンパイラにおいて割り込みハンドラとする C 言語の関数には, 特別な修飾子__interrupt() を付加をして関数宣言をします[*1]. 特別な修飾子の付加が必要な理由は, 割り込みハンドラと普通の関数とでは呼び出しと復帰のための機械語命令列が異なり, C コンパイラが適切な機械語命令列を生成できるようにするためです.
- 割り込みの禁止 (disable) と許可 (enable)
  割り込みハンドラが呼び出されては困る時には割り込みを禁止し, それ以外の時には割り込み許可とします. 割り込み禁止は, 主たる処理と割り込みハンドラとの間で共有データの一貫性が崩れないようにするために使用します.
  割り込み許可ビットと割り込み発生フラグビットがともに 1 の時, すなわちともに割り込みが許可されている時にのみ割り込みハンドラが呼び出されます. マイコン全体での割り込みの禁止と許可を設定するビットが 2 つあります. GIE ビットは全ての割り込みの許可・禁止を設定するビットで, PEIE ビットで周辺モジュールによる割り込みの許可・禁止を設定するビットです.
  割り込み要因が発生すると対応する割り込み発生フラグビットが 1 になりますが, 割り込みが禁止されていれば直ちには割り込みハンドラは呼び出されません. 割り込みハンドラの呼び出しは割り込み許可となるまで延期されます. つまり割り込みが許可された時になって割り込みハンドラが呼び出されます.
- 割り込みハンドラの呼び出し

[*1] XC8 コンパイラのバージョン 1 では修飾子 interrupt を使用します.

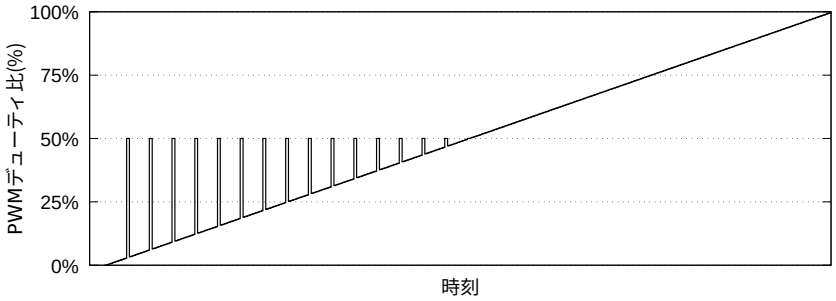


図 10.7: 低速ブースト機能による PWM デューティ比の時間変化 (時間とともに速度を徐々に上げてゆく場合)

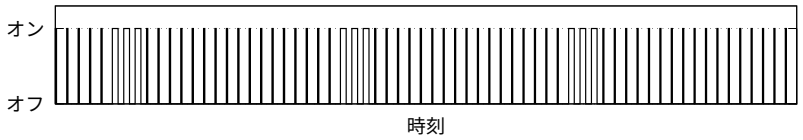


図 10.8: 低速ブースト機能による高 PWM デューティ比期間を挿入した PWM 波形

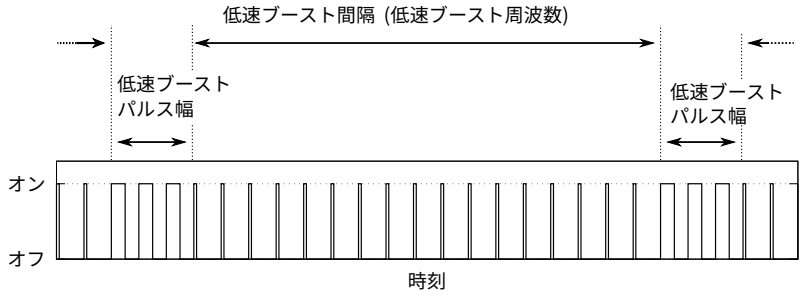


図 10.9: PWM 波形における低速ブーストパルス幅と低速ブースト周波数

割り込みハンドラの呼び出しはマイコン内部のハードウエアにより行われます. 割り込みが発生するとマイコンは実行中のプログラムの一時中断を行い, 割り込みハンドラを呼び出します. そして割り込みハンドラの実行が終ると復帰します. 実行中のプログラムはこの一時中断と復帰を意識する必要はなく, すべてハードウエアにより自動的に行われます.

10.4　低速ブースト機能の実現

低速ブースト機能の詳細とファームウエアでの実現方法の概要を説明します. 低速ブースト機能の目的はモータのトルクの増加であり, 1 秒間に数十回の頻度で強パルスを出力することで実現します. 現在のファームウエアでは, PWM デューティ比 50% を強パルスとしています. 様々な実験を行い試行錯誤した結果, この値が最も平均的に低速ブースト機能が良く機能するようです. もちろん全ての模型車両でうまく機能するかは分かりません. コアレスモータ搭載の模型車両では低速ブースト機能はオフにする方が良いようです.

低速ブースト機能による出力の例を図 10.7 に示します. この図では, 横軸が時間で, 縦軸が PWM デューティ比です. 時刻 0 に PWM デューティ比 0 から開始して, 時間の進行とともに基本の PWM デューティ比を 100% まで増加した場合を示しています. 基本の PWM デューティ比が 50% 未満の場合は, 定期的にごく瞬間的に PWM デューティ比を 50% にします. すると図 10.1 に示した通り瞬間的にモータの駆動力は静摩擦力を上回り, モータはごくわずかに回転して低速走行が可能となります.

図 10.8 では, 基本の PWM デューティ比が 10% で, 低速ブースト機能により高 PWM デューティ比 50% の

期間が挿入されている様子を示します. この図の横軸は時間で, 縦軸はモータのオン・オフを表しています.

低速ブースト機能の設定には図 10.9 に示す通り 2 つのパラメータがあります.

- 低速ブーストパルス幅 SBPWDT
  1 回あたりの強パルスを出力する時間の長さです. この設定値が大きいと 1 回あたりの強パルスの出力時間が長くなります.
- 低速ブースト周波数 SBFREQ
  単位時間あたり強パルスを出力する頻度です. この設定値が大きいと低速ブーストパルスを出力する頻度が高くなり, 単位時間あたりに強パルスを出力する回数が多くなります.

10.5　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 10.1: 【ソースファイル一覧】 パワーパック: 自動加減速・低速ブースト機能型

初	頁	番号	ファイル名	内容	行数
✓	189	10.2	PowerPack-SlowBoost.c	パワーパックメイン関数	51
✓	189	10.1	pp.h	パワーパック関数宣言: ヘッダ	46
	172	9.2	pp-const.h	パワーパック定数: ヘッダ	17
✓	194	10.5	pp-control.c	自動加減速パワーパック: 実装	103
✓	193	10.4	pp-control-vr.c	ボリューム制御インターフェース: 実装	58
✓	191	10.3	pp-misc.c	パワーパック諸関数: 実装	111
✓	196	10.6	pp-sb.h	低速ブースト機能: ヘッダ	8
✓	198	10.7	pp-sb.c	低速ブースト機能: 実装	171
	129	6.1	incr-lin.h	線形増加アルゴリズム: ヘッダ	9
	130	6.2	incr-lin.c	線形増加アルゴリズム: 実装	78
	170	9.1	io.h	I/O ピン割付: ヘッダ	50
	173	9.4	mcr.h	MCR・周辺モジュール: ヘッダ	8
	174	9.5	mcr.c	MCR・周辺モジュール: 実装	84
	175	9.6	mcr-motor.h	モーター制御周辺モジュール: ヘッダ	25
	177	9.7	mcr-motor.c	モーター制御周辺モジュール: 実装	98
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76

計 1002 行 (うち初出 548 行, 既出 454 行)

低速ブースト機能の観点からの本ファームウエアの概要を図 10.10 に示します. 低速ブースト機能の実現は, 通常の PWM デューティ比を出力する状態と強パルスを出力する状態の 2 状態を交互に遷移することで行います. 状態の遷移はタイマ割り込みの発生により引き起こし, 低速ブースト期間の長さの設定は, タイマ割り込みが発生するまでの時間の長さの設定により行います.

ソースファイル pp.h (パワーパック関数宣言: ヘッダ)

パワーパックで使用する各種の変数や関数の型を宣言します. p_で始まる変数は, 可変抵抗器で設定された各種のパラメータを保持する変数です.

```
【ソースファイル PowerPack-SBoost.c】

void main(void)
{
  初期化;
  pp_control();
}

__interrupt() void isr(void)   /* 割込ハンドラ */
{
  if (TMR1が割り込み要因) {
    TMR1_isr();
  }
}

【ソースファイル pp-control.c】

void pp_control(void)
{
  for(;;) {
    md = モータPWMデューティ比;
    pp_sb_duty_set(md);
  }
}

【ソースファイル pp-sb.c】

uint16_t sb_mduty = 0;  /* 通常期間のPWMデューティー値 */
状態 sb_state = 通常期間状態;  /* 通常期間/ブースト期間の切替用 */

void pp_sb_duty_set(uint16_t mduty)
{
  sb_mduty = mduty;
}

void TMR1_isr(void)
{
  switch (sb_state) {
  case 通常期間状態:     /* 通常期間の終了 (ブースト期間の開始) */
    sb_state = ブースト期間状態;
    PWMデューティ比出力 = 低速ブースト用高PWMデューティ比;
    次のTMR1割り込み(ブースト期間終了後)をスケジュール;
    break;
  case ブースト期間状態: /* ブースト期間の終了 (通常期間の開始) */
    sb_state = 通常期間状態;
    PWMデューティ比出力 = sb_duty;
    次のTMR1割り込み(通常期間終了後)をスケジュール;
    break;
  }
}
```

図 10.10: 低速ブースト機能の実現方法 (概要)

ソースファイル番号 10.1: pp.h

```
#if !defined(__PP_H__)
#define __PP_H__

/* 走行方向 */
enum dir_e {
  DIR_NONE = 0,
  DIR_FWD,
  DIR_REV
};

/* パラメータ値 */
extern volatile uint16_t  p_speed;
extern volatile uint16_t  p_sbfreq;
extern volatile uint16_t  p_sbpwdt;
extern volatile uint16_t  p_accel;
extern volatile uint16_t  p_decel;
extern volatile uint16_t  p_vmax;
extern volatile uint16_t  p_light;

/* ADC 読み取り */
extern uint16_t   pp_adc_read(uint8_t ch);
/* パラメータ読み込み */
extern void       pp_param_read(void);
/* 走行方向SW読み込み */
extern enum dir_e pp_dir_read(void);
/* モーター回転方向出力 */
extern void       pp_motor_dir_set(enum dir_e dir);
/* 指定ミリ秒の経過を待機 */
extern void       pp_delay_ms(uint16_t t_msec);
/* 制御器の定位置復帰の待機  */
extern void       pp_home_position(void);
/* パワーパック走行制御 */
extern void       pp_control(void);
/* 目標走行速度を求める */
extern uint16_t   pp_controler_speed(uint16_t speed);
/* 加速度&減速度を求める */
extern uint16_t   pp_controler_accel(uint16_t target_speed,
                                     uint16_t speed);
/* 走行速度SPDをモーターデューティー比(最高時:VMAX)に変換 */
#define PP_CONV_SPEED_TO_MDUTY(SPD,MAX)                          \
  (((uint32_t)(SPD) * (uint32_t)(MAX) + 1023) >> 10)
/* 逆変換 */
#define PP_CONV_MDUTY_TO_SPEED(MD,MAX)          \
  (((uint32_t)(MD) << 10) / (uint32_t)(MAX))

#endif /*__PP_H__*/
```

ソースファイル PowerPack-SlowBoost.c (パワーパックメイン関数)

これがパワーパックのメインループを構成するソースファイルです. 初期化の部分は従来とほぼ同一です. ただし, 本ファームエアではタイマ割り込みを使用して低速ブースト機能を周期的に実行しますが, この実現方法は後述します. 本ソースファイルでは以下の 2 つの関数の呼び出しを繰り返します.

1. 関数 pp_home_position()
   ラッシュ防止機能を実現する関数です. この関数ではスピードボリュームがゼロの位置に戻るまで待機し続けて, ゼロの位置に戻れば実行を終了します.
2. 関数 pp_control()
   この関数内が実質的なパワーパックの機能を実現します. 詳しくは後述します.

ソースファイル番号 10.2: PowerPack-SlowBoost.c

```
/*************************************************************
 * パワーパック(低速ブースト・加減速機能PWM型) / PIC16F18857
 * by Hirotsugu Kakugawa
 *************************************************************/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "mcr-motor.h"
#include "adc.h"
#include "io.h"
#include "pp.h"
#include "pp-const.h"
#include "pp-sb.h"

/*-----------------------------------------------------------
 * MAIN
 *-----------------------------------------------------------*/
void main(void)
{
  /* システム初期化 */
  System_Init();
  /* ウオッチドッグタイマ: 256 [mSec] */
  WDTCON0bits.WDTPS = 0x08; /* 0b01000 */
  CLRWDT();
  /* モータ出力初期化 */
  MOTOR_DIR_NONE();
  MOTOR_PWM_START();
  /* 低速ブースト初期化 */
  pp_sb_init();
  /* 割り込み許可 */
  GIE  = 1;
  PEIE = 1;
  /* 主ループ */
  for (;;) {
    CLRWDT();
    /* 制御器の定位置復帰を待機  */
    pp_home_position();
    /* パワーパック走行制御 */
    pp_control();
  }
}

/*-------------------------------------------*/
/* 割込ハンドラ */
void interrupt isr(void)
{
  if ((TMR1IE != 0) && (TMR1IF != 0)) {
    TMR1IF = 0;
    TMR1_isr();
  }
}
```

このソースファイルでは割り込みハンドラも記述しています. 本ファームウエアでは TMR1 割り込みのみを使用しますので, TMR1 に対応する割り込み発生フラグビット TMR1IF を調べて, このビットをクリアしてから実際の処理を行う別の関数 TMR1_isr() を呼び出します.

ソースファイル pp-misc.c (パワーパック諸関数: 実装)

パワーパックの動作で有用な各種の関数です.

- 関数 pp_adc_read()
  ADC モジュールを使用して, 指定された ADC チャンネルの電圧値を読み取ります.
- 関数 pp_param_read()
  各種可変抵抗器の設定を読み取り, 実際の使用で適切な値の範囲になるように変換してパラメータ変数に代入します.

この関数内で割り込みを禁止している区域があります. その理由は, 割り込みハンドラ内でもパラメータ値の変数を参照しているためです. AD 変換結果の値は 2 バイト (16 ビット) ですが, PIC マイコンでは 1 バイト単位でメモリへのアクセスを行います. 1 バイト目を書き込み終わった直後に割り込みが発生し, 割り込みハンドラ内でパラメータ変数を 2 バイト読み込むことがもし仮に発生したとします. すると割り込みハンドラは 1 バイト目は新しい値を読み, 2 バイト目は古い値を読むことになり, データの一貫性が保たれていないことになります. そのような事態が生じないように, 通常プログラム部と割り込みハンドラ部で共有されているデータへのアクセスは割り込み禁止を行い, データの一貫性を保つようにします. より深く理解するには, オペレーティグシステム理論や並行プログラミング理論の教科書を参照してください.

- 関数 pp_dir_read()
  走行方向スイッチ DIR を読んで, 進行方向を返します.
- 関数 pp_motor_dir_set()
  モータドライバの出力を指定された走行方向に対応させます.
- 関数 pp_delay_ms()
  指定された時間を待機します.

ソースファイル番号 10.3: pp-misc.c

```
1  /*------------------------------------------------------
2   * パワーパック関係　諸関数
3   *------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "mcr-motor.h"
8  #include "adc.h"
9  #include "io.h"
10 #include "pp.h"
11 #include "pp-const.h"
12
13 /* パラメータ値 */
14 volatile uint16_t  p_speed  = 0;
15 volatile uint16_t  p_sbfreq = 0;
16 volatile uint16_t  p_sbpwdt = 0;
17 volatile uint16_t  p_accel  = 0;
18 volatile uint16_t  p_decel  = 0;
19 volatile uint16_t  p_vmax   = 0;
20 volatile uint16_t  p_light  = 0;
21
22 /*------------------------------------------------------
23  * ADC 読み取り
24  *------------------------------------------------------*/
25 uint16_t pp_adc_read(adc_ch_t ch)
26 {
27   /* ADC値が十分小さければ値0とみなす
28    * (可変抵抗器が厳密に0オームにならない場合への対応) */
29   adc_val_t v = ADC_Read(ch);
30   if (v < ADC_MARGIN) {
31     v = 0;
32   } else {
33     v = v - ADC_MARGIN;
34   }
35   return (uint16_t)v;
36 }
37
38 /*------------------------------------------------------
39  * 各種パラメータ読み込み
40  *------------------------------------------------------*/
41 void pp_param_read(void)
```

```
{
  /* 設定値(生)の読み込み */
  uint16_t vr_speed  = pp_adc_read(ADC_CH_SPEED);
  uint16_t vr_sbfreq = pp_adc_read(ADC_CH_SBFREQ);
  uint16_t vr_sbpwdt = pp_adc_read(ADC_CH_SBPWDT);
  uint16_t vr_accel  = pp_adc_read(ADC_CH_ACCEL);
  uint16_t vr_decel  = pp_adc_read(ADC_CH_DECEL);
  uint16_t vr_vmax   = pp_adc_read(ADC_CH_VMAX);
  uint16_t vr_light  = pp_adc_read(ADC_CH_LIGHT);
  /* 割り込み禁止 */
  int8_t ogie  = GIE;
  int8_t opeie = PEIE;
  GIE  = 0;
  PEIE = 0;
  /* パラメータ値の決定 */
  p_speed  = vr_speed;          /* 0 .. 1023 */
  p_sbfreq = vr_sbfreq;         /* 0 .. 1023 */
  p_sbpwdt = vr_sbpwdt;         /* 0 .. 1023 */
  p_accel  = vr_accel << 1;     /* 0 .. 2047 */
  p_decel  = vr_decel << 1;     /* 0 .. 2047 */
  p_vmax   = vr_vmax;           /* 0 .. 1023 */
  p_light  = vr_light >> 2;     /* 0 ..  128 */
  /* 割り込み禁止/許可状態を復元 */
  GIE  = ogie;
  PEIE = opeie;
}

/*-----------------------------------------------------------
 * 走行方向SW読込
 *-----------------------------------------------------------*/
enum dir_e pp_dir_read(void)
{
  enum dir_e  dir = DIR_NONE;
  if (SW_DIR1F()) {
    dir = DIR_FWD;
  } else if (SW_DIR1R()) {
    dir = DIR_REV;
  }
  return dir;
}

/*-----------------------------------------------------------
 * モーター回転方向出力
 *-----------------------------------------------------------*/
void pp_motor_dir_set(enum dir_e dir)
{
  switch (dir) {
  case DIR_FWD:
    MOTOR_DIR_FWD();
    break;
  case DIR_REV:
    MOTOR_DIR_REV();
    break;
  case DIR_NONE:
    MOTOR_DIR_NONE();
    break;
  }
}

/*-----------------------------------------------------------
 * 指定ミリ秒の経過を待機
 *-----------------------------------------------------------*/
void pp_delay_ms(uint16_t t_msec)
{
  while (t_msec != 0) {
    CLRWDT();
    __delay_ms(1);
    t_msec--;
  }
}
```

ソースファイル pp-control-vr.c (ボリューム制御インターフェース: 実装)

このソースファイルには可変抵抗器によるスピードボリューム操作に関係する関数を集めています. 第 23 章ではこのソースファイルを同じ名称の関数群を持つ別のソースファイルに置き換えてリモコン制御を実現します. このソースファイルには以下の関数が含まれています.

- pp_home_position()
  ラッシュ防止機能を実現する関数です. スピードボリュームが定位置であるゼロの位置に戻るまでループし続けます. このときに緑 LED INDICATOR を点滅させてラッシュ防止機能が作動中であることを示します. また, ループし続けている間に緊急停止スイッチ ESTOP が押されると, モータドライバの過電流検出回路による保護を解除します.
- pp_controler_speed()
  目標走行速度を返します. ブレゼンハムの線形増加アルゴリズムではこの値を目標値として, 走行速度の値を徐々に変化させていきます.
- pp_controler_accel()
  加速度と減速度を返します. 本ファームウエアでは単に可変抵抗器 ACC と DEC による設定値を返すだけです.

ソースファイル番号 10.4: pp-control-vr.c

```
1  /*-----------------------------------------------------------
2   * パワーパック関係　諸関数
3   *-----------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "mcr-motor.h"
8  #include "adc.h"
9  #include "io.h"
10 #include "pp.h"
11 #include "pp-const.h"
12
13 /*-----------------------------------------------------------
14  * 速度ボリュームの定位置復帰を待機
15  *-----------------------------------------------------------*/
16 void pp_home_position(void)
17 {
18   for (;;) {
19     CLRWDT();
20     /* ボリューム読み込み */
21     pp_param_read();
22     if (p_speed == 0) { /* 定位置へ復帰 */
23       break;
24     }
25     /* 未復帰: インジケータLEDを点滅 */
26     LED_INDICATOR_TOGGLE();
27     pp_delay_ms(T_LED_WARNING);
28     /* モータドライバ LSI TB6643KQ 制御 (過電流検出の解除) */
29     if (SW_EMG_STOP()) { /* 解除要求の発生 */
30       extern void MotorDriver_TB6643KQ_Reset(void);
31       MotorDriver_TB6643KQ_Reset();
32     }
33   }
34   LED_INDICATOR_OFF();
35 }
36
37 /*-----------------------------------------------------------
38  * 目標走行速度を求める
39  *-----------------------------------------------------------*/
40 uint16_t pp_controler_speed(uint16_t speed)
```

```
41 {
42   return p_speed;
43 }
44
45 /*-----------------------------------------------------------
46  * 加速度&減速度を求める
47  *-----------------------------------------------------------*/
48 uint16_t pp_controler_accel(uint16_t target_speed,
49                             uint16_t speed)
50 {
51   uint16_t a = 0;
52   if (target_speed > speed) {
53     a = p_accel; /* 加速 */
54   } else {
55     a = p_decel; /* 減速 */
56   }
57   return a;
58 }
```

ソースファイル pp-control.c (自動加減速パワーパック: 実装)

パワーパックの動作を実現する中心的な部分をこのソースファイルに記述しています. ループの内側でのおおまかな動作は以下の通りです.

1. 緊急停止スイッチ ESTOP が押されているか調べます. もし押されていればモータドライバの過電流検出回路による保護を解除して, ループを抜けて模型車両への緊急停止を行います.
2. 各種パラメータを読み込みます. 可変抵抗器の設定や走行方向スイッチ DIR などです.
3. 走行方向の切り替えを行います. 走行中逆転防止機能はここで実現しています. 走行方向スイッチ DIR の切り替えがある場合に走行速度がゼロならば走行方向の変更を行います. もしゼロでなければ走行方向は変更せず, 緑 LED INDICATOR を点滅させて注意喚起をします. 走行速度がゼロになるか, あるいは走行方向スイッチ DIR が元に位置に戻るまでは緑 LED INDICATOR を点滅し続けます. この間は走行中の模型車両は同じ方向で走行を続け, スピードボリュームを操作して速度の制御が可能です.
4. 目標走行速度と加減速度をもとに, 走行速度をブレゼンハムの線形増加アルゴリズムを使用して求めます. 自動可減速機能はここで実現しています. また, 最高速度の設定に応じた比例計算により, 走行速度を PWM デューティ比に換算します.
5. 走行速度に対応する PWM デューティ比をモータドライバが出力するようにします. ただし直ちに周辺モジュールへ PWM デューティ比を設定はせず, 低速ブースト機能を実現する関数を呼び出します.
6. 速度がゼロの場合には黄 LED STOPPED を点灯して, 走行方向スイッチ DIR の切り替えが可能であることを表示します.

ソースファイル番号 10.5: pp-control.c

```
1  /*-----------------------------------------------------------
2   * パワーパック走行制御
3   *-----------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "mcr-motor.h"
8  #include "adc.h"
9  #include "io.h"
10 #include "pp.h"
11 #include "pp-const.h"
12 #include "pp-sb.h"
13 #include "incr-lin.h"
14
15 uint16_t   speed = 0;
```

```
16  uint16_t   target_speed = 0;
17  enum dir_e dir_next = DIR_NONE;
18  enum dir_e dir_curr = DIR_NONE;
19
20  /*-----------------------------------------------------------
21   * パワーパック走行制御
22   *-----------------------------------------------------------*/
23  void pp_control(void)
24  {
25    /* モータドライバIC のリセット */
26    MotorDriver_TB6643KQ_Reset();
27    /* 現在走行方向 */
28    dir_next = DIR_NONE;
29    dir_curr = pp_dir_read();
30    pp_motor_dir_set(dir_curr);
31    /* 現在走行速度 */
32    target_speed = 0;
33    speed = 0;
34    incr_lin_init(0);
35    pp_sb_duty_set(0);
36    /* インジケータLEDの点滅時間管理 */
37    uint16_t t_led = 0;
38    /* 状態インジケータLED点灯 */
39    LED_INDICATOR_ON();
40    /* 停車中表示 LED 点灯 */
41    LED_STOPPED_ON();
42    /* 主ループ */
43    for (;;) {
44      pp_delay_ms(RUN_INTERVAL);
45      CLRWDT();
46      /* 緊急停止/モータードライバLSI TB6643KQの過電流検出の解除 */
47      if (SW_EMG_STOP()) { /* 解除要求の発生 */
48        break;
49      }
50      /* パラメータの読込 */
51      pp_param_read();
52      /* 走行方向の読込 */
53      dir_next = pp_dir_read();
54      /* 走行方向の切替(必要に応じて) */
55      if (dir_curr == dir_next) { /* 同一の走行方向が継続 */
56        if (dir_next == DIR_NONE) {
57          LED_INDICATOR_OFF(); /* ニュートラル */
58        } else {
59          LED_INDICATOR_ON();  /* 前進/後進*/
60        }
61        t_led = 0;
62      } else { /* 走行方向SW切替が発生 */
63        if (speed == 0) { /* 現在走行速度が0ならば切替 */
64          if (dir_next == DIR_NONE) {
65            LED_INDICATOR_OFF(); /* ニュートラル */
66          } else {
67            LED_INDICATOR_ON();  /* 前進/後進*/
68          }
69          dir_curr = dir_next;
70          pp_motor_dir_set(dir_curr);
71          t_led = 0;
72        } else { /* 走行中の方向切替は禁止: 警告点灯(LED点滅) */
73          if (t_led >= T_LED_WARNING) {
74            LED_INDICATOR_TOGGLE();
75            t_led = 0;
76          } else {
77            t_led += RUN_INTERVAL;
78          }
79        }
80      }
81      /* 目標走行速度 */
82      target_speed = pp_controler_speed(speed);
83      /* 加減速度 */
84      uint16_t a = pp_controler_accel(target_speed, speed);
```

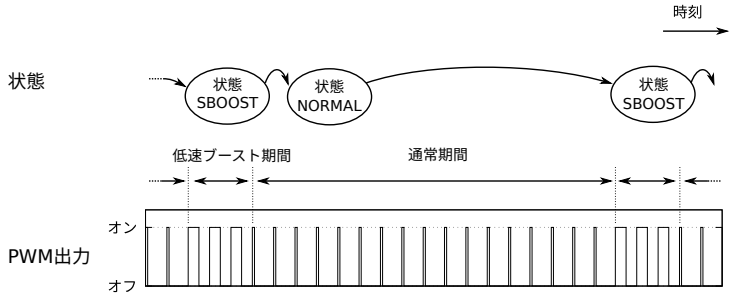


図 10.11: 通常期間と低速ブースト期間での状態遷移

```
85      /* 次の走行速度 */
86      speed = incr_lin_next(a, SPEED_INC_DY, target_speed);
87      /* 走行速度を変換して，モーターPWMデューティー比を得る */
88      uint16_t md = PP_CONV_SPEED_TO_MDUTY(speed, p_vmax);
89      /* モーターPWMデューティー比を出力 */
90      pp_sb_duty_set(md);
91      /* 停車中表示 LED */
92      if ((speed == 0) || (dir_curr == DIR_NONE)) {
93        LED_STOPPED_ON();
94      } else {
95        LED_STOPPED_OFF();
96      }
97    }
98    /* 停止 */
99    MotorDriver_TB6643KQ_Reset();
100   pp_motor_dir_set(DIR_NONE);
101   pp_sb_duty_set(0);
102   LED_INDICATOR_OFF();
103 }
```

ソースファイル pp-sb.h (低速ブースト機能: ヘッダ)

低速ブースト機能を実現する関数のプロトタイプ宣言を行います.

ソースファイル番号 10.6: pp-sb.h

```
1  #if !defined(__PP_SB_H__)
2  #define __PP_SB_H__
3
4  extern void pp_sb_init(void);
5  extern void pp_sb_duty_set(uint16_t mduty);
6  extern void TMR1_isr(void);
7
8  #endif /*__PP_SB_H__*/
```

ソースファイル pp-sb.c (低速ブースト機能: 実装)

低速ブースト機能を実現します. 図 10.11 に示すように, 通常の PWM デューティ比の期間と強パルスの期間が交互に現れる有限状態機械を用います.

- 状態 NORMAL: 通常期間
  模型車両の走行速度に従った PWM デューティ比を出力します.
- 状態 SBOOST: 低速ブースト期間
  低速ブーストパルス幅 SBPWDT と低速ブースト周波数 SBFREQ の設定に従った期間だけ, 強パルスすなわちデューティ比 50% の PWM 波を出力します.

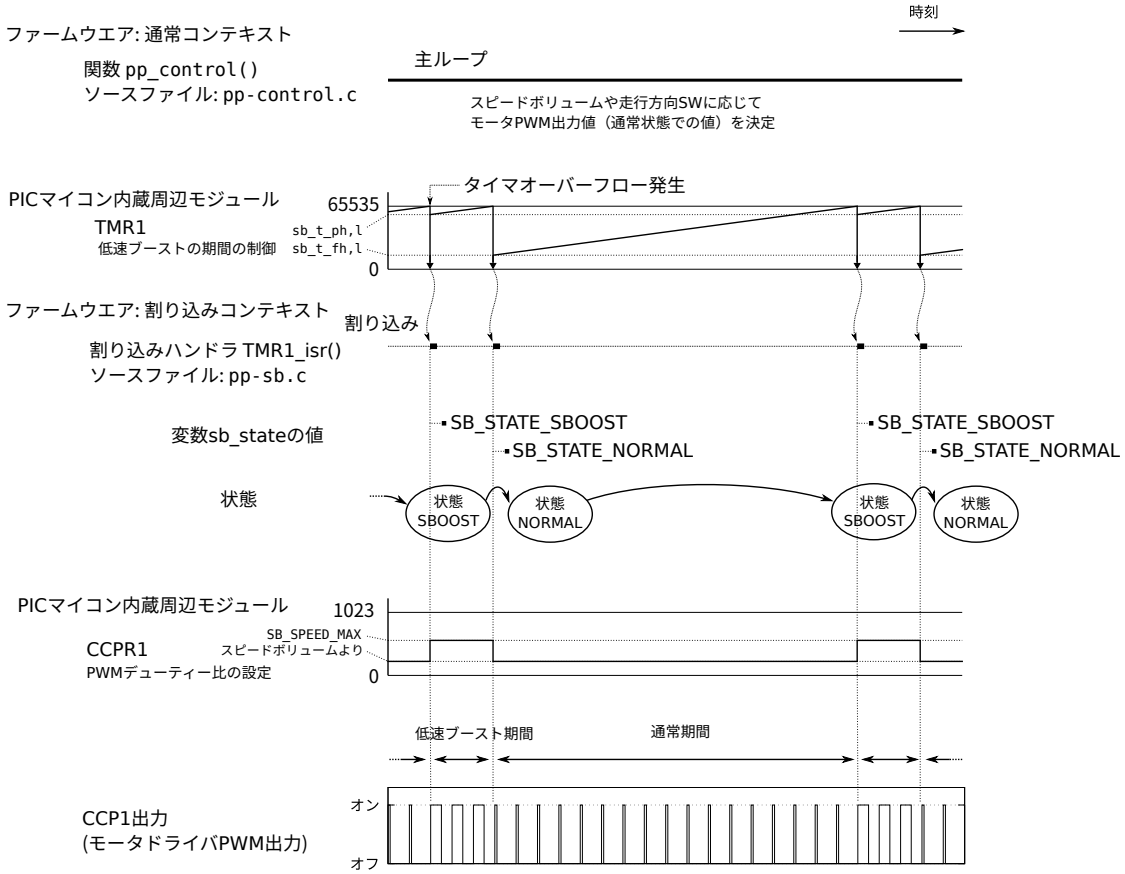


図 10.12: 通常期間と低速ブースト期間の状態遷移の実現方法と PWM 波出力

これら 2 つの状態間の遷移は時間の経過によって行います. ソフトウエアだけで正確な時間の計測は困難ですので, ハードウエアによるタイマ TMR1 の割り込みを使用し, 割り込みハンドラ内で状態遷移をします. タイマ設定と状態遷移は以下の通りです.

- タイマ割り込み発生時の状態が NORMAL の場合
  状態 SBOOST へ遷移します. 低速ブースト期間の終了時にタイマ割り込みが発生するようにタイマ TMR1 の値を設定します.
- タイマ割り込み発生時の状態が SBOOST の場合
  状態 NORMAL へ遷移します. 通常期間の終了時にタイマ割り込みが発生するようにタイマ TMR1 の値を設定します.

このソースファイルには以下の関数が含まれています. なお通常期間と低速ブースト期間の状態遷移の実現方法の概略を図 10.12 に示します.

- 関数 pp_sb_init()
  低速ブースト機能の初期化を行います.
- 関数 pp_sb_duty_set()
  通常期間に出力する PWM デューティ比を, 引数で指定された値に設定します. このときに通常期間および低速ブースト期間に対応するタイマの設定値も計算します.

- 関数 TMR1_isr()
  割り込みハンドラで, TMR1 の割り込みが発生した時に呼び出され, 通常状態と低速ブースト状態の間の状態遷移を行います. そしてそれぞれの期間の長さに対応するタイマ値を TMR1 のレジスタに設定して, 次のタイマ割り込みの発生をスケジュールします. スイッチにより低速ブースト機能がオフになっている場合は状態遷移はしますが, 低速ブースト期間で出力される PWM デューティ比は通常期間での PWM デューティ比と同じとしています.
  常点灯機能はここで実現しています. 最終的に出力する PWM デューティ比が, 常点灯 PWM デューティ比よりも小さくならないようにします.
- 関数 TMR1_init()
  タイマモジュール TMR1 の初期化を行います. 通常状態と低速ブースト状態の交代がちょうど良い頻度で発生するタイマのカウントアップの速度とします.

ソースファイル番号 10.7: pp-sb.c

```
1  /*-------------------------------------------------------
2   * 低速ブースト機能
3   *   実装の概要
4   *   - TMR1割込発生ごとに通常期間とブースト期間を交代させる
5   *-------------------------------------------------------*/
6  #include <xc.h>
7  #include <stdint.h>
8  #include "mcr.h"
9  #include "mcr-motor.h"
10 #include "adc.h"
11 #include "io.h"
12 #include "pp.h"
13 #include "pp-const.h"
14 #include "pp-sb.h"
15
16 /* 低速ブーストをOnとする速度の最大値 */
17 #define SB_SPEED_MAX      512
18
19 /* 通常期間/ブースト期間 */
20 enum sb_state_e {
21   SB_STATE_NORMAL,
22   SB_STATE_SBOOST,
23 };
24 volatile static enum sb_state_e sb_state = 0;
25
26 /* モーター出力のPWMデューティ値 */
27 volatile uint16_t sb_mduty = 0;
28
29 /* タイマ設定値 */
30 volatile static uint8_t  sb_t_fh = 0; /* TMR1(H): 周波数 */
31 volatile static uint8_t  sb_t_fl = 0; /* TMR1(L): 周波数 */
32 volatile static uint8_t  sb_t_ph = 0; /* TMR1(H): パルス幅 */
33 volatile static uint8_t  sb_t_pl = 0; /* TMR1(L): パルス幅 */
34
35 void TMR1_Init(void);
36
37 /*-------------------------------------------------------
38  * 初期化
39  *-------------------------------------------------------*/
40 void pp_sb_init(void)
41 {
42   sb_state = SB_STATE_NORMAL;
43   sb_mduty = 0;
44   sb_t_fh = 0;
45   sb_t_fl = 0;
46   sb_t_ph = 0;
47   sb_t_pl = 0;
48   pp_param_read();
```

```
49    pp_sb_duty_set(0);
50    TMR1_Init();
51  }
52
53  /*-------------------------------------------------------
54   * モーターPWMデューティ比設定
55   *-------------------------------------------------------*/
56  void pp_sb_duty_set(uint16_t mduty)
57  {
58    /* 割り込み禁止/許可状態を保存 */
59    int8_t ogie  = GIE;
60    int8_t opeie = PEIE;
61    /* 割り込み禁止 */
62    /*   sb_t_phとsb_t_pl及びsb_t_fhとsb_t_flそれぞれを
63     *   アトミックに更新するため */
64    GIE  = 0;
65    PEIE = 0;
66    /* モーターPWMデューティ値(通常出力値) */
67    sb_mduty = mduty;
68    /* 低速ブーストパルス幅の計算:
69     * - TMR1カウント値 pw=0x0000 のとき
70     *     TMR1割込間隔: 0.00 [mSec]
71     *     = Prescale * 0x0000 / (FOCS/4) [Sec]
72     *     = 8 * 0x0000 / (32*10^6 / 4) [Sec]
73     *     = 0.00 * 10^{-3} [Sec]
74     * - TMR1カウント値 pw=0x2000 のとき
75     *     TMR1割込間隔: 8.192 [mSec]
76     *     = Prescale * 0x2000 / (FOCS/4) [Sec]
77     *     = 8 * 0x2000 / (32*10^6 / 4) [Sec]
78     *     = 8.192 * 10^{-3} [Sec]
79     */
80    /* p_sbpwdtの値[0..0x0400) を pw:[0..0x2000) へスケール */
81    uint16_t pw = p_sbpwdt << 3;
82    if (pw < 1) {
83      pw = 1;
84    }
85    uint16_t sb_t_p = 0x0000 - pw;
86    sb_t_ph = (uint8_t)(sb_t_p >> 8);
87    sb_t_pl = (uint8_t)(sb_t_p & 0xFF);
88    /* 低速ブースト周波数の計算:
89     * - TMR1カウント数 cy=0x1000 のとき
90     *     TMR1割込間隔: 4.096 [mSec]
91     *     = Prescale * 0x1000 / (FOCS/4) [Sec]
92     *     = 8 * 0x1000 / (32*10^6 / 4) [Sec]
93     *     = 4.096 * 10^{-3} [Sec]
94     * - TMR1カウント値 cy=0x8000 のとき
95     *     TMR1割込間隔: 32.768 [mSec]
96     *     = Prescale * 0x8000 / (FOCS/4) [Sec]
97     *     = 8 * 0x8000 / (32*10^6 / 4) [Sec]
98     *     = 32.768 * 10^{-3} [Sec]
99     */
100   /* p_sbfreqの値[0..0x0400) を cy:[0x4000..0x8000) へ変換 */
101   uint16_t cy = (SPEED_MAX - p_sbfreq) << 4; /* [0x0000..0x4000) */
102   cy = cy + 0x4000;                          /* [0x4000..0x8000) */
103   uint16_t sb_t_f = 0x0000 - cy;
104   sb_t_fh = (uint8_t)(sb_t_f >> 8);
105   sb_t_fl = (uint8_t)(sb_t_f & 0xFF);
106   /* 割り込み禁止/許可状態を復元 */
107   GIE  = ogie;
108   PEIE = opeie;
109 }
110
111 /*-------------------------------------------------------
112  * TMR1割込: 通常期間とブースト期間の交替
113  *-------------------------------------------------------*/
114 void TMR1_isr(void)
115 {
116   /* TMR1の動作を停止 */
117   T1CONbits.TMR1ON = 0;
118   /* モーターPWMデューティサイクル値を求める */
```

```
119   uint16_t md = 0;
120   if ((sb_t_ph == 0) && (sb_t_pl == 0)) { /* パルス幅 0 */
121     /* 常にブーストしない */
122     md = sb_mduty;
123   } else {
124     /* ブースト期間/通常期間の交替を繰り返す */
125     switch (sb_state) {
126     default:
127     case SB_STATE_NORMAL:          /* 通常期間の終了 */
128       sb_state = SB_STATE_SBOOST; /* ブースト期間の開始 */
129       TMR1H = sb_t_ph;
130       TMR1L = sb_t_pl;
131       if (sb_mduty == 0) {               /* 停車中 */
132         md = 0;
133       } else if (SW_SBEN() &&
134                  (sb_mduty < SB_SPEED_MAX)) { /* ブーストON */
135         md = SB_SPEED_MAX;
136       } else {                                /* ブーストOFF */
137         md = sb_mduty;
138       }
139       break;
140     case SB_STATE_SBOOST:          /* ブースト期間の終了 */
141       sb_state = SB_STATE_NORMAL; /* 通常期間の開始 */
142       TMR1H = sb_t_fh;
143       TMR1L = sb_t_fl;
144       md = sb_mduty;
145       break;
146     }
147   }
148   /* 常点灯処理 */
149   if (md < p_light) {
150     md = p_light;
151   }
152   /* モーターPWMデューティサイクル値を出力 */
153   MOTOR_PWM_OUT(md);
154   /* TMR1の動作を再開 */
155   T1CONbits.TMR1ON = 1;
156 }
157
158 /*-------------------------------------------*/
159 /* TMR1 */
160 void TMR1_Init(void)
161 {
162   T1CONbits.TMR1ON = 0; /* Disabled */
163   T1CONbits.CKPS   = 3; /* Prescale: 1:8  */
164   T1CLKbits.CS     = 1; /* FOSC/4 */
165   T1GCON = 0x00;
166   TMR1H = 0;
167   TMR1L = 0;
168   TMR1IF  = 0;
169   TMR1IE  = 1;
170   T1CONbits.TMR1ON = 1; /* Enabled */
171 }
```

課題

1. 本ファームウエアでは速度の自動加減速機能にブレゼンハムの線形増加アルゴリズムを使用しています. 自動加速にブレゼンハムの指数増加アルゴリズムを使用するようにファームウエアを変更して下さい.
2. 本ファームウエアでは低速ブースト期間の PWM デューティ比を 50% と固定にしています. これを可変抵抗器で調節できるようにして下さい. これには回路とファームウエアの両方に変更が必要になります.
3. 低速時のトルクの増強を行うために, 他のパワーパックや DCC デコーダではどのような方法を用いているかを調査して下さい.

第 11 章

ワンハンドルマスコン型
パワーパック

本章では自動加減速・低速ブースト機能を有するワンハンドルマスコン型のパワーパックを紹介します. ハードウエアは第 9 章で紹介したものをそのまま使用して, ワンハンドルマスコンを新たに製作して追加します. 図 11.3 がワンハンドルマスコンの製作例です. ファームウエアは第 10 章で紹介したものの一部を差し替えてワンハンドルマスコン対応にします. 本ファームウエアの特徴は第 10 章で紹介したものすべてを引き継いでおり, 走行速度の制御がワンハンドルマスコンによる操作となった点が違いです.

本章で紹介する回路とファームウエアの特徴は以下の通りです.

- ワンハンドルマスコンによる操作
  ワンハンドルマスコンの仕様は以下に示す全 6 段です (図 11.1).
  - 制動 3 段 — 制動 1(弱), 制動 2(中) , 制動 3(強)
  - 力行 2 段 — 力行 1(弱), 力行 2(強)
  - 惰行 (ニュートラル)
- 第 10 章で紹介した自動加減速・低速ブースト機能型パワーパックが有する走行中逆転防止機能, 自動加減速機能, 低速ブースト機能, ラッシュ防止機能, 走行中逆転防止機能, 緊急停止機能, 最高速度設定機能, 常点灯機能, などの機能を引き継いでいます.



図 11.1: マスコンでの力行・惰行・制動による速度の制御

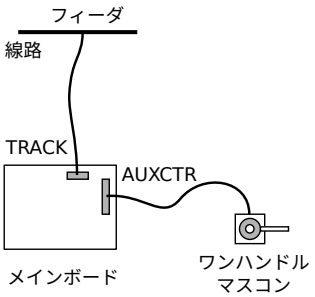


図 11.2: 接続方法

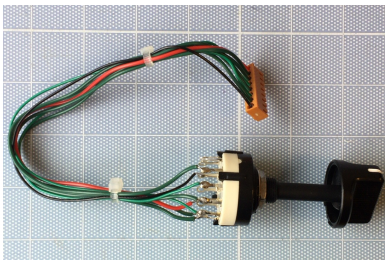


図 11.3: ワンハンドルマスコンの
コントローラ部の製作例

```
1  void main(void)
2  {
3    for(;;) {
4      ロータリースイッチを読む;
5      speed_target = 目標速度を決定;
6        (力行: 最大速度, 惰行: 現在速度のまま; 制動: 速度0)
7      speed = ブレゼンハムアルゴリズムで次の現在速度を決定;
8      md = PWMデューティ比を計算;
9    }
10 }
11
12 タイマ割り込みハンドラ: (低速ブーストの実現)
13 {
14   PWMデューティ比をモータドライバに出力(低速ブーストの実現方法に従
       う);
15 }
```

図 11.4: ファームウエア構造 (概要)

本章で学ぶ内容は以下の通りです.

- ロータリスイッチの使用方法

本章のファームウエアの大まかな構造を図 11.4 に示します. 自動加減速の機能はブレゼンハムアルゴリズムで実現します. ロータリースイッチの位置により力行/惰行/制動の種別および加減速度を決定します. 第 10 章で紹介したスピードボリュームによる制御を行うパワーパックではスピードボリューム位置そのものを目標走行速度としていたのが大きな違いです.

11.1　使用方法

本装置の使用方法を以下に説明します. 部品名称は第 9 章の図 9.4 (161 ページ) で示した通りで, 接続は図 11.2 を参考にして下さい.

操作方法は以下の通りです.

1. 電源スイッチ PWR-SW はオフにしておきます.
2. マスコンは制動または惰行の位置にします.
3. 電源スイッチ PWR-SW を入れます. するとパイロットランプ PWR-ON が点灯します. もしこのときにマスコンが制動または惰行の位置になければラッシュ防止機能が作動します.
4. 加減速度や低速ブーストなどの設定や緊急停止機能などは第 10 章で説明した通りです.
5. マスコンを操作して制動, 力行, 惰行を切替えながら模型車両の走行を制御します.

11.2　回路図

図 11.5 にロータリスイッチを使用したワンハンドルマスコンの回路図を示し, 部品表を表 11.1 に, それぞれ示します. マスコンは全 6 段なので 1 回路 6 接点あるいはそれ以上のロータリスイッチを使用します. またロータリスイッチの軸の回転はいくらでも回転しつづけることがないように, つまり制動 3 を越えて力行 2 へ行くとか, 力行 2 を越えて制動 3 へ行くなど 1 回転しないように, 回転防止ストッパを入れられるものを使用して下さい.

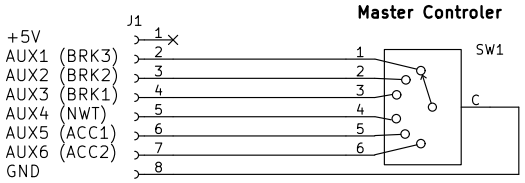


図 11.5: 【回路図】 ワンハンドルマスコン

表 11.1: 【部品表】 ワンハンドルマスコン

記号	数量	部品名
SW1	1	ロータリースイッチ 1 回路 6 接点　[P-00101]
		2 回路 6 接点や 1 回路 12 接点等でも可. 必要に応じ回転防止ストッパを取付.
J1	1	コネクタハウジング 8 ピン　JST XH コネクタ XHP-8 [C-12261]
	8	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
		ほか基板・配線材等

11.3　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します. 新たなソースファイルは 3 つだけで, ほかは第 10 章で紹介したものをそのまま使用します.

表 11.2: 【ソースファイル一覧】 パワーパック: ワンハンドルマスコン型

初	頁	番号	ファイル名	内容	行数
	189	10.2	`PowerPack-SlowBoost.c`	パワーパックメイン関数	51
	189	10.1	`pp.h`	パワーパック関数宣言: ヘッダ	46
	172	9.2	`pp-const.h`	パワーパック定数: ヘッダ	17
	194	10.5	`pp-control.c`	自動加減速パワーパック: 実装	103
✓	204	11.1	`pp-control-mc.c`	主幹制御器インターフェース: 実装	122
	191	10.3	`pp-misc.c`	パワーパック諸関数: 実装	111
	196	10.6	`pp-sb.h`	低速ブースト機能: ヘッダ	8
	198	10.7	`pp-sb.c`	低速ブースト機能: 実装	171
	129	6.1	`incr-lin.h`	線形増加アルゴリズム: ヘッダ	9
	130	6.2	`incr-lin.c`	線形増加アルゴリズム: 実装	78
	170	9.1	`io.h`	I/O ピン割付: ヘッダ	50
✓	206	11.2	`mcr.h`	MCR・周辺モジュール: ヘッダ	8
✓	206	11.3	`mcr.c`	MCR・周辺モジュール: 実装	84
	175	9.6	`mcr-motor.h`	モーター制御周辺モジュール: ヘッダ	25
	177	9.7	`mcr-motor.c`	モーター制御周辺モジュール: 実装	98
	117	5.5	`adc.h`	ADC モジュール: ヘッダ	9
	118	5.6	`adc.c`	ADC モジュール: 実装	76

(つづく)

(つづき)

初　頁	番号	ファイル名	内容	行数
			計 1066 行 (うち初出 214 行, 既出 852 行)	

ソースファイル pp-control-mc.c (主幹制御器インターフェース: 実装)

ロータリスイッチを使用した走行制御を実現します. これは第 10 章で紹介したボリューム制御インターフェース用のソースファイル pp-control-vr.c に対する置き換えです. 可変抵抗器を使用している pp-control-vr.c に含まれている関数と同一名称の関数を, ロータリスイッチを使用するものに作り変えます. このソースファイルの関数それぞれは以下の動作をします.

- pp_home_position()
  ラッシュ防止機能を実現する関数です. マスコンが惰行 (ニュートラル) または制動のいずれかの位置になるまで待機し続けます.
- pp_controler_speed()
  目標走行速度を返します. 制動の場合はゼロ, 力行の場合は最大速度, 惰行の場合は現在速度そのままが, それぞれの場合での目標走行速度とします.
- pp_controler_accel()
  加速度と減速度を返します. マスコンの位置に基づき, 可変抵抗器 ACC と DEC による設定値から加速度や減速度を算出します.

ソースファイル番号 11.1: pp-control-mc.c

```
1  /*------------------------------------------------------
2   * 主幹制御器による走行制御
3   *------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "mcr-motor.h"
8  #include "adc.h"
9  #include "io.h"
10 #include "pp.h"
11 #include "pp-const.h"
12 #include "pp-sb.h"
13 #include "incr-lin.h"
14
15 /* 主幹制御器 */
16 enum mc_value_e {
17   MC_VALUE_NONE,
18   MC_VALUE_BRK3,   /* 制動3 (最強) */
19   MC_VALUE_BRK2,   /* 制動2 */
20   MC_VALUE_BRK1,   /* 制動1 */
21   MC_VALUE_NWT,    /* 緩メ */
22   MC_VALUE_ACC1,   /* 力行  */
23   MC_VALUE_ACC2,   /* 力行 (最強) */
24 };
25
26 /*------------------------------------------------------
27  * 主幹制御器の位置
28  *------------------------------------------------------*/
29 static enum mc_value_e read_mc(void)
30 {
31   enum mc_value_e mcv = MC_VALUE_NONE;
32   if (SW_MC_BRK3()) {
33     mcv = MC_VALUE_BRK3;
```

```
  } else if (SW_MC_BRK2()) {
    mcv = MC_VALUE_BRK2;
  } else if (SW_MC_BRK1()) {
    mcv = MC_VALUE_BRK1;
  } else if (SW_MC_NWT()) {
    mcv = MC_VALUE_NWT;
  } else if (SW_MC_ACC1()) {
    mcv = MC_VALUE_ACC1;
  } else if (SW_MC_ACC2()) {
    mcv = MC_VALUE_ACC2;
  }
  return mcv;
}

/*-------------------------------------------------------
 * 主幹制御器の定位置への復帰を待機（緩メ/制動）
 *-------------------------------------------------------*/
void pp_home_position(void)
{
  uint16_t t = 0;
  uint8_t noks = 0;
  while (noks != 100) { /* 復帰状態が十分な回数連続 */
    __delay_ms(1);
    CLRWDT();
    /* 未復帰: インジケータLEDを点滅 */
    if (t != 0) {
      t--;
    } else {
      t = T_LED_WARNING;
      LED_INDICATOR_TOGGLE();
    }
    /* 緩メ/制動への復帰の検査（復帰回数を勘定） */
    switch (read_mc()) {
    case MC_VALUE_BRK3: noks++;   break;
    case MC_VALUE_BRK2: noks++;   break;
    case MC_VALUE_BRK1: noks++;   break;
    case MC_VALUE_NWT:  noks++;   break;
    case MC_VALUE_ACC1: noks = 0; break;
    case MC_VALUE_ACC2: noks = 0; break;
    default:
    case MC_VALUE_NONE: noks = 0; break;
    }
    /* モータードライバ LSI TB6643KQ 制御（過電流検出の解除） */
    if (SW_EMG_STOP()) { /* 解除要求の発生 */
      extern void MotorDriver_TB6643KQ_Reset(void);
      MotorDriver_TB6643KQ_Reset();
    }
  }
  LED_INDICATOR_OFF();
}

/*-------------------------------------------------------
 * 目標走行速度を求める
 *-------------------------------------------------------*/
uint16_t pp_controler_speed(uint16_t speed)
{
  uint16_t s = 0;
  switch (read_mc()) {
  case MC_VALUE_BRK3: s = 0;         break;
  case MC_VALUE_BRK2: s = 0;         break;
  case MC_VALUE_BRK1: s = 0;         break;
  case MC_VALUE_NWT:  s = speed;     break;
  case MC_VALUE_ACC1: s = SPEED_MAX; break;
  case MC_VALUE_ACC2: s = SPEED_MAX; break;
  default:
  case MC_VALUE_NONE: s = speed;     break;
  }
  return s;
}

/*-------------------------------------------------------
```

```
105   * 加速度&減速度を求める
106   *-------------------------------------------------------*/
107  uint16_t pp_controler_accel(uint16_t target_speed,
108                              uint16_t speed)
109  {
110    uint16_t a = 0;
111    switch (read_mc()) {
112    case MC_VALUE_BRK3: a =  p_decel;       break;
113    case MC_VALUE_BRK2: a = (p_decel << 1); break;
114    case MC_VALUE_BRK1: a = (p_decel << 2); break;
115    case MC_VALUE_NWT:  a =  p_accel;       break;
116    case MC_VALUE_ACC1: a = (p_accel << 2); break;
117    case MC_VALUE_ACC2: a = (p_accel << 1); break;
118    default:
119    case MC_VALUE_NONE: a = 4096; break; /* 大きな値(緩やかに加減速) */
120    }
121    return a;
122  }
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 11.2: mcr.h

```
1  #if !defined(__MCR_H__)
2  #define __MCR_H__
3
4  #define _XTAL_FREQ 32000000 /* 32MHz */
5
6  extern void System_Init(void);
7
8  #endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います. 内容は第 9 章のものとほぼ同一ですが, マスコン接続のための I/O ピンの設定に関する部分が異なります.

ソースファイル番号 11.3: mcr.c

```
1  /*-------------------------------------------------------
2   * MCR設定&周辺モジュール (PIC16F18857)
3   *-------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "mcr-motor.h"
8  #include "adc.h"
9
10 /*-----------------------------------------------*/
11 /* MCR初期化 */
12 void System_Init(void)
13 {
14   /* 発振器 (FOSC=32MHz) */
15   OSCCON1 = 0x10; /* HFINTOSC with 2x PLL */
16   OSCCON3 = 0x00;
17   OSCEN   = 0x00;
18   OSCFRQ  = 0x05; /* 16MHz */
19   OSCTUNE = 0x00;
20   while (PLLR == 0) {
21     ;
22   }
23   /* I/Oピン */
24   LATA    = 0x00;
25   LATB    = 0x00;
26   LATC    = 0x00;
```

```
  ANSELA = 0xBF; /* ANA7,ANA5,ANA4,ANA3,ANA2,ANA1,(ANA0) */
  ANSELB = 0x00; /* - */
  ANSELC = 0x00; /* - */
  TRISA  = 0xFF; /* RA7,RA6,RA5,RA4,RA3,RA2,RA1,(RA0) */
  TRISB  = 0xFF; /* (RB7,RB6),RB5,RB4,RB3,RB2,RB1,RB0 */
  TRISC  = 0x87; /* (RC7),RC2,RC1,RC0 */
  WPUA   = 0x01; /* RA0 */
  WPUB   = 0xFF; /* RB7,RB6,RB5,RB4,RB3,RB2,RB1,RB0 */
  WPUC   = 0x80; /* RC7 */
  WPUE   = 0x00;
  ODCONA = 0x00;
  ODCONB = 0x00;
  ODCONC = 0x00;
  int8_t ogie = GIE;
  GIE = 0;
  PPSLOCK = 0x55;
  PPSLOCK = 0xAA;
  PPSLOCKbits.PPSLOCKED = 0;
  RC5PPS = 0x05;   /* RC5 <- CWG1A */
  RC6PPS = 0x06;   /* RC6 <- CWG1B */
  PPSLOCK = 0x55;
  PPSLOCK = 0xAA;
  PPSLOCKbits.PPSLOCKED = 1;
  GIE = ogie;
  /* 周辺モジュール */
  ADC_Init();
  MCR_Motor_Init();
}

/*-------------------------------------------*/
/* CONFIG1 */
#pragma config FEXTOSC  = OFF
#pragma config RSTOSC   = HFINTPLL
#pragma config CLKOUTEN = OFF
#pragma config CSWEN    = ON
#pragma config FCMEN    = ON
/* CONFIG2 */
#pragma config MCLRE    = OFF /* MCLRをI/Oピン */
#pragma config PWRTE    = OFF
#pragma config LPBOREN  = OFF
#pragma config BOREN    = ON
#pragma config BORV     = LO
#pragma config ZCD      = OFF
#pragma config PPS1WAY  = ON
#pragma config STVREN   = ON
#pragma config DEBUG    = OFF
/* CONFIG3 */
#pragma config WDTCPS   = WDTCPS_31
#pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
#pragma config WDTCWS   = WDTCWS_7
#pragma config WDTCCS   = SC
/* CONFIG4 */
#pragma config WRT      = OFF
#pragma config SCANE    = available
#pragma config LVP      = OFF /* 低電圧書込不可 */
/* CONFIG5 */
#pragma config CP       = OFF
#pragma config CPD      = OFF
```

課題

1. 全 12 段のワンハンドル型のマスコンを持つパワーパックを設計・製作して下さい.
2. ツーハンドル型のマスコンを持つパワーパックを設計・製作して下さい.
3. 実車ではノッチによって最高速度が異なります. これを実現するように本章のファームウエアを変更して下さい.
4. 本ファームウエアでは速度の自動加減速機能にブレゼンハムの線形増加アルゴリズムを使用しています. 上の問題に対するファームウエア変更から更に, 自動加速にブレゼンハムの指数増加アルゴリズムを使

用するようにファームウエアを変更して下さい.

5. 上問では改造例をいくつか挙げました. しかしそれぞれの場合に対して個別にハードウエアとファームウエアを開発するのは動作テストや保守が面倒です. そこでワンハンドル・ツーハンドルどちらでも選択でき, またマスコンの段数を任意に選択できる, 汎用的なパワーパックを設計・製作して下さい.

第 III 部

ポイントの制御

第 12 章

ソレノイド方式ポイントマシン用
マイコン制御ポイント切替器

組み立ての際に配線を間違えたりソースコードの入力を間違えると, ポイントマシンを焼損あるいは発火するなどの重大な事故になります. 十分に注意して取りかかるようにして下さい.

本章では 2 線ソレノイド方式のポイントマシンを制御する装置を紹介します. この方式のポイントマシンの例として KATO ユニトラックや TOMIX ファイントラックなどでの電動ポイントなどが挙げられます. PECO PL-10 Turnour Motor は 3 線ソレノイド方式のポイントマシンですが, 後で紹介するアダプタを用いることで制御が可能になります. 本章で紹介する回路とファームウエアの特徴は以下の通りです.

- 保護回路内蔵のモータドライバ IC を使用
  短絡防止機能・加熱防止機能などが内蔵されており, 手軽で安価に安全性の高い回路が実現できます.
- 電子制御が可能
  機械的なスイッチは使用せず電子信号で制御しますので, 他装置との連携動作が容易です. 第 29 章と第 30 章で紹介する鉄道模型の運転自動システムで使用します.
- ポイントマシンへの通電時間の厳密な制御による安全性
  手動のスイッチによる通電では通電時間の超過によるコイル損傷の心配は常につきまといます. 本ファームウエアでは通電時間をソフトウエアで制御するのでその心配はありません. ただしソフトウエアのバグへの十分な対策は必須であり, これが本章のテーマです.

本章で学ぶ内容は以下の通りです.

- ウオッチドッグタイマによる高信頼化の手法

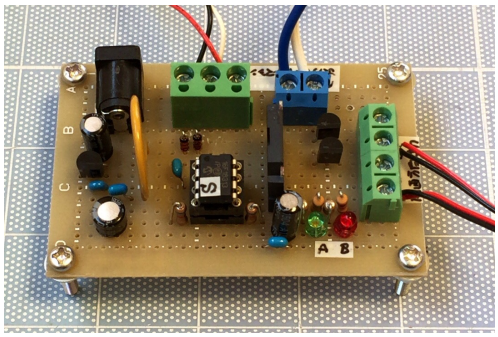


図 12.1: ポイント切替器の試作例

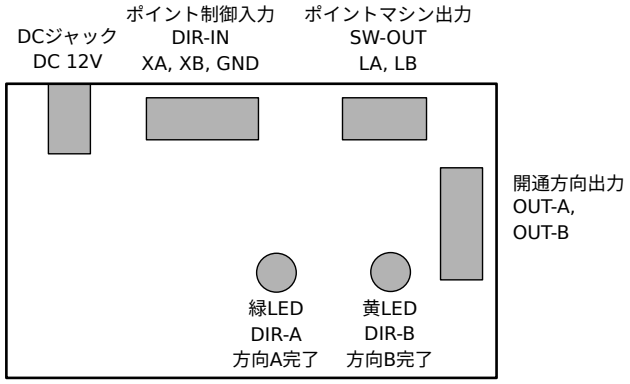


図 12.2: 操作表示部品の配置例と名称

```
1  main()
2  {
3    for (;;) {
4      if（ポイント制御入力に変化が発生）{
5        if（変化の発生は頻繁ではない）{
6          入力に従った開通方向へモータドライバをオンにする;
7          50ミリ秒の経過を待つ;
8          モータドライバをオフにする;
9        }
10     }
11   }
12 }
```

図 12.3: ファームウエア構造 (概要)

- ウオッチドッグタイマモジュール WDT の使用方法
- ソレノイド式ポイントマシンの制御方法

本章のポイント切替器の特徴はポイントの切り替えを電子制御で行える点にあり, 他装置から簡単に利用できます. 第 29 章と第 30 章で紹介する鉄道模型の運転自動システムでも使用します. 伝統的なポイント切り替えの手法は一瞬だけ電流が流れるように機構を工夫した機械的スイッチや, 大容量コンデンサと機械的スイッチの組み合わせなどであり, 電子制御は行えませんでした. 本装置のマイコンは, 外部から電子信号を受けるとモータドライバを駆動し, ソレノイドに電流を流してポイントの切り替えを行います.

駆動対象はソレノイド方式のポイントマシンですので電流を流す時間はごく短くしなければなりません. さもなくばポイントマシンのコイルを焼損しますから, ファームウエアはこの点に十分注意を払う必要があります. これが本章で取り組むべきソフトウエア技法です. 具体的には以下の 2 点によりポイントマシンの保護を行います.

- 長時間通電バグへの対処: マイコン内蔵のウォッチドッグタイマ機能の利用
  ファームウエアのバグにより, ポイントマシンへの通電中に無限ループに入る, あるいは通電停止ができていない, という状況への対処です. マイコン内部のハードウエアのウォッチドッグタイマ機能により, 一定時間が経過すれば強制的にマイコンをリセットします. 万が一, 長時間通電が生じても強制的に通電を停止して, ポイントマシンを保護します.
- 連続した切り替え操作への対処: ファームウエアによる強制冷却期間の導入
  外部からの切り替え操作がきわめて頻繁に行なわれ続ける, という状況への対処です. ファームウエアにバグがなくても, この場合にはポイントマシンの加熱が心配です. 手作業で切り替えする場合にはあまり考えられませんが, 電子制御で切り替えを行う場合には考えなくてはなりません. なぜなら, 切り替え操作を行う外部のハードウエア・ソフトウエアのバグにより, 切り替え操作が頻繁に生じ続ける可能性があるからです. このような不適切な外部からの入力があればそれを検出して, しばらくは切り替えを停止する仕組みをファームウエアに作ります. つまり連続した切り替えが生じればポイントの切り替えは一時禁止にして, 加熱しかけたポイントマシンを強制的に冷却する期間を設けます.

十分に注意をしてバグがないプログラムを書くのは当然なのですが, しかしそれでもバグは生じてしまうものです. ですから注意力だけに頼ることはせずに, 不注意な場合でも事故が起きない対策をあらかじめ入念に用

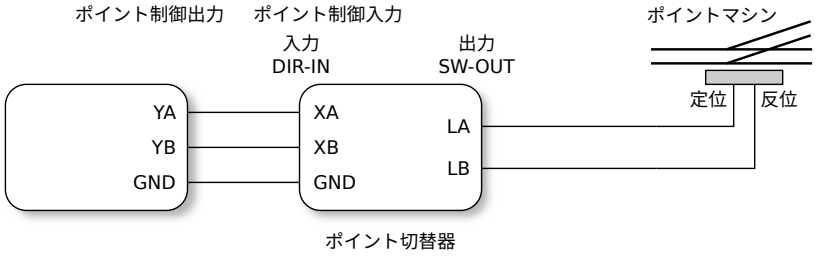


図 12.4: ポイントマシンとの接続

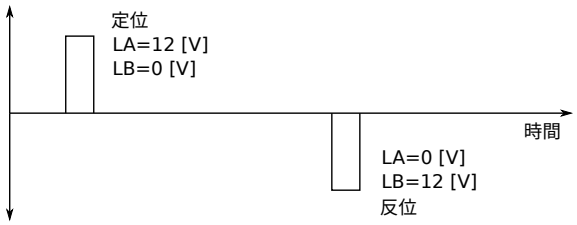


図 12.5: 出力波形の模式図

意することが重要です. 以下のように何段階もの保護手段が入っています.

- 過電流保護機能を内蔵した AC アダプタの使用
- 過電流保護機能を内蔵したモータドライバの使用
- ポリスイッチによる保護
- ファームウエアによる強制冷却期間
- マイコン内蔵のウォッチドッグタイマ機能

なお本章の回路は, 第 13 章で紹介するモータ方式用のポイント切替器の回路と共通です. 同じハードウエアに対して異なるファームウエアを開発することで, ハードウエアの開発の手間を減らしつつも異なる種類のポイントマシンへの対応を実現します.

この装置のファームウエアの大まかな構造を図 12.3 に示します. 入力の変化が発生すればモータドライバを制御して, ポイントマシンのコイルへパルス電流を流すという, 極めて単純なものです. なおこのパルス幅の実際の値としては 50 ミリ秒間に設定しています.

12.1　使用方法

本装置の使用方法を説明します. 以下の説明で使用する部品名称を図 12.2 に配置例とともに示しています. 基本的な接続を図 12.4 に示し, 手順は以下の通りです.

- DC ジャックに出力電圧が 12V の AC アダプタを接続します. ポイントマシンによってはもう少し高い電圧にしないと動作しない場合もあるかもしれません. 特に PECO PL-10 は 12V では動作しないので, 電圧が 16V から 20V 程度の AC アダプタを使用します.
- ターミナル DIR-IN がポイント制御入力です. これにスイッチを接続します.
- ターミナル SW-OUT の LA と LB をポイントマシンに接続します. 通常のポイントマシンではポイントの直通側を定位とし分岐側を反位としていますので, 本章の回路もそれを想定しています. 代表的なポイントマシンの配線の極性を表 12.1 に示していますので参考にして下さい.

表 12.1: 代表的なポイントマシンへの接続

ポイントマシン	電線との接続		備考
	LA	LB	
KATO ユニトラック	黒	赤	黒プラス, 赤マイナスで直通 黒マイナス, 赤プラスで分岐
TOMIX ファイントラック	茶	白	茶プラス, 白マイナスで直通 茶マイナス, 白プラスで分岐

表 12.2: ポイント切替器の仕様

入力		出力		機能
XA	XB	LA	LB	
L	HiZ	+	−	定位に開通
HiZ	L	−	+	反位に開通
HiZ	HiZ			開通方向を保持

しかし安全側線のポイントのように分岐側を定位とする場合もあります. そのように変更する場合には, 出力 LA と LB を逆に配線することで物理的な切り替え方向を逆にして下さい. なお入力 DIR-IN は論理的な観点で入力を決めていますので, ここを逆に配線すると論理が逆になってしまいます. 論理と物理の概念は区別します.

- ターミナル OUT-A と OUT-B は開通方向を示すアクセサリ制御出力で詳しくは後述しますが, 使用しなくても構いません. 開通方向は LED の点灯により表示されます.

ポイント切替の仕様を表 12.2 に示します. 以降ではポイントの開通方向を指示する入力を**ポイント制御入力**と呼ぶことにします. この入力は機械的なスイッチだけでなく電子制御が可能です.

トグルスイッチによるポイント切り替え操作

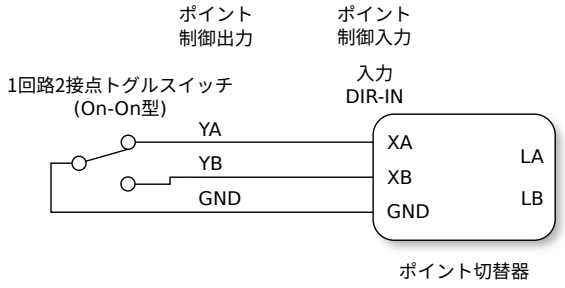
ポイント制御入力へは図 12.6a に示すように 1 回路 2 接点 On-On 型のトグルスイッチを接続します. このスイッチの回路図を図 12.6b に示します. ポイント切り替えの操作は以下のように行います.

- YA を GND に短絡
  ポイントは定位に開通します. このときポイント制御入力では XA=L, XB=HiZ です.
- YB を GND に短絡
  ポイントは反位に開通します. このときポイント制御入力では XA=HiZ, XB=L です.

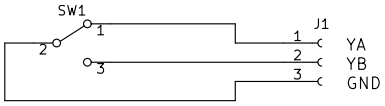
タッチ式のポイント切り替え操作

本装置では昔ながらのタッチ式の切り替え操作も可能です. 図 12.7 に示す通り XA と XB それぞれを端子に配線して, 別途用意したテスタ棒を GND に接続します.

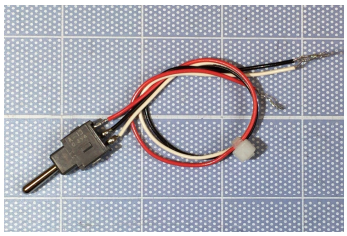
- YA 端子にテスタ棒をタッチする
  XA=L, XB=HiZ となりますから, ポイントは定位に開通します.
- YB 端子にテスタ棒を触れる



(a) 接続図



(b) 回路図



(c) 試作例

図 12.6: トグルスイッチによるポイント切り替え

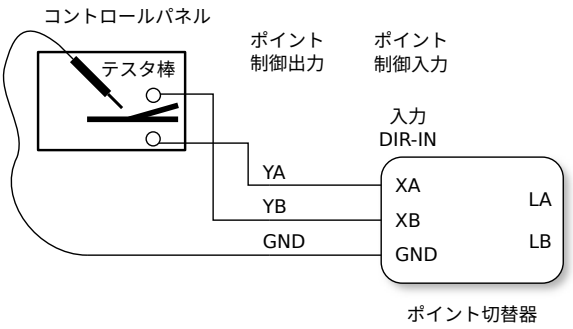


図 12.7: テスタ棒によるタッチ式のポイント切り替え

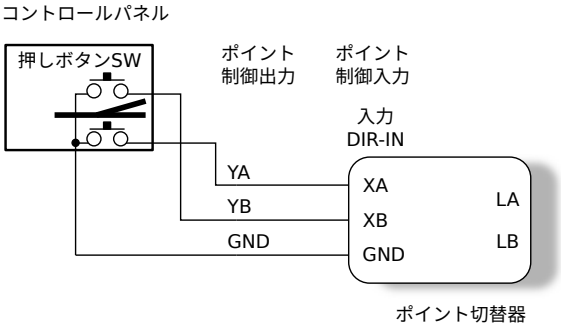


図 12.8: 押しボタンスイッチによるポイント切り替え

YB=L, YA=HiZ となりますから, ポイントは反位に開通します.

テスタ棒を端子から離しても切り替わった開通方向が保持されるのは, と XB がともに HiZ の場合は現在の開通方向を保持するという仕様にしているためです.

押ボタンスイッチ式のポイント切り替え操作

2 つのモーメンタリ型の押しボタンスイッチを用いて, 図 12.8 に示す形での切り替え操作も可能です. 以上で紹介したトグルスイッチ, タッチ式, 押しボタンスイッチのいずれも, 外見や操作は異なりますが電気的には同一です.

12.2　回路図

ポイント切替器

ポイント切替器の回路図を図 12.9 に, 部品リストを表 12.3 に, それぞれ示します. 図 12.5 に示す波形の出力が本装置の目標です.

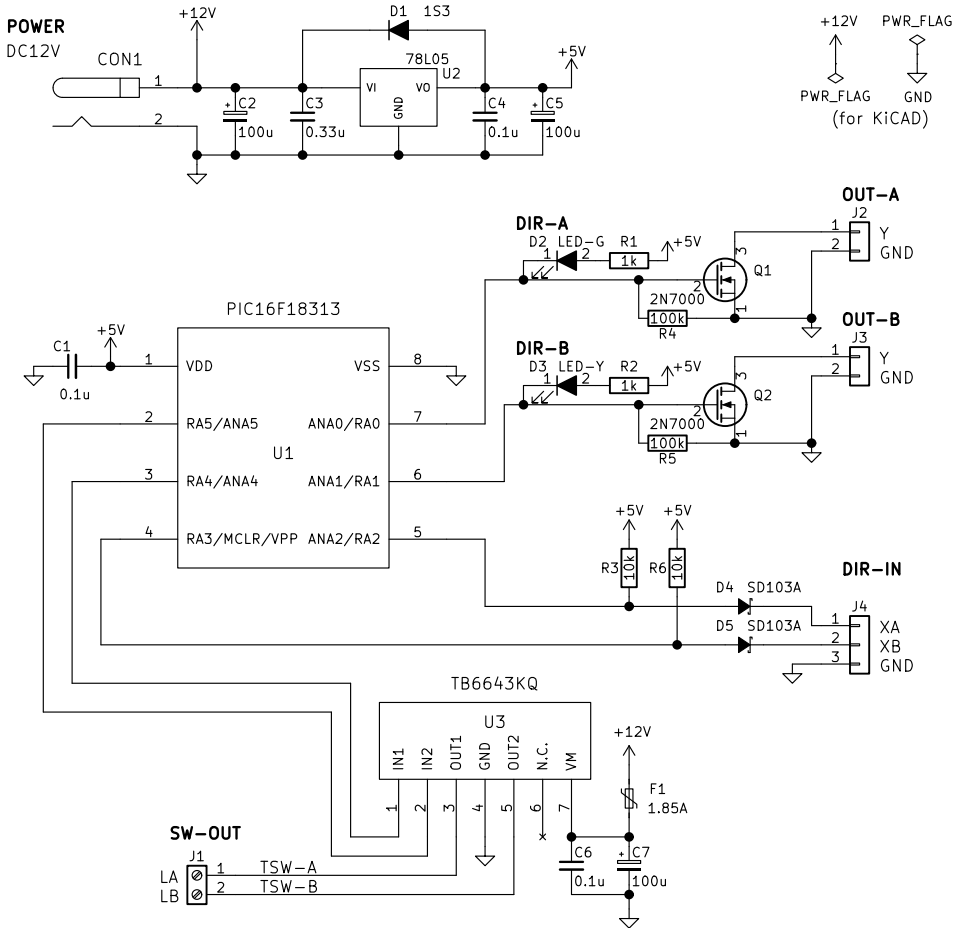


図 12.9: 【回路図】 ポイント切替器 (ソレノイド方式/モータ方式共通)

表 12.3: 【部品表】 ポイント切替器

記号	数量	部品名
U1	1	PIC 16F18313-I/P [I-10889]
U2	1	3 端子レギュレータ L78L05ACZ (又は同等品) [I-01250]
U3	1	モータドライバ TB6643KQ [I-07688]
Q1, Q2	2	FET 2N7000 [I-09723]
D1	1	ダイオード 1S3 30V 1A [I-01707]
D4, D5	2	ショットキーバリアダイオード SD103A [I-04271]
D2	1	LED 緑 [I-06413]
D3	1	LED 黄 [I-11639]
F1	1	ポリスイッチ MF-R185 1.85A [P-12629]
C2, C7	2	電解コンデンサ 100μF 耐圧 25V 以上 [P-07760]
C5	1	電解コンデンサ 100μF 耐圧 16V 以上 [P-08290]
C1, C4, C6	3	セラミックコンデンサ 0.1μF [P-00090]

(つづく)

表 12.4: 開通状態出力の仕様

Y	意味
HiZ	開通
L	非開通

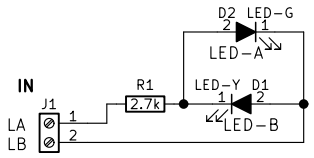
(つづき)

記号	数量	部品名
C3	1	セラミックコンデンサ 0.33μF [P-08147]
R1, R2	2	抵抗 1kΩ　1/4W [R-25102]
R3, R6	2	抵抗 10kΩ　1/4W [R-25103]
R4, R5	2	抵抗 100kΩ　1/4W [R-25104]
	1	IC ソケット 8 ピン　[P-00035]
J1	1	ターミナルブロック 2 ピン　[P-01306]
J2, J3	2	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
J4	1	基板用コネクタ 3 ピン　JST XH コネクタ B3B-XH-A(LF) [C-12248]
	6	コネクタハウジング 2 ピン　JST XH コネクタ XHP-2 [C-12255]
	2	コネクタハウジング 3 ピン　JST XH コネクタ XHP-3 [C-12256]
	18	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
CON1	1	DC ジャック　[C-09408]
	1	AC アダプタ 12V　出力 3A [M-10661]
		ほか基板・配線材等

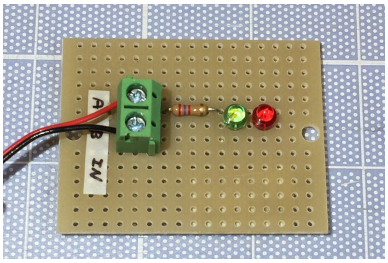
ポイントマシンへの出力はモータドライバ TB6643KQ を用います. モータドライバ TB6643KQ は H ブリッジ式モータドライバですので, その 2 つの出力 OUT1 と OUT2 の極性の反転は簡単に行えます. 2 線ソレノイド方式ポイントマシンは 2 線の極性に従って開通方向が切り替わりますから, H ブリッジ式モータドライバは大変都合よい部品です.

回路には 2 つ開通状態出力 OUT-A と OUT-B が備えられています. 開通状態出力とはポイントの開通方向を知らせるためのもので, 他の装置と連携できるようアクセサリ制御入出力の仕様に従っています. OUT-A はポイント定位, OUT-B はポイント反位の開通状態の出力で, これら開通方向出力の仕様を表 12.4 に示します.

開通状態出力の仕様において非開通の場合を L としている理由を説明します. 第 18 章で紹介する信号機ではこの開通状態出力を利用して, ポイントが非開通の場合には出発信号機を停止現示とする形で利用しています. 例えば出発信号機においては, 駅から本線に出るまでに設置されているポイントがひとつでも開通していない場合には出発信号機は赤を現示して列車の出発を禁止します. この「ひとつでも」とは論理和演算にほかなりません. ポイント非開通は危険な状態であり, システムにひとつでも危険な状態があれば安全のために停止が必要です. すると「危険な状態」をワイヤード OR で検出するには, 危険な状態を信号レベル L で表現すれば良いことになります. なぜならワイヤード OR は入力がひとつでも L ならば出力を L にするからです.

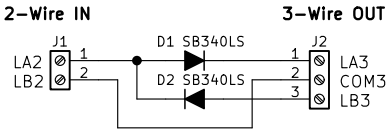


(a) 回路図

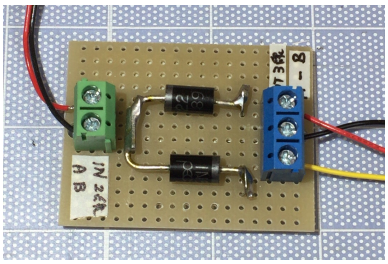


(b) 試作例

図 12.10: 動作確認用テスタ (2 線ソレノイド方式用/モータ方式用共通)



(a) 回路図



(b) 試作例

図 12.11: 3 線ソレノイド方式用ポイントマシン用アダプタ

動作確認用テスタ (2 線ソレノイド方式用)

本装置を作成するに当たっては, ここで紹介する動作確認用テスタを作成して必ず十分に動作テストを行ったうえでポイントマシンに接続して下さい.

ポイント切替器のファームウエアにもしバグがあればポイントマシンのコイルは焼損してしまいますから, ポイント切替器は危険極まりない装置です. そのため動作の確認は念入りに行わなければなりません.

動作確認用テスタの回路図を図 12.10a に, 部品リストを表 12.5 に, それぞれ示します. これはモータドライバの出力を LED で点灯するだけの簡単な回路です. LED の点灯は一瞬だけです. なぜならソレノイド方式のポイントマシンの駆動では一瞬しか電流を流さないからです. もし LED が点灯し続けたとすれば回路かファームウエアに間違いがあります. もしこれがポイントマシンに配線されていたら間違いなく焼損していたところですので, 落ち着いて回路やファームウエアの間違いを修正して下さい.

表 12.5: 【部品表】 動作確認用テスタ (2 線ソレノイド方式用/モータ方式用共通)

記号	数量	部品名
D1	1	LED 黄 [I-11639]
D2	1	LED 緑 [I-06413]
R1	1	抵抗 2.7kΩ 1/4W [R-25272]
J1	1	ターミナルブロック 2 ピン [P-01306]

12.3　3 線ソレノイド方式ポイントマシン用アダプタ

PECO PL-10 などの 3 線ソレノイド方式ポイントマシンのためのアダプタを紹介します. このアダプタにより, 図 12.12 に示す波形の電圧が出力されます. なおこの回路はずいぶん昔から知られているもののようで, 調べても初出はどこかは分かりませんでした. 回路図を図 12.11a に, 部品リストを表 12.6 に, それぞれ示します.

この回路の原理を概説します. 2 線式のポイントマシンの制御は, 2 線の極性によってソレノイドの駆動方向を決めるのでした (55 ページ図 2.11a). 3 線式のポイントマシンの制御は 1 本を共通線として, 残りの 2 本のど

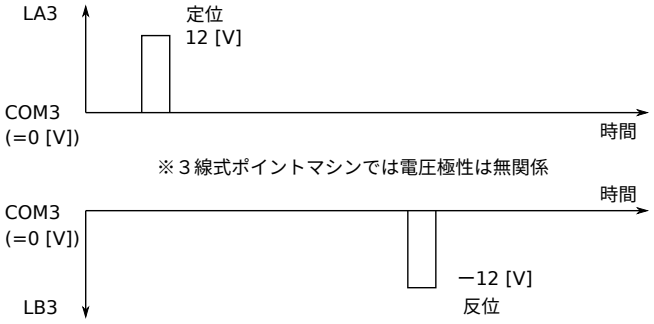
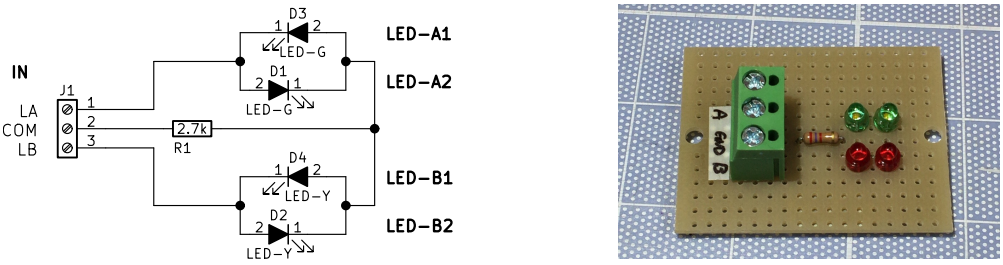


図 12.12: 出力波形の模式図 (3 線式アダプタの出力)



(a) 回路図　　**(b)** 試作例

図 12.13: 3 線ソレノイド方式用動作確認用テスタ

- 定位へ切り替えの場合
  電流は LA2, ダイオード D1, LA3, ソレノイド, COM3, LB2 と流れます.
- 反位へ切り替えの場合
  電流は LB2, COM3, ソレノイド, LB3, ダイオード D2, LA2 と流れます.

表 12.6: 【部品表】 3 線ソレノイド方式ポイントマシン用アダプタ

記号	数量	部品名
D1, D2	2	ダイオード SB430LS　整流用 3A [I-07788]
J1	1	ターミナルブロック 2 ピン [P-01306]
J2	1	ターミナルブロック 3 ピン [P-01307]

最後に 3 線式ポイント用の動作確認用テスタを紹介します. 本装置を 3 線式ポイントで使用する場合でも, ここで紹介する動作確認用テスタを作成して必ず十分な動作テストを行って下さい. 動作確認用テスタの回路図を図 12.13a に, 部品リストを表 12.7 に, それぞれ示します.

表 12.7: 【部品表】 3 線ソレノイド方式用動作確認用テスタ

記号	数量	部品名
D1, D3	2	LED 緑 [I-06413]
D2, D4	2	LED 黄 [I-11639]
R1	1	抵抗 2.7kΩ　1/4W [R-25272]
J1	1	ターミナルブロック 3 ピン [P-01307]

12.4　電源投入時の動作

表 12.8: 本書のポイント切替器の電源投入時の動作 (“–” は非動作を表す)

装置	ポイント制御入力 XA, XB の初期値			
	HiZ, HiZ	L, HiZ	HiZ, L	L, L
第 12 章: 2 線/3 線ソレノイド方式用 (モータドライバ IC 使用)	–	–	–	–
第 13 章: DC モータ方式用 (モータドライバ IC 使用)	–	–	–	–
第 14 章: 2 線ソレノイド方式用 (リレー&電解コンデンサ使用)	–	–	反位	反位
第 14 章: 3 線ソレノイド方式用 (リレー&電解コンデンサ使用)	–	–	反位	反位
第 14 章: DC モータ方式用 (リレー使用)	定位	定位	反位	反位
第 15 章: サーボモータ方式用	定位	定位	反位	反位

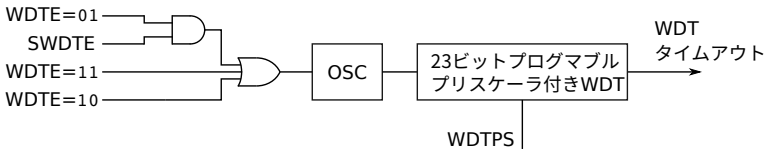


図 12.14: ウオッチドッグタイマモジュール WDT の内部構造の概略

す. もし電源投入時にポイントマシンが駆動されるとしたら, 多数のポイントマシンが一斉に動作して, レイアウト全体では大電流となってブレーカが落ちてしまいかねません. この事態を避ける簡単な方法が電源投入時にポイントマシンを駆動しないようにすることです. レイアウトに電源を投入した後にポイントの開通方向は様々ですから, 明示的にポイントを切り替えて開通方向を確実にして下さい. 切り替えを順次行うことにより, 一度に大電流が流れることが避けられます.

本書では他にも各種のポイント切替器を紹介しており, 電源投入時の動作を表 12.8 にまとめてあります. この表から分かるように, 装置が異なってもできるだけ同じ振る舞いをするように設計していますが, 回路上の都合でポイント切替器ごとに若干の仕様の違いがあります. そのため電源投入時のポイントの開通方向は不定だと考えるのが確実です. 電源を投入したときには明示的にポイントの切り替え操作を行い, 開通方向を確定して下さい. その手順は, まず反位に切り替え, 続いて定位に切り替えるというものです. するとどのポイント切替器でも開通方向は確実に定位になり, これ以降の操作では確実にポイント切替が行われます.

12.5　WDT: ウォッチドッグタイマモジュール

PIC マイコンでは内蔵周辺モジュール WDT (Watchdog Timer; ウオッチドッグタイマ) を使用して無限ループ等のバグからの脱出を行い, ファームウエアの信頼性の向上をハードウエア的に実現します. 図 12.14 にウオッチドッグタイマモジュール WDT の内部構造の概略を示します.

WDT の動作の概要を説明します. WDT はタイマを内部に装備しており, 一定時間ごとにタイマはカウントアップされます. カウントアップの速度はプリスケーラの設定により変更可能です. CLRWDT 命令を実行する

```
1  main()
2  {
3    WDTの初期設定;
4    for (;;) {
5      CLRWDT();
6      処理本体;
7    }
8  }
```

図 12.15: WDT を用いたファームウエアの構造

ことでタイマはクリアされますが, クリアされなければやがてタイマがタイムアウトすなわちオーバーフローが発生して, マイコンはリセットされて再起動されます.

WDT を使用したファームウエアの典型的な構造を図 12.15 に示します. バグによる無限ループ等の理由により, もし処理本体の実行を一定時間内に完了できなければマイコンはリセットされて再起動されます. WDT は予期しない動作からの自律的な回復手段として非常に有効です.

16F18313 の WDT で使用する主なレジスタ類を以下に概説します.

- WDTE
  コンフィグレーションビット CONFIG2 中の 2 ビットで, WDT の動作を設定します. コンフィグレーションビットビットですのでファームウエア実行中には変更はできません. 2 ビットの値の組み合わせにより 4 通りの設定があります.
  - WDTE=ON (値 3)
    WDT を有効にします. SWDTEN の値は無視します.
  - WDTE=SLEEP (値 2)
    WDT を有効にしますが, アイドル時とスリープ時には無効にします. SWDTEN の値は無視します.
  - WDTE=SWDTEN (値 1)
    WDT の有効/無効を SWDTEN で制御します. SWDTEN=1 の場合は WDT を有効にし, SWDTEN=0 の場合は無効にします.
  - WDTE=OFF (値 0)
    WDT を無効にします. SWDTEN の値は無視します.
- PCON0.RWDT
  リセット要因が WDT によるものか否かを示すフラグです. 0 の場合はリセット要因は WDT のタイムアウトによるもので, 1 の場合はそれ以外です.
- WDTCON.WDTPS
  WDT プリスケーラを設定します. これにより WDT のタイムアウト時間が決まります. 2 進数で 00000 の場合は約 1 ミリ秒, 00001 の場合は約 2 ミリ秒となり, 10010 の場合には最大の約 256 秒となります. タイムアウト時間は厳密である保証はありませんから, 実行タイミングの調整などには使えません.
- WDTCON.SWDTEN
  WDT の有効/無効を設定します. ただし効果があるのは WDTE=1 の場合に限られます. 1 の場合は WDT を有効にし, 0 の場合は無効にします.

WDT のタイムアウト時間の設定値はシステムの信頼性向上において非常に重要です. 本章のソレノイド方

式ポイントマシンへの通電の場合を例として, タイムアウト時間の設定値の良し悪しの判断基準を説明します. ソレノイド方式ポイントマシンの場合では, タイムアウト時間を 10 秒に設定したのではまったく役に立ちません. なぜなら, バグにより通電状態のままで無限ループに入って 10 秒間が経過すれば, その間にソレノイドは焼損してしまうからです. タイムアウト時間が 1 秒以下ならば焼損はたぶん避けられるでしょう. タイムアウト時間の設定は, 正常な場合には決してタイムアウトにならず, バグが発生した場合にはシステムが危険な状態になる前にタイムアウトになるようにします.

12.6　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 12.9: 【ソースファイル一覧】 ポイント切替器: ソレノイド方式用

初	頁	番号	ファイル名	内容	行数
✓	226	12.4	TurnoutCtr-Solenoid.c	主ソースコード	152
✓	222	12.2	turnout.h	諸定義: ヘッダ	24
✓	223	12.3	instate.c	ポイント制御入力と動作: 実装	91
✓	221	12.1	io.h	I/O ピン割付: ヘッダ	21
✓	228	12.5	mcr.h	MCR・周辺モジュール: ヘッダ	8
✓	229	12.6	mcr.c	MCR・周辺モジュール: 実装	51
				計 347 行 (うち初出 347 行, 既出 0 行)	

本ファームウエアは 3 状態の有限状態機械を基礎としており, その構造を詳しく示したのが図 12.16 です. この有限状態機械の状態遷移図を図 12.17 に示します.

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

ソースファイル番号 12.1: io.h

```
1  #if !defined(__IO_H__)
2  #define __IO_H__
3
4  /* ポイント開通方向A(定位)入力: RA2 (H:非開通, L:開通) */
5  #define DIR_IN_A()              (RA2 == 0)
6  /* ポイント開通方向B(反位)入力: RA3 (H:非開通, L:開通) */
7  #define DIR_IN_B()              (RA3 == 0)
8  /* モータードライバ制御(ポイントマシンコイル)出力: RA4, RA5 */
9  #define OUT_TURNOUT_A_ON()      (LATA4 = 1)
10 #define OUT_TURNOUT_A_OFF()     (LATA4 = 0)
11 #define OUT_TURNOUT_B_ON()      (LATA5 = 1)
12 #define OUT_TURNOUT_B_OFF()     (LATA5 = 0)
13 /* 開通方向出力: RA0, RA1 (FETを通じたオープンドレイン出力)
14  *  - 開通:   OD出力=Z ... ポートL出力 (FET OFF)
15  *  - 非開通: OD出力=L ... ポートH出力 (FET ON)   */
16 #define OUT_OD_DIR_A_Z()        (LATA0 = 0)
17 #define OUT_OD_DIR_A_LOW()      (LATA0 = 1)
18 #define OUT_OD_DIR_B_Z()        (LATA1 = 0)
19 #define OUT_OD_DIR_B_LOW()      (LATA1 = 1)
20
21 #endif /*__IO_H__*/
```

ソースファイル turnout.h (諸定義: ヘッダ)

このソースファイルで重要な定数の定義は以下のものです.

```
state = STATE_ILDE;
for (;;) {
  __delay_ms(T_CYCLE);
  switch (state) {
  case STATE_IDLE:       /* 待機期間 */
    ポイント制御入力から切替要求を検出;
    if (切替要求がある) {
      state = STATE_CURRENT_ON;
      ポイントマシンへ通電を開始;
    }
    break;
  case STATE_CURRENT_ON: /* 通電期間 */
    if (通電時間終了) {
      state = STATE_ILDE;
      ポイントマシンへの通電を終了;
      if (頻繁な通電が起きた) {
        state = STATE_COOL;
      }
    }
    break;
  case STATE_COOL:       /* 冷却期間 */
    if (冷却期間終了) {
      state = STATE_IDLE;
    }
    break;
  }
}
```

図 12.16: ファームウエア構造 (詳細)

- 実行 1 サイクルの時間 T_CYCLE
  実行の 1 サイクルを 10 ミリ秒としています.

ソースファイル番号 12.2: turnout.h

```
#if !defined(__TURNOUT_H__)
#define __TURNOUT_H__

```

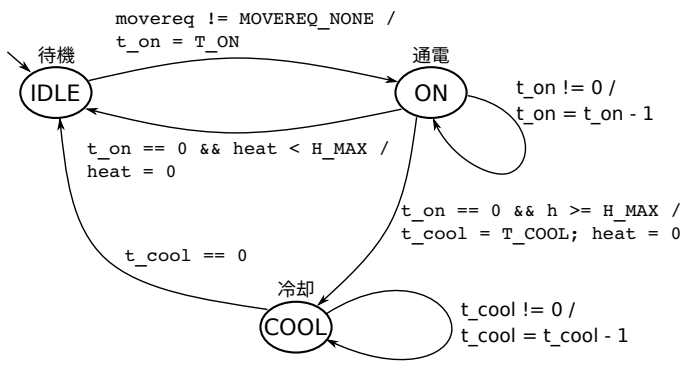


図 12.17: 有限状態機械の状態遷移図

```
4  /* 実行１サイクルの時間 */
5  #define T_CYCLE     10   /* [mSec] */
6
7  /* 入力によるポイント切替の要求 */
8  enum e_movereq {
9    MOVEREQ_NONE = 0, /* 要求なし */
10   MOVEREQ_A,        /* 定位へ切替要求 */
11   MOVEREQ_B,        /* 反位へ切替要求 */
12 };
13
14 /* ポイント開通方向(定位・反位) */
15 enum e_turnout_dir {
16   TURNOUT_DIR_NONE = 0,
17   TURNOUT_DIR_A, /* 定位 */
18   TURNOUT_DIR_B, /* 反位 */
19 };
20
21 extern enum e_movereq instate_init(void);
22 extern enum e_movereq instate_next(void);
23
24 #endif /* __TURNOUT_H__ */
```

ソースファイル instate.c (ポイント制御入力と動作: 実装)

ポイント制御入力の部分を解説します. 2 つの入力 XA と XB の変化がどのような組み合わせで起こっても正しく動作しなくてはなりません. そこで有限状態機械による動作の設計を行います. 図 12.18 にポイント制御入力の入力変化と動作を有限状態機械の形で示しています. 状態遷移図において各状態は入力 XA と XB の組で, T (True) と F (False) はそれぞれ C 言語の論理値の真と偽を表します. 回路の観点では L と HiZ で入力の値を表記していましたが, 本ファームウエアではそれぞれ真と偽で表現しています. つまり論理値の T は回路での L にあたり, 論理値の F は回路での HiZ にあたります. 初期状態は電源投入時の入力 XA と XB の値によって決まり, 入力の変化にともなって状態遷移が生じます. 状態遷移にはポイントマシンを駆動する遷移と駆動しない遷移の 2 種類があります. 状態遷移図において状態遷移の条件はポイント制御入力の変化ですから記入は省略し, 遷移時の動作であるポイントの切替動作だけが記入してあります.

プログラムコードではこの状態遷移図をそのまま書いています. プログラムの仕様や正しさの検討では最初に状態遷移図に対して検討を行い, そして状態遷移図とプログラムコードとの一致を確認します. 抽象度が低いプログラムコードを読んで検討するのと比較すれば, 抽象度が高い状態遷移図を検討するほうが間違いの発見が容易で, 作業はずいぶんと楽です.

ソースファイル番号 12.3: instate.c

```
1  #include <xc.h>
2  #include <stdint.h>
3  #include "mcr.h"
4  #include "io.h"
5  #include "turnout.h"
6
7  /* 有限状態機械(ポイント制御入力用) */
8  enum e_instate {
9    INSTATE_UNDEF = 0,
10   INSTATE_F_F,      /* XA=F, XB=F */
11   INSTATE_T_F,      /* XA=T, XB=F */
12   INSTATE_F_T,      /* XA=F, XB=T */
13   INSTATE_T_T,      /* XA=T, XB=T */
14 };
15
16 static enum e_instate curr = INSTATE_UNDEF;
17 static enum e_instate instate_read(void);
18
19 enum e_movereq instate_init(void)
20 {
```

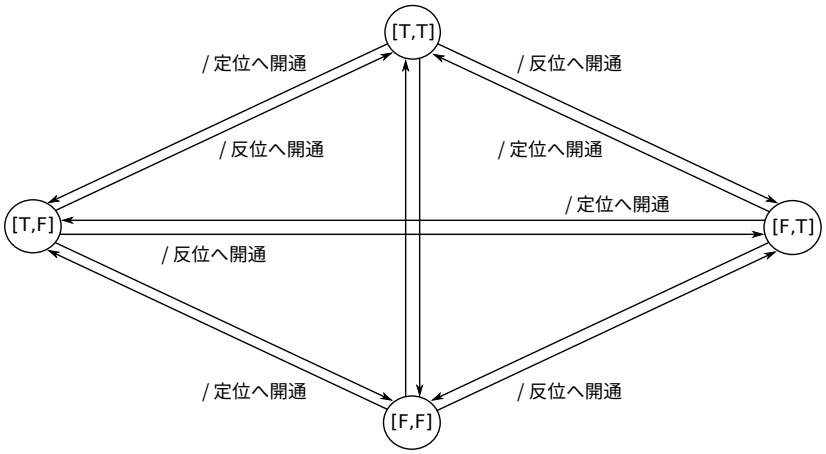


図 12.18: ポイント制御入力に対する状態遷移図 (状態は XA,XB の値の組. T は L, F は HiZ にそれぞれ対応.)

```
21    curr = INSTATE_UNDEF;
22    return instate_next();
23  }
24
25  enum e_movereq instate_next(void)
26  {
27    enum e_movereq movereq = MOVEREQ_NONE;
28    enum e_instate next = instate_read();
29    switch (curr) {
30    default:
31    case INSTATE_UNDEF:
32      switch (next) {
33      case INSTATE_F_F: movereq = MOVEREQ_A; break;
34      case INSTATE_T_F: movereq = MOVEREQ_A; break;
35      case INSTATE_F_T: movereq = MOVEREQ_B; break;
36      case INSTATE_T_T: movereq = MOVEREQ_B; break;
37      }
38      break;
39    case INSTATE_F_F:
40      switch (next) {
41      case INSTATE_F_F: break;
42      case INSTATE_T_F: movereq = MOVEREQ_A; break;
43      case INSTATE_F_T: movereq = MOVEREQ_B; break;
44      case INSTATE_T_T: break;
45      }
46      break;
47    case INSTATE_T_F:
48      switch (next) {
49      case INSTATE_F_F: break;
50      case INSTATE_T_F: break;
51      case INSTATE_F_T: movereq = MOVEREQ_B; break;
52      case INSTATE_T_T: movereq = MOVEREQ_B; break;
53      }
54      break;
55    case INSTATE_F_T:
56      switch (next) {
57      case INSTATE_F_F: break;
58      case INSTATE_T_F: movereq = MOVEREQ_A; break;
59      case INSTATE_F_T: break;
60      case INSTATE_T_T: movereq = MOVEREQ_A; break;
61      }
62      break;
63    case INSTATE_T_T:
64      switch (next) {
65      case INSTATE_F_F: break;
66      case INSTATE_T_F: movereq = MOVEREQ_A; break;
67      case INSTATE_F_T: movereq = MOVEREQ_B; break;
68      case INSTATE_T_T: break;
```

```
69      }
70      break;
71    }
72    curr = next;
73    return movereq;
74  }
75
76  static enum e_instate instate_read(void)
77  {
78    int8_t in_a_curr = DIR_IN_A();
79    int8_t in_b_curr = DIR_IN_B();
80    enum e_instate instate = INSTATE_UNDEF;
81    if        (!in_a_curr && !in_b_curr) {
82      instate = INSTATE_F_F;
83    } else if ( in_a_curr && !in_b_curr) {
84      instate = INSTATE_T_F;
85    } else if (!in_a_curr &&  in_b_curr) {
86      instate = INSTATE_F_T;
87    } else if ( in_a_curr &&  in_b_curr) {
88      instate = INSTATE_T_T;
89    }
90    return instate;
91  }
```

それぞれの状態に遷移したときのポイントの開通方向は以下の通りです.

- 状態 [F,F]
  開通方向は定位の場合も反位の場合もあります. 遷移前の開通方向と同じままです.
- 状態 [T,F]
  開通方向は定位です.
- 状態 [F,T]
  開通方向は反位です.
- 状態 [T,T]
  開通方向は定位の場合も反位の場合もあります. どの状態からこの状態へ遷移したかにより開通方向が決まります.

このソースファイルには状態 UNDEF が定義されていますが, これは第 15 章のサーボモータの制御でこのソースファイルを再利用可能にするためのものです. 関数 instate_next() は前回と今回のポイント制御入力の差分を検出するだけであり, 本ファームウエアではこの機能で充分です. しかし第 15 章のサーボモータの制御では電源投入時のポイント制御入力を知る必要があるため, これでは不十分です. そこで関数 instate_init() に少し工夫をして, 状態 UNDEF を定義することで電源投入時のポイント制御入力が分かるようにしています.

ソースファイル TurnoutCtr-Solenoid.c (主ソースコード)

このソースファイルで重要な定数の定義は以下のものです.

- ソレノイドに電流を流すサイクル数 T_ON
  ソレノイドに電流を流す期間を 5 サイクルと定義していますので, ソレノイドに電流を流す時間は $10 \times 5 = 50$ ミリ秒です.

ポイントマシンのソレノイドが焼損しないように, ソレノイドに電流が流れる時間の管理は慎重に行わないといけません. 十分に注意をしてプログラムコードを書くのが第一ですが, もしプログラムにバグがあってもソレノイドの焼損を防ぐよう, 以下の形でウオッチドッグタイマを用います.

- ソレノイドに電流を流している間はウオッチドッグタイマをクリアしない.
- バグによりソレノイドへの通電が長時間になれば, ウオッチドッグタイマはマイコンをリセットして再起動する.
- マイコンの起動時にはソレノイドに流す電流を止める.

プログラムコードの冒頭部ではリセット要因を調べており, もしウオッチドッグタイマによるリセットであれば無限ループに入るようにしています. もちろんここでも再びウオッチドッグタイマがタイムアウトしてマイコンにリセットがかかります. つまりファームウエアにもしバグがあれば, 人手でいったん電源を切らない限りはポイントマシンへの通電は一切行わないようにしています.

本ファームウエアではさらに, 短期間に頻繁のポイント切り替えの入力が続くとしばらくの間はポイント切り替えを禁止する機能を実現しています. 異常に頻繁な切り替えの発生があってもこの機能によりポイントマシンのソレノイドを発熱から守ります.

この機能の実現方法の考え方を以下に説明します. ポイントマシンは通電により熱くなると考えて, その熱さを仮想的に数値で勘定します. ただし物理的な温度を測るのではなく, ただの変数値です. 通電すると熱さの値を大きく増加させ, 通電していないときは熱さの値を少しづつ減少させてゆきます. そして熱さの値が一定以上の場合には, 熱さの値が 0 に減少するまでポイントマシンの切り替えを停止します. この仕組みにより, 何度かまでは頻繁に切り替えはできても, それが続けばポイントマシンの切り替えをしばらく受け付けないという振る舞いを実現します.

図 12.17 に示す状態遷移図を持つ有限状態機械を実現するソースファイルを以下に示します.

ソースファイル番号 12.4: TurnoutCtr-Solenoid.c

```
1  /*********************************************
2   * ソレノイド式ポイント切替コントローラ / PIC16F18313
3   * by Hirotsugu Kakugawa
4   *********************************************/
5  /*
6   * 対象：ソレノイド式 - Kato, Tomix, Peco 等
7   *
8   * ※自己停止形DCモータ式(Fulgrex等)には使用禁止
9   *   (正常に切替動作しない)
10  */
11 #include <xc.h>
12 #include <stdint.h>
13 #include "mcr.h"
14 #include "io.h"
15 #include "turnout.h"
16
17 /* ソレノイドに電流を流す期間 */
18 #define T_ON          5   /* [サイクル数] */
19 /* 通電期間1サイクルの重み(コイル加熱防止用) */
20 #define H_W           10
21 /* 通電期間重みの蓄積上限値(コイル加熱防止用) */
22 #define H_MAX       500
23 /* 切替禁止期間(コイル加熱防止用) */
24 #define T_COOL     1000   /* [サイクル数] */
25
26 /* 有限状態機械(通電時間制御用) */
27 enum e_state {
28   STATE_IDLE = 0,   /* 待機状態 */
29   STATE_ON,         /* 通電状態 */
30   STATE_COOL,       /* 冷却状態 */
31 };
32
33 /*-------------------------------------------------------
34  * MAIN
35  *-------------------------------------------------------*/
```

```
36 void main(void)
37 {
38   /* システム初期化 */
39   System_Init();
40   /* 出力の初期化 */
41   OUT_TURNOUT_A_OFF();
42   OUT_TURNOUT_B_OFF();
43   /* 双方向非開通 */
44   OUT_OD_DIR_A_LOW();
45   OUT_OD_DIR_B_LOW();
46   /* リセット要因がウオッチドックタイマなら,
47    * 安全性のため無限ループに入る.(ソフトウエアの重大バク、
48    * またはハードウエア障害と考えられるため.) */
49   if (STATUSbits.nTO == 0) {
50     for (;;) {
51       ;
52     }
53   }
54   /* ウオッチドッグタイマ初期化 (512[mSec] typ.) */
55   WDTCONbits.WDTPS = 0b01001;
56   CLRWDT();
57   /* 入力(=SWあるいは他機器の出力)の安定を待機 (1[Sec]) */
58   uint8_t t_wakeup = 100;
59   while (t_wakeup != 0) {
60     t_wakeup--;
61     __delay_ms(10);
62     CLRWDT();
63   }
64   /* 制御状態の初期化 */
65   enum e_state state     = STATE_IDLE;
66   enum e_turnout_dir dir = TURNOUT_DIR_NONE;
67   uint16_t t_on   = 0;  /* 通電残り時間 */
68   uint16_t t_cool = 0;  /* 冷却残り時間 */
69   uint16_t heat = 0;  /* 通電による熱の量 */
70   /* 入力状態の初期化 */
71   instate_init();
72   /* 主ループ */
73   for (;;) {
74     __delay_ms(T_CYCLE);
75     /* (ここでウオッチドッグタイマはクリアしない) */
76     /* ポイント切替のための有限状態機械 */
77     switch (state) {
78     case STATE_IDLE: /* 待機状態 */
79       CLRWDT();
80       OUT_TURNOUT_A_OFF();
81       OUT_TURNOUT_B_OFF();
82       if (heat != 0) {
83         heat--;
84       }
85       /* ポイント制御入力を読む */
86       enum e_movereq movereq = instate_next();
87       /* ポイント切替要求 */
88       switch (movereq) {
89       default:
90       case MOVEREQ_NONE:
91         break;
92       case MOVEREQ_A: /* 定位への切替要求 */
93         state = STATE_ON;
94         dir = TURNOUT_DIR_A;
95         /* 開通方向を出力(双方向非開通) */
96         OUT_OD_DIR_A_LOW();
97         OUT_OD_DIR_B_LOW();
98         /* 通電開始 */
99         OUT_TURNOUT_A_ON();
100        OUT_TURNOUT_B_OFF();
101        t_on = T_ON;
102        break;
103      case MOVEREQ_B: /* 反位への切替要求 */
104        state = STATE_ON;
```

```
105           dir = TURNOUT_DIR_B;
106           /* 開通方向を出力(双方向非開通) */
107           OUT_OD_DIR_A_LOW();
108           OUT_OD_DIR_B_LOW();
109           /* 通電開始 */
110           OUT_TURNOUT_A_OFF();
111           OUT_TURNOUT_B_ON();
112           t_on = T_ON;
113           break;
114         }
115         break;
116       case STATE_ON: /* 通電状態 */
117         if (t_on != 0) { /* 通電の継続 */
118           t_on = t_on - 1; /* 残り時間 */
119           heat = heat + H_W; /* 通電蓄積時間 */
120         } else {       /* 通電の終了 */
121           CLRWDT();
122           state = STATE_IDLE;
123           OUT_TURNOUT_A_OFF();
124           OUT_TURNOUT_B_OFF();
125           /* 開通方向を出力 */
126           if (dir == TURNOUT_DIR_A) {
127             OUT_OD_DIR_A_Z();
128             OUT_OD_DIR_B_LOW();
129           } else if (dir == TURNOUT_DIR_B) {
130             OUT_OD_DIR_A_LOW();
131             OUT_OD_DIR_B_Z();
132           }
133           /* 必要に応じて冷却期間に入る */
134           if (heat >= H_MAX) {
135             /* 通電蓄積時間が一定値を超過: 冷却状態へ */
136             state = STATE_COOL;
137             t_cool = T_COOL;
138             heat = 0;
139           }
140         }
141         break;
142       case STATE_COOL: /* 冷却状態 */
143         CLRWDT();
144         if (t_cool != 0) { /* 冷却期間の継続 */
145           t_cool--; /* 残り時間 */
146         } else {           /* 冷却期間の終了 */
147           state = STATE_IDLE;
148         }
149         break;
150       }
151     }
152   }
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言を行います.

ソースファイル番号 12.5: mcr.h

```
1  #if !defined(__MCR_H__)
2  #define __MCR_H__
3
4  #define _XTAL_FREQ 32000000  /* 32MHz */
5
6  extern void System_Init(void);
7
8  #endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化や, I/O ピンのデジタル入出力の設定を行います.

ソースファイル番号 12.6: mcr.c

```
/*-------------------------------------------------------
 * MCR設定&周辺モジュール (PIC16F18313)
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"

/*-----------------------------------------*/
/* MCR初期化 */
void System_Init(void)
{
  /* 発振器 (FOSC=32MHz) */
  OSCCON1 = 0x00; /* HFINTOSC with 2x PLL */
  OSCCON3 = 0x00;
  OSCEN   = 0x00;
  OSCFRQ  = 0x06; /* 32MHz */
  OSCTUNE = 0x00;
  while (PLLR == 0) {
    ;
  }
  /* I/Oピン */
  LATA   = 0x00;
  TRISA  = 0x0C; /* RA3,RA2 */
  ANSELA = 0x00;
  WPUA   = 0x00;
  ODCONA = 0x00;
}

/*-----------------------------------------*/
/* CONFIG1 */
#pragma config FEXTOSC  = OFF
#pragma config RSTOSC   = HFINT32
#pragma config CLKOUTEN = OFF
#pragma config CSWEN    = ON
#pragma config FCMEN    = ON
/* CONFIG2 */
#pragma config MCLRE    = OFF /* MCLRをI/Oピン */
#pragma config PWRTE    = OFF
#pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
#pragma config LPBOREN  = OFF
#pragma config BOREN    = ON
#pragma config BORV     = LOW
#pragma config PPS1WAY  = ON
#pragma config STVREN   = ON
#pragma config DEBUG    = OFF
/* CONFIG3 */
#pragma config WRT      = OFF
#pragma config LVP      = OFF /* 低電圧書込不可 */
/* CONFIG4 */
#pragma config CP       = OFF
#pragma config CPD      = OFF
```

このソースファイルでもうひとつ重要な点があります. コンフィグレーションビット WDTE の値は, SWDTE ビットの値に関わらず WDT を有効とする設定にしています. つまりファームウエアのバグが SWDTE ビットにどのような値を書き込んでも, WDT を無効にはできないという設定です.

課題

1. 表 12.4 で定義した開通状態出力の仕様では HiZ を開通としています. これとは逆に L を開通とする仕様も考えられます. これら 2 通りの仕様に対してそれぞれの長短を論じて下さい.
2. 表 12.4 で定義した開通状態出力の仕様では HiZ を開通としています. もし仮に L を開通と変更した場合には, 本書の他装置ではどのような変更が必要となるでしょうか.
3. パワーパックにおける WDT のタイムアウト時間はどのような値にするのが適切かを議論して下さい.

第 13 章

DC モータ方式用
マイコン制御ポイント切替器

本章のファームウエアをソレノイド方式のポイントマシンで使用すると焼損したり発火するなど重大な事故になります. 第 12 章との違いはマイコンのファームウエアだけなので外見からは全く区別が付きません. 必ずマイコンチップに目立つ印を付けるなど, 外見で区別が付くようにして下さい.

本章では DC モータ方式のポイントマシンを制御する装置を紹介します. ハードウエアは第 12 章で作成したポイント切替器をそのまま使用して, マイコンのファームウエアのみを変更します. 本装置で対象とするのは Fulgrex ACC001 Slow Action Point Motor などで, 切替が完了すればポイントマシン内蔵の接点により自動的に通電が遮断される DC モータ方式のポイントマシンです. この種のポイントマシンは電圧を常に印加し続ける形で使用しますので, 本章のファームウエアでもそのような動作を行います.

本章で紹介するファームウエアの特徴は以下の通りです.

- 保護回路内蔵のモータドライバ IC TB6643KQ を使用
  短絡防止機能・加熱防止機能などが内蔵されており, 手軽にしかも安価に安全性の高い回路が実現できます.
- 電子制御が可能
  機械的なスイッチは使用しないので手動操作を不要にできます. 電子信号でポイントの開通方向の制御をしますので, 他の装置との連携動作が容易です.

本章で学ぶ内容は以下の通りです.

- DC モータ式ポイントマシンの制御方法

この装置のファームウエアの大まかな構造を図 13.1 に示します. 入力に従ってモータドライバを制御して, DC モータへ電流を流し続けるという, 極めて単純なものです. 本装置には開通方向を出力する機能を持たせていますが, 切替完了を機械的に検知する方法がないため, 一定時間の経過により切替完了を出力しています. 少々複雑になっている理由はこの機能の実現のためです.

13.1 使用方法

本装置の使用方法を説明します. 主な配線の概略は図 12.4 (210 ページ) で示したものと全く同じです. 基本接続は以下の通りです.

1. DC ジャックに AC アダプタを接続します.
2. ターミナル DIR-IN がポイント切り替え方向を制御する入力です. これにスイッチを接続します.
3. ターミナル SW-OUT とポイントマシンに接続します.

```
1  main()
2  {
3    for (;;) {
4      if（ポイント制御入力に変化が発生）{
5        入力に従った開通方向へモータドライバをオンに
             する;
6      }
7      if（切り替えが発生してから一定時間が経過）{
8        開通方向を出力;
9      }
10   }
11 }
```

図 13.1: ファームウエア構造 (概要)

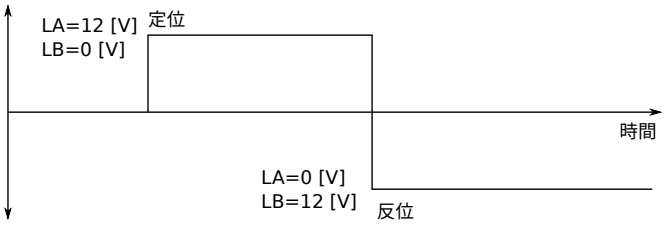


図 13.2: 出力波形の模式図

4. ターミナル OUT-A と OUT-B は開通方向を示すアクセサリ制御出力です. これは使用しなくても構いません.

ポイント切替の仕様は表 12.2 (210 ページ) で示したものと全く同じです. また電源投入時にはポイント制御入力の値に関わらずポイントマシンを駆動しない点も第 12 章のファームウエアと同様です. 第 12 章と本章のファームウエアの特徴は, ポイント切替の仕様がともに同一という点にあります. つまりポイントマシンがソレノイド方式であるか DC モータ方式であるかを操作する側は一切の区別をせずに使用できるという点です. レイアウト工事でポイントマシンを変更する際にポイント切替器も合わせて変更すれば, それ以外の機器の変更は一切不要です.

図 13.2 に示す波形の出力が本装置の目標です.

13.2　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 13.1: 【ソースファイル一覧】 ポイント切替器: DC モータ方式用

初	頁	番号	ファイル名	内容	行数
✓	232	13.1	TurnoutCtr-Motor.c	主ソースコード	157
	222	12.2	turnout.h	諸定義: ヘッダ	24
	223	12.3	instate.c	ポイント制御入力と動作: 実装	91
	221	12.1	io.h	I/O ピン割付: ヘッダ	21
	228	12.5	mcr.h	MCR・周辺モジュール: ヘッダ	8

(つづく)

(つづき)

初　頁	番号	ファイル名	内容	行数
229	12.6	mcr.c	MCR・周辺モジュール: 実装	51
			計 352 行 (うち初出 157 行, 既出 195 行)	

本ファームウエアは 3 状態の有限状態機械を基礎としており, その構造を詳しく示したのが図 13.3 です.

ソースファイル TurnoutCtr-Motor.c (主ソースコード)

本ファームウエアは, 開通方向にモータドライバの電圧を出力する簡単な構造です. 開通方向出力 OUT-A と OUT-B は, ポイントの切替中は非開通を出力し切替完了後に開通を出力します. ただし切替完了を機械的に検知する方法を持っていないため, 一定時間の経過により切替完了とみなして開通を出力します. なおこの時間の具体的な値は 3 秒 (= 300 × 10 ミリ秒) としてあります.

ソースファイル番号 13.1: TurnoutCtr-Motor.c

```
1  /*********************************************
```

```
1  初期化;
2  state = STATE_ILDE;
3  for (;;) {
4    __delay_ms(T_CYCLE);
5    switch (state) {
6    case STATE_IDLE:       /* 待機中 */
7      if (定位へ切替要求がある) {
8        state = STATE_A;
9        t = T_TO_FINISH;
10       定位へモータドライバの通電を開始;
11     } else if (反位へ切替要求がある) {
12       state = STATE_B;
13       t = T_TO_FINISH;
14       反位へモータドライバの通電を開始;
15     }
16     break;
17   case STATE_A: /* 定位へ切り替え動作中 */
18     t = t - 1;
19     if (t == 0) {
20       開通方向を出力;
21     }
22     break;
23   case STATE_B: /* 反位へ切り替え動作中 */
24     t = t - 1;
25     if (t == 0) {
26       開通方向を出力;
27     }
28     break;
29   }
30 }
```

図 13.3: ファームウエア構造 (詳細)

```
 * モータ式ポイント切替コントローラ / PIC16F18313
 * by Hirotsugu Kakugawa
 ********************************************/
/*
 * 対象：自己停止形DCモータ式 - Fulgrex等
 *
 * ※ソレノイド式(Kato, Tomix, Peco 等)には使用禁止
 *   (コイル焼損となる)
 */
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "turnout.h"

/* (擬似)切替完了時間 [サイクル数] */
#define T_TO_FINISH  300

/* 有限状態機械(開通方向出力制御用) */
enum e_state {
  STATE_IDLE = 0,   /* 待機中 */
  STATE_A,          /* 定位へ切替動作中 */
  STATE_B,          /* 反位へ切替動作中 */
};

void move_to_a(void);
void move_to_b(void);

/*-----------------------------------------------------
 * MAIN
 *-----------------------------------------------------*/
void main(void)
{
  /* システム初期化 */
  System_Init();
  /* モーター出力の初期化 */
  OUT_TURNOUT_A_OFF();
  OUT_TURNOUT_B_OFF();
  /* 開通方向出力 */
  OUT_OD_DIR_A_LOW(); /* 定位: 非開通 */
  OUT_OD_DIR_B_LOW(); /* 反位: 非開通 */
  /* リセット要因がウオッチドックタイマなら,
   * 安全性のため無限ループに入る.(ソフトウエアの重大バク、
   * またはハードウエア障害と考えられるため.) */
  if (STATUSbits.nTO == 0) {
    for (;;) {
      ;
    }
  }
  /* ウオッチドッグタイマ初期化: 512 [mSec] */
  WDTCONbits.WDTPS = 0x09; /* 0b01001 */
  CLRWDT();
  /* 入力(=SWあるいは他機器の出力)の安定を待機 (1[Sec]) */
  uint8_t t_wakeup = 100;
  while (t_wakeup != 0) {
    t_wakeup--;
    __delay_ms(10);
    CLRWDT();
  }
  /* 切替完了残り予想時間 */
  uint16_t t = 0;
  /* 入力状態の初期化 */
  instate_init();
  /* 開通方向出力制御の状態 */
  enum e_state state = STATE_IDLE;
  /* 主ループ */
  for (;;) {
    __delay_ms(T_CYCLE);
    CLRWDT();
```

```
71      /* ポイント制御入力を読む */
72      enum e_movereq movereq = instate_next();
73      /* 開通方向出力の制御 */
74      switch (state) {
75      case STATE_IDLE:   /* 待機中 */
76        switch (movereq) {
77        case MOVEREQ_NONE:
78          break;
79        case MOVEREQ_A: /* 定位への切替要求 */
80          state = STATE_A;
81          t = T_TO_FINISH;
82          move_to_a();
83          break;
84        case MOVEREQ_B: /* 反位への切替要求 */
85          state = STATE_B;
86          t = T_TO_FINISH;
87          move_to_b();
88        }
89        break;
90      case STATE_A: /* 定位へ切替動作中 */
91        /* タイマ */
92        t = t - 1;
93        if (t == 0) { /* 切替完了 */
94          state = STATE_IDLE;
95          /* 開通方向出力 */
96          OUT_OD_DIR_A_Z();   /* 定位: 開通 */
97          OUT_OD_DIR_B_LOW(); /* 反位: 非開通 */
98        } else {      /* 切替中 */
99          switch (movereq) {
100         case MOVEREQ_NONE:
101           break;
102         case MOVEREQ_A: /* 定位への切替要求 */
103           break;
104         case MOVEREQ_B: /* 反位への切替要求 */
105           state = STATE_B;
106           t = T_TO_FINISH;
107           move_to_b();
108           break;
109         }
110       }
111       break;
112     case STATE_B: /* 反位へ切替動作中 */
113       /* タイマ */
114       t = t - 1;
115       if (t == 0) { /* 切替完了 */
116         state = STATE_IDLE;
117         OUT_OD_DIR_A_LOW(); /* 定位: 非開通 */
118         OUT_OD_DIR_B_Z();   /* 反位: 開通 */
119       } else {      /* 切替中 */
120         switch (movereq) {
121         case MOVEREQ_NONE:
122           break;
123         case MOVEREQ_A: /* 定位への切替要求 */
124           state = STATE_A;
125           t = T_TO_FINISH;
126           move_to_a();
127           break;
128         case MOVEREQ_B: /* 反位への切替要求 */
129           break;
130         }
131       }
132       break;
133     }
134   }
135 }
136
137 /* 定位に向けて切替動作を開始 */
138 void move_to_a(void)
139 {
```

```
140     /* 開通方向出力 */
141     OUT_OD_DIR_A_LOW(); /* 定位: 非開通 */
142     OUT_OD_DIR_B_LOW(); /* 反位: 非開通 */
143     /* 通電 */
144     OUT_TURNOUT_A_ON();
145     OUT_TURNOUT_B_OFF();
146   }
147
148   /* 反位に向けて切替動作を開始 */
149   void move_to_b(void)
150   {
151     /* 開通方向を出力 */
152     OUT_OD_DIR_A_LOW(); /* 定位: 非開通 */
153     OUT_OD_DIR_B_LOW(); /* 反位: 非開通 */
154     /* 通電 */
155     OUT_TURNOUT_A_OFF();
156     OUT_TURNOUT_B_ON();
157   }
```

課題

1. 本装置の状態遷移図を描いて下さい. ただしポイント制御入力 DIR-IN, モータの通電状態, 開通方向出力 OUT-A と OUT-B, および変数 t のそれぞれの値が分かるように描いて下さい.
2. 本装置のファームウエアでは DC モータに常に電圧を印加しています. ある程度の時間が経過すればポイントの切替は完了しているはずなので, 電圧の印加を止めても良いように思えます. 電圧の印加は継続したほうが良いのか, 止めたほうが良いのか, 検討を行って下さい.

第 14 章

リレー制御型ポイント切替器

本章ではマイコンやモータドライバは使用せずに, リレーや大容量電解コンデンサを使用した簡易な構造のポイント切替器を紹介します. 2 線ソレノイド方式用ポイントマシン, 3 線ソレノイド方式用ポイントマシン, DC モータ方式用ポイントマシンそれぞれに専用の回路を設計します. 本章で紹介する回路の特徴は以下の通りです.

- マイコン不使用の単純な回路
- 低価格
- 電子制御が可能

本章で学ぶ内容は以下の通りです.

- リレーの制御方法

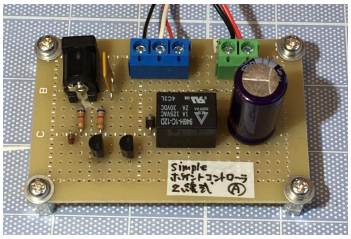
ソレノイド方式のポイントマシンを駆動する方法として, 昔から大容量電解コンデンサを用いた方法が知られています. 電解コンデンサの充電・放電の際には一瞬だけ電流が流れることから, ソレノイド方式のポイントマシンの駆動とは相性が良いからです. 本章ではこれにポイント制御入力のインターフェースを用意して, 電子的な制御を可能とします. そのため第 29 章と第 30 章で紹介する鉄道模型の運転自動システムでも本装置を利用可能です.

本装置の特色として低価格であることを延べましたが, これは組立時における短期的な観点での話です. 電解コンデンサやリレーの寿命を考えれば, 長期的な観点でもそうであるかは検討を必要とします.

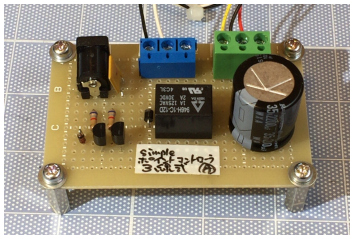
14.1　使用方法

本章で紹介する各装置の接続方法は以下の通りです.

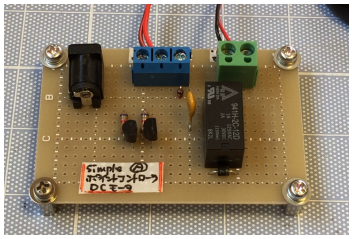
- 2 線ソレノイド方式ポイント切替器 (第 14.2 節): 図 14.2a
- 3 線ソレノイド方式ポイント切替器 (第 14.3 節): 図 14.2b



(a) 2 線ソレノイド方式用



(b) 2 線ソレノイド方式用

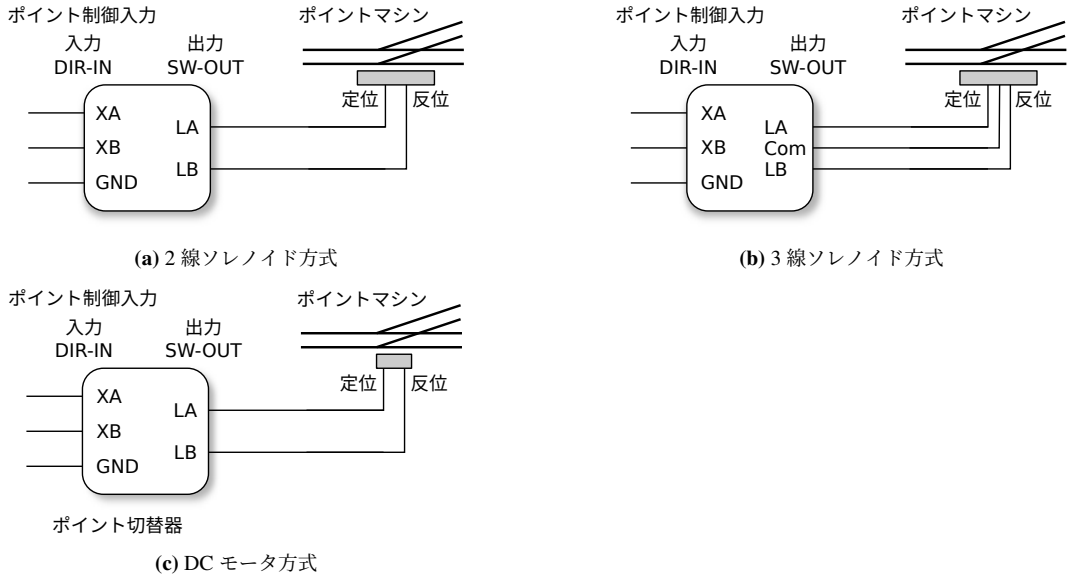


(c) DC モータ方式用

図 14.1: ポイント切替器の試作例

表 14.1: ポイント切替器の仕様

入力		開通方向
XA	XB	
*	HiZ	定位
*	L	反位



(a) 2 線ソレノイド方式

(b) 3 線ソレノイド方式

(c) DC モータ方式

図 14.2: 接続方法

- DC モータ方式用ポイント切替器 (第 14.4 節) 図 14.2c

ポイント切替の仕様を表 14.1 に示します. この仕様は第 12 章の表 12.2 (213 ページ) に示した仕様と部分的な互換性がありますが, 完全な互換ではありません. すなわち電源投入時にポイント制御入力 XB の値の変化があればポイントマシンは駆動され切替動作が起きる設計となっている点が第 12 章で紹介したポイント切替器の動作の違いであり, 注意を要する点です.

本章でのソレノイド式ポイントマシン用のポイント切替器には別の使用上の注意があります. それは装置の電源投入時には電解コンデンサが充電されているとは限らないことに起因するものです. レイアウト全体に電源投入をして鉄道模型の運転を開始したとします. この時にポイントの開通方向はまちまちで, ポイントを定位に開通しようとしても切り替えが不発に終わる場合があります. なぜなら, 定位に切り替えるには電解コンデンサが充電されている必要があるのですが, 電源投入時には充電されているとは限らないからです. そのため電源投入時には初期操作として, ポイントを反位, 定位の順で切り替えて, 定位に開通させるようにします. こうすることで以降では切り替えが不発に終わることは生じません. この初期操作は第 12 章で紹介した装置では不要なものでしたが, 本章の装置では簡易型としたゆえに生じる制約です.

14.2　2 線ソレノイド方式ポイント切替器

2 線ソレノイド方式用ポイント切替器を紹介します. まず最初に設計方針を説明します. ポイントの切り替えの制御はポイント制御入力によりリレーを駆動して, 大容量電解コンデンサの充放電でソレノイドを駆動します. 従来の方式では機械的なスイッチを人間が手作業で駆動していたものをリレーに置き換えています. ポイ

表 14.2: ポイント切替の仕様

入力		出力		開通	動作	
XA	XB	LA	LB		リレー	C1
*	HiZ	+	−	定位	オフ	放電
*	L	−	+	反位	オン	充電

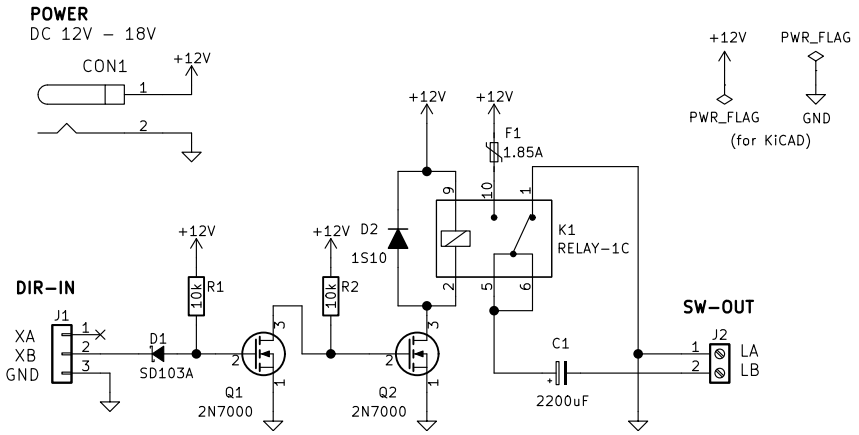


図 14.3: 【回路図】 2 線ソレノイド方式用リレー制御型ポイント切替器

ント切替の仕様を表 14.2 に示します. これは表 12.2 と同じものですが, 回路内部の状態の説明も加えていますので参考にしてください.

本章の回路では, リレーと電解コンデンサという寿命が短く信頼性の低い部品を使用しています. リレーは接点の固着などの機械的な故障があり, 電解コンデンサはドライアップが起きるため, やがて交換すべき時が来ます. 一番危険なものは電解コンデンサの 2 端子間がショート状態になるショート故障です. このときには電源とソレノイドが短絡してソレノイドが焼損しますから, 過電流を遮断するために保護部品のポリスイッチを挿入します.

表 14.3: 【部品表】 2 線ソレノイド方式用リレー制御型ポイント切替器

記号	数量	部品名
Q1, Q2	2	FET 2N7000 [I-09723]
D2	1	ダイオード 1S10　100V 1A [I-00126]
D1	1	ショットキーバリアダイオード SD103A [I-04271]
F1	1	ポリスイッチ MF-R185　1.85A [P-12629]
C1	1	電解コンデンサ 2200μF　耐圧 35V 以上 [P-02723]
R1, R2	2	抵抗 10kΩ　1/4W [R-25103]
K1	1	リレー 1 回路 C 接点　12V 1A [P-01345]
J2	1	ターミナルブロック 2 ピン [P-01306]
J1	1	基板用コネクタ 3 ピン　JST XH コネクタ B3B-XH-A(LF) [C-12248]
	2	コネクタハウジング 3 ピン　JST XH コネクタ XHP-3 [C-12256]
	6	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]

(つづく)

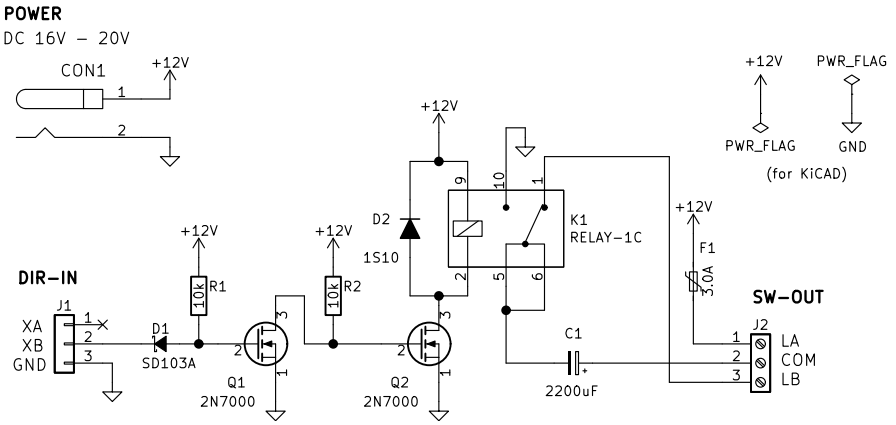


図 14.4: 【回路図】 3 線ソレノイド方式用リレー制御型ポイント切替器

(つづき)

記号	数量	部品名
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 12V　出力 3A [M-10661]
		ほか基板・配線材等

2 線ソレノイド方式用ポイント切替器の回路図を図 14.3 に, 部品リストを表 12.3 に, それぞれ示します. 本回路ではリレーと電解コンデンサの寿命を縮めない工夫をしています. 劣化が進むのは, リレーはコイルに電流が流れている状態の時および電解コンデンサは電圧が印加されて充電状態の時ですから, ポイントが定位に開通している間はリレーのコイルには電流が流れず, 電解コンデンサには電圧が印加されない回路にしています. 逆を言えば, ポイントが反位に開通している間に限りリレーのコイルに電流を流し, 電解コンデンサに電圧を印加して充電します. ポイントは通常は定位に開通させておきますから, 可能な限りリレーと電解コンデンサの劣化が遅れることになります. 第 12 章では, 安全側線のポイントのように分岐側を定位とする場合には本装置のターミナル SW-OUT の出力 LA と LB を逆に配線する旨を述べました. その理由は第 12 章で述べた論理的な観点だけでなく, 本装置のリレーと電解コンデンサの劣化を遅らせるという物理的な観点にもよるものです.

14.3　3 線ソレノイド方式ポイント切替器

3 線ソレノイド方式用ポイント切替器を紹介します. これは 2 線ソレノイド方式の場合とほとんど一緒です. 3 線ソレノイド方式用ポイント切替器の回路図を図 14.4 に, 部品リストを表 14.4 に, それぞれ示します. ポイントマシンによってはポリスイッチに 3A のものを, リレーに 3A あるいは 5A のものを用いたほうが良いかもしれません. ポイントマシンに流れる電流を各メーカは公式に発表していないため, 想定すべき値は残念ながら何とも言えません.

表 14.4: 【部品表】 3 線ソレノイド方式用リレー制御型ポイント切替器

記号	数量	部品名
Q1, Q2	2	FET 2N7000 [I-09723]
D2	1	ダイオード 1S10　100V 1A [I-00126]

(つづく)

(つづき)

記号	数量	部品名
D1	1	ショットキーバリアダイオード SD103A [I-04271]
F1	1	ポリスイッチ MF-R300　3.0A [P-12630]
C1	1	電解コンデンサ 2200μF　耐圧 35V 以上 [P-02723]
R1, R2	2	抵抗 10kΩ　1/4W [R-25103]
K1	1	リレー 1 回路 C 接点　12V 2A [P-07343]
J2	1	ターミナルブロック 3 ピン [P-01307]
J1	1	基板用コネクタ 3 ピン　JST XH コネクタ B3B-XH-A(LF) [C-12248]
	2	コネクタハウジング 3 ピン　JST XH コネクタ XHP-3 [C-12256]
	6	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 16V　出力 4A [M-00407]
		ほか基板・配線材等

14.4　DC モータ方式用ポイント切替器

DC モータ方式用ポイント切替器を紹介します. 上で説明したソレノイド方式の場合のように電解コンデンサを用いて一瞬だけ電流を流すのではなく, 常にポイントマシンに電圧を印加し続ける点で大きく異なります. そのため DC モータ方式用ポイント切替器を間違ってソレノイド方式のポイントマシンに使用するとポイントマシンは焼損しますから十分に注意して下さい. DC モータ方式用ポイント切替器の回路図を図 14.5 に, 部品リストを表 14.5 に, それぞれ示します. 本装置ではリレーに 2 回路型のものを使用し, 逆転スイッチの形にして DC モータに印加する電圧の極性を反転させます.

表 14.5: 【部品表】 DC モータ方式用リレー制御型ポイント切替器

記号	数量	部品名
Q1, Q2	2	FET 2N7000 [I-09723]

(つづく)

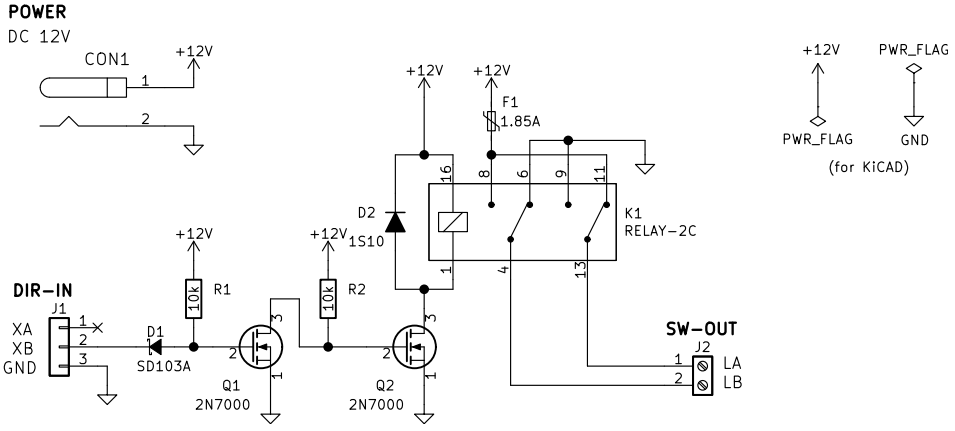


図 14.5: 【回路図】 DC モータ方式用リレー制御型ポイント切替器

(つづき)

記号	数量	部品名
D2	1	ダイオード 1S10　100V 1A [I-00126]
D1	1	ショットキーバリアダイオード SD103A　[I-04271]
F1	1	ポリスイッチ MF-R185　1.85A [P-12629]
R1, R2	2	抵抗 10kΩ　1/4W [R-25103]
K1	1	リレー 2 回路 C 接点　12V 2A [P-01228]
J2	1	ターミナルブロック 2 ピン　[P-01306]
J1	1	基板用コネクタ 3 ピン　JST XH コネクタ B3B-XH-A(LF) [C-12248]
	2	コネクタハウジング 3 ピン　JST XH コネクタ XHP-3 [C-12256]
	6	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
CON1	1	DC ジャック　[C-09408]
	1	AC アダプタ 12V　出力 3A [M-10661]
		ほか基板・配線材等

課題

1. マイコンは使用せず, モータドライバ IC を使用するポイント切替器を設計・製作して下さい. ポイントへの通電時間の制御はアナログ回路によるパルス生成で行って下さい.

第 15 章

サーボモータ制御器

本章ではホビー用サーボモータを制御する装置を紹介します. スローアクションポイントマシンや踏切の腕木式遮断機など, 鉄道模型レイアウト上のさまざまギミックの制御への応用が可能です. 注意が必要なのは, サーボモータには統一的な規格は存在せず, サーボモータにより制御方法が多少異なるという点です. 本装置は典型的なサーボモータで動作するように設計していますが, 全てのサーボモータで使用できるとは限りません. 各サーボモータが本装置で利用可能かどうかは各自で調査検討をして下さい.

本章で紹介する回路とファームウエアの特徴は以下の通りです.

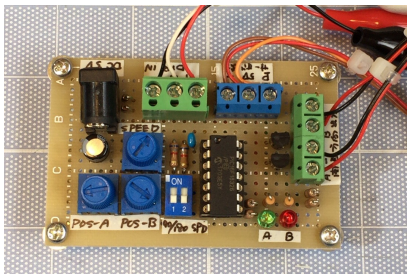
- 入力信号の仕様がポイント切替器と同一
  第 12 章で紹介したポイント切替器と同一の入力信号の仕様としていますから, 他のポイント切替器と同じ使い方で本装置をスローアクションポイントマシンとして使用可能です.
- サーボモータの回転角度や回転速度を可変抵抗器で調節可能
  サーボモータの動作の調節は基板上の可変抵抗器の操作で簡単に回転角度や回転速度が調節可能です. ファームウエア内の定数の調整で行うとか, あるいは PECO PLS-120 SmartSwitch のような専用の設定装置を必要としません.

本章で学ぶ内容は以下の通りです.

- PIC マイコンのタイマモジュール TMR1
- サーボモータの使用方法

これまでに作成した装置を合わせれば, 2 線ソレノイド方式, 3 線ソレノイド方式, DC モータ方式, サーボモータ方式のポイントマシンの全てを, それらの違いを意識することなく統一的なポイント制御入出力の信号で外部から制御可能となります.

図 15.2 に示すように入力信号に従ってサーボモータの出力を制御するのが本装置の目標です.



(a) 試作例



(b) サーボモータの例

図 15.1: サーボモータ制御器

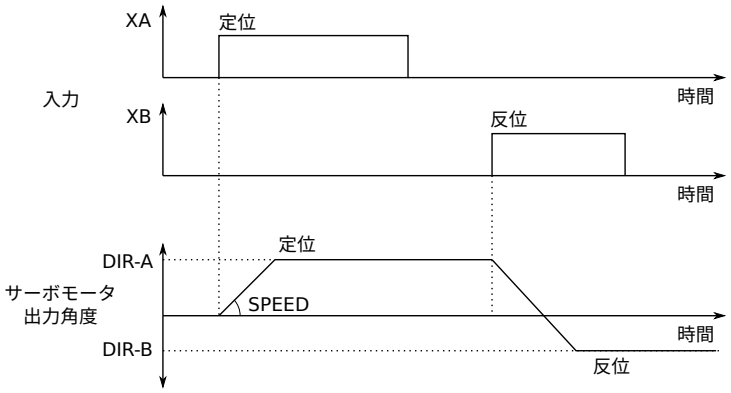


図 15.2: 入力信号とサーボモータ出力の模式図

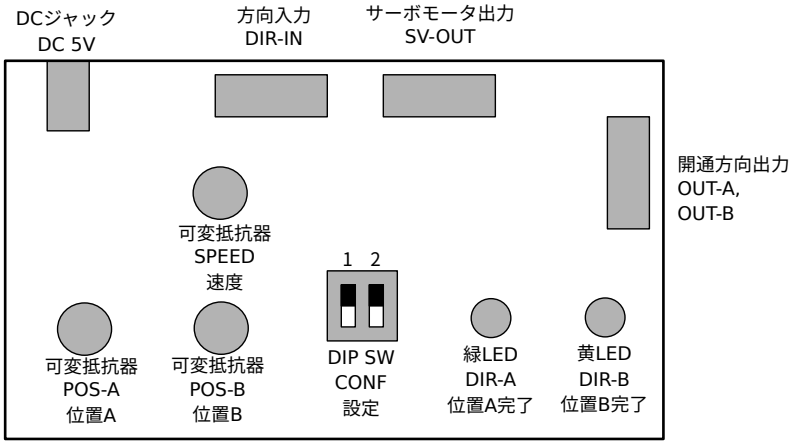


図 15.3: 操作表示部品の配置例と名称

15.1　使用方法

本装置の使用方法を説明します. 以下の説明で使用する部品名称は図 15.3 に配置例とともに示しています. 基本接続は以下の通りです.

1. DC ジャックに 5V の AC アダプタを接続します. **間違って 12V 出力の AC アダプタを使用すると回路が破損する場合がありますので, 十分に注意して下さい.**
2. ポイント制御入力 DIR-IN と 2 つの開通方向出力 OUT-A および OUT-B は, 第 12 章で説明したものと同一です.
3. サーボモータ出力 SV-OUT はサーボモータに接続します. 図 15.4 に例を示します.

本装置には設定用の可変抵抗器が 3 つあります.

- 回転速度設定 SPEED
  モータ軸の回転速度を設定します. サーボモータの最高速度での高速回転から非常にゆっくりとした回転まで, 任意に設定可能です.
- 目標回転角度 POS-A
  ポイント制御入力 XA が入力された時に目標回転角度を設定します. サーボモータの回転角度の範囲で

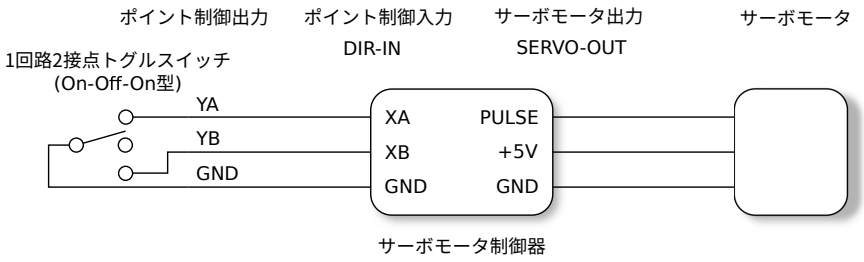


図 15.4: 接続例

任意に設定できます. 目標回転角度に到達すると LED-A が点灯して完了を知らせます.

- 目標回転角度 POS-B
  ポイント制御入力 XB が入力された時に目標回転角度を設定します. サーボモータの回転角度の範囲で任意に設定できます. 目標回転角度に到達すると LED-B が点灯して完了を知らせます.

目標回転角度 POS-A と POS-B の値はそれぞれ任意かつ独立に設定でき, 例えば以下のように移動方向を逆にすることも可能です.

- 例 1. 目標回転角度 POS-A を $-45°$ に, 目標回転角度 POS-B を $+45°$ にする.
- 例 2. 目標回転角度 POS-A を $+45°$ に, 目標回転角度 POS-B を $-45°$ にする.

スローアクションポイントマシンで利用する場合, この POS-A と POS-B による設定方法は大変に便利です.

- ポイントのトングレールの可動範囲を細かに設定できます.
- 定位と反位とを簡単に交換できます.
- 特殊な設定装置は不要で, 手作業で簡単に調整できます.

本装置には設定用の DIP スイッチ CONF があり, 設定可能な内容は以下の通りです.

- SW1: 可動角度範囲設定
  使用するサーボモータの可動角度範囲に合わせて設定します.
  - オフ
    可動範囲を $-45°$ から $+45°$ までの $90°$ にします.
  - オン
    可動範囲を $-90°$ から $+90°$ までの $180°$ にします.
- SW2: 移動速度範囲設定
  可変抵抗器 SPEED で調整可能な回転速度の範囲を選択します.
  - オフ
    高速で回転します. スローアクションポイントマシンとして使用する場合にはこの設定が良いでしょう.
  - オン
    低速で回転します. 各種のギミックでゆっくりと時間をかけて動作させる場合にはこの設定が良いでしょう.

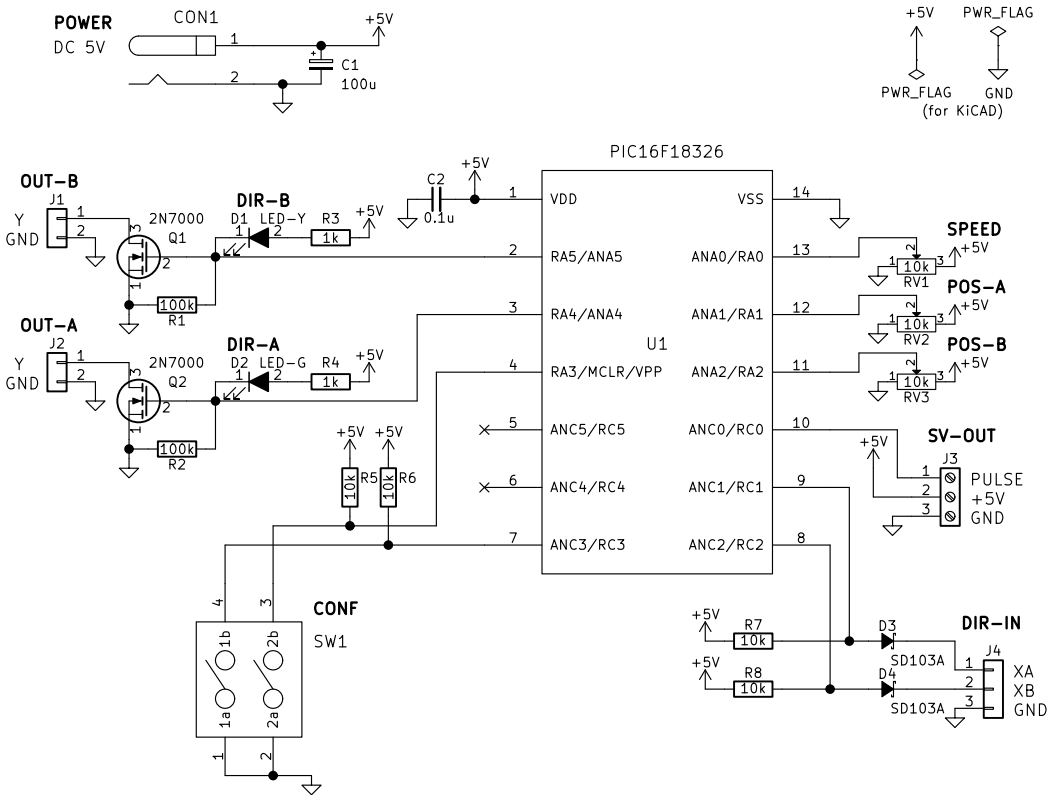


図 15.5: 【回路図】　サーボモータコントローラ

15.2　回路図

本装置の回路図を図 15.5 に, 部品リストを表 15.1 に, それぞれ示します. これまでに出てきた回路と大して変わりませんので特に説明は不要と思います.

表 15.1: 【部品表】　サーボモータコントローラ

記号	数量	部品名
U1	1	PIC 16F18326-I/P [I-11886]
Q1, Q2	2	FET 2N7000 [I-09723]
D3, D4	2	ショットキーバリアダイオード SD103A [I-04271]
D2	1	LED 緑 [I-06413]
D1	1	LED 黄 [I-11639]
C1	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C2	1	セラミックコンデンサ 0.1μF [P-00090]
R3, R4	2	抵抗 1kΩ　1/4W [R-25102]
R5, R6, R7, R8	4	抵抗 10kΩ　1/4W [R-25103]
R1, R2	2	抵抗 100kΩ　1/4W [R-25104]
RV1, RV2, RV3	3	可変抵抗器 10kΩ　B カーブ, 半固定型 [P-08012]

(つづく)

(つづき)

記号	数量	部品名
	1	IC ソケット 14 ピン [P-00028]
SW1	1	DIP SW 2P [P-08922]
J1, J2	2	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
J4	1	基板用コネクタ 3 ピン　JST XH コネクタ B3B-XH-A(LF) [C-12248]
J3	1	ターミナルブロック 3 ピン [P-01310]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか基板・配線材等

15.3　サーボモータの概要

ラジコンなどのホビー用に使用されるサーボモータの基本的な制御方法を説明します. サーボモータにはモータ軸があり, この回転角度を制御します. このモータ軸の回転角度は −90° から +90° までの 180° の範囲で稼働します. サーボモータによっては −45° から +45° までの 90° の範囲のものもあります. モータ軸の目標回転角度の指定はサーボモータへの制御信号で行います. モータ軸の回転角度が目標回転角度よりずれているとモータを駆動し, 目標回転角度に一致するとモータの駆動を停止します. この目標回転角度に一致させる閉ループ制御はサーボモータ内部で行われます.

サーボモータには電源の 5V と GND および制御信号の計 3 本の電線があります. 制御信号はパルス信号であり, そのパルス幅でモータ軸の回転角度を指定します. 制御信号の仕様は以下の通りで, 図 15.6 に図示します.

- パルス電圧
  オフの場合は 0V で, オンの場合は 4.8V です. 標準的な電圧は 4.8V ですが, 5V 以下であれば大丈夫です.
- パルス周期
  10m 秒から 20m 秒程度です. 周波数で言えば 50Hz から 100Hz 程度です.
- パルス幅
  パルス幅が狭いと回転角度は小さく, 広いと回転角度は大きくなります. 回転角度がニュートラルの場合のパルス幅は 1520μ 秒, パルス幅の可変範囲は ±500μ 秒なのが標準的な値です. この値はサーボモータで共通ではなく, ニュートラルの場合のパルス幅が 1500μ 秒のものや, パルス幅の可変範囲が ±600μ 秒のものもあります.

サーボモータ使用の際の注意は, 与えるパルス幅が回転角度の範囲を越えるとサーボモータが損傷することがあるという点です. 本章のファームウエアで全てのサーボモータを安全に使用できるとは限りませんから, 各自でサーボモータの仕様書を良く読んで十分に安全を確認して使用して下さい.

15.4　TMR1/3/5: 16 ビットタイマモジュール

PIC マイコンの内蔵周辺モジュール TMR1 (Timer1; タイマ 1) を使用して時間の計測を行います. TMR1 と同じ構造を持つタイマに TMR3, TMR5 があり, 各 PIC マイコンでの装備の有無は表 3.6 を参照して下さい. 以下では TMR1, TMR3, TMR5 を総称する場合には TMR1 系タイマと呼ぶことにします.

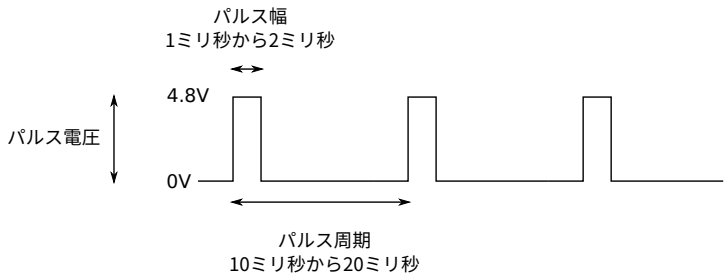


図 15.6: サーボモータの制御信号

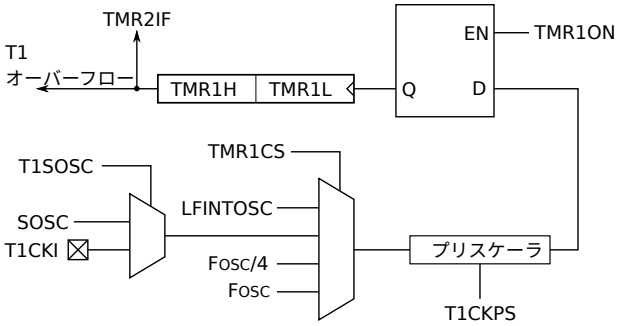


図 15.7: TMR1 の内部構造の概略 (ゲート制御機能は省略)

TMR1 は 16 ビットのタイマで, PIC マイコン内部のクロックや外部信号によりカウントアップを行います. TMR1 には多彩な機能がありますが, 本節では PIC マイコン内部のクロックでカウントアップを行う基本的な機能のみを概説します. 図 15.7 に TMR1 モジュールの内部構造の概略を示します. TMR1 のゲート制御機能はタイマのカウントアップの制御を外部信号で制御する機能で TMR1 の大きな特徴ですが, 本書ではその機能は使用しませんので図 15.7 と下記の説明では省略しています.

16F18326 での TMR1 系タイマで使用する主なレジスタやビットを以下に概説します. なお以下での記法 TMRx は, TMR1 や TMR3 などの TMR1 系タイマを表すものとします.

- `TxCON.TMRxCS`
  クロック源を指定します. マイコン内部のクロック源あるいは外部クロック源から選択します.
- `TxCON.TxCKPS`
  プリスケーラを指定します. 1/1, 1/2, 1/4, 1/8 の比率を指定可能で, 指定した比率に入力クロックの周波数を分周します.
- `TxCON.TxSOSC`
  外部クロック源を選択した場合に, ピンからクロックを入力するか, あるいはピンに発振子を接続するかを選択します.
- `TxCON.TMRxON`
  TMRx のオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.
- `TMRxH`
  タイマの上位 8 ビットです.
- `TMRxL`
  タイマの下位 8 ビットです.

- PIR1.TMR1IF
  TMR1 割り込み発生フラグです. オーバーフローが発生してタイマの値が 0 に戻ったときにこのビットが 1 になります. 他の TMR1 系タイマでも同等なフラグが装備されています.
- PIE1.TMR1IE
  TMR1 割り込み許可ビットです. 他の TMR1 系タイマでも同等なビットが装備されています.

15.5　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 15.2: 【ソースファイル一覧】 サーボモータ制御器

初	頁	番号	ファイル名	内容	行数
✓	250	15.3	ServoMotorCtr.c	主ソースコード	179
✓	249	15.2	servo.h	サーボモータ関係諸定義: ヘッダ	38
✓	253	15.4	servo-pulse.c	サーボモータパルス発生: 実装	62
	222	12.2	turnout.h	諸定義: ヘッダ	24
	223	12.3	instate.c	ポイント制御入力と動作: 実装	91
	129	6.1	incr-lin.h	線形増加アルゴリズム: ヘッダ	9
	130	6.2	incr-lin.c	線形増加アルゴリズム: 実装	78
✓	248	15.1	io.h	I/O ピン割付: ヘッダ	29
✓	254	15.5	mcr.h	MCR・周辺モジュール: ヘッダ	13
✓	255	15.6	mcr.c	MCR・周辺モジュール: 実装	104
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76
				計 712 行 (うち初出 425 行, 既出 287 行)	

本ファームウエアの骨子は図 15.8 に示す通り, 2 状態の有限状態機械です. ブレゼンハムアルゴリズムを用いることで, サーボモータの出力をゆっくりと回転させる機能を実現します.

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

ソースファイル番号 15.1: io.h

```
1  #if !defined(__IO_H__)
2  #define __IO_H__
3
4  /* サーボモータ移動速度設定のADCチャンネル: ANA0 */
5  #define ADC_CH_SPEED            0x00
6  /* サーボモータ位置A設定のADCチャンネル:   ANA1 */
7  #define ADC_CH_POS_A            0x01
8  /* サーボモータ位置B設定のADCチャンネル:   ANA2 */
9  #define ADC_CH_POS_B            0x02
10 /* 方向入力(位置A): RC1 */
11 #define DIR_IN_A()              (RC1 == 0)
12 /* 方向入力(位置B): RC2 */
13 #define DIR_IN_B()              (RC2 == 0)
14 /* サーボモータへのパルス出力: RC0 */
15 #define OUT_SERVO_PULSE_ON()    (LATC0 = 1)
16 #define OUT_SERVO_PULSE_OFF()   (LATC0 = 0)
```

```
main()
{
  int pos_current; /* サーボモータの現在の回転角度 */
  int pos_target_a;  /* 位置Aのサーボモータの目標角度 */
  int pos_target_b;  /* 位置Bのサーボモータの目標角度 */
  for (;;) {
    switch (state) {
    case STATE_A:  /* 位置A(定位) */
      if (pos_current != pos_target_a) { /*位置Aへ切り替え中*/
        pos_current = ブレゼンハムアルゴリズムで次の値を計算;
      }
      if (位置Bへの切り替え要求あり) {
        state = STATE_B;
      }
      break;
    case STATE_B:  /* 位置B(反位) */
      if (pos_current != pos_target_b) { /*位置Bへ切り替え中*/
        pos_current = ブレゼンハムアルゴリズムで次の値を計算;
      }
      if (位置Aへの切り替え要求あり) {
        state = STATE_A;
      }
      break;
    }
    サーボモータへpos_currentの値に従った制御パルスを送信;
  }
}
```

図 15.8: ファームウエア構造

```
/* 方向出力 (位置A, 位置B) : RA4, RA5
 *  - 開通:   OD出力=Z ... ポートL出力 (FET OFF)
 *  - 非開通: OD出力=L ... ポートH出力 (FET ON)   */
#define OUT_OD_DIR_A_Z()       (LATA4 = 0)
#define OUT_OD_DIR_A_LOW()     (LATA4 = 1)
#define OUT_OD_DIR_B_Z()       (LATA5 = 0)
#define OUT_OD_DIR_B_LOW()     (LATA5 = 1)
/* 可動範囲 [Off:90度, On:180度] : RC3 */
#define SW_DEGREE_RANGE_90()   (RC3 != 0)
/* 移動速度 [Off:NORMAL, On:SLOW]: RA3 */
#define SW_MOVE_SPEED_NORMAL() (RA3 != 0)

#endif /*__IO_H__*/
```

ソースファイル servo.h (サーボモータ関係諸定義: ヘッダ)

定数などの諸定義を行います.

ソースファイル番号 15.2: servo.h

```
#if !defined(__SERVO_H__)
#define __SERVO_H__

/* 設定SW: サーボモータ可動範囲 */
enum e_degree_range {
  DEGREE_RANGE_90 = 0,
  DEGREE_RANGE_180
```

```
8  };
9
10 /* 設定SW: サーボモータ動作速度 */
11 enum e_move_speed {
12   MOVE_SPEED_NORMAL = 0,
13   MOVE_SPEED_SLOW
14 };
15
16 /* サーボモータのパルス周期 20000 [uSec] = 20 [mSec] */
17 #define PULSE_CYCLE          20000
18
19 /* サーボモータのパルス幅: 中点*/
20 #define PULSE_WIDTH_NEUTRAL  1500 /* [uSec] */
21
22 /* サーボモータのパルス幅  最小/最大 */
23 #define PULSE_WIDTH_90_MIN  (PULSE_WIDTH_NEUTRAL-500)
24 #define PULSE_WIDTH_90_MAX  (PULSE_WIDTH_NEUTRAL+500)
25 #define PULSE_WIDTH_180_MIN (PULSE_WIDTH_NEUTRAL-1000)
26 #define PULSE_WIDTH_180_MAX (PULSE_WIDTH_NEUTRAL+1000)
27
28 /* 出力パルス幅を計算 (0 <= p <= 1023) [uSec]
29  *   範囲 90度: p + 1000  (= 1000 ... 2023 [uSec])
30  *   範囲180度: 2*p + 500 (= 500 ... 2546 [uSec]) */
31 #define PULSE_WIDTH_90(p)  ((p)        + PULSE_WIDTH_90_MIN)
32 #define PULSE_WIDTH_180(p) (((p) << 1) + PULSE_WIDTH_180_MIN)
33
34 extern enum e_degree_range sw_degree_range;
35 extern enum e_move_speed sw_move_speed;
36 extern void servo_pulse(uint16_t pos);
37
38 #endif /*__SERVO_H__*/
```

ソースファイル ServoMotorCtr.c (主ソースコード)

基本的な方針は以下の通りです.

- サーボモータの制御信号のパルスはソフトウエア制御で生成します.
- 定位, 反位, 回転速度の設定値を ADC で読み取ります.
- ポイント制御入力を読み, 定位か反位かによって目標回転角度を決定します.
- ブレゼンハムのアルゴリズムを使用して, 現在の回転角度から目標回転角度に向けてゆっくりと回転角度を変化してゆきます. 回転角度が目標回転角度に到達すれば回転角度の変化を終了して, その回転角度を維持します.
- 回転角度を変化させている間は方向出力は行わず, 回転角度が目標回転角度に到達すれば方向出力を行います.

ソースファイルは以下の通りです. 電源投入時にはポイント制御入力に応じた目標回転角度に即座に回転させます. 電源投入時の動作は第 12 章で示した表 12.8 も参照して下さい. 電源投入後にポイント制御入力の変化があれば, ブレゼンハムアルゴリズムにより目標回転角度に向けてゆっくりとサーボモータの回転角度を変化させてゆきます.

ソースファイル番号 15.3: ServoMotorCtr.c

```
1  /*********************************************
2   * サーボモーターコントローラ / PIC16F18326
3   * by Hirotsugu Kakugawa
4   *********************************************/
5  #include <xc.h>
6  #include <stdint.h>
7  #include "mcr.h"
8  #include "adc.h"
```

```
9  #include "io.h"
10 #include "servo.h"
11 #include "turnout.h"
12 #include "incr-lin.h"
13
14 /* ------------------------------------------------------ */
15
16 /* 移動速度の係数（大きいほど早くなる） */
17 #define SPEED_NORMAL   1024
18 #define SPEED_SLOW      128
19
20 /* 移動速度設定値の下限（この値以下だと即座に移動） */
21 #define SPEED_VR_THR     4
22
23 /* 設定SW */
24 enum e_degree_range sw_degree_range = DEGREE_RANGE_90;
25 enum e_move_speed sw_move_speed     = MOVE_SPEED_NORMAL;
26
27 /* サーボモータ移動速度 */
28 uint16_t speed = 0;
29 /* 回転角度: 位置A（定位） */
30 uint16_t pos_target_a = 0;
31 /* 回転角度: 位置B（反位） */
32 uint16_t pos_target_b = 0;
33 /* 速度定数 */
34 uint16_t spd_const = 0;
35
36 /* 有限状態機械 */
37 enum e_state {
38   STATE_A,  /* 位置A(定位) */
39   STATE_B,  /* 位置B(反位) */
40 };
41
42 static void param_read(void);
43
44 /*-------------------------------------------------------
45  * MAIN
46  *-------------------------------------------------------*/
47 void main(void)
48 {
49   /* システム初期化 */
50   System_Init();
51   /* ウオッチドッグタイマ: 256 [mSec] */
52   WDTCONbits.WDTPS  = 0x08; /* 0b01000 */
53   CLRWDT();
54   /* 開通方向出力: 切替動作中 */
55   OUT_OD_DIR_A_LOW(); /* 位置A(定位): 非開通 */
56   OUT_OD_DIR_B_LOW(); /* 位置B(反位): 非開通 */
57   /* 入力・パラメータ読込 */
58   param_read();
59   /* サーボモータ現在回転角度 */
60   uint16_t pos_current = 0;
61   /* 初期ポイント制御入力を読む */
62   enum e_movereq movereq = instate_init();
63   /* 制御状態とパラメータの初期化 */
64   enum e_state state = STATE_A;
65   switch (movereq) {
66   case MOVEREQ_NONE:
67   case MOVEREQ_A:
68     state = STATE_A;
69     pos_current = pos_target_a;
70     break;
71   case MOVEREQ_B:
72     state = STATE_B;
73     pos_current = pos_target_b;
74   }
75   incr_lin_init(pos_current);
76   /* 主ループ */
77   for (;;) {
78     CLRWDT();
```

```
79      /* 入力・パラメータ読込 */
80      param_read();
81      /* ポイント制御入力を読む */
82      movereq = instate_next();
83      /* サーボモータ制御 */
84      switch (state) {
85      case STATE_A: /* 位置A(定位) */
86        if (pos_current == pos_target_a) { /* 切替完了 */
87          /* 開通方向出力: 位置A(定位) */
88          OUT_OD_DIR_A_Z();   /* 位置A(定位): 開通 */
89          OUT_OD_DIR_B_LOW(); /* 位置B(反位): 非開通 */
90        } else {      /* 切替中 */
91          /* 現在位置を更新 */
92          if (speed == 0) { /* 即座 */
93            pos_current = pos_target_a;
94          } else {                  /* ゆっくり */
95            pos_current
96              = incr_lin_next(speed, spd_const, pos_target_a);
97          }
98        }
99        /* 切替要求 */
100       switch (movereq) {
101       case MOVEREQ_NONE:
102         break;
103       case MOVEREQ_A: /* 位置A(定位)への切替要求 */
104         break;
105       case MOVEREQ_B: /* 位置B(反位)への切替要求 */
106         /* 切替動作中 */
107         state = STATE_B;
108         incr_lin_init(pos_current);
109         OUT_OD_DIR_A_LOW(); /* 位置A(定位): 非開通 */
110         OUT_OD_DIR_B_LOW(); /* 位置B(反位): 非開通 */
111         break;
112       }
113       break;
114     case STATE_B: /* 位置B(反位) */
115       if (pos_current == pos_target_b) { /* 切替完了 */
116         /* 開通方向出力: 位置B(反位) */
117         OUT_OD_DIR_A_LOW(); /* 位置A(定位): 非開通 */
118         OUT_OD_DIR_B_Z();   /* 位置B(反位): 開通 */
119       } else {      /* 切替中 */
120         /* 現在位置を更新 */
121         if (speed == 0) { /* 即座 */
122           pos_current = pos_target_b;
123         } else {                  /* ゆっくり */
124           pos_current
125             = incr_lin_next(speed, spd_const, pos_target_b);
126         }
127       }
128       /* 切替要求 */
129       switch (movereq) {
130       case MOVEREQ_NONE:
131         break;
132       case MOVEREQ_A: /* 位置A(定位)への切替要求 */
133         /* 切替動作中 */
134         state = STATE_A;
135         incr_lin_init(pos_current);
136         OUT_OD_DIR_A_LOW(); /* 位置A(定位): 非開通 */
137         OUT_OD_DIR_B_LOW(); /* 位置B(反位): 非開通 */
138         break;
139       case MOVEREQ_B: /* 位置B(反位)への切替要求 */
140         break;
141       }
142       break;
143     }
144     /* サーボモータへパルスを送信 */
145     servo_pulse(pos_current);
146   }
```

```
147 }
148
149 static void param_read(void)
150 {
151   /* 可変抵抗器読取 */
152   adc_val_t adc_speed = ADC_Read(ADC_CH_SPEED);
153   if (adc_speed <= SPEED_VR_THR) {
154     speed = 0;
155   } else {
156     speed = adc_speed >> 2;
157   }
158   pos_target_a = ADC_Read(ADC_CH_POS_A);
159   pos_target_b = ADC_Read(ADC_CH_POS_B);
160   /* 設定スイッチ読取: 可動角度範囲 */
161   sw_degree_range = DEGREE_RANGE_90;
162   if (!SW_DEGREE_RANGE_90()) {
163     sw_degree_range = DEGREE_RANGE_180;
164   }
165   /* 設定スイッチ読取: 移動速度 */
166   sw_move_speed = MOVE_SPEED_NORMAL;
167   if (!SW_MOVE_SPEED_NORMAL()) {
168     sw_move_speed = MOVE_SPEED_SLOW;
169   }
170   switch (sw_move_speed) {
171   default:
172   case MOVE_SPEED_NORMAL:
173     spd_const = SPEED_NORMAL;
174     break;
175   case MOVE_SPEED_SLOW:
176     spd_const = SPEED_SLOW;
177     break;
178   }
179 }
```

このプログラム構造は第 12 章のポイント切替器と類似していますので特に説明は不要かと思います. 対象がずいぶん異っても, ファームウエアの基本構造はあまり変わらないことが見て取れます. これは有限状態機械によるモデル化を用いたことによるものであり, プログラムの設計と実装が容易になることが理解できると思います.

ソースファイル servo-pulse.c (サーボモータパルス発生: 実装)

関数 servo_pulse() により, サーボモータ用の制御パルスを 1 回だけ発生します.

1. パルス幅の時間をサーボモータの一般的な仕様に従って計算します.
2. パルス幅の時間だけ I/O ピンをオンにし, その時間が経過したら I/O ピンをオフにします.
3. パルス間隔を 20 ミリ秒とするために, 20 ミリ秒からパルス幅を引いた時間を計算し, その時間の経過を待ちます. これでひとつのパルスの生成は終わりです.

パルス幅の時間の経過を待機するのが関数 delay_use_by_tmr() で, 以下のようにタイマモジュール TMR1 を使用します.

1. パルス幅の時間に相当する TMR1 のカウント値を求め, TMR1 に設定します.
2. TMR1 を動作させます. 一定の早さでタイマのカウントアップが始まります.
3. TMR1 のオーバーフローフラグを監視し続けます. タイマはカウントアップされ続けてやがてオーバフローが生じます. そしてこの時に TMR1 のオーバーフローフラグ TMR1F の値は 1 に設定されますから, この検出により規定の時間の経過を検出できます.

ソースファイルは以下の通りです.

ソースファイル番号 15.4: servo-pulse.c

```
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "servo.h"
#include "io.h"

static void delay_usec_by_tmr(uint16_t t_usec);

/*-------------------------------------------------------
 * サーボモータへパルスを送信
 *   16ビットタイマを使用 (e.g., TMR1)
 *-------------------------------------------------------*/
void servo_pulse(uint16_t pos)
{
  /* サーボモータパルス幅の計算 [uSec] */
  uint16_t pulse_width = PULSE_WIDTH_NEUTRAL;
  switch(sw_degree_range) {
  case DEGREE_RANGE_90:   /* 可動範囲:  90度 */
    pulse_width = PULSE_WIDTH_90(pos);
    if        (pulse_width < PULSE_WIDTH_90_MIN) {
      pulse_width = PULSE_WIDTH_90_MIN;
    } else if (pulse_width > PULSE_WIDTH_90_MAX) {
      pulse_width = PULSE_WIDTH_90_MAX;
    }
    break;
  case DEGREE_RANGE_180:  /* 可動範囲: 180度 */
    pulse_width = PULSE_WIDTH_180(pos);
    if        (pulse_width < PULSE_WIDTH_180_MIN) {
      pulse_width = PULSE_WIDTH_180_MIN;
    } else if (pulse_width > PULSE_WIDTH_180_MAX) {
      pulse_width = PULSE_WIDTH_180_MAX;
    }
    break;
  }
  /* サーボモータへ１パルスを出力 */
  OUT_SERVO_PULSE_ON();
  delay_usec_by_tmr(pulse_width);
  OUT_SERVO_PULSE_OFF();
  delay_usec_by_tmr(PULSE_CYCLE - pulse_width);
}

/*-------------------------------------------------------
 * 遅延関数
 *   16ビットタイマを使用 (e.g., TMR1)
 *-------------------------------------------------------*/
static void delay_usec_by_tmr(uint16_t t_usec)
{
  /* 設定より: タイマ1周 (0x10000カウント)は 32.768 [mSec] */
  /* t_usec [uSec] の経過 = (2*t_usec) [カウント] の経過 */
  /* 20 [mSec] 間で 40000 [カウント] */
  /* 1 [uSec] で 2 [カウント] (= 0x100000 / 32768 = 2) */
  uint16_t tmr1_val = 0 - (t_usec << 1);
  TMR1_Stop();
  TMR1_Write(tmr1_val);
  TMR1_Start();
  for (;;) {
    if (TMR1_IsExpired()) {
      break;
    }
  }
  TMR1_Stop();
}
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言を行います.

ソースファイル番号 15.5: mcr.h

```
#if !defined(__MCR_H__)
#define __MCR_H__

#define _XTAL_FREQ 32000000  /* 32MHz */

extern void System_Init(void);
extern void   TMR1_Init(void);
extern void   TMR1_Start(void);
extern void   TMR1_Stop(void);
extern int8_t TMR1_IsExpired(void);
extern void   TMR1_Write(uint16_t t);

#endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化や, I/O ピンのデジタル入出力の設定を行います.

ソースファイル番号 15.6: mcr.c

```
/*-----------------------------------------------------------
 * MCR設定&周辺モジュール (PIC16F18326)
 *-----------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "adc.h"

/*-------------------------------------------*/
/* MCR初期化 */
void System_Init(void)
{
  /* 発振器 (FOSC=32MHz) */
  OSCCON1 = 0x00; /* HFINTOSC with 2x PLL */
  OSCCON3 = 0x00;
  OSCEN   = 0x00;
  OSCFRQ  = 0x06; /* 32MHz */
  OSCTUNE = 0x00;
  while (PLLR == 0) {
    ;
  }
  /* I/Oピン */
  LATA   = 0x00;
  LATC   = 0x00; /* (RC5,RC4) */
  ANSELA = 0x07; /* ANA2,ANA1,ANA0 */
  ANSELC = 0x00;
  TRISA  = 0x0F; /* RA3,RA2,RA1,RA0 */
  TRISC  = 0x3E; /* (RC5,RC4),RA3,RC2,RC1 */
  WPUA   = 0x00;
  WPUC   = 0x30; /* (RC5,RC4) */
  ODCONA = 0x00;
  ODCONC = 0x00;
  /* 周辺モジュール */
  ADC_Init();
  TMR1_Init();
}

/*-------------------------------------------*/
/* TMR1 */
/* Src: FOSC/4, Prescale 1:4, タイマー周期: 32.768 [mSec] */
void TMR1_Init(void)
{
  T1CONbits.TMR1ON = 0; /* Disabled */
  T1CONbits.T1CKPS = 2; /* Prescale: 1:4 */
  T1CONbits.TMR1CS = 0; /* FOSC/4 */
  T1GCON = 0x00;
  TMR1H = 0;
  TMR1L = 0;
  TMR1IF = 0;
```

```
50    TMR1IE = 0;
51  }
52
53  void TMR1_Start(void)
54  {
55    T1CONbits.TMR1ON = 1; /* Enabled */
56  }
57
58  void TMR1_Stop(void)
59  {
60    T1CONbits.TMR1ON = 0; /* Disabled */
61  }
62
63  int8_t TMR1_IsExpired(void)
64  {
65    int8_t expired = TMR1IF;
66    if (expired == 1) {
67      TMR1IF = 0;
68    }
69    return expired;
70  }
71
72  void TMR1_Write(uint16_t t)
73  {
74    uint8_t oon = T1CONbits.TMR1ON;
75    T1CONbits.TMR1ON = 0; /* Disabled */
76    TMR1IF = 0;
77    TMR1H = (t >> 8);
78    TMR1L = (t & 0xFF);
79    T1CONbits.TMR1ON = oon; /* Go back */
80  }
81
82  /*-----------------------------------------------*/
83  /* CONFIG1 */
84  #pragma config FEXTOSC  = OFF
85  #pragma config RSTOSC   = HFINT32
86  #pragma config CLKOUTEN = OFF
87  #pragma config CSWEN    = ON
88  #pragma config FCMEN    = ON
89  /* CONFIG2 */
90  #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
91  #pragma config PWRTE    = OFF
92  #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
93  #pragma config LPBOREN  = OFF
94  #pragma config BOREN    = ON
95  #pragma config BORV     = LOW
96  #pragma config PPS1WAY  = ON
97  #pragma config STVREN   = ON
98  #pragma config DEBUG    = OFF
99  /* CONFIG3 */
100 #pragma config WRT      = OFF
101 #pragma config LVP      = OFF /* 低電圧書込不可 */
102 /* CONFIG4 */
103 #pragma config CP       = OFF
104 #pragma config CPD      = OFF
```

課題

1. 本章で紹介した回路では, 回転軸の停止する位置を基板上の 2 つの可変抵抗器で設定していました. しかし本装置をレイアウトに設置した後に, 振動により可変抵抗器の設定が次第にずれてくるといった問題の発生が予想されます. しかし 2 つの可変抵抗器による手作業での設定の利便性は捨てがたいものがあります. その両立を行う新たな方法を考案して下さい.
2. 連続回転サーボ (ローテーションサーボ) の制御を行う装置を設計・製作して下さい. 仕様は鉄道模型での使用を考えて決定して下さい.

第 IV 部

鉄道模型を楽しむアクセサリ

第 16 章

車両センサ

本章では各種の車両センサの紹介をします．車両センサは走行速度測定器 (第 17 章), 色灯式信号器 (第 18 章), 自動往復運転システム (第 29 章) およびプログラム型自動運転システム (第 30 章) で使用します．本章で紹介する回路の特徴は以下の通りです．

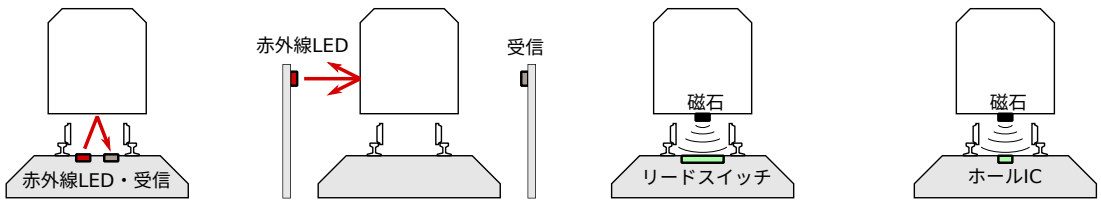
- 様々な種類の車両センサ
  設置の容易さ, 検出の精度, 価格など, さまざまな種類の車両センサを紹介します．
- 統一的な車両センサインターフェース仕様
  本章では車両センサに統一的な車両センサインターフェース仕様を定めます．車両センサインターフェース仕様に従っている車両センサならレイアウトや模型車両の都合に合ったもの任意に選んで使用できます．
- アナログ制御・デジタル制御のどちらにも対応
  線路とは電気的に絶縁していますから DCC システムでも使用可能です．

本章で学ぶ内容は以下の通りです．

- 光変調型フォト IC と赤外線 LED の使用方法
- リードスイッチの使用方法
- ホール IC の使用方法

本書では全体を通して統一した仕様の車両センサインターフェースを使用します．そこで本章では仕様を定めて以下の 4 種の車両センサを紹介します．

- 変調赤外線反射検出型車両センサ (図 16.1a. 第 16.2 節, 260 ページ)
- 変調赤外線遮断検出型車両センサ (図 16.1b. 第 16.3 節, 262 ページ)
- 磁気リードスイッチ型車両センサ (図 16.1c. 第 16.4 節, 263 ページ)
- 磁気ホール IC 型車両センサ (図 16.1d. 第 16.5 節, 264 ページ)



(a) 変調赤外線反射検出型　**(b)** 変調赤外線遮断検出型　**(c)** 磁気リードスイッチ型　**(d)** 磁気ホール IC 型

図 16.1: 本書で紹介する車両センサ 4 種の検出方式

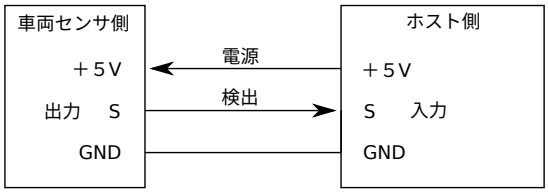


図 16.2: 車両センサインターフェース仕様の概念図

なお車両センサごとに検出方法が違いますから, 模型車両には検出方法に合致した加工が必要となる場合があります. また車両センサインターフェース仕様に従っていれば, 上記以外のセンサに置き換えて使用できます[*1].

鉄道模型における車両センサは大別して 2 通りあります.

- 在線検知型
  線路のある一定の区間に模型車両が存在するか否かを検出する方式です. これは閉塞区間を実現するために便利な方式です. 実際の鉄道では輪軸が左右のレールを電気的に短絡することを利用して, 電流の流れを検出することで在線を検出します. 鉄道模型ではフィーダと線路の間に電流センサを入れておき, 流れる電流の検出で在線の有無を検出するのが一般的な手法のようです. ただし電流が流れないと検出できませんから, モータ, ヘッドライト, テールライト, 室内灯, あるいはダミーの抵抗器などを装備した, 電流を消費する模型車両しか検出できません.
- 通過検知型
  線路のある一点上に模型車両が存在するか否かを検出する方式です. これは模型車両がある一点を通過したことを検出するに便利な方式です. 実際の鉄道では定位置停止装置などで使用されており, ホームドアに合わせた正確な位置での停車を実現しています. 鉄道模型では正確な位置で停車をする自動運転において便利です. この方式では電流を消費しない模型車両でも検出が可能です.

このように 2 通りの種別がありますが, 本章では後者の通過検知型の車両センサを紹介します.

16.1　車両センサインターフェイス仕様

本書で使用する車両センサインターフェイス仕様を説明しますす.
検出に関して関係するのはセンサ本体と信号を受信する側の二者です.

- 車両センサ
  模型車両の検出結果を電気信号により出力します. 車輌センサから 5V 電源の供給を受けて動作します.
- ホスト
  車両センサからの検出結果を電気信号により入力します. 車輌センサに 5V 電源の供給を行います.

車両センサとホストの間は図 16.2 に示すように 3 本の電線を使用します.

- +5V
  ホストから車両センサへ供給する電源で, 電流は最大 20mA とします.
- S

[*1] TOMIX 社の TCS センサレールもコネクタ部分を工夫すれば車両センサインターフェース仕様に合致するようですので, 本書の中の装置での使用が可能と思われます. ただしメーカは仕様の公開をしていませんから, そのような使用はメーカ保証外となります.

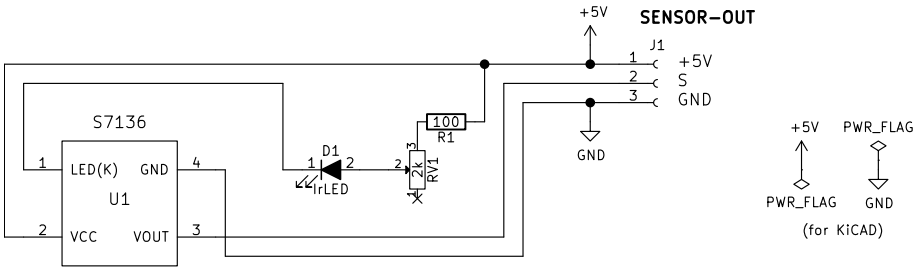


図 16.3: 【回路図】 車両センサ (変調赤外線反射検出型)

車両センサの出力で, 検出時には L (GND 電位), 非検出時には HiZ とし, 電流は最大 20mA とします. 機械的なスイッチのオン・オフのほかに, FET のオープンドレイン出力やトランジスタのオープンコレクタ出力をセンサ出力にできます.

- GND
  基準電位です.

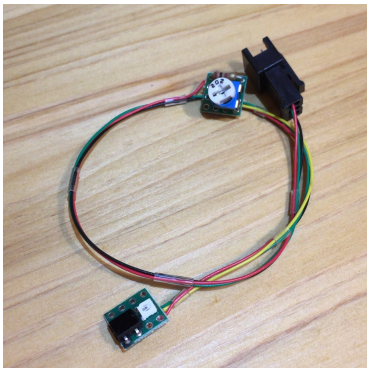
本章では「車両センサ」と呼んではいますが, 必ずしも検出対象が模型車両である必要はなく, また使用デバイスがセンサである必要もありません. 電源を除けばアクセサリ制御出力の仕様と互換性がありますので, アクセサリ制御出力を有する装置を車両センサの代わりとして使用できます. このようにインターフェース規約を統一的に定めることで, 様々な装置を柔軟に接続して幅広い応用が可能です.

本章で紹介する各車両センサの長所と短所をまとめます. 各自の状況に応じて適切な車両センサを選んでください. 図 16.1 に各検出方式の概略図を示しますので設置の参考にして下さい.

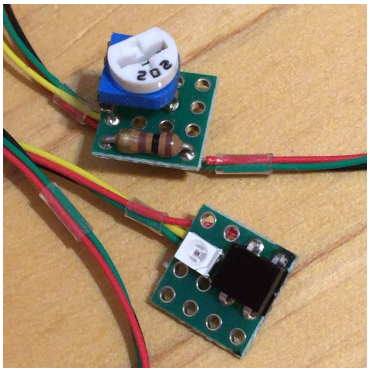
- 変調赤外線反射検出型車両センサ
  長所: 外乱光に強く, 検出が確実. 外観が地上子に似ており不自然な感じはあまりない.
  短所: 部品代が高い. センサ回路の組み立てに手間がかかる. 模型車両の床下に反射材の貼付が必要.
- 変調赤外線遮断検出型車両センサ
  長所: 外乱光に強く, 検出が確実. 模型車両の加工は不要.
  短所: 部品代が高い. センサ回路の組み立てに手間がかかる. センサ設置を隠すストラクチャが必要.
- 磁気リードスイッチ型車両センサ
  長所: 部品代が非常に安い. センサ回路の組み立てが楽. 線路下にセンサを隠せる.
  短所: 模型車両の床下に磁石の貼付が必要. 磁力の影響で道床内蔵型ポイントマシン上の走行がぎこちない. 反応速度が遅いので動作の確実さが若干心配.
- 磁気ホール IC 型車両センサ
  長所: 部品代が安い. 線路下にセンサを隠せる.
  短所: 模型車両の床下に磁石の貼付が必要. 磁力の影響で道床内蔵型ポイントマシン上の走行がぎこちない.

16.2　変調赤外線反射検出型車両センサ

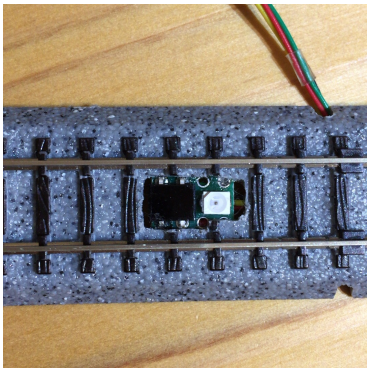
浜松フォトニクス社の光変調型フォト ICS7136 および S6809 を使用した変調赤外線反射検出型車両センサを紹介します. 光変調型フォト IC S7136 および S6809 は独自に変調をかけた赤外線を検出する IC です. この IC の大きな特徴は外乱光の下でも誤検出が少ないという点にあります. IC は変調を行って赤外線 LED の点灯を駆動して, その変調に合致した受光があれば検出と判断することで精度の高い検出を実現しています.



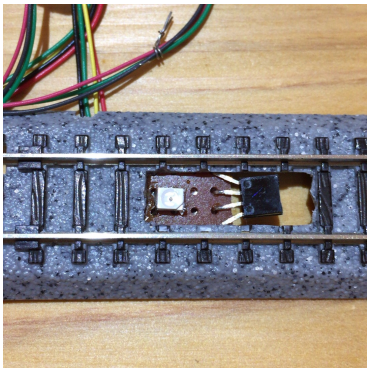
(a) 車両センサの全体像 (S7136)



(b) 車両センサの拡大図 (S7136)



(c) 線路への組み込み例 (S7136)



(d) 線路への組み込み例 (S6809)

図 16.4: 変調赤外線反射検出型車両センサの試作例 (S7136/S6809)

この車両センサでは電流を消費しない付随車や貨車でも検出が可能です. 模型車両の床下が黒色の場合には赤外線が十分に反射しない場合には, 床下に白いビニールテープなどの反射材の貼り付けを行って下さい.

表 16.1: 【部品表】 車両センサ (変調赤外線反射検出型)

記号	数量	部品名
U1	1	光変調型フォト IC S7136　又は S6809 [I-02499]
D1	1	赤外線 LED [I-06427]
R1	1	抵抗 100Ω　1/4W [R-25101]
RV1	1	可変抵抗器 2kΩ　B カーブ, 半固定型 [P-03273]
J1	1	コネクタハウジング 3 ピン　JST ZH コネクタ ZHR-3 [C-14150]
	3	コネクタハウジング用コンタクト　JST ZH コネクタ SZH-002T-P0.5 [C-14154]
		反射材　適宜
		ほか接続用ケーブル・基板・配線材等

回路図を図 16.3 に, 部品表を表 16.1 に, それぞれ示します. 赤外線 LED は砲弾型でも表面実装型でも構いません. 可変抵抗器は赤外線 LED の輝度を調整に使用します. 輝度を高くしすぎると赤外線が直接的に入光して誤検出となります. なお車両センサには接続を容易にするコネクタを取り付けると良いでしょう. 車両センサ

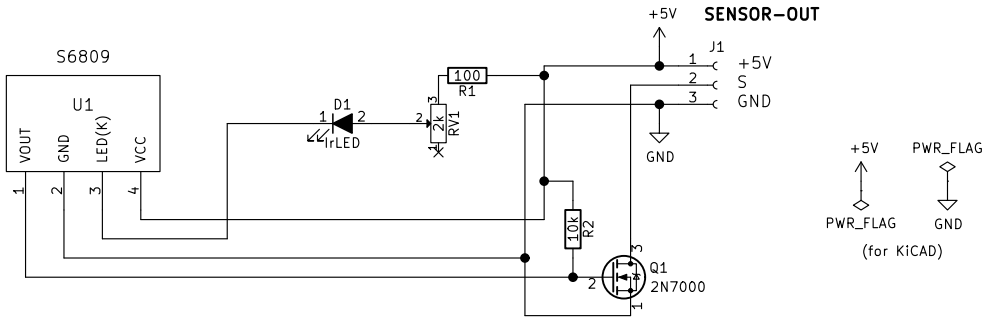


図 16.5: 【回路図】 車両センサ (変調赤外線遮断検出型)

のケーブルは長く引き伸ばして使うことになりますので, ケーブルは任意に延長可能にしておくと便利です. 具体的には中継用コネクタを用いた延長ケーブルを作成します. コネクタメーカのカタログを見ると電線対電線という種別のコネクタがありますので, そのオスとメスを組み合わせます. なお中継用コネクタは部品リストには含めていませんので注意して下さい.

図 16.4 が製作例と KATO ユニトラック (N ゲージ) 線路への組み込み例です. 細かなハンダ付けと線路本体や線路下に加工が必要なのでレイアウトへの組み込みは少々面倒です. しかし上手に組み込むと定位置停止装置の地上子のようにも見えますので, 各自で工夫して工作して下さい.

16.3　変調赤外線遮断型車両センサ

浜松フォトニクス社の光変調型フォト ICS7136 および S6809 を使用した変調赤外線遮断型検出型車両センサを紹介します. 模型車両が通過する際に車体が赤外線を遮るように設置をし, 受光の遮断を検出して模型車両の検出を行います.

この車両センサでは赤外線の遮断の有無で模型車両の検出を行うため模型車両への加工は不要です. しかし模型車両の車体側面が著しく低い長物車などで積荷がない場合には, 赤外線の遮断が十分にできないかもしれません.

回路図を図 16.5 に, 部品表を表 16.2 に, それぞれ示します. 赤外線 LED は砲弾型でも表面実装型でも構いません. 接続を容易にするコネクタを取り付けると良いでしょう. センサのケーブルは長く引き伸ばして使うことになりますので延長ケーブルも作成しておきます.

表 16.2: 【部品表】 車両センサ (変調赤外線遮断検出型)

記号	数量	部品名
U1	1	光変調型フォト IC S6809　又は S7136 [I-02499]
D1	1	赤外線 LED　[I-04313]
Q1	1	FET 2N7000　[I-09723]
R1	1	抵抗 100Ω　1/4W [R-25101]
R2	1	抵抗 10kΩ　1/4W [R-25103]
RV1	1	可変抵抗器 2kΩ　B カーブ, 半固定型 [P-03273]
J1	1	コネクタハウジング 3 ピン　JST ZH コネクタ ZHR-3 [C-14150]
	3	コネクタハウジング用コンタクト　JST ZH コネクタ SZH-002T-P0.5 [C-14154]
		ほか基板・配線材等

図 16.6 が製作例と KATO ユニトラック (N ゲージ) 線路への組み込み例です. この組み込み例のように線路の左右から赤外線 LED と光変調型フォト IC を向かい合わせに設置して, 模型車両の通過時には赤外線が遮光されるようにします. 線路本体や線路下に加工は不要でレイアウトへの組み込みは楽ですが, 車両センサをストラクチャの中に入れて目立たなくする工夫が必要です.

16.4　磁気リードスイッチ型車両センサ

磁力に反応してオン・オフするスイッチであるリードスイッチを使用した磁気リードスイッチ型車両センサを紹介します. 模型車両がセンサ上を通過する際にリードスイッチがオンになることで模型車両の検出を行います.

この車両センサでは模型車両の床下に磁石を取り付ける必要があります. しかし模型車両によっては磁石を取り付けるに十分な空間がない場合もありますので事前に確認をして下さい.

表 16.3: 【部品表】 車両センサ (磁気リードスイッチ型)

記号	数量	部品名
SW1	1	リードスイッチ FR2S1015　[P04400]
J1	1	コネクタハウジング 3 ピン　JST ZH コネクタ ZHR-3 [C-14150]
	3	コネクタハウジング用コンタクト　JST ZH コネクタ SZH-002T-P0.5 [C-14154]
		磁石　適宜
		ほか接続用ケーブル・基板・配線材等

回路図を図 16.7 に, 部品表を表 16.3 に, それぞれ示します. 磁気リードスイッチはどれもほぼ同じ働きを持っていますから, 部品表に記載のもの以外を使用してもまず大丈夫でしょう.

図 16.8 に線路への組み込み例と模型車両の床下への磁石の取り付け例を示します. 線路上にリードスイッチを取り付けていますが, 磁力が線路下に十分に届くようでしたら線路下への設置もできます. 取り付け例では

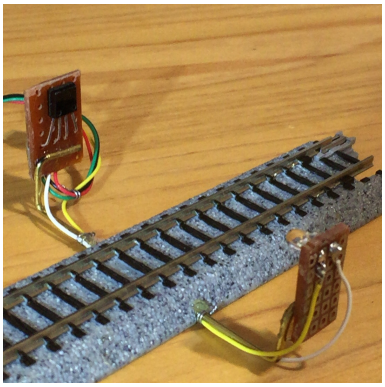


図 16.6: 赤外線遮断検出車両センサ (S6809) の試作例

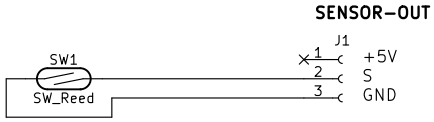
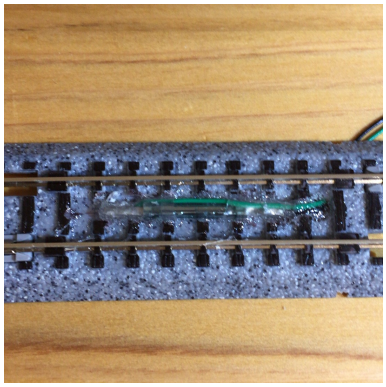
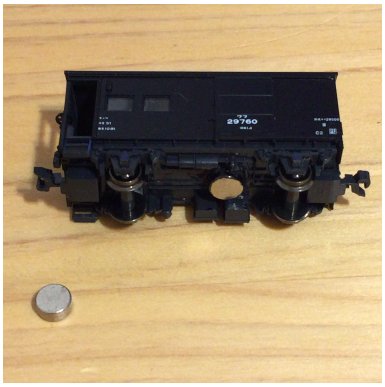


図 16.7: 車両センサ (磁気リードスイッチ型)



(a) 線路への組み込み例



(b) ネオジム磁石と車両への取り付け例

図 16.8: 磁気リードスイッチ型車両センサ

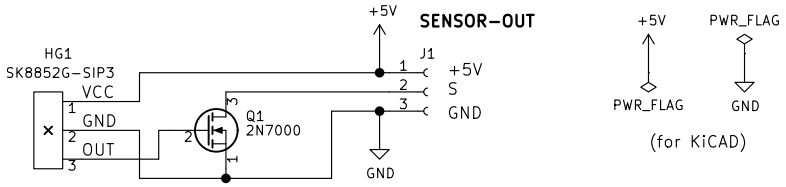


図 16.9: 【回路図】　車両センサ (磁気ホール IC 型)

KATO N ゲージ ワフ 29500 に強力で超小型のネオジム磁石を貼り付けた様子を示しています. 磁石の銀色は目立ちますので, 磁石の横面に黒色マジックを塗っています.

この方法の欠点は, 道床内蔵型ポイントマシンの上を模型車両が走行する時に床下に取り付けた磁石と道床内蔵型ポイントマシン内の金属とが引き合い, 模型車両の動きがぎこちなくなることです. 磁力の弱い磁石を使えば良いのですが, 高速走行する模型車両にリードスイッチが反応するか十分な事前確認が必要です.

16.5　ホール IC 型車両センサ

磁力に反応するホール効果によるセンサであるホール IC を使用した磁気ホール IC 型車両センサを紹介します. 模型車両がセンサ上を通過する際にホール IC が磁力を検出することで模型車両の検出を行います. この車両センサではリードスイッチの場合と同様に模型車両の床下に磁石を取り付ける必要があります.

表 16.4: 【部品表】　車両センサ (磁気ホール IC 型)

記号	数量	部品名
HG1	1	ホール IC SK8852G SIP3　[I-11029]
Q1	1	FET 2N7000　[I-09723]
J1	1	コネクタハウジング 3 ピン　JST ZH コネクタ ZHR-3 [C-14150]
	3	コネクタハウジング用コンタクト　JST ZH コネクタ SZH-002T-P0.5 [C-14154]
		磁石　適宜
		ほか基板・配線材等

回路図を図 16.9 に, 部品表を表 16.4 に, それぞれ示します. 使用しているホール IC SK8852 は磁気の S 極・

N 極どちらでも検出を行う両極性のもので, 磁石の向きを気にせずに模型車両に取り付けができて便利です. 単極性のものでは検出する極性に合わせて磁石を取り付ける必要があるので面倒です.

図 16.10 が製作と線路への組み込みの例です. ネオジム磁石は磁力が強力なので, 線路の裏面にホール IC を張り付けるだけ十分に磁気を検出できて線路上面の加工は不要でした.

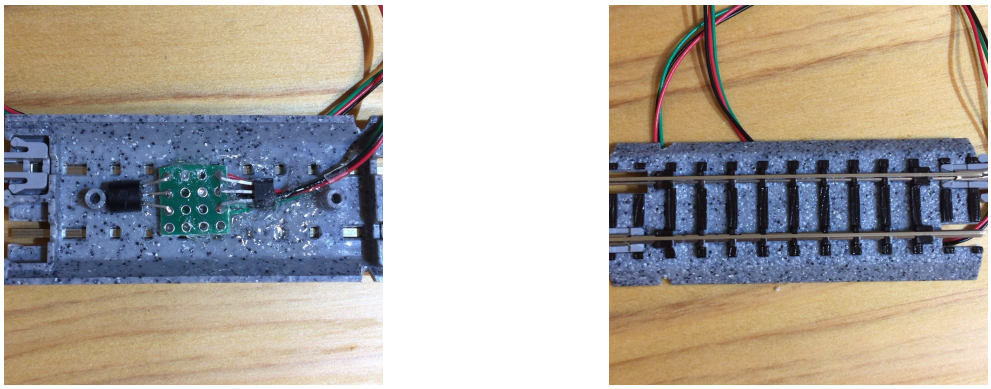
16.6　センサテスタ

組み立てた車両センサの動作確認や LED 輝度の調整の際に便利なテスタを紹介します. 回路図を図 16.11 に, 部品表を表 16.5 に, それぞれ示します. 車両の検出時に LED が光りますので視覚的にセンサの動作確認が行えます.

表 16.5: 【部品表】 車両センサ用テスタ

記号	数量	部品名
D1	1	ショットキーバリアダイオード SD103A [I-04271]
D2	1	LED 赤 [I-04780]
CRD1	1	CRD 10mA [I-00185]
J1	1	コネクタハウジング 3 ピン　JST ZH コネクタ ZHR-3 [C-14150]
	3	コネクタハウジング用コンタクト　JST ZH コネクタ SZH-002T-P0.5 [C-14154]
CON1	1	DC ジャック [C-09408]

(つづく)



(a) 試作例と線路への組み込み例 (裏面)　　**(b)** 線路上面は加工不要

図 16.10: 磁気ホール IC 型車両センサの試作例

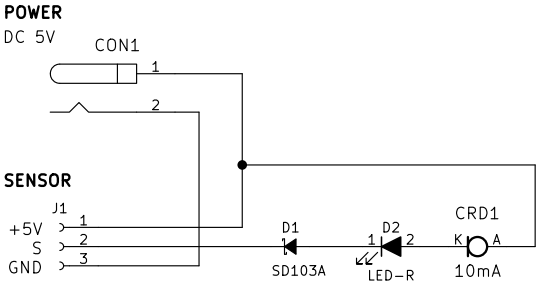


図 16.11: 【回路図】 車両センサ用テスタ

(つづき)

記号	数量	部品名
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか基板・配線材等

課題

1. CdS セルとはどのような電子部品かを調べて説明して下さい. 車両センサに CdS セルを使用した場合の長所と短所を述べて下さい.
2. フォトリフレクタとはどのような電子部品かを調べて説明して下さい. 車両センサにフォトリフレクタを使用した場合の長所と短所を述べて下さい.
3. 電流センサを用いた在線検知型のセンサを設計・製作して下さい.

第 17 章

走行速度測定器

本章では模型車両の走行速度を測定する装置を紹介します. 従来の同様な装置では線路電圧を測定して時速に換算する方式が多数のようです. しかし電圧と走行速度との間の比例係数は模型車両ごとに違いますから, 模型車両ごとに比例係数の調整が必要です. 本章で紹介する装置ではそのような調整が不要で, 走行時間と走行距離から本物の走行速度を計算して表示します. ただし車両センサを設置している場所でしか測定できないのが短所です.

本章で紹介する回路とファームウエアの特色は以下の通りです.

- スケールスピードの表示
  2 つの車両センサの間の走行時間から, 走行速度を km/h の単位で算出して表示します. 実際の走行速度の計測に基づいた表示が本装置の最大の長所です.
- 2 つの車両センサ間の距離を任意に設定可能
  設置後に車両センサ間の距離を物差しで測り, その距離を本装置の可変抵抗器で設定します. この機能により車両センサの設置場所の自由度が上がります.
- 様々な縮尺の鉄道模型に対応
  Z スケールから 2 番までの各種の縮尺を DIP スイッチを切り替えて設定できます.

本章で学ぶ内容は以下の通りです.

- 7 セグメント LED による文字表示
- 7 セグメント LED のダイナミック点灯法
- タイマ割り込みによる周期タスク
- タイマによる経過時間の測定

実際の走行速度を表示する仕組みは以下の通りです.

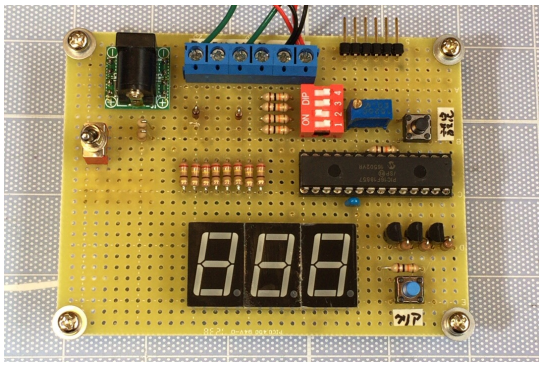


図 17.1: 走行速度測定器

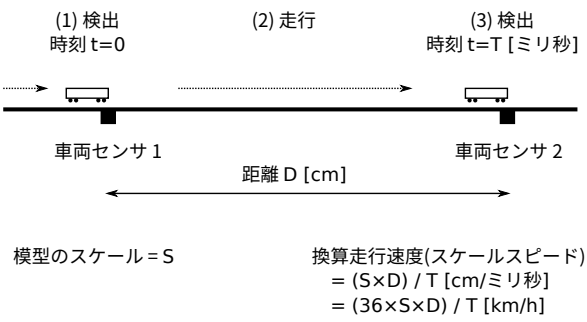


図 17.2: 走行速度の測定・算出方法の原理

```
1  void main(void)
2  {
3    if (設定ボタンが押されている) {
4      設定モードに入る;
5    }
6    for(;;) {
7      2つの車両センサの内のいずれかでの検出を待機;
8      時間の計測を開始;
9      他方の車両センサが検出すれば時間の計測を終了;
10     走行速度を計算;
11     7セグメントLEDに走行速度を表示;
12   }
13 }
```

図 17.3: ファームウエア構造 (概要)

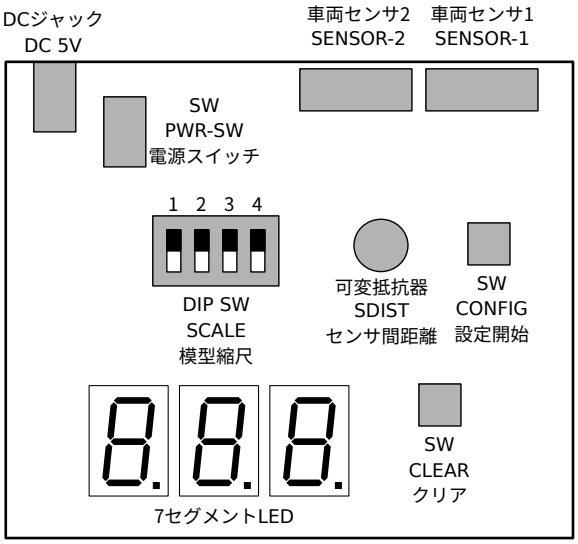


図 17.4: 操作表示部品の配置例と名称

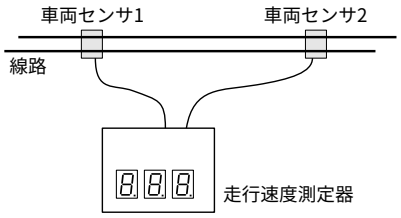


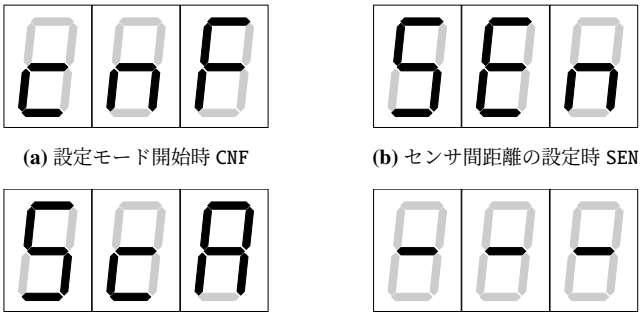
図 17.5: 接続方法

- 2 つの車両センサを設置します.
- 可変抵抗器で 2 つの車両センサの間の距離を設定します.
- DIP スイッチで模型の縮尺を設定します.
- 模型車両が走行した時に 2 つの車両センサの通過に要する走行時間を計測します.
- 模型の縮尺, 2 センサ間の距離, 走行時間の 3 つから実車両での走行速度 (スケールスピード) に換算します. 表示は km/h の単位で, その計算法を図 17.2 に示します.

本章のファームウエアの大まかな構造を図 17.3 に示します. 2 つの車両センサでの検出時間から走行速度を求めて表示する仕組みです. なお 7 セグメント LED での実際の表示にはダイナミック点灯と呼ばれる手法を使い, これにはタイマ割り込みを用います.

17.1　使用方法

本装置の使用方法を説明します. 以下の説明で使用する部品名称は図 17.4 に配置例とともに示しています.



(a) 設定モード開始時 CNF　**(b)** センサ間距離の設定時 SEN

(c) 模型縮尺の設定時 SCA　**(d)** 測定準備完了時

図 17.6: 設定モード時の表示

設置と初期設定

本装置を設置するときに行う設定作業は以下の通りです.

1. 電源スイッチ PWR-SW をオフにしておき, 5V の AC アダプタを DC ジャックに差し込みます. **間違って 12V 出力の AC アダプタを使用すると回路が破損する場合がありますので, 十分に注意して下さい.**
2. 図 17.5 に示すように 2 つの車両センサを設置し, 本装置に接続します. 車両センサ 1 と 2 の左右は関係なく走行方向に関わらず走行速度は測定できます. なお車両センサ間の距離が小さすぎると測定精度が低くなります. 例えば N ゲージの場合では 30cm 程度で良いでしょう. 2 つのセンサの間の距離を物差しなどを使ってセンチメートル単位で測り, メモしておきます.
3. 設定モードでの起動
   設定用の押しボタンスイッチ CONF を押しながら電源スイッチ PWR-SW をオンにします. すると設定モードに移行して, 7 セグメント LED には図 17.6a に示すように “CNF” (configuration [設定] の意) がしばらく表示されます.
4. 車両センサ間の距離の設定
   7 セグメント LED には図 17.6b に示すように “SEN” (sensor distance [センサ間距離] の意) がしばらく表示され, センサ間距離の設定に入ります. 可変抵抗器 SDIST を調整して, 7 セグメント LED の表示が 2 センサ間の距離になるようにします. なお表示の単位はセンチメートルです. 可変抵抗器 SDIST を調整が済めば設定用の押しボタンスイッチ CONF を押して次に進みます.
5. 模型縮尺の設定
   7 セグメント LED には図 17.6c に示すように “SCA” (scale [模型縮尺] の意) がしばらく表示され, 模型縮尺の設定に入ります. 16 種類の縮尺があらかじめ用意されていますので, DIP スイッチ SCALE にある 4 つのスイッチのオン・オフで設定します. DIP スイッチ設定を表 17.1 に示します. DIP スイッチの設定が済めば, 設定用の押しボタンスイッチ CONF を押して次に進みます.
6. 7 セグメント LED に設定内容が確認のために表示されます.

以上で設定が完了です. 設定を変更する場合は再度この手順を行います.

走行速度の測定

1. 電源スイッチ PWR-SW をオフにしておき, 5V の AC アダプタを DC ジャックに差し込みます.
2. 電源スイッチ PWR-SW をオンにします. 設定内容が 7 セグメント LED に表示されますので確認しま

表 17.1: DIP スイッチによる模型縮尺の設定

DIP スイッチ				縮尺	呼称
1	2	3	4		
OFF	OFF	OFF	OFF	1/220	Z スケール
OFF	OFF	OFF	ON	1/160	N スケール (欧米)
OFF	OFF	ON	OFF	1/150	N ゲージ (日本)
OFF	OFF	ON	ON	1/148	N ゲージ (英)
OFF	ON	OFF	OFF	1/120	TT スケール
OFF	ON	OFF	ON	1/87.1	H0 スケール (欧)
OFF	ON	ON	OFF	1/87	HO スケール (米)
OFF	ON	ON	ON	1/80	16 番 (日)
ON	OFF	OFF	OFF	1/76	OO ゲージ (英)
ON	OFF	OFF	ON	1/64	S スケール
ON	OFF	ON	OFF	1/48	0 番, O スケール (米)
ON	OFF	ON	ON	1/45	0 番 (日欧), OJ ゲージ (日)
ON	ON	OFF	OFF	1/43.5	0 番, O スケール (英)
ON	ON	OFF	ON	1/32	1 番 (欧米)
ON	ON	ON	OFF	1/30.5	1 番 (英)
ON	ON	ON	ON	1/22.5	2 番

す. その後に, セグメント LED には図 17.6d に示すように “---” が表示されて測定可能になります.

3. 模型列車を走らせます. 2 つの車両センサ間を通り抜けると 7 セグメント LED に走行速度が表示されます. 表示の数値の単位は km/h です. 測定を中断するには押しボタンスイッチ CLEAR を押します.
4. 計測が終わった後に車両センサによる検出が 10 秒以上なければ, 次の走行速度の測定が可能になります.

無検出の状態がある程度の時間経過してから次の測定を始めるのは, 長大編成の模型列車が車両センサを走り抜ける前に次の計測が始まらないようにするためです.

17.2　回路図

走行速度測定器の回路図を図 17.7 に, 部品リストを表 17.2 に, それぞれ示します. センサ間の距離を設定する可変抵抗器 SDIST には精密な設定に便利な多回転半固定型の可変抵抗器を使用します. なお部品リストには車両センサと本装置を接続するためのコネクタハウジング, コンタクト, 電線等は記載していませんので, 各自の接続方法に合わせてケーブルを作成して下さい.

表 17.2: 【部品表】 走行速度測定器

記号	数量	部品名
U1	1	PIC 16F18857-I/SP [I-11888]
Q1, Q2, Q3	3	FET 2N7000 [I-09723]
7SEG1, 7SEG2, 7SEG3	3	7 セグメント LED (カソードコモン) [I-04115]

(つづく)

(つづき)

記号	数量	部品名
D1, D2	2	ショットキーバリアダイオード SD103A [I-04271]
C1	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C2	1	セラミックコンデンサ 0.1μF [P-00090]
R6, R7, R8, R9, R10, R11, R12, R13	8	抵抗 240Ω　1/4W [R-25241]
R1, R2, R3, R4, R5, R14, R15, R16	8	抵抗 10kΩ　1/4W [R-25103]
R17, R18, R19	3	抵抗 100kΩ　1/4W [R-25104]
RV1	1	可変抵抗器 10kΩ　多回転半固定型 [P-00975]
	1	IC ソケット 28 ピン [P-01339]
SW1	1	DIP SW 4P [P-08923]
SW2, SW3	2	プッシュ SW [P-03651]
SW4	1	単極双投トグル SW　On-On 型 [P-02399]
J1, J2	2	基板用コネクタ 3 ピン　JST ZH コネクタ B3B-ZR-3.4(LF)(SN) [C-14156]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか接続用ケーブル・基板・配線材等

17.3　7 セグメント LED による文字の表示

7 セグメント LED を使用した文字の表示法を説明します. 数字の表示が 7 セグメント LED の主な使用方法ですが, 本章では数字以外の幾つかの文字の表示も行います. 図 17.8 に示すように, 7 セグメント LED は表示のための 7 つのセグメント (表示要素) に小数点を加えた合計 8 つのセグメントで構成されています. 8 つのセグメントそれぞれが独立した LED で, 表示したい文字の表示パターンに合わせて点灯する LED を選択します. 数字を表示するための 7 つのセグメントにはそれぞれ A から G までの名前が, 小数点のセグメントには DP の名前がそれぞれ付けられています.

数字やローマ字などを表示するための点灯パターンを図 17.9 に示します. 数字の 5 とローマ字の S が同じ点灯パターンになっていたり, ローマ字の大文字と小文字が混在していますが, あまり気にしないことにします.

17.4　7 セグメント LED のダイナミック点灯

ダイナミック点灯とは, 少ないマイコンの使用ピン数により複数の 7 セグメント LED を点灯する技法です. ひとつの 7 セグメント LED には 8 つの LED があるので単純に考えれば, 7 セグメント LED 1 つでは 8 本のマイコンのピンが必要で, 2 つでは 16 本, 3 つでは 24 本, ... となり, これではマイコンのピンは全く足りません. ダイナミック点灯法では, 7 セグメント LED 1 つでは 9 本, 2 つでは 10 本, 3 つでは 11 本, ... と, 少ないピン数で点灯します.

ダイナミック点灯法を具体的に説明すれば, 図 17.10 に示すように点灯する 7 セグメント LED の選択を順次切り替えてゆく技法です. 点灯させる 7 セグメント LED を電光石火の早業で次々に切り替えると, 人間の目の残像効果により全ての 7 セグメント LED が同時に点灯しているように見えます.

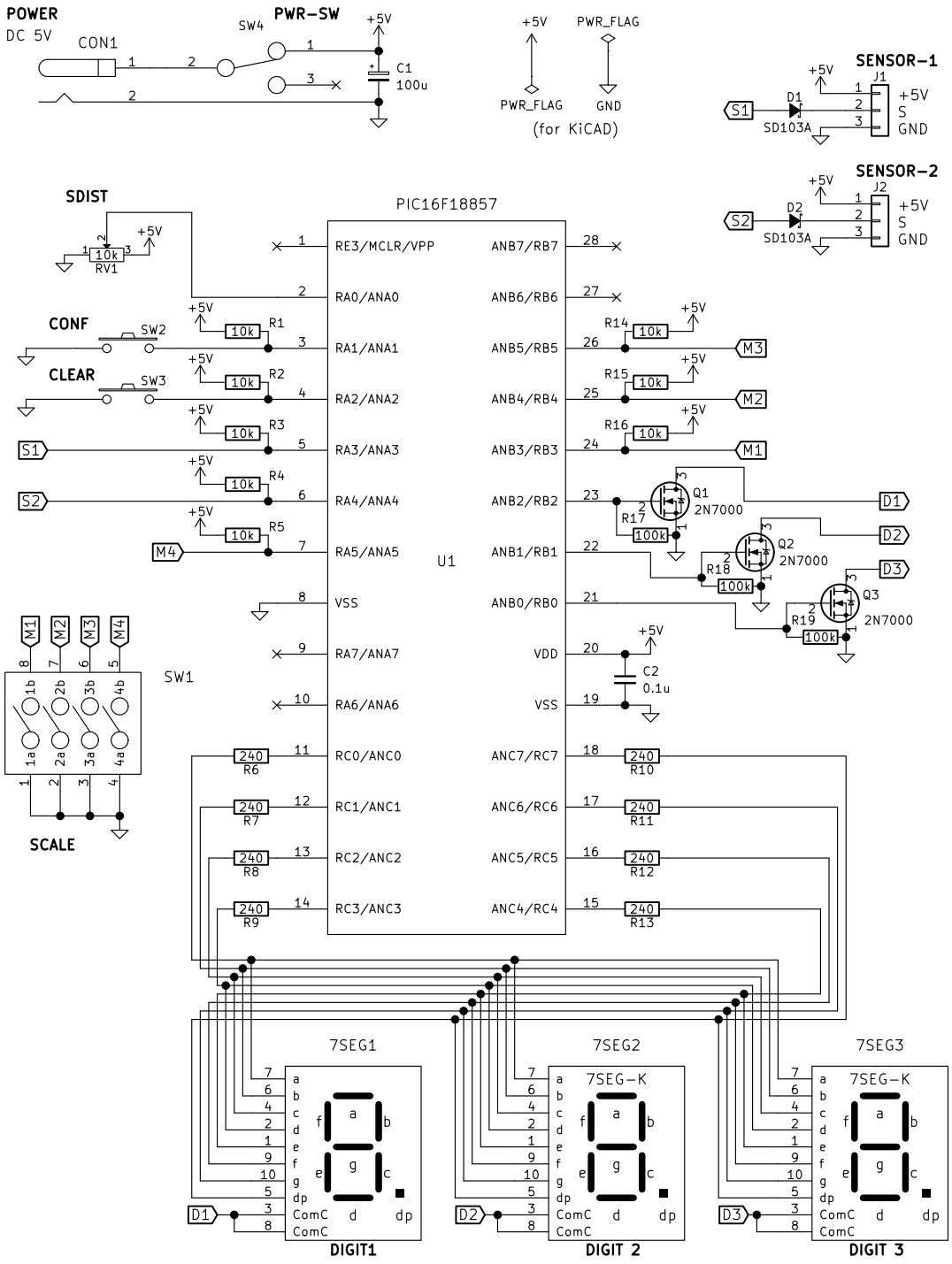


図 17.7: 【回路図】 走行速度測定器

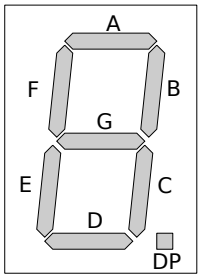
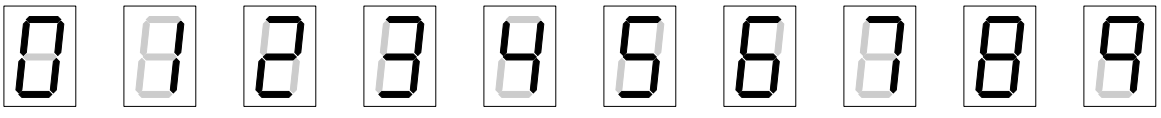
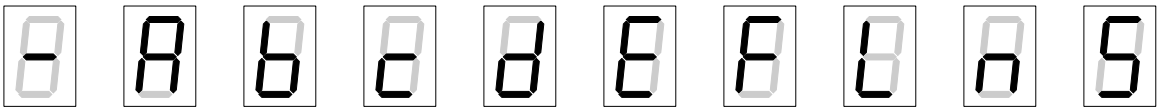


図 17.8: セグメントの名称



(a) 0 **(b)** 1 **(c)** 2 **(d)** 3 **(e)** 4 **(f)** 5 **(g)** 6 **(h)** 7 **(i)** 8 **(j)** 9



(k) – **(l)** A **(m)** B **(n)** C **(o)** D **(p)** E **(q)** F **(r)** L **(s)** N **(t)** S

図 17.9: 7 セグメント LED による文字の表示

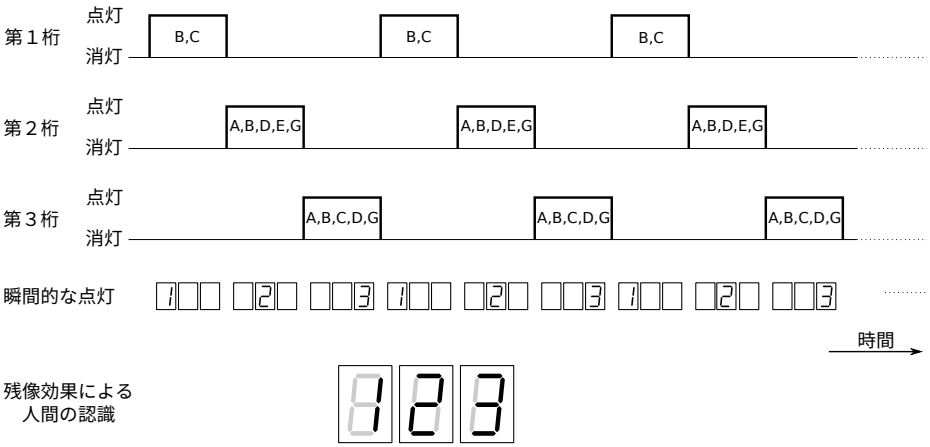


図 17.10: 7 セグメント LED のダイナミック点灯 (3 つ点灯する場合)

7 セグメント LED の各セグメントに 1 本のマイコンピンを使用し, 各 7 セグメント LED の選択に 1 本のマイコンピンを使用します. すると 7 セグメント LED が 1 つの場合に使用するマイコンのピンは 9 本, 2 つの場合には 10 本, 3 つの場合には 11 本, ... となります. 図 17.7 に示す回路では各 7 セグメント LED の選択を FET を介して駆動しています. もし FET を使用せずに PIC マイコンの I/O ポートを直接接続して駆動すれば 8 つの LED の電流すべてが 1 つのポートに流れ込んで, 1 ポート当たりの許容電流値を超える可能性があります. そこで十分な電流を流せる FET 2N7000 を使用します.

17.5　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 17.3: 【ソースファイル一覧】 走行速度測定器

初	頁	番号	ファイル名	内容	行数
✓	275	17.3	Speedometer.c	主ソースコード	224
✓	275	17.2	defs.h	定数定義: ヘッダ	19
✓	279	17.4	config.h	設定部: ヘッダ	8
✓	279	17.5	config.c	設定部: 実装	100
✓	281	17.6	speed-timer.h	速度測定部: ヘッダ	13
✓	281	17.7	speed-timer.c	速度測定部: 実装	64
✓	282	17.8	led7seg.h	7 セグメント LED 点灯: ヘッダ	11
✓	283	17.9	led7seg.c	7 セグメント LED 点灯: 実装	321
✓	274	17.1	io.h	I/O ピン割付: ヘッダ	32
✓	288	17.10	mcr.h	MCR・周辺モジュール: ヘッダ	14
✓	288	17.11	mcr.c	MCR・周辺モジュール: 実装	131
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76
				計 1022 行 (うち初出 937 行, 既出 85 行)	

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

ソースファイル番号 17.1: io.h

```
#if !defined(__IO_H__)
#define __IO_H__

/* センサ間距離設定ボリューム ADC: ANA0 */
#define ADC_CH_SDIST          0x00
/* 設定開始SW: RA1 */
#define SW_CONF_PUSHED()      (RA1 == 0)
/* クリアSW: RA2 */
#define SW_CLR_PUSHED()       (RA2 == 0)
/* スケール設定SW: RB3, RB4, RB5, RA5 */
#define SW_SCALE_1_ON()       (RB3 == 0)
#define SW_SCALE_2_ON()       (RB4 == 0)
#define SW_SCALE_3_ON()       (RB5 == 0)
#define SW_SCALE_4_ON()       (RA5 == 0)
/* 通過センサ: RA3, RA4 */
#define SENSOR_1_DET()        (RA3 == 0)
#define SENSOR_2_DET()        (RA4 == 0)
/* 7-SEG LED ダイナミック点灯用の桁選択: RB0, RB1, RB2 */
#define LED7SEG_SEL_D1_PORT   LATB2 /* 左桁 */
#define LED7SEG_SEL_D2_PORT   LATB1 /* 中桁 */
#define LED7SEG_SEL_D3_PORT   LATB0 /* 右桁 */
/* 7-SEG LED セグメント: RC0, RC1, ..., RC7 */
#define LED7SEG_A_PORT        LATC0
#define LED7SEG_B_PORT        LATC1
#define LED7SEG_C_PORT        LATC2
#define LED7SEG_D_PORT        LATC3
#define LED7SEG_E_PORT        LATC4
#define LED7SEG_F_PORT        LATC5
#define LED7SEG_G_PORT        LATC6
#define LED7SEG_DP_PORT       LATC7

#endif /*__IO_H__*/
```

ソースファイル defs.h (定数定義: ヘッダ)

定数の定義, 関数のプロトタイプ宣言, およびグローバル変数の宣言を行います.

ソースファイル番号 17.2: defs.h

```
1  #if !defined(__DEFS_H__)
2  #define __DEFS_H__
3
4  /* 計測タイムアウト(強制中断)時間 [Sec] */
5  #define TIMEOUT_VALUE        60.0
6
7  /* 計測直後の不感時間 [mSec] */
8  #define INSENSITIVE_PERIOD   10000  /* 10 [Sec] */
9
10 extern float sensor_dist_cm;
11 extern char  sensor_dist_cm_str[];
12 extern float        model_scale;
13 extern const char  *model_scale_str;
14
15 extern void read_sensor_distance(void);
16 extern void read_scale(void);
17 extern void delay_ms(uint16_t t);
18
19 #endif /*__DEFS_H__*/
```

ソースファイル Speedometer.c (主ソースコード)

動作の主となる部分の関数 main() の概要を説明します. 設定用の押しボタンスイッチ CONF を押しながら電源を入れると設定モードに入ります. 設定モードを実現するソースファイルは後述します. 設定が終われば関数 main() に戻り, 走行速度の計測を行います. 動作は以下の繰り返しです.

1. 2 つの車両センサのどちらかが車両を検出するまで待機します. 検出があれば, どちらの車両センサかを覚えておきます. これは模型車両の走行方向がどちらの場合でも対応できるようにするためです.
2. 時間の計測を開始します.
3. 最初に模型車両の検出をした方ではない車両センサが模型車両を検出があるまでループします. もし検出があればループを終了します. ループの間にクリア用の押しボタンスイッチ CLEAR や設定用の押しボタンスイッチ CONF が押されると測定を強制終了して, 対応する動作をします. もし長時間経っても検出がなければ, エラーとして測定を強制終了します.
4. 時間の計測を終了し, 走行速度を計算します. 図 17.2 で示したように, 2 つの車両センサ間の走行に要した時間, 2 つの車両センサ間の距離, 模型の縮尺の 3 つの値より, 実車に換算した走行速度 (スケールスピード) を km/h の単位で求めます.
5. 走行速度を 7 セグメント LED に表示します.
6. 一定時間は車両センサの検出に反応しないようにして, 長い編成の列車の走行時に不要な計測が始まらないようにします.

ソースファイル番号 17.3: Speedometer.c

```
1  /*********************************************
2   * 速度計（通過センサ型）/ PIC16F18857
3   * by Hirotsugu Kakugawa
4   *********************************************/
5  #include <xc.h>
6  #include <stdint.h>
7  #include <stdio.h>
8  #include "mcr.h"
9  #include "adc.h"
```

```
10  #include "io.h"
11  #include "led7seg.h"
12  #include "speed-timer.h"
13  #include "config.h"
14  #include "defs.h"
15
16  /* ------------------------------------------------------ */
17
18  /* センサ間距離 */
19  float  sensor_dist_cm  = 0;  /* 距離 [cm] */
20  char   sensor_dist_cm_str[LED7SEG_STR_SIZ_MAX];
21
22  /* 縮尺 */
23  struct scale_info_s {
24    float       value;
25    const char *str;
26  };
27  const struct scale_info_s scale_info[16] = {
28    { 220.0, "220"   }, /* Zスケール */
29    { 160.0, "160"   }, /* Nスケール(欧米) */
30    { 150.0, "150"   }, /* Nゲージ(日) */
31    { 148.0, "148"   }, /* Nゲージ(英) */
32    { 120.0, "120"   }, /* TTスケール */
33    {  87.1,  "87.1" }, /* H0スケール(欧) */
34    {  87.0, " 87"   }, /* HOスケール(米) */
35    {  80.0, " 80"   }, /* 16番(日) */
36    {  76.0, " 76"   }, /* OOゲージ */
37    {  64.0, " 64"   }, /* Sスケール */
38    {  48.0, " 48"   }, /* 0番/Oスケール(米) */
39    {  45.0, " 45"   }, /* 0番(日欧), OJ(日) */
40    {  43.5,  "43.5" }, /* 0番/Oスケール(英) */
41    {  32.0, " 32"   }, /* 1番(欧米) */
42    {  30.5,  "30.5" }, /* 1番(英) */
43    {  22.5,  "22.5" }  /* 2番 */
44  };
45  float        model_scale     = 0;
46  const char  *model_scale_str = "";
47
48  /*-------------------------------------------------------
49   * MAIN
50   *-------------------------------------------------------*/
51  void main(void)
52  {
53    /* システム初期化 */
54    System_Init();
55    /* ウオッチドッグタイマ初期化: 2 [Sec] */
56    WDTCON0bits.WDTPS = 0x0B; /* 0b01011 */
57    CLRWDT();
58    /* LED表示初期化 */
59    led7seg_init();
60    /* 計時初期化 */
61    speed_timer_init();
62    /* センサ距離設定ボリュームとスケール選択SWを読む */
63    read_sensor_distance();
64    read_scale();
65    /* 計測のタイムアウト値 */
66    uint16_t tmr_max =
67      (uint16_t)(1000.0 * TIMEOUT_VALUE / SPEED_TMR_CYCLE);
68    /* 割込許可 */
69    GIE = 1;
70    PEIE = 1;
71    /* 設定モード */
72    if (SW_CONF_PUSHED()) {
73      config_set();
74    }
75    /* 設定値表示 */
76   l_conf_disp:
77    config_display();
78    /* 主ループ */
```

```
 l_restart:
  led7seg_display_const("---");
  for (;;) {
    CLRWDT();
    uint16_t t = 0;
    /* センサ検出待ち */
    uint8_t sns = 0;
    for (;;) {
      CLRWDT();
      if (SENSOR_1_DET()) {
        sns = 1;
        break;
      } else if (SENSOR_2_DET()) {
        sns = 2;
        break;
      }
      if (SW_CLR_PUSHED()) {  /* クリア */
        goto l_restart;
      }
      if (SW_CONF_PUSHED()) { /* 設定値表示 */
        goto  l_conf_disp;
      }
    }
    /* 計時開始 */
    uint8_t timeout = 0;
    uint16_t t_prg = 0;
    uint8_t i_prg = 0;
    speed_timer_start();
    for (;;) {
      CLRWDT();
      /* 計時完了の検査 */
      if ((sns == 1) && SENSOR_2_DET()) {
        break;
      } else if ((sns == 2) && SENSOR_1_DET()) {
        break;
      }
      if (SW_CLR_PUSHED()) {  /* クリア(測定中断) */
        goto l_restart;
      }
      if (SW_CONF_PUSHED()) { /* 設定値表示(測定中断) */
        goto  l_conf_disp;
      }
      /* 計時経過時間が長いとタイムアウト */
      uint32_t tmr = speed_timer_read();
      if (tmr > tmr_max) {
        timeout = 1;
        break;
      }
      /* 計時中の LED 表示 */
      uint16_t t_prg_x = tmr >> 7;
      if (t_prg != t_prg_x) {
        t_prg = t_prg_x;
        switch (i_prg) {
        case 0: led7seg_display_const("   ");  i_prg++;
          break;
        case 1: led7seg_display_const("   ."); i_prg++;
          break;
        case 2: led7seg_display_const("  . "); i_prg++;
          break;
        case 3: led7seg_display_const(".  ");  i_prg = 0;
          break;
        }
      }
    }
    /* 計時終了 */
    uint16_t speed_km_h = speed_timer_end();
    /* 速度表示 */
    char speed_str[4];
    if (timeout == 1) {
      speed_km_h = 0;
    }
```

```
150       sprintf(speed_str, "%3d", speed_km_h);
151       for (t = 5; t != 0; t--) {
152         CLRWDT();
153         led7seg_display(speed_str);
154         delay_ms(400);
155         if (t != 1) {
156           led7seg_display_const("   ");
157           delay_ms(100);
158         }
159       }
160       /* 列車編成全体の通過（と推測できるまで）を待つ.
161        * 最後にセンサが検出してから十分時間が経過したら,
162        * 列車編成全体の通過したと判断する */
163       t = 0;
164       while (t != INSENSITIVE_PERIOD) {
165         CLRWDT();
166         t++;
167         delay_ms(1);
168         if (SENSOR_1_DET() || SENSOR_2_DET()) {  /* 通過中 */
169           t = 0;
170         }
171         if (SW_CLR_PUSHED()) {
172           goto l_restart;
173         }
174         if (SW_CONF_PUSHED()) {
175           goto  l_conf_disp;
176         }
177       }
178       /* 通過完了のインジケーション */
179       for (t = 6; t != 0; t--) {
180         CLRWDT();
181         led7seg_display_const("   ");
182         delay_ms(100);
183         led7seg_display(speed_str);
184         delay_ms(100);
185       }
186     }
187   }
188
189   /* センサ間距離の設定値を読み取る */
190   void read_sensor_distance(void)
191   {
192     adc_val_t sd = ADC_Read(ADC_CH_SDIST);
193     sensor_dist_cm = 5.0 + (45.0 * (float)sd / 1024.0);
194     sprintf(sensor_dist_cm_str, "%4.1f", sensor_dist_cm);
195   }
196
197   /* スケールの設定値を読み取る */
198   void read_scale(void)
199   {
200     uint8_t scale_selector = 0;
201     if (SW_SCALE_1_ON()) {
202       scale_selector += 0x01;
203     }
204     if (SW_SCALE_2_ON()) {
205       scale_selector += 0x02;
206     }
207     if (SW_SCALE_3_ON()) {
208       scale_selector += 0x04;
209     }
210     if (SW_SCALE_4_ON()) {
211       scale_selector += 0x08;
212     }
213     model_scale     = scale_info[scale_selector].value;
214     model_scale_str = scale_info[scale_selector].str;
215   }
216
217   void delay_ms(uint16_t t)
218   {
219     while (t != 0) {
220       t--;
```

```
221       CLRWDT();
222       __delay_ms(1);
223     }
224   }
```

ソースファイル config.h (設定部: ヘッダ)

設定モードに関する関数のプロトタイプ宣言を行います.

ソースファイル番号 17.4: config.h

```
1  #if !defined(__CONFIG_H__)
2  #define __CONFIG_H__
3
4  extern void config_set(void);
5  extern void config_display(void);
6  extern void config_sw_rel_wait(void);
7
8  #endif /*__CONFIG_H__*/
```

ソースファイル config.c (設定部: 実装)

設定モードに関する関数 config_set() の動作は以下の通りです.

1. センサ間の距離の設定します. 可変抵抗器 SDIST の電圧値を AD 変換器で読み, 利用に便利な範囲にスケールして cm 単位の値にします. 値は常時 7 セグメント LED に表示して調整の便を図ります.
2. 模型の縮尺 (スケール) の設定をします. DIP スイッチの設定を読み, 記憶している縮尺の表から縮尺の値を得ます. 値は常時 7 セグメント LED に表示して調整の便を図ります.
3. 調整が終わると再確認のために設定内容を表示します.

ソースファイル番号 17.5: config.c

```
1  /*
2   * スケール/センサ距離設定
3   */
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "io.h"
8  #include "led7seg.h"
9  #include "config.h"
10 #include "defs.h"
11
12 /* 設定 */
13 void config_set(void)
14 {
15   uint8_t t;
16   /* 設定モード開始の表示 */
17   led7seg_display_const("CNF");
18   delay_ms(5000);
19   for (;;) {
20     CLRWDT();
21     if (!SW_CONF_PUSHED()) {
22       delay_ms(500);
23       break;
24     }
25   }
26   /* センサ間距離の設定 */
27   for (t = 0; t < 15; t++) {
28     CLRWDT();
29     led7seg_display_const("   ");
30     delay_ms(100);
31     led7seg_display_const("SEN");
```

```
32      delay_ms(400);
33    }
34    for (;;) {
35      CLRWDT();
36      read_sensor_distance();
37      led7seg_display(sensor_dist_cm_str);
38      if (SW_CONF_PUSHED()) {
39        delay_ms(500);
40        config_sw_rel_wait();
41        break;
42      }
43    }
44    /* スケールの設定 */
45    for (t = 0; t < 15; t++) {
46      CLRWDT();
47      led7seg_display_const("   ");
48      delay_ms(100);
49      led7seg_display_const("SCA");
50      delay_ms(400);
51    }
52    for (;;) {
53      CLRWDT();
54      read_scale();
55      led7seg_display_const(model_scale_str);
56      if (SW_CONF_PUSHED()) {
57        delay_ms(500);
58        config_sw_rel_wait();
59        break;
60      }
61    }
62    /* 設定終了の表示 */
63    for (t = 0; t < 10; t++) {
64      CLRWDT();
65      led7seg_display_const("   ");
66      delay_ms(200);
67      led7seg_display_const("...");
68      delay_ms(200);
69    }
70  }
71
72  /* 表示 */
73  void config_display(void)
74  {
75    /* センサ間距離の表示 */
76    led7seg_display_const("SEN");
77    delay_ms(1500);
78    led7seg_display(sensor_dist_cm_str);
79    delay_ms(3000);
80    /* スケールの表示 */
81    led7seg_display_const("SCA");
82    delay_ms(1500);
83    led7seg_display_const(model_scale_str);
84    delay_ms(3000);
85  }
86
87  /* 設定SWの解放を待機 */
88  void config_sw_rel_wait(void)
89  {
90    uint8_t i = 0;
91    while (i != 50) {
92      CLRWDT();
93      delay_ms(1);
94      if (!SW_CONF_PUSHED()) {
95        i++;
96      } else {
97        i = 0;
98      }
99    }
100 }
```

ソースファイル speed-timer.h (速度測定部: ヘッダ)

走行速度を計測するための関数のプロトタイプ宣言を行います.

ソースファイル番号 17.6: speed-timer.h

```
#if !defined(__SPEED_TIMER_H__)
#define __SPEED_TIMER__H__

/* 通過時間タイマの１サイクル時間 [mSec] */
#define SPEED_TMR_CYCLE     2.048

extern void     speed_timer_init(void);
extern void     speed_timer_start(void);
extern uint16_t speed_timer_end(void);
extern uint16_t speed_timer_read(void);
extern void     speed_timer_isr(void);

#endif /*__SPEED_TIMER_H__*/
```

ソースファイル speed-timer.c (速度測定部: 実装)

走行速度を計測するための関数の実装を行います. 計時にはタイマジュール TMR4 を使用して, 予め設定した一定のカウント速度でカウントアップしてゆきます. 走行速度の実際の計算は, カウント速度, カウント数, 模型の縮尺より行います. TMR4 は 8 ビットのタイマですので, 走行速度が遅い場合にはオーバーフローが発生します. しかしカウントアップの速度を遅してオーバーフローが起きないようにすれば時間の測定の精度が下がります. そこで本ファームウエアでは, 時間の測定精度を保つために十分なカウントアップの速度を設定し, タイマにオーバーフローが起きてもその発生回数も勘定することでオーバーフローへの対処と測定精度の両立を図ります. タイマがオーバーフローすれば割り込みが発生しますから, 割り込みハンドラが呼び出された回数を勘定することでオーバーフローの発生回数が分かります. なお走行速度の演算には float 型による浮動小数点数を使用します.

ソースファイル番号 17.7: speed-timer.c

```
/* ------------------------------------------------------
 * 通過時間計測タイマ
 *------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "speed-timer.h"
#include "defs.h"

volatile static uint16_t speed_timer_overflows = 0;

/* 初期化 */
void speed_timer_init(void)
{
  TMR4_Init();
}

/* 計測開始 */
void speed_timer_start(void)
{
  speed_timer_overflows = 0;
  TMR4_Stop();
  TMR4_Write(0);
  TMR4_Start();
}

/* 計測終了: 計測結果を返す */
uint16_t speed_timer_end(void)
```

```
30  {
31    /* [km/h] = 1000/3600[m/Sec] = 1/3.6[m/Sec]
32     *        = 100/3600 [cm/mSec] = 1/36 [cm/mSec] */
33    TMR4_Stop();
34    float t_ms
35      = SPEED_TMR_CYCLE
36      * ((float)speed_timer_overflows +
37         (float)TMR4_Read() / 256.0);
38    float speed_km_h = 0;
39    if (t_ms == 0) {
40      speed_km_h = 999;
41    } else {
42      speed_km_h = 36.0 * model_scale * sensor_dist_cm / t_ms;
43    }
44    return (uint16_t)(speed_km_h + 0.5);
45  }
46
47  /* 計測終了: 計測結果を返す */
48  uint16_t speed_timer_read(void)
49  {
50    uint16_t t;
51    int8_t ogie  = GIE;
52    int8_t opeie = PEIE;
53    GIE   = 0;
54    PEIE  = 0;
55    t = speed_timer_overflows;
56    GIE   = ogie;
57    PEIE  = opeie;
58    return t;
59  }
60  /* タイマオーバーフロー発生回数を勘定 */
61  void speed_timer_isr(void)
62  {
63    speed_timer_overflows++;
64  }
```

ソースファイル led7seg.h (7 セグメント LED 点灯: ヘッダ)

7 セグメント LED に表示を行う関数のプロトタイプ宣言を行います.

ソースファイル番号 17.8: led7seg.h

```
1  #if !defined(__LED7SEG_H__)
2  #define __LED7SEG_H__
3
4  #define LED7SEG_STR_SIZ_MAX  7
5
6  extern void led7seg_init(void);
7  extern void led7seg_display(char *s);
8  extern void led7seg_display_const(const char *s);
9  extern void led7seg_itimer_isr(void);
10
11 #endif /*__LED_H__*/
```

ソースファイル led7seg.c (7 セグメント LED 点灯: 実装)

7 セグメント LED に表示を行う関数の実装を行います. このソースファイルでは以下の関数が含まれており, それぞれの動作の概要を説明します. なお本装置で使用する 7 セグメント LED 数は 3 つですが, 使用する 7 セグメント LED 数が異なる他の装置でもこのソースファイルを使用できるように, 条件コンパイルを行っています.

- 関数 led7seg_init()
  初期化を行います.

- 関数 led7seg_itimer_isr()
  ダイナミック点灯を実現します. 一定周期でこの関数が呼び出されるように TMR1 によるタイマ割り込みを使用します. TMR1 タイマ割り込みは 4.096 ミリ秒間隔としていますので, 各桁の点灯の頻度は 1 秒あたり 80 回程度になります. タイマ割り込みの間隔を長くすると残像効果は得られず表示はちらつきます. 点灯する 7 セグメント LED の表示する桁番号は変数 digit_sel で記憶しておき, 表示する桁の各セグメントをデジタル出力ポートに出力します. そして表示する桁番号を次の桁に更新します. なお表示する桁の各セグメントは関数 led7seg_display() の中で予め設定されています.
- 関数 led7seg_ch_to_sgb()
  引数に与えられた文字から, 点灯させるセグメントを求める関数です. これは図 17.9 に示す内容をプログラムコード化したものです.
- 関数 led7seg_digit()
  引数で指定された 7 セグメント LED で点灯させるセグメントを設定します.
- 関数 led7seg_display_const()
  引数で与えられた文字列を 7 セグメント LED で表示します. ただし文字列はプログラムコード領域の場合です. 実際の処理は次に説明する関数 led7seg_display() で行います. PIC 16F マイコンではデータが置かれているメモリがプログラムコード領域か RAM 領域かにより, データをアクセスする機械語コードが異なります. 透明性のあるアクセスを C 言語レベルでは実現できないため, 本ソースコードのように私たちプログラマが明示的にソースコード上に記述する必要があります. 少々面倒ですが, PIC マイコンのアーキテクチャ上これは仕方ありません.
- 関数 led7seg_display()
  7 セグメント LED で表示する内容を引数の文字列で指定します. ただし文字列は RAM 領域の場合です. この関数は 3 つの 7 セグメント LED の各セグメントを点灯するか否かを変数に記憶するだけで, 直ちに表示はしません. 実際の表示はタイマ割り込みの発生時に関数 led7seg_itimer_isr() が行います. この関数が少々複雑なのは小数点の取扱いがあるためです. 小数点さえなければ内容は単純で, 文字を順番に 3 つの 7 セグメント LED の各セグメントへ分解しているだけに過ぎません. 複雑になっている理由は, 例えば表示内容が “1.23” の場合には数字と小数点 1. を左端の 7 セグメント LED に表示する必要があるからです. また表示内容が “...” の場合も適切に表示します.

ソースファイル番号 17.9: led7seg.c

```
1  /*-------------------------------------------------------
2   * 7セグメント LED 表示 (ダイナミック点灯)
3   *-------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include <ctype.h>
7  #include "mcr.h"
8  #include "io.h"
9  #include "led7seg.h"
10
11 /* セグメントビット */
12 #define SEG_NONE  0x00
13 #define SEG_A     0x01
14 #define SEG_B     0x02
15 #define SEG_C     0x04
16 #define SEG_D     0x08
17 #define SEG_E     0x10
18 #define SEG_F     0x20
19 #define SEG_G     0x40
20 #define SEG_DP    0x80
```

```
21 |
22 | /* 選択点灯桁(ダイナミック点灯) */
23 | volatile static uint8_t digit_sel = 0;
24 | /* 左桁 セグメント点灯/消灯 */
25 | volatile static int8_t seg_d1_a, seg_d1_b, seg_d1_c;
26 | volatile static int8_t seg_d1_d, seg_d1_e, seg_d1_f;
27 | volatile static int8_t seg_d1_g, seg_d1_dp;
28 | /* 中桁 セグメント点灯/消灯 */
29 | volatile static int8_t seg_d2_a, seg_d2_b, seg_d2_c;
30 | volatile static int8_t seg_d2_d, seg_d2_e, seg_d2_f;
31 | volatile static int8_t seg_d2_g, seg_d2_dp;
32 | /* 右桁 セグメント点灯/消灯 */
33 | volatile static int8_t seg_d3_a, seg_d3_b, seg_d3_c;
34 | volatile static int8_t seg_d3_d, seg_d3_e, seg_d3_f;
35 | volatile static int8_t seg_d3_g, seg_d3_dp;
36 |
37 | volatile static uint8_t ndigits = 0;
38 |
39 | void TMR1_Init(void);
40 | void TMR1_Start(void);
41 | void TMR1_isr(void);
42 |
43 | /* 7セグメント LED 表示の初期化 */
44 | void led7seg_init(void)
45 | {
46 |   ndigits = 0;
47 |   digit_sel = 0;
48 | #if defined(LED7SEG_SEL_D1_PORT)
49 |   LED7SEG_SEL_D1_PORT = 0; ndigits++;
50 | #endif
51 | #if defined(LED7SEG_SEL_D2_PORT)
52 |   LED7SEG_SEL_D2_PORT = 0; ndigits++;
53 | #endif
54 | #if defined(LED7SEG_SEL_D3_PORT)
55 |   LED7SEG_SEL_D3_PORT = 0; ndigits++;
56 | #endif
57 |   led7seg_display_const("   ");
58 |   TMR1_Init();
59 |   TMR1_Start();
60 | }
61 |
62 | /* 周期起動ルーチン: LED ダイナミック点灯 */
63 | void led7seg_itimer_isr(void)
64 | {
65 | #if defined(LED7SEG_SEL_D1_PORT)
66 |   LED7SEG_SEL_D1_PORT = 0;
67 | #endif
68 | #if defined(LED7SEG_SEL_D2_PORT)
69 |   LED7SEG_SEL_D2_PORT = 0;
70 | #endif
71 | #if defined(LED7SEG_SEL_D3_PORT)
72 |   LED7SEG_SEL_D3_PORT = 0;
73 | #endif
74 |   if (digit_sel == ndigits) {
75 |     digit_sel = 0;
76 |   }
77 |   switch (digit_sel) {
78 |   default:
79 |   case 0: /* 左桁 D1 */
80 |     LED7SEG_A_PORT  = seg_d1_a;
81 |     LED7SEG_B_PORT  = seg_d1_b;
82 |     LED7SEG_C_PORT  = seg_d1_c;
83 |     LED7SEG_D_PORT  = seg_d1_d;
84 |     LED7SEG_E_PORT  = seg_d1_e;
85 |     LED7SEG_F_PORT  = seg_d1_f;
86 |     LED7SEG_G_PORT  = seg_d1_g;
87 |     LED7SEG_DP_PORT = seg_d1_dp;
88 | #if defined(LED7SEG_SEL_D1_PORT)
89 |     LED7SEG_SEL_D1_PORT = 1;
90 | #endif
91 |     digit_sel = 1;
```

```
92      break;
93    case 1: /* 中桁 D2 */
94      LED7SEG_A_PORT  = seg_d2_a;
95      LED7SEG_B_PORT  = seg_d2_b;
96      LED7SEG_C_PORT  = seg_d2_c;
97      LED7SEG_D_PORT  = seg_d2_d;
98      LED7SEG_E_PORT  = seg_d2_e;
99      LED7SEG_F_PORT  = seg_d2_f;
100     LED7SEG_G_PORT  = seg_d2_g;
101     LED7SEG_DP_PORT = seg_d2_dp;
102 #if defined(LED7SEG_SEL_D2_PORT)
103     LED7SEG_SEL_D2_PORT = 1;
104 #endif
105     digit_sel = 2;
106     break;
107   case 2: /* 右桁 D3 */
108     LED7SEG_A_PORT  = seg_d3_a;
109     LED7SEG_B_PORT  = seg_d3_b;
110     LED7SEG_C_PORT  = seg_d3_c;
111     LED7SEG_D_PORT  = seg_d3_d;
112     LED7SEG_E_PORT  = seg_d3_e;
113     LED7SEG_F_PORT  = seg_d3_f;
114     LED7SEG_G_PORT  = seg_d3_g;
115     LED7SEG_DP_PORT = seg_d3_dp;
116 #if defined(LED7SEG_SEL_D3_PORT)
117     LED7SEG_SEL_D3_PORT = 1;
118 #endif
119     digit_sel = 3;
120     break;
121   }
122 }
123
124 /* フォント: 文字コード -> セグメントビット */
125 static uint8_t led7seg_ch_to_sgb(char ch)
126 {
127   uint8_t sgb = SEG_NONE;
128   switch (toupper(ch)) {
129   case '0':
130     sgb = (SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F);
131     break;
132   case '1':
133     sgb = (SEG_B | SEG_C);
134     break;
135   case '2':
136     sgb = (SEG_A | SEG_B | SEG_D | SEG_E | SEG_G);
137     break;
138   case '3':
139     sgb = (SEG_A | SEG_B | SEG_C | SEG_D | SEG_G);
140     break;
141   case '4':
142     sgb = (SEG_B | SEG_C | SEG_F | SEG_G);
143     break;
144   case '5':
145     sgb = (SEG_A | SEG_C | SEG_D | SEG_F | SEG_G);
146     break;
147   case '6':
148     sgb = (SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G);
149     break;
150   case '7':
151     sgb = (SEG_A | SEG_B | SEG_C);
152     break;
153   case '8':
154     sgb = (SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G);
155     break;
156   case '9':
157     sgb = (SEG_A | SEG_B | SEG_C | SEG_F | SEG_G);
158     break;
159   case ' ':
160     sgb = (SEG_NONE);
161     break;
162   case '-':
163     sgb = (SEG_G);
```

```
164       break;
165     case 'A':
166       sgb = (SEG_A | SEG_B | SEG_C | SEG_E | SEG_F | SEG_G);
167       break;
168     case 'B':
169       sgb = (SEG_C | SEG_D | SEG_E | SEG_F | SEG_G);
170       break;
171     case 'C':
172       sgb = (SEG_D | SEG_E | SEG_G);
173       break;
174     case 'D':
175       sgb = (SEG_B | SEG_C | SEG_D | SEG_E | SEG_G);
176       break;
177     case 'E':
178       sgb = (SEG_A | SEG_D | SEG_E | SEG_F | SEG_G);
179       break;
180     case 'F':
181       sgb = (SEG_A | SEG_E | SEG_F | SEG_G);
182       break;
183     case 'L':
184       sgb = (SEG_D | SEG_E | SEG_F);
185       break;
186     case 'N':
187       sgb = (SEG_C | SEG_E | SEG_G);
188       break;
189     case 'S':
190       sgb = (SEG_A | SEG_C | SEG_D | SEG_F | SEG_G);
191       break;
192     }
193     return sgb;
194   }
195
196   /* 各セグメント点灯/消灯の値 */
197   static void led7seg_digit(uint8_t x, uint8_t sgb)
198   {
199     switch (x) {
200     case 0: /* 左桁 D1 */
201       seg_d1_a  = ((sgb & SEG_A)  == 0) ? 0 : 1;
202       seg_d1_b  = ((sgb & SEG_B)  == 0) ? 0 : 1;
203       seg_d1_c  = ((sgb & SEG_C)  == 0) ? 0 : 1;
204       seg_d1_d  = ((sgb & SEG_D)  == 0) ? 0 : 1;
205       seg_d1_e  = ((sgb & SEG_E)  == 0) ? 0 : 1;
206       seg_d1_f  = ((sgb & SEG_F)  == 0) ? 0 : 1;
207       seg_d1_g  = ((sgb & SEG_G)  == 0) ? 0 : 1;
208       seg_d1_dp = ((sgb & SEG_DP) == 0) ? 0 : 1;
209       break;
210     case 1: /* 中桁 D2 */
211       seg_d2_a  = ((sgb & SEG_A)  == 0) ? 0 : 1;
212       seg_d2_b  = ((sgb & SEG_B)  == 0) ? 0 : 1;
213       seg_d2_c  = ((sgb & SEG_C)  == 0) ? 0 : 1;
214       seg_d2_d  = ((sgb & SEG_D)  == 0) ? 0 : 1;
215       seg_d2_e  = ((sgb & SEG_E)  == 0) ? 0 : 1;
216       seg_d2_f  = ((sgb & SEG_F)  == 0) ? 0 : 1;
217       seg_d2_g  = ((sgb & SEG_G)  == 0) ? 0 : 1;
218       seg_d2_dp = ((sgb & SEG_DP) == 0) ? 0 : 1;
219       break;
220     case 2: /* 右桁 D3 */
221       seg_d3_a  = ((sgb & SEG_A)  == 0) ? 0 : 1;
222       seg_d3_b  = ((sgb & SEG_B)  == 0) ? 0 : 1;
223       seg_d3_c  = ((sgb & SEG_C)  == 0) ? 0 : 1;
224       seg_d3_d  = ((sgb & SEG_D)  == 0) ? 0 : 1;
225       seg_d3_e  = ((sgb & SEG_E)  == 0) ? 0 : 1;
226       seg_d3_f  = ((sgb & SEG_F)  == 0) ? 0 : 1;
227       seg_d3_g  = ((sgb & SEG_G)  == 0) ? 0 : 1;
228       seg_d3_dp = ((sgb & SEG_DP) == 0) ? 0 : 1;
229       break;
230     }
231   }
232
233   /* 文字列表示(3文字): 文字列がプログラムコード領域の場合 */
234   void led7seg_display_const(const char *s)
235   {
```

```
236     char r[LED7SEG_STR_SIZ_MAX];
237     uint8_t i;
238     for (i = 0; i < LED7SEG_STR_SIZ_MAX; i++) {
239       r[i] = s[i];
240       if (s[i] == '\0') {
241         break;
242       }
243     }
244     r[LED7SEG_STR_SIZ_MAX-1] = '\0';
245     led7seg_display(r);
246   }
247
248   /* 文字列表示(3文字) */
249   void led7seg_display(char *s)
250   {
251     char ch_prev = '\0';
252     uint8_t x = 0;
253     uint8_t i = 0;
254     uint8_t sgb = 0;
255     x = 0;
256     i = 0;
257     ch_prev = '.';
258     do {
259       if (ch_prev == '.') {
260         if (s[i] == '.') {
261           sgb = (unsigned)led7seg_ch_to_sgb(' ') | SEG_DP;
262           led7seg_digit(x, sgb);
263           x++;
264         } else {
265           ch_prev = s[i];
266         }
267       } else {
268         if (s[i] == '.') {
269           sgb = (unsigned)led7seg_ch_to_sgb(ch_prev) | SEG_DP;
270         } else {
271           sgb = (unsigned)led7seg_ch_to_sgb(ch_prev);
272         }
273         led7seg_digit(x, sgb);
274         x++;
275         ch_prev = s[i];
276       }
277       i++;
278     } while ((x < 3) && (s[i-1] != '\0'));
279     while (x < 3) {
280       led7seg_digit(x, SEG_NONE);
281       x++;
282     }
283   }
284
285   /*-------------------------------------------*/
286   /* TMR1 */
287   /* 7セグメントLED ダイナミック点灯用タイマとして使用 */
288   /* Src: FOSC/4, Prescale 1:1, タイマー周期: 4.096mSec */
289
290   #if (_XTAL_FREQ == 32000000)
291   #define TMR1_RELOAD_VALUE_H  0x80
292   #define TMR1_RELOAD_VALUE_L  0x00
293   #else
294   #error "CLOCK FREQ UNKNOWN"
295   #endif
296
297   void TMR1_Init(void)
298   {
299     T1CONbits.TMR1ON = 0; /* Disabled */
300     T1CONbits.CKPS   = 0; /* Prescale: 1:1 */
301     T1CLKbits.CS     = 1; /* FOSC/4 */
302     T1GCON = 0x00;
303     TMR1H = TMR1_RELOAD_VALUE_H;
304     TMR1L = TMR1_RELOAD_VALUE_L;
305     PIR4bits.TMR1IF = 0;
306     PIE4bits.TMR1IE = 1;
307   }
```

```
308
309 void TMR1_Start(void)
310 {
311   T1CONbits.TMR1ON = 1;
312 }
313
314 void TMR1_isr(void)
315 {
316   T1CONbits.TMR1ON = 0;
317   TMR1H = TMR1_RELOAD_VALUE_H;
318   TMR1L = TMR1_RELOAD_VALUE_L;
319   T1CONbits.TMR1ON = 1;
320   led7seg_itimer_isr();
321 }
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 17.10: mcr.h

```
1  #if !defined(__MCR_H__)
2  #define __MCR_H__
3
4  #define _XTAL_FREQ  32000000 /* 32MHz */
5
6  extern void System_Init(void);
7  extern void    TMR4_Init(void);
8  extern void    TMR4_Start(void);
9  extern void    TMR4_Stop(void);
10 extern void    TMR4_Write(uint8_t t);
11 extern uint8_t TMR4_Read(void);
12 extern void    TMR4_isr(void);
13
14 #endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います.

ソースファイル番号 17.11: mcr.c

```
1  /*-----------------------------------------------------
2   * MCR設定&周辺モジュール (PIC16F18857)
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8  #include "speed-timer.h"
9
10 /*-------------------------------------------*/
11 /* MCR初期化 */
12 void System_Init(void)
13 {
14   /* 発振器 (FOSC=32MHz) */
15   OSCCON1 = 0x10; /* HFINTOSC with 2x PLL */
16   OSCCON3 = 0x00;
17   OSCEN   = 0x00;
18   OSCFRQ  = 0x05; /* 16MHz */
19   OSCTUNE = 0x00;
20   while (PLLR == 0) {
21     ;
22   }
23   /* I/Oピン */
24   LATA   = 0x00;
25   LATB   = 0x00;
26   LATC   = 0x00;
```

```
27    TRISA  = 0xFF; /* (RA7,RA6),RA5,RA4,RA3,RA2,RA1,RA0 */
28    TRISB  = 0xF8; /* (RB7,RB6),RB5,RB4,RB3 */
29    TRISC  = 0x00; /* - */
30    ANSELA = 0x01; /* ANA0 */
31    ANSELB = 0x00;
32    ANSELC = 0x00;
33    WPUA   = 0xC0; /* (RA7,RA6) */
34    WPUB   = 0xC0; /* (RB7,RB6) */
35    WPUC   = 0x00;
36    ODCONA = 0x00;
37    ODCONB = 0x00;
38    ODCONC = 0x00;
39    /* 周辺モジュール */
40    TMR4_Init();
41    ADC_Init();
42  }
43
44  /*-------------------------------------------*/
45  /* TMR4 */
46  /* 計時タイマとして使用: 1周期 2.048[mSec]
47   *   256 * 32MHz / (4 * 16 * 4)
48   *   = 2.048 [mSec]
49   */
50  void TMR4_Init(void)
51  {
52    T4CONbits.TMR4ON = 0; /* Disabled */
53    T4CONbits.T4CKPS  = 0x2; /* Prescaler:  1:4  */
54    T4CONbits.T4OUTPS = 0xF; /* Postscaler: 1:16 */
55    T4CLKCONbits.CS = 1;     /* Clock: FOSC/4 */
56    PR4 = 0xFF;
57    TMR4 = 0x00;
58    TMR4IF = 0;
59    TMR4IE = 1;
60  }
61
62  void TMR4_Start(void)
63  {
64    T4CONbits.TMR4ON = 1; /* Enabled */
65  }
66
67  void TMR4_Stop(void)
68  {
69    T4CONbits.TMR4ON = 0; /* Disabled */
70  }
71
72  void TMR4_Write(uint8_t t)
73  {
74    TMR4 = t;
75  }
76
77  uint8_t TMR4_Read(void)
78  {
79    return TMR4;
80  }
81
82  void TMR4_isr(void)
83  {
84    speed_timer_isr();
85  }
86
87  /*-------------------------------------------*/
88  /* 割り込みハンドラ */
89  __interrupt() void isr(void)
90  {
91    if ((TMR1IE == 1) && (TMR1IF == 1)) {
92      extern void TMR1_isr(void);
93      TMR1IF = 0;
94      TMR1_isr();
95    }
96    if ((TMR4IE == 1) && (TMR4IF == 1)) {
97      extern void TMR4_isr(void);
98      TMR4IF = 0;
```

```
99      TMR4_isr();
100   }
101 }
102
103 /*-----------------------------------------*/
104 /* CONFIG1 */
105 #pragma config FEXTOSC  = OFF
106 #pragma config RSTOSC   = HFINTPLL
107 #pragma config CLKOUTEN = OFF
108 #pragma config CSWEN    = ON
109 #pragma config FCMEN    = ON
110 /* CONFIG2 */
111 #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
112 #pragma config PWRTE    = OFF
113 #pragma config LPBOREN  = OFF
114 #pragma config BOREN    = ON
115 #pragma config BORV     = LO
116 #pragma config ZCD      = OFF
117 #pragma config PPS1WAY  = ON
118 #pragma config STVREN   = ON
119 #pragma config DEBUG    = OFF
120 /* CONFIG3 */
121 #pragma config WDTCPS   = WDTCPS_31
122 #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
123 #pragma config WDTCWS   = WDTCWS_7
124 #pragma config WDTCCS   = SC
125 /* CONFIG4 */
126 #pragma config WRT      = OFF
127 #pragma config SCANE    = available
128 #pragma config LVP      = OFF /* 低電圧書込不可 */
129 /* CONFIG5 */
130 #pragma config CP       = OFF
131 #pragma config CPD      = OFF
```

課題

1. 本装置値では, 走行速度の表示の単位を km/h (キロメートル/時) としています. これを mph (マイル/時) でも表示できるようにして下さい. 具体的な方法は以下のように何通りも考えられます.
   - ハードウエアの改造による方法
     DIP スイッチを 5 回路のものに変更して, DIP スイッチの設定で切り替えるようにして下さい.
   - ファームウエアの改造だけによる方法
     ハードウエアは無改造で実現して下さい. 表示を km/h と mph の両方を交互に行う方法, 設定モードでどちらか一方の表示を選択する方法, あるいは PIC マイコン内蔵の EEPROM に選択を書き込んでおく方法など, 様々な方法が考えられます.
2. 単に線路電圧を測定して換算する方法は走行速度が常に表示されますが, 数値には正確性が欠けることは本章冒頭で述べた通りです. 本装置の数値には正確性がありますが, 逆にあらかじめ設置した車両センサ間を実際に走行した時でないと表示できません. そこでこれら 2 つの方法を融合する手法で, 車両センサ間を走行していない間も常に速度の表示ができるように改造して下さい.
   - 車両センサを走行した時
     本章で説明した方法で走行速度を計測します. この時に線路電圧も同時に計測し, 走行速度と線路電圧の比例係数を計算します.
   - 車両センサを走行していない時
     現時点の線路電圧を測定して, 走行速度と線路電圧の比例係数を基に走行速度を推定して表示します.

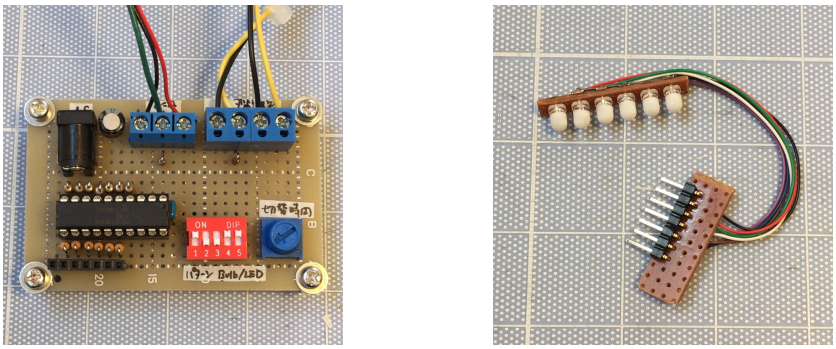
第 18 章

色灯式信号機

本章では二灯から六灯までの色灯式信号機の紹介をします. 鉄道の信号機には多種多様なものがありますが, そのなかでも色灯式の閉塞信号機, 出発信号機, 場内信号機を対象とし, 以下では信号機とはこれらの信号機を指すものとします. 在線検知による閉塞と連携した信号機を本格的に作るのは大変ですから, 車両センサによる検出とタイマ動作による簡易版を作成します. つまり車両の検知で直ちに停止現示とし, 一定時間が経つごとに注意現示, 進行現示と切り替えてゆきます. 図 18.2 が三灯式信号機での動作の例です.

本章で紹介する回路とファームウエアの特徴は以下の通りです.

- 11 種類の現示パターンを用意



(a) 制御部　　**(b)** 表示部 (六灯式信号機の例)

図 18.1: 色灯式信号機の試作例

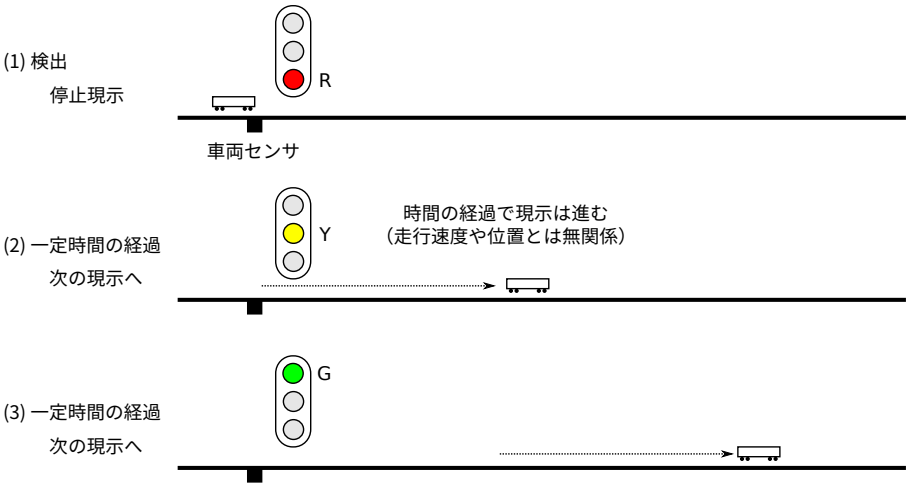


図 18.2: 信号の動作の例 (三灯式信号機の例)

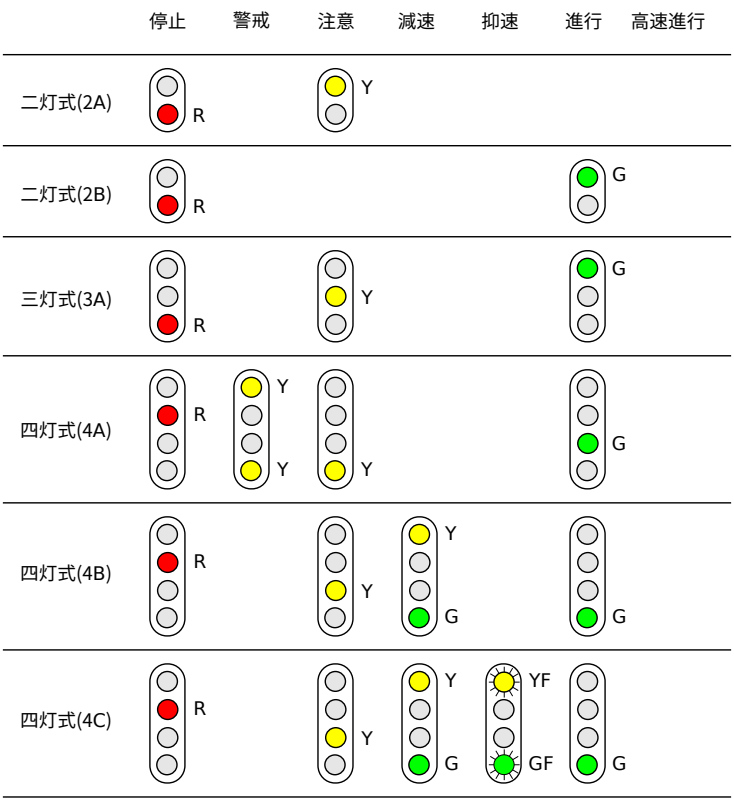


図 18.3: 現示パターン (1/2)

信号機には二灯式から六灯式まで様々なものが使用されていますが, 図 18.3 に示す 11 通りの現示パターンを実現します. これには高速進行現示や毎分 80 回点滅するフリッカー現示も含まれています.

- DIP スイッチによる現示パターンの選択
  マイコンのファームウエアを書き換えることなく切り替えができます. ひとつのファームウエアですべての現示パターンを実現します.
- 信号機の表示部はコネクタで差し替え可能
  信号機制御部はそのままで, 表示部を差し替えるだけで異なる灯数の信号機にできます.
- 電球と LED の点灯の違いをソフトウエアでシミュレート
  DIP スイッチの設定により電球風点灯と LED 風点灯と選択します. 本回路の表示部には LED を使用しますが, 第 7 章で紹介した電球風点灯の技法を使用して電球の輝度変化を模倣します.
- 停止現示入力
  外部からの停止現示入力により強制的に停止現示にする機能です. ポイントが適切に切り替えられていない, あるいは駅長の出発合図が出ていない場合には, 出発信号機には進行現示を出せません. そのような機能を実現します.

本章で学ぶ内容は以下の通りです.

- 複数の有限状態機械の並行動作
- 動作仕様のデータ記述とその逐次解釈により, 複数の動作仕様を統一的に実現する技法

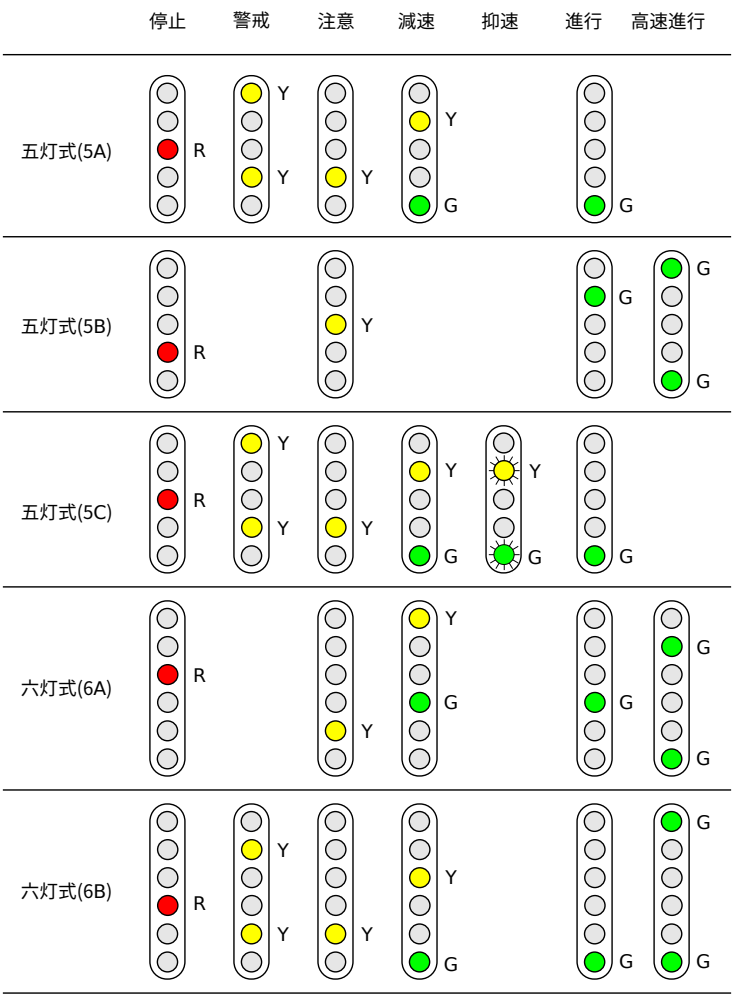


図 18.3: 現示パターン (2/2)

18.1　使用方法

本装置の使用方法を以下に説明します. 本装置は制御部と表示部で構成されています. 以下の説明で使用する部品名称は図 18.4 に配置例とともに示しています.

1. 図 18.5 に示すように車両センサを設置して, 本装置の制御部に接続します.
2. 表示部を制御部に接続します.
3. DIP スイッチで現示パターンと点灯シミュレーションの設定をします. DIP スイッチ設定は表 18.1 に示す通りです. 現示パターンの選択を DIP スイッチ 1 から 4 で, LED 風/電球風の点灯シミュレーションを DIP スイッチ 5 で, それぞれ選択します.
4. 5V の AC アダプタを DC ジャックに差し込みます. **間違って 12V 出力の AC アダプタを使用すると回路が破損する場合がありますので, 十分に注意して下さい.**
5. 可変抵抗器 TIME を調節して, 現示の切り替わる時間を設定します.
6. 必要に応じて強制停止現示入力への接続をしますが, この接続はしなくても動作します. 2 系統の強制停止現示入力が本装置には備えてあります.

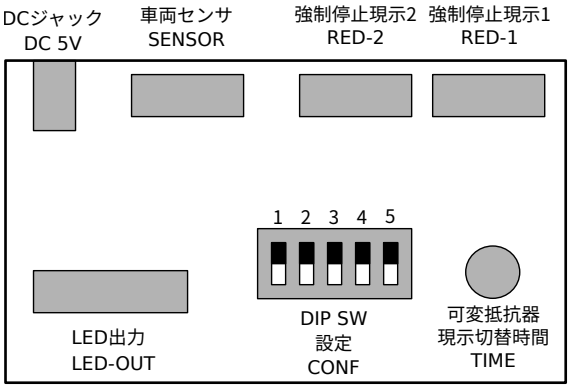


図 18.4: 操作表示部品の配置例と名称

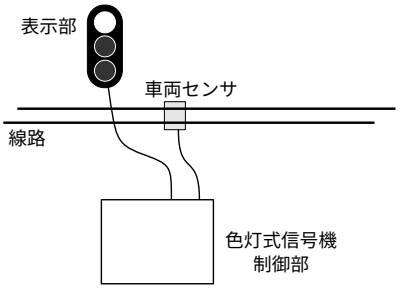


図 18.5: 接続方法

表 18.1: DIP スイッチによる現示パターンの設定

DIP スイッチ					現示パターン	使用する表示部
1	2	3	4	5		
OFF	OFF	OFF	OFF	*	二灯式 (2A)	Module 2A
ON	OFF	OFF	OFF	*	二灯式 (2B)	Module 2B
OFF	ON	OFF	OFF	*	三灯式 (3A)	Module 3A
ON	ON	OFF	OFF	*	四灯式 (4A)	Module 4A
OFF	OFF	ON	OFF	*	四灯式 (4B)	Module 4B
ON	OFF	ON	OFF	*	四灯式 (4C)	Module 4B
OFF	ON	ON	OFF	*	五灯式 (5A)	Module 5A
ON	ON	ON	OFF	*	五灯式 (5B)	Module 5B
OFF	OFF	OFF	ON	*	五灯式 (5C)	Module 5A
ON	OFF	OFF	ON	*	六灯式 (6A)	Module 6A
OFF	ON	OFF	ON	*	六灯式 (6B)	Module 6B
*	*	*	*	OFF	LED 風点灯	
*	*	*	*	ON	電球風点灯	

強制停止現示入力について説明します. この入力の仕様はアクセサリ制御入力としてありますので, 例えば以下のものが接続可能です.

- ポイント開通出力
  ポイントが非開通の場合に停止現示ができます. 第 12 章や第 15 章で紹介したポイント切替器が備えている開通状態出力を本装置の強制停止現示入力へ接続します. なお第 14 章で紹介した簡易型ポイント切替器には開通状態出力はありません. また Fulgrex ACC001 Slow Action Point Motor のように補助接点を有するポイントマシンの場合には, 補助接点を強制停止現示入力に接続する方法もあります.
- 出発許可スイッチ
  出発許可を出ていない間は強制的に停止現示とし, 出発許可が出れば直ちに進行現示にできます. 例えば人間が手作業でスイッチを操作して出発許可を出すという使い方や, マイコン等で自動制御した出発許可を出すという使い方ができます. 押しボタンスイッチで出発許可を行うときには, スイッチはオンのと

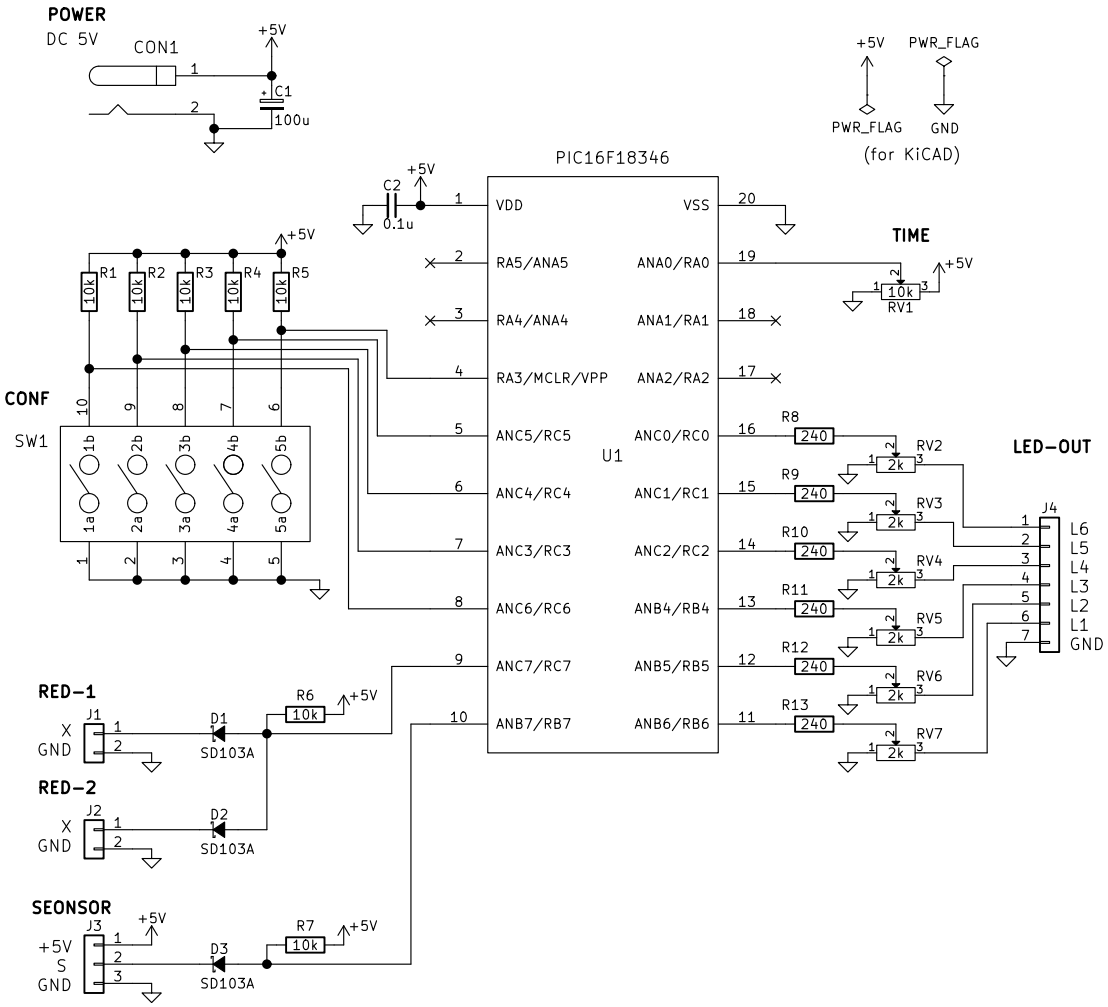


図 18.6: 【回路図】 色灯式信号器制御部

きに GND に短絡し, オフのときに開放と配線します. するとスイッチがオンのときに停止現示となります.

本装置では 2 系統の強制停止現示入力を備えていますが, ひとつでも停止の入力があれば停止現示とします. 危険な条件がすべて解除されなければ進行してはならないからです. 例えば駅の場内から本線に出るまでの経路にポイントが 2 つある場合には, 本線までの経路が開通しない限り出発信号を停止現示にできます.

18.2　回路図

色灯式信号機制御部の回路図を図 18.6 に, 部品リストを表 18.2 に, それぞれ示します. 表示部の回路図を図 18.7 に, 部品リストを表 18.3 に, 試作例の写真を図 18.1b に, それぞれ示します. 接続のためのケーブルなどは各自の使用に合わせて作成して下さい.

表 18.2: 【部品表】 色灯式信号機制御部

記号	数量	部品名
U1	1	PIC 16F18346-I/P [I-11886]

(つづく)

(つづき)

記号	数量	部品名
D1, D2, D3	3	ショットキーバリアダイオード SD103A [I-04271]
C1	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C2	1	セラミックコンデンサ 0.1μF [P-00090]
R8, R9, R10, R11, R12, R13	6	抵抗 240Ω　1/4W [R-25241]
R1, R2, R3, R4, R5, R6, R7	7	抵抗 10kΩ　1/4W [R-25103]
RV1	1	可変抵抗器 10kΩ　B カーブ, 半固定型 [P-08012]
RV2, RV3, RV4, RV5, RV6, RV7	6	可変抵抗器 2kΩ　B カーブ, 半固定型 [P-03273]
	1	IC ソケット 20 ピン [P-00031]
SW1	1	DIP SW 5P [P-7340]
J1, J2	2	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
J3	1	基板用コネクタ 3 ピン　JST ZH コネクタ B3B-ZR-3.4(LF)(SN) [C-14156]
J4	1	基板用コネクタ 7 ピン　JST XH コネクタ B7B-XH-A(LF) [C-12252]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか接続用ケーブル・基板・配線材等

表 18.3: 【部品表】 色灯式信号機表示部

記号	数量	部品名
D5, D11, D12, D17, D18, D19, D22, D23, D27, D29, D31, D32, D37	13	LED 緑 [I-06413]
D1, D3, D6, D7, D8, D10, D14, D16, D20, D24, D26, D30, D33, D34, D36	15	LED 黄 [I-11639]
D2, D4, D9, D13, D15, D21, D25, D28, D35	9	LED 赤 [I-04780]
J1, J2, J3, J4, J5, J6, J7, J8, J9	9	コネクタハウジング 7 ピン　JST XH コネクタ XHP-7 [C-12260]
	63	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
		(全 LED モジュールでの総数)
		ほか接続用ケーブル・基板・配線材等

表 12.4 で示したように, ポイント切替器の開通状態出力では非開通を L に, 開通を HiZ とした理由を説明します. ポイントが複数ある場合には, ひとつでも非開通のポイントがあれば停止現示としなくてはなりません. 「ひとつでも」というのは論理和の計算ですから, 非開通を L とすればワイヤード OR 回路で簡単に計算できるため回路は簡単になります. 本装置の回路図では 2 系統ある強制停止現示入力の部分は単なるワイヤード OR

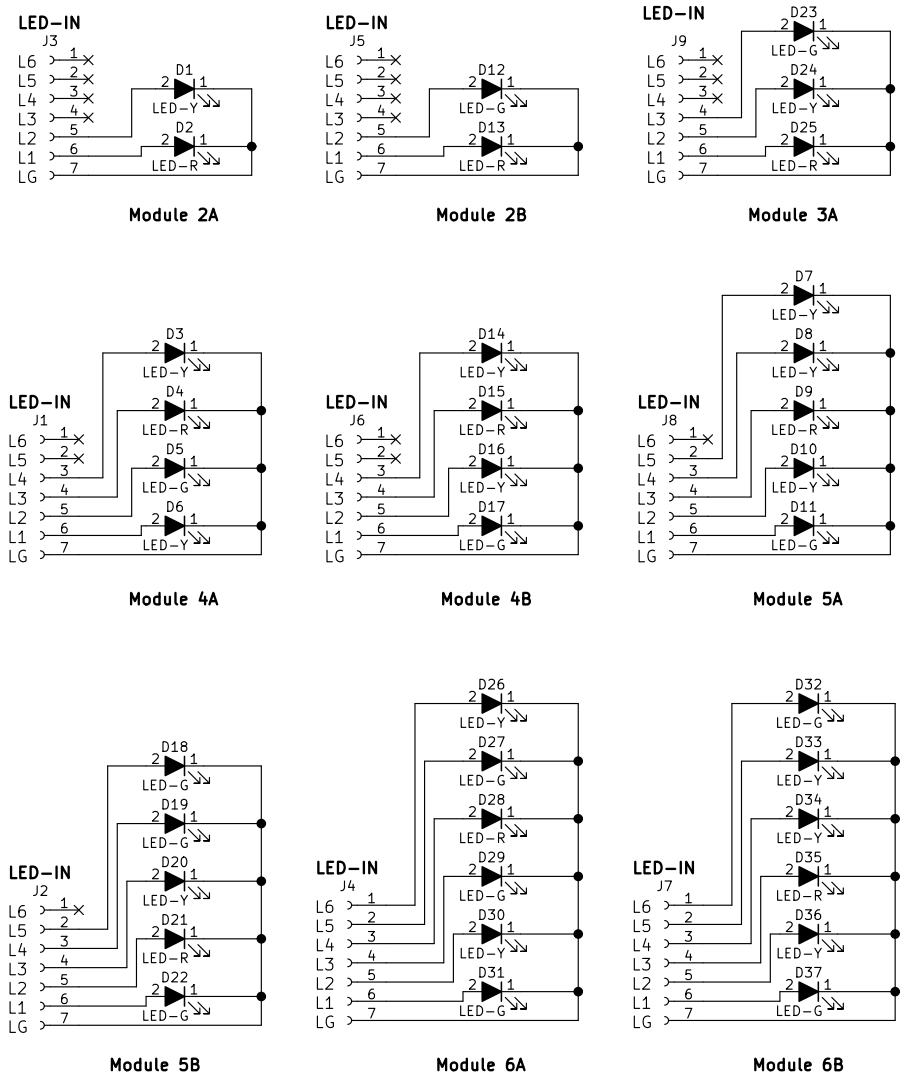


図 18.7: 【回路図】 色灯式信号器表示部

回路としており, 系統数を増やすことも簡単です. ただし開通状態出力のケーブルが断線すると HiZ となって開通と認識されます. すなわちこの方式はフェイルセーフではないことを必ず理解して, 安全性に注意を払わなくてはなりません.

18.3　動作仕様のデータ記述化

本章で実現する二灯式から六灯式までの様々な信号機は, それぞれの点灯パターンをデータ化しておき, 使用するデータを選択することで実現します. つまり別個にファームウエアを開発するのではありません. マイコンからは 6 つの LED 出力を用意しておき, 信号機で実際に使う LED のみを配線します. そしてファームウエアでは, 6 つの LED 出力の中から点灯させるもの順次変化させます. すると信号機の現示は時々刻々と変化してゆきます.

点灯パターンのデータ化の考え方を, 図 18.3 で示す三灯式 (3A) による三灯式信号機の動作を例にして説明します. この信号機には赤, 黄, 緑の 3 つの LED を用いています. 回路図 18.7 によれば, 緑は LED3, 黄は LED2, 赤は LED1 に, それぞれ対応付けられています.

まず最初に第 1 案として, 信号機がとるべき動作そのままに従ったプログラム構造を以下に示します.

```
Restart:
/* 現示: 停止 */
LED3(緑)消灯, LED2(黄)消灯, LED1(赤)点灯;
一定時間待機
 (ただし車両センサで検出があれば, 直ちに待機を終了して goto Restart);
/* 現示: 注意 */
LED3(緑)消灯, LED2(黄)点灯, LED1(赤)消灯;
一定時間待機
 (ただし車両センサで検出があれば, 直ちに待機を終了して goto Restart);
/* 現示: 進行 */
LED3(緑)点灯, LED2(黄)消灯, LED1(赤)消灯;
for (;;) {
  一定時間待機
   (ただし車両センサで検出があれば, 直ちに待機を終了して goto Restart);
}
```

次に第 2 案として, 上記のプログラム構造で共通化できる部分をまとめてみたプログラム構造を以下に示します.

```
int state = 0; /* 現在の現示 */
for (;;) {
  switch (state) {
  case 0: /* 現示: 停止 */
    LED3(緑)消灯, LED2(黄)消灯, LED1(赤)点灯; state++; break;
  case 1: /* 現示: 注意 */
    LED3(緑)消灯, LED2(黄)点灯, LED1(赤)消灯; state++; break;
  case 2: /* 現示: 進行 */
    LED3(緑)点灯, LED2(黄)消灯, LED1(赤)消灯; break;
  }
  一定時間待機
   (ただし車両センサで検出があれば, 直ちに待機を終了し state = 0 を実行);
}
```

最終案として, 上記ではプログラムロジックで書いていた点灯/消灯のパターンを, データで記述したプログラム構造を以下に示します. なお現示パターンにおける現示の内容は赤黄緑の 3 つの点灯情報で表してあり, if 文での*は 0 と 1 のどちらでも良いことを意味します.

```
int pattern[] = {  /* 現示パターン: 三灯式3A */
  "001", /* 現示: 停止 */
  "010", /* 現示: 注意 */
  "100", /* 現示: 進行; 最後のデータ */
};
int state = 0; /* 現在の現示 */
for (;;) {
  if (pattern[state] が "1**" の形) {  /* LED1:緑 */
    LED3(緑)点灯;
  } else {
    LED3(緑)消灯;
  }
  if (pattern[state] が "*1*" の形) {  /* LED2:黄 */
    LED2(黄)点灯;
  } else {
```

```
16      LED2(黄)消灯;
17    }
18    if (pattern[state] が "**1" の形) {  /* LED3:赤 */
19      LED1(赤)点灯;
20    } else {
21      LED1(赤)消灯;
22    }
23    if (pattern[state]は最後のデータではない) {
24      state++;
25    }
26    一定時間待機
27     (ただし車両センサで検出があれば, 直ちに待機を終了し state = 0 を実行);
28  }
```

本章のファームウエアの基本的な構造はこの最終案に基づいています. なお上記の説明は実際のファームウエアとは若干異なっていますので注意して下さい. これは特にフリッカー現示や電球風の点灯・消灯を実現しているためです. 最終案では行数が増えて複雑になったように見えますが, 利用可能な信号機を増やす際には現示パターンをデータとして追加するだけで済みます. このときプログラムロジックを追加して書くわけではないので, バグが入り込むことはまず起きません. 信号機のファームウエアは比較的単純なので第 2 案のプログラム構造を採用しても悪くはないのですが, 第 28 章のポイント一括切替器や第 30 章のスクリプトによる自動運転システムに向けたソフトウエア技法の基礎学習という観点から, 最終案のプログラム構造を本章では採用します. このように動作内容をデータで記述してデータを随時解釈しながら実行を進める方式は, ドメイン固有言語 やスクリプト実行機構の原始的な形態です.

実際のファームウエアでは LED 点灯情報のほか, フリッカー現示を示す情報やデータの終了を示す情報も含めて 8 ビット (1 バイト) で表現します. 具体的には以下の 8 つの情報を 1 つの単位として現示を表現するデータを構成します. そしてこのデータの列によりひとつの信号機の現示パターンを定めます.

- LED1 から LED6 まで, それぞれの LED の点灯を表す情報 (6 ビット)
- フリッカー現示を示す情報 (1 ビット)
- データの終了を示す情報 (1 ビット)

詳細はこの後のファームウエアの部分で説明します.

18.4　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 18.4: 【ソースファイル一覧】 色灯式信号機

初	頁	番号	ファイル名	内容	行数
✓	309	18.6	Signal-TimerBased.c	主ソースコード	197
✓	306	18.4	aspect.h	現示パターン: ヘッダ	72
✓	307	18.5	aspect.c	現示パターン: 実装	111
✓	312	18.7	light-fsm.h	点灯用有限状態機械: ヘッダ	10
✓	313	18.8	light-fsm.c	点灯用有限状態機械: 実装	133
✓	304	18.2	led.h	LED 点灯制御関数: ヘッダ	17
✓	304	18.3	led.c	LED 点灯制御関数: 実装	87

(つづく)

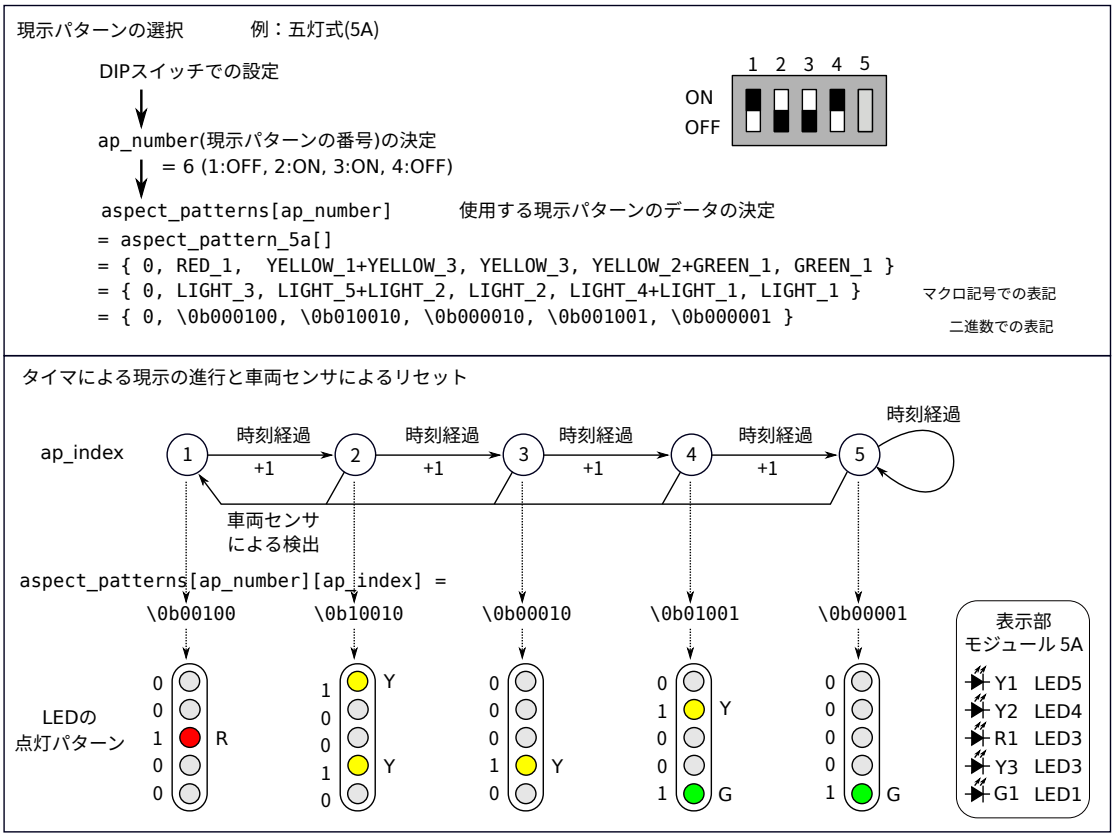


図 18.8: 現示パターンのデータ表現と現示の進行の仕組み

(つづき)

初	頁	番号	ファイル名	内容	行数
	134	6.3	incr-exp.h	指数増加アルゴリズム: ヘッダ	8
	134	6.4	incr-exp.c	指数増加アルゴリズム: 実装	77
✓	303	18.1	io.h	I/O ピン割付: ヘッダ	52
✓	316	18.9	mcr.h	MCR・周辺モジュール: ヘッダ	13
✓	316	18.10	mcr.c	MCR・周辺モジュール: 実装	162
✓	117	5.5	adc.h	ADC モジュール: ヘッダ	9
✓	118	5.6	adc.c	ADC モジュール: 実装	76

計 1024 行 (うち初出 939 行, 既出 85 行)

本ファームウエアは複雑そうに見えますが, その基本構造は以外に簡単です. 主要な変数は以下の 3 つで, 現示パターンのデータ表現と現示の進行の仕組みを図 18.8 に示します.

- 変数 ap_number
  DIP スイッチで設定した信号機の現示パターンの番号を記憶します.
- 変数 ap_index
  現時点における信号機の現示の番号を記憶します.
- 変数 aspect_patterns[][]

```
1  main() {
2    ap_number = DIPスイッチによる灯式の設定の値;
3    for (;;) {
4      if (車両センサによる検出あり) {
5        ap_index = 1;  /*停止現示*/
6        timer_aspect = 現示の切り替えまでの残り時間;
7      }
8      if (aspect_patterns[ap_number][ap_index]は最終の現示(進行現示)でない) {
9        if (timer_aspect != 0) { /*現示の切り替えまでまだ時間がある*/
10         timer_aspect--; /*残り時間の減少*/
11       } else { /*ちょうど現示の切り替えのタイミング*/
12         ap_index++; /*次の現示へ*/
13         timer_aspect = 現示の切り替えまでの残り時間;
14       }
15     } /*最終の現示ならいまの現示のままにする*/
16     if (強制現示の入力あり) {
17       LEDをaspect_patterns[ap_number][1]の内容で点灯;
18     } else {
19       LEDをaspect_patterns[ap_number][ap_index]の内容で点灯;
20     }
21   }
22 }
```

図 18.9: ファームウエア構造 (概要)

信号機の現示パターンすべてをデータで表現する二次元配列です. 時刻が進むごとに変数 ap_index の値を 1 づつ増加してゆきます.

値 aspect_patterns[ap_number][ap_index] が現時点における現示の内容を表すデータです. ただし強制停止現示の入力があるときには aspect_patterns[ap_number][1] すなわち停止が現示されます. このように現示パターンは全てデータで表現されていますので, 新たな現示パターンの登録は簡単に行えます.

ファームウエアの動作の骨子は以下の通り単純なものです.

1. 車両センサによる模型車両の検出を待機します.
2. 車両センサによる模型車両の検出があれば即座に変数 ap_index の値を 1 にします.
   そして aspect_patterns[ap_number][ap_index] の内容を点灯しますが, この内容は停止現示と決めてありますから, 停止が現示さます. つまり模型車両の検出があれば停止現示になります. そして計時タイマを動作させます.
3. 前回の現示の変更から一定時刻が経過すると変数 ap_index の値を 1 増やします.
   なお aspect_patterns[ap_number][ap_index] の内容がデータの終わりなら変数 ap_index の値は増やしません. aspect_patterns[ap_number][ap_index] の内容を点灯して, 再び計時タイマを動作させます.
4. 強制停止現示入力があれば即座に停止現示にしますが, この場合には変数 ap_index の値は変更しません. 強制停止現示入力がなくなれば元の現示に戻します.

本章のファームウエアの大まかな構造を図 18.9 に示します. ただしフリッカー現示や電球風の点灯切り替えの仕組みは省略しています.

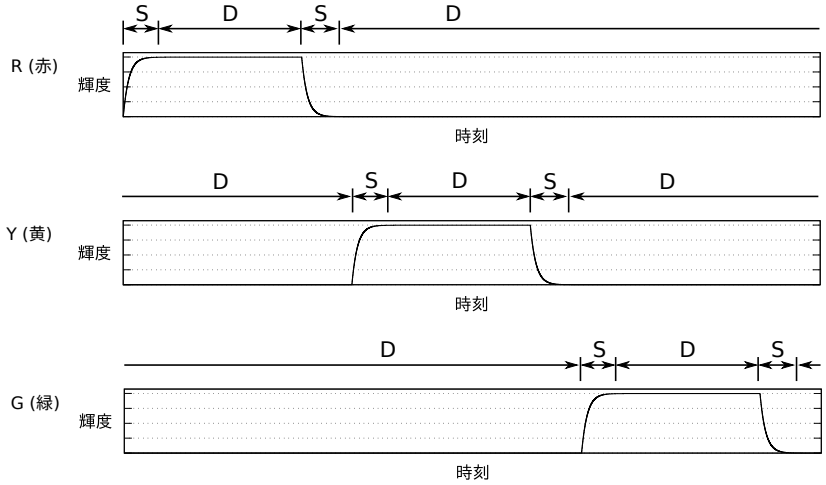
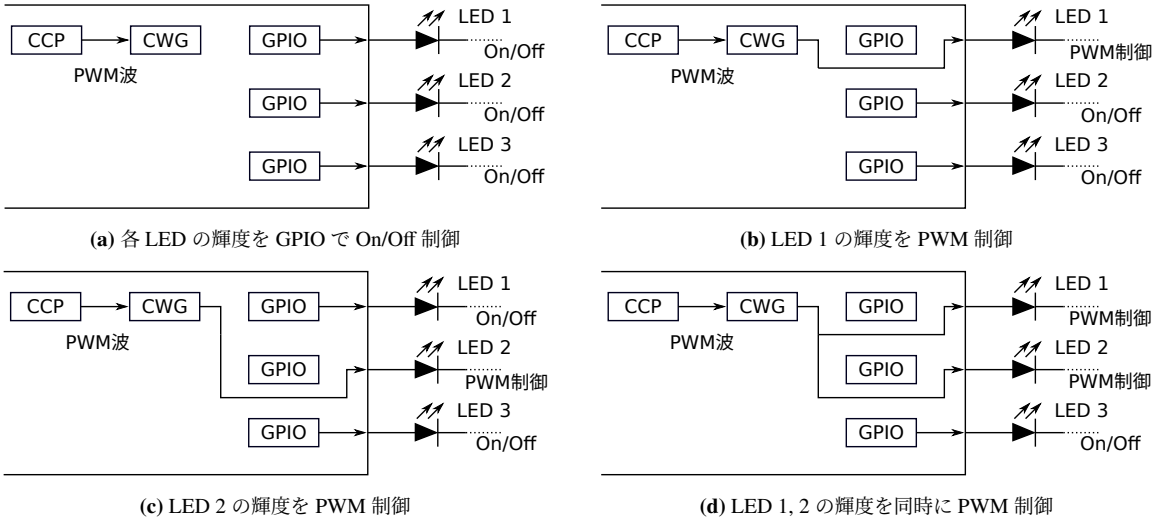


図 18.10: 電球風点灯を実現する LED 輝度制御 (3 灯式信号機での例). D はデジタルポート (GPIO) 出力, S は CGW モジュールのステアリング出力による制御を表す. 時間によって I/O ピンへの出力元を切り替える.



(a) 各 LED の輝度を GPIO で On/Off 制御

(b) LED 1 の輝度を PWM 制御

(c) LED 2 の輝度を PWM 制御

(d) LED 1, 2 の輝度を同時に PWM 制御

図 18.11: CWG と GPIO を組み合わせた LED の輝度の制御の概要

本ファームウエアには少々技巧的な点があり, それは最大 6 つの LED を個別に電球風に点灯しなくてはならないことに起因します. 以下にその概要を説明します. LED の電球風点灯の実現は, 第 5 章および第 7 章と同様に CCP モジュールを使用して輝度の制御を行います. マイコンに 6 つの CCP モジュールが内蔵されていれば個別に 6 つの LED の輝度の制御ができてファームウエアは簡単になるのですが, 残念ながら CCP モジュールは 6 つも内蔵されていません. そこで苦肉の策として, 以下の方法で 6 つの LED の輝度を制御します. 図 18.11 にこの方法の概要を示します.

- ひとつの CCP モジュール (CCP1) と, ふたつの CWG モジュール (CWG1 と CGW2) を使用します. そして CCP1 の出力を CWG1 と CWG2 双方への入力とします.
- ステアリング出力の機能を利用して, CWG1 は 4 出力, CGW2 は 2 出力とし, 6 つの LED の輝度を制御します. 信号機では輝度変化の発生は同時に起きるため, 異なる LED で互いに異なる輝度変化は不要だ

からです.

- しかしこれだけでは LED 全部が同時に輝度変化しますから, PWM 制御により輝度変化を行う LED と行わない LED とを設定します. PWM 制御による輝度変化を行わない LED は, CWG ステアリング出力からデジタルポート出力 (GPIO) へ切り替えます. つまり輝度変化を行うときにのみ CWG ステアリング出力に切り替えて輝度の PWM 制御を行います.

以上の方針に従ってマイコンの I/O ピンを制御する内蔵周辺モジュールを時々刻々切り替えます. 今の時代ではこのような技巧的な手法は好ましからざるものですが, ハードウエアの制約上やむを得ないと言わざるを得ません.

また本ファームウエアでは複数の有限状態機械を並列的に動作させていくつもの動作が同時進行しますが, 動作ごとに有限状態機械を用意することで一見複雑な動作も簡潔なプログラム構造で実現が可能です. 本ファームウエアを有限状態機械による技法を用いずに実装するとしたら, 著しく複雑なプログラム構造になり, その実現の困難さは想像を超えます.

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います. LED の数は 6 つありますから, 以降では LED1 から LED6 と番号を振って区別をします. LED が接続されている I/O ピンを CWG ステアリング出力に設定するマクロ (LED_1_SRC_PWM() など) やデジタルポート出力に設定するマクロ (LED_1_SRC_GPIO() など) を定義しています. LED と CWG ステアリング出力との対応は, LED1 から LED4 は CWG1 のステアリング出力 A から D, LED5 から LED6 は CWG2 のステアリング出力 A から B としています.

ソースファイル番号 18.1: io.h

```
#if !defined(__IO_H__)
#define __IO_H__

/* 現示切り替え時間設定ボリュームADCチャンネル: ANA0 */
#define ADC_CH_TIME         0x00
/* LED輝度 0(off)..1023(on): CCP1 */
#define LED_PWM(D)          CCP1_SetDutyCycle(D)
/* 車両検出センサ (センサ入力値L:存在、H:非存在): RB7 */
#define CAR_DETECT_PORT()       (RB7 == 0)
#define CAR_DETECT_IOC()        (IOCBFbits.IOCBF7 == 1)
#define CAR_CLEAR_IOC()         (IOCBFbits.IOCBF7 = 0)
/* LED1: RB6 (CWG1A) */
#define LED_1_GPIO_ON()    (CWG1STRbits.OVRA = 1)
#define LED_1_GPIO_OFF()   (CWG1STRbits.OVRA = 0)
#define LED_1_SRC_PWM()    (CWG1STRbits.STRA = 1)
#define LED_1_SRC_GPIO()   (CWG1STRbits.STRA = 0)
/* LED2: RB5 (CWG1B) */
#define LED_2_GPIO_ON()    (CWG1STRbits.OVRB = 1)
#define LED_2_GPIO_OFF()   (CWG1STRbits.OVRB = 0)
#define LED_2_SRC_PWM()    (CWG1STRbits.STRB = 1)
#define LED_2_SRC_GPIO()   (CWG1STRbits.STRB = 0)
/* LED3: RB4 (CWG1C) */
#define LED_3_GPIO_ON()    (CWG1STRbits.OVRC = 1)
#define LED_3_GPIO_OFF()   (CWG1STRbits.OVRC = 0)
#define LED_3_SRC_PWM()    (CWG1STRbits.STRC = 1)
#define LED_3_SRC_GPIO()   (CWG1STRbits.STRC = 0)
/* LED4: RC2 (CWG1D) */
#define LED_4_GPIO_ON()    (CWG1STRbits.OVRD = 1)
#define LED_4_GPIO_OFF()   (CWG1STRbits.OVRD = 0)
#define LED_4_SRC_PWM()    (CWG1STRbits.STRD = 1)
#define LED_4_SRC_GPIO()   (CWG1STRbits.STRD = 0)
/* LED5: RC1 (CWG2A) */
#define LED_5_GPIO_ON()    (CWG2STRbits.OVRA = 1)
#define LED_5_GPIO_OFF()   (CWG2STRbits.OVRA = 0)
#define LED_5_SRC_PWM()    (CWG2STRbits.STRA = 1)
```

```
36 #define LED_5_SRC_GPIO()  (CWG2STRbits.STRA = 0)
37 /* LED6: RC0 (CWG2B) */
38 #define LED_6_GPIO_ON()   (CWG2STRbits.OVRB = 1)
39 #define LED_6_GPIO_OFF()  (CWG2STRbits.OVRB = 0)
40 #define LED_6_SRC_PWM()   (CWG2STRbits.STRB = 1)
41 #define LED_6_SRC_GPIO()  (CWG2STRbits.STRB = 0)
42 /* 現示パターン選択スイッチ */
43 #define SW_PATTERN_NUMBER_1()         (RC6 == 0)
44 #define SW_PATTERN_NUMBER_2()         (RC3 == 0)
45 #define SW_PATTERN_NUMBER_3()         (RC4 == 0)
46 #define SW_PATTERN_NUMBER_4()         (RC5 == 0)
47 /* 点灯切り替えモード選択スイッチ */
48 #define SW_LIGHT_MODE_BULB()          (RA3 == 0)
49 /* 強制赤現示入力（出発禁止，ポイント不開通等） */
50 #define FORCE_RED()                   (RC7 == 0)
51
52 #endif /*__IO_H__*/
```

ソースファイル led.h (LED 点灯制御関数: ヘッダ)

点灯内容のデータ化では LED1 から LED6, フリッカ現示, データ終了の合計 8 つの情報を扱いますが, これら情報をビットパターンで表現するための定数を定義します. また LED の点灯を行う関数のプロトタイプ宣言を行います.

ソースファイル番号 18.2: led.h

```
1  #if !defined(__LED_H__)
2  #define __LED_H__
3
4  /* ライト番号 */
5  #define LIGHT_1         0x01 /* 0b00000001 */
6  #define LIGHT_2         0x02 /* 0b00000010 */
7  #define LIGHT_3         0x04 /* 0b00000100 */
8  #define LIGHT_4         0x08 /* 0b00001000 */
9  #define LIGHT_5         0x10 /* 0b00010000 */
10 #define LIGHT_6         0x20 /* 0b00100000 */
11 #define LIGHT_FLICKER   0x40 /* 0b01000000 */
12 #define LIGHT_END       0x80 /* 0b10000000 */
13
14 extern void led_gpio_set(uint8_t lpat);
15 extern void led_pwm_steer_mode_set(uint8_t pwm_lights);
16
17 #endif /*__LED_H__*/
```

ソースファイル led.c (LED 点灯制御関数: 実装)

LED の点灯を行うための以下の 2 つ関数を実装しています.

- led_pwm_steer_mode_set()
  引数に指定されたビットパターンに従い, I/O ピンをデジタル出力ポートあるいは CWG ステアリング出力に設定します.
- led_gpio_set()
  引数に指定されたビットパターンに従い, デジタル出力ポートの出力値を設定します. CWG ステアリング出力と設定されている I/O ピンに対しては LED の表示には反映されません. 反映されるのは I/O ピンがデジタル出力ポートに設定されている間に限ります.

ソースファイル番号 18.3: led.c

```
1 #include <xc.h>
2 #include <stdint.h>
```

```
3  #include "mcr.h"
4  #include "io.h"
5  #include "led.h"
6
7  /*-----------------------------------------------------
8   * PWM出力ピンの変更 (PWMステアリングモードによる)
9   *-----------------------------------------------------*/
10 void led_pwm_steer_mode_set(uint8_t pwm_lights)
11 {
12   if ((pwm_lights & LIGHT_1) != 0) {
13     LED_1_SRC_PWM();
14   } else {
15     LED_1_SRC_GPIO();
16   }
17   if ((pwm_lights & LIGHT_2) != 0) {
18     LED_2_SRC_PWM();
19   } else {
20     LED_2_SRC_GPIO();
21   }
22   if ((pwm_lights & LIGHT_3) != 0) {
23     LED_3_SRC_PWM();
24   } else {
25     LED_3_SRC_GPIO();
26   }
27   if ((pwm_lights & LIGHT_4) != 0) {
28     LED_4_SRC_PWM();
29   } else {
30     LED_4_SRC_GPIO();
31   }
32   if ((pwm_lights & LIGHT_5) != 0) {
33     LED_5_SRC_PWM();
34   } else {
35     LED_5_SRC_GPIO();
36   }
37   if ((pwm_lights & LIGHT_6) != 0) {
38     LED_6_SRC_PWM();
39   } else {
40     LED_6_SRC_GPIO();
41   }
42 }
43
44 /*-----------------------------------------------------
45  * 表示 - GPIOでの点灯
46  *   (PWMステアリングモードでPWM出力にされているピンの出力は
47  *    この関数では反映されない)
48  *-----------------------------------------------------*/
49 void led_gpio_set(uint8_t lpat)
50 {
51     /* Light 1 */
52     if ((lpat & LIGHT_1) != 0) {
53       LED_1_GPIO_ON();
54     } else {
55       LED_1_GPIO_OFF();
56     }
57     /* Light 2 */
58     if ((lpat & LIGHT_2) != 0) {
59       LED_2_GPIO_ON();
60     } else {
61       LED_2_GPIO_OFF();
62     }
63     /* Light 3 */
64     if ((lpat & LIGHT_3) != 0) {
65       LED_3_GPIO_ON();
66     } else {
67       LED_3_GPIO_OFF();
68     }
69     /* Light 4 */
70     if ((lpat & LIGHT_4) != 0) {
71       LED_4_GPIO_ON();
72     } else {
73       LED_4_GPIO_OFF();
74     }
```

```
75      /* Light 5 */
76      if ((lpat & LIGHT_5) != 0) {
77        LED_5_GPIO_ON();
78      } else {
79        LED_5_GPIO_OFF();
80      }
81      /* Light 6 */
82      if ((lpat & LIGHT_6) != 0) {
83        LED_6_GPIO_ON();
84      } else {
85        LED_6_GPIO_OFF();
86      }
87  }
```

ソースファイル aspect.h (現示パターン: ヘッダ)

現示パターンそれぞれにおける LED の記号名を定義します. LED の記号名として LIGHT_1 から LIGHT_6 がすでに定義されていますが, 可読性の向上のために現示パターンごとに更に別名を付けます.

ソースファイル番号 18.4: aspect.h

```
1   #if !defined(__ASPECT_H__)
2   #define __ASPECT_H__
3
4   /* 現示パターン2Aでの色とライト番号の対応関係 */
5   #define AP_2A_YELLOW_1    LIGHT_2
6   #define AP_2A_RED_1       LIGHT_1
7   /* 現示パターン2Bでの色とライト番号の対応関係 */
8   #define AP_2B_GREEN_1     LIGHT_2
9   #define AP_2B_RED_1       LIGHT_1
10  /* 現示パターン3Aでの色とライト番号の対応関係 */
11  #define AP_3A_GREEN_1     LIGHT_3
12  #define AP_3A_YELLOW_1    LIGHT_2
13  #define AP_3A_RED_1       LIGHT_1
14  /* 現示パターン4Aでの色とライト番号の対応関係 */
15  #define AP_4A_YELLOW_1    LIGHT_4
16  #define AP_4A_RED_1       LIGHT_3
17  #define AP_4A_GREEN_1     LIGHT_2
18  #define AP_4A_YELLOW_2    LIGHT_1
19  /* 現示パターン4Bでの色とライト番号の対応関係 */
20  #define AP_4B_YELLOW_1    LIGHT_4
21  #define AP_4B_RED_1       LIGHT_3
22  #define AP_4B_YELLOW_2    LIGHT_2
23  #define AP_4B_GREEN_1     LIGHT_1
24  /* 現示パターン4Cでの色とライト番号の対応関係 */
25  #define AP_4C_YELLOW_1    LIGHT_4
26  #define AP_4C_RED_1       LIGHT_3
27  #define AP_4C_YELLOW_2    LIGHT_2
28  #define AP_4C_GREEN_1     LIGHT_1
29  /* 現示パターン5Aでの色とライト番号の対応関係 */
30  #define AP_5A_YELLOW_1    LIGHT_5
31  #define AP_5A_YELLOW_2    LIGHT_4
32  #define AP_5A_RED_1       LIGHT_3
33  #define AP_5A_YELLOW_3    LIGHT_2
34  #define AP_5A_GREEN_1     LIGHT_1
35  /* 現示パターン5Bでの色とライト番号の対応関係 */
36  #define AP_5B_GREEN_1     LIGHT_5
37  #define AP_5B_GREEN_2     LIGHT_4
38  #define AP_5B_YELLOW_1    LIGHT_3
39  #define AP_5B_RED_1       LIGHT_2
40  #define AP_5B_GREEN_3     LIGHT_1
41  /* 現示パターン5Cでの色とライト番号の対応関係 */
42  #define AP_5C_YELLOW_1    LIGHT_5
43  #define AP_5C_YELLOW_2    LIGHT_4
44  #define AP_5C_RED_1       LIGHT_3
45  #define AP_5C_YELLOW_3    LIGHT_2
46  #define AP_5C_GREEN_1     LIGHT_1
47  /* 現示パターン6Aでの色とライト番号の対応関係 */
```

```
48 #define AP_6A_YELLOW_1   LIGHT_6
49 #define AP_6A_GREEN_1    LIGHT_5
50 #define AP_6A_RED_1      LIGHT_4
51 #define AP_6A_GREEN_2    LIGHT_3
52 #define AP_6A_YELLOW_2   LIGHT_2
53 #define AP_6A_GREEN_3    LIGHT_1
54 /* 現示パターン6Bでの色とライト番号の対応関係 */
55 #define AP_6B_GREEN_1    LIGHT_6
56 #define AP_6B_YELLOW_1   LIGHT_5
57 #define AP_6B_YELLOW_2   LIGHT_4
58 #define AP_6B_RED_1      LIGHT_3
59 #define AP_6B_YELLOW_3   LIGHT_2
60 #define AP_6B_GREEN_2    LIGHT_1
61
62 /* 現示パターンでのデータ規約 */
63 #define AP_INDEX_ALL_OFF  0 /* 添字0: 全消灯データ          */
64 #define AP_INDEX_RED      1 /* 添字1: 「停止」現示データ(赤) */
65
66 /* 現示パターン表 */
67 extern const uint8_t *aspect_patterns[];
68
69 /* 現示パターン数 */
70 #define ASPECT_N_PATTERNS  11
71
72 #endif /*__ASPECT_H__*/
```

ソースファイル aspect.c (現示パターン: 実装)

現示パターンをデータとして表現します. これは図 18.3 と図 18.3 に示した現示パターンをデータとして記述するものです. 以下に示す変数名における *xxx* は現示パターン名に対応し, 例えば現示パターン 3A に対して *xxx* は 3a となります.

- aspect_pattern_*xxx*[0]
  全消灯とします. このようなデータ記述の決まりにすることで, 後述のフリッカー現示の実現が簡単になります.
- aspect_pattern_*xxx*[1]
  停止現示に対応する点灯内容をデータで記述します.
- aspect_pattern_*xxx*[2], ...
  現示内容に対応する点灯内容をデータで記述します. フリッカー現示とする場合には LIGHT_FLICKER ビットを付加します. 最後のデータには目印として LIGHT_END ビットを付加します.

現示パターンの表 aspect_patterns[] の宣言を行い, この表の構成と DIP スイッチの設定を対応づけます.

ソースファイル番号 18.5: aspect.c

```
1  #include <xc.h>
2  #include <stdint.h>
3  #include "led.h"
4  #include "aspect.h"
5
6  /* 現示パターン2Aでの点灯データ列 (モジュール2A使用) */
7  const uint8_t aspect_pattern_2a[] = {
8    0,
9    (AP_2A_RED_1),
10   (AP_2A_YELLOW_1 | LIGHT_END)
11 };
12 /* 現示パターン2Bでの表示データ列 (モジュール2B使用) */
13 const uint8_t aspect_pattern_2b[] = {
14   0,
15   (AP_2B_RED_1),
16   (AP_2B_GREEN_1 | LIGHT_END)
```

```
17  };
18  /* 現示パターン3Aでの表示データ列（モジュール3A使用）*/
19  const uint8_t aspect_pattern_3a[] = {
20    0,
21    (AP_3A_RED_1),
22    (AP_3A_YELLOW_1),
23    (AP_3A_GREEN_1 | LIGHT_END)
24  };
25  /* 現示パターン4Aの表示データ列（モジュール4A使用）*/
26  const uint8_t aspect_pattern_4a[] = {
27    0,
28    (AP_4A_RED_1),
29    (AP_4A_YELLOW_1 | AP_4A_YELLOW_2),
30    (AP_4A_YELLOW_2),
31    (AP_4A_GREEN_1 | LIGHT_END)
32  };
33  /* 現示パターン4Bの表示データ列（モジュール4B使用）*/
34  const uint8_t aspect_pattern_4b[] = {
35    0,
36    (AP_4B_RED_1),
37    (AP_4B_YELLOW_2),
38    (AP_4B_YELLOW_1 | AP_4B_GREEN_1),
39    (AP_4B_GREEN_1 | LIGHT_END)
40  };
41  /* 現示パターン4Cの表示データ列（モジュール4B使用）*/
42  const uint8_t aspect_pattern_4c[] = {
43    0,
44    (AP_4C_RED_1),
45    (AP_4C_YELLOW_2),
46    (AP_4C_YELLOW_1 | AP_4C_GREEN_1),
47    (AP_4C_YELLOW_1 | AP_4C_GREEN_1 | LIGHT_FLICKER),
48    (AP_4C_GREEN_1 | LIGHT_END)
49  };
50  /* 現示パターン5Aの表示データ列（モジュール5A使用）*/
51  const uint8_t aspect_pattern_5a[] = {
52    0,
53    (AP_5A_RED_1),
54    (AP_5A_YELLOW_1 | AP_5A_YELLOW_3),
55    (AP_5A_YELLOW_3),
56    (AP_5A_YELLOW_2 | AP_5A_GREEN_1),
57    (AP_5A_GREEN_1 | LIGHT_END)
58  };
59  /* 現示パターン5Bの表示データ列（モジュール5B使用）*/
60  const uint8_t aspect_pattern_5b[] = {
61    0,
62    (AP_5B_RED_1),
63    (AP_5B_YELLOW_1),
64    AP_5B_GREEN_2,
65    (AP_5B_GREEN_1 | AP_5B_GREEN_3 | LIGHT_END)
66  };
67  /* 現示パターン5Cの表示データ列（モジュール5C使用）*/
68  const uint8_t aspect_pattern_5c[] = {
69    0,
70    (AP_5C_RED_1),
71    (AP_5C_YELLOW_1 | AP_5C_YELLOW_3),
72    (AP_5C_YELLOW_3),
73    (AP_5C_YELLOW_2 | AP_5C_GREEN_1),
74    (AP_5C_YELLOW_2 | AP_5C_GREEN_1 | LIGHT_FLICKER),
75    (AP_5C_GREEN_1 | LIGHT_END)
76  };
77  /* 現示パターン6Aの表示データ列（モジュール6A使用）*/
78  const uint8_t aspect_pattern_6a[] = {
79    0,
80    (AP_6A_RED_1),
81    (AP_6A_YELLOW_2),
82    (AP_6A_YELLOW_1 | AP_6A_GREEN_2),
83    (AP_6A_GREEN_2),
84    (AP_6A_GREEN_1 | AP_6A_GREEN_3 | LIGHT_END)
85  };
86  /* 現示パターン6Bの表示データ列（モジュール6B使用）*/
87  const uint8_t aspect_pattern_6b[] = {
```

```
88      0,
89      (AP_6B_RED_1),
90      (AP_6B_YELLOW_1 | AP_6B_YELLOW_3),
91      (AP_6B_YELLOW_3),
92      (AP_6B_YELLOW_2 | AP_6B_GREEN_2),
93      (AP_6B_GREEN_2),
94      (AP_6B_GREEN_1 | AP_6B_GREEN_2 | LIGHT_END)
95  };
96
97  /* 現示パターン表 */
98  const uint8_t *aspect_patterns[ASPECT_N_PATTERNS] = {
99      /*       DIPスイッチ    1   2   3   4  */
100     aspect_pattern_2a, /* OFF OFF OFF OFF */
101     aspect_pattern_2b, /* ON  OFF OFF OFF */
102     aspect_pattern_3a, /* OFF ON  OFF OFF */
103     aspect_pattern_4a, /* ON  ON  OFF OFF */
104     aspect_pattern_4b, /* OFF OFF ON  OFF */
105     aspect_pattern_4c, /* ON  OFF ON  OFF */
106     aspect_pattern_5a, /* OFF ON  ON  OFF */
107     aspect_pattern_5b, /* ON  ON  ON  OFF */
108     aspect_pattern_5c, /* OFF OFF OFF ON  */
109     aspect_pattern_6a, /* ON  OFF OFF ON  */
110     aspect_pattern_6b, /* OFF ON  OFF ON  */
111 };
```

ソースファイル Signal-TimerBased.c (主ソースコード)

このソースファイル中のメイン関数 main() が主たる動作を行います. おおよその動作は先述の通りですので繰り返しませんが, フリッカー現示および強制停止現示の実現を以下に簡単に説明します. フリッカー現示は関数 main() 内に以下の原理に基づく有限状態機械で実現します.

- 変数 state_flicker
  フリッカー現示の状態を保持します. 状態 0 で消灯を, 状態 1 で点灯とします.
- 状態遷移
  時間の経過による状態遷移を行い, 状態 0 と状態 1 を交互に遷移します. 変数 timer_flicker を使用して時間の計測を行います. 関数 main() のループ 1 周期は 25 ミリ秒ですから, これは 40Hz となります. 15 周期で 1 回の状態遷移をすると, 1 分間には $60 \times 40/15 = 160$ 回の状態遷移が発生します. 2 回の状態遷移で 1 回の点滅ですから, 1 分間に 80 回の点滅をすることになります.
- 消灯と点灯
  aspect_pattern_*xxx*[] の添字 0 の内容と ap_index の内容の点灯を交互に行います. 添字 0 の内容は全消灯のデータですので, 状態遷移の度に点灯・消灯が交互に行われます.

強制停止現示は以下の原理により実現しています.

- 強制停止現示の入力がある場合
  aspect_pattern_*xxx*[] の添字 1 の内容で点灯を行います.
- 強制停止現示の入力がない場合
  ap_index の内容で点灯します.

点灯内容が切り替わった場合には LED 点灯用有限状態を動作させて, 電球風点灯などを実現します.

ソースファイル番号 18.6: Signal-TimerBased.c

```
1   /*********************************************
2    * タイマ動作型の色灯式信号機 (2~6灯) / PIC16F18346
3    * by Hirotsugu Kakugawa
```

```
4   ************************************************/
5   #include <xc.h>
6   #include <stdint.h>
7   #include "mcr.h"
8   #include "adc.h"
9   #include "io.h"
10  #include "aspect.h"
11  #include "light-fsm.h"
12  #include "led.h"
13  #include "incr-exp.h"
14
15  /* ------------------------------------------------------ */
16
17  /* 周期実行間隔: 25 [mSec] (=1/40 [Sec]) */
18  #define RUN_INTERVAL_MSEC  25
19
20  /* フリッカー点灯での点滅間隔: 15 (15*2*25=0.75) */
21  /*   点滅回数は 80 [回/分] = 80/60 [回/Sec] なので
22   *   1周期は 60/80 = 0.75 [Sec] */
23  #define FLICKER_INTERVAL   15
24
25  /* スイッチで選択した信号の灯式の番号
26   * (aspect_patterns[i][j]の添字i) */
27  uint8_t ap_number = 0;
28
29  /* タイマ動作に基づく信号の現示
30   * (aspect_patterns[i][j]の添字j) */
31  uint8_t ap_index = 0;
32
33  /* 実際の表示内容 (ap_indexに強制停止現示やフリッカーを加味) */
34  uint8_t displayed_lpi = 0;
35
36  static uint8_t  sw_aspect_pattern_number(void);
37  static void     flush_led(void);
38  static uint16_t get_aspect_change_interval(void);
39
40  /*-----------------------------------------------------------
41   * MAIN
42   *-----------------------------------------------------------*/
43  void main(void)
44  {
45    /* システム初期化 */
46    System_Init();
47    /* ウオッチドッグタイマ初期化: 2 [Sec] */
48    WDTCONbits.WDTPS = 0x0B; /* 0b01011 */
49    CLRWDT();
50    /* 現示切替タイマ(切替までの残り周期回数) */
51    uint16_t timer_aspect = 0;
52    /* フリッカー点灯用タイマ(切替までの残り周期回数) */
53    uint8_t  timer_flikcer = 0;
54    /* フリッカー点灯状態 (0: 消灯, 1: 点灯) */
55    int8_t   state_flicker = 0;
56    /* 信号の灯式の番号(スイッチでの選択)読み取り */
57    for (;;) {
58      ap_number = sw_aspect_pattern_number();
59      /* 登録済みの現示パターンとなるまではLED点滅してエラー表示 */
60      if (ap_number < ASPECT_N_PATTERNS) {
61        break;
62      }
63      flush_led();
64    }
65    /* 全LEDをGPIO制御で消灯 */
66    led_gpio_set(0x00);
67    led_pwm_steer_mode_set(0x00);
68    led_gpio_set(0x00);
69    /* ライト輝度制御用の状態機械を初期化 */
70    light_fsm_init();
71    /* 「進行」現示のap_indexを探す (LIGHT_END印) */
72    ap_index = 0;
73    while ((aspect_patterns[ap_number][ap_index]
```

```
74            & LIGHT_END) == 0) {
75        ap_index++;
76      }
77      displayed_lpi = ap_index;
78      /* 進行現示 */
79      led_gpio_set(aspect_patterns[ap_number][displayed_lpi]);
80      /* 主ループ */
81      for (;;) {
82        __delay_ms(RUN_INTERVAL_MSEC);
83        CLRWDT();
84        /* 信号の灯式の番号の変更があればリセットして再始動 */
85        if (ap_number != sw_aspect_pattern_number()) {
86          RESET();
87        }
88        /* センサによる車両検出 */
89        if (CAR_DETECT_PORT() || CAR_DETECT_IOC()) {
90          CAR_CLEAR_IOC();
91          /* 現示を「停止」に変更 */
92          ap_index = AP_INDEX_RED;
93          /* 次の現示切替の時間にむけて時間計測開始 */
94          timer_aspect = get_aspect_change_interval();
95        }
96        /* 現示を一定時間間隔で切替 */
97        if ((aspect_patterns[ap_number][ap_index] & LIGHT_END) == 0) {
98          if (timer_aspect != 0) {
99            timer_aspect--;
100         } else { /* 現示切替実行 */
101           /* 次の現示に変更 */
102           ap_index++;
103           /* フリッカー現示用の状態機械の制御 */
104           if ((aspect_patterns[ap_number][ap_index]
105                & LIGHT_FLICKER) != 0) {
106             /* フリッカ点灯用を開始 */
107             state_flicker = 1;
108             timer_flikcer = FLICKER_INTERVAL;
109           }
110           /* 次の現示切替の時に向けて時間計測を再始動 */
111           timer_aspect = get_aspect_change_interval();
112         }
113       }
114       /* 次の表示内容を displayed_lpi_next に求める */
115       uint8_t displayed_lpi_next = ap_index;
116       /* フリッカー処理  */
117       if (timer_flikcer != 0) {
118         timer_flikcer--;
119       } else {
120         /* 点灯状態の切替 */
121         state_flicker = 1 - state_flicker; /* 0 <-> 1 */
122         timer_flikcer = FLICKER_INTERVAL;
123       }
124       if (((aspect_patterns[ap_number][displayed_lpi_next]
125             & LIGHT_FLICKER) != 0) &&
126           (state_flicker == 0)) {
127         /* 消灯 */
128         displayed_lpi_next = AP_INDEX_ALL_OFF;
129       }
130       /* 強制停止現示（出発禁止,ポイント不開通等） */
131       if (FORCE_RED()) {
132         /*  「停止」を現示 */
133         displayed_lpi_next = AP_INDEX_RED;
134       }
135       /* ライト輝度制御用の状態機械 */
136       if (light_fsm_is_idle() &&
137           (displayed_lpi != displayed_lpi_next)) {
138         /* 現在の点灯から変更有: 表示変更を開始 */
139         light_fsm_start(displayed_lpi, displayed_lpi_next);
140         displayed_lpi = displayed_lpi_next;
141       }
142       /* 現示切替/輝度制御用の状態機械 */
143       if (SW_LIGHT_MODE_BULB()) { /* 電球(BULB)風 */
```

```
144         light_fsm_bulb_run();
145       } else {                      /* LED風 */
146         light_fsm_led_run();
147       }
148     }
149   }
150
151   /*-----------------------------------------------------------
152    * 信号現示パターン選択用の設定スイッチを読む
153    *-----------------------------------------------------------*/
154   static uint8_t sw_aspect_pattern_number(void)
155   {
156     uint8_t apn = 0;
157     /* 設定スイッチそれぞれの重みを加算 */
158     if (SW_PATTERN_NUMBER_1()) {
159       apn += 1;
160     }
161     if (SW_PATTERN_NUMBER_2()) {
162       apn += 2;
163     }
164     if (SW_PATTERN_NUMBER_3()) {
165       apn += 4;
166     }
167     if (SW_PATTERN_NUMBER_4()) {
168       apn += 8;
169     }
170     return apn;
171   }
172
173   /* LEDを1回点滅(エラー表示用) */
174   static void flush_led(void)
175   {
176     led_pwm_steer_mode_set(0x00);
177     CLRWDT();
178     led_gpio_set(0x3F); /*点灯*/
179     __delay_ms(900);
180     led_gpio_set(0x00); /*消灯*/
181     __delay_ms(100);
182   }
183
184   /*-----------------------------------------------------------
185    * 現示切替の時間間隔をボリューム設定より得る
186    *   タイマの1周期が1単位時間
187    *-----------------------------------------------------------*/
188   static uint16_t get_aspect_change_interval(void)
189   {
190     /* 現示切替時間の計算
191      *    タイマ1周期 T = RUN_INTERVAL = 25 [mSec]
192      *   (40 + x/2) * T [mSec],  0 <= x <= 1023
193      *     = 1.0 [Sec]  ..  13.8 [Sec]
194      */
195     uint16_t t = ADC_Read(ADC_CH_TIME);
196     return (uint16_t) 40 + (t >> 1);
197   }
```

ソースファイル light-fsm.h (点灯用有限状態機械: ヘッダ)

LED の点灯を制御する関数のプロトタイプ宣言です.

ソースファイル番号 18.7: light-fsm.h

```
1   #if !defined(__LIGHT_FSM_H__)
2   #define __LIGHT_FSM_H__
3
4   extern void   light_fsm_init(void);
5   extern int8_t light_fsm_is_idle(void);
6   extern void   light_fsm_start(int8_t disp_lpi, int8_t disp_lpi_next);
7   extern void   light_fsm_led_run(void);
8   extern void   light_fsm_bulb_run(void);
```

```
9
10 #endif /*__LIGHT_FSM_H__*/
```

ソースファイル light-fsm.c（点灯用有限状態機械: 実装）

LED の点灯を制御する関数の実装です. 信号の現示が変化したときには, DIP スイッチの設定に従って LED 風に, あるいは電球風に LED の点灯・消灯を行います. I/O ピンをステアリング出力あるいはデジタル出力ポートに切り替えるといった技巧的な部分が含まれているのがこの関数です. 詳しくはソースファイルの後で説明をします.

ソースファイル番号 18.8: light-fsm.c

```
1  /*-----------------------------------------------------
2   * 表示切り替え管理用の状態機械
3   * （電球風点灯・消灯を実現）
4   *-----------------------------------------------------*/
5  #include <xc.h>
6  #include <stdint.h>
7  #include "mcr.h"
8  #include "io.h"
9  #include "light-fsm.h"
10 #include "led.h"
11 #include "incr-exp.h"
12
13 /* 状態 */
14 #define LIGHT_FSM_STATE_IDLE   0
15 #define LIGHT_FSM_STATE_START  1
16
17 /* LED輝度の最大値 */
18 #define LED_INTENSITY_MAX        1023
19
20 /* 点灯・消灯の時定数(「じわっと点灯・消灯」) */
21 #define LIGHT_TIME_CONSTANT_ON      6
22 #define LIGHT_TIME_CONSTANT_OFF     3
23
24 uint8_t light_fsm_state = LIGHT_FSM_STATE_IDLE; /*状態*/
25 uint8_t light_fsm_tmr = 0;                      /*タイマ*/
26 uint8_t light_fsm_lpat_curr; /*今の点灯内容*/
27 uint8_t light_fsm_lpat_next; /*次の灯内内容*/
28 uint8_t light_fsm_lpat_off;  /*消灯すべきライト（今点灯しているもの）*/
29 uint8_t light_fsm_lpat_on;   /*点灯すべきライト（次に点灯すべきもの）*/
30
31 extern const uint8_t *aspect_patterns[];
32 extern uint8_t ap_number;
33 extern uint8_t ap_index;
34
35 /* 状態機械の初期化 */
36 void light_fsm_init(void)
37 {
38   light_fsm_state = LIGHT_FSM_STATE_IDLE;
39 }
40
41 /* 状態機械はアイドル状態か否か */
42 int8_t light_fsm_is_idle(void)
43 {
44   return (light_fsm_state == LIGHT_FSM_STATE_IDLE);
45 }
46
47 /* 状態機械の動作開始 */
48 void light_fsm_start(int8_t disp_lpi, int8_t disp_lpi_next)
49 {
50   /* 状態機械を動作開始 */
51   light_fsm_state = LIGHT_FSM_STATE_START;
52   /* 灯内内容 */
53   light_fsm_lpat_curr
54     = aspect_patterns[ap_number][disp_lpi];
```

```
55    light_fsm_lpat_next
56      = aspect_patterns[ap_number][disp_lpi_next];
57    light_fsm_lpat_off
58      = light_fsm_lpat_curr & (unsigned)~light_fsm_lpat_next;
59    light_fsm_lpat_on
60      = light_fsm_lpat_next & (unsigned)~light_fsm_lpat_curr;
61  }
62
63  /* LED風の有限状態機械: すぐに瞬時に点灯・消灯 */
64  void light_fsm_led_run(void)
65  {
66    switch (light_fsm_state) {
67    default:
68    case LIGHT_FSM_STATE_IDLE:
69      break;
70    case LIGHT_FSM_STATE_START:
71      led_gpio_set(light_fsm_lpat_next);
72      light_fsm_state = LIGHT_FSM_STATE_IDLE;
73      break;
74    }
75  }
76
77  /* 電球風の有限状態機械: じわっと点灯・消灯 */
78  void light_fsm_bulb_run(void)
79  {
80    uint16_t led_intensity = 0;
81    switch (light_fsm_state) {
82    default:
83    case LIGHT_FSM_STATE_IDLE:
84      break;
85    case LIGHT_FSM_STATE_START:
86      /* PWM制御による「じわっと消灯」開始  */
87      LED_PWM(LED_INTENSITY_MAX); /* PWMによる輝度設定 */
88      /* 消灯LEDの入力をPWMへ */
89      led_pwm_steer_mode_set(light_fsm_lpat_off);
90      incr_exp_init(LED_INTENSITY_MAX);
91      light_fsm_state = 2;
92      break;
93    case 2: /* PWM制御による「じわっと消灯」継続中  */
94      led_intensity
95        = incr_exp_next(LIGHT_TIME_CONSTANT_OFF, 0);
96      LED_PWM(led_intensity); /* PWMによる輝度設定 */
97      if (led_intensity < 4) {  /* 消灯完了(にする) */
98        /* PWM制御による「じわっと消灯」完了 */
99        LED_PWM(0);
100       light_fsm_tmr = 5;  /* 空白時間の設定 */
101       light_fsm_state = 3;
102     }
103     break;
104   case 3: /* 消灯と点灯間の空白時間 */
105     if (light_fsm_tmr != 0) { /* 空白時間中 */
106       light_fsm_tmr--;
107     } else {                  /* 空白時間完了 */
108       light_fsm_state = 4;
109     }
110     break;
111   case 4: /* PWM制御による「じわっと点灯」開始  */
112     /* 出力ピンをPWMモードへ：PWMによる輝度設定 */
113     LED_PWM(0);
114     /* 点灯LEDの入力をPWMへ */
115     led_pwm_steer_mode_set(light_fsm_lpat_on);
116     led_gpio_set(light_fsm_lpat_next);
117     incr_exp_init(0);
118     light_fsm_state = 5;
119     break;
120   case 5: /* PWM制御による「じわっと点灯」継続中  */
121     led_intensity
122       = incr_exp_next(LIGHT_TIME_CONSTANT_ON, LED_INTENSITY_MAX);
123     LED_PWM(led_intensity); /* PWMによる輝度変更 */
124     if (led_intensity > LED_INTENSITY_MAX-4) { /* 点灯完了(にする) */
```

```
125         /* PWM制御による「じわっと点灯」完了 */
126         LED_PWM(LED_INTENSITY_MAX);
127         /* 出力ピンをデジタルモード: 0/1で輝度設定 */
128         led_pwm_steer_mode_set(0x00);
129         light_fsm_state = LIGHT_FSM_STATE_IDLE;
130       }
131       break;
132     }
133   }
```

主な関数は以下の 3 つです.

- light_fsm_start()
  有限状態機械を初期化します. この関数への引数については後述します. LED 風点灯, 電球風点灯どちらの場合でもこの関数で初期化します.
- light_fsm_led_run()
  LED 風点灯を行う有限状態機械を 1 ステップ動作させます.
- light_fsm_bulb_run()
  電球風点灯を行う有限状態機械を 1 ステップ動作させます.

関数 light_fsm_start() の引数には, 今の点灯内容と次の点灯パターンのふたつが与えられます. そして関数 light_fsm_start() はこれら 2 つの引数から以下の 4 つの変数に値を設定します.

- 変数 light_fsm_lpat_curr
  今の点灯内容を表します.
- 変数 light_fsm_lpat_next
  次の点灯内容を表します.
- 変数 light_fsm_lpat_off
  今の点灯内容では点灯していて, 次の点灯内容では消灯となる LED を表します.
- 変数 light_fsm_lpat_on
  今の点灯内容では消灯していて, 次の点灯内容では点灯となる LED を表します.

LED 風点灯を行う有限状態機械 light_fsm_led_run() では, 次の灯内内容を表す変数 light_fsm_lpat_next の値をデジタル出力ポートへ直ちに出力し, LED の点灯を切り替えます. I/O ピンは初期化の際にデジタル出力ポートに設定されていることに注意して下さい.

電球風点灯を行う有限状態機械 light_fsm_bulb_run() では, 図 18.10 と図 18.11 で概略を示した技巧的な動作をします. 動作内容の詳しい説明は以下の通りです.

1. 消灯すべき LED の消灯 (変数 light_fsm_lpat_off で指定の LED の消灯)
   今回新たに消灯する LED の I/O ピンを CWG ステアリング出力に, 点灯あるいは消灯を継続する LED の I/O ピンをデジタル出力ポートに, それぞれ切り替えます. そしてブレゼンハム指数アルゴリズムを使用して, CWG ステアリング出力を PWM デューティ比 100% から 0% に減少させることで輝度を 100% から 0% まで減光・消灯します.
2. 空白時間
3. 点灯すべき LED の点灯 (変数 light_fsm_lpat_on で指定の LED の点灯)
   最初に, 点灯の LED に対応するデジタル出力ポートに 1 を, 消灯の LED には 0 をそれぞれ設定し, デジタル出力ポートに切り替えられたときに点灯内容に合致したデジタル出力となるように予め設定してお

きます. 次に, 消灯から点灯に変化する LED の I/O ピンを CWG ステアリング出力に切り替え, 消灯あるいは点灯を継続する LED の I/O ピンをデジタル出力ポートに切り替えます. そしてブレゼンハム指数アルゴリズムを使用して, 輝度 0% から輝度 100% まで増光・点灯します.

4. 点灯動作の完了
   輝度が 100% になれば点灯は完了で, LED の I/O ピンをすべてデジタル出力ポートに切り替えます. このとき予め設定したデジタル出力ポートの内容が出力されますが, 上述の通りその点灯内容は変数 light_fsm_lpat_next の値になります.

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 18.9: mcr.h

```
1  #if !defined(__MCR_H__)
2  #define __MCR_H__
3
4  #define _XTAL_FREQ 32000000 /* 32MHz */
5
6  extern void System_Init(void);
7  extern void CCP1_Init(void);
8  extern void CCP1_SetDutyCycle(uint16_t duty);
9  extern void CWG1_Init(void);
10 extern void CWG2_Init(void);
11 extern void TMR2_Init(void);
12
13 #endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います.

ソースファイル番号 18.10: mcr.c

```
1  /*-----------------------------------------------------
2   * MCR設定&周辺モジュール (PIC16F18346)
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "adc.h"
8
9  /*-------------------------------------------*/
10 /* MCR初期化 */
11 void System_Init(void)
12 {
13   /* 発振器 (FOSC=32MHz) */
14   OSCCON1 = 0x00; /* HFINTOSC & 2x PLL */
15   OSCCON3 = 0x00;
16   OSCEN   = 0x00;
17   OSCFRQ  = 0x06; /* 32MHz */
18   OSCTUNE = 0x00;
19   while (PLLR == 0) {
20     ;
21   }
22   /* I/Oピン */
23   LATA   = 0x00;
24   LATB   = 0x00;
25   LATC   = 0x00;
26   TRISA  = 0x3F; /* (RA5,RA4),RA3,(RA2,RA1),RA0 */
27   TRISB  = 0x80; /* RB7 */
28   TRISC  = 0xF8; /* RC7,RC6,RC5,RC4,RC3 */
29   ANSELA = 0x01; /* ANA0 */
```

```
30    ANSELB = 0x00;
31    ANSELC = 0x00;
32    WPUA   = 0x36; /* (RA5,RA4),(RA2,RA1) */
33    WPUB   = 0x00;
34    WPUC   = 0x00;
35    ODCONA = 0x00;
36    ODCONB = 0x00;
37    ODCONC = 0x00;
38    IOCBFbits.IOCBF7 = 0;
39    IOCBNbits.IOCBN7 = 1;
40    IOCBPbits.IOCBP7 = 0;
41    PIE0bits.IOCIE = 0;
42    int8_t ogie = GIE;
43    GIE = 0;
44    PPSLOCK = 0x55;
45    PPSLOCK = 0xAA;
46    PPSLOCKbits.PPSLOCKED = 0x00;
47    RB6PPS = 0x08;   /* RB6 <- CWG1A */
48    RB5PPS = 0x09;   /* RB5 <- CWG1B */
49    RB4PPS = 0x0A;   /* RB4 <- CWG1C */
50    RC2PPS = 0x0B;   /* RC2 <- CWG1D */
51    RC1PPS = 0x10;   /* RC1 <- CWG2A */
52    RC0PPS = 0x11;   /* RC0 <- CWG2B */
53    PPSLOCK = 0x55;
54    PPSLOCK = 0xAA;
55    PPSLOCKbits.PPSLOCKED = 0x01;
56    GIE = ogie;
57    /* 周辺モジュール */
58    ADC_Init();
59    CCP1_Init();
60    TMR2_Init();
61    CWG1_Init();
62    CWG2_Init();
63  }
64
65  /*---------------------------------------------*/
66  /* CCP1 */
67  void CCP1_Init(void)
68  {
69    CCP1CONbits.CCP1EN   = 1;   /* Enabled */
70    CCP1CONbits.CCP1FMT  = 0;   /* 右寄せ形式 */
71    CCP1CONbits.CCP1MODE = 0xF; /* PWMモード */
72    CCPTMRSbits.C1TSEL   = 0x1; /* TMR2 */
73    CCPR1L = 0x00;
74    CCPR1H = 0x00;
75  }
76
77  void CCP1_SetDutyCycle(uint16_t duty)
78  {
79    CCPR1H = duty >> 8; /* 2 MSBs */
80    CCPR1L = duty;      /* 8 LSBs */
81  }
82
83  /*---------------------------------------------*/
84  /* CWG1 */
85  void CWG1_Init(void)
86  {
87    CWG1CON0bits.EN = 0; /* Disabled */
88    CWG1CON0bits.MODE = 0; /* 非同期ステアリングモード */
89    CWG1CON1bits.POLA = 0; /* Normal Polarity */
90    CWG1CON1bits.POLB = 0; /* Normal Polarity */
91    CWG1CON1bits.POLC = 0; /* Normal Polarity */
92    CWG1CON1bits.POLD = 0; /* Normal Polarity */
93    CWG1CLKCONbits.CS = 0; /* Clock source: FOSC */
94    CWG1DATbits.DAT = 0x3; /* Data Source: CCP1 */
95    CWG1STRbits.OVRA = 0;
96    CWG1STRbits.OVRB = 0;
97    CWG1STRbits.OVRC = 0;
98    CWG1STRbits.OVRD = 0;
99    CWG1STRbits.STRA = 0;
100   CWG1STRbits.STRB = 0;
101   CWG1STRbits.STRC = 0;
```

```
102     CWG1STRbits.STRD = 0;
103     CWG1CON0bits.EN = 1; /* Enabled */
104   }
105
106   /*-------------------------------------------*/
107   /* CWG2 */
108   void CWG2_Init(void)
109   {
110     CWG2CON0bits.EN = 0; /* Disabled */
111     CWG2CON0bits.MODE = 0; /* 非同期ステアリングモード */
112     CWG2CON1bits.POLA = 0; /* Normal Polarity */
113     CWG2CON1bits.POLB = 0; /* Normal Polarity */
114     CWG2CLKCONbits.CS = 0; /* Clock source: FOSC */
115     CWG2DATbits.DAT = 0x3; /* Data Source: CCP1 */
116     CWG2STRbits.OVRA = 0;
117     CWG2STRbits.OVRB = 0;
118     CWG2STRbits.STRA = 0;
119     CWG2STRbits.STRB = 0;
120     CWG2CON0bits.EN = 1; /* Enabled */
121   }
122
123   /*-------------------------------------------*/
124   /* TMR2 */
125   /* LED輝度のPWM制御用のタイマ
126    *   Prescaler: 1:1, Postscaler: 1:1, PR Match Value: 0xFF
127    *   1周期 32.0 [uSec]
128    */
129   void TMR2_Init(void)
130   {
131     T2CONbits.TMR2ON = 0; /* Disabled */
132     T2CONbits.T2OUTPS = 0x0; /* Postscaler: 1:1 */
133     T2CONbits.T2CKPS  = 0x0; /* Prescaler:  1:1 */
134     PR2  = 0xFF;
135     TMR2 = 0x00;
136     TMR2IF = 0;
137     T2CONbits.TMR2ON = 1; /* Enabled */
138   }
139
140   /*-------------------------------------------*/
141   /* CONFIG1 */
142   #pragma config FEXTOSC  = OFF
143   #pragma config RSTOSC   = HFINT32
144   #pragma config CLKOUTEN = OFF
145   #pragma config CSWEN    = ON
146   #pragma config FCMEN    = ON
147   /* CONFIG2 */
148   #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
149   #pragma config PWRTE    = OFF
150   #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
151   #pragma config LPBOREN  = OFF
152   #pragma config BOREN    = ON
153   #pragma config BORV     = LOW
154   #pragma config PPS1WAY  = ON
155   #pragma config STVREN   = ON
156   #pragma config DEBUG    = OFF
157   /* CONFIG3 */
158   #pragma config WRT      = OFF
159   #pragma config LVP      = OFF /* 低電圧書込不可 */
160   /* CONFIG4 */
161   #pragma config CP       = OFF
162   #pragma config CPD      = OFF
```

課題

1. 本章では汎用化した共通ハードウエアにより, 2 灯式から 6 灯式までの信号機を使用できるようにしています. 次は逆に特定化をして下さい. つまり 2 灯式から 6 灯式までの信号機それぞれに個別対応したハードウエアとファームウエアを設計・制作して下さい. 使用するマイコンや LED 出力線などを最小限度にし, DIP スイッチは不使用にするなど, ハードウエアは最小限にして下さい. ただしファームウエア

は最大限の共通化を行うことで開発・保守の手間を低減して下さい. ファームウエアの共通化の工夫がこの課題の目的です.

2. 第 18.3 節 (297 ページ) では動作仕様のデータ記述化について説明し, そこでファームウエアの第 1 案を示しました. この構造に従ったファームウエアを書いてみて下さい. 灯式ごと個別のプログラムロジックで記述するのは当然ですが, プログラムロジックでの記述を容易にするよう, 点灯の切り替え部分や一定時間の待機部分をすべての灯式で共通化して下さい. 特に, フリッカー現示, 電球風の点灯のシミュレート, および停止現示入力の機能を持たせながらも, 明瞭なプログラムロジックで記述する方法を考え出して下さい.

第 19 章

線路電圧極性検出器

本章では線路に印加されている電圧の極性を検出する装置を紹介します. 直流 (DC) アナログ制御方式の鉄道模型では電圧の極性により模型車両の走行方向が決まることを第 2 章で説明しました. そこで本装置により模型車両の走行方向を判別できます. 本装置の検出結果は 2 系統のアクセサリ制御出力に出力されます. 例えば第 18 章で紹介した信号機の停止現示強制入力に本装置を接続すると, 模型車両の走行方向が逆の場合には信号機は常に停止現示となります. 本章で紹介する回路とファームウエアの特徴は以下の通りです.

- 確実な電圧極性の検出
  PWM 方式のパワーパックのわずかなパルスでも極性を検出します.
- 2 系統のアクセサリ制御出力を装備
  前進, 後進のそれぞれに対応するアクセサリ制御出力を有します.
- 保守的/楽観的の 2 通りの極性検出の方法を選択可能
  正方向と逆方向のどちらにも電圧の検出がない場合の出力を設定可能です.

本章で学ぶ内容は以下の通りです.

- PIC マイコンのコンパレータモジュール CM
- PIC マイコンの固定参照電圧モジュール FVR
- フォトカプラによる絶縁インターフェース

この装置のファームウエアの大まかな構造を図 19.4 に示します. 検出判定にはノイズ除去をした検出の機能を, 検出結果の出力には DIP スイッチによる設定選択を, それぞれソフトウエアにより実現します.

19.1 使用方法

本装置の使用方法を以下に説明します. 以下の説明で使用する部品名称は図 19.2 に部品配置例とともに示しています.

1. 本装置の TRACK-IN と線路を図 19.3 に示すように接続します. 線路に接続する代わりにパワーパックの出力を二分岐して接続するのでも構いません.
2. 検出出力 DET-FWD と DET-REV を使用する装置に接続します. DET-FWD は正方向, DET-REV は逆方向の検出結果をそれぞれ出力します.
3. DIP スイッチ CONF の設定を行います. DIP スイッチは 2 回路あり, それぞれ後述の説明に従って設定します.
4. 5V の AC アダプタを DC ジャックに差し込みます. **間違って 12V 出力の AC アダプタを使用すると回路が破損する場合がありますので, 十分に注意して下さい.**
5. 線路電圧が正の場合は緑 LED FWD が点灯し, 負の場合は赤 LED REV が点灯します.

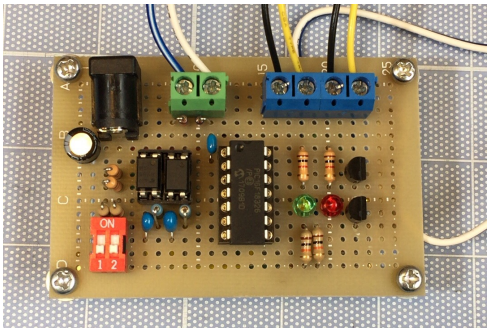


図 19.1: 線路電圧極性検出器

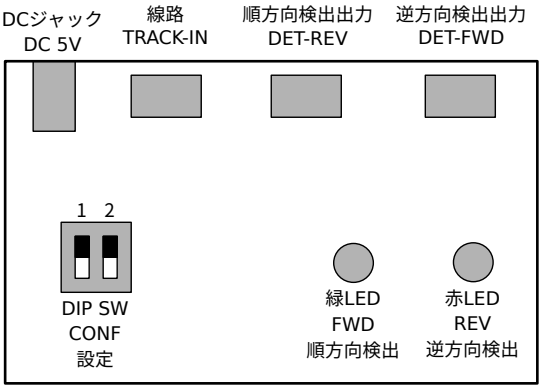


図 19.2: 操作表示部品の配置例と名称

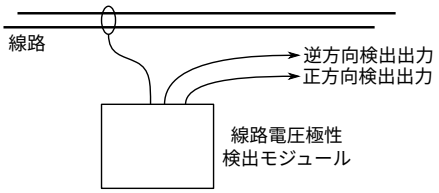


図 19.3: 接続方法

```
1  main()
2  {
3    for(;;) {
4      コンパレータ出力を読む;
5      検出判定;
6      判定結果を出力;
7    }
8  }
```

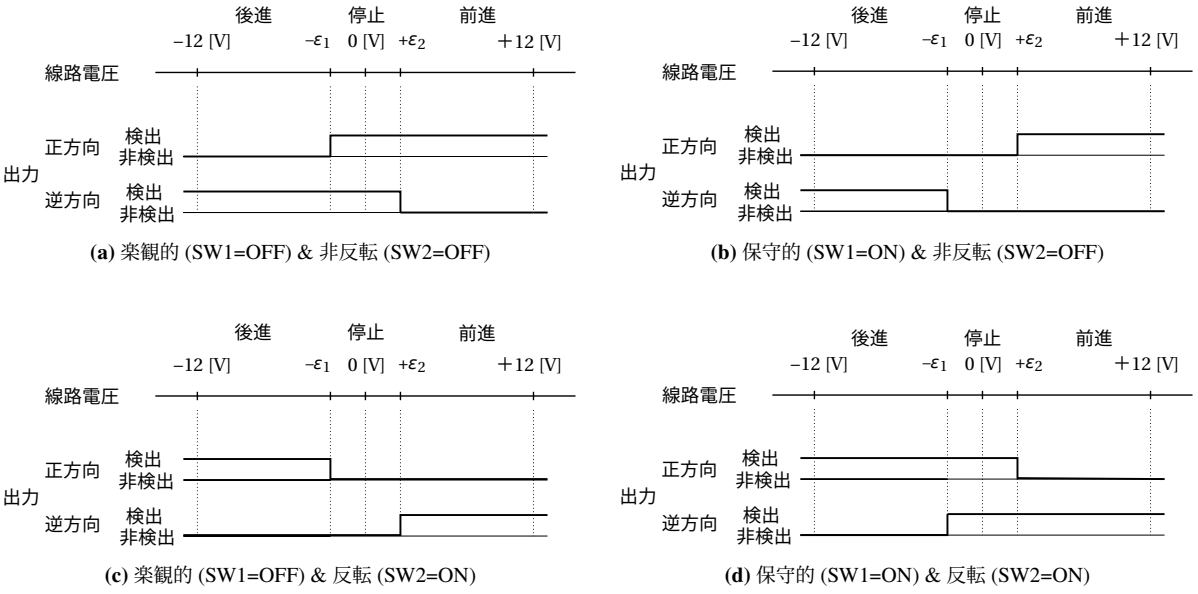
図 19.4: ファームウエア構造 (概要)

DIP スイッチ設定を表 19.1 に示し, その詳細は以下の通りです. この線路電圧と出力との関係を図 19.5 に示します.

- SW1: 出力の決定法の設定
  線路電圧が低く極性が正方向とも逆方向とも判別つかない場合の出力を選択します.
  - 楽観的出力 (SW1=OFF)
    正方向検出出力 DET-FWD と逆方向検出出力 DET-REV ともに検出を出力します. 反対方向の検出がない限りは検出とする楽観的な決定方法です.
  - 保守的出力 (SW1=ON)
    正方向検出出力 DET-FWD と逆方向検出出力 DET-REV ともに非検出を出力します. 順方向の検出

表 19.1: DIP スイッチによるの設定

DIP スイッチ		設定内容
SW1	SW2	
OFF	∗	楽観的出力
ON	∗	保守的出力
∗	OFF	非反転出力
∗	ON	反転出力



(a) 楽観的 (SW1=OFF) & 非反転 (SW2=OFF)

(b) 保守的 (SW1=ON) & 非反転 (SW2=OFF)

(c) 楽観的 (SW1=OFF) & 反転 (SW2=ON)

(d) 保守的 (SW1=ON) & 反転 (SW2=ON)

図 19.5: 線路電圧と検出結果 (ε_1 と ε_2 は正の小さな値でハードウエア依存)

がない限りは検出としない悲観的な決定方法です.

- SW2: 反転出力の設定
  正方向検出出力 DET-FWD と逆方向検出出力 DET-REV それぞれで, 出力の反転を行います. つまり L の場合には HiZ に, HiZ の場合には L に反転します.
  - 非反転出力 (SW2=OFF)
    検出結果をそのまま出力します.
  - 反転出力 (SW2=ON)
    検出結果を反転して出力します.

19.2　回路図

線路電圧極性検出器の回路図を図 19.6 に, 部品リストを表 19.2 に, それぞれ示します. 線路に印加される電圧の極性の検出にはフォトカプラを用います. 線路側と本装置側はフォトカプラにより電気的に絶縁されますので, 別々の電源系統を使用できます.

フォトカプラの入力側は線路に接続して, 流れる電流は最大 10mA とします. この際に極性に関わらず電流の制限を行うために CRD を 2 つ対向して直列接続します. また 2 つのフォトカプラの入力側を相互に反対に

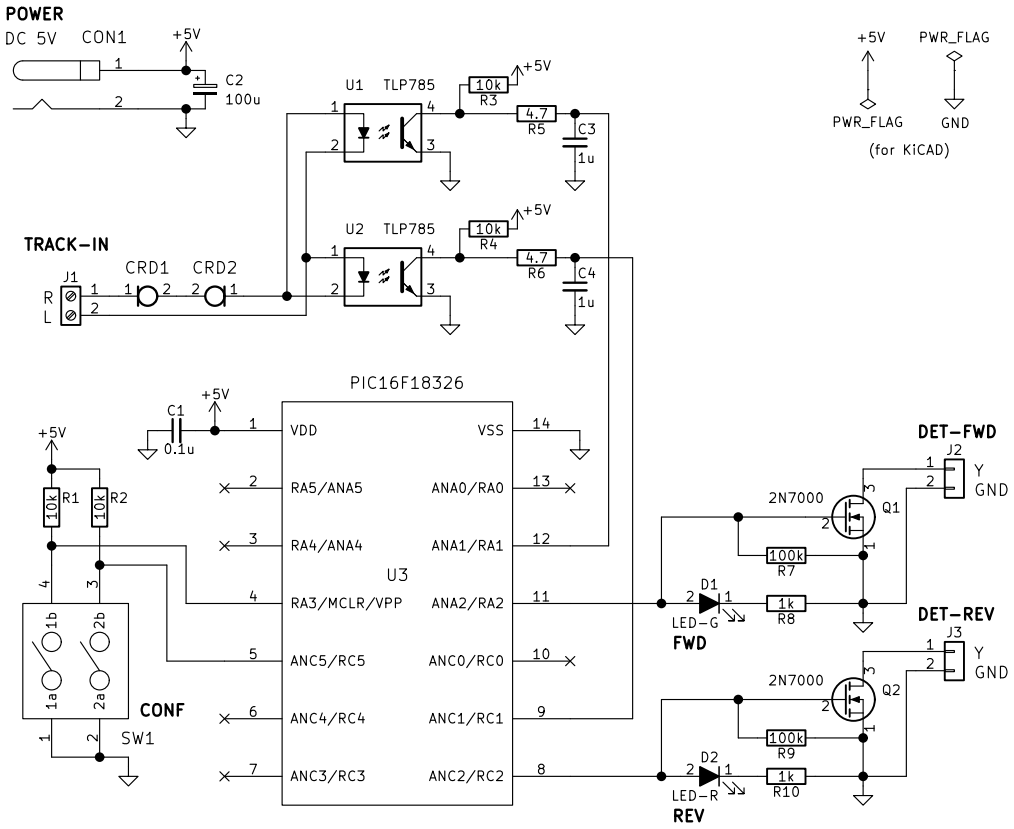


図 19.6: 【回路図】 線路電圧極性検出器

することで逆電圧による破損を防止します. 線路に電圧が印加されればフォトカプラ内のフォトトランジスタがオンとなり, 電圧レベルは 5V から 0V へと下がります. ファームウエアではこのアナログ電圧をデジタル値として読んで H と L を判別するのではなく, 電圧のアナログ値がある閾値を越えるか否かで判別します. 具体的には, 固定参照電圧モジュール FVR で閾値を定め, コンパレータモジュール CM で比較を行い, 閾値を越えるか否かを判別します.

フォトカプラには素子の特性にばらつきがあるため, 正方向と逆方向では検出できる電圧値が異なる場合がありますので注意して下さい. これは特に純直流式のパワーパックの場合に見られます. PWM 式パワーパックの場合はパルス電圧が 12V ですから, 素子の特性のばらつきは表面化せず, 平均電圧値がごくわずかな場合でも電圧極性の検出ができます.

表 19.2: 【部品表】 線路電圧極性検出器

記号	数量	部品名
U3	1	PIC 16F18326-I/P [I-11886]
Q1, Q2	2	FET 2N7000 [I-09723]
D1	1	LED 緑 [I-06413]
D2	1	LED 赤 [I-04780]
U1, U2	2	フォトカプラ TLP785　GB ランクでも可 [I-07554]
CRD1, CRD2		

(つづく)

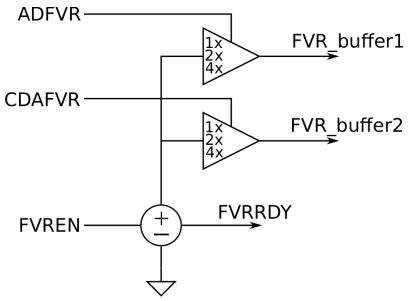


図 19.7: 固定参照電圧モジュール FVR の内部構造の概略

(つづき)

記号	数量	部品名
	2	CRD 10mA [I-00185]
C2	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C1	1	セラミックコンデンサ 0.1μF [P-00090]
C3, C4	2	セラミックコンデンサ 1μF [P-08150]
R5, R6	2	抵抗 4.7Ω　1/4W [R-25047]
R8, R10	2	抵抗 1kΩ　1/4W [R-25102]
R1, R2, R3, R4	4	抵抗 10kΩ　1/4W [R-25103]
R7, R9	2	抵抗 100kΩ　1/4W [R-25104]
	1	IC ソケット 14 ピン [P-00028]
SW1	1	DIP SW 2P [P-08922]
J1	1	ターミナルブロック 2 ピン [P-02333]
J2, J3	2	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
	2	コネクタハウジング 2 ピン　JST XH コネクタ XHP-2 [C-12255]
	4	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか基板・配線材等

19.3　FVR: 固定参照電圧モジュール

PIC マイコンでは内蔵周辺モジュール FVR (Fixed Voltage Reference; 固定参照電圧) を使用して一定の電圧の発生を行います. 図 19.7 に固定参照電圧モジュール FVR の内部構造の概略を示します. FVR の出力には FVR_buffer1 と FVR_buffer2 の 2 系統があり, それぞれで発生する電圧はマイコンの電源電圧とは独立に 1.024V, 2.048V, 4.096V の 3 種から選択可能です.

- FVR_buffer1
  ADC モジュールへ供給可能で, AD 変換の参照電圧として使用可能です.
- FVR_buffer2
  コンパレータモジュール CMP あるいは DAC(デジタルアナログ変換) モジュールへ供給可能です.

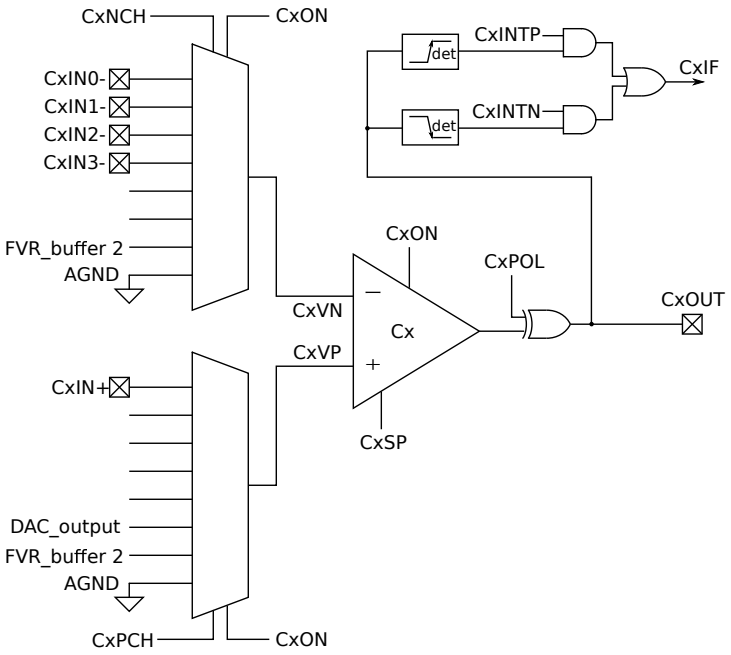


図 19.8: コンパレータモジュール CMx の内部構造の概略

16F18326 の FVR で使用する主なレジスタ類を以下に概説します.

- `FVRCOM.FVREN`
  FVR モジュールのオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.
- `FVRCOM.FVRRDY`
  FVR モジュールの出力が使用可能か否かを表すビットです. 1 の場合は使用可能で, 0 の場合は使用不可あるいはオフの状態です.
- `FVRCOM.ADFVR`
  FVR_buffer1 のゲインを設定します. 設定により出力電圧は 4.096V, 2.048V, 1.024V, あるいは出力オフとなります.
- `FVRCOM.CDAFVR`
  FVR_buffer2 のゲインを設定します. 設定により出力電圧は 4.096V, 2.048V, 1.024V, あるいは出力オフとなります.

19.4　CM: コンパレータモジュール

PIC マイコンでは内蔵周辺モジュール CM (Comparator; コンパレータ) を使用してアナログ電圧値の大小の比較を行います. 16F18326 にはふたつのコンパレータモジュールが内蔵されており, それぞれ CM1, CM2 と名付けられています. なお以下での記法 CMx は, CM1 あるいは CM2 を表すものとします. 図 19.8 にコンパレータモジュール CMx の内部構造の概略を示します.

CMx で使用する主なレジスタ類を以下に概説します.

- `CMxCON0.CxON`
  CMx のオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.
- `CMxCON0.CxOUT`

出力方法の設定をするビットです. CMxCON0.CMxPOL が 1 の場合の出力は以下の通りです.

- 出力 1 : CxVP < CxVN の場合
- 出力 0 : CxVP > CxVN の場合

CMxCON0.CMxPOL が 0 の場合の出力は以下の通りです.

- 出力 1 : CxVP > CxVN の場合
- 出力 0 : CxVP < CxVN の場合

- CMxCON0.CxPOL
  反転出力の設定をするビットです. 1 の場合は比較結果を反転して出力し, 0 の場合は反転せずにそのまま出力します.
- CMxCON0.CxSP
  コンパレータの動作速度を設定するビットです. 1 の場合は通常パワー・高速モードです. 本書執筆の時点では 0 への設定は禁止されています.
- CMxCON1.CxINTP
  コンパレータ出力が 0 から 1 へ立ち上がり時に割り込みを発生するか否かを設定するビットです. 1 の場合は CxIF を 1 にして割り込みを発生させ, 0 の場合は割り込みは発生させません.
- CMxCON1.CxINTN
  コンパレータ出力が 1 から 0 へ立ち下がり時に割り込みを発生するか否かを設定するビットです. 1 の場合は CxIF を 1 にして割り込みを発生させ, 0 の場合は割り込みは発生させません.
- CMxCON1.CxPCH
  コンパレータの正入力 CxVP の入力チャンネルを選択します. 多くの選択肢があり, 詳しくはデータシートを参照して下さい.
- CMxCON1.CxNCH
  コンパレータの負入力 CxNP の入力チャンネルを選択します. 多くの選択肢があり, 詳しくはデータシートを参照して下さい.
- PIE2.CxIE
  PIR2.CxIF が 1 の場合に割り込みハンドラを呼び出すか否かを設定します. 1 の場合は割り込みハンドラを呼び出し, 0 の場合は呼び出しません.
- PIR2.CxIF
  割り込み発生フラグです. 1 の場合は割り込み要因の発生があり, 0 の場合はありません. 0 を書き込むことで割り込み発生フラグはクリアできます.
  もし割り込み要因の発生があれば, たとえ割り込みハンドラの呼び出しが禁止されていてもこの割り込み発生フラグの値は 1 になります. 従って割り込み発生フラグの値をソフトウエアで監視することで割り込み要因の発生を検出できます.

19.5　線路電圧極性の検出

線路電圧極性の検出方法の概要を説明します. 本章のファームウエアでは, 図 19.9 に示す形で 2 つのコンパレータモジュール CM1 と CM2, 固定参照電圧モジュール FVR を使用して電圧極性を検出します.

コンパレータモジュール CMx の入力には正入力 CxVP と負入力 CxVN があり, 出力には CxOUT が装備されています. 正入力 CxVP と負入力 CxVN はともにアナログ入力です. CxVP と CxVN に接続可能な入力源は複数あり, それぞれに装備してあるマルチプレクサを設定することで入力源を選択します. 出力 CxOUT の値は

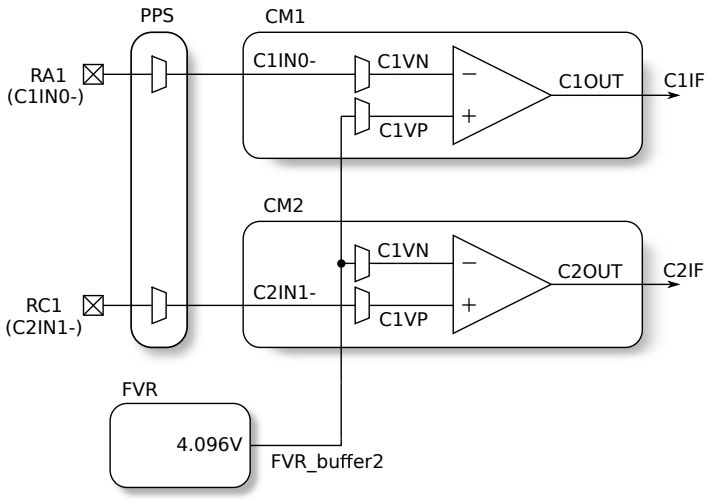


図 19.9: コンパレータモジュール CM と固定参照電圧モジュール FVR による電圧極性の検出

ふたつの入力の電圧値の大小関係により決まります.

- CxVN > CxVP のとき: CxOUT= 0
- CxVN < CxVP のとき: CxOUT= 1

以下の設定を行うことで図 19.9 に示す構成にします.

- 固定参照電圧モジュール FVR では, 出力電圧を 4.096V に設定します.
- コンパレータモジュール CM1 では, 負入力 C1VN を RA1 ピンからの入力と設定し, 正入力 C1VP を FVR 出力を入力と設定します. これは C1OUT が 1, すなわち RA1 ピンの電圧が 4.096V を下回れば, C1IF ビットが 1 になる設定です. 線路電圧が正の値になればフォトカプラ内のフォトトランジスタがオンになり, その結果 RA1 ピンの電圧は下がります. つまり線路電圧が正の値の場合に C1IF ビットは 1 になります.
- コンパレータモジュール CM2 では, 負入力 C2VN を FVR 出力を入力と設定し, 正入力 C2VP を RC1 ピンからの入力と設定します. これは C2OUT が 1, すなわち RC1 ピンの電圧が 4.096V を越えると, C2IF ビットが 1 になる設定です. 線路電圧が負の値になればフォトカプラ内のフォトトランジスタがオンになり, その結果 RC1 ピンの電圧は下がります. つまり線路電圧が負の値の場合に C2IF ビットは 1 になります.

ファームウエアでは初期化の際に上記の設定を行い, その後は C1IF ビットと C2IF ビットの観察により線路電圧極性の検出が可能となります.

19.6　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 19.3: 【ソースファイル一覧】 線路電圧極性検出器

初	頁	番号	ファイル名	内容	行数
✓	329	19.2	DirDetect.c	主ソースコード	145

(つづく)

(つづき)

初	頁	番号	ファイル名	内容	行数
✓	328	19.1	io.h	I/O ピン割付: ヘッダ	27
✓	331	19.3	mcr.h	MCR・周辺モジュール: ヘッダ	11
✓	331	19.4	mcr.c	MCR・周辺モジュール: 実装	106

計 289 行 (うち初出 289 行, 既出 0 行)

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

ソースファイル番号 19.1: io.h

```
1  #if !defined(__IO_H__)
2  #define __IO_H__
3
4  /* 検出結果の出力 (FETを通じてオープンドレイン出力: HiZ/L) */
5  #define OUT_FWD_LOW()    (LATA2 = 1)
6  #define OUT_FWD_HIZ()    (LATA2 = 0)
7  #define OUT_REV_LOW()    (LATC2 = 1)
8  #define OUT_REV_HIZ()    (LATC2 = 0)
9  /* 設定SW 1:  出力決定法の選択
10  *   SW OFF (H) の場合: 楽観的(Optimistic)
11  *   SW ON  (L) の場合: 保守的(Conservative) */
12 #define CONF_SW_MODESEL_CONSERVATIVE()  (RA3 == 1)
13 #define CONF_SW_MODESEL_OPTIMISIC()     (RA3 == 0)
14 /* 設定SW 2:  出力反転の選択
15  * SW OFF (H) の場合: 非反転出力
16  * SW ON  (L) の場合: 反転出力 (線路への接続が左右逆の場合に同じ) */
17 #define CONF_SW_OUT_NONINV()            (RC5 == 1)
18 #define CONF_SW_OUT_INV()               (RC5 == 0)
19 /* コンパレータ (CMP1, CMP2) による検出結果 */
20 #define CMP_FWD_DETECTED()  \
21         ((CM1CON0bits.C1OUT == 1) || (PIR2bits.C1IF == 1))
22 #define CMP_REV_DETECTED()  \
23         ((CM2CON0bits.C2OUT == 1) || (PIR2bits.C2IF == 1))
24 #define CMP_FWD_CLEAR()     (PIR2bits.C1IF = 0)
25 #define CMP_REV_CLEAR()     (PIR2bits.C2IF = 0)
26
27 #endif /*__IO_H__*/
```

ソースファイル DirDetect.c (主ソースコード)

動作の主となる部分です. 電圧の検出はコンパレータモジュールと固定参照電圧モジュールがハードウエア的に行うので, ファームウエアで行うことはほとんどありません. 関数 main() の動作の骨子は以下の動作を繰り返すだけの極めて簡単なものです.

1. 線路電圧の読み取りを行い, 電圧極性の検出を行います. ただし電圧をアナログ値として読み取るのではなくコンパレータモジュール CM1 と CM2 の出力の有無を読み取ります.
2. 検出した値を DIP スイッチの設定に従い, 正方向検出出力 DET-FWD と逆方向検出出力 DET-REV の値を出力します.

線路電圧の読み取りでは C1IF および C2IF に記録されている値を読み取ります. ただし読み出した C1IF および C2IF の値を直ちに検出結果とはせず, ヒステリシス特性によるノイズの除去を行って検出結果を決定します. 一定回数連続して同じ値であれば検出値として決定することでヒステリシス特性の実現を実現しています.

ソースファイル番号 19.2: DirDetect.c

```
/*************************************************
 * 線路電圧極性検出モジュール / PIC16F18326
 * by Hirotsugu Kakugawa
 *************************************************/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"

/* ------------------------------------------------------ */

/* 電圧検出 */
enum vdet_e {
  VDET_FWD = 0,  /* 順方向(右プラス) */
  VDET_REV,      /* 逆方向(左プラス) */
  VDET_MAX
};

/* 線路電圧極性の検出状態 */
struct detect_state_s {
  int8_t   detected; /* 線路電圧検出有り */
  uint8_t  hyscnt;   /* ヒステリシス特性用カウンタ */
};

/* 状態変化のヒステリシス特性実現用の定数 */
#define HYS_THRESHOLD      4

/* 検出状態 */
struct detect_state_s detect_state[VDET_MAX];

void detect_state_init(enum vdet_e vd);
void detect_state_read(enum vdet_e vd);

/*-----------------------------------------------------------
 * MAIN
 *-----------------------------------------------------------*/

void main(void)
{
  /* システム初期化 */
  System_Init();
  /* ウオッチドッグタイマ 256 [mSec] typ. */
  WDTCONbits.WDTPS = 0b01000;
  CLRWDT();
  /* 初期化 */
  detect_state_init(VDET_FWD);
  detect_state_init(VDET_REV);
  /* 主ループ */
  for (;;) {
    __delay_ms(25);
    CLRWDT();
    /* 線路電圧の読取 */
    detect_state_read(VDET_FWD);
    detect_state_read(VDET_REV);
    /* 出力値決定(検出モードに従う) */
    int8_t det_fwd = 0; /* 順方向(右プラス)を検出 */
    int8_t det_rev = 0; /* 逆方向(左プラス)を検出 */
    if (CONF_SW_MODESEL_CONSERVATIVE()) {
      /* 保守的判断: 正極性を検出したときのみ */
      det_fwd = detect_state[VDET_FWD].detected;
      det_rev = detect_state[VDET_REV].detected;
    } else {
      /* 楽観的判断: 逆極性を検出しないときのみ */
      det_fwd = !detect_state[VDET_REV].detected;
      det_rev = !detect_state[VDET_FWD].detected;
    }
    /* 検出結果の出力 */
    if (CONF_SW_OUT_NONINV()) { /* 非反転出力 */
```

```
69        if (det_fwd) {
70          OUT_FWD_LOW();
71        } else {
72          OUT_FWD_HIZ();
73        }
74        if (det_rev) {
75          OUT_REV_LOW();
76        } else {
77          OUT_REV_HIZ();
78        }
79      } else {                    /* 反転出力 */
80        if (det_fwd) {
81          OUT_FWD_HIZ();
82        } else {
83          OUT_FWD_LOW();
84        }
85        if (det_rev) {
86          OUT_REV_HIZ();
87        } else {
88          OUT_REV_LOW();
89        }
90      }
91    }
92  }
93
94  /* 検出状態初期化 */
95  void detect_state_init(enum vdet_e vd)
96  {
97    detect_state[vd].detected = 0;
98    detect_state[vd].hyscnt = 0;
99  }
100
101 /* 検出状態更新 */
102 void detect_state_read(enum vdet_e vd)
103 {
104   /* コンパレータ出力の読取 */
105   int8_t det = 0;
106   switch (vd) {
107   default:
108   case VDET_FWD:
109     det = CMP_FWD_DETECTED();
110     CMP_FWD_CLEAR();
111     break;
112   case VDET_REV:
113     det = CMP_REV_DETECTED();
114     CMP_REV_CLEAR();
115     break;
116   }
117   /* 検出状態の更新（ヒステリシス特性） */
118   switch (detect_state[vd].detected) {
119   default:
120   case 0: /* 現在出力: 電圧検出なし */
121     if (!det) { /* 電圧検出なし */
122       detect_state[vd].hyscnt = HYS_THRESHOLD;
123     } else {    /* 電圧検出あり */
124       if (detect_state[vd].hyscnt != 0) {
125         detect_state[vd].hyscnt--;
126       } else { /* 出力切替 */
127         detect_state[vd].detected = 1;
128         detect_state[vd].hyscnt = HYS_THRESHOLD;
129       }
130     }
131     break;
132   case 1: /* 現在出力: 電圧検出あり */
133     if (det) { /* 電圧検出あり */
134       detect_state[vd].hyscnt = HYS_THRESHOLD;
135     } else {   /* 電圧検出なし */
136       if (detect_state[vd].hyscnt != 0) { /* ヒステリシス特性 */
137         detect_state[vd].hyscnt--;
138       } else { /* 出力切替 */
139         detect_state[vd].detected = 0;
```

```
140           detect_state[vd].hyscnt = HYS_THRESHOLD;
141         }
142       }
143       break;
144     }
145   }
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 19.3: mcr.h

```
1  #if !defined(__MCR_H__)
2  #define __MCR_H__
3
4  #define _XTAL_FREQ  32000000 /* 32MHz */
5
6  extern void System_Init(void);
7  extern void FVR_Init(void);
8  extern void CM1_Init(void);
9  extern void CM2_Init(void);
10
11 #endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います. 図 19.9 に示した通りの設定をします. まず固定参照電圧モジュール FVR の出力電圧を 4.096V に設定し, コンパレータ CM1 および CM2 の検出方法を設定します. 次に, 検出出力 C1OUT と C2OUT が変化すると割り込み要因ビット C1IF と C2IF を 1 にするよう設定しますが, 割り込み許可ビット C1IE と C2IE は 0 にして割り込みハンドラの呼び出しは発生しないようにします. その代わりに関数 main() 内でこれら割り込み要因ビットを参照することで C1OUT および C2OUT の変化を捕捉します. 割り込み要因ビットを用いることで, C1OUT および C2OUT が 1 になる時間が極めて短い PWM 波であってもその事象の発生をハードウエアが確実に捕捉します.

ソースファイル番号 19.4: mcr.c

```
1  /*-----------------------------------------------------
2   * MCR設定&周辺モジュール (PIC16F18326)
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7
8  /*---------------------------------------------*/
9  /* MCR初期化 (FOSC=32MHz) */
10 void System_Init(void)
11 {
12   /* 発振器 */
13   OSCCON1 = 0x00; /* HFINTOSC with 2x PLL */
14   OSCCON3 = 0x00;
15   OSCEN   = 0x00;
16   OSCFRQ  = 0x06; /* 32MHz */
17   OSCTUNE = 0x00;
18   while (PLLR == 0) {
19     ;
20   }
21   /* I/Oピン */
22   LATA   = 0x00;
23   LATC   = 0x00;
24   TRISA  = 0x3B; /* (RA5,RA4),RA3,RA1,(RA0) */
25   TRISC  = 0x3A; /* RC5,(RC4,RC3),RC1 */
26   ANSELA = 0x02; /* RA1:C1IN0- */
```

```
27    ANSELC = 0x02; /* RC1:C2IN1- */
28    WPUA   = 0x31; /* (RA5,RA4),        (RA0) */
29    WPUC   = 0x18; /*     (RC4,RC3)     */
30    ODCONA = 0x00;
31    ODCONC = 0x00;
32    /* 周辺モジュール */
33    FVR_Init();
34    CM1_Init();
35    CM2_Init();
36  }
37
38  /*-------------------------------------------*/
39  /* FVR */
40  void FVR_Init(void)
41  {
42    FVRCONbits.FVREN  = 1; /* Enabled */
43    //FVRCONbits.TSEN   = 0; /* Temp ind.: Disabed */
44    //FVRCONbits.TSRNG  = 0; /* Temp ind.: Low range */
45    FVRCONbits.CDAFVR = 3; /* FVR_buffer2: 4x (4.096V) */
46    FVRCONbits.ADFVR  = 0; /* FVR_buffer1: Off */
47    while (!FVRCONbits.FVRRDY) {
48      ;
49    }
50  }
51
52  /*-------------------------------------------*/
53  /* CM1 */
54  void CM1_Init(void)
55  {
56    CM1CON0bits.C1OUT  = 0;
57    CM1CON0bits.C1POL  = 0; /* Output is not inverted */
58    CM1CON0bits.C1SP   = 1; /* Higher speed mode */
59    //CM1CON0bits.C1HYS  = 0; /* Hysterisis disabled */
60    //CM1CON0bits.C1SYNC = 0; /* Asynch */
61    CM1CON1bits.C1INTP = 1; /* C1IF is set when C1OUT rises */
62    CM1CON1bits.C1INTN = 1; /* C1IF is set when C1OUT falls */
63    CM1CON1bits.C1PCH  = 6; /* C1VP connects to FVR buffer 2 */
64    CM1CON1bits.C1NCH  = 0; /* C1VN connects to C1IN0- */
65    CM1CON0bits.C1ON   = 1; /* Enabled */
66  }
67
68  /*-------------------------------------------*/
69  /* CM2 */
70  void CM2_Init(void)
71  {
72    CM2CON0bits.C2OUT  = 0;
73    CM2CON0bits.C2POL  = 0; /* Output is not inverted */
74    CM2CON0bits.C2SP   = 1; /* Higher speed mode */
75    //CM2CON0bits.C2HYS  = 0; /* Hysterisis disabled */
76    //CM2CON0bits.C2SYNC = 0; /* Asynch */
77    CM2CON1bits.C2INTP = 1; /* C2IF is set when C2OUT rises */
78    CM2CON1bits.C2INTN = 1; /* C2IF is set when C2OUT falls */
79    CM2CON1bits.C2PCH  = 6; /* C2VN connects to FVR buffer 2 */
80    CM2CON1bits.C2NCH  = 1; /* C2VN connects to C2IN1- */
81    CM2CON0bits.C2ON   = 1; /* Enabled */
82  }
83
84  /*-------------------------------------------*/
85  /* CONFIG1 */
86  #pragma config FEXTOSC  = OFF
87  #pragma config RSTOSC   = HFINT32
88  #pragma config CLKOUTEN = OFF
89  #pragma config CSWEN    = ON
90  #pragma config FCMEN    = ON
91  /* CONFIG2 */
92  #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
93  #pragma config PWRTE    = OFF
94  #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
95  #pragma config LPBOREN  = OFF
96  #pragma config BOREN    = ON
97  #pragma config BORV     = LOW
98  #pragma config PPS1WAY  = ON
```

```
99  #pragma config STVREN   = ON
100 #pragma config DEBUG    = OFF
101 /* CONFIG3 */
102 #pragma config WRT      = OFF
103 #pragma config LVP      = OFF /* 低電圧書込不可 */
104 /* CONFIG4 */
105 #pragma config CP       = OFF
106 #pragma config CPD      = OFF
```

課題

1. 各自手持ちの純アナログ式パワーパックにおいて, 電圧極性の検出が可能な最小電圧を実験により求めて下さい.
2. 各自手持ちの PWM 式パワーパックにおいて, 電圧極性の検出が可能な最小平均電圧を実験により求めて下さい.

第 20 章

DCC デコーダとの接続

本章では DCC システム内で本書の各種装置を利用可能とする DCC デコーダ接続インターフェース装置を紹介します. 本章で紹介する回路の特徴は以下の通りです.

- DCC システムで本書の装置が使用可能
  DCC デコーダのファンクション出力をアクセサリ制御出力に変換します.
- DCC デコーダと電気的な絶縁
  DCC システムの電源系統と本書の装置での電源系統を別々にできますので, 電源系統の設置に自由度があります.

本章で学ぶ内容は以下の通りです.

- DCC デコーダのファンクション出力の基礎

20.1　使用方法

DCC システムについて簡単に説明します. DCC システムの概略図は図 20.2 に示す通りです. DCC デコーダにはヘッドライト, テールライト, 室内灯の点灯など, デコーダ外部の機器を制御するための出力が備えられており, と呼ばれています. DCC デコーダには複数のファンクション出力が装備されていることがほとんどです. ファンクション出力の制御は DCC スロットルのファンクションボタンで行われます.

DCC デコーダのファンクション出力は, 1 本の共通線と 1 ファンクション出力につき 1 本の線で構成されます. 表 20.1 に DCC デコーダの標準的な入出力の規格を示します. ただしデコーダにより機能の多少はありますので注意して下さい. コモンがファンクション間で共通の線であり, 極性は常にプラスです. ファンクション出力は, オフの場合にはコモンと等電位となり, オンの場合には 0V となります. 従ってファンクション出力がオンの場合には, コモンからファンクション出力へと電流が流れます. この電流の流れる方向には注意して下さい.

本装置と DCC デコーダの接続を図 20.3 に示します. DCC デコーダのファンクション出力のオン・オフをア

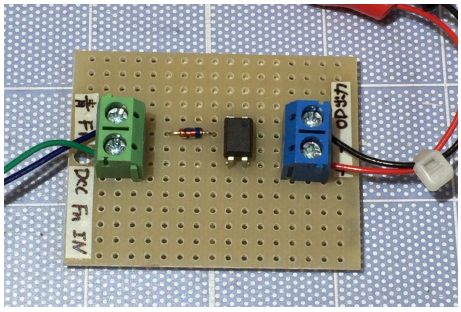


図 20.1: DCC デコーダ接続インターフェースの試作例

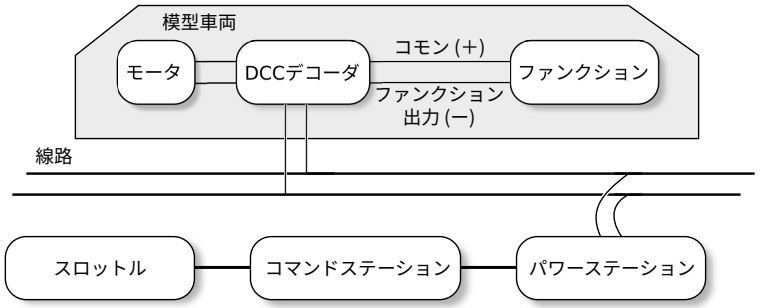


図 20.2: DCC システム

表 20.1: DCC デコーダのリード線

機能名	電線色	備考
右レール	赤	
左レール	黒	
モータ (右)	オレンジ	
モータ (左)	灰	
コモン (+)	青	共通線
ファンクション出力 1 (–)	白	ヘッドライト
ファンクション出力 2 (–)	黄	テールライト
ファンクション出力 3 (–)	緑	
ファンクション出力 4 (–)	紫	

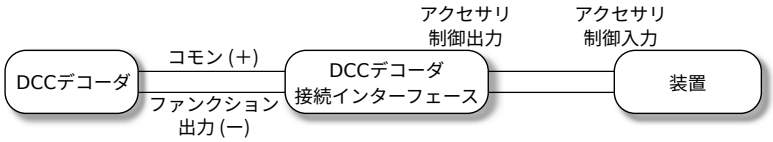


図 20.3: DCC デコーダとの接続

クセサリ制御出力でのオン・オフに変換しますから, 例えば本書で紹介する LED 点灯器や PCM サウンド再生器などの各種アクセサリ類を DCC スロットルから制御できます.

20.2　回路図

DCC デコーダ接続インターフェース装置の回路図を図 20.4 に, 部品リストを表 20.2 に, それぞれ示します.

DCC システムと本書のシステムとは電源系統が異なりますから, フォトカプラを用いて絶縁した回路になっている点に注意して下さい. フォトカプラ内の LED に流れる電流を 10mA 程度にするとフォトカプラ内のフォトダイオードがちょうど良く導通状態となります. しかし DCC デコーダのファンクション出力電圧はパワーステーションの出力電圧に依存します. DCC 規格 NMRA S-9.1 では DCC デコーダへの入力電圧は 7V から 27V の範囲と定められており, 幅広い入力電圧で動作することが求められています (文献 [8]). 従って DCC デコーダのファンクション出力電圧もほぼこの電圧の範囲となります. そこで定電流ダイオード CRD を使用することで幅広い電圧値に対してフォトカプラに流れる電流を一定にしています. 本回路の入力側にはダイオードを入れて, 万が一 DCC デコーダとの接続を間違って逆向き, つまり COM と FN を逆に接続しても, 本回路と

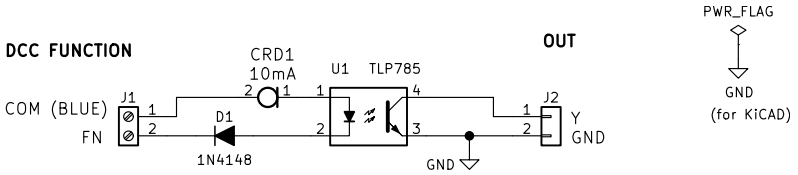


図 20.4: 【回路図】 DCC デコーダ接続インターフェース装置

DCC デコーダが過電流で破損しないようにしています.

表 20.2: 【部品表】 DCC デコーダ接続インターフェース装置

記号	数量	部品名
U1	1	フォトカプラ TLP785　GB ランクでも可 [I-07554]
D1	1	ダイオード 1N4148　同等品でも可 [I-00941]
CRD1	1	CRD 10mA [I-00185]
J1	1	ターミナルブロック 2 ピン [P-02333]
J2	1	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
		ほか基板・配線材等

課題

1. DCC デコーダのファンクション出力で本書で紹介しているポイント切替器の制御を可能とする回路を設計して下さい. ポイントの切替だけでなく, 第 15 章で紹介したサーボモータ制御器により, DCC スロットルからサーボモータの操作が可能となります.

第 21 章

装置の相互結合

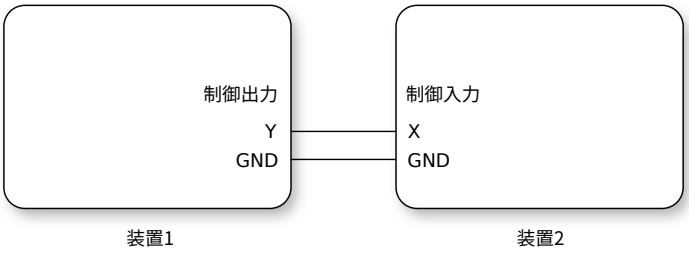
本章では複数の装置を相互結合する装置を紹介します. 本書で作成する様々な装置はどれも基本的な機能だけしかありませんが, 互いに組み合わせることで多彩な利用が可能になり, 鉄道模型レイアウトの規模の拡大に応じて装置を追加できます.

本章で紹介する回路の特徴は以下の通りです.

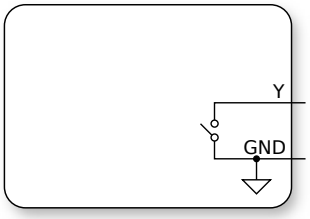
- アクセサリ制御出力の分配が可能
  ひとつの装置のアクセサリ制御出力で複数の装置を制御できます.
- アクセサリ制御出力の集約が可能
  複数の装置のアクセサリ制御出力でひとつの装置を制御できます.

本章で学ぶ内容は以下の通りです.

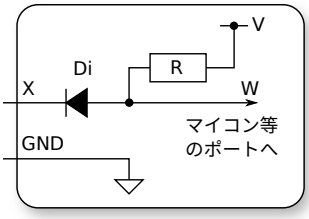
- ワイヤード OR の概念
- 制御入出力の相互結合の方法



(a) アクセサリ制御入出力による 2 装置の接続



(b) アクセサリ制御出力部回路の概略



(c) アクセサリ制御入力部回路の概略

図 21.1: 装置間の相互接続

21.1　アクセサリ制御入出力

本書で製作する装置の多くには, 外部に動作結果を信号で伝えるためのアクセサリ制御出力や, 外部から信号を受けるためのアクセサリ制御入力が備わっています. 図 21.1a に示すように, ある装置のアクセサリ制御出力を別の装置のアクセサリ制御入力に接続して装置間の連携動作が可能です.

- アクセサリ制御出力
  装置内のアクセサリ制御出力部の回路を概念的に示したのが図 21.1b です. 概念的には片側が GND に接続されたスイッチです. 実際には, 機械的なスイッチであったり, FET のオープンドレイン出力やトランジスタのオープンコレクタ出力などで実現されます. 回路図において出力端子は Y で始まる記号で表します. なお「アクセサリ制御出力」という名称の中で「出」という文字が使用されてはいますが, アクセサリ制御出力では電流が装置に入ってくる点に注意して下さい. 論理的なオン・オフという状態を装置が出力すると考えており, 電流の物理的な方向ではありません.
- アクセサリ制御入力
  装置内のアクセサリ制御入力部の回路を概念的に示したのが図 21.1c です. 抵抗でプルアップしてマイコン等の入力ポートに接続されて, アクセサリ制御入力の状態を読み取ることを可能としています. ここでダイオードを使っている理由は後述します. 回路図において入力端子は X で始まる記号で表します. なお「アクセサリ制御入力」という名称の中で「入」という文字が使用されてはいますが, アクセサリ制御入力からは電流が外部に出てゆく点に注意して下さい. 図 21.1c に描いている通り, マイコンのポートへ接続される線を W として以下の説明では参照します.

回路図から分かる通り, アクセサリ制御出力側の装置とアクセサリ制御入力側の装置は GND レベルが共通なだけであり, それぞれの電源電圧は異なっていても構いません. 例えば 5V 電源の装置と 12V 電源の装置をアクセサリ制御出力とアクセサリ制御入力とで結ぶことができます.

21.1.1　アクセサリ制御入出力の仕様

アクセサリ制御入出力の電気的な仕様を説明します. ある装置のアクセサリ制御出力を図 21.1a に示すように, 別の装置のアクセサリ制御入力に接続することで動作の制御を行います. 出力側の装置は図 21.1b に示す通りで, Y=HiZ (スイッチがオフ) の時には図 21.1c に示す入力側の装置は動作せず, Y=GND (スイッチがオン) の時には動作する, という形の制御を一般的な使い方とします. オン状態は理想的には GND レベルの 0V ですが, FET 等を使用した場合を考慮して 0.3V 以下ならオン状態とします.

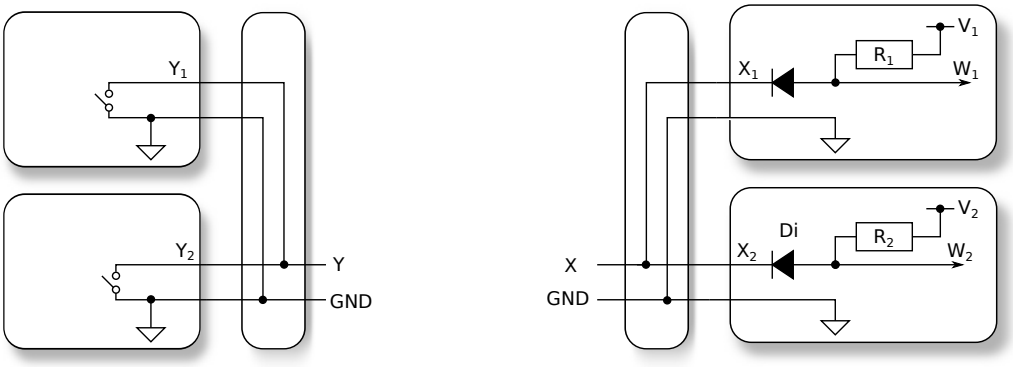
アクセサリ制御入出力の仕様

制御状態	論理値	Y	備考
オフ	0	HiZ	
オン	1	GND	0.3V 以下

アクセサリ制御入出力の最大規格

- 最大電流値: 20mA
- 最大電圧値: 24V

図 21.1c から分かる通り, 入力 X が 0V (GND レベル) の場合はマイコン等への接続 W も 0V になり, マイコンでの入力は L になります. 入力 X が HiZ の場合は W は電源電圧 Vcc に等しくなり, マイコンでの入力は H



(a) ワイヤード OR による論理和　　**(b)** 分岐

図 21.2: アクセサリ制御出力の論理和と分岐

になります. アクセサリ制御入力での電源電圧 Vcc や抵抗 R は, アクセサリ制御出力に流れ込む総電流値が最大規格を越えないように設定します. この設定はひとつの装置単体で考えるだけではなく, 複数の装置が接続された場合も考慮しないといけません.

21.1.2　アクセサリ制御出力のワイヤード OR

図 21.1b に示すアクセサリ制御出力の回路の利点はワイヤード OR (wired-OR) 接続が構成可能な点にあります. ワイヤード OR とは単に結線をするだけで論理和の計算が実現できることを言います. 2 つの装置のアクセサリ制御出力を結線したものを図 21.2a に示します. 2 つの装置のアクセサリ制御出力をそれぞれ Y_1, Y_2 とし, 結線した結果を Y とします.

- Y_1 と Y_2 ともにオフ, つまり HiZ の場合
  Y はオフとなります. なぜなら Y は HiZ だからです.
- Y_1 と Y_2 のうち少なくとも 1 つがオン, つまり GND の場合
  Y はオンとなります. なぜなら Y は GND だからです.

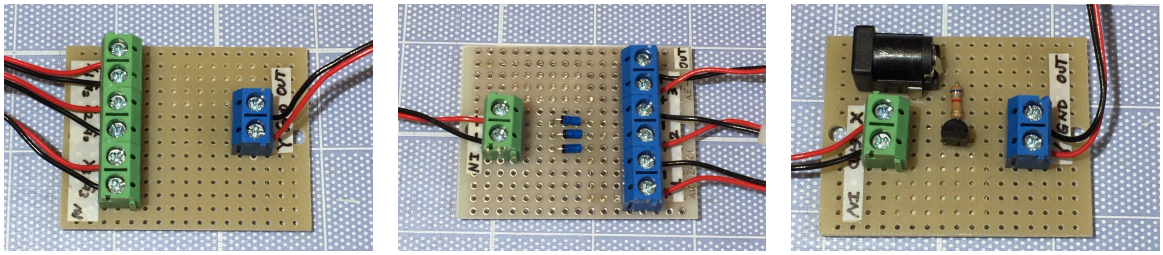
アクセサリ制御入出力の制御規格の表では, オフを論理値 0, オンを論理値 1 と決めていました. この論理値を使用して Y_1, Y_2, Y の関係を真理値表で表すと以下の通りであり, 論理和であることが分かります.

入力		出力
Y_1	Y_2	Y
0 (オフ)	0 (オフ)	0 (オフ)
1 (オン)	0 (オフ)	1 (オン)
0 (オフ)	1 (オン)	1 (オン)
1 (オン)	1 (オン)	1 (オン)

以上では 2 つのアクセサリ制御出力のワイヤード OR の説明をしましたが, 3 つ以上の場合も同様に結線するだけで論理和の計算ができます.

21.1.3　アクセサリ制御出力の分岐

アクセサリ制御出力を分岐して複数の装置のアクセサリ制御入力に接続する方法と, 図 21.1c でダイオードを使用している理由を説明します. 本書の方式では複数の装置のアクセサリ制御入力を単に結線するだけでひ



(a) 論理和 (3 入力 1 出力)　**(b)** 分岐 (1 入力 3 出力)　**(c)** 論理否定

図 21.3: 相互接続用装置の試作例

とつのアクセサリ制御出力を共用ができます. つまりアクセサリ制御出力を単に分岐するだけで良いのです. 図 21.2b ではひとつのアクセサリ制御出力を 2 つに分岐して, 2 つの装置のアクセサリ制御入力に接続している場合を示しています. それぞれの装置では異なる電源電圧値, 異なる抵抗値の場合がありますが, それでも正しく分岐できる理由は以下の通りです.

- アクセサリ制御出力 Y がオフ, つまり HiZ の場合
  X_1 と X_2 どちらからとも電流は流れ出しません. なぜなら Y は HiZ なので Y へ電流は流れ出しませんし, 一方の装置から他方の装置へはダイオードの作用で電流は流れないからです. このとき一方の装置のマイコンの入力 W_1 は電源電圧 V_1 なので H, 他方の入力 W_2 は電源電圧 V_2 なので H となります. つまり 2 つの電源電圧 V_1 と V_2 が異なっていても正しく動作をします.
- アクセサリ制御出力 Y がオン, つまり GND の場合
  X_1 と X_2 両方から電流が Y へと流れ出します. このとき一方の装置のマイコンの入力 W_1 は 0V なので L, 他方の入力 W_2 も同様に L となります. 厳密に言えば 0V ではなく, 各ダイオード順方向電圧による電圧降下分が W_1, W_2 それぞれの電圧値となります. この理由により, マイコンが L レベルと認識するに十分小さな順方向電圧値のダイオードを使用します.

以上で相互接続の基礎的な原理の説明は終わりました. 原理を考えれば明らかですが, ワイヤード OR したアクセサリ制御出力を複数の装置のアクセサリ制御入力に分岐することも可能です. 従って本書で紹介する装置間の接続は電流値の総和だけに注意して, あとは単に接続・分岐・結合をすれば良いということです.

21.2　相互接続の回路

アクセサリ制御入出力を相互に接続するための各種回路を紹介します. 図 21.3 に試作例の写真を示しますが, 部品リストに掲載している基板用コネクタは使用せずターミナルを代わりに使用している点に注意して下さい. 他章の試作例でも同様なものがあります.

21.2.1　論理和

アクセサリ制御出力の論理和を計算する装置を紹介します. 先述の通りワイヤード OR により論理和が求まりますので, 単にアクセサリ制御出力を結線してひとつにするだけです. 3 入力 1 出力の論理和装置の回路図を図 21.4 に, 部品リストを表 21.1 に, それぞれ示します. 入力数が 3 以外の場合も同様に構成できます.

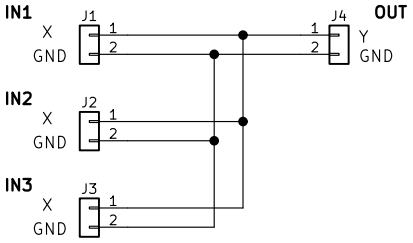


図 21.4: 【回路図】 論理和装置 (3 入力 1 出力)

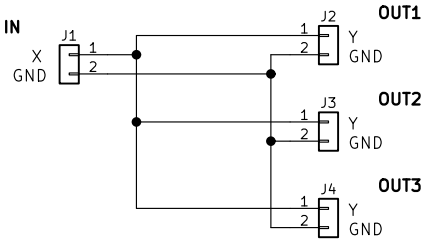


図 21.5: 【回路図】 出力分岐装置 (1 入力 3 出力)

表 21.1: 【部品表】 論理和装置 (3 入力 1 出力)

記号	数量	部品名
J1, J2, J3, J4	4	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
		ほか基板・配線材等

この回路は 3 入力ですが, この装置の入力に実際に接続する数はちょうど 3 である必要はありません. 1 つでも 2 つでも構いません. 空き入力のターミナルはなにもせず, そのままにしておきます.

21.2.2　分岐

アクセサリ制御出力を分岐する装置を紹介します. 先述の通りアクセサリ制御出力を単に分岐して, 各装置のアクセサリ制御入力につなげるようにするだけです. 1 入力 3 出力の分岐装置の回路図を図 21.5 に, 部品リストを表 21.2 に, それぞれ示します. 出力数が 3 以外の場合も同様に構成できます.

表 21.2: 【部品表】 出力分岐装置 (1 入力 3 出力)

記号	数量	部品名
J1, J2, J3, J4	4	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
		ほか基板・配線材等

21.2.3　論理否定

アクセサリ制御出力の論理反転をする装置を紹介します. この装置は入力のオン・オフを逆のオフ・オンに出力します. 真理値表は以下の通りです.

入力	出力
X	Y
0 (オフ)	1 (オン)
1 (オン)	0 (オフ)

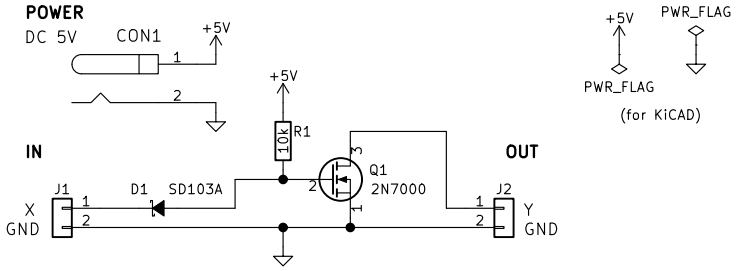


図 21.6: 【回路図】 論理否定装置

表 21.3: 【部品表】 論理否定装置

記号	数量	部品名
Q1	1	FET 2N7000 [I-09723]
D1	1	ショットキーバリアダイオード SD103A [I-04271]
R1	1	抵抗 10kΩ　1/4W [R-25103]
J1, J2	2	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか基板・配線材等

21.2.4　アクセサリ制御出力モニタ

アクセサリ制御出力用の状態を LED で点灯表示するアクセサリ制御出力モニタを紹介します. これは様々な装置を作る際に動作の確認をするために大変便利です. 動作確認したい装置のアクセサリ制御出力にこの装置のアクセサリ制御入力を接続して使用します. この装置の動作は以下の通りです.

入力	出力	
X	LED-ON	LED-OFF
0 (オフ)	消灯	点灯
1 (オン)	点灯	消灯

回路図を図 21.7 に, 部品リストを表 21.4 に, それぞれ示します.

表 21.4: 【部品表】 アクセサリ制御出力モニタ

記号	数量	部品名
Q1, Q2	2	FET 2N7000 [I-09723]
D1	1	ショットキーバリアダイオード SD103A [I-04271]
D3	1	LED 緑 [I-06413]
D2	1	LED 赤 [I-04780]
R1	1	抵抗 10kΩ　1/4W [R-25103]
R2,R3	2	抵抗 1kΩ　1/4W [R-25102]

(つづく)

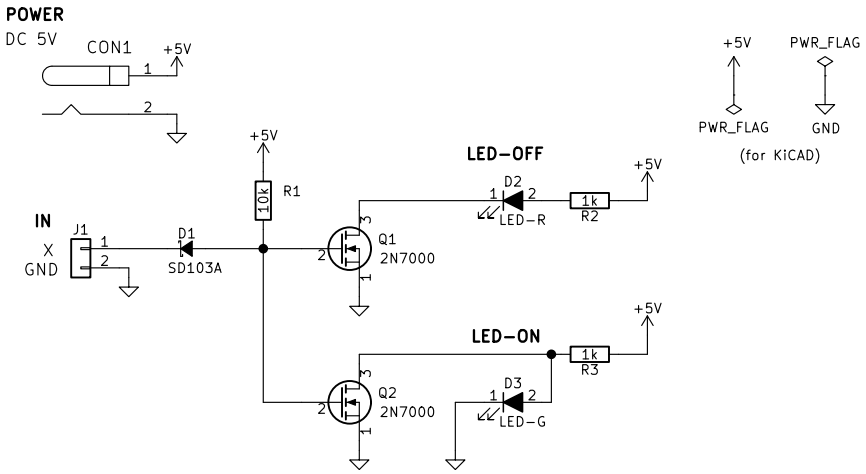


図 21.7: 【回路図】 アクセサリ制御出力モニタ

(つづき)

記号	数量	部品名
J1	1	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
	2	コネクタハウジング 2 ピン　JST XH コネクタ XHP-2 [C-12255]
	4	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
CON1	1	DC ジャック　[C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか基板・配線材等

課題

1. 4 つのアクセサリ制御出力のワイヤード OR の回路図を描き, 真理値表を求めて下さい. そして論理和となっていることを確認して下さい.
2. 図 21.1b に示すアクセサリ制御入力回路内部の電源電圧 Vcc が 5V で抵抗 R が 10kΩ の場合, ひとつのアクセサリ制御出力に最大何個のアクセサリ制御入力を接続できるでしょうか. 算出方法を示してください. ただしアクセサリ制御出力に流れ込む電流は最大 20mA です. 電源電圧 Vcc が 12V で抵抗 R が 1kΩ の場合も計算して下さい.
3. 図 21.2b の回路においてもしダイオードがない場合に何が起きるでしょうか. V_1=5V, R_1=10kΩ, V_2=12V, R_1=10kΩ とし, Y が HiZ の場合を考えます. この場合において W_1 と W_2 はそれぞれ何ボルトになるかを計算して下さい.
4. 3 入力 1 出力論理和装置において実際に接続される入力の数が 3 より小さい場合, 空き入力には何も接続はせずそのままで良いと述べました. なぜこれでいいのでしょうか. 回路図に基づいて誤動作はしないことを説明して下さい.
5. 図 21.6 の論理否定装置の動作を回路図に基づいて説明して下さい.
6. 図 21.7 のアクセサリ制御出力モニタは電源電圧を 5V として設計しています. しかし鉄道模型での電源環境を考えると, 12V や 18V の電源でも動作すると大変助かります. そこで, 電源電圧が 5V から 18V

までの範囲で動作が可能なアクセサリ制御出力モニタの回路を設計して下さい. そして設計した回路の各部品の絶対規格が守られていることを示して下さい.

7. 本章では論理積演算を行う装置の紹介がありません. そこで使いやすい論理積装置を設計して回路図を示し, その使い方と動作原理を説明して下さい.

第 V 部

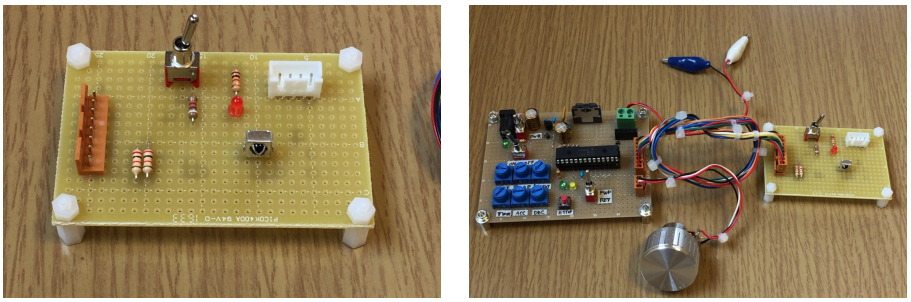
テレビリモコン式のパワーパック

第 22 章

テレビリモコン式パワーパック

本章では赤外線テレビリモコンを使用して遠隔制御可能なパワーパックを紹介します. 第 9 章で紹介したパワーパックをメインボードとし, 拡張ボードを新たに製作して接続します. 図 22.1a に拡張ボードの試作例を示します。リモコン制御機能はメインボード上のマイコンのファームウエアの差し替えで実現します. 第 23 章では本章の内容を拡張してポイント制御もリモコン操作可能にします. 本章で紹介する回路とファームウエアの特徴は以下の通りです.

- リモコン式パワーパック
  自由な位置から模型列車の走行の操作が可能です.
- コントローラには赤外線テレビリモコンを使用
  赤外線テレビリモコンは入手が容易です。また低消費電力で長時間動作が可能なうえ交換容易で汎用的



(a) 拡張ボード　　**(b)** パワーパック本体との接続

図 22.1: テレビリモコン式パワーパックの試作例



図 22.2: 汎用テレビリモコンの例. (左より JVC RM-A533, AudioComm AV-R310Z)

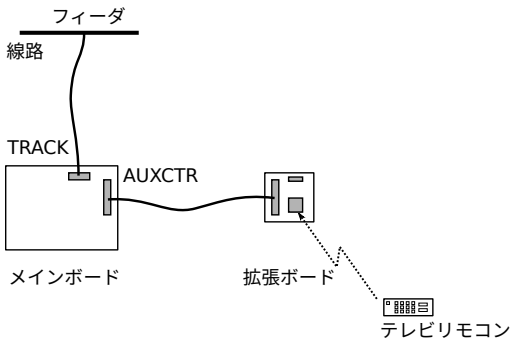


図 22.3: 接続方法

表 22.1: テレビリモコンのボタン機能一覧

ボタン	機能
電源	緊急停止
11	走行方向を前進に設定
12	走行方向を後進に設定
音量大	走行速度のスピードステップを 1 増加
音量小	走行速度のスピードステップを 1 減少
チャンネル＋	選択中のポイントを定位に切替
チャンネルー	選択中のポイントを反位に切替
1	ポイント番号 1 を選択
2	ポイント番号 2 を選択
⋮	⋮
8	ポイント番号 8 を選択

な乾電池を使用しますから電池切れの場合も安心です.

- 複数の赤外線テレビリモコンをサポート
  NEC フォーマットと呼ばれるデータ形式に従った赤外線テレビリモコン各種をサポートし, 日立製作所, 東芝, 三洋, LG のテレビに付属する赤外線テレビリモコンの多くが使用可能です. 複数のメーカに対応する汎用の赤外線テレビリモコンを購入して本装置専用にすると良いでしょう.
- 自動加減速・低速ブースト機能つき
  第 10 章で紹介した自動加減速・低速ブースト機能型パワーパックが有する機能をすべて引き継いでいます.

本章で学ぶ内容は以下の通りです.

- 赤外線式テレビリモコンの仕様
- テレビリモコンのフレームデコーダの設計手法

タブレット PC やスマートフォンによるハイテクな鉄道模型のリモコン走行制御システムがありますが, 流行り廃りが非常に激しく 10 年後にも使用できるか心配です. 一方, 赤外線式テレビリモコンはローテクですが枯れた技術であるが故に, 30 年後も使用できるものと期待しています.

本章のファームウエアの大まかな構造を図 22.4 に示します. 第 10 章で紹介した自動加減速・低速ブースト機能型パワーパックとプログラム構造は同じです. 目標速度の決定にスピードボリュームを読んだ値を使うのではなく, テレビリモコンから受信した値を使うところだけの違いです. このテレビリモコンの受信部分が若干複雑なので, 詳細は後に説明します.

22.1　使用方法

本装置の使用方法を以下に説明します. 第 9 章で紹介したパワーパックをメインボードとして, 本章で紹介する拡張ボードを接続します. 以下の説明で使用する部品名称は図 22.5 に配置例とともに示しています. 本章の回路図では回路とファームウエアの解説の関係からパワーパック本体に拡張ボードをコネクタで接続する形としていますが, 最初から両者を一体として組み立てるほうが良いかも知れません.

```
void main(void)
{
  for(;;) {
    speed = ブレゼンハムアルゴリズムで次の現在速度を決定;
    md = PWMデューティ比を計算;
  }
}

タイマ割り込みハンドラ: (低速ブーストの実現)
{
  PWMデューティ比をモータドライバに出力(低速ブーストの実現方法に従
      う);
}

IOC割り込みハンドラ (赤外線リモコン受信モジュールの出力が変化した時)
{
  受信ビットが0 か1かを判定;
  if (フレームの受信終了) {
    リモコンで押されたキーを判定;
    switch (押されたキー) {
    case 緊急停止:
      走行速度を0に設定(speed=0)してモータドライバへ直ちに出力;
      break;
    case 前進または後進:
      走行方向を変更;
      break;
    case 加速または減速:
      スピードステップを計算;
      speed_target = スピードステップより目標速度を決定;
      break;
    }
  }
}
```

図 22.4: ファームウエア構造 (概要)

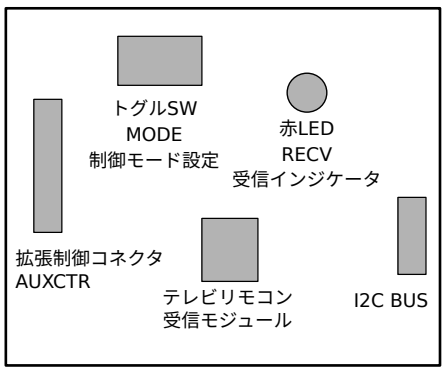


図 22.5: 部品の配置例と名称

本装置で使用するものは以下の通りです.

- パワーパック本体 (第 9 章)
  これをメインボードとして使用します.
- 拡張ボード (本章)
  メインボードに接続する拡張ボードで, 赤外線テレビリモコンの信号を受信します.
- 赤外線テレビリモコン
  各種メーカに対応している汎用リモコンで NEC フォーマットの信号を出力するものを用意します. 本装置では日立製作所, 東芝, 三洋, フナイ, ナナオ, LG エレクトロニクス, bydisign のテレビリモコンを利用可能にしていますが, すべてのテレビリモコンが使えるとは限りません. 同じメーカのテレビリモコンでも信号が異なるものがあるためです.

操作手順は以下の通りです.

1. 電源スイッチを切った状態でパワーパック本体の DC ジャックに 12V の AC アダプタを接続します.
2. 図 22.3 に示すように接続をして下さい.
   - メインボード TRACK ↔ 線路のフィーダ
   - メインボード AUXCTR ↔ 拡張ボード AUXCTR
3. 拡張ボードのスイッチ MODE はリモコン制御モード側に切り替えます. なお拡張ボードのスイッチ MODE を手動制御モード側に切り替えて手動制御も可能です. スイッチ MODE を切り替えた場合には電源は入れ直して下さい.
4. 電源スイッチを入れます. すると LED はフラッシュ点滅してテレビリモコンの自動選択機能の動作中であることを示します. テレビリモコンのボタンをいろいろと押して自動選択を行います. テレビリモコンはメーカや製品ごとに出力するデータが違いますので, その選択をここで行います. ボタンの中でも特に, **電源** **音量大** **チャンネル＋** 1 12 のいずれかで特定される場合が多いです. LED のフラッシュ点滅が止まり, 点灯が続けば選択は完了です. もし選択ができない場合には, テレビリモコンのメーカコード設定をいろいろ変えて試して下さい. なお使用中のテレビが信号を受信して動作してしまわないように, テレビとは違うメーカコード設定にします.
5. 模型車両の走行操作の準備は終わりました. テレビリモコンを使用して模型車両を走行させましょう. 表 22.1 にテレビリモコンのボタン機能一覧を示します. なお本章の範囲内ではまだポイント制御機能はなく, 第 23 章で実装します.

テレビリモコンは使用せずに手動で走行させることも可能です. その場合にはいったん電源を切り, 拡張ボードのスイッチ MODE を手動制御モード側に切り替えて電源を入れます. すると第 10 章のときと同じ動作をします.

22.2　回路図

表 22.2: 【部品表】 パワーパック (テレビリモコン式) 拡張ボード

記号	数量	部品名
U1	1	OSRB38C9AA　赤外線リモコン受信モジュール [I-04659]
D1	1	LED 赤　[I-04780]

(つづく)

(つづき)

記号	数量	部品名
R1	1	抵抗 1kΩ　1/4W [R-25102]
R3, R4	2	抵抗 2kΩ　1/4W [R-25202]
R2	1	抵抗 10kΩ　1/4W [R-25103]
SW1	1	単極双投トグル SW　On-On 型 [P-02399]
J1	1	基板用コネクタ 8 ピン　JST XH コネクタ B8B-XH-A(LF) [C-12253]
J2	1	基板用コネクタ 4 ピン　JST XH コネクタ B4B-XH-A(LF) [C-12249]
	2	コネクタハウジング 8 ピン　JST XH コネクタ XHP-8 [C-12261]
	16	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
		ほか接続用ケーブル・基板・配線材等

拡張ボードの回路図を図 22.6 に, 部品リストを表 22.2 に, それぞれ示します. 本装置での赤外線リモコン受信器には OptoSupply 社の OSRB38C9AA を使用していますが, 機能的に同等なものが多数ありますので互換性のある赤外線リモコン受信モジュールを使用して構いません. ただし機能は同等でもピン配置が異なるものが結構ありますので注意して下さい. 図 22.7 に示すパワーパック拡張ケーブルを作成して, パワーパック本体と拡張ボードとの接続に使用します. I^2C は SCL と SDA の 2 本の通信線で構成される通信規格で, 詳しくは第 23 章で説明します. SCL と SDA の 2 本の通信線はそれぞれ抵抗器によるプルアップを行います.

22.3　赤外線テレビリモコン規格

家庭用テレビのリモコンとして赤外線通信によるリモコンが広く普及しており, いくつかの赤外線テレビリモコンの規格が制定されています. 送信には波長 950nm の赤外線を使用して, 図 22.8 に示すように周波数 38kHz の副搬送波のオン・オフによるパルス幅変調により送信データの 0 と 1 を符号化します. データのフォーマットには主に NEC フォーマット, 家電製品協会 (AEHA) フォーマット, SONY フォーマットの 3 種類の規格があり, それぞれパルスの時間幅や 1 パケット中のデータ表現方法などが異なります. 以下では本章のファームウエアで使用する NEC フォーマットの概略を説明します. なおここで示す周波数や時間の数値には多少の幅がありますので注意してください.

- 1 回に送信されるデータの単位をフレーム (frame) と呼びます.
- フレームの構造は図 22.8a に示す通り, 先頭部分から順にリーダー部 (leader), カスタマコード部 (customer code), データ部 (data), ストップビット部 (stop bit) で構成されます. リモコンのボタンを押し続けた場合を表すためのリピート用のフレームもありますが説明は省略します.
- リーダー部は, 9000 マイクロ秒のオン期間の後に 4500 マイクロ秒のオフ期間が続きます.
- カスタマコード部とデータ部はともに 2 バイト (16 ビット) で, 両方を合わせて 4 バイト (32 ビット) で構成されます. ストップビット部は 1 ビットで構成されます.
- カスタマコード部の 2 バイトは, 各バイトの最下位ビット (LSB) から最上位ビット (MSB) の順で送信されます. データ部の 2 バイトについても同様な順序で送信されます.
- ビット 0 と 1 の表現は図 22.8b に示す通り, パルス幅により行います. ビット 0 は 560 マイクロ秒のオン期間の後に 560 マイクロ秒のオフ期間が続けることで表現します. ビット 1 は 560 マイクロ秒のオン期間の後に 1690 マイクロ秒のオフ期間が続けることで表現します.

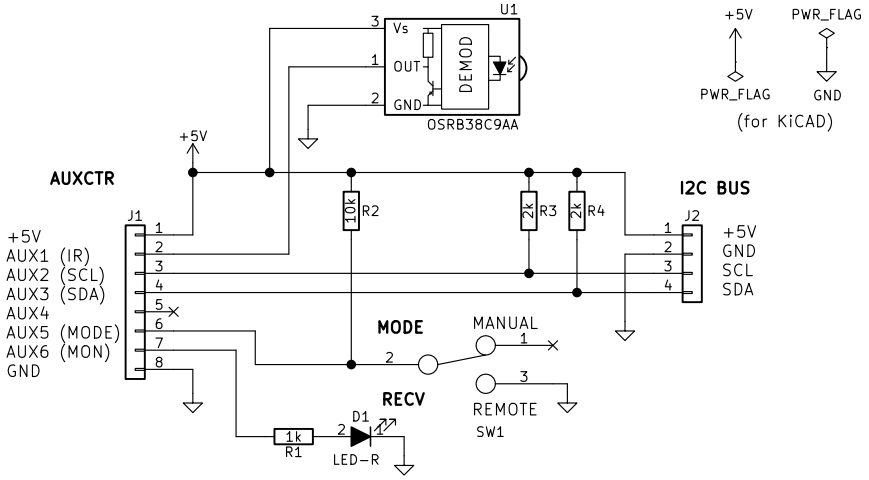


図 22.6: 【回路図】 パワーパック (テレビリモコン式) 拡張ボード

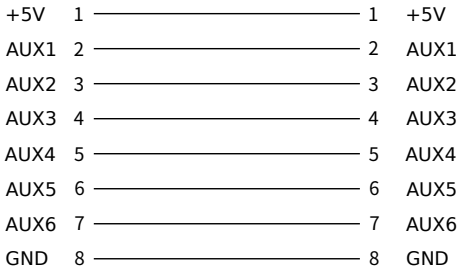


図 22.7: パワーパック拡張ケーブル

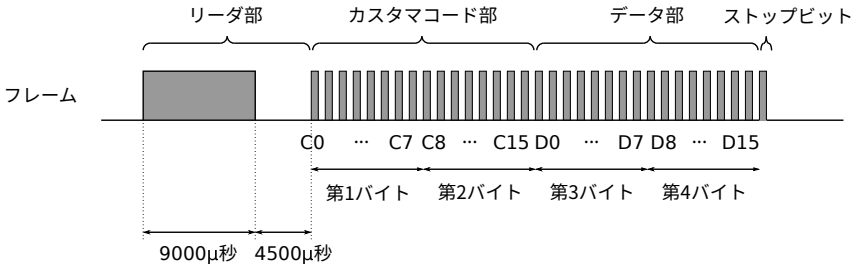
- 図 22.8c にビット列 01101 の場合の例を示します.
- 図 22.8d に信号オン・オフ, 副搬送波, 搬送波の 3 者の関係を示します. 搬送波は波長 950nm の赤外線であり, これを 38kHz の副搬送波で点灯・消灯を行う変調をして, 更にこれをオン・オフすることで信号のオン・オフを表現します.

赤外線テレビリモコンの一般的な規格は上記の通りで, カスタマコード部とデータ部の具体的な値はメーカごとに異なります. テレビリモコンの電源オフのボタンが押されたときに送信されるカスタマコード部とデータ部の 4 バイトの例を以下に示します.

日立製作所	0x50, 0xAF, 0x17, 0xE8
東芝	0x40, 0xBF, 0x12, 0xED
三洋	0x30, 0xCF, 0x00, 0xFF
LG エレクトロニクス	0x04, 0xFB, 0x08, 0xF7

22.4　赤外線リモコン受信モジュール

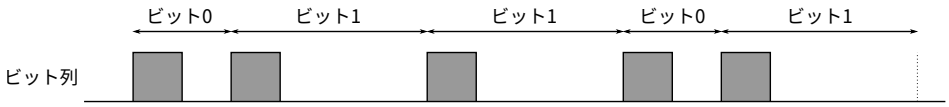
赤外線リモコン受信モジュールは副搬送波 38kHz の赤外線のオン・オフを受信して, 論理レベル出力 H・L に変換する回路を内蔵した電子部品です. 同様な機能を持った回路を自作するのは大変ですが低価格なものが各種発売されており, 本装置ではそのなかのひとつである OptoSupply 社の OSRB38C9AA を利用します. この外観を図 22.9 に示します. 本節では OSRB38C9AA の概要を説明しますが, 他の赤外線リモコン受信モジュールでも同じ機能を持ったものが多数あります.



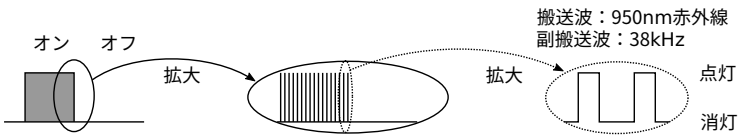
(a) フレームの構造



(b) パルス幅によるビット 0 と 1 の表現



(c) ビット列 01101 の例



(d) 信号オン・オフ, 副搬送波, 搬送波

図 22.8: 赤外線テレビリモコン NEC フォーマット

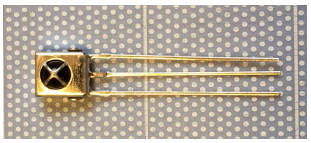


図 22.9: 赤外線リモコン受信モジュール OSRB38C9AA

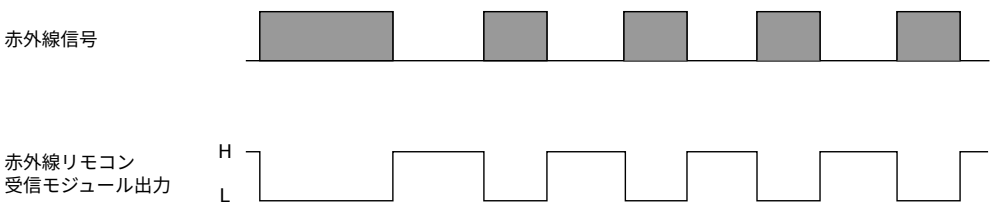


図 22.10: 赤外線リモコン受信モジュール OSRB38C9AA の出力

図 22.10 に赤外線リモコン受信モジュール OSRB38C9AA の出力を示します. 入力はリモコンからの赤外線信号で, 出力は論理レベルの H・L です. この図の通り赤外線信号を受信していない間は H を, 受信している間は L を, それぞれ出力します. この出力において副搬送波は除去されている点に注意してください.

22.5　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 22.3: 【ソースファイル一覧】 パワーパック: テレビリモコン式

初	頁	番号	ファイル名	内容	行数
✓	355	22.2	PowerPack-IR.c	パワーパックメイン関数	67
✓	356	22.3	pp-control-ir.c	テレビリモコン制御パワーパック: 実装	134
	193	10.4	pp-control-vr.c	ボリューム制御インターフェース: 実装	58
✓	359	22.4	led-ir-mon.h	受信モニタ LED: ヘッダ	10
✓	359	22.5	led-ir-mon.c	受信モニタ LED: 実装	27
✓	360	22.6	keysym.h	キーシンボル: ヘッダ	32
✓	360	22.7	cmd.h	遠隔制御実行: ヘッダ	12
✓	361	22.8	cmd.c	遠隔制御実行: 実装	112
✓	363	22.9	ir-frame.h	赤外線フレームデコーダ: ヘッダ	10
✓	365	22.10	ir-frame.c	赤外線フレームデコーダ: 実装	110
✓	367	22.11	rc.h	テレビリモコン情報: ヘッダ	45
✓	367	22.12	rc.c	テレビリモコン情報: 実装	30
✓	368	22.13	rc-data.c	各種テレビリモコン仕様: 実装	315
✓	373	22.14	auto-sel.c	テレビリモコン自動選択機能: 実装	63
✓	374	22.15	if.h	ポイント制御: ヘッダ	8
✓	374	22.16	if-toq.h	ポイント切替動作キュー: ヘッダ	12
✓	374	22.17	if-toq-dummy.c	ポイント切替動作キュー: ダミー実装	44
	189	10.1	pp.h	パワーパック関数宣言: ヘッダ	46
	172	9.2	pp-const.h	パワーパック定数: ヘッダ	17
	191	10.3	pp-misc.c	パワーパック諸関数: 実装	111
	196	10.6	pp-sb.h	低速ブースト機能: ヘッダ	8
	198	10.7	pp-sb.c	低速ブースト機能: 実装	171
	129	6.1	incr-lin.h	線形増加アルゴリズム: ヘッダ	9
	130	6.2	incr-lin.c	線形増加アルゴリズム: 実装	78
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76
✓	354	22.1	io.h	I/O ピン割付: ヘッダ	45
✓	375	22.18	mcr.h	MCR・周辺モジュール: ヘッダ	9
✓	375	22.19	mcr.c	MCR・周辺モジュール: 実装	161
	175	9.6	mcr-motor.h	モーター制御周辺モジュール: ヘッダ	25
	177	9.7	mcr-motor.c	モーター制御周辺モジュール: 実装	98

(つづく)

(つづき)

初　頁	番号	ファイル名	内容	行数

計 1952 行 (うち初出 1246 行, 既出 706 行)

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

ソースファイル番号 22.1: io.h

```
#if !defined(__IO_H__)
#define __IO_H__

/* 設定VR ADC 走行速度: ANA0 */
#define ADC_CH_SPEED            0x00
/* 設定VR ADC 最高速度: ANA1 */
#define ADC_CH_VMAX             0x01
/* 設定VR ADC 低速ブースト周波数設定: ANA2 */
#define ADC_CH_SBFREQ           0x02
/* 設定VR ADC 低速ブーストパルス幅設定: ANA3 */
#define ADC_CH_SBPWDT           0x03
/* 設定VR ADC 加速度設定: ANA4 */
#define ADC_CH_ACCEL            0x04
/* 設定VR ADC 減速度設定: ANA5 */
#define ADC_CH_DECEL            0x05
/* 設定VR ADC - 常点灯設定: ANA7 */
#define ADC_CH_LIGHT            0x07

/* 状態インジケータLED: RC3 */
#define LED_INDICATOR_ON()      (LATC3 = 1)
#define LED_INDICATOR_OFF()     (LATC3 = 0)
#define LED_INDICATOR_TOGGLE()  (LATC3 = ~LATC3)
/* 停車中表示 LED: RC4 */
#define LED_STOPPED_ON()        (LATC4 = 1)
#define LED_STOPPED_OFF()       (LATC4 = 0)
#define LED_STOPPED_TOGGLE()    (LATC4 = ~LATC4)
/* テレビリモコン受信モニタLED: RB5 */
#define LED_IR_MON_ON()         (LATB5 = 1)
#define LED_IR_MON_OFF()        (LATB5 = 0)
#define LED_IR_MON_TOGGLE()     (LATB5 = ~LATB5)
#define LED_IR_MON_IS_ON()      (LATB5 == 1)

/* 進行方向スイッチFWD: RC0 */
#define SW_DIR1F()              (RC0 == 0)
/* 進行方向スイッチREV: RC1 */
#define SW_DIR1R()              (RC1 == 0)
/* 緊急停止ボタン: RA6 */
#define SW_EMG_STOP()           (RA6 == 0)
/* 低速ブースト有効スイッチ: RC2 */
#define SW_SBEN()               (RC2 == 0)
/* 制御モード切替(リモコン/手動): RB4 */
#define SW_CTRMODE_REMOTE()     (RB4 == 0)
#define SW_CTRMODE_MANUAL()     (RB4 == 1)

#endif /*__IO_H__*/
```

ソースファイル PowerPack-IR.c (パワーパックメイン関数)

パワーパックのメイン関数です. この内容はこれまで紹介してきたパワーパックのメイン関数と同様ですが, テレビリモコン受信やポイント制御関係の初期化が加わっています. ただしポイント切替制御関係の関数は本章のファームウエアでは何もしない関数をダミー実装しますので今は気にせず読み飛ばして下さい.

ソースファイル番号 22.2: PowerPack-IR.c

```
/*************************************************************
 * パワーパック(低速ブースト・加減速機能PWM型・RS-232C制御)
 *   / PIC16F18857
 * by Hirotsugu Kakugawa
 *************************************************************/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "mcr-motor.h"
#include "io.h"
#include "ir-frame.h"
#include "keysym.h"
#include "rc.h"
#include "cmd.h"
#include "adc.h"
#include "if.h"
#include "if-toq.h"
#include "pp.h"
#include "pp-const.h"
#include "pp-sb.h"

enum ctrmode_e fsm_ctrmode = CTRMODE_REMOTE;
extern void auto_select(void);

/*-----------------------------------------------------------
 * MAIN
 *-----------------------------------------------------------*/
void main(void)
{
  /* システム初期化 */
  System_Init();
  /* ウオッチドッグタイマ: 256 [mSec] */
  WDTCON0bits.WDTPS = 0x08; /* 0b01000 */
  CLRWDT();
  /* モータ出力初期化 */
  MOTOR_DIR_NONE();
  MOTOR_PWM_START();
  /* LED */
  LED_INDICATOR_OFF();  /* 状態インジケータLED */
  LED_STOPPED_OFF();    /* 停車中表示LED */
  __delay_ms(100);
  CLRWDT();
  /* ポイント切替動作キュー初期化 */
  if_toq_init();
  /* 低速ブースト初期化 */
  pp_sb_init();
  /* 割り込み許可 */
  GIE  = 1;
  PEIE = 1;
  /* 制御モード: リモコン/手動 */
  if (SW_CTRMODE_REMOTE()) {
    fsm_ctrmode = CTRMODE_REMOTE;
    auto_select();      /* リモコン自動選択 */
    LED_IR_MON_ON();    /* リモコン受信モニタ点灯 */
  } else {
    fsm_ctrmode = CTRMODE_MANUAL;
    pp_home_position(); /* ラッシュ防止機能 */
  }
  /* 主ループ */
  for (;;) {
    CLRWDT();
    /* フレーム受信フラグをクリア */
    ir_rcv_flag = 0;
    /* パワーパック走行制御 */
    pp_control();
  }
}
```

ソースファイル pp-control-ir.c (テレビリモコン制御パワーパック: 実装)

このソースファイルでパワーパックの機能の実質的な実現を行いますが, これまでに見てきたパワーパックのソースファイルと比べてあまり変わることはありません. 違いは以下の 2 点が追加された程度です. なお手動制御モード時は従来の通りの動作となります.

- テレビリモコンの信号を受信してパワーパック操作に対応する動作を行う.
- ポイント切替制御関係の関数を呼び出す.

これまでのパワーパックでは, 走行方向や走行速度設定ボリュームなどのパラメータの値をマイコンの I/O ポートや ADC などのハードウエアより読んで, 対応する変数に設定していました. 本ファームウエアではこの点の変更を行います. 具体的には, 後述のソースファイル cmd.c がテレビリモコンより受信した信号に対応するパラメータ値を求め, 対応する変数に設定します. このソースファイル pp-control-ir.c ではパラメータに関してはハードウエアに関係ない形で記述していますので, 従来のパワーパックのソースファイルと同様な内容となっています. このようにハードウエアのレイヤとアプリケーションのレイヤの間に抽象化レイヤを入れることで非常に見通しの良いソフトウエア構造になります.

ソースファイル番号 22.3: pp-control-ir.c

```
1  /*------------------------------------------------------
2   * パワーパック走行制御 (赤外線式テレビリモコン)
3   *------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "mcr-motor.h"
8  #include "adc.h"
9  #include "io.h"
10 #include "ir-frame.h"
11 #include "keysym.h"
12 #include "rc.h"
13 #include "cmd.h"
14 #include "led-ir-mon.h"
15 #include "if.h"
16 #include "if-toq.h"
17 #include "pp.h"
18 #include "pp-const.h"
19 #include "pp-sb.h"
20 #include "incr-lin.h"
21
22 uint16_t   speed = 0;
23 uint16_t   target_speed = 0;    /* リモコンからの制御 */
24 enum dir_e dir_next = DIR_NONE; /* リモコンからの制御 */
25 enum dir_e dir_curr = DIR_NONE;
26 uint8_t    pp_reset = 0;        /* リモコンからのセット要求*/
27
28 /*------------------------------------------------------
29  * 走行制御 (リモコン運転モード/手動運転モード)
30  *------------------------------------------------------*/
31 void pp_control(void)
32 {
33   /* モータドライバIC のリセット */
34   MotorDriver_TB6643KQ_Reset();
35   /* 現在走行方向初期化 */
36   dir_next = DIR_NONE;
37   dir_curr = DIR_NONE;
38   pp_motor_dir_set(dir_curr);
39   /* 現在走行速度初期化 */
40   speed = 0;
41   /* 目標走行速度初期化 */
42   target_speed = 0;
43   /* 速度制御初期化 */
```

```
44    incr_lin_init(0);
45    pp_sb_duty_set(0);
46    /* インジケータLEDの点滅時間管理 */
47    uint16_t t_led = 0;
48    /* 状態インジケータLED */
49    LED_INDICATOR_OFF();
50    /* 停車中表示LED */
51    LED_STOPPED_OFF();
52    /* コマンド実行の初期化（リモコン制御モード時） */
53    if (fsm_ctrmode == CTRMODE_REMOTE) {
54      cmd_init();
55    }
56    /* 主ループ */
57    for (;;) {
58      __delay_ms(RUN_INTERVAL);
59      CLRWDT();
60      /* 緊急停止 */
61      if (SW_EMG_STOP()) {
62        break;
63      }
64      /* ポイント切替制御 */
65      if_toq_next(RUN_INTERVAL);
66      /* パラメータの読込 */
67      pp_param_read();
68      /* 走行制御 */
69      if (fsm_ctrmode == CTRMODE_REMOTE) { /* リモコン制御モード */
70        led_ir_mon_interval(RUN_INTERVAL);
71        if (ir_rcv_flag != 0) {
72          if (cmd_action(rc_keysym(ir_data_buff))) {
73            led_ir_mon_on();
74          }
75          ir_rcv_flag = 0;
76          if (pp_reset != 0) {
77            break;
78          }
79        }
80      } else {                                /* 手動制御モード */
81        /* 走行方向の読込 */
82        dir_next = pp_dir_read();
83        /* 目標走行速度 */
84        target_speed = pp_controler_speed(speed);
85      }
86      /* 走行方向の切替(必要に応じて) */
87      if (dir_curr == dir_next) { /* 同一の走行方向が継続 */
88        if (dir_next == DIR_NONE) {
89          LED_INDICATOR_OFF(); /* ニュートラル */
90        } else {
91          LED_INDICATOR_ON();  /* 前進/後進*/
92        }
93        t_led = 0;
94      } else { /* 走行方向SW切替が発生 */
95        if (speed == 0) { /* 現在走行速度が0ならば切替 */
96          if (dir_next == DIR_NONE) {
97            LED_INDICATOR_OFF(); /* ニュートラル */
98          } else {
99            LED_INDICATOR_ON();  /* 前進/後進*/
100         }
101         dir_curr = dir_next;
102         pp_motor_dir_set(dir_curr);
103         t_led = 0;
104       } else { /* 走行中の方向切替は禁止: 警告点灯(LED点滅) */
105         if (t_led >= T_LED_WARNING) {
106           LED_INDICATOR_TOGGLE();
107           t_led = 0;
108         } else {
109           t_led += RUN_INTERVAL;
110         }
111       }
112     }
```

```
113       /* 加減速度 */
114       uint16_t a = pp_controler_accel(target_speed, speed);
115       /* 次の走行速度 */
116       speed = incr_lin_next(a, SPEED_INC_DY, target_speed);
117       /* 走行速度を変換して，モーターPWMデューティー比を得る */
118       uint16_t md = PP_CONV_SPEED_TO_MDUTY(speed, p_vmax);
119       /* モーターPWMデューティー比を出力 */
120       pp_sb_duty_set(md);
121       /* 停車中表示 LED */
122       if ((speed == 0) || (dir_curr == DIR_NONE)) {
123         LED_STOPPED_ON();
124       } else {
125         LED_STOPPED_OFF();
126       }
127     }
128     /* 停止 */
129     MotorDriver_TB6643KQ_Reset();
130     pp_motor_dir_set(DIR_NONE);
131     pp_sb_duty_set(0);
132     LED_INDICATOR_OFF();
133     cmd_reset();
134   }
```

図 22.11 に赤外線信号の変化によって引き起こされる関数呼び出しとコマンドの実行の関係を示します. 通常は関数 pp_control() の主ループが実行を続けています. 赤外線リモコン受信モジュールの出力の変化は内蔵周辺モジュール IOC がこれを捕捉するともに割り込みを発生させます. 割り込みハンドラでは, 赤外線リモコン受信モジュールの出力の値を得るとともに, タイマ TMR3 の値によりパルスの幅を得ます. そして図 22.14

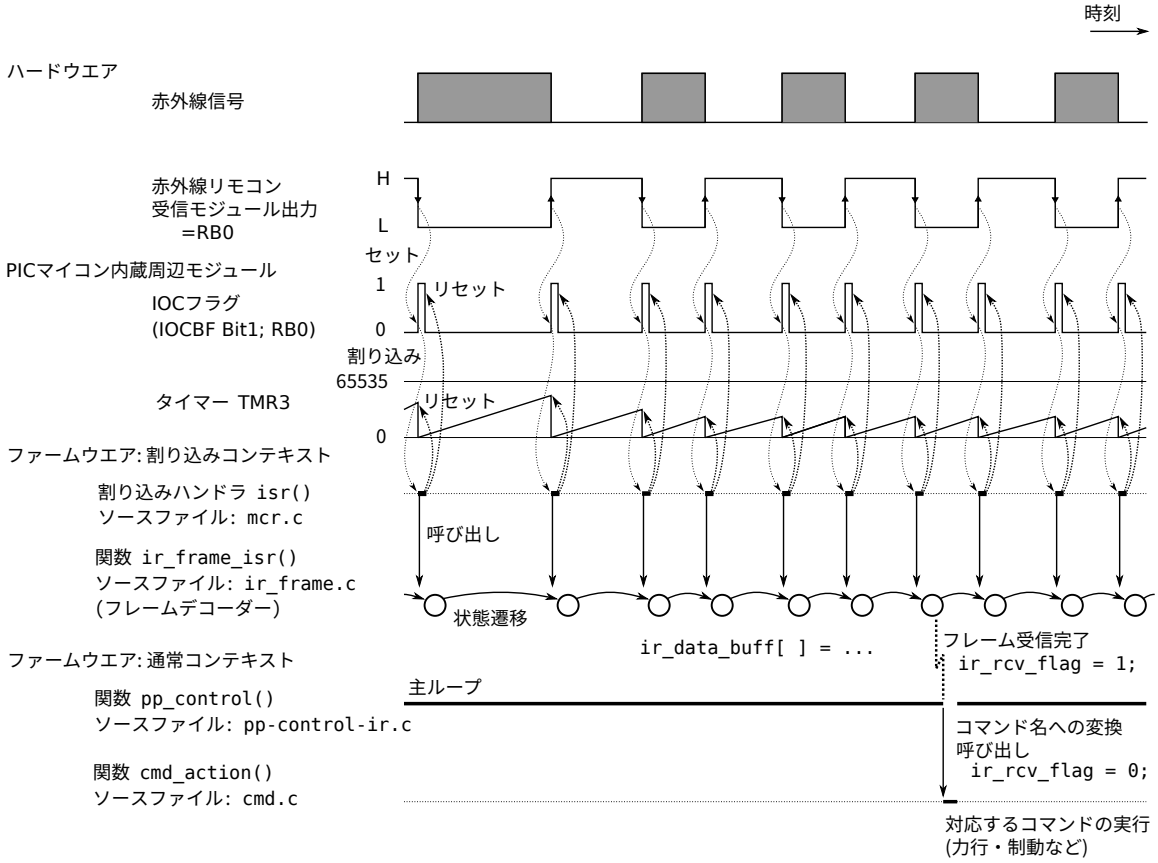


図 22.11: 赤外線信号の変化からコマンドの実行までの関数呼出しの関係

で示した赤外線フレームデコーダによりフレームを構成します. 4 バイトのフレームは変数 ir_data_buff[] に格納されてゆき, フレームが完成すれば変数 ir_rcv_flag を 1 にします. もちろんフレームが完成するのは割り込み発生ごとではなく, 何度も割り込みが発生することで幾つものパルスを受信してはじめてフレームは完成します. 割り込みハンドラの実行が終了すると, 関数 pp_control() の主ループの実行に復帰します. 関数 pp_control() は繰り返し ir_rcv_flag の値を調べていて, その値が 1 になったことを検知すると受信フレームの内容をコマンド名に解釈し, 関数 cmd_action() を呼び出して力行や制動などの対応する処理を行います. なお割り込み処理には上記のほかに低速ブースト機能を実現する処理もありますが, 赤外線フレーム受信とは独立した動作なのでここでは気にする必要はありません.

ソースファイル led-ir-mon.h (受信モニタ LED: ヘッダ)

この後に紹介するソースファイル led-ir-mon.c で実装する関数群のプロトタイプ宣言を行います.

ソースファイル番号 22.4: led-ir-mon.h

```
1  #if !defined (__LED_IR_MON__)
2
3  #define __LED_IR_MON__
4
5  #define LED_IR_MON_TIME  100  /* mSec */
6
7  extern void led_ir_mon_on(void);
8  extern void led_ir_mon_interval(int16_t t);
9
10 #endif /*__LED_IR_MON__*/
```

ソースファイル led-ir-mon.c (受信モニタ LED: 実装)

赤外線フレームの受信を LED を点滅して視覚的に確認できる機能を実現します. 通常動作中は LED は点灯し続けますが, 赤外線フレームを受信したときに一瞬だけ消灯します. 具体的には, LED を消灯してから一定時間経過すると再び LED を点灯することでこの機能を実現します. 消灯期間は 100 ミリ秒としています.

LED が実際に点滅するのは選択された赤外線リモコンから, しかも表 22.1 に記載のボタンが押された場合に限ります. この場合にのみ関数 pp_control() の中から関数 led_ir_mon_on() を呼び出しています.

ソースファイル番号 22.5: led-ir-mon.c

```
1  /*-----------------------------------------------------
2   * モニタ用LED点灯制御
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "io.h"
8  #include "led-ir-mon.h"
9
10 static int16_t  timer = 0;
11
12 /* 通知開始 */
13 void led_ir_mon_on(void)
14 {
15   LED_IR_MON_OFF();  /* 消灯 */
16   timer = LED_IR_MON_TIME;
17 }
18
19 /* 時間経過 */
20 void led_ir_mon_interval(int16_t t)
21 {
22   if (timer >= t) {
23     timer = timer - t;
24   } else { /* 一定時間経過 */
```

```
25      LED_IR_MON_ON(); /* 点灯 */
26    }
27  }
```

ソースファイル keysym.h (キーシンボル: ヘッダ)

テレビリモコンのボタンの抽象化を行います. 赤外線パケットに含まれるデータは単なる 4 バイトの数値列ですが, その数値に対応するテレビリモコンのボタンの記号定数を定め, キーシンボルと以降では呼ぶことにします. 構造体 keysym_s によりキーシンボルと 4 バイトの数値列の対応を表現します.

ソースファイル番号 22.6: keysym.h

```
1   #if !defined(__KEYSYM_H__)
2   #define __KEYSYM_H__
3
4   /* キーシンボル */
5   enum keysym_e {
6     KEYSYM_NULL = 0,
7     KEYSYM_PWR,
8     KEYSYM_CH1,
9     KEYSYM_CH2,
10    KEYSYM_CH3,
11    KEYSYM_CH4,
12    KEYSYM_CH5,
13    KEYSYM_CH6,
14    KEYSYM_CH7,
15    KEYSYM_CH8,
16    KEYSYM_CH9,
17    KEYSYM_CH10,
18    KEYSYM_CH11,
19    KEYSYM_CH12,
20    KEYSYM_CHUP,
21    KEYSYM_CHDN,
22    KEYSYM_VOLUP,
23    KEYSYM_VOLDN,
24  };
25
26  /* キーシンボル表 */
27  struct keysyms_s {
28    enum keysym_e keysym;
29    uint8_t  b0, b1, b2, b3;
30  };
31
32  #endif /*__KEYSYM_H__*/
```

ソースファイル cmd.h (遠隔制御実行: ヘッダ)

遠隔制御を実行するソースファイル cmd.c 内の関数のプロトタイプ宣言を行います.

ソースファイル番号 22.7: cmd.h

```
1   #if !defined(__CMD_H__)
2   #define __CMD_H__
3
4   /* スピードステップ */
5   #define CMD_SPEED_STEPS    32
6   #define CMD_SPEED_CONV(S)  ((SPEED_MAX * (unsigned)(S)) >> 5)
7
8   extern void   cmd_init(void);
9   extern void   cmd_reset(void);
10  extern int8_t cmd_action(enum keysym_e keysym);
11
12  #endif /*__CMD_H__*/
```

ソースファイル cmd.c (遠隔制御実行: 実装)

リモコンからの遠隔制御を実行します. リモコンからの受信内容に対応するよう, 走行の速度や方向などのパラメータを変更します. 実際の走行の制御はソースファイル pp-control-ir.c の中で行われることは既に述べた通りです. ポイント制御関係の処理も行いますが, 詳しくは第 23 章で説明します.

走行速度の管理には PWM デューティ比を直接は扱わず, スピードステップ方式を採用しています. スピードステップ 0 から 31 までの値の 32 ステップとし, スピードステップの値 0 から 31 を PWM デューティ比 0% から 100% に換算します. つまりテレビリモコンの**音量大**ボタンを 31 回押せば走行速度は最高になります. スピードステップ数が少ないと微妙な速度設定ができず不便ですし, スピードステップ数が多いと多くの回数ボタンを押さないと加速ができず不便です. そこでちょうど良いスピードステップ数として 32 を選びました.

ソースファイル番号 22.8: cmd.c

```
1  /*------------------------------------------------------
2   * テレビリモコンからのコマンドの実行
3   *------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include "mcr.h"
9  #include "io.h"
10 #include "pp.h"
11 #include "pp-const.h"
12 #include "keysym.h"
13 #include "cmd.h"
14 #include "if.h"
15 #include "if-toq.h"
16
17 /* 状態 */
18 uint8_t  cmd_speed_step     = 0;
19 uint16_t cmd_turnout_number = 0;
20
21 extern uint16_t   speed;
22 extern uint16_t   target_speed;
23 extern enum dir_e dir_curr;
24 extern enum dir_e dir_next;
25 extern uint8_t    pp_reset;  /* リセット要求 */
26
27 /*------------------------------------------------------
28  * 初期化
29  *------------------------------------------------------*/
30 void cmd_init(void)
31 {
32   pp_reset = 0;
33   cmd_reset();
34 }
35
36 /*------------------------------------------------------
37  * リセット
38  *------------------------------------------------------*/
39 void cmd_reset(void)
40 {
41   cmd_speed_step = 0;
42   target_speed = CMD_SPEED_CONV(cmd_speed_step);
43   dir_next = DIR_NONE;
44 }
45
46 /*------------------------------------------------------
47  * 実行
48  *------------------------------------------------------*/
49 int8_t cmd_action(enum keysym_e keysym)
50 {
51   int8_t cmd_done = 1;
52   switch (keysym) {
53   default:
```

```
54      cmd_done = 0;
55      break;
56    case KEYSYM_PWR: /* 緊急停止 */
57      pp_reset = 1;
58      cmd_reset();
59      break;
60    case KEYSYM_CH11: /* 走行方向: 前 */
61      dir_next = DIR_FWD;
62      break;
63    case KEYSYM_CH12: /* 走行方向: 後 */
64      dir_next = DIR_REV;
65      break;
66    case KEYSYM_CHUP: /* ポイント切替: 定位 */
67      if_toq_set_normal(cmd_turnout_number);
68      break;
69    case KEYSYM_CHDN: /* ポイント切替: 反位 */
70      if_toq_set_reverse(cmd_turnout_number);
71      break;
72    case KEYSYM_VOLUP: /* 走行速度 増*/
73      if ((dir_curr != DIR_NONE) &&
74          (cmd_speed_step < CMD_SPEED_STEPS-1)) {
75        cmd_speed_step++;
76        target_speed = CMD_SPEED_CONV(cmd_speed_step);
77      }
78      break;
79    case KEYSYM_VOLDN: /* 走行速度 減*/
80      if ((dir_curr != DIR_NONE) &&
81          (cmd_speed_step > 0)) {
82          cmd_speed_step--;
83          target_speed = CMD_SPEED_CONV(cmd_speed_step);
84      }
85      break;
86    case KEYSYM_CH1: /* ポイント番号設定 */
87      cmd_turnout_number = 1;
88      break;
89    case KEYSYM_CH2: /* ポイント番号設定 */
90      cmd_turnout_number = 2;
91      break;
92    case KEYSYM_CH3: /* ポイント番号設定 */
93      cmd_turnout_number = 3;
94      break;
95    case KEYSYM_CH4: /* ポイント番号設定 */
96      cmd_turnout_number = 4;
97      break;
98    case KEYSYM_CH5: /* ポイント番号設定 */
99      cmd_turnout_number = 5;
100     break;
101   case KEYSYM_CH6: /* ポイント番号設定 */
102     cmd_turnout_number = 6;
103     break;
104   case KEYSYM_CH7: /* ポイント番号設定 */
105     cmd_turnout_number = 7;
106     break;
107   case KEYSYM_CH8: /* ポイント番号設定 */
108     cmd_turnout_number = 8;
109     break;
110   }
111   return cmd_done;
112 }
```

テレビリモコンのボタン押下からパワーパックの動作までのデータの流れを図 22.12 に示します. 図の左側がテレビリモコンを, 右側がパワーパックを表します. この図では加速する場合を例として, 日立製作所のテレビリモコンの場合に送信されるデータを描いています. 赤外線信号として実際に送信されるデータの内容は使用するテレビリモコンの種類によって異なります. つまり同じ加速する場合でも, 異なるテレビリモコンの場合では異なるデータが送信されます.

テレビリモコンごとに加速や減速などの動作を記述するのは実に面倒ですので, 以下に説明する抽象化の導

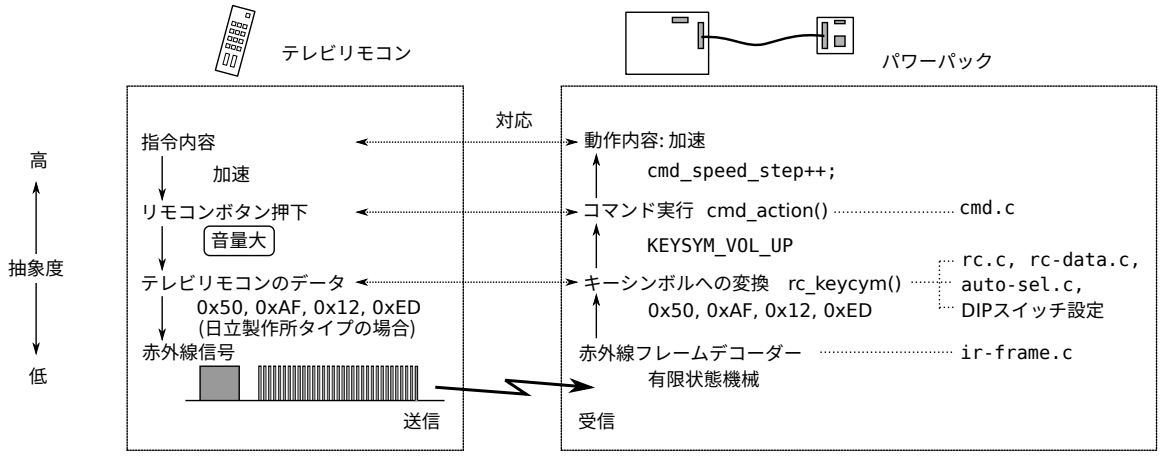


図 22.12: テレビリモコンのボタン押下からパワーパックの動作までのデータの流れ

入によりファームウエアを簡潔にします. まずファームウエアの下位の層では実際の 4 バイトの受信データからテレビリモコンのボタンを抽象化したキーシンボルへ変換します. そしてファームウエアの上位の層ではキーシンボルの値に基づいてパワーパックの動作を行います. 図 22.12 では, テレビリモコンとパワーパックで同じ抽象レベルに対応する部分では, 扱うデータは互いに対応するものになっている点に注意して下さい. 上位の層では下位の層を意識する必要をなくしています. この設計方法の採用により, 多種多様なリモコンを使用可能にする下位の層と, 多種多様な機能を持つパワーパックを実現する上位の層とを分離して, 多機能ながらも単純な構造を保ったファームウエアとなっています. このファームウエアの構造を活用すれば赤外線テレビリモコンだけでなく, 無線 (WiFi, Bluetooth, ZigBee 等) や有線 (RS-232C, RS-485, Ethernet 等) でリモコン制御可能なパワーパックの実現は容易に可能です. なぜなら通信固有の部分とキーシンボル変換の部分を作成するだけで良いからです.

ソースファイル ir-frame.h (赤外線フレームデコーダ: ヘッダ)

受信したパルスから赤外線フレームの 4 バイトを再構成するデコーダを実現する関数のプロトタイプ宣言です.

ソースファイル番号 22.9: ir-frame.h

```
1  #if !defined(__IR_FRAME_H__)
2  #define __IR_FRAME_H__
3
4  #define IR_FRAME_SIZE   4
5
6  extern void  ir_frame_isr(uint8_t lev, uint16_t t);
7  extern volatile uint8_t  ir_data_buff[IR_FRAME_SIZE];
8  extern volatile uint8_t  ir_rcv_flag;
9
10 #endif /*__IR_FRAME_H__*/
```

ソースファイル ir-frame.c (赤外線フレームデコーダ: 実装)

受信したパルスから赤外線フレームの 4 バイトを再構成するデコーダを実現する関数の実装です. テレビリモコンはパルスの時間幅で送信データを表現していますので, このソースファイルではパルスの時間幅からヘッダやビットの値の判定を行います. 4 バイトのデータの構成ができればバッファ変数 ir_data_buff にデータをコピーし, フラグ変数 ir_rcv_flag で受信フラグを立てます. この部分の大まかな構造を図 22.13 に示します.

```
1  IOC割り込みハンドラ {
2    siglev = 赤外線受信モジュールの出力;
3    sigpw  = 赤外線のパルスの時間幅;
4    ir_frame_isr(siglev, sigpw);
5  }
6
7  ir_frame_isr(siglev, sigpw)
8  {
9    siglev(赤外線受信モジュールの出力)とsigpw(パルスの時間幅)の値に
10     基づいて，フレームデコーダを状態遷移図に従って動作させる;
11   それと同時に受信データをrcv_buff[]に記録してゆく;
12   if (フレームの終わりに到達) {
13     rcv_buff[]の内容をir_data_buff[]へコピーする;
14     ir_rcv_flag = 1;
15   }
16 }
```

図 22.13: 赤外線フレームデコーダの構造 (概要)

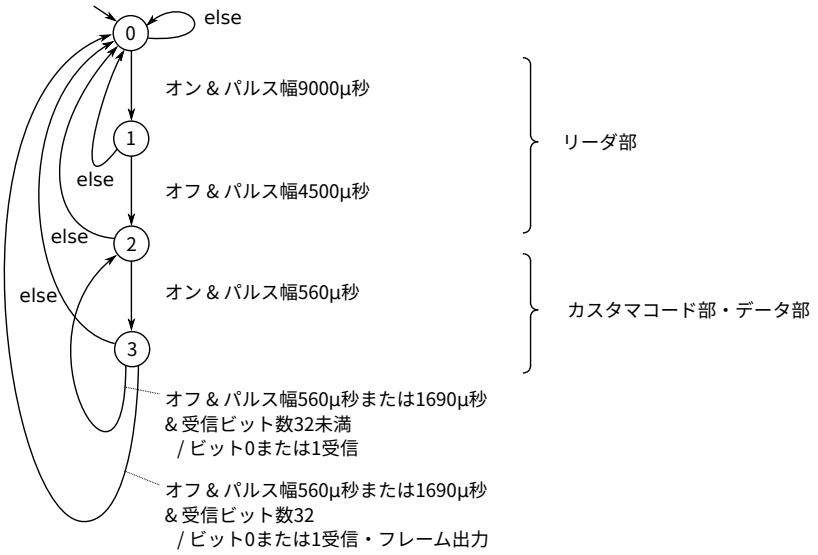


図 22.14: 赤外線フレームデコーダの有限状態遷移図

ソースファイル ir-frame.c では関数 ir_frame_isr() を実現しており, 以下にこの関数の説明をします. これはソースファイル mcr.c の割り込みハンドラから呼び出される関数で, これ自身も割り込みハンドラの一部です. この関数が呼び出されるのは, 図 22.10 でいえば赤外線リモコン受信モジュールの出力に変化が発生したときです. この割り込み発生は IOC 割り込み により実現しています. この関数の引数は赤外線リモコン受信モジュールの出力 siglev と, 前回の出力変化からの時間経過 sigpw の 2 つです. つまり赤外線リモコン受信モジュールの出力が siglev であるパルス幅が, sigpw であることを引数は表します.

関数 ir_frame_isr() の内部は図 22.14 に示す有限状態機械であり, この状態遷移に従ってテレビリモコンの送信するフレームのデコードを行います. この有限状態機械の構成は, 図 22.8a で示したフレームの構造と図 22.8b で示したパルス幅とビットの値の関係から容易に行えます.

DCC のパケットも同様にパルス幅でデータが表現されていますから, 上述の技法を応用すれば DCC デコー

ダのパケット受信部分も容易に実現可能です.

ソースファイル番号 22.10: ir-frame.c

```
/*------------------------------------------------------
 * 赤外線テレビリモコンのフレームの信号をデコード
 * (NECフォーマット限定)
 *------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "ir-frame.h"
#include "keysym.h"

/* フレームバッファ */
volatile uint8_t  ir_data_buff[IR_FRAME_SIZE];
/* フレーム受信フラグ */
volatile uint8_t  ir_rcv_flag = 0;

/*------------------------------------------------------
 * 信号変化割り込みハンドラ: フレームデコーダ
 *------------------------------------------------------*/

/* パルス時間幅の検査 */
#define PW_EQ(TMRVAL,TUS)                       \
  (((PW(TUS) - (PW(TUS) >> 1)) <= TMRVAL) && \
   (TMRVAL <= PW(TUS) + (PW(TUS) >> 1)))

/* マイクロ秒->TMRカウント値 変換式 */
/*    CLK=32MHz, CS=FOSC/4, Prescale=1:2 */
/*    式: ((TUS)*32/(4*2)) = ((TUS)*4) = ((TUS) << 2) */
#define PW(TUS)   ((uint16_t)(TUS) << 2)

/*パルス幅定数 */
#define T_LEADER_H  9000   /* 9000uSec */
#define T_LEADER_L  4500   /* 4500uSec */
#define T_REPEAT_L  2250   /* 2250uSec */
#define T_DATA_H     560   /*  560uSec */
#define T_DATA_L_0   560   /*  560uSec */
#define T_DATA_L_1  1690   /* 1690uSec */

/* フレームデコーダ状態 */
static volatile uint8_t rcv_state  = 0;
static volatile uint8_t rcv_nbits  = 0;
static volatile uint8_t rcv_nbytes = 0;
static volatile uint8_t rcv_buff[IR_FRAME_SIZE] = 0;

void ir_frame_isr(uint8_t siglev, uint16_t sigpw)
{
  switch (rcv_state) {
  default:
  case 0: /* リーダ H */
    if ((siglev == 1) && PW_EQ(sigpw, T_LEADER_H)) {
      rcv_state = 1;
      rcv_nbits  = 0;
      rcv_nbytes = 0;
    } else {
      rcv_state = 0;
    }
    break;
  case 1: /* リーダ/リピート L */
    if ((siglev == 0) && PW_EQ(sigpw, T_LEADER_L)) {
      rcv_state = 2; /* リーダ */
    } else if ((siglev == 0) && PW_EQ(sigpw, T_REPEAT_L)) {
      rcv_state = 0; /* リピート: 無視 */
    } else {
      rcv_state = 0;
    }
    break;
```

```
67    case 2: /* カスタムコード/データコード H */
68      if ((siglev == 1) && PW_EQ(sigpw, T_DATA_H)) {
69        rcv_state = 3;
70      } else {
71        rcv_state = 0;
72      }
73      break;
74    case 3: /* カスタムコード/データコード L */
75      if ((siglev == 0) && (PW_EQ(sigpw, T_DATA_L_0) ||
76                            PW_EQ(sigpw, T_DATA_L_1)) &&
77          (rcv_nbytes < IR_FRAME_SIZE)) {
78        rcv_state = 2;
79        /* 受信ビットの値 */
80        uint8_t data = 0x00;
81        if (PW_EQ(sigpw, T_DATA_L_1)) {
82          data = 0x80;
83        }
84        /* 受信フレームの構成 */
85        rcv_buff[rcv_nbytes]
86          = data | ((unsigned)rcv_buff[rcv_nbytes] >> 1);
87        rcv_nbits++;
88        if (rcv_nbits == 8) {
89          rcv_nbits = 0;
90          rcv_nbytes++;
91          if (rcv_nbytes == IR_FRAME_SIZE) {
92            rcv_state = 0;
93            /* ローカルバッファir_data_buff[]へコピー */
94            if (ir_rcv_flag == 0) {
95              uint8_t i = 0;
96              for (i = 0; i < IR_FRAME_SIZE; i++) {
97                ir_data_buff[i] = rcv_buff[i];
98              }
99              ir_rcv_flag = 1;
100           } else {
101             ; /* フレーム破棄 */
102           }
103         }
104       }
105     } else {
106       rcv_state = 0;
107     }
108     break;
109   }
110 }
```

2 つの配列 rcv_buff[] ir_data_buff[] を用意して, いったん rcv_buff[] に受信してその内容を ir_data_buff[] にコピーするのはなんだか無駄な動作のように見えます. 仮に 1 つの配列 ir_data_buff[] だけを使った場合に何が起きるかを考えてみましょう. フレームを受信した後は ir_data_buff[] の内容を解析しながら処理動作を行いますが, その間にも次の赤外線の信号がやってくる可能性があります. すると処理動作の真っ最中に, 割り込みハンドラ内で ir_data_buff[] の内容の書き換えが起きてしまい, 処理動作がおかしくなってしまいます.

このようなことが起きないように, 上記のソースファイルでは以下の構造のプログラムにしています.

- 割り込みハンドラが呼び出すフレームデコーダ内:
  ir_rcv_flag が 0 の場合に限り, rcv_buff[] から ir_data_buff[] へコピーします. 1 の場合には処理動作が未完了なので受信フレームは破棄します.
- 処理動作の部分
  受信したフレームに対する処理動作が終われば ir_rcv_flag の値を 1 に設定して, 次のフレームの受信が可能であることを表します.

ソースファイル rc.h (テレビリモコン情報: ヘッダ)

動作モードやリモコン型を表す記号定数やリモコン仕様を表す構造体などを定義します.

ソースファイル番号 22.11: rc.h

```
#if !defined(__RC_H__)
#define __RC_H__

/* 制御モード */
enum ctrmode_e {
  CTRMODE_MANUAL, /* 手動制御モード */
  CTRMODE_REMOTE  /* リモコン制御モード */
};
extern enum ctrmode_e fsm_ctrmode;

/* リモコン型 */
enum rc_type_e {
  RC_TYPE_UNDEF = 0,
  RC_TYPE_ANY,         /* リモコン自動選択 / フレームダンプ */
  RC_TYPE_HITACHI_1,   /* 日立製作所 */
  RC_TYPE_HITACHI_2,   /* 日立製作所 */
  RC_TYPE_HITACHI_3,   /* 日立製作所 */
  RC_TYPE_HITACHI_4,   /* 日立製作所 */
  RC_TYPE_TOSHIBA_1,   /* 東芝 */
  RC_TYPE_TOSHIBA_2,   /* 東芝 */
  RC_TYPE_SANYO_1,     /* 三洋 */
  RC_TYPE_FUNAI_DX_1, /* フナイ/DX */
  RC_TYPE_NANAO_1,     /* ナナオ */
  RC_TYPE_LG_1,        /* LG */
  RC_TYPE_LG_2,        /* LG */
  RC_TYPE_BYDESIGN_1, /* BYD:SIGN */
};

/* リモコン仕様 */
struct rcspec_s {
  const char              *name;
  enum rc_type_e           type;
  const struct keysyms_s  *keysyms;
};

typedef uint8_t rcs_id_t;
#define RCS_ID_NULL  0
extern rcs_id_t rcs_id;

extern const struct rcspec_s *rcspec[];
#define RCSPEC_MAX_N_ITEMS  32

extern enum keysym_e rc_keysym(volatile uint8_t *buff);

#endif /*__RC_H__*/
```

ソースファイル rc.c (テレビリモコン情報: 実装)

このソースファイルでは引数に与えられた受信フレーム内容の 4 バイトをキーシンボルに変換する関数 rc_keysym() を実装します. フレームとキーシンボルの対応関係は, リモコン仕様表 rcspec[] の中で現在選択されているリモコン型に該当する部分を参照します.

ソースファイル番号 22.12: rc.c

```
/*-------------------------------------------------------
 * 赤外線テレビリモコンの仕様表へのアクセス関数
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "ir-frame.h"
```

```
#include "keysym.h"
#include "rc.h"

/* リモコン仕様番号 */
rcs_id_t rcs_id = 0;

/*------------------------------------------------------
 * フレームからキーシンボルへ変換
 *------------------------------------------------------*/
enum keysym_e rc_keysym(volatile uint8_t *buff)
{
  const struct keysyms_s *kst = rcspec[rcs_id]->keysyms;
  uint8_t k = 0;
  for (k = 0; kst[k].keysym != KEYSYM_NULL; k++) {
    if ((kst[k].b0 == buff[0]) &&
        (kst[k].b1 == buff[1]) &&
        (kst[k].b2 == buff[2]) &&
        (kst[k].b3 == buff[3])) {
      return kst[k].keysym; /* 登録あり */
    }
  }
  return KEYSYM_NULL; /* 登録なし */
}
```

ソースファイル rc-data.c (各種テレビリモコン仕様: 実装)

各テレビリモコン固有の情報を表の形で記述します. 具体的には各キーシンボルとそれに対応する 4 バイトの赤外線パケットのデータの対により表現します. また表示をするためにテレビリモコンの文字列による名称も与えます. リモコン仕様表 rcspec[] はここで定義されます.

ソースファイル番号 22.13: rc-data.c

```
/*------------------------------------------------------
 * 赤外線テレビリモコンの仕様表
 *------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "ir-frame.h"
#include "keysym.h"
#include "rc.h"

/* キーンボル表 */
const struct keysyms_s keysyms_null[] = {
  { KEYSYM_NULL, 0x00, 0x00, 0x00, 0x00 },
};
const struct keysyms_s keysyms_hitachi_1[] = {
  /* 09101, 09102, 09105 | 213 */
  { KEYSYM_PWR,   0x50, 0xAF, 0x17, 0xE8 },
  { KEYSYM_CH1,   0x56, 0xA9, 0x71, 0x8E },
  { KEYSYM_CH2,   0x56, 0xA9, 0x72, 0x8D },
  { KEYSYM_CH3,   0x56, 0xA9, 0x73, 0x8C },
  { KEYSYM_CH4,   0x56, 0xA9, 0x74, 0x8B },
  { KEYSYM_CH5,   0x56, 0xA9, 0x75, 0x8A },
  { KEYSYM_CH6,   0x56, 0xA9, 0x76, 0x89 },
  { KEYSYM_CH7,   0x56, 0xA9, 0x77, 0x88 },
  { KEYSYM_CH8,   0x56, 0xA9, 0x78, 0x87 },
  { KEYSYM_CH9,   0x56, 0xA9, 0x79, 0x86 },
  { KEYSYM_CH10,  0x56, 0xA9, 0x70, 0x8F },
  { KEYSYM_CH11,  0x56, 0xA9, 0x7A, 0x85 },
  { KEYSYM_CH12,  0x56, 0xA9, 0x7B, 0x84 },
  { KEYSYM_CHUP,  0x50, 0xAF, 0x19, 0xE6 },
  { KEYSYM_CHDN,  0x50, 0xAF, 0x18, 0xE7 },
  { KEYSYM_VOLUP, 0x50, 0xAF, 0x12, 0xED },
  { KEYSYM_VOLDN, 0x50, 0xAF, 0x15, 0xEA },
  { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
};
const struct keysyms_s keysyms_hitachi_2[] = {
```

```
  /* --- | 249 */
  { KEYSYM_PWR,   0x50, 0xAF, 0x17, 0xE8 },
  { KEYSYM_CH1,   0x56, 0xA9, 0x0D, 0xF2 },
  { KEYSYM_CH2,   0x56, 0xA9, 0x0E, 0xF1 },
  { KEYSYM_CH3,   0x56, 0xA9, 0x0F, 0xF0 },
  { KEYSYM_CH4,   0x56, 0xA9, 0x1C, 0xE3 },
  { KEYSYM_CH5,   0x56, 0xA9, 0x1D, 0xE2 },
  { KEYSYM_CH6,   0x56, 0xA9, 0x1E, 0xE1 },
  { KEYSYM_CH7,   0x56, 0xA9, 0x1F, 0xE0 },
  { KEYSYM_CH8,   0x56, 0xA9, 0x04, 0xFB },
  { KEYSYM_CH9,   0x56, 0xA9, 0x05, 0xFA },
  { KEYSYM_CH10,  0x56, 0xA9, 0x0C, 0xF3 },
  { KEYSYM_CH11,  0x56, 0xA9, 0x06, 0xF9 },
  { KEYSYM_CH12,  0x56, 0xA9, 0x07, 0xF8 },
  { KEYSYM_CHUP,  0x50, 0xAF, 0x19, 0xE6 },
  { KEYSYM_CHDN,  0x50, 0xAF, 0x18, 0xE7 },
  { KEYSYM_VOLUP, 0x50, 0xAF, 0x12, 0xED },
  { KEYSYM_VOLDN, 0x50, 0xAF, 0x15, 0xEA },
  { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
};
const struct keysyms_s keysyms_hitachi_3[] = {
  /*--- | 251 */
  { KEYSYM_PWR,   0x50, 0xAF, 0x17, 0xE8 },
  { KEYSYM_CH1,   0x56, 0xA9, 0x0E, 0xF1 },
  { KEYSYM_CH2,   0x56, 0xA9, 0x0F, 0xF0 },
  { KEYSYM_CH3,   0x56, 0xA9, 0x1C, 0xE3 },
  { KEYSYM_CH4,   0x56, 0xA9, 0x1D, 0xE2 },
  { KEYSYM_CH5,   0x56, 0xA9, 0x1E, 0xE1 },
  { KEYSYM_CH6,   0x56, 0xA9, 0x1F, 0xE0 },
  { KEYSYM_CH7,   0x56, 0xA9, 0x04, 0xFB },
  { KEYSYM_CH8,   0x56, 0xA9, 0x05, 0xFA },
  { KEYSYM_CH9,   0x56, 0xA9, 0x0C, 0xF3 },
  { KEYSYM_CH10,  0x56, 0xA9, 0x06, 0xF9 },
  { KEYSYM_CH11,  0x56, 0xA9, 0x07, 0xF8 },
  { KEYSYM_CH12,  0x56, 0xA9, 0x49, 0xB6 },
  { KEYSYM_CHUP,  0x50, 0xAF, 0x19, 0xE6 },
  { KEYSYM_CHDN,  0x50, 0xAF, 0x18, 0xE7 },
  { KEYSYM_VOLUP, 0x50, 0xAF, 0x12, 0xED },
  { KEYSYM_VOLDN, 0x50, 0xAF, 0x15, 0xEA },
  { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
};
const struct keysyms_s keysyms_hitachi_4[] = {
  /* 09103 | --- */
  { KEYSYM_PWR,   0x50, 0xAF, 0x17, 0xE8 },
  { KEYSYM_CH1,   0x50, 0xAF, 0x0D, 0xF2 },
  { KEYSYM_CH2,   0x50, 0xAF, 0x0E, 0xF1 },
  { KEYSYM_CH3,   0x50, 0xAF, 0x0F, 0xF0 },
  { KEYSYM_CH4,   0x50, 0xAF, 0x1C, 0xE3 },
  { KEYSYM_CH5,   0x50, 0xAF, 0x1D, 0xE2 },
  { KEYSYM_CH6,   0x50, 0xAF, 0x1E, 0xE1 },
  { KEYSYM_CH7,   0x50, 0xAF, 0x1F, 0xE0 },
  { KEYSYM_CH8,   0x50, 0xAF, 0x04, 0xFB },
  { KEYSYM_CH9,   0x50, 0xAF, 0x05, 0xFA },
  { KEYSYM_CH10,  0x50, 0xAF, 0x06, 0xF9 },
  { KEYSYM_CH11,  0x50, 0xAF, 0x07, 0xF8 },
  { KEYSYM_CH12,  0x50, 0xAF, 0x0C, 0xF3 },
  { KEYSYM_CHUP,  0x50, 0xAF, 0x19, 0xE6 },
  { KEYSYM_CHDN,  0x50, 0xAF, 0x18, 0xE7 },
  { KEYSYM_VOLUP, 0x50, 0xAF, 0x12, 0xED },
  { KEYSYM_VOLDN, 0x50, 0xAF, 0x15, 0xEA },
  { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
};
const struct keysyms_s keysyms_toshiba_1[] = {
  /* 06101, 06102 | 208, 209, 210, 275 */
  { KEYSYM_PWR,   0x40, 0xBF, 0x12, 0xED },
  { KEYSYM_CH1,   0x40, 0xBF, 0x01, 0xFE },
  { KEYSYM_CH2,   0x40, 0xBF, 0x02, 0xFD },
  { KEYSYM_CH3,   0x40, 0xBF, 0x03, 0xFC },
  { KEYSYM_CH4,   0x40, 0xBF, 0x04, 0xFB },
  { KEYSYM_CH5,   0x40, 0xBF, 0x05, 0xFA },
  { KEYSYM_CH6,   0x40, 0xBF, 0x06, 0xF9 },
  { KEYSYM_CH7,   0x40, 0xBF, 0x07, 0xF8 },
  { KEYSYM_CH8,   0x40, 0xBF, 0x08, 0xF7 },
```

```
    { KEYSYM_CH9,   0x40, 0xBF, 0x06, 0xF6 },
    { KEYSYM_CH10,  0x40, 0xBF, 0x0A, 0xF5 },
    { KEYSYM_CH11,  0x40, 0xBF, 0x0B, 0xF4 },
    { KEYSYM_CH12,  0x40, 0xBF, 0x0C, 0xF3 },
    { KEYSYM_CHUP,  0x40, 0xBF, 0x1B, 0xE4 },
    { KEYSYM_CHDN,  0x40, 0xBF, 0x1F, 0xE0 },
    { KEYSYM_VOLUP, 0x40, 0xBF, 0x1A, 0xE5 },
    { KEYSYM_VOLDN, 0x40, 0xBF, 0x1E, 0xE1 },
    { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
  };
  const struct keysyms_s keysyms_toshiba_2[] = {
    /* 06103 | 207, 211 */
    { KEYSYM_PWR,   0x40, 0xBF, 0x12, 0xED },
    { KEYSYM_CH1,   0x40, 0xBF, 0x61, 0x9E },
    { KEYSYM_CH2,   0x40, 0xBF, 0x62, 0x9D },
    { KEYSYM_CH3,   0x40, 0xBF, 0x63, 0x9C },
    { KEYSYM_CH4,   0x40, 0xBF, 0x64, 0x9B },
    { KEYSYM_CH5,   0x40, 0xBF, 0x65, 0x9A },
    { KEYSYM_CH6,   0x40, 0xBF, 0x66, 0x99 },
    { KEYSYM_CH7,   0x40, 0xBF, 0x67, 0x98 },
    { KEYSYM_CH8,   0x40, 0xBF, 0x68, 0x97 },
    { KEYSYM_CH9,   0x40, 0xBF, 0x66, 0x96 },
    { KEYSYM_CH10,  0x40, 0xBF, 0x6A, 0x95 },
    { KEYSYM_CH11,  0x40, 0xBF, 0x6B, 0x94 },
    { KEYSYM_CH12,  0x40, 0xBF, 0x6C, 0x93 },
    { KEYSYM_CHUP,  0x40, 0xBF, 0x1B, 0xE4 },
    { KEYSYM_CHDN,  0x40, 0xBF, 0x1F, 0xE0 },
    { KEYSYM_VOLUP, 0x40, 0xBF, 0x1A, 0xE5 },
    { KEYSYM_VOLDN, 0x40, 0xBF, 0x1E, 0xE1 },
    { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
  };
  const struct keysyms_s keysyms_sanyo_1[] = {
    /* 02101 | 200, 202 */
    { KEYSYM_PWR,   0x30, 0xCF, 0x00, 0xFF },
    { KEYSYM_CH1,   0x30, 0xCF, 0x10, 0xEF },
    { KEYSYM_CH2,   0x30, 0xCF, 0x11, 0xEE },
    { KEYSYM_CH3,   0x30, 0xCF, 0x12, 0xED },
    { KEYSYM_CH4,   0x30, 0xCF, 0x13, 0xEC },
    { KEYSYM_CH5,   0x30, 0xCF, 0x14, 0xEB },
    { KEYSYM_CH6,   0x30, 0xCF, 0x15, 0xEA },
    { KEYSYM_CH7,   0x30, 0xCF, 0x16, 0xE9 },
    { KEYSYM_CH8,   0x30, 0xCF, 0x17, 0xE8 },
    { KEYSYM_CH9,   0x30, 0xCF, 0x18, 0xE7 },
    { KEYSYM_CH10,  0x30, 0xCF, 0x19, 0xE6 },
    { KEYSYM_CH11,  0x30, 0xCF, 0x1A, 0xE5 },
    { KEYSYM_CH12,  0x30, 0xCF, 0x1B, 0xE4 },
    { KEYSYM_CHUP,  0x30, 0xCF, 0x59, 0xA6 },
    { KEYSYM_CHDN,  0x30, 0xCF, 0x5A, 0xA5 },
    { KEYSYM_VOLUP, 0x30, 0xCF, 0x09, 0xF6 },
    { KEYSYM_VOLDN, 0x30, 0xCF, 0x0A, 0xF5 },
    { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
  };
  const struct keysyms_s keysyms_funai_dx_1[] = {
    /* --- | 229, 256 */
    { KEYSYM_PWR,   0x84, 0xE0, 0x20, 0xDF },
    { KEYSYM_CH1,   0x84, 0xE0, 0x01, 0xFE },
    { KEYSYM_CH2,   0x84, 0xE0, 0x02, 0xFD },
    { KEYSYM_CH3,   0x84, 0xE0, 0x03, 0xFC },
    { KEYSYM_CH4,   0x84, 0xE0, 0x04, 0xFB },
    { KEYSYM_CH5,   0x84, 0xE0, 0x05, 0xFA },
    { KEYSYM_CH6,   0x84, 0xE0, 0x06, 0xF9 },
    { KEYSYM_CH7,   0x84, 0xE0, 0x07, 0xF8 },
    { KEYSYM_CH8,   0x84, 0xE0, 0x08, 0xF7 },
    { KEYSYM_CH9,   0x84, 0xE0, 0x09, 0xF6 },
    { KEYSYM_CH10,  0x84, 0xE0, 0x0A, 0xF5 },
    { KEYSYM_CH11,  0x84, 0xE0, 0x0B, 0xF4 },
    { KEYSYM_CH12,  0x84, 0xE0, 0x0C, 0xF3 },
    { KEYSYM_CHUP,  0x84, 0xE0, 0x50, 0xAF },
    { KEYSYM_CHDN,  0x84, 0xE0, 0x51, 0xAE },
    { KEYSYM_VOLUP, 0x84, 0xE0, 0x60, 0x9F },
    { KEYSYM_VOLDN, 0x84, 0xE0, 0x61, 0x9E },
    { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
  };
```

```
 const struct keysyms_s keysyms_nanao_1[] = {
   /* --- | 242, 243 */
   { KEYSYM_PWR,   0x84, 0x26, 0x18, 0xE7 },
   { KEYSYM_CH1,   0x84, 0x26, 0x00, 0xFF },
   { KEYSYM_CH2,   0x84, 0x26, 0x01, 0xFE },
   { KEYSYM_CH3,   0x84, 0x26, 0x02, 0xFD },
   { KEYSYM_CH4,   0x84, 0x26, 0x03, 0xFC },
   { KEYSYM_CH5,   0x84, 0x26, 0x04, 0xFB },
   { KEYSYM_CH6,   0x84, 0x26, 0x05, 0xFA },
   { KEYSYM_CH7,   0x84, 0x26, 0x06, 0xF9 },
   { KEYSYM_CH8,   0x84, 0x26, 0x07, 0xF8 },
   { KEYSYM_CH9,   0x84, 0x26, 0x08, 0xF7 },
   { KEYSYM_CH10,  0x84, 0x26, 0x1D, 0xE2 },
   { KEYSYM_CH11,  0x84, 0x26, 0x1E, 0xE1 },
   { KEYSYM_CH12,  0x84, 0x26, 0x1F, 0xE0 },
   { KEYSYM_CHUP,  0x84, 0x26, 0x1B, 0xE4 },
   { KEYSYM_CHDN,  0x84, 0x26, 0x13, 0xEC },
   { KEYSYM_VOLUP, 0x84, 0x26, 0x0F, 0xF0 },
   { KEYSYM_VOLDN, 0x84, 0x26, 0x0B, 0xF4 },
   { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
 };
 const struct keysyms_s keysyms_lg_1[] = {
   /* 15101 | 228, 294, 295 */
   { KEYSYM_PWR,   0x04, 0xFB, 0x08, 0xF7 },
   { KEYSYM_CH1,   0x04, 0xFB, 0xDB, 0x31 },
   { KEYSYM_CH2,   0x04, 0xFB, 0xDB, 0x32 },
   { KEYSYM_CH3,   0x04, 0xFB, 0xDB, 0x33 },
   { KEYSYM_CH4,   0x04, 0xFB, 0xDB, 0x34 },
   { KEYSYM_CH5,   0x04, 0xFB, 0xDB, 0x35 },
   { KEYSYM_CH6,   0x04, 0xFB, 0xDB, 0x36 },
   { KEYSYM_CH7,   0x04, 0xFB, 0xDB, 0x37 },
   { KEYSYM_CH8,   0x04, 0xFB, 0xDB, 0x38 },
   { KEYSYM_CH9,   0x04, 0xFB, 0xDB, 0x39 },
   { KEYSYM_CH10,  0x04, 0xFB, 0xDB, 0x3A },
   { KEYSYM_CH11,  0x04, 0xFB, 0xDB, 0x3B },
   { KEYSYM_CH12,  0x04, 0xFB, 0xDB, 0x3C },
   { KEYSYM_CHUP,  0x04, 0xFB, 0x00, 0xFF },
   { KEYSYM_CHDN,  0x04, 0xFB, 0x01, 0xFE },
   { KEYSYM_VOLUP, 0x04, 0xFB, 0x02, 0xFD },
   { KEYSYM_VOLDN, 0x04, 0xFB, 0x03, 0xFC },
   { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
 };
 const struct keysyms_s keysyms_lg_2[] = {
   /* --- | 296 */
   { KEYSYM_PWR,   0x04, 0xFB, 0x08, 0xF7 },
   { KEYSYM_CH1,   0x04, 0xFB, 0x11, 0xEE },
   { KEYSYM_CH2,   0x04, 0xFB, 0x12, 0xED },
   { KEYSYM_CH3,   0x04, 0xFB, 0x13, 0xEC },
   { KEYSYM_CH4,   0x04, 0xFB, 0x14, 0xEB },
   { KEYSYM_CH5,   0x04, 0xFB, 0x15, 0xEA },
   { KEYSYM_CH6,   0x04, 0xFB, 0x16, 0xE9 },
   { KEYSYM_CH7,   0x04, 0xFB, 0x17, 0xE8 },
   { KEYSYM_CH8,   0x04, 0xFB, 0x18, 0xE7 },
   { KEYSYM_CH9,   0x04, 0xFB, 0x19, 0xE6 },
   { KEYSYM_CH10,  0x04, 0xFB, 0x10, 0xEF },
   { KEYSYM_CH11,  0x04, 0xFB, 0x1C, 0xE3 },
   { KEYSYM_CH12,  0x04, 0xFB, 0x1D, 0xE2 },
   { KEYSYM_CHUP,  0x04, 0xFB, 0x00, 0xFF },
   { KEYSYM_CHDN,  0x04, 0xFB, 0x01, 0xFE },
   { KEYSYM_VOLUP, 0x04, 0xFB, 0x02, 0xFD },
   { KEYSYM_VOLDN, 0x04, 0xFB, 0x03, 0xFC },
   { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
 };
 const struct keysyms_s keysyms_bydesign_1[] = {
   /* --- | 231, 264 */
   { KEYSYM_PWR,   0xF1, 0x0E, 0x08, 0xF7 },
   { KEYSYM_CH1,   0xF1, 0x0E, 0x11, 0xEE },
   { KEYSYM_CH2,   0xF1, 0x0E, 0x12, 0xED },
   { KEYSYM_CH3,   0xF1, 0x0E, 0x13, 0xEC },
   { KEYSYM_CH4,   0xF1, 0x0E, 0x14, 0xEB },
   { KEYSYM_CH5,   0xF1, 0x0E, 0x15, 0xEA },
   { KEYSYM_CH6,   0xF1, 0x0E, 0x16, 0xE9 },
   { KEYSYM_CH7,   0xF1, 0x0E, 0x17, 0xE8 },
```

```
256     { KEYSYM_CH8,   0xF1, 0x0E, 0x18, 0xE7 },
257     { KEYSYM_CH9,   0xF1, 0x0E, 0x19, 0xE6 },
258     { KEYSYM_CH10,  0xF1, 0x0E, 0x10, 0xEF },
259     { KEYSYM_CH11,  0xF1, 0x0E, 0xFE, 0x01 },
260     { KEYSYM_CH12,  0xF1, 0x0E, 0x39, 0xC6 },
261     { KEYSYM_CHUP,  0xF1, 0x0E, 0x00, 0xFF },
262     { KEYSYM_CHDN,  0xF1, 0x0E, 0x01, 0xFE },
263     { KEYSYM_VOLUP, 0xF1, 0x0E, 0x02, 0xFD },
264     { KEYSYM_VOLDN, 0xF1, 0x0E, 0x03, 0xFC },
265     { KEYSYM_NULL,  0x00, 0x00, 0x00, 0x00 },
266   };
267
268   /* リモコン仕様 */
269   const struct rcspec_s  rcspec_null = {
270     "UNDEFINED",      RC_TYPE_UNDEF,      keysyms_null };;
271   const struct rcspec_s  rcspec_any = {
272     "(ANY FORMAT)",   RC_TYPE_ANY,        keysyms_null };;
273   const struct rcspec_s  rcspec_hitachi_1 = {
274     "HITACHI(1)",     RC_TYPE_HITACHI_1,  keysyms_hitachi_1 };;
275   const struct rcspec_s  rcspec_hitachi_2 = {
276     "HITACHI(2)",     RC_TYPE_HITACHI_2,  keysyms_hitachi_2 };;
277   const struct rcspec_s  rcspec_hitachi_3 = {
278     "HITACHI(3)",     RC_TYPE_HITACHI_3,  keysyms_hitachi_3 };;
279   const struct rcspec_s  rcspec_hitachi_4 = {
280     "HITACHI(4)",     RC_TYPE_HITACHI_4,  keysyms_hitachi_4 };;
281   const struct rcspec_s  rcspec_toshiba_1 = {
282     "TOSHIBA(1)",     RC_TYPE_TOSHIBA_1,  keysyms_toshiba_1 };;
283   const struct rcspec_s  rcspec_toshiba_2 = {
284     "TOSHIBA(2)",     RC_TYPE_TOSHIBA_2,  keysyms_toshiba_2 };;
285   const struct rcspec_s  rcspec_sanyo_1 = {
286     "SANYO(1)",       RC_TYPE_SANYO_1,    keysyms_sanyo_1 };;
287   const struct rcspec_s  rcspec_funai_dx_1 = {
288     "FUNAI/DX",       RC_TYPE_FUNAI_DX_1, keysyms_funai_dx_1 };;
289   const struct rcspec_s  rcspec_nanao_1 = {
290     "NANAO",          RC_TYPE_NANAO_1,    keysyms_nanao_1 };;
291   const struct rcspec_s  rcspec_lg_1 = {
292     "LG(1)",          RC_TYPE_LG_1,       keysyms_lg_1 };;
293   const struct rcspec_s  rcspec_lg_2 = {
294     "LG(2)",          RC_TYPE_LG_2,       keysyms_lg_2 };;
295   const struct rcspec_s  rcspec_bydesign_1 = {
296     "BYD:SIGN",       RC_TYPE_BYDESIGN_1, keysyms_bydesign_1 };;
297
298   /* リモコン仕様表 */
299   const struct rcspec_s *rcspec[RCSPEC_MAX_N_ITEMS] = {
300     &rcspec_null, /* ダミーデータ: この行を消去しないこと */
301     &rcspec_any,
302     &rcspec_hitachi_1,
303     &rcspec_hitachi_2,
304     &rcspec_hitachi_3,
305     &rcspec_hitachi_4,
306     &rcspec_toshiba_1,
307     &rcspec_toshiba_2,
308     &rcspec_sanyo_1,
309     &rcspec_funai_dx_1,
310     &rcspec_nanao_1,
311     &rcspec_lg_1,
312     &rcspec_lg_2,
313     &rcspec_bydesign_1,
314     &rcspec_null, /* 終了印データ: この行を消去しないこと */
315   };
```

ソースファイル auto-sel.c (テレビリモコン自動選択機能: 実装)

テレビリモコン自動選択機能をこのソースファイルで実装します. 自動選択の動作原理は以下の通りの単純なものです.

1. テレビリモコンからフレームの受信を待機します.
2. リモコン仕様表 rcspec[] を参照して, 受信した 4 バイトのフレームが登録されているテレビリモコン

が何種類あるかを数えます.

3. 該当するテレビリモコンが 1 種類だけならば, そのテレビリモコンしか候補はありませんので特定は完了です. そうでなければ上記の手順を繰り返します.

ソースファイル番号 22.14: auto-sel.c

```
/* リモコン型の自動選択(NECフォーマット) */
/* 押されるリモコンボタンから自動判別 */

#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "ir-frame.h"
#include "keysym.h"
#include "rc.h"
#include "led-ir-mon.h"

void auto_select(void)
{
  uint8_t ncands = 0;
  rcs_id_t r_found = RCS_ID_NULL;
  rcs_id_t r       = RCS_ID_NULL;
  /* 候補絞り込み */
  uint16_t t = 0;
  do {
    CLRWDT();
    __delay_ms(1);
    /* 候補絞り込み中はLEDを中速フラッシュ */
    if (t == 0) {
      if (LED_IR_MON_IS_ON()) {
        LED_IR_MON_OFF();
        t = 250;
      } else {
        LED_IR_MON_ON();
        t = 25;
      }
    }
    t--;
    /* フレーム受信 */
    if (ir_rcv_flag != 0) {
      /* 該当候補数を調べる */
      ncands = 0; /* 該当候補数 */
      for (r = 1;
           (r < RCSPEC_MAX_N_ITEMS)
             && (rcspec[r]->type != RC_TYPE_UNDEF);
           r++) {
        const struct keysyms_s *kst = rcspec[r]->keysyms;
        uint16_t k = 0;
        for (k = 0; kst[k].keysym != KEYSYM_NULL; k++) {
          if ((kst[k].b0 == ir_data_buff[0]) &&
              (kst[k].b1 == ir_data_buff[1]) &&
              (kst[k].b2 == ir_data_buff[2]) &&
              (kst[k].b3 == ir_data_buff[3])) { /* 発見 */
            ncands++;    /* 該当候補数 */
            r_found = r; /* 検索結果 */
            break;
          }
        }
      }
    }
    /* フレーム受信完了 */
    ir_rcv_flag = 0;
  } while (ncands != 1);
  /* 設定完了 */
  LED_IR_MON_ON();
  /* 検索結果 */
  rcs_id = r_found;
```

```
63  }
```

ソースファイル if.h (ポイント制御: ヘッダ)

ポイント制御を実現する関数群のプロトタイプ宣言と関係する定数を定義します. 多数のポイントが同時に切り替えられることがないように, 複数のポイントの切り替えは 100 ミリ秒の時間差で順次行います.

ソースファイル番号 22.15: if.h

```
1   #if !defined(__IF_H__)
2   #define __IF_H__
3
4   /* ポイント番号 */
5   typedef uint8_t if_to_id;
6   #define IF_TO_NULL   0
7
8   #endif /*__IF_H__*/
```

ソースファイル if-toq.h (ポイント切替動作キュー: ヘッダ)

ポイント切替動作キューに関連する関数のプロトタイプ宣言を行います. 本章ではこの機能は実装しないので読み飛ばして下さい. 第 23 章でポイント切替制御機能を実現する際に使用しますので, このソースファイルはその時に読んで下さい.

ソースファイル番号 22.16: if-toq.h

```
1   #if !defined(__IF_TOQ_H__)
2   #define __IF_TOQ_H__
3
4   extern void   if_toq_init(void);
5   extern void   if_toq_set_normal(if_to_id p);
6   extern void   if_toq_set_reverse(if_to_id p);
7   extern void   if_toq_set_all_normal(void);
8   extern void   if_toq_set_all_reverse(void);
9   extern int8_t if_toq_is_done(void);
10  extern int8_t if_toq_next(uint16_t tcyc);
11
12  #endif /*__IF_TOQ_H__*/
```

ソースファイル if-toq-dummy.c (ポイント切替動作キュー: ダミー実装)

ポイント切替動作キューに関連する関数のダミー実装です. 第 23 章では実際にこれらの関数を実装してポイント切替制御機能を実現します.

ソースファイル番号 22.17: if-toq-dummy.c

```
1   /*-----------------------------------------------------
2    * ダミー関数群
3    *-----------------------------------------------------*/
4   #include <xc.h>
5   #include <stdint.h>
6   #include "mcr.h"
7   #include "io.h"
8   #include "if.h"
9   #include "if-toq.h"
10
11  void if_toq_init(void)
12  {
13    ;
14  }
15
16  void if_toq_set_normal(if_to_id p)
17  {
```

```
  ;
}

void if_toq_set_reverse(if_to_id p)
{
  ;
}

void if_toq_set_all_normal(void)
{
  ;
}

void if_toq_set_all_reverse(void)
{
  ;
}

int8_t if_toq_is_done(void)
{
  return 1;
}

int8_t if_toq_next(uint16_t tcyc)
{
  return 0;
}
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 22.18: mcr.h

```
#if !defined(__MCR_H__)
#define __MCR_H__

#define _XTAL_FREQ 32000000 /* 32MHz */

extern void System_Init(void);
extern void TMR3_Init(void);

#endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います.

ソースファイル番号 22.19: mcr.c

```
/*-------------------------------------------------------
 * MCR設定&周辺モジュール (PIC16F18857)
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "mcr-motor.h"
#include "ir-frame.h"
#include "adc.h"
#include "com.h"
#include "pp-sb.h"

/*-----------------------------------------*/
/* MCR初期化 */
void System_Init(void)
{
  /* 発振器 (FOSC=32MHz) */
  OSCCON1 = 0x10; /* HFINTOSC with 2x PLL */
```

```
  OSCCON3 = 0x00;
  OSCEN   = 0x00;
  OSCFRQ  = 0x05; /* 16MHz */
  OSCTUNE = 0x00;
  while (PLLR == 0) {
    ;
  }
  /* I/Oピン */
  LATA   = 0x00;
  LATB   = 0x00;
  LATC   = 0x00;
  ANSELA = 0xBF; /* ANA7,ANA5,ANA4,ANA3,ANA2,ANA1,ANA0 */
  ANSELB = 0x00; /* - */
  ANSELC = 0x00; /* - */
  TRISA  = 0xFF; /* RA7,RA6,RA5,RA4,RA3,RA2,RA1,RA0 */
  TRISB  = 0xDF; /* (RB7,RB6),RB4,(RB3),RB2,RB1,RB0 */
  TRISC  = 0x87; /* (RC7),RC2,RC1,RC0 */
  IOCBP  = 0x01; /* RB0 */
  IOCBN  = 0x01; /* RB0 */
  IOCAF  = 0x00;
  WPUA   = 0x00;
  WPUB   = 0xC8; /* RB7,RB6,RB3 */
  WPUC   = 0x00;
  WPUE   = 0x00;
  ODCONA = 0x00;
  ODCONB = 0x00;
  ODCONC = 0x00;
  int8_t ogie = GIE;
  GIE = 0;
  PPSLOCK = 0x55;
  PPSLOCK = 0xAA;
  PPSLOCKbits.PPSLOCKED = 0;
  RC5PPS = 0x05;     /* RC5 <- CWG1A */
  RC6PPS = 0x06;     /* RC6 <- CWG1B */
  RB1PPS = 0x14;     /* RB1 <- SCL1 */
  RB2PPS = 0x15;     /* RB2 <- SDA1 */
  SSP1CLKPPS = 0x09; /* RB1 -> SCL1 */
  SSP1DATPPS = 0x0A; /* RB2 -> SDA1 */
  PPSLOCK = 0x55;
  PPSLOCK = 0xAA;
  PPSLOCKbits.PPSLOCKED = 1;
  GIE = ogie;
  /* 周辺モジュール */
  TMR3_Init();
  ADC_Init();
  MCR_Motor_Init();
  IOCIE = 1;
}

/*-----------------------------------------*/
/* TMR3 */
/* 計時タイマとして使用 */
/* Src: FOSC/4, Prescale 1:2, タイマー周期: 16.384mSec */

void TMR3_Init(void)
{
  T3CONbits.TMR3ON = 0;
  T3CONbits.CKPS = 0x01; /* Prescale: 1:2 */
  T3CLKbits.CS   = 0x01; /* FOSC/4 */
  T3GCON = 0x00;
  TMR3H = 0;
  TMR3L = 0;
  TMR3IF = 0;
  TMR3IE = 1;
  T3CONbits.TMR3ON = 1;
}

uint16_t TMR3_Read(void)
{
  T3CONbits.TMR3ON = 0;
  uint16_t t = ((uint16_t)TMR3H << 8) | (uint16_t)TMR3L;
  TMR3H = 0;
```

```
91    TMR3L = 0;
92    T3CONbits.TMR3ON = 1;
93    return t;
94  }
95
96  void TMR3_isr(void)
97  {
98    T3CONbits.TMR3ON = 0;
99    TMR3H = 0xFF;
100   TMR3L = 0xFF;
101 }
102
103 /*-------------------------------------------*/
104 /* 割込ハンドラ */
105 __interrupt() void isr(void)
106 {
107   /* TMR1: 低速ブースト機能用 (pp-sb.c参照) */
108   if ((TMR1IE == 1) && (TMR1IF == 1)) {
109     TMR1IF = 0;
110     TMR1_isr();
111   }
112   /* TMR2: PWM周波数発生用 (割り込みハンドラなし/mcr-motor.c参照) */
113   /* TMR3: テレビリモコンのパケットデコーダ用 */
114   if ((TMR3IE == 1) && (TMR3IF == 1)) {
115     TMR3IF = 0;
116     TMR3_isr();
117     ir_frame_isr(2, 0xFFFF); /* デコーダ状態遷移: リセット */
118   }
119   /* IOC RB0 */
120   if ((IOCIE == 1) && (IOCIF == 1)) {
121     IOCIF = 0;
122     if ((IOCBF & 0x01) != 0) {
123       IOCBF &= ~0x01;
124       if (RB0 == 1) {
125         ir_frame_isr(1, TMR3_Read()); /* デコーダ状態遷移: オン期間終了 */
126       } else {
127         ir_frame_isr(0, TMR3_Read()); /* デコーダ状態遷移: オフ期間終了 */
128       }
129     }
130   }
131 }
132
133 /*-------------------------------------------*/
134 /* CONFIG1 */
135 #pragma config FEXTOSC  = OFF
136 #pragma config RSTOSC   = HFINTPLL
137 #pragma config CLKOUTEN = OFF
138 #pragma config CSWEN    = ON
139 #pragma config FCMEN    = ON
140 /* CONFIG2 */
141 #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
142 #pragma config PWRTE    = OFF
143 #pragma config LPBOREN  = OFF
144 #pragma config BOREN    = ON
145 #pragma config BORV     = LO
146 #pragma config ZCD      = OFF
147 #pragma config PPS1WAY  = ON
148 #pragma config STVREN   = ON
149 #pragma config DEBUG    = OFF
150 /* CONFIG3 */
151 #pragma config WDTCPS   = WDTCPS_31
152 #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
153 #pragma config WDTCWS   = WDTCWS_7
154 #pragma config WDTCCS   = SC
155 /* CONFIG4 */
156 #pragma config WRT      = OFF
157 #pragma config SCANE    = available
158 #pragma config LVP      = OFF /* 低電圧書込不可 */
159 /* CONFIG5 */
160 #pragma config CP       = OFF
161 #pragma config CPD      = OFF
```

課題

1. NEC フォーマットにおけるリピートコードについて調査し, 説明してください.
2. 家電製品協会 (AEHA) フォーマットを調査し, 説明してください.
3. SONY フォーマットを調査し, 説明してください.
4. 本装置で使用した OptoSupply 社 OSRB38C9AA の代替として使用可能な赤外線リモコン受信モジュールをいくつか探して下さい.
5. 本章で紹介した赤外線リモコンのフレームデコーダの設計技法を応用して DCC デコーダのパケット受信部分を設計してください.

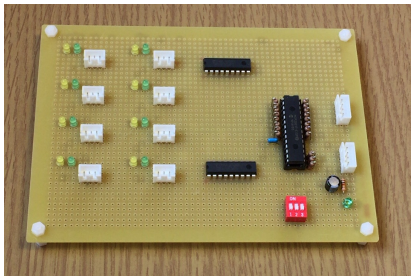
第 23 章

ポイント制御機能
テレビリモコン式パワーパック

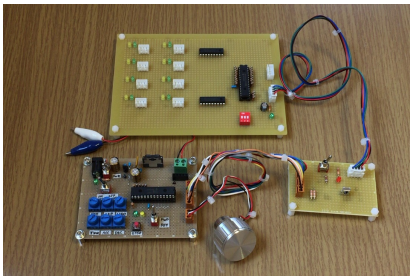
本章ではポイント制御機能付きの赤外線テレビリモコンで遠隔制御可能なパワーパックを紹介します．ポイント制御機能は図 23.1a に示すポイントインターフェースボードを新たに製作して，第 22 章で紹介した拡張ボードに接続します．そしてテレビリモコンによるポイント切り替え機能はメインボード上のマイコンのファームウエアを差し替えて実現します．本章で紹介する回路とファームウエアの特徴は以下の通りで，第 22 章で紹介した機能すべてそのまま引き継いでいます．

- リモコンによるポイント切替え機能
- 最大 8 つのポイントを制御
- ポイント制御入出力による各種ポイント切替器が利用可能

本章で学ぶ内容は以下の通りです．



(a) ポイントインターフェースボード



(b) パワーパック本体との接続

図 23.1: ポイント制御機能付テレビリモコン式パワーパックの試作例

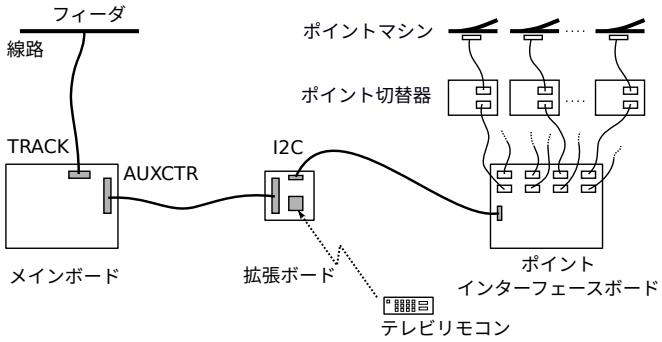


図 23.2: 接続方法

- I^2C の概要
- PIC マイコンの同期シリアル通信モジュール MSSP
- PIC マイコン用 I^2C 通信ファームウエア
- I^2C I/O エクスパンダ IC MCP23017 の使用方法

23.1　使用方法

本装置で使用するものは以下の通りです.

- パワーパック本体 (第 9 章)
- 拡張ボード (第 22 章)
- 赤外線テレビリモコン (第 22 章を参照)
- ポイントインタフェースボード (本章)
  ポイントインターフェースボードは拡張ボードと I^2C バスを通じて接続して 8 つのポイント切替器を制御します. このポイントインターフェースボードは第 28 章で紹介するポイント一括切替器でも使用します. ハードウエア的には最大 8 枚まで接続可能ですが本章では 1 枚だけを使用します.
- ポイント切替器 (第 12 章と第 14 章)
  ポイントインタフェースボードからのポイント制御信号を入力とし, ポイントマシンの制御を行います.

操作手順は以下の通りです.

1. 電源スイッチを切った状態でパワーパック本体の DC ジャックに 12V の AC アダプタを接続します.
2. 図 23.2 に示すように接続をして下さい.
   - メインボード TRACK ↔ 線路のフィーダ
   - メインボード AUXCTR ↔ 拡張ボード AUXCTR
   - 拡張ボード I2C ↔ ポイントインターフェースボード I2C
   - ポイントインターフェースボードのポイント制御出力 ↔ ポイント切替器のポイント制御入力
   - ポイント切替器の出力 ↔ ポイントマシン
3. ポイントインターフェースボードの DIP スイッチ 1~3 をすべてオフにします.
4. 拡張ボードのスイッチ MODE はリモコン制御モード側に切り替えます. なお拡張ボードのスイッチ MODE を手動制御モード側に切り替えて手動制御も可能です. スイッチ MODE を切り替えた場合には電源は入れ直して下さい. なお手動制御モードではポイントの切り替えはできません.
5. 電源スイッチを入れます. すると LED はフラッシュ点滅して, 第 22 章の場合と同様にテレビリモコンの自動選択が行われます. LED のフラッシュ点滅が止まり, 点灯が続けば選択は完了です.
6. 模型車両の走行操作の準備は終わりました. テレビリモコンを使用して模型車両を走行させましょう. テレビリモコンのボタン機能一覧は表 22.1 (347 ページ) で示したとおりです.

23.2　回路図

ポイントインターフェースボードの回路図を図 23.3 に, 部品リストを表 23.1 に, それぞれ示します. ポイントインターフェースボードの電源は I^2C バスケーブルを通じてメインボードより供給を受けますので, ポイントインターフェースボード自体には DC ジャックはありません. 中心となる部品が I^2C I/O エクスパンダ IC MCP23017 で, I^2C バスを通じて 16 本の I/O ポートを操作する機能を持ちます. I^2C および MCP23017 については後で詳しく説明します. TBD62083APG は多数のトランジスタを 1 チップに収めたトランジスタアレイと

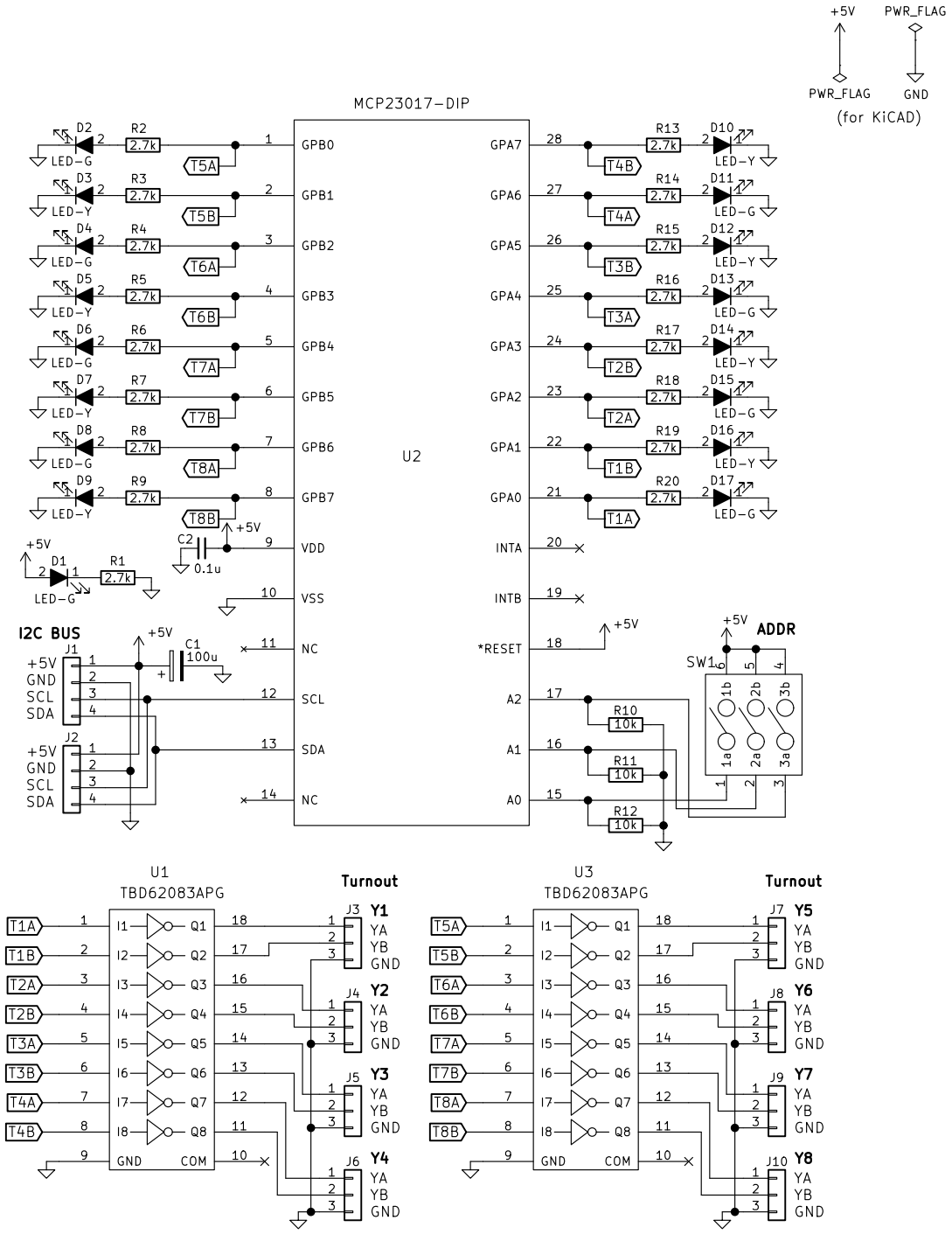


図 23.3: 【回路図】 ポイントインターフェースボード

呼ばれる IC で, ポイント制御出力おいて十分な電流量を確保するために使用します. また TBD62083APG の出力はオープンドレインである点も注意して下さい. ポイントの切り替え状況はすべて LED の点灯で表示されますからひと目で状況の確認が可能です.

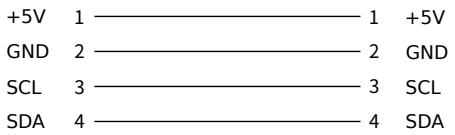


図 23.4: I^{2}C バスケーブル

表 23.1: 【部品表】 ポイントインターフェースボード

記号	数量	部品名
U2	1	MCP23017-E/SP　I/O エキスパンダ [I-09486]
U1, U3	2	TBD62083APG　トランジスタアレイ [I-10669]
D1, D2, D4, D6, D8, D11, D13, D15, D17	9	LED 緑 [I-06413]
D3, D5, D7, D9, D10, D12, D14, D16	8	LED 黄 [I-11639]
C1	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C2	1	セラミックコンデンサ 0.1μF [P-00090]
R1, R2, R3, R4, R5, R6, R7, R8, R9, R13, R14, R15, R16, R17, R18, R19, R20	17	抵抗 2.7kΩ　1/4W [R-25272]
R10, R11, R12	3	抵抗 10kΩ　1/4W [R-25103]
SW1	1	DIP SW 3P [P-09992]
	1	IC ソケット 28 ピン [P-01339]
J1, J2	2	基板用コネクタ 4 ピン　JST XH コネクタ B4B-XH-A(LF) [C-12249]
J3, J4, J5, J6, J7, J8, J9, J10	8	基板用コネクタ 3 ピン　JST XH コネクタ B3B-XH-A(LF) [C-12248]
	2	コネクタハウジング 4 ピン　JST XH コネクタ XHP-4 [C-12257]
	8	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
		ほか基板・配線材等

拡張ボードとポイントインターフェースボードの接続のため, 図 23.4 に示す I^{2}C バスケーブルも作成して下さい.

23.3　同期 2 線式シリアル通信 I^{2}C

I^{2}C 通信とは, クロック信号線 SCL とデータ信号線 SDA の 2 線による双方向の同期シリアル通信で, 100kbps から 1Mbps 程度の通信速度を持ち, バス型通信により 2 つ以上のデバイスの接続が可能です [11]. 基板内あるいは基板間での通信を想定した規格のためノイズに弱く, 通信可能な距離は長くても数メートル程度までです. ここでは電気的な観点ではなく論理的な観点から I^{2}C 通信の概要を説明します.

I^{2}C バスに接続されるデバイスは以下の 2 種に大別されます.

- マスタ
  データ転送と開始と終了を指示します. 通常はマイクロプロセッサがマスタです. データ転送時にはクロック信号を出力します. ひとつの I^{2}C バスには少なくともひとつのマスタが存在します. なお本書で

はマスタ数は 1 つの場合のみを説明します.

- スレーブ
  マスタからの指示を受けて動作します. 通常は EEPROM やセンサなどのデバイスがスレーブです. I^{2}C バスには 1 つあるいはそれ以上の数のスレーブが存在します. 各スレーブには固有のアドレスが割り振られ, マスタはアドレスを指定することで通信相手のスレーブを指定します.

I^{2}C 通信は双方向であるため, ひとつの I^{2}C バス上で送信側と受信側は時々刻々と変化します.

- トランスミッタ
  I^{2}C バスにデータを送信するデバイスです.
- レシーバ
  I^{2}C バスのデータを受信するデバイスです.

I^{2}C バスは以下の 2 線で構成されます.

- クロック信号線 SCL
  I^{2}C バスでの通信の同期信号を伝える信号線です. マスタが SCL へ信号を出力します.
- データ信号線 SDA
  I^{2}C バスでのデータ信号および制御信号を伝える双方向の信号線です. マスタとスレーブのどちらもが SDA を共用して信号を出力しますから, 複数のデバイスが同時に SDA へ信号を出力しないように送信手順を守る必要があります.

SCL と SDA の 2 つの線により双方向の通信を行う方法のアイディアを説明します.

- マスタからスレーブへの送信
  マスタがデータ線 SDA に値を設定するたびにクロック SCL を変化させて, データ線上にデータが送出されていることをスレーブに知らせます. そしてスレーブは SCL の信号の変化に合わせてデータを読み取ります.
- スレーブからマスタへの送信
  このときもマスタがクロック SCL を変化させます. スレーブは SCL の信号の変化に合わせてデータ線 SDA にデータを送出して, マスタはその信号を読み取ります.
- スレーブがデータを送るべきかどうかはマスタが最初に送出する制御バイト内の 1 ビットに情報が含まれていますし, マスタのみが制御バイトの送出をしますから, どちらが 1 つのデータ線上にデータを送出するかは曖昧さなく定まります.

I^{2}C 通信の基本的な用語を以下に説明します.

- スレーブアドレス
  スレーブアドレスとは各スレーブに割り当てられ固有の番号で, マスタが通信相手を指定する際に使用します. スレーブアドレスには 7 ビット方式と 10 ビット方式がありますが, 本書では 7 ビット方式のみを説明します.
- スタートコンディション (START) とストップコンディション (STOP)
  I^{2}C 通信ではマスタとスレーブ間での通信の最小単位であるトランザクションを単位として行われます. マスタは I^{2}C バスに START 信号を出力してトランザクションを開始し, STOP 信号を出力してトラン

ザクションを終了します. ひとつのトランザクションの実行中に他のトランザクションの実行はできません.

- リピートスタートコンディション (repeated START)
  トランザクションを終了し直ちに次のトランザクションを開始を開始する場合に使用する信号で, STOP 信号とそれに続く START 信号の代わりとするものです.
- バイト・フォーマット
  通信データのバイトはすべて 8 ビット単位です. ひとつのトランザクションで複数バイトを送信することができますが, 受信側は 1 バイト送信するごとに後述のアクノッリジあるいはノットアクノリッジを送信する必要があります.
- アクノリッジ (ACK) とノットアクノリッジ (NO ACK)
  レジーバによる返答です. これらはレシーバが送信するものですので, マスタおよびスレーブのどちらもが ACK および NO ACK を送信する場合があります. 正常に受信をした場合には ACK が返されますが, 例えば以下のような場合には NO ACK が返されます.
  - 指定したスレーブアドレスのデバイスが存在ない場合
  - レシーバがデータの受信を終了する場合
  - レシーバがデータを受信できない場合

I^2C バスでの通信は以下の考え方で行われます.

- トランザクションの開始と終了はマスタが行います
  マスタが START 信号を送信してトランザクションが開始され, マスタが STOP 信号を送信してトランザクションは終了します.
- データ送信デバイスはマスタが決定します
  トランザクションの第 1 バイトの送信は必ずマスタが行います. 第 1 バイトはスレーブアドレスと R/W ビットで構成されます.
  - スレーブアドレス
    通信対象のスレーブを指定します.
  - R/W ビット
    第 2 バイト以降の送信をマスタとスレーブのうちのどちらが行うか指定します. R/W ビットの値が 0 の場合はマスタが送信し, スレーブは受信をします. R/W ビットの値が 1 の場合はスレーブが送信し, マスタは受信をします.

  スレーブが第 1 バイトの内容に従うことで, 複数のデバイスが同時に I^2C バスへ出力をしない通信プロトコルとなっています.
- トランザクションの終了はレシーバ側でも決定可能です
  レシーバが NO ACK を返せばマスタはトランザクションを終了します.

23.4　MSSP: 同期シリアル通信モジュール

PIC マイコンの内蔵周辺モジュール MSSP (Master Synchronous Serial Port; マスタ同期シリアルポート) を使用して I^2C 通信を行います. MSSP は SPI 通信も行える内蔵周辺モジュールで, I^2C モードと SPI モードを設定により切り替えて使用します. 16F18857 にはふたつの MSSP モジュールが内蔵されており, それぞれ MSSP1, MSSP2 と名付けられています. MSSP1 の I^2C モードで使用する主なレジスタやビットを以下に概説します.

- SSP1ADD
  通信速度を設定する 8 ビットのレジスタです.
- SSP1BUF
  通信データを保持する 8 ビットのレジスタです. 送信と受信の双方でこのレジスタを使用します.
  - 送信
    SSP1BUF にデータを書き込むことで自動的に送信が開始されます. R_nW ビットが 0 になれば送信完了です.
  - 受信
    RCEN ビットで受信を指示し, 受信完了後まで待機します. そしてこのレジスタからデータを読み出します.
- SSP1STAT.R_nW
  I^2C バスへのデータの読み出しあるいは書き込みを指定するビットです. 1 の場合は読み出しで, 0 の場合は書き込みです.
- SSP1CON1.WCOL
  書き込み衝突の発生を検出するビットです. 1 の場合は衝突発生があり, 0 の場合はなしです.
- SSP1CON1.SSPEN
  MSSP のオン・オフを選択するビットです. 1 の場合はオンで, 0 の場合はオフです.
- SSP1CON1.SSPM
  動作モードを選択する 4 ビットです. I^2C または SSP, マスタまたはスレーブを決定します.
- SSP1CON2.ACKSTAT
  ACK 状態を表すビットです. 1 の場合は ACK の検出があり, 0 の場合はありません.
- SSP1CON2.ACKDT
  NO ACK/ACK を指定するビットで, 受信時に使用します. 1 を書き込むと NO ACK を, 0 を書き込むと ACK を送信します. ただし実際の送信の開始は ACKEN に 1 を書き込むことで行われます.
- SSP1CON2.ACKEN
  NO ACK/ACK の送信開始を指示するビットで, 送信内容は ACKDT の値に従います. 1 を書き込むと送信を開始します. このビットの値が 0 になると送信終了です.
- SSP1CON2.RCEN
  受信開始を指示するビットです. 1 を書き込むと受信を開始します. このビットの値が 0 になると受信終了です.
- SSP1CON2.PEN
  ストップコンディションの送信開始を指示するビットです. 1 を書き込むと送信を開始します. このビットの値が 0 になると送信終了です.
- SSP1CON2.RSEN
  リピートスタートコンディションの送信開始を指示するビットです. 1 を書き込むと送信を開始します. このビットの値が 0 になると送信終了です.
- SSP1CON2.SEN
  スタートコンディションの送信開始を指示するビットです. 1 を書き込むと送信を開始します. このビットの値が 0 になると送信終了です.

ボーレートの値を計算する式は Fosc/{4 × (SSP1ADD + 1)} です. 例えば 400kbps で通信をする際には

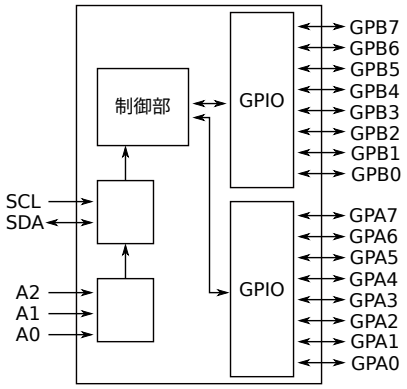


図 23.5: I/O エクスパンダ IC MCP23017 の内部構造の概略

SSP1ADD=0x13 と設定します. この計算は以下の通りです.

$$\begin{aligned} \text{bps} &= \text{Fosc}/\{4 \times (\texttt{SSP1ADD} + 1)\} \\ \texttt{SSP1ADD} &= \{\text{Fosc}/(4 \times \text{bps})\} - 1 \\ &= \{32 \times 10^6/(4 \times 400000)\} - 1 \\ &= 19 \\ &= 0\text{x}13 \end{aligned}$$

23.5　I^{2}C I/O エクスパンダ IC MCP23017

I/O エクスパンダ IC MCP23017 は 16 ビットのデジタル入出力ポートを有する半導体部品で, マイコンのピン数の不足を補うことを目的とする部品です. MCP23017 は I^{2}C バスによりマイコンに接続され, 3 本の I^{2}C アドレス設定ピンにより最大 8(= 2^3) 個までをひとつの I^{2}C バス上に接続可能です. 図 23.5 に MCP23017 の内部構造の概略を示します. ふたつの 8 ビット GPIO (general purpose input/output) A と B があり, ひとつのビットがひとつのデジタル入出力ポートで GPA0, ..., GPA7, GPB0, ..., GPB7 と名前が与えられています. 各ポートそれぞれに独立して設定が可能で, 設定可能な項目の一例は以下の通りです.

- IODIR: 入力/出力の方向の設定
- IOPOL: 入力の極性反転読み込みの設定
- GPINTEN: 入力変化による割り込み発生の設定
- DEFVAL: 入力変化の比較基準値の設定
- INTCON: 入力変化の比較値を DEFVAL の値とするかポート入力の前の値とするかを選択
- GPPU: プルアップの設定

これら以外にも設定項目がありますが本書では使用しないので説明は省略します. 詳しくはデータシートを各自で参照して下さい.

23.5.1　レジスタ

MCP23017 の内部には以下を目的とするレジスタが各種用意されています.

- MCP23017 の設定
- 各ポートの入力値の読み出し
- 各ポートの出力値の書き込み

表 23.2: MCP23017 のレジスタ表の抜粋 (BANK=0)

レジスタ名	アドレス	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
IODIRA	0x00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0
IODIRB	0x01	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0
IPOLA	0x02	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0
IPOLB	0x03	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0
GPINTENA	0x04	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0
GPINTENB	0x05	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0
DEFVALA	0x06	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0
DEFVALB	0x07	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0
INTCONA	0x08	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0
INTCONB	0x09	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0
IOCON	0x0A	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	–
IOCON	0x0B	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	–
GPPUA	0x0C	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0
GPPUB	0x0D	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0
INTCAPA	0x10	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0
INTCAPB	0x11	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0
GPIOA	0x12	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0
GPIOB	0x13	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0

- 各ポートの設定

表 23.2 に MCP23017 のレジスタ表の抜粋を示します. 各設定項目名は GPIO A と B それぞれに用意されており, 例えば GPIO A に対する IODIR は IODIRA, GPIO B には IODIRB とそれぞれ名前がつけられています. この表から分かる通り各レジスタは GPIO A および B のそれぞれ 8 ビットをひとまとめにした単位で構成されています. デジタル I/O 入出力は以下のレジスタを使用して行います.

- GPIOA: GPA0, ..., GPA7 の入力および出力のためのレジスタ
- GPIOB: GPB0, ..., GPB7 の入力および出力のためのレジスタ

レジスタの中でのビットの順序は GPA0 は GPIOA レジスタ第 0 ビット, GPA1 は GPIOA レジスタ第 1 ビット, ... となっています. 他のレジスタについても同様です.

なお表 23.2 は BANK=0 の場合における場合です. MCP23017 にはバンク切り替えによりレジスタのアドレス体系を切り替えることが可能で, IOCON レジスタの第 7 ビット BANK で設定します. リセット直後では BANK=0 と設定されており, 明示的に変更しない限りはその値は保持されます. 本書ではバンク切り替えを使用しないので詳細は省略します.

23.5.2　入出力設定

各ポートに対する基本的な入出力の設定を説明します. 以下では具体例として GPA0 の場合を示しますが他の場合も同様です.

- IODIRA.IO0
  - 1: GPA0 ピンを入力として設定します.
  - 0: GPA0 ピンを出力として設定します.
- IPOLA.IP0

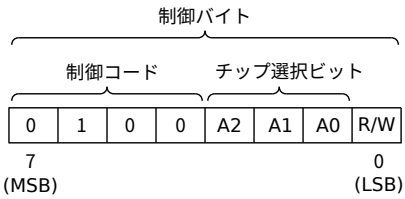


図 23.6: MCP23017 の制御バイト

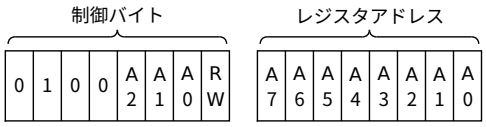


図 23.7: MCP23017 のアドレスシーケンス

  - 1: GPIOA.GP0 の値と GPA0 ピンの値は反転した値とします.
  - 0: GPIOA.GP0 の値と GPA0 ピンの値は同じ値とします.
- GPINTENA.GPINT0
  - 1: 入力変化割り込み機能を有効にします.
  - 0: 入力変化割り込み機能を無効にします.

  DEFVALA.DEF0 および INTCONA.IOC0 の設定も関係します.
- DEFVALA.DEF0

  INTCONA.IOC0 の設定による比較値を設定します.
- INTCONA.IOC0

  入力変化割り込みの発生要因を設定します.
  - 1: 入力ピン GPA0 の値が DEFVALA.DEF0 と異なる値になると入力変化割り込みを発生します.
  - 0: 比較対象を入力ピン GPA0 の前の値とします. すなわち入力ピン GPA0 の値が変化すると入力変化割り込みを発生します.
- GPPUA.PU0
  - 1: プルアップ機能を有効にします.
  - 0: プルアップ機能を無効にします.
- GPIOA.GP0
  - IODIRA.IO0=1 (入力) の場合:

    ピン GPA0 の入力値がこのビットの値として読み出しできます.
  - IODIRA.IO0=0 (出力) の場合:

    このビットに書き込んだ値がピン GPA0 の出力値となります.

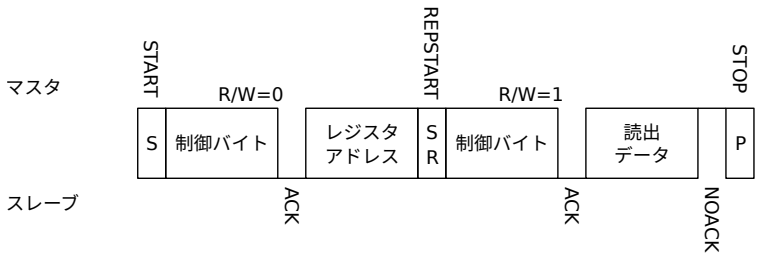
  ただし IPOLA.IP0 で反転設定している時は反転した値となります.
- INTCAPA.ICP0

  割り込み発生時の GPA0 ピンの値を保持します. この値の保持は INTCAPA.ICP0 または GPIOA.GP0 の値が読み出されるまで続きます.
  - 1: 割り込み発生時の GPA0 ピンの値は 1.
  - 0: 割り込み発生時の GPA0 ピンの値は 0.

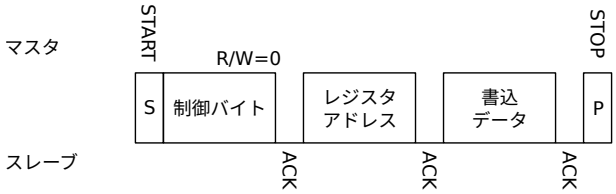
23.5.3 I^2C プロトコル

I/O エクスパンダ IC MCP23017 の I^2C 通信プロトコルを説明します. ここでは本書の内容の理解に必要なものだけを説明しますので, 残りは各自でデータシートを参照して下さい.

図 23.6 に MCP23017 の制御バイトを示します. 制御バイト内にはチップ選択ビット A2, A1, A0 があり, MCP23017 のハードウエアでのアドレス設定に合致する場合にのみ以降の I^2C パケットを処理します. チップ



(a) レジスタ読み出し



(b) レジスタ書き込み

図 23.8: MCP23017 の I^{2}C 通信プロトコル (一部)

選択ビットが 3 ビットあることから, I^{2}C バス上には最大 8 つの MCP23017 が接続可能です.

図 23.7 に MCP23017 のアドレスシーケンスを示します. 制御バイトに続いてアクセス対象のレジスタアドレスを 8 ビットで指定します. レジスタのアドレスは表 23.2 で示した通りです. 図 23.8 に MCP23017 I^{2}C 通信プロトコルを示しますが, ここでは一度にひとつのレジスタを読み書きする場合のみを示しています. 複数のレジスタの順次読み出しあるいは順次書き込みも可能ですが, 説明は省略します.

23.6　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します. ほとんどは第 22 章と共通で, 新たに作成するのは PIC マイコンにおける I^{2}C 通信を実現する MSSP モジュール部分とポイント制御部分です. 適宜第 22 章の説明を読み返して復習して下さい.

表 23.3: 【ソースファイル一覧】 パワーパック: ポイント制御機能付テレビリモコン式

初	頁	番号	ファイル名	内容	行数
	355	22.2	PowerPack-IR.c	パワーパックメイン関数	67
	356	22.3	pp-control-ir.c	テレビリモコン制御パワーパック: 実装	134
	193	10.4	pp-control-vr.c	ボリューム制御インターフェース: 実装	58
	359	22.4	led-ir-mon.h	受信モニタ LED: ヘッダ	10
	359	22.5	led-ir-mon.c	受信モニタ LED: 実装	27
	360	22.6	keysym.h	キーシンボル: ヘッダ	32
	360	22.7	cmd.h	遠隔制御実行: ヘッダ	12
	361	22.8	cmd.c	遠隔制御実行: 実装	112
	363	22.9	ir-frame.h	赤外線フレームデコーダ: ヘッダ	10
	365	22.10	ir-frame.c	赤外線フレームデコーダ: 実装	110
	367	22.11	rc.h	テレビリモコン情報: ヘッダ	45
	367	22.12	rc.c	テレビリモコン情報: 実装	30

(つづく)

(つづき)

初	頁	番号	ファイル名	内容	行数
	368	22.13	rc-data.c	各種テレビリモコン仕様: 実装	315
	373	22.14	auto-sel.c	テレビリモコン自動選択機能: 実装	63
	390	23.1	if.h	ポイント制御: ヘッダ	28
	374	22.16	if-toq.h	ポイント切替動作キュー: ヘッダ	12
✓	392	23.2	if-toq.c	ポイント切替動作キュー: 実装	122
	189	10.1	pp.h	パワーパック関数宣言: ヘッダ	46
	172	9.2	pp-const.h	パワーパック定数: ヘッダ	17
	191	10.3	pp-misc.c	パワーパック諸関数: 実装	111
	196	10.6	pp-sb.h	低速ブースト機能: ヘッダ	8
	198	10.7	pp-sb.c	低速ブースト機能: 実装	171
	129	6.1	incr-lin.h	線形増加アルゴリズム: ヘッダ	9
	130	6.2	incr-lin.c	線形増加アルゴリズム: 実装	78
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76
✓	394	23.3	i2c.h	I^2C インターフェース関数: ヘッダ	25
✓	394	23.4	i2c.c	I^2C インターフェース関数: 実装	111
✓	396	23.5	i2c-mcp23017.h	MCP23017 アクセス関数: ヘッダ	53
✓	397	23.6	i2c-mcp23017.c	MCP23017 アクセス関数: 実装	109
✓	399	23.7	if-turnout.c	インターフェースボード・ポイント部: 実装	110
	354	22.1	io.h	I/O ピン割付: ヘッダ	45
	375	22.18	mcr.h	MCR・周辺モジュール: ヘッダ	9
	375	22.19	mcr.c	MCR・周辺モジュール: 実装	161
	175	9.6	mcr-motor.h	モーター制御周辺モジュール: ヘッダ	25
	177	9.7	mcr-motor.c	モーター制御周辺モジュール: 実装	98

計 2458 行 (うち初出 530 行, 既出 1928 行)

ソースファイル if.h (ポイント制御: ヘッダ)

ポイント制御を実現する関数群のプロトタイプ宣言と関係する定数を定義します. 多数のポイントが同時に切り替えられることがないように, 複数のポイントの切り替えは 100 ミリ秒の時間差で順次行います.

ソースファイル番号 23.1: if.h

```
1  #if !defined(__IF_H__)
2  #define __IF_H__
3
4  /* ポイント番号 */
5  typedef uint8_t if_to_id;
6  #define IF_TO_NULL  0
7
8  /* ポイント開通方向 */
9  enum if_turnout_dir_e {
10   TURNOUT_DIR_NORMAL,   /* 定位 */
11   TURNOUT_DIR_REVERSE, /* 反位 */
12   TURNOUT_DIR_NONE,     /* 未定 */
```

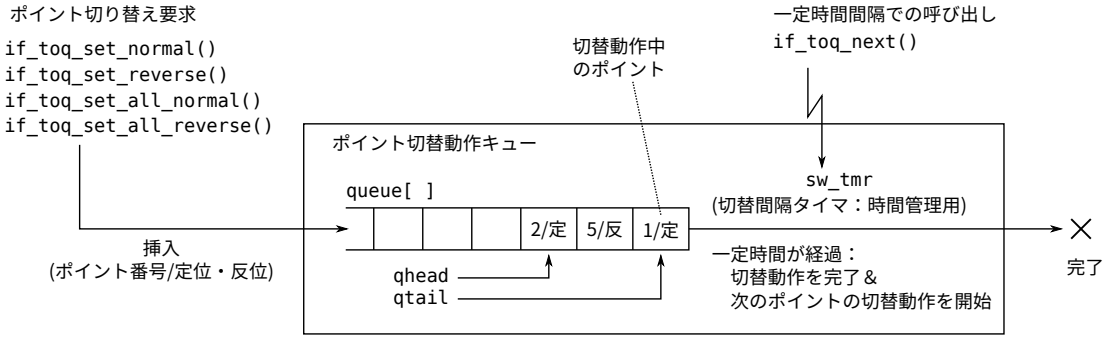


図 23.9: ポイント切替動作キューによるポイントの順次切り替えの仕組み

```
13  };
14
15  /* ポイント切り替え時間間隔 [mSec] */
16  #define IF_TURNOUT_TIME     100
17
18  /* 最大インターフェースモジュール数 */
19  #define IF_MAX_MODULES        8
20  /* 1モジュールあたりのポイント数 */
21  #define N_TURNOUTS_PER_MOD    8
22  /* 最大のポイント数 */
23  #define IF_MAX_TURNOUTS    (IF_MAX_MODULES * N_TURNOUTS_PER_MOD)
24
25  extern void    if_turnout_init(void);
26  extern int8_t if_turnout_action(if_to_id p, enum if_turnout_dir_e d);
27
28  #endif /*__IF_H__*/
```

ソースファイル if-toq.c (ポイント切替動作キュー: 実装)

ポイント切替動作キューに関連する関数の実装を行います. ここで, キュー (queue) とは待ち行列のことです. このソースファイルの目的は, 多数のポイントが一斉に切り替られて大電流が流れてブレーカが落ちることがないようにすることです. この目的を達成するために時間差をおいてポイントを順次切り替えるようにします. このソースファイルでは図 23.9 に示すように切り替えるポイントを記憶する待ち行列を持ち, 一定時間が経過すれば次に切り替えるポイントを待ち行列から取り出して順次切り替えを実行してゆきます. 実際の出力は MCP23017 のポートを操作して行いますが, その動作は後述のソースファイル if-turnout.c で行います.

- if_toq_init()
  ポイント切替動作キューを初期化します.
- if_toq_set_normal(), if_toq_set_reverse()
  指定したポイントの切り替えを要求しますが, この関数では要求を単に待ち行列に入れるだけです.
- if_toq_set_all_normal(), if_toq_set_all_reverse()
  すべてのポイントの切り替えを要求しますが, この関数では要求を単に待ち行列に入れるだけです.
- if_toq_is_done()
  要求されたポイントの切り替えすべてが完了したか否かを検査します. 完了していれば真を, まだ完了していなければ偽を返します.
- if_toq_next()
  ポイントの切り替え処理を進めます. もし一定時間経過して 2 つのポイントの切り替え時間間隔が過ぎれば次のポイントの切り替えを行います. 実際の切り替えは後述のソースファイル if-turnout.c の中

の関数 if_turnout_action() で行います. そしてこの関数の返り値には, 要求されたポイントの切り替えがすべて完了したか否かを返します. まだ完了していなければ真を, 完了していれば偽を返します.

ソースファイル番号 23.2: if-toq.c

```
/*------------------------------------------------------
 * ポイント切替動作キュー
 *------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "if.h"
#include "if-toq.h"

/* 切替動作キュー */
struct if_toq_s {
  if_to_id              p;
  enum if_turnout_dir_e d;
};
static struct if_toq_s queue[IF_MAX_TURNOUTS];
static uint8_t qnitems = 0;
static uint8_t qhead = 0;
static uint8_t qtail = 0;

/* 切替間隔タイマ */
static uint16_t sw_tmr = 0;

/*------------------------------------------------------
 * 初期化
 *------------------------------------------------------*/
void if_toq_init(void)
{
  /* 切替動作キュー初期化 */
  qnitems = 0;
  qhead = 0;
  qtail = 0;
  sw_tmr = 0;
  /* インターフェースボード初期化 */
  if_turnout_init();
}

/*------------------------------------------------------
 * 切替動作のエンキュー
 *------------------------------------------------------*/

static void if_toq_set(if_to_id p,
                       enum if_turnout_dir_e d)
{
  if (qnitems == IF_MAX_TURNOUTS) {
    return; /* 失敗(満杯) */
  }
  qnitems++;
  queue[qhead].p = p;
  queue[qhead].d = d;
  qhead++;
  if (qhead == IF_MAX_TURNOUTS) {
    qhead = 0;
  }
}

void if_toq_set_normal(if_to_id p)
{
  if_toq_set(p, TURNOUT_DIR_NORMAL);
}

void if_toq_set_reverse(if_to_id p)
{
  if_toq_set(p, TURNOUT_DIR_REVERSE);
}
```

```
66
67  /*-------------------------------------------------------
68   * 切替動作の設定(全ポイント)
69   *-------------------------------------------------------*/
70
71  static void if_toq_set_all(enum if_turnout_dir_e d)
72  {
73    if_to_id p = 1;
74    for (p = 1; p <= IF_MAX_TURNOUTS; p++) {
75      if_toq_set(p, d);
76    }
77  }
78
79  void if_toq_set_all_normal(void)
80  {
81    if_toq_set_all(TURNOUT_DIR_NORMAL);
82  }
83
84  void if_toq_set_all_reverse(void)
85  {
86    if_toq_set_all(TURNOUT_DIR_REVERSE);
87  }
88
89  /*-------------------------------------------------------
90   * 切替動作はすべて完了したか否か
91   *-------------------------------------------------------*/
92  int8_t if_toq_is_done(void)
93  {
94    return (qnitems == 0);
95  }
96
97  /*-------------------------------------------------------
98   * 切替動作をデキューして実行
99   *-------------------------------------------------------*/
100 int8_t if_toq_next(uint16_t tcyc)
101 {
102   /* 前回の切替から一定時間以上が経過したか */
103   if (sw_tmr < IF_TURNOUT_TIME) { /* していない */
104     sw_tmr += tcyc;
105   } else {                        /* した */
106     if (if_toq_is_done()) { /* すべて完了 */
107       return 0; /* 次はない */
108     } else {                    /* 次の切替を行う */
109       /* インターフェースボード接続あり: 実際に切替動作の完了を待機 */
110       /* インターフェースボード接続なし: 待機せずに次の切替へ */
111       qnitems--;
112       if (if_turnout_action(queue[qtail].p, queue[qtail].d)) {
113         sw_tmr = 0;
114       }
115       qtail++;
116       if (qtail == IF_MAX_TURNOUTS) {
117         qtail = 0;
118       }
119     }
120   }
121   return 1; /* 次がある */
122 }
```

ソースファイル i2c.h (I²C インターフェース関数: ヘッダ)

この後に紹介する i2c.c で実装する関数群のプロトタイプ宣言です．このソースファイルと次に紹介する i2c.c は I/O エクスパンダ IC MCP23017 での使用に限定したものではなく，さまざまな I²C デバイスに利用可能な汎用的なものです．第 25 章の EEPROM 書き込み器や第 28 章のポイント一括切替器ではこのソースファイルを利用して I²C EEPROM メモリのアクセスを行います．本ソースファイルで実装する関数の概要は以下の通りです．

- I2C_Init()
  I²C インターフェース関数群の初期化を行います.
- I2C_Deinit()
  I²C インターフェース関数群の使用終了を行います.
- I2C_Start()
  I²C バスにスタートコンディション (START 信号) を出力します.
- I2C_Repstart()
  I²C バスにリピートスタートコンディションを出力します.
- I2C_Stop()
  I²C バスにストップコンディション (STOP 信号) を出力します.
- I2C_Exist()
  指定アドレスのデバイスが I²C バスに接続されているか否かを検出します.
- I2C_Read()
  I²C バスからデータを 1 バイト読み出します.
- I2C_Write()
  I²C バスへデータを 1 バイト書き込みます.

ソースファイル番号 23.3: i2c.h

```
#if !defined(__I2C_H__)
#define __I2C_H__

/* ACK Type */
enum i2c_ack_type_e {
  I2C_ACKTYPE_ACK,
  I2C_ACKTYPE_NOACK
};

/* I2Cボーレート設定値, 計算式: (_XTAL_FREQ/BITRATE)/4-1 */
#define I2C_BAUD_RATE_SSPxADD_1MHZ(XFREQ)   \
        ((XFREQ/1000000)/4 - 1)
#define I2C_BAUD_RATE_SSPxADD_400KHZ(XFREQ) \
        ((XFREQ/ 400000)/4 - 1)

extern void    I2C_Init(uint8_t ssp1add_value);
extern void    I2C_Deinit(void);
extern void    I2C_Start(void);
extern void    I2C_Repstart(void);
extern void    I2C_Stop(void);
extern int8_t  I2C_Exist(uint8_t ctlbyte, uint8_t addr);
extern uint8_t             I2C_Read(enum i2c_ack_type_e acktype);
extern enum i2c_ack_type_e I2C_Write(uint8_t data);

#endif /*__I2C_H__*/
```

ソースファイル i2c.c (I²C インターフェース関数: 実装)

I²C インターフェース関数の実装です. PIC マイコンの MSSP モジュールを使用して通信を行います. データシート従い I²C バスの操作を行います.

ソースファイル番号 23.4: i2c.c

```
/*********************************************
 * I2Cドライバ
 * by Hirotsugu Kakugawa
 *********************************************/
```

```
#include <xc.h>
#include <stdint.h>
#include "i2c.h"
#include "io.h"

/* I2Cモジュール初期化 */
void I2C_Init(uint8_t ssp1add_value)
{
  /* 注意(データシートより):
   *   SCL と SDA はデジタル入力ピンとして設定しないと,
   *   I2Cは正しく動作しない */
  SSP1IF = 0;
  BCL1IF = 0;
  SSP1STAT = 0x80;  /* SMP = 1 */
  SSP1CON2 = 0x00;
  SSP1CON3 = 0x00;
  SSP1ADD  = ssp1add_value;
  SSP1CON1bits.SSPM  = 0x08; /* I2C Master */
  SSP1CON1bits.SSPEN = 1;    /* Enabled */
}

/* I2Cモジュール停止 */
void I2C_Deinit(void)
{
  SSP1CON1 = 0;
  SSP1ADD  = 0;
}

/* START 送出 */
void I2C_Start(void)
{
  SSP1CON2bits.SEN = 1;
  while (SSP1CON2bits.SEN != 0) {
    ;
  }
}

/* REPSTART 送出 */
void I2C_Repstart(void)
{
  SSP1CON2bits.RSEN = 1;
  while (SSP1CON2bits.RSEN != 0) {
    ;
  }
}

/* STOP 送出 */
void I2C_Stop(void)
{
  SSP1CON2bits.PEN = 1;
  while (SSP1CON2bits.PEN != 0) {
    ;
  }
}

/* 1バイト送信 */
enum i2c_ack_type_e I2C_Write(uint8_t data)
{
  SSP1BUF = data;
  while (SSP1STATbits.R_nW != 0) {
    ;
  }
  return (SSP1CON2bits.ACKSTAT == 0)
    ? I2C_ACKTYPE_ACK : I2C_ACKTYPE_NOACK;
}

/* 1バイト読出 */
uint8_t I2C_Read(enum i2c_ack_type_e acktype)
{
  uint8_t data;
  SSP1CON2bits.RCEN = 1;
```

```
76    while (SSP1CON2bits.RCEN != 0) {
77      ;
78    }
79    data = SSP1BUF;
80    if (acktype == I2C_ACKTYPE_ACK) {
81      SSP1CON2bits.ACKDT = 0;
82    } else {
83      SSP1CON2bits.ACKDT = 1;
84    }
85    SSP1CON2bits.ACKEN = 1;
86    while (SSP1CON2bits.ACKEN != 0) {
87      ;
88    }
89
90    if (acktype == I2C_ACKTYPE_NOACK) {
91      I2C_Stop();
92    }
93    return data;
94  }
95
96  /* 指定アドレスのデバイス有無の検査 */
97  int8_t I2C_Exist(uint8_t ctlbyte, uint8_t addr)
98  {
99    int8_t exist = 0;
100   uint8_t i2c_addr = ctlbyte | ((unsigned)addr << 1);
101   I2C_Start();
102   switch (I2C_Write(i2c_addr)) {
103   case I2C_ACKTYPE_ACK:   /* 有 */
104     exist = 1;
105     break;
106   case I2C_ACKTYPE_NOACK: /* 無 */
107     exist = 0;
108     break;
109   }
110   return exist;
111 }
```

ソースファイル i2c-mcp23017.h (MCP23017 アクセス関数: ヘッダ)

MCP23017 へのアクセスを行う関数群のプロトタイプ宣言を行います. レジスタ番号を記号定数として定義します.

ソースファイル番号 23.5: i2c-mcp23017.h

```
1   /*********************************************
2    * MCP23017
3    * Microchip Technology
4    * I2C 16-bit I/O Extender
5    *********************************************/
6   #if !defined(__I2C_MCP23017_H__)
7   #define __I2C_MCP23017_H__
8
9   /* 制御バイト */
10  #define I2C_MCP23017_CTLBYTE        0x40
11
12  /* レジスタ (BANK=0) */
13  #define I2C_MCP23017_REG_IODIRA     0x00
14  #define I2C_MCP23017_REG_IODIRB     0x01
15  #define I2C_MCP23017_REG_IPOLA      0x02
16  #define I2C_MCP23017_REG_IPOLB      0x03
17  #define I2C_MCP23017_REG_GPINTENA   0x04
18  #define I2C_MCP23017_REG_GPINTENB   0x05
19  #define I2C_MCP23017_REG_DEFVALA    0x06
20  #define I2C_MCP23017_REG_DEFVALB    0x07
21  #define I2C_MCP23017_REG_INTCONA    0x08
22  #define I2C_MCP23017_REG_INTCONB    0x09
23  #define I2C_MCP23017_REG_IOCON      0x0A
24  #define I2C_MCP23017_REG_IOCON_     0x0B
25  #define I2C_MCP23017_REG_GPPUA      0x0C
```

```
26 #define I2C_MCP23017_REG_GPPUB      0x0D
27 #define I2C_MCP23017_REG_INTFA      0x0E
28 #define I2C_MCP23017_REG_INTFB      0x0F
29 #define I2C_MCP23017_REG_INTCAPA    0x10
30 #define I2C_MCP23017_REG_INTCAPB    0x11
31 #define I2C_MCP23017_REG_GPIOA      0x12
32 #define I2C_MCP23017_REG_GPIOB      0x13
33 #define I2C_MCP23017_REG_OLATA      0x14
34 #define I2C_MCP23017_REG_OLATB      0x15
35
36 extern void    i2c_mcp23017_init(uint8_t caddr);
37 extern void    i2c_mcp23017_reg_write(uint8_t caddr,
38                                       uint8_t reg,
39                                       uint8_t data);
40 extern uint8_t i2c_mcp23017_reg_read(uint8_t caddr,
41                                      uint8_t reg);
42 extern void    i2c_mcp23017_a_dir_set(uint8_t caddr,
43                                       uint8_t dir);
44 extern void    i2c_mcp23017_b_dir_set(uint8_t caddr,
45                                       uint8_t dir);
46 extern void    i2c_mcp23017_a_write(uint8_t caddr,
47                                     uint8_t data);
48 extern void    i2c_mcp23017_b_write(uint8_t caddr,
49                                     uint8_t data);
50 extern uint8_t i2c_mcp23017_a_read(uint8_t caddr);
51 extern uint8_t i2c_mcp23017_b_read(uint8_t caddr);
52
53 #endif /*__I2C_MCP23017_H__*/
```

ソースファイル i2c-mcp23017.c (MCP23017 アクセス関数: 実装)

MCP23017 へのアクセスを行う関数群の実装を行います. ソースファイル i2c.c が提供する関数群を呼び出して図 23.8 に示した MCP23017 I^2C 通信プロトコルを実装します.

ソースファイル番号 23.6: i2c-mcp23017.c

```
1  /*********************************************
2   * MCP23017
3   * Microchip Technology
4   * I2C 16-bit I/O Extender
5   *********************************************/
6  #include <xc.h>
7  #include <stdint.h>
8  #include "i2c.h"
9  #include "i2c-mcp23017.h"
10
11 /*-------------------------------------------*/
12
13 enum i2c_mcp23017_rw_e {
14   I2C_MCP23017_WR = 0,
15   I2C_MCP23017_RD = 1,
16 };
17
18 /*-------------------------------------------*/
19
20 /* 制御バイト送出 */
21 static enum i2c_ack_type_e
22 i2c_mcp23017_send_ctl_byte(enum i2c_mcp23017_rw_e r_w,
23                            uint8_t caddr)
24 {
25   return
26     I2C_Write(/* 制御バイト */
27               I2C_MCP23017_CTLBYTE
28               /* A2,A1,A0: チップ選択ビット */
29               | (((unsigned)caddr & 0x07) << 1)
30               /* 読み/書き */
31               | ((r_w == I2C_MCP23017_RD) ? 0x01 : 0x00));
32 }
33
```

```
34  /* 初期化 */
35  void i2c_mcp23017_init(uint8_t caddr)
36  {
37    ;
38  }
39
40  /* レジスタへの書き込み */
41  void i2c_mcp23017_reg_write(uint8_t caddr,
42                              uint8_t reg, uint8_t data)
43  {
44  retry:
45    I2C_Start();
46    if (i2c_mcp23017_send_ctl_byte(I2C_MCP23017_WR, caddr)
47        != I2C_ACKTYPE_ACK) {
48      goto retry;
49    }
50    I2C_Write(reg);
51    I2C_Write(data);
52    I2C_Stop();
53  }
54
55  /* レジスタの読み出し */
56  uint8_t i2c_mcp23017_reg_read(uint8_t caddr, uint8_t reg)
57  {
58   retry:
59    I2C_Start();
60    if (i2c_mcp23017_send_ctl_byte(I2C_MCP23017_WR, caddr)
61        != I2C_ACKTYPE_ACK) {
62      goto retry;
63    }
64    I2C_Write(reg);
65    I2C_Repstart();
66    if (i2c_mcp23017_send_ctl_byte(I2C_MCP23017_RD, caddr)
67        != I2C_ACKTYPE_ACK) {
68      goto retry;
69    }
70    uint8_t data = I2C_Read(I2C_ACKTYPE_NOACK);
71    I2C_Stop();
72    return data;
73  }
74
75  /* PORT A I/O 方向設定 */
76  void i2c_mcp23017_a_dir_set(uint8_t caddr, uint8_t dir)
77  {
78    i2c_mcp23017_reg_write(caddr, I2C_MCP23017_REG_IODIRA, dir);
79  }
80
81  /* PORT B I/O 方向設定 */
82  void i2c_mcp23017_b_dir_set(uint8_t caddr, uint8_t dir)
83  {
84    i2c_mcp23017_reg_write(caddr, I2C_MCP23017_REG_IODIRB, dir);
85  }
86
87  /* PORT A 書き込み */
88  void i2c_mcp23017_a_write(uint8_t caddr, uint8_t data)
89  {
90    i2c_mcp23017_reg_write(caddr, I2C_MCP23017_REG_GPIOA, data);
91  }
92
93  /* PORT B 書き込み */
94  void i2c_mcp23017_b_write(uint8_t caddr, uint8_t data)
95  {
96    i2c_mcp23017_reg_write(caddr, I2C_MCP23017_REG_GPIOB, data);
97  }
98
99  /* PORT A 読み出し */
100 uint8_t i2c_mcp23017_a_read(uint8_t caddr)
101 {
102   return i2c_mcp23017_reg_read(caddr, I2C_MCP23017_REG_GPIOA);
103 }
104
```

```
105 /* PORT B 読み出し */
106 uint8_t i2c_mcp23017_b_read(uint8_t caddr)
107 {
108   return i2c_mcp23017_reg_read(caddr, I2C_MCP23017_REG_GPIOB);
109 }
```

ソースファイル if-turnout.c (ポイント制御: 実装)

I^2C 通信を通じて MCP23017 を操作し, ポイント制御出力の制御を実現します. 本装置ではポイントインターフェースボードは 1 枚のみの接続としていますが, 第 28 章で紹介するポイント一括切替器では最大 8 枚の接続を可能としています. そこで本ソースファイルは 8 枚の接続を想定した汎用的な構造にしています.

実装している関数とその概要は以下の通りです.

- if_turnout_init()
  MCP23017 の初期化を行います. I^2C にコマンドを送り MCP23017 の接続の有無を調べ, 接続がある場合には全ポートを出力に設定し, 初期の出力は全ポートを L とします. ポートの出力は TBD62083APG を介してポイント制御出力とする回路になっている点に注意して下さい. TBD62083APG の出力はオープンドレインであることから, MCP23017 のポートが L の場合はポイント制御出力は HiZ になり, H の場合はポイント制御出力は L となります. 従って初期化直後ではポイント制御出力はすべて HiZ になります.
- if_turnout_action()
  ポイント切替を実行します. この関数の第 1 引数はポイント番号 $(1, 2, ..., 64)$ で, 第 2 引数は開通方向です. まず引数のポイント番号から, そのポイントに該当するポイントインターフェースボードの番号 $(0, 1, , ..., 7)$ とそのポイントインターフェースボード内部でのポイント番号 $(0, 1, ..., 7)$ を求めます. そして開通方向から該当するポートの値を決定して, 該当するポイントインターフェースボードの出力を変更します.

ソースファイル番号 23.7: if-turnout.c

```
1  /*-----------------------------------------------------
2   * インターフェースボード(ポイント制御出力)
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "io.h"
8  #include "if.h"
9  #include "if-toq.h"
10 #include "i2c.h"
11 #include "i2c-mcp23017.h"
12
13 /* MCP23017 ポート割付
14  *   GPA0,GPA1: ポイント1
15  *   GPA2,GPA3: ポイント2
16  *   GPA4,GPA5: ポイント3
17  *   GPA6,GPA7: ポイント4
18  *   GPB0,GPB1: ポイント5
19  *   GPB2,GPB3: ポイント6
20  *   GPB4,GPB5: ポイント7
21  *   GPB6,GPB7: ポイント8
22  */
23
24 /* インターフェースボードの接続の有無 */
25 static uint8_t if_module[IF_MAX_MODULES];
26 /* ポートA,ポートBの書き込み値 */
27 static uint8_t if_wr_gpa[IF_MAX_MODULES];
```

```
28  static uint8_t if_wr_gpb[IF_MAX_MODULES];
29
30  /*-----------------------------------------------------
31   * 初期化
32   *-----------------------------------------------------*/
33  void if_turnout_init(void)
34  {
35    /* I2C モジュール初期化 */
36    I2C_Init(I2C_BAUD_RATE_SSPxADD_1MHZ(_XTAL_FREQ));
37    /* 各I2Cアドレスに対してMCP23017の接続を確認し初期化 */
38    uint8_t caddr = 0;
39    for (caddr = 0; caddr < IF_MAX_MODULES; caddr++) {
40      CLRWDT();
41      /* 書き込み値 */
42      if_wr_gpa[caddr] = 0x00;
43      if_wr_gpb[caddr] = 0x00;
44      if (!I2C_Exist(I2C_MCP23017_CTLBYTE, caddr)) {
45        /* このアドレスに接続無し */
46        if_module[caddr] = 0;
47      } else {
48        /* このアドレスに接続有り */
49        if_module[caddr] = 1;
50        /* 初期化 */
51        i2c_mcp23017_init(caddr);
52        if ((0 <= caddr) && (caddr < IF_MAX_MODULES)) {
53          /* IODIR: I/O方向 (出力) */
54          i2c_mcp23017_a_dir_set(caddr, 0x00);
55          i2c_mcp23017_b_dir_set(caddr, 0x00);
56        }
57        /* ポート初期値書き込み */
58        i2c_mcp23017_a_write(caddr, if_wr_gpa[caddr]);
59        i2c_mcp23017_b_write(caddr, if_wr_gpb[caddr]);
60      }
61    }
62  }
63
64  /*-----------------------------------------------------
65   * 切替実行
66   *-----------------------------------------------------*/
67  int8_t if_turnout_action(if_to_id p,
68                           enum if_turnout_dir_e d)
69  {
70    if ((p == IF_TO_NULL) || (IF_MAX_TURNOUTS < p)) {
71      return 0;
72    }
73    /* 該当のインターフェースボード番号とポート番号
74     * caddr := (p-1) / N_TURNOUTS_PER_MOD
75     *     g := (p-1) % N_TURNOUTS_PER_MOD */
76    uint8_t caddr = 0;
77    int8_t g = p - 1;
78    while (g >= N_TURNOUTS_PER_MOD) {
79      g -= N_TURNOUTS_PER_MOD;
80      caddr++;
81    }
82    /* ポイント制御出力の更新 (ポート割付依存) */
83    const uint8_t bit_mask[4] = { 0x03, 0x0C, 0x30, 0xC0 };
84    const uint8_t bit_nor[4]  = { 0x01, 0x04, 0x10, 0x40 }; /*定位*/
85    const uint8_t bit_rev[4]  = { 0x02, 0x08, 0x20, 0x80 }; /*反位*/
86    /* 出力 (ポート割付依存) */
87    if ((0 <= g) && (g <= 3)) { /* ポートA */
88      if_wr_gpa[caddr] &= ~bit_mask[g];
89      if (d == TURNOUT_DIR_NORMAL) { /*定位*/
90        if_wr_gpa[caddr] |= bit_nor[g];
91      } else {                       /*反位*/
92        if_wr_gpa[caddr] |= bit_rev[g];
93      }
94      if (if_module[caddr] != 0) {
95        i2c_mcp23017_a_write(caddr, if_wr_gpa[caddr]);
96      }
97    } else if ((4 <= g) && (g <= 7)) { /* ポートB */
```

```
98        if_wr_gpb[caddr] &= ~bit_mask[g - 4];
99        if (d == TURNOUT_DIR_NORMAL) { /*定位*/
100         if_wr_gpb[caddr] |= bit_nor[g - 4];
101       } else {                      /*反位*/
102         if_wr_gpb[caddr] |= bit_rev[g - 4];
103       }
104       if (if_module[caddr] != 0) {
105         i2c_mcp23017_b_write(caddr, if_wr_gpb[caddr]);
106       }
107     }
108     /* 返り値: 実際にインターフェースボード接続があるか否か */
109     return (if_module[caddr] != 0);
110   }
```

課題

1. 手動制御モード時には手動でポイントの切り替えができるよう, その方法を考案し実現して下さい.
2. 本書で紹介したポイント制御機能では 8 つのポイントまでしか対応していません. より多くのポイントに対応するようにファームウエアを変更して下さい. テレビリモコンで 10 以上の値の入力の実現が必要で, 使用方法が単純明快でなければなりません.

第 VI 部

EEPROM を用いた高度なカスタマイズ技法

第 24 章

インテル HEX 形式への変換

本章ではパーソナルコンピュータ上のファイルをインテル HEX 形式 (Intel HEX format) に変換するソフトウエアを紹介します. インテル HEX 形式とは, 制御文字などの非表示文字を含んでいるファイルを表示文字だけからなるテキストファイルに符号化する方法のひとつです.

本ソフトウエアはサウンドデータ (第 27 章) や自動運転スクリプト (第 30 章) を第 25 章で紹介する EEPROM 書き込み器を使用する際に使用します. 一般的な一連の作業の全体像は図 24.1 で示す通りです.

本章で紹介するソフトウエアは図 24.2 に示すように Unix 系オペレーティングシステム (Linux や FreeBSD など) あるいは Microsoft 社 Windows オペレーティングシステムなどのパーソナルコンピュータ上で実行するものであり, PIC マイコン用のファームウエアではありません. 従って本ソフトウエアは開発時に使用するものであり, 開発環境の一部をなすものです.

本章で紹介するソフトウエアの特徴は以下の通りです.

- ファイルをインテル HEX 形式に符号化
- 標準 C 言語で記述した可搬性の高いコンソールアプリケーション
  特定のオペレーティングシステムや GUI に依存することなく様々なコンピュータ環境で永きに渡って動作可能です.

本章で学ぶ内容は以下の通りです.

- インテル HEX 形式
- 16 進ダンプリストの読み方

第 25 章で紹介する EEPROM メモリ書き込み器は, パーソナルコンピュータからインテル HEX 形式のファイルを受信して EEPROM メモリに書き込みます. 例えばサウンドデータファイルは制御文字などの非表示文

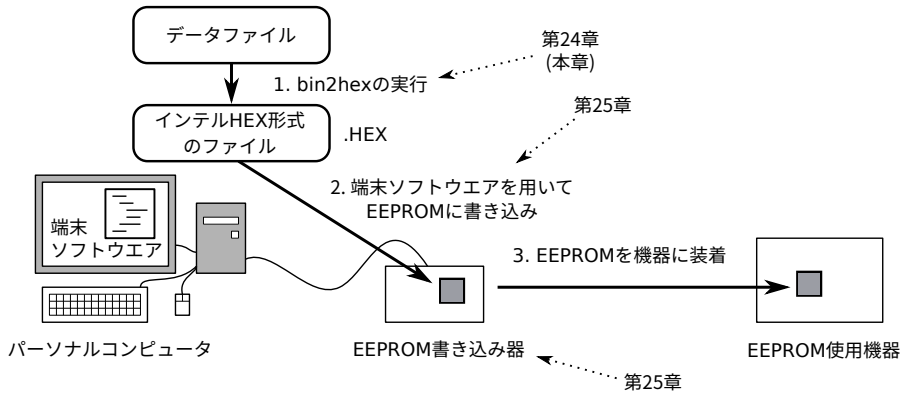


図 24.1: インテル HEX ファイルによる EEPROM 使用機器の利用のための作業手順



図 24.2: bin2hex 使用中の様子 (Ubuntu 上)

```
:020000040000FA ..... 配置アドレスを0x0000に指定（レコード種別0x04）
:10000000390304EE66ED2D24DB370A65ADF509D31F
:10001000619C56AC676FEE3A70B77E28C927C9025B
:100020002ACEF090BB1DB49655BFFC02B405D51581
:10003000A12CC2089BB0430C67C13430E8FE3212D9
:00000001FF ..... ファイル終端（レコード種別0x01）
```

各レコードで16バイトのデータを表現（レコード種別0x00）

図 24.3: インテル HEX 形式のファイル全体の例

字を含んでいるファイルですから, ファイルをそのまま送信してもファイル中のデータが制御文字と解釈される場合があり正しく送信できません. 特に Xon/Xoff との混同が起きると通信は停止します. これを防ぐために本書では, 特に小型マイコンでのファイルの送受信では広く用いられているインテル HEX 形式を採用します.

24.1　インテル HEX 形式

インテル HEX 形式の書式の概要を説明します. 以下に示すのは, 64 バイトのデータが含まれているファイルをインテル HEX 形式で符号化したものです.

```
:020000040000FA
:10000000390304EE66ED2D24DB370A65ADF509D31F
:10001000619C56AC676FEE3A70B77E28C927C9025B
:100020002ACEF090BB1DB49655BFFC02B405D51581
:10003000A12CC2089BB0430C67C13430E8FE3212D9
:00000001FF
```

図 24.3 ではこのインテル HEX 形式ファイルの内容の概略を説明しています.

上で示したインテル HEX 形式の元となったファイルの 16 進ダンプリストは以下の通りです. 各行には最大 16 バイトのデータが含まれていて, 行の先頭の 8 桁はその行のデータの開始アドレスを表します. 例えばファイルの 0 バイト目は 0x39 で, 0x11(=17) バイト目は 0x9C です.

```
          +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00000000: 39 03 04 EE 66 ED 2D 24 DB 37 0A 65 AD F5 09 D3
00000010: 61 9C 56 AC 67 6F EE 3A 70 B7 7E 28 C9 27 C9 02
00000020: 2A CE F0 90 BB 1D B4 96 55 BF FC 02 B4 05 D5 15
00000030: A1 2C C2 08 9B B0 43 0C 67 C1 34 30 E8 FE 32 12
```

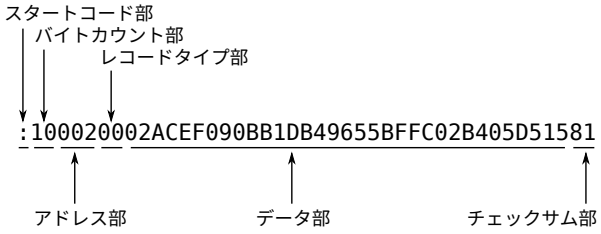


図 24.4: インテル HEX 形式のレコードの例

インテル HEX 形式とは以下の形式をした符号化手法です.

- ファイル全体では複数のレコード (record) で構成されます.
- 1 つのレコードでファイルの 1 行が構成されます.
- 各レコードは ASCII 文字からなる文字列で構成されます.
- 各レコードはコロン: で始まります.
- 各レコードは改行で終わります. その他の制御文字は含みません.

各レコード (各行) の構成を説明します. 以下では上で示したインテル HEX 形式の例での 4 行目を具体例として説明します. 図 24.4 も参照して下さい.

1. スタートコード部 (例 `:`)
   1 文字のコロン: で構成されます.
2. バイトカウント部 (例 `10`)
   2 桁の 16 進数文字列で構成され, このレコードに含まれているデータのバイト数を 8 ビットで表します. 以降の説明ではこのバイトカウント部が表す数値を n とします. 例では 0x10 すなわち 16 バイトがこのレコードに含まれています.
3. アドレス部 (例 `0020`)
   4 桁の 16 進数文字列で構成され, このレコードに含まれているデータが配置されるべきアドレスを 16 ビットで表します. 例ではアドレスは 0x0020 すなわち 32 であることを示しています.
4. レコードタイプ部 (例 `00`)
   2 桁の 16 進数文字列で構成され, このレコードの種別を表します. 詳しくは後述します. 例ではレコードの種別は 0x00 であることを示しています.
5. データ部 (例 `2ACEF090BB1DB49655BFFC02B405D515`)
   $2n$ 桁の 16 進数文字列で構成され, このレコードに含まれている n バイトのデータの列を表します. レコードの種別によってはデータ部がないものもあります. 例では 16 バイトのデータ 0x2A, 0xCE, 0xF0, ..., 0xD5, 0x15 を表しています. この順番でこれらデータは現在の配置アドレスから配置されてゆきます.
6. チェックサム部 (例 `81`)
   2 桁の 16 進数文字列で構成され, 伝送途中での通信エラーを検出するためのデータを表します. 通信エラーが発生していなければチェックサム部も含めてレコードに含まれるすべてのデータの和は 0 になります. もし計算結果が 0 でなければ通信エラーが発生しています. 通信エラーが発生していてもたまたま計算結果が 0 になる場合がありますのでエラー検出は完全ではありません.

レコードの種別には以下の通り 0x00 から 0x05 までの 6 種類があります.

- レコード種別 0x00 : データ列
  メモリに配置するデータの列を表します. バイトカウント部ではデータ数を, アドレス部では配置アドレスを, それぞれ表します.
- レコード種別 0x01 : ファイル終端
  インテル HEX 形式の最後の行であることを表します. 通常はバイトカウント部, アドレス部ともに 0 です.
- レコード種別 0x02 : 拡張セグメントアドレス
  アドレス部を 16 倍した値を配置アドレスとして設定します.
- レコード種別 0x03 および 0x05
  特定のマイクロプロセッサのための機能なので説明は省略します.
- レコード種別 0x04 : 拡張線形アドレス
  32 ビットアドレス方式を使用して配置アドレスを設定します. このレコードのアドレス部は不使用なので無視します. 配置アドレスの 32 ビットの中の上位 16 ビットにデータ部の最初の 2 バイトの値を設定します. レコード種別 0x04 が現れるまで, すなわち初期状態では 32 ビットアドレスの上位 16 ビットの値は 0 とします.

24.2　インテル HEX 形式変換ソフトウエア bin2hex

ファイルの内容をインテル HEX 形式で符号化するソフトウエア bin2hex を紹介します. このプログラムは標準 C 言語で書かれており, しかもコマンドライン上で動作するので様々な実行環境で動作可能です. 最初にこのソフトウエアの使い方を説明します. 以下の実行例では斜体字の部分が手入力する部分です. なお実行の際にはあらかじめプログラムへの実行パスが通っているようにして下さい.
例 1. ファイル bell.wav の 16 進ダンプをファイル bell.hex に書き込みます. ただしすでにファイル bell.hex が存在していたら実行を中断して強制終了します. これは重要なバイナリファイルを間違って上書きしないための機能です.

```
> bindump bell.wav bell.hex
```

ファイル bell.wav をインテル HEX 形式で画面に表示します.

```
> bin2hex bell.wav
```

例 2. ファイル bell.wav の 16 進ダンプをファイル bell.hex に書き込みます. ただしすでにファイル bell.txt が存在していたらそのファイルへ上書きします. この使い方をする場合には, 間違って上書きをして重要なバイナリファイルを失わないよう十分に気をつけて下さい.

```
> bindump -f bell.wav bell.hex
```

ソースコードを以下に示します. 入力のバイナリファイルを上述のインテル HEX 形式で表示するだけの簡単なプログラムです. プログラムは以下の方針で作成してありますので参考にして下さい.

- アドレスは 32 ビットで管理します.
- ファイルの先頭をアドレス 0 とします.

- ファイル内のデータは先頭から順番に, 16 バイトごとにインテル HEX 形式で出力します. ただしファイルの終端部では 16 バイト未満の場合があります.
- アドレスの下位 16 ビットの値が 0 の時には, レコード種別 0x04 の拡張線形アドレスにより上位 16 ビットの値をインテル HEX 形式で出力します.

ソースファイル番号 24.1: bin2hex.c

```
1  /*
2   * bin2hex, From a binary file to an Intel HEX file
3   * by h.kakugawa
4   */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <sys/stat.h>
9
10 #define IHEX_LINE_NBYTES   16
11
12 int   opt_overwrite = 0;
13 char *filename_bin = NULL;
14 char *filename_hex = NULL;
15
16 void read_args(char argc, char **argv);
17 void check_args(void);
18 void usage(void);
19 void bin2hex(void);
20 void emit_hex(FILE *fp_hex, FILE *fp_bin);
21
22 #if defined(_MSC_VER)
23 #  define FOPEN(p,f,m)  fopen_s(p, f, m)
24 #else
25 #  define FOPEN(p,f,m)  (((*(p)=fopen(f, m)) != NULL) ? 0 : 1)
26 #endif
27
28 int main(char argc, char **argv)
29 {
30   read_args(argc, argv);
31   check_args();
32   bin2hex();
33 }
34
35 void read_args(char argc, char **argv)
36 {
37   for (argc--, argv++; argc > 0; argc--, argv++) {
38     if (strcmp(*argv, "-f") == 0) {
39       opt_overwrite = 1;
40     } else {
41       if (filename_bin == NULL) {
42         filename_bin = *argv;
43       } else if (filename_hex == NULL) {
44         filename_hex = *argv;
45       } else {
46         usage();
47       }
48     }
49   }
50 }
51
52 void check_args(void)
53 {
54   if (filename_bin == NULL) {
55     printf("ERROR - NO FILENAME FOR BIN\n");
56     usage();
57   }
58   if (filename_hex == 0) {
59     printf("ERROR - NO FILENAME FOR HEX\n");
60     usage();
61   }
62   if (opt_overwrite == 0) {
```

```
63      struct stat fs;
64      if (stat(filename_hex, &fs) == 0) {
65        printf("ERROR - FILE EXISTS: %s\n", filename_hex);
66        exit(1);
67      }
68    }
69  }
70
71  void usage(void)
72  {
73    printf("hex2bin - Convert Binary file to Intel-HEX file\n");
74    printf("Usage: bin2hex [OPTION] FILE-BIN FILE-HEX\n");
75    printf("Option:\n");
76    printf("  -f     Force overwriting FILE-HEX\n");
77    printf("Examples:\n");
78    printf("  bin2hex xdata.bin xdata.hex\n");
79    exit(1);
80  }
81
82  void bin2hex(void)
83  {
84    FILE *fp_bin = NULL;
85    FILE *fp_hex = NULL;
86    if (FOPEN(&fp_bin, filename_bin, "rb") != 0) {
87      fprintf(stderr, "ERROR: CANNOT OPEN TO READ: %s\n",
88              filename_bin);
89      exit(1);
90    }
91    if (FOPEN(&fp_hex, filename_hex, "w") != 0) {
92      fprintf(stderr, "ERROR: CANNOT OPEN TO WRITE: %s\n",
93              filename_hex);
94      exit(1);
95    }
96    emit_hex(fp_hex, fp_bin);
97    fclose(fp_bin); fp_bin = NULL;
98    fclose(fp_hex); fp_hex = NULL;
99  }
100
101 void emit_hex(FILE *fp_hex, FILE *fp_bin)
102 {
103   int addr = 0;
104   int nbytes = 0;
105   int data[IHEX_LINE_NBYTES];
106   int done = 0;
107   int emit = 0;
108   int i = 0;
109   for (;;) {
110     if (emit == 1) {
111       emit = 0;
112       if (nbytes != 0) {
113         int chksum = 0;
114         if ((addr & 0xFFFF) == 0x0000) {
115           /* Emit extended lineat address (32-bit) */
116           int d2 = (addr >> 16) & 0xFF;
117           int d3 = (addr >> 24) & 0xFF;
118           chksum = 0x02 + 0x00 + 0x00 + 0x04 + d2 + d3;
119           fprintf(fp_hex,
120                   ":02000004%02X%02X%02X\n",
121                   d3, d2, (0x00 - chksum) & 0xFF);
122         }
123         /* Emit data, 16-bit addressing */
124         int a0 =  addr       & 0xFF;
125         int a1 = (addr >> 8) & 0xFF;
126         fprintf(fp_hex,
127                 ":%02X%02X%02X00", nbytes, a1, a0);
128         chksum = nbytes + a1 + a0 + 0x00;
129         for (i = 0; i < nbytes; i++) {
130           addr++;
131           fprintf(fp_hex,
132                   "%02X", (data[i] & 0xFF));
133           chksum += data[i];
134         }
```

```
135           nbytes = 0;
136           fprintf(fp_hex,
137                   "%02X\n", (0x00 - chksum) & 0xFF);
138         }
139       }
140       if (done == 1) {
141         /* Emit EOF */
142         fprintf(fp_hex, ":00000001FF\n");
143         break;
144       }
145       int d = fgetc(fp_bin);
146       if (d < 0) {
147         emit = 1;
148         done = 1;
149       } else {
150         data[nbytes] = d;
151         nbytes++;
152         if (nbytes == IHEX_LINE_NBYTES) {
153           emit = 1;
154         }
155       }
156     }
157   }
```

24.3　ファイル 16 進ダンプソフトウエア bindump

ファイルを 16 進数で表示するソフトウエア bindump を紹介します. このプログラムも標準 C 言語で書かれており, しかもコマンドライン上で動作するので様々な実行環境で動作可能です. 最初にこのソフトウエアの使い方を説明します.

例 1. ファイル bell.wav の 16 進ダンプを画面に表示します.

```
> ./bindump bell.wav
```

例 2. ファイル bell.wav の 16 進ダンプをファイル bell.txt に書き込みます.

```
> ./bindump bell.wav bell.txt
```

ソースコードを以下に示します. ファイルの内容を 1 バイト読んでは 16 進数で表示を行うことを繰り返す簡潔なプログラムです.

ソースファイル番号 24.2: bindump.c

```
1   /*
2    * bindump, Dump a binary file in hexadecimal
3    * by h.kakugawa
4    */
5   #include <stdio.h>
6   #include <stdlib.h>
7   #include <string.h>
8
9   char *filename_bin = NULL;
10  char *filename_txt = NULL;
11
12  void read_args(char argc, char **argv);
13  void check_args(void);
14  void verbose(void);
15  void dumpbin(void);
16  void emit(FILE *fp_txt, FILE *fp_bin);
17  void usage(void);
18
19  #if defined(_MSC_VER)
20  #  define FOPEN(p,f,m)  fopen_s(p, f, m)
21  #else
22  #  define FOPEN(p,f,m)  (((*(p)=fopen(f, m)) != NULL) ? 0 : 1)
23  #endif
```

```
24
25  int main(char argc, char **argv)
26  {
27    read_args(argc, argv);
28    check_args();
29    dumpbin();
30  }
31
32  void read_args(char argc, char **argv)
33  {
34    for (argc--, argv++; argc > 0; argc--, argv++) {
35      if (filename_bin == NULL) {
36        filename_bin = *argv;
37      } else if (filename_txt == NULL) {
38        filename_txt = *argv;
39      }
40    }
41  }
42
43  void check_args(void)
44  {
45    if (filename_bin == NULL) {
46      printf("ERROR: NO FILENAME FOR BIN\n");
47      usage();
48    }
49  }
50
51  void usage(void)
52  {
53    printf("bindump - Dump Binary file in hexadecimal code\n");
54    printf("Usage: bindump FILENAME-IN [FILENAME-OUT]\n");
55    printf("Examples:\n");
56    printf("  bindump xdata.bin\n");
57    printf("  bindump xdata.bin xdata.txt\n");
58    exit(1);
59  }
60
61  void dumpbin(void)
62  {
63    FILE *fp_bin = NULL;
64    FILE *fp_txt = NULL;
65    if (FOPEN(&fp_bin, filename_bin, "rb") != 0) {
66      fprintf(stderr, "ERROR: CANNOT OPEN TO READ: %s\n", filename_bin);
67      exit(1);
68    }
69    if (filename_txt != NULL) {
70      if (FOPEN(&fp_txt, filename_txt, "w") != 0) {
71        fprintf(stderr, "ERROR: CANNOT OPEN TO WRITE: %s\n", filename_txt);
72        exit(1);
73      }
74    } else {
75      fp_txt = stdout;
76    }
77    emit(fp_txt, fp_bin);
78    fclose(fp_bin);
79    fp_bin = NULL;
80    if (filename_txt != NULL) {
81      fclose(fp_txt);
82    }
83    fp_txt = NULL;
84  }
85
86  void emit(FILE *fp_txt, FILE *fp_bin)
87  {
88    int addr;
89    int d;
90    int a8 = 0;
91    for (addr = 0; (d = fgetc(fp_bin)) != EOF; addr++) {
92      a8 = addr & 0xFF;
93      if (a8 == 0x00) {
94        if (addr != 0) {
95          fprintf(fp_txt, "\n");
```

```
96        }
97        fprintf(fp_txt, "           ");
98        fprintf(fp_txt, "+0 +1 +2 +3 +4 +5 +6 +7 ");
99        fprintf(fp_txt, "+8 +9 +A +B +C +D +E +F\n");
100     }
101     if ((a8 & 0x0F) == 0x00) {
102       fprintf(fp_txt, "%08X: ", addr);
103     }
104     fprintf(fp_txt, "%02X ", d);
105     if ((a8 & 0x0F) == 0x0F) {
106       fprintf(fp_txt, "\n");
107     }
108   }
109   if ((a8 & 0x0F) != 0x0F) {
110     fprintf(fp_txt, "\n");
111   }
112 }
```

24.4　各種オペレーティングシステムでのビルト方法

Linux や FreeBSD などの Unix 系オペレーティングシステムおよび Windows オペレーティングシステムにおいて bin2hex.c および bindump.c をビルトする方法を説明します．特にここでは GCC (GNU Compiler Collection) と make を用いた方法を説明します．C コンパイラは GCC の他にも幾つかありますので, GCC 以外のものを利用する場合は各自で工夫して下さい．

- C 言語処理系である C コンパイラがインストールされていることを確認して下さい．もしインストールされていなければ, フリーソフトウエアの C コンパイラをインストールすると良いでしょう．Linux や FreeBSD などの Unix 系オペレーティングシステムでは, 例えば GCC が挙げられます[*1]．Windows オペレーティングシステムでは, GCC が含まれている MinGW (Minimalist GNU for Windows) が挙げられます[*2]．それぞれバイナリパッケージが配布されていますので, オペレーティングシステムごとでのインストール方法に従ってインストールして下さい．
- ビルド自動化ツールである make がインストールされていることを確認して下さい．もしインストールされていなければ, フリーソフトウエアの make をインストールすると良いでしょう．Linux や FreeBSD などの Unix 系オペレーティングシステムでは, 例えば GNU make が挙げられます[*3]．Windows オペレーティングシステムでは, MinGW に GNU make が含まれていますので, MinGW のインストールの際に GCC と同時にインストール指定すると良いでしょう．

以下では GCC と make を使用したコンパイル方法を示します．ビルド自動化ツール make のためのコンパイル方法を記述するファイルが `Makefile` と呼ばれるファイルです．`Makefile` の具体的な内容はすぐこの後に示しますが, いまは仮に `Makefile` が用意できたものとして, 以下の通り斜体字の部分を手入力して make コマンドを実行すると, 自動的にビルドが行われます．なお Windows オペレーティングシステムの場合には, コマンド名は make ではなく mingw32-make となります．

```
> make                    — make コマンドを実行
gcc  -o bin2hex bin2hex.c    — 以下 make コマンドの実行内容
gcc  -o bindump bindump.c
```

[*1] Free Software Foundation, Inc., “GCC, the GNU Compiler Collection,” https://gcc.gnu.org/, (2018 年 6 月 2 日閲覧).
[*2] MinGW.org, “Home of the MinGW and MSYS Projects,” http://www.mingw.org/ (2018 年 6 月 2 日閲覧).
[*3] Free Software Foundation, Inc., “GNU Make,” https://www.gnu.org/software/make/, (2018 年 6 月 2 日閲覧).

ビルドのための Makefile の内容を以下に示します. この Makefile は Unix 系オペレーティングシステムと Windows オペレーティングシステムのどちらでも make によるビルドが可能なように書き方を工夫しています.

ソースファイル番号 24.3: Makefile

```
ifeq ($(OS), Windows_NT)
  EXEEXT = .exe
  CC = gcc
  RM = cmd.exe /C del
else
  EXEEXT =
  CC = gcc
  RM = rm -f
endif

PROGRAM_bin2hex=bin2hex$(EXEEXT)
PROGRAM_bindump=bindump$(EXEEXT)

.PHONY: all clean distclean
all: $(PROGRAM_bin2hex) $(PROGRAM_bindump)
clean:
  -$(RM) $(PROGRAM_bin2hex) $(PROGRAM_bindump)
distclean:
  -$(RM) *.o core *.core *~

$(PROGRAM_bin2hex): bin2hex.c
  $(CC) -o $(PROGRAM_bin2hex) bin2hex.c
$(PROGRAM_bindump): bindump.c
  $(CC) -o $(PROGRAM_bindump) bindump.c
```

課題

1. 本文中で例として示したインテル HEX 形式のファイルを復号化すると, 本文中で示した 16 進ダンプリストに一致することを示して下さい.
2. 本文中で例として示したインテル HEX ファイルのチェックサムが正しいか, 実際に計算して示して下さい.
3. 非表示文字を含んでいるファイルを表示文字だけからなるテキストファイルに符号化する方法には, インテル HEX 形式の他にモトローラ S 形式 (Motorola S-format) が知られています. モトローラ S 形式を調査して, 説明して下さい.
4. ファイルをモトローラ S 形式に符号化するソフトウエアを作成して下さい.

第 25 章

RS-232C 接続
EEPROM 書き込み器

本章では I^{2}C EEPROM 書き込み器を紹介します. 図 25.2 に示すように, 本装置はパーソナルコンピュータと RS-232C を介して接続し, パーソナルコンピュータから受信したインテル HEX 形式のファイルを EEPROM に書き込みます. 本書では EEPROM の中でも特に I^{2}C 通信でアクセスするメモリを対象とし, 例えば Microchip Technology 社の 24FC1025 や 24FC64 などが挙げられます. 第 27 章では 24FC1025 に PCM サウンドデータを, 第 30 章では 24FC64 に自動運転スクリプトを記憶させてカスタマイズを行います. 従って本装置は他装置のカスタマイズで使用するものですから, 本装置は開発環境の一部をなすものと考えて下さい.

本章で紹介する回路とファームウエアの特色は以下の通りです.

- 様々な I^{2}C EEPROM をサポート

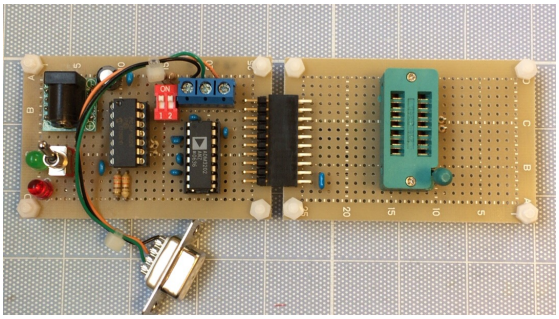


図 25.1: EEPROM 書き込み器

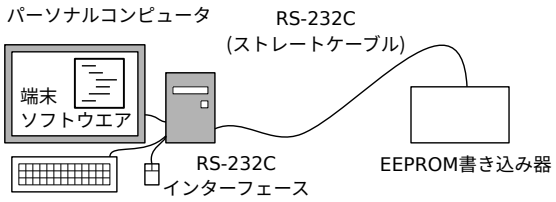


図 25.2: 接続方法



図 25.3: 端末ソフトウエア gtkterm を通して使用中の様子 (Ubuntu 上)

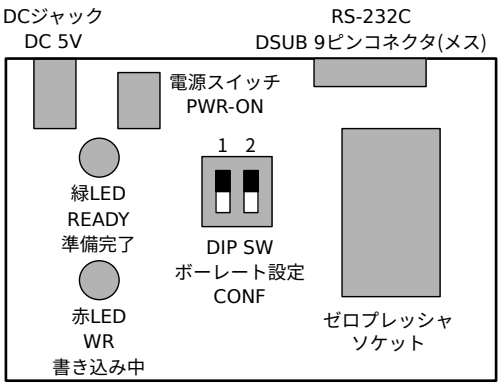


図 25.4: 操作表示部品の配置例と名称

Microchip Technology 社 24FC1025, 24FC512, 24FC256, 24FC64 などの読み書きが行えます.

- 汎用的な RS-232C を通信インターフェースに使用
  十分に枯れた技術の RS-232C 通信を使用しますから, 永きに渡って様々なパーソナルコンピュータから使用可能です.
- 拡張可能なファームウエア構造
  簡単なファームウエアの変更で別の EEPROM にも使用可能です.

本章で学ぶ内容は以下の通りです.

- RS-232C
- PIC マイコンの非同期通信モジュール EUSART
- 巡回バッファ構造による入出力バッファリング
- I^2C EEPROM 24FC64・24FC1025 等の使用方法

本ファームウエアは機能の割には規模が大きいのは, 人間が対話的な操作をして本装置を使用することに起因します. 使いやすいユーザーインターフェースの実現にはどうしてもプログラム量は増加してしまいます.

25.1　使用方法

本装置の使用方法を以下に説明します. 以下の説明で使用する部品名称は図 25.4 に配置例とともに示しています. まず以下のものを用意して下さい.

- RS-232C ケーブル
  パーソナルコンピュータ側の RS-232C インターフェースと本装置との接続に使用するケーブルです. 図 25.5a に例を示します. パーソナルコンピュータ側が 9 ピン・オスの場合には 9 ピン・メス–9 ピン・オスのストレートケーブルを使用する場合がほとんどと思いますが, 使用する RS-232C インターフェースを十分に調べた上で適合するケーブルを用意して下さい. 典型的な接続方法の例を図 25.6 に示しますので参考にして下さい.
  コネクタには D サブ 9 ピンや D サブ 25 ピンのほかにオス・メスの違いがあり, さらにストレートケーブルやクロスケーブルなど多くのバリエーションがありますので注意が必要です. 詳しくは後述します.
- RS232C インターフェースを有するパーソナルコンピュータ
  RS-232C インターフェースがない場合は, 図 25.5b に示すような USB ポートへ接続可能な RS-232C ア



(a) RS-232C ケーブル



(b) USB ポート接続型 RS-232C アダプタ

図 25.5: RS-232C 接続で使用する品

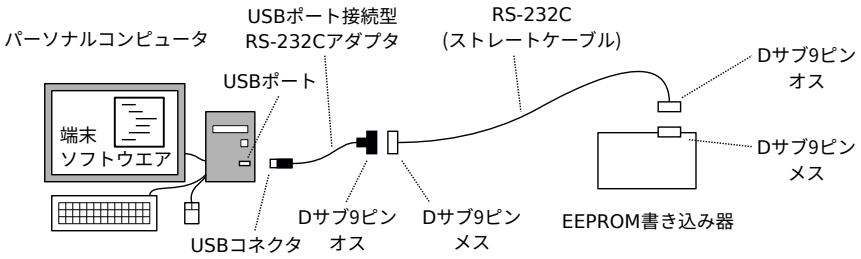


図 25.6: 本装置の一般的な接続方法 (このように並べた後に嵌合(かんごう))

ダプタを用意して下さい. 「USB シリアル変換ケーブル」あるいは「USB-RS232C 変換アダプタ」などの名称で販売されています.

また, オペレーティングシステムは特には問いませんが, 次に説明する端末ソフトウエアが利用可能であることが必須です.

- パーソナルコンピュータで動作する端末ソフトウエア

  RS232C インターフェースを介して通信が可能で, 以下の機能を有するものを用意して下さい.
  - 通信速度は 4800, 9600, 19200, 38400bps のいずれかが利用できるもの.
  - ファイルのアップロード機能を有するもの.
  - フロー制御に Xon/Xoff を使用できるもの. 端末ソフトウエアによっては「ソフトウエア制御」と表記している場合があります.

本装置の起動までの手順を以下に説明します.

1. 図 25.2 に示すように接続して下さい.
2. 電源スイッチを切った状態で DC ジャックに 5V の AC アダプタを接続します. **間違って 12V 出力の AC アダプタを使用すると回路が破損する場合がありますので, 十分に注意して下さい.**
3. RS-232C ケーブルを介して本装置とパーソナルコンピュータを接続します.
4. DIP スイッチ CONF を適切な通信速度の設定にします. 問題がない限りは最高速の 38400bps にすると良いでしょう.
5. パーソナルコンピュータ上で端末ソフトウエアを起動して, RS-232C 通信パラメータの設定を行います.
   - インターフェース: 使用している RS-232C インターフェースを選択します.

表 25.1: DIP スイッチによる通信速度の設定

DIP スイッチの設定		通信速度
1	2	
OFF	OFF	4800bps
OFF	ON	9600bps
ON	OFF	19200bps
ON	ON	38400bps

表 25.2: EEPROM 書き込み器コマンド一覧

コマンド	動作内容
U	EEPROM デバイス選択に戻る
W	書き込み
R	読み出し
V	ベリファイ
D	ダンプ
E	消去
P	EEPROM デバイス情報の表示

表 25.3: サポート済み EEPROM

EEPROM デバイス	メモリ容量		通信速度	ページサイズ
24FC1025	1Mbit	(128Kbyte)	1Mbps	128
24LC1025	1Mbit	(128Kbyte)	400kbps	128
24FC512	512Kbit	(64Kbyte)	1Mbps	128
24LC512	512Kbit	(64Kbyte)	400Kbps	128
24FC256	256Kbit	(32Kbyte)	1Mbps	64
24LC256	256Kbit	(32Kbyte)	400Kbps	64
24FC64	64Kbit	(8Kbyte)	1Mbps	32
24LC64	64Kbit	(8Kbyte)	400Kbps	32

- 通信速度: 本装置の設定に合わせます.
- 1 文字あたりのデータビット数: 8
- ストップビット数: 1
- パリティ: なし
- フロー制御: Xon/Xoff (端末ソフトウエアによっては「ソフトウエア制御」と表記しています.)
- 改行文字: CRLF

6. 本装置の電源スイッチを入れます. 端末ソフトウエアの画面上に起動メッセージが表示されます. もし表示されない, あるいは文字化けらしき表示となる場合は, 以下のことが原因と考えられます.
   - 端末ソフトウエアでのパラメータの設定が正しくない.
   - RS-232C ケーブルが正しくない. ストレートケーブルとクロスケーブルを間違えている可能性もあります.
   - 装置の組み立てを間違えている.

   使用する端末ソフトウエアによっては, インテル HEX ファイルの書き込みの途中で通信が停止する問題が発生するかもしれません. その場合には, 端末ソフトウエアにおいて行末時の遅延時間を設定をしてみて下さい. 実際に Linux 上での gtkterm ではこの現象が発生しましたが, 遅延時間の値を 10 ミリ秒に設定すれば通信は停止することなくなりました. なお遅延時間の値を大きくするほど送信に要する時間は長くなりますから, できるだけ小さな値にすると良いでしょう.

以上の準備の後は, パーソナルコンピュータで動作している端末ソフトウエアを使用して RS-232C を介して

本装置と対話的に EEPROM への書き込みを行います. コマンド一覧をまとめたものを表 25.2 に示します. サポート済みの EEPROM の一覧が表 25.3 で, いずれも Microchip Technology 社の製品です.

- 起動メッセージ
  以下のように起動メッセージとともにサポートされている EEPROM の一覧が表示されます.

```
=============
EEPROM WRITER
=============

DEVICE LIST:
  (1) 24FC1025  (2) 24LC1025  (3) 24FC512  (4) 24LC512
  (5) 24FC256   (6) 24LC256  (7) 24FC64  (8) 24LC64
SELECT A DEVICE FROM LIST:
```

- EEPROM の選択
  一覧の中から使用する EEPROM の番号を入力します. 例えば 24FC64 を使用する場合は 7 を入力します. その後は現在選択している EEPROM がプロンプトに表示されます.

```
SELECT A DEVICE FROM LIST: 7
[24FC64]
```

- 書き込み (Write)
  W コマンドにより, パーソナルコンピュータにあるインテル HEX ファイルを EEPROM に書き込みます.

```
[24FC64] W
```

  端末ソフトウエアのアップロード機能を使用してインテル HEX ファイルを本装置へアップロード (送信) します. 書き込みが終了すると COMPLETED と表示されます.

```
UPLOAD INTEL-HEX FILE TO WRITE
WRITING.
COMPLETED
```

- ベリファイ (Verify)
  V コマンドにより, 書き込んだ内容が正しいか否かをベリファイ (照合) します. 端末ソフトウエアより再びインテル HEX ファイルをアップロードします. EEPROM の内容とアップロードの内容が一致すれば COMPLETED と表示されます.

```
[24FC64] V
UPLOAD INTEL-HEX FILE TO VERIFY
VERIFYING.
COMPLETED
```

- ダンプ (Dump)
  D コマンドにより, EEPROM の内容を 256 バイトごとに 16 進ダンプします. 256 バイトを 16 進ダンプすると, 先のアドレスに進むか終了するかが尋ねられます. スペースキーを押すと 16 進ダンプを続けます. Q を押すと終了です.

```
[24FC64] D
          +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00000000: 23 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D
00000010: 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 0A 23 20
   ...(略)...
000000E0: 9C E8 BB 8A 28 E6 9D B1 E5 90 91 29 0A 23 20 E5
000000F0: 88 9D E6 9C 9F E5 8C 96 0A 20 20 44 20 31 20 31
--MORE-- (HIT SPACE KEY TO CONTINUE; 'Q' TO EXIT)
```

- 読み出し (Read)

  R コマンドにより, EEPROM の内容をインテル HEX フォーマットで表示します. 端末ソフトウエアのダウンロード機能と組み合わせると, EEPROM の内容をパーソナルコンピュータのファイルに保存できます. インテル HEX フォーマットのファイル終端 (レコード種別 0x01), すなわち EEPROM の末尾に来ると, 表示を停止して何かキーが押されるまで待機が続きます. 端末ソフトウエアでダウンロード終了の指示を終えたら何かキーを押して下さい. すると COMPLETED と表示されコマンドの実行は終了します.

```
[24FC64] R
READY? (Y/N)
:0200000400000FA
:10000000232D2D2D2D2D2D2D2D2D2D2D2D2D2D2A
:100010002D2D2D2D2D2D2D2D2D2D2D2D0A23204A
   ...(略)...
:101FE000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF01
:101FF000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF1
:00000001FF
COMPLETED
```

- EEPROM 情報 (Print EEPROM Infomation)

  P コマンドにより, 選択している EEPROM の情報を表示します.

```
[24FC64] P
MICROCHIP TECHNOLOGY, I2C, 64KBIT (8KBYTE)
```

- 消去 (Erase)

  E コマンドにより, EEPROM の内容をすべて 0 にします. 進み具合が分かるように進行経過が随時表示されます.

```
[24FC64] E
ERASE MEMORY? (Y/N) y
ERASING MEMORY...
  000000 000200 000400 000600 000800 000A00 000C00 000E00
  001000 001200 001400 001600 001800 001A00 001C00 001E00
COMPLETED
```

- 選択解除 (Unselect)

  U コマンドにより, EEPROM の選択を解除して別の EEPROM の選択を行います.

```
[24FC64] U

DEVICE LIST:
  (1) 24FC1025  (2) 24LC1025  (3) 24FC512  (4) 24LC512
  (5) 24FC256   (6) 24LC256  (7) 24FC64  (8) 24LC64
SELECT A DEVICE FROM LIST:
```

- その他: コマンドでない文字
  コマンド一覧表が表示されます.

```
[24FC64]
COMMAND LIST:
  U - UNSELECT DEVICE
  W - WRITE (UPLOAD INTEL-HEX FILE)
  R - READ (DOWNLOAD INTEL-HEX FILE)
  V - VERIFY (COMPARE MEMORY AND INTEL-HEX FILE)
  D - DUMP (PRINT MEMORY CONTENTS)
  E - ERASE (BLANK MEMORY CONTENTS)
  P - PRINT DEVICE DESCRPTION
```

25.2　回路図

EEPROM メモリ書き込み器の回路図を図 25.7 に, 部品リストを表 25.4 に, それぞれ示します. ADM3202AN は RS-232C での電圧レベルと 5V ロジックでの電圧レベルとの間の変換を行います. I^2C バスは SCL と SDA の 2 本の通信線で構成され, それぞれ抵抗器によるプルアップが必要です. 試作例では本体とゼロプレッシャソケットとを部品リスト外のコネクタ等で抜き差し可能にして, ゼロプレッシャソケット部分を交換可能にしています. 本装置を拡張して, ピン形状の違うメモリや SPI EEPROM の書き込みを行う場合に備えています.

RS-232C 用の D サブ 9 ピンコネクタにはハンダ付け型のものと基板取り付け型の 2 種類があります. 基板取り付け型のピン配置は 2.54mm ピッチのグリッド状ではないため, そのままではユニバーサル基板には入りません. 部品リスト外のピッチ変換基板を使用するか, 図 25.1 の試作例のように D サブ 9 ピンコネクタと ADM3202AN との接続にターミナルブロックを用いるとよいでしょう.

USB ポートを利用した RS-232C アダプタ製品ではコネクタが D サブ 9 ピンのオスの製品がほとんどのようですから, RS-232C アダプタ製品と本装置の接続には D サブ 9P オス・D サブ 9 メスのストレートケーブルを使用します. あるいはケーブルを用意せずとも RS-232C アダプタ製品のコネクタを本装置のコネクタに直接接続することも可能です.

表 25.4: 【部品表】 EEPROM メモリ書き込み器

記号	数量	部品名
U2	1	PIC 16F18326-I/P [I-11886]
U1	1	ADM3202AN [I-09001]
D1	1	LED 緑 [I-06413]
D2	1	LED 赤 [I-04780]
C7	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]

(つづく)

(つづき)

記号	数量	部品名
C1, C2, C3, C4, C5, C6, C8		
	7	セラミックコンデンサ 0.1μF [P-00090]
R3, R4	2	抵抗 1kΩ　1/4W [R-25102]
R5, R6	2	抵抗 2kΩ　1/4W [R-25202]
R1, R2	2	抵抗 10kΩ　1/4W [R-25103]
X1	1	ゼロプレッシャ IC ソケット (ZIF) 14 ピン　300mil 幅 [P-12073]
	1	IC ソケット 16 ピン [P-00029]
	1	IC ソケット 14 ピン [P-00028]
SW2	1	単極双投トグル SW　On-On 型 [P-02399]
SW1	1	DIP SW 2P [P-08922]
J1	1	D サブ 9 ピンコネクタ・メス [C-00645]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか接続用ケーブル・基板・配線材等

25.3　非同期シリアル通信 RS-232C

IndexTermENRS-232C とは, 送信線 TxD と受信線 RxD の 2 線を使用した非同期的シリアル通信のひとつです. ノイズには比較的強く, 通信可能な距離は 15 メートル程度までです. TxD と RxD それぞれは単方向の通信ですが, 別々の通信線ですので同時に送受信が可能です. 通信速度は主に 300bps から 115.2kbps 程度の範囲で使用されます. クロック信号線を使用しない非同期通信ですから, 送信側と受信側で予め互いに通信速度を合意して同一の通信速度に設定せねばなりません. 本書では一般的に広く呼称されている RS-232C の呼称を使用しますが, 現在の規格の正式名称は ANSI/EIA-232-E です.

RS-232C で通信を行う際には通信速度以外にも各種のパラメータがあります. 送信側と受信側は, 通信速度だけでなくこれらパラメータも予め互いに合意して設定する必要があります. もしパラメータに不一致があれば正しく通信できません.

- 通信速度 (ボーレート)
  通信の速度を指定します. 主に 300bps から 115.2kbps 程度の範囲です.
- 1 文字あたりのデータビット数
  1 文字を通信するのに使用するビット数を指定します. 5, 6, 7, 8 ビットから選択します.
- ストップビット長
  1 文字のデータの終端を表すビットの長さを指定します. 1, 1.5, 2 ビットから選択します.
- パリティ
  エラー検査に使用するパリティの決定方法を指定します. 偶数 (even), 奇数 (odd), なし (none) から選択します.
- フロー制御
  受信バッファの溢れを防ぐ方法を指定します. RTS/CTS, Xon/Xoff, なし (none) から指定します. RTS/CTS はハードウエアによるフロー制御方式で, Xon/Xoff はソフトウエアによるフロー制御方式

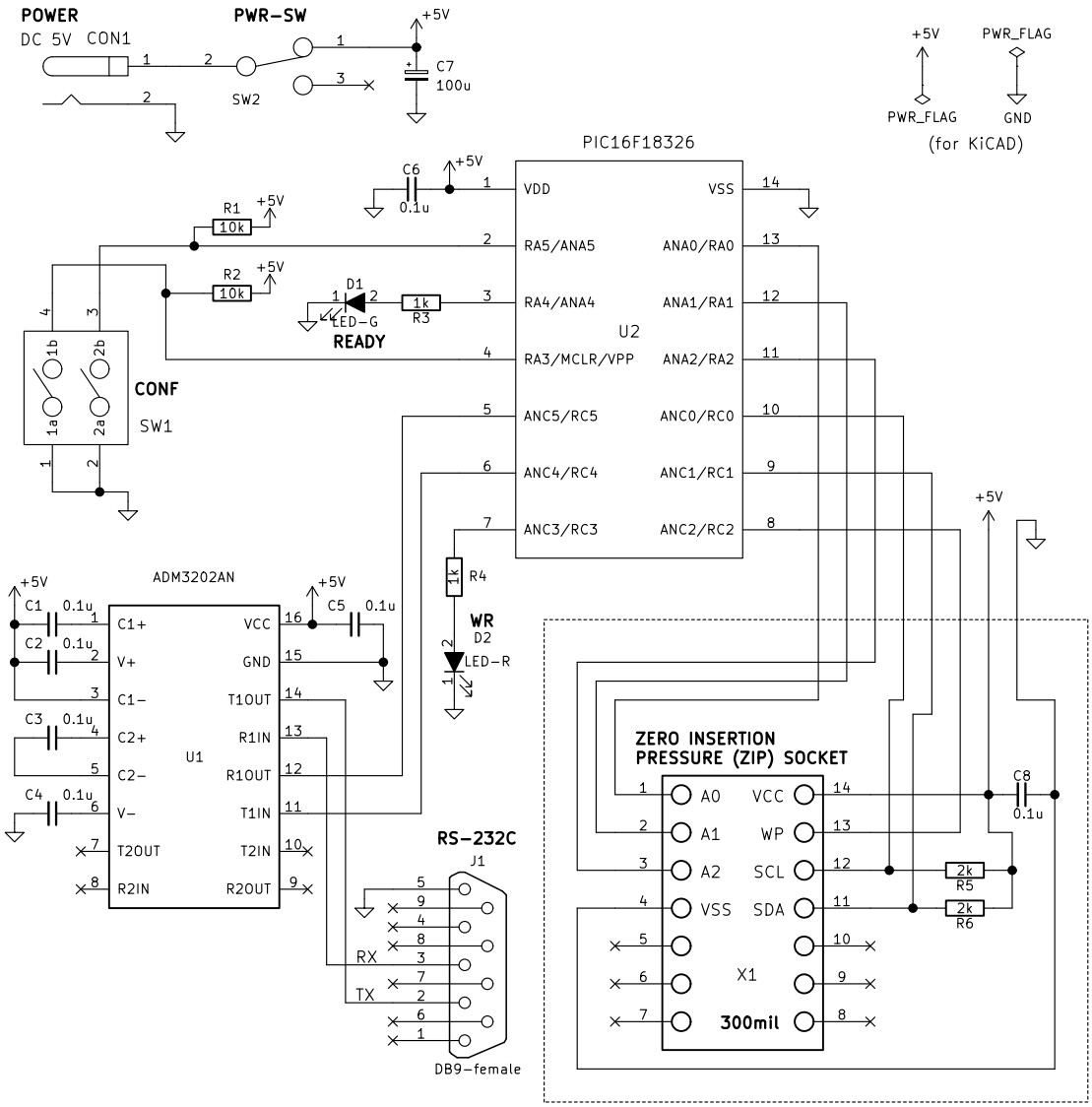


図 25.7: 【回路図】 EEPROM メモリ書き込み器

です.

RS-232C は電話回線とモデムを通じて大型コンピュータに端末機を接続するために考案されたことに源を発するもので, 2 つの装置間での通信を行うものです. この歴史的背景より RS-232C 通信機器は以下の 2 種類に大別されます.

- DTE (データ端末装置; Data Terminal Equipment)
  パーソナルコンピュータなどの装置が該当します.
- DCE (データ回線終端装置; Data Circuit-Terminating Equipment)
  モデムなどの装置が該当します.

RS-232C 用のコネクタには 25 ピンあるいは 9 ピンの D サブコネクタが広く使用されています. コネクタにはピン数の他にオス・メスの区別があります. DTE をオス, DCE をメスのコネクタを使用するのが一般的です



(a) オス　　**(b)** メス

図 25.8: D サブ 9 ピンコネクタの配列 (正面より)

表 25.5: RS-232C での一般的なピン番号と信号名 (DTE 側: オス, DCE 側: メス; D サブ 9 ピン)

ピン番号	信号名	信号の方向 DTE	信号の方向 DCE	説明
1	DCD	入力	出力	キャリア検出 (Data Carrier Detect)
2	RxD	入力	出力	受信データ (Received Data)
3	TxD	出力	入力	送信データ (Transmitted Data)
4	DTR	出力	入力	データ端末レディ (Data Terminal Ready)
5	GND	–	–	グランド (Commpn Ground)
6	DSR	入力	出力	データセットレディ (Data Set Ready)
7	RTS	出力	入力	送信リクエスト (Request To Send)
8	CTS	入力	出力	送信可 (Cleat To Send)
9	RI	入力	出力	被呼表示 (Ring Indicator)

が, 厳密に決まっているわけでないので接続の際のコネクタ選びには注意して下さい. パーソナルコンピュータでは D サブ 9 ピンコネクタがよく使用されており, 規格 EIA-574 が制定されています. D サブ 9 ピンコネクタのピン配列を図 25.8 に示し, ピン番号と信号名の一般的な対応を表 25.5 に示します. この対応はあくまでも一般的なものであり, 機器により異なる場合があるので十分な注意が必要です. これら図と表で重要なのは RxD, TxD, GND の 3 線で, これらがあれば最低限の双方向通信が可能です. ただしフロー制御にはソフトウエアによる制御方式である Xon/Xoff のみが使用でき, ハードウエアによる制御方式である RTS/CTS は使用できません.

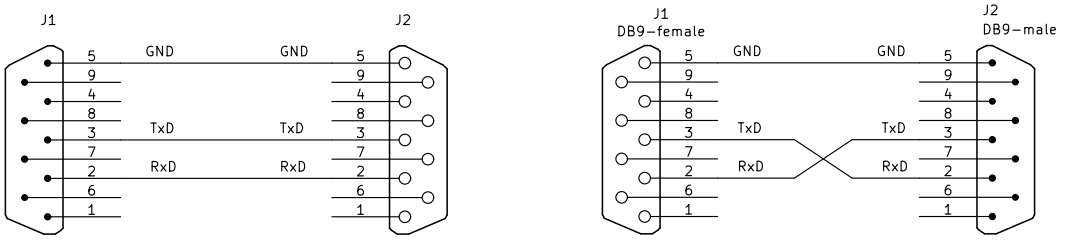
DTE と DCE を接続する際にはストレートケーブルと呼ばれる RS-232C ケーブルを使用します. これはケーブルの両端のコネクタの同じピンを互いに接続したケーブルで, 図 25.9a に示す結線がされています. 表 25.5 から分かる通り, 互いに対応する信号が DTE と DCE とで対をなしています.

DTE 同士での接続も可能ですが, その際には表 25.5 から分かる通り信号線が衝突を起こします. これを避けるために配線を入れ替えたクロスケーブルと呼ばれる RS-232C ケーブルを使用します. クロスケーブルでは図 25.9b に示す通り TxD と RxD をたすき掛けに接続して信号線の衝突を避けます. このように DTE と DCE を対として接続する必要は必ずしもなく, クロスケーブルを用いることで DTE 同士あるいは DCE 同士の接続が可能です.

25.4　EUSART: 非同期シリアル通信モジュール

PIC マイコンでは内蔵周辺モジュール EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transmitter; 強化型ユニバーサル同期非同期送受信) を使用して RS-232C 通信を行います. 16F18326 にはひとつの EUSART モジュールが内蔵されており, EUSART1 と名付けられています. EUSART1 で使用する主なレジスタやビットを以下に概説します.

- `TX1STA.TXEN`



(a) ストレートケーブル (DTE–DCE 間)　　**(b)** クロスケーブル (DTE–DTE 間)

図 25.9: 【回路図】 RS-232C ケーブル (D サブ 9 ピンコネクタ)

トランスミッタのオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.

- TX1STA.SYNC
  動作モードを設定するビットです. 1 の場合は同期モードで, 0 の場合は非同期モードです.
- TX1STA.BRGH
  高速ボーレートを有効にするビットで, EUSART モードが非同期の場合にのみ有効です. 1 の場合は高速モードで, 0 の場合は低速モードです.
- RC1STA.SPEN
  シリアルポートのオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.
- RC1STA.CREN
  連続受信のオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.
- RC1STA.OERR
  オーバーランエラー状態を保持するビットです 1 の場合はオーバーランエラーありで, 0 の場合はなしです.
- BAUD1CON.BRG16
  ボーレートジェネレータの設定ビットです. 1 の場合は 16 ビットボーレートジェネレータを, 0 の場合は 8 ビットボーレートジェネレータを使用します.
- RC1REG
  受信したデータを保持する 8 ビットのレジスタです.
- TX1REG
  送信するデータを書き込む 8 ビットのレジスタです.
- SP1BRGL
  通信速度を設定するレジスタです. ボーレートの下位 8 ビットを保持します.
- SP1BRGL, SP1BRGH
  通信速度を設定するレジスタです. ボーレートの上位 8 ビットを保持します.
- PIE1.TXIE
  送信完了割り込み許可ビットです.
- PIR1.TXIF
  送信完了割り込み発生フラグです.
- PIE1.RCIE
  受信完了割り込み許可ビットです.
- PIR1.RCIF

表 25.6: ボーレート決定の計算式 (Fosc はクロック周波数, n は SP1BRGH:SP1BRGH の値)

設定ビット		モード	ボーレート計算式
BRG16	BRGH		
0	0	8 ビット	Fosc/[64 (n+1)]
0	1	8 ビット	Fosc/[16 (n+1)]
1	0	16 ビット	Fosc/[16 (n+1)]
1	1	16 ビット	Fosc/[4 (n+1)]

受信完了割り込み発生フラグです.

非同期モード時でのボーレート決定の計算式を表 25.6 に示します. 本ファームウエアでは Fosc=32MHz, BRG16=1, BRGH=1 の設定で使用します. この設定時において例えば 19200bps で通信をする際の設定は SP1BRGH=0x01, SP1BRGH=0xA0 となります. この設定値の計算は以下の通りです.

$$
\begin{aligned}
\text{bps} &= \text{Fosc}/\{4 \times (\text{SP1BRGH} : \text{SP1BRGL} + 1)\} \\
\text{SP1BRGH} : \text{SP1BRGL} &= \{\text{Fosc}/(4 \times \text{bps})\} - 1 \\
&= \{32 \times 10^6/(4 \times 19200)\} - 1 \\
&= 415.6667 \\
&\approx 416 \\
&= 256 + 160 \\
&= 0\text{x}01 \times 256 + 0\text{xA}0
\end{aligned}
$$

25.5　巡回バッファ構造による入出力バッファリング

バッファリング (buffering) とは, 入出力のデータを一時的に蓄えておく手法です. バッファリングのためのデータ構造をバッファ (buffer) と呼んでいます. バッファリングは割り込み処理と組み合わせて, 特に入出力データの発生がまばらな場合や実時間性を要する場合に大きく威力を発揮します.

- 受信バッファリング
  データ到着の割り込みが発生すれば, 割り込みハンドラ内で直ちに受信バッファに蓄えます. 他の処理の最中にデータが到着しても直ちには受信データの処理を行えないからです. 更に続けて受信データが到着する場合もあるため, 受信バッファは十分な容量を準備する必要があります. 受信バッファが一杯のときにデータが到着すれば破棄するしかありません.
- 送信バッファリング
  送信するデータはいったん送信バッファリングに蓄えて, すぐに他の処理に取り掛かります. そして送信完了の割り込みが発生すれば, 割り込みハンドラは送信バッファ中の次の送信データの送信を開始します. 送信バッファも十分な容量を準備する必要があります. 送信バッファが一杯のときに更にデータを送信しようとすれば, 実時間性をあきらめて送信完了まで待機するかあるいは送信データを破棄するしかありません.

巡回バッファ (cyclic buffer) とは, バッファを実現するデータ構造のひとつです. 巡回バッファに対する主要な操作は以下の 2 つです.

- put()

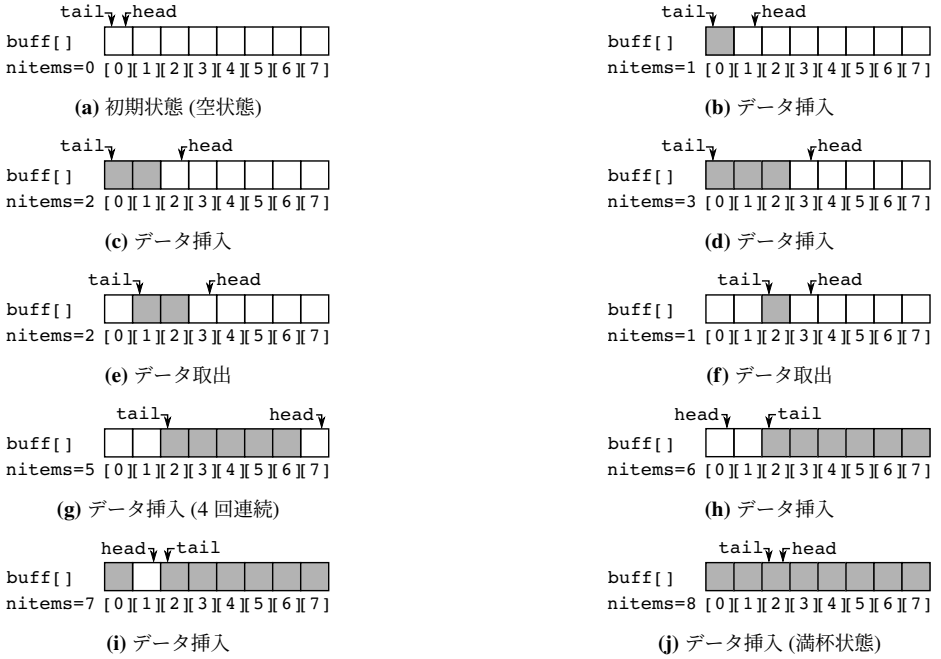


図 25.10: 巡回バッファによるバッファリング

データをバッファに挿入します.

- get()
  バッファからデータを取り出します.

巡回バッファで使用する主要な変数は以下の通りです.

- buff[]
  バッファ本体で, データを記憶する配列変数です.
- nitems
  バッファに記憶してあるデータ総数を保持する整数変数です.
- head
  バッファ本体 buff[] での次の挿入場所の添字を保持する整数変数です. ただし nitems がバッファサイズに等しい場合は除外します.
- tail
  バッファ本体 buff[] での次の取出場所の添字を保持する整数変数です. ただし nitems が 0 に等しい場合は除外します.

図 25.10 に巡回バッファによるバッファリングの概念を示します. この図では説明の簡単化のために配列の要素数を 8 としています. バッファへのデータの挿入は buff[head] にデータを書き込んで head を 1 増加します. ただし head が添字の範囲を越えたなら 0 に戻します. バッファからのデータの取出は buff[tail] のデータを読み出して tail を 1 増加します. ただし tail が添字の範囲を越えたなら 0 に戻します. このように head と tail がクルクルと回りながら配列を順次アクセスすることでバッファリングが実現されます.

25.6　I^{2}C EEPROM 24FC64 と 24FC1025

25.6.1　24FC64

I^{2}C EEPROM 24FC64 の通信プロトコルを概説します. 24FC64 は 8K バイトの容量を持つ EEPROM で, 不揮発性であることから各種の設定データの保持をするのに便利です. 24FC64 と同じメモリ容量の EEPROM に 24LC64 がありますが, 通信速度の上限が異なるものの通信プロトコルは同一です.

- デバイスには図 3.29 に示す通り 3 本のチップ選択ピンがあります. このピンをハードウエア的に設定することで 1 つの I^{2}C バスに最大 8 (= 2^3) 個の 24FC64 を同時接続できます.
- 各デバイスの 7 ビットの I^{2}C アドレスは図 25.11a に示す通り 4 ビットの制御コードと 3 ビットのチップ選択ビットで構成されます.
- メモリは 1 バイト単位でアドレス指定され, メモリアドレスは 13 ビットで表現されます. メモリの容量とメモリアドレスのビット数の関係は 8K = $8 \times 1024 = 2^{13}$ の式で表される通りです.
- 書き込みページバッファは 32 バイトです.

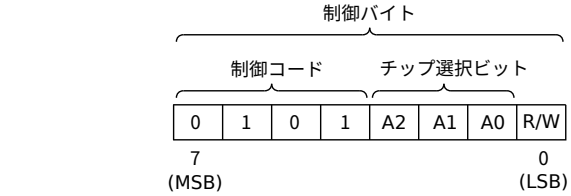
図 25.11b にアドレス指定の際に使用する 3 バイトのビット構成を示します. 図 25.13 に 24FC64 I^{2}C 通信プロトコルを示します. 24FC64 の内部には次に読み書きするデータアドレスの記憶があり, 連続して書き込みあるいは読み出しを行う場合には, データアドレスは 1 づつ増加されます. なお書き込みページバッファは 32 バイトであることから, ひとつのトランザクションで書き込むデータはすべて同じ 32 バイト単位のメモリアドレス内になければなりません.

データの書き込みのトランザクションはマスタの STOP 信号の送信により EEPROM 内のバッファから実際の書き込みが開始します. しかしこの書き込みは即座には完了せず, データシートによれば 5 ミリ秒を要するとの記載があります. この時間はライトサイクル時間と呼ばれており, 書き込みに要する最長の時間です. ただし実際の書き込みはこの時間よりも早く済む場合もあります. ライトサイクル中は次のトランザクションを開始できず, ACK 返答はありません. 次のトランザクションが可能となるまで待機する方法には 2 通りあります. そのひとつはライトサイクル時間の経過を待機することです. もうひとつの方法はアクノリッジポーリング (acknowldge polling) と呼ばれる技法です. これは R/W ビットの値を 0 とする制御バイトの送信を ACK があるまで繰り返すというもので, EEPROM がライトサイクルが終了すれば ACK を返せることを利用した確認方法です. ライトサイクル時間の経過を待機する方法では早く書き込みが終了しても一定の時間を待機しているために無駄な待機時間が生じます. 一方, アクノリッジポーリングでは実際の書き込みが終了するとただちに次のトランザクションを開始できますから, 無駄な待機時間は生じません.

25.6.2　24FC1025

I^{2}C EEPROM 24FC1025 の通信プロトコルを概説します. 上述の 24FC64 と類似した方法で使用が可能なので簡単な説明に留めます. 24FC1025 と同じメモリ容量の EEPROM に 24LC1025 がありますが, 通信速度の上限が異なるものの通信プロトコルは同一です.

- デバイスには図 3.29 で示した通り 3 本のチップ選択ピンがあります. しかし A2 ピンは H に固定する決まりになっており, 設定変更が可能なのは 2 本です. 従って 1 つの I^{2}C バスに同時接続できる 24FC1025 は最大 4 (= 2^2) 個までです.
- 各デバイスの 7 ビットの I^{2}C アドレスは図 25.12a に示す通り, 4 ビットの制御コード, 1 ビットのブロック選択ビット B0, 2 ビットのチップ選択ビットで構成されます. 24FC1025 の内部は 64K バイトのブロック 2 つで構成されており, ブロック選択ビットはアクセスするブロックの指定に使用します.

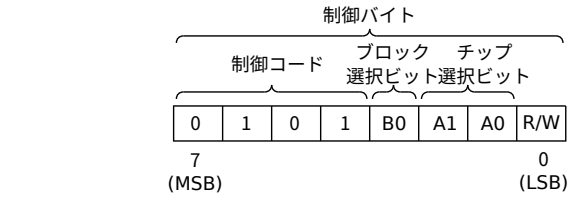


(a) 制御バイト



(b) アドレスシーケンス

図 25.11: I^{2}C EEPROM 24FC64 の制御バイトとアドレスシーケンス



(a) 制御バイト



(b) アドレスシーケンス

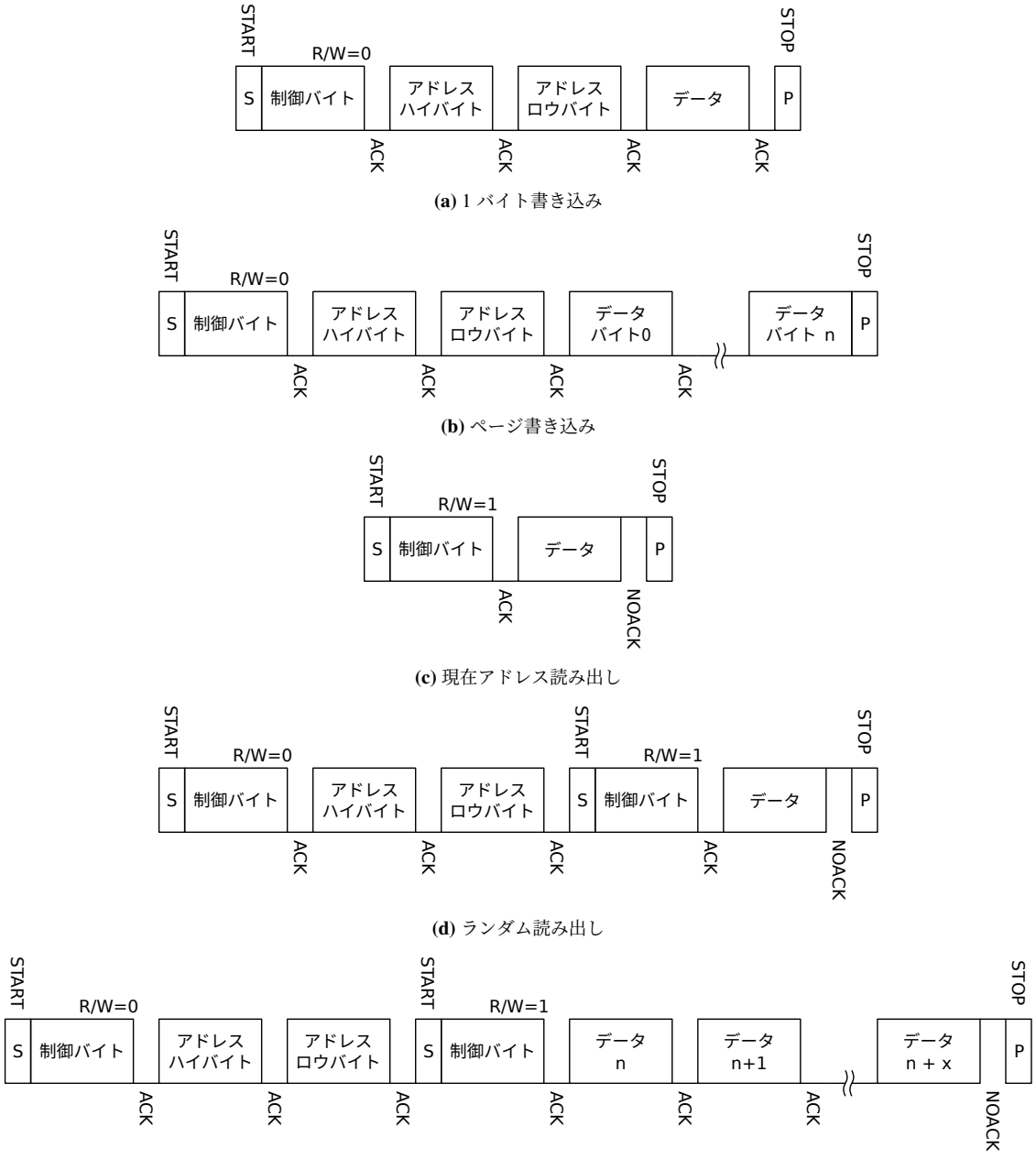
図 25.12: I^{2}C EEPROM 24FC1025 の制御バイトとアドレスシーケンス

- メモリ内容は 1 バイト単位でアドレス指定され, メモリアドレスは 17 ビットで表現されます. メモリの容量とメモリアドレスのビット数の関係は 128K = 128 × 1024 = 2^{17} の式で表されます.
- 書き込みページバッファは 128 バイトです.

図 25.12b にアドレス指定の際に使用する 3 バイトのビット構成を示します. 24FC1025 I^{2}C 通信プロトコルは先述の図 25.13 と同じですが, 制御バイトは図 25.12b の規約に従います. 書き込みページバッファは 128 バイトなのでひとつのトランザクションで書き込むデータはすべて同じ 128 バイト単位のメモリアドレス内になければなりません. 更に 64K バイトのブロック境界を越えた連続アクセスはできないので注意が必要です. ブロック境界を越える時にはアドレスを再設定せねばなりません. 24FC1025 も 24FC64 と同様にライトサイクル中は ACK を返しませんから, アクノリッジポーリング (acknowldge polling) によるライトサイクルの終了の検出が可能です.

25.7　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します. 図 25.14 に主要なソースファイル間の関係を示します.



(a) 1 バイト書き込み

(b) ページ書き込み

(c) 現在アドレス読み出し

(d) ランダム読み出し

(e) 逐次読み出し

図 25.13: I^{2}C EEPROM 24FC64 と 24FC1025 の I^{2}C 通信プロトコル

表 25.7: 【ソースファイル一覧】 EEPROM 書き込み器

初	頁	番号	ファイル名	内容	行数
✓	431	25.2	`Writer-EEPROM.c`	主ソースコード	178
✓	434	25.3	`devinfo.h`	メモリデバイス情報表: ヘッダ	29
✓	434	25.4	`devinfo.c`	メモリデバイス情報表: 実装	114

(つづく)

(つづき)

初	頁	番号	ファイル名	内容	行数
✓	436	25.5	cmd.h	コマンド関数: ヘッダ	17
✓	436	25.6	cmd.c	コマンド関数: 実装	405
✓	442	25.7	ihex.h	インテル HEX ファイル関数: ヘッダ	26
✓	443	25.8	ihex.c	インテル HEX ファイル関数: 実装	225
✓	453	25.13	f-24fc64.h	24FC64 インターフェース: ヘッダ	18
✓	453	25.14	f-24fc64.c	24FC64 インターフェース: 実装	63
✓	454	25.15	f-24fc1025.h	24FC1025 インターフェース: ヘッダ	18
✓	455	25.16	f-24fc1025.c	24FC1025 インターフェース: 実装	63
	394	23.3	i2c.h	I^2C インターフェース関数: ヘッダ	25
	394	23.4	i2c.c	I^2C インターフェース関数: 実装	111
✓	456	25.17	i2c-24fc64.h	24FC64 アクセス関数: ヘッダ	20
✓	457	25.18	i2c-24fc64.c	24FC64 アクセス関数: 実装	133
✓	459	25.19	i2c-24fc1025.h	24FC1025 アクセス関数: ヘッダ	18
✓	459	25.20	i2c-24fc1025.c	24FC1025 アクセス関数: 実装	135
✓	446	25.9	cons.h	コンソール関数: ヘッダ	16
✓	446	25.10	cons.c	コンソール関数: 実装	90
✓	448	25.11	com.h	RS-232C 通信関数: ヘッダ	29
✓	450	25.12	com.c	RS-232C 通信関数: 実装	234
✓	429	25.1	io.h	I/O ピン割付: ヘッダ	23
✓	461	25.21	mcr.h	MCR・周辺モジュール: ヘッダ	10
✓	461	25.22	mcr.c	MCR・周辺モジュール: 実装	105

計 2105 行 (うち初出 1969 行, 既出 136 行)

本章のファームウエアの大まかな構造を図 25.15 に示します. 概略をわかりやすくするためにメインループと書き込み処理だけを示しています.

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

ソースファイル番号 25.1: io.h

```
#if !defined(__IO_H__)
#define __IO_H__

/* ボーレート選択SW: RA3, RA5 */
#define BR_SW1_ON()          (RA3 == 0)
#define BR_SW2_ON()          (RA5 == 0)
/* LED WRITING: RC3 */
#define OUT_LED_MEM_OFF()    (LATC3 = 0)
#define OUT_LED_MEM_ON()     (LATC3 = 1)
/* LED ACTIVE: RA4 */
#define OUT_LED_ACT_OFF()    (LATA4 = 0)
#define OUT_LED_ACT_ON()     (LATA4 = 1)
/* メモリデバイス制御: RA0, RA1, RA2, RC2 */
#define OUT_MEM_A0_LOW()     (LATA0 = 0)
#define OUT_MEM_A0_HIGH()    (LATA0 = 1)
#define OUT_MEM_A1_LOW()     (LATA1 = 0)
```

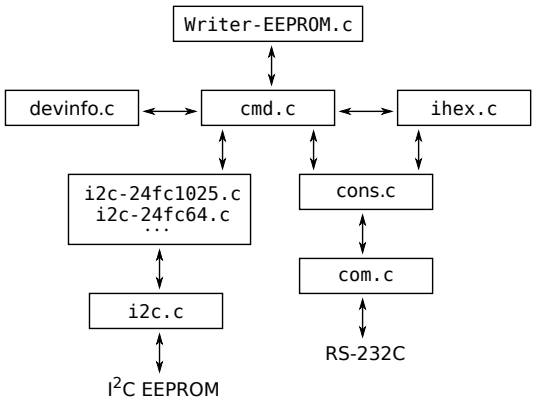


図 25.14: ファームウエアにおける主要ソースファイル間の関係

```
main(){
  for (;;) {
    EEPROMデバイスを選択;
    while (EEPROMデバイスを選択中) {
      コマンドを受信;
      if (コマンド == 選択解除) {
        EEPROMデバイスの選択を終了;
      } else {
        コマンドに対応する関数を呼び出す;
      }
    }
  }
}

EEPROMへの書き込み (関数cmd_write()とihex_read_line_main()にて):
{
  EEPROMのI2C接続を初期化;
  for (;;) {
    インテルHEXファイルの1レコード(1行)をRS232Cより受信;
    if (レコードは配置アドレスの設定) {
      ihex_exaddr = 指定された配置アドレス;
    } else if (レコードはデータ) {
      受信したデータをI2Cバスを通じてEEPROMに書き込む
        (ただし書き込みはメモリアドレスihex_exaddrより);
      ihex_exaddrをデータ数分だけ増加;
    } else if (レコードは終端) {
      break;
    }
  }
  EEPROMのI2C接続を終了;
}
```

図 25.15: ファームウエア構造 (概要)

```
17 #define OUT_MEM_A1_HIGH()   (LATA1 = 1)
18 #define OUT_MEM_A2_LOW()    (LATA2 = 0)
19 #define OUT_MEM_A2_HIGH()   (LATA2 = 1)
20 #define OUT_MEM_WP_LOW()    (LATC2 = 0)
21 #define OUT_MEM_WP_HIGH()   (LATC2 = 1)
22
23 #endif /*__IO_H__*/
```

ソースファイル Writer-EEPROM.c (主ソースコード)

動作の主となる部分です. 関数 main() が初期化を終えた後は以下の動作の繰り返しです.

1. EEPROM デバイス選択
   登録済みの EEPROM デバイスからひとつを選択します.
2. EEPROM デバイス接続確認
   EEPROM がゼロプレッシャソケットに載せられているか確認します.
3. コマンド選択と実行
   EEPROM に対するコマンドを受け付けて対応する動作の実行を繰り返します. ただし終了コマンド U の場合には EEPROM デバイスの選択に戻ります.

コマンド: は特殊な目的で用意しています. コマンド選択時に間違ってインテル HEX ファイルをアップロードした場合に, インテル HEX ファイルの 1 行全体を無視して意図しないコマンドが実行されるのを防ぎます. これにはインテル HEX ファイルの各行の最初の文字が: であることを利用して, コマンド: の実行内容を 1 行を無視しています. 人間の操作ミスを補うためのプログラム上の仕組みは実に面倒です.

関数 main() 以外では, RS-232C のボーレートを DIP スイッチ設定より求める関数や, 登録済み EEPROM デバイスからひとつを選択する関数などが含まれています.

ソースファイル番号 25.2: Writer-EEPROM.c

```
1  /*********************************************
2   * I2C EEPROM ライタ / PIC16F18326
3   * by Hirotsugu Kakugawa
4   *********************************************/
5  #include <xc.h>
6  #include <stdint.h>
7  #include <string.h>
8  #include <ctype.h>
9  #include "mcr.h"
10 #include "io.h"
11 #include "ihex.h"
12 #include "devinfo.h"
13 #include "cmd.h"
14 #include "com.h"
15 #include "cons.h"
16 #include "i2c.h"
17 #include "i2c-24fc1025.h"
18 #include "i2c-24fc64.h"
19 #include "f-24fc1025.h"
20 #include "f-24fc64.h"
21
22 static void    read_sw_baud_rate(void);
23 static void    device_print_list(void);
24 static uint8_t device_selected(char ch);
25
26 /*-------------------------------------------------------
27  * MAIN
28  *-------------------------------------------------------*/
29 void main(void)
30 {
31   /* 初期化 */
```

```
32    System_Init();
33    /* ウオッチドッグタイマ初期化: 2 [Sec] */
34    WDTCONbits.WDTPS = 0x0B; /* 0b01011 */
35    CLRWDT();
36    __delay_ms(500);
37    /* コンソール */
38    com_init();
39    read_sw_baud_rate();
40    /* 割り込み許可 */
41    GIE = 1;
42    PEIE = 1;
43    /* 起動メッセージ */
44    CLRWDT();
45    cons_print_newline();
46    cons_print_newline();
47    cons_print_newline();
48    cons_print_newline();
49    cons_print_pgm("=============");
50    cons_print_newline();
51    cons_print_pgm("EEPROM WRITER");
52    cons_print_newline();
53    cons_print_pgm("=============");
54    OUT_LED_ACT_ON();
55    OUT_LED_MEM_OFF();
56    /* 主ループ */
57    for (;;) {
58      CLRWDT();
59      /* EEPROMデバイス選択 */
60      uint8_t dev = DEV_NULL;
61      while (dev == DEV_NULL) {
62        CLRWDT();
63        cons_print_newline();
64        device_print_list();
65        cons_print_newline();
66        cons_print_pgm("SELECT A DEVICE FROM LIST: ");
67        int16_t c = 0;
68        while ((c = cons_getc()) < 0) {
69          CLRWDT();
70        }
71        dev = device_selected((char)c);
72        if (dev != DEV_NULL) {
73          cons_putc((char)c);
74        }
75      }
76      /* EEPROMデバイス接続確認 */
77      if ((dev != DEV_NULL) && !(*devs[dev].f_init)(dev)) {
78        dev = DEV_NULL;
79        cons_print_newline();
80        cons_print_pgm("ERROR: MEMORY DEVICE NOT CONNECTED");
81      }
82      /* コマンド選択 */
83      uint8_t prompt = 1;
84      while (dev != DEV_NULL) {
85        CLRWDT();
86        if (prompt == 1) {
87          cons_print_newline();
88          cons_print_pgm("[");
89          cons_print_pgm(devs[dev].devnam);
90          cons_print_pgm("] ");
91        }
92        prompt = 1;
93        int16_t c = 0;
94        while ((c = cons_getc()) < 0) {
95          CLRWDT();
96        }
97        char ch = toupper((char)c);
98        switch (ch) {
99        case 'U': cons_putc(ch); dev = DEV_NULL;  break;
100       case 'W': cons_putc(ch); cmd_write(dev);  break;
101       case 'R': cons_putc(ch); cmd_read(dev);   break;
102       case 'V': cons_putc(ch); cmd_verify(dev); break;
```

```
103         case 'D': cons_putc(ch); cmd_dump(dev);   break;
104         case 'E': cons_putc(ch); cmd_erase(dev);  break;
105         case 'P': cons_putc(ch); cmd_desc(dev);   break;
106         case ':': cmd_skip_line(); prompt = 0;    break;
107         default:  cmd_usage();                    break;
108         }
109       }
110     }
111   }
112
113   /* RS232Cボーレート選択SWを読む */
114   static void read_sw_baud_rate(void)
115   {
116     if        (!BR_SW1_ON() && !BR_SW2_ON()) {
117       com_baud_rate_set(COM_BAUD_RATE_4800);
118     } else if ( BR_SW1_ON() && !BR_SW2_ON()) {
119       com_baud_rate_set(COM_BAUD_RATE_9600);
120     } else if (!BR_SW1_ON() &&  BR_SW2_ON()) {
121       com_baud_rate_set(COM_BAUD_RATE_19200);
122     } else if ( BR_SW1_ON() &&  BR_SW2_ON()) {
123       com_baud_rate_set(COM_BAUD_RATE_38400);
124     }
125   }
126
127   /*-------------------------------------------------------
128    * EEPROMデバイス選択
129    *-------------------------------------------------------*/
130
131   const char devselect_ch[] = "123456789abcdefghijklmnopqrstuvwxyz";
132
133   /* EEPROMデバイスの一覧を表示（選択肢文字も表示） */
134   static void device_print_list(void)
135   {
136     uint16_t xpos = CONS_WIDTH + 1;
137     cons_print_newline();
138     cons_print_pgm("DEVICE LIST:");
139     uint8_t i = 0;
140     for (i = 1;
141          (devs[i].devnam != NULL)
142            && (devselect_ch[(unsigned)i-1] != '\0');
143          i++) {
144       uint16_t sl = strlen(devs[i].devnam) + 6;
145       /* 行末に近ければ改行 */
146       CLRWDT();
147       if (xpos + sl >= CONS_WIDTH-1) {
148         cons_print_newline();
149         cons_print_pgm("  ");
150         xpos = 2;
151       }
152       /* 選択肢文字を表示 */
153       CLRWDT();
154       cons_print_pgm("(");
155       cons_putc(devselect_ch[(unsigned)i-1]);
156       cons_print_pgm(") ");
157       /* EEPROMデバイス名を表示 */
158       CLRWDT();
159       cons_print_pgm(devs[i].devnam);
160       cons_print_pgm("  ");
161       xpos = xpos + sl;
162     }
163   }
164
165   /* devselect_ch[] の文字に対応する devs[] の添字を返す */
166   static uint8_t device_selected(char ch)
167   {
168     uint8_t i;
169     for (i = 0;
170          (devselect_ch[i] != '\0')
171            && (devs[(unsigned)i+1].devnam != NULL);
172          i++) {
173       if (ch == devselect_ch[i]) {
```

```
174         return (unsigned)i + 1;
175       }
176     }
177     return DEV_NULL;
178   }
```

ソースファイル devinfo.h (EEPROM デバイス情報表: ヘッダ)

構造体 device_info_s を定義します. この構造体は EEPROM デバイスの抽象化機構であり, 各 EEPROM デバイスの固有情報である登録名称, メモリ容量, ボーレート設定値の他, 読み書きを行う関数へのポインタなどを保持します.

ソースファイル番号 25.3: devinfo.h

```
1   #if !defined(__DEVINFO_H__)
2   #define __DEVINFO_H__
3
4   /* EEPROMデバイス情報表 */
5   struct device_info_s {
6     const char *devnam;       /* EEPROMデバイス名 */
7     uint32_t   size_in_byte;  /* メモリ容量(単位：バイト) */
8     uint8_t    ssp1add_value; /* SSP1ADD（ボーレート）設定値 */
9     const char *desc;         /* デバイス説明 */
10    int8_t  (*f_init)(uint8_t dev);
11    void    (*f_deinit)(uint8_t dev);
12    void    (*f_erase)(uint8_t dev);
13    void    (*f_read_begin)(uint8_t dev,
14                            uint32_t addr,
15                            uint32_t bytes);
16    uint8_t (*f_read_byte)(uint8_t dev);
17    void    (*f_read_done)(uint8_t dev);
18    void    (*f_write_begin)(uint8_t dev,
19                             uint32_t addr,
20                             uint32_t nbytes);
21    void    (*f_write_byte)(uint8_t dev, uint8_t d);
22    void    (*f_write_done)(uint8_t dev);
23  };
24
25  extern const struct device_info_s devs[];
26
27  #define DEV_NULL  0
28
29  #endif /*__DEVINFO_H__*/
```

ソースファイル devinfo.c (EEPROM デバイス情報表: 実装)

構造体 device_info_s の配列変数 devs[] を宣言して, 各 EEPROM デバイスの情報を保持します. 本ファームウエアで新たな EEPROM デバイスを利用可能にするには, 読み書きを行う関数を作成し, EEPROM デバイスの固有情報と読み書きを行う関数へのポインタをこの配列に登録するだけです.

ソースファイル番号 25.4: devinfo.c

```
1   /* EEPROMデバイス情報表 */
2   #include <xc.h>
3   #include <stdint.h>
4   #include <string.h>
5   #include "mcr.h"
6   #include "devinfo.h"
7   #include "i2c.h"
8   #include "i2c-24fc1025.h"
9   #include "i2c-24fc64.h"
10  #include "f-24fc1025.h"
11  #include "f-24fc64.h"
12
13  const struct device_info_s devs[] = {
```

```
14    { "",   0, 0,
15      NULL, NULL, NULL, NULL, NULL, NULL },
16    { "24FC1025", MEMSIZE_24FC1025,
17      I2C_BAUD_RATE_SSPxADD_1MHZ(_XTAL_FREQ),
18      "MICROCHIP TECHNOLOGY, I2C, 1024KBIT (128KBYTE)",
19      f_24fc1025_init,
20      f_24fc1025_deinit,
21      f_24fc1025_erase,
22      f_24fc1025_read_begin,
23      f_24fc1025_read_byte,
24      f_24fc1025_read_done,
25      f_24fc1025_write_begin,
26      f_24fc1025_write_byte,
27      f_24fc1025_write_done },
28    { "24LC1025", MEMSIZE_24FC1025,
29      I2C_BAUD_RATE_SSPxADD_400KHZ(_XTAL_FREQ),
30      "MICROCHIP TECHNOLOGY, I2C, 1024KBIT (128KBYTE)",
31      f_24fc1025_init,
32      f_24fc1025_deinit,
33      f_24fc1025_erase,
34      f_24fc1025_read_begin,
35      f_24fc1025_read_byte,
36      f_24fc1025_read_done,
37      f_24fc1025_write_begin,
38      f_24fc1025_write_byte,
39      f_24fc1025_write_done },
40    { "24FC512", MEMSIZE_24FC512,
41      I2C_BAUD_RATE_SSPxADD_1MHZ(_XTAL_FREQ),
42      "MICROCHIP TECHNOLOGY, I2C, 512KBIT (64KBYTE)",
43      f_24fc64_init,
44      f_24fc64_deinit,
45      f_24fc64_erase,
46      f_24fc64_read_begin,
47      f_24fc64_read_byte,
48      f_24fc64_read_done,
49      f_24fc64_write_begin,
50      f_24fc64_write_byte,
51      f_24fc64_write_done },
52    { "24LC512", MEMSIZE_24FC512,
53      I2C_BAUD_RATE_SSPxADD_400KHZ(_XTAL_FREQ),
54      "MICROCHIP TECHNOLOGY, I2C, 512KBIT (64KBYTE)",
55      f_24fc64_init,
56      f_24fc64_deinit,
57      f_24fc64_erase,
58      f_24fc64_read_begin,
59      f_24fc64_read_byte,
60      f_24fc64_read_done,
61      f_24fc64_write_begin,
62      f_24fc64_write_byte,
63      f_24fc64_write_done },
64    { "24FC256", MEMSIZE_24FC256,
65      I2C_BAUD_RATE_SSPxADD_1MHZ(_XTAL_FREQ),
66      "MICROCHIP TECHNOLOGY, I2C, 256KBIT (32KBYTE)",
67      f_24fc64_init,
68      f_24fc64_deinit,
69      f_24fc64_erase,
70      f_24fc64_read_begin,
71      f_24fc64_read_byte,
72      f_24fc64_read_done,
73      f_24fc64_write_begin,
74      f_24fc64_write_byte,
75      f_24fc64_write_done },
76    { "24LC256", MEMSIZE_24FC256,
77      I2C_BAUD_RATE_SSPxADD_400KHZ(_XTAL_FREQ),
78      "MICROCHIP TECHNOLOGY, I2C, 256KBIT (32KBYTE)",
79      f_24fc64_init,
80      f_24fc64_deinit,
81      f_24fc64_erase,
82      f_24fc64_read_begin,
83      f_24fc64_read_byte,
84      f_24fc64_read_done,
85      f_24fc64_write_begin,
86      f_24fc64_write_byte,
```

```
87      f_24fc64_write_done },
88    { "24FC64", MEMSIZE_24FC64,
89      I2C_BAUD_RATE_SSPxADD_1MHZ(_XTAL_FREQ),
90      "MICROCHIP TECHNOLOGY, I2C, 64KBIT (8KBYTE)",
91      f_24fc64_init,
92      f_24fc64_deinit,
93      f_24fc64_erase,
94      f_24fc64_read_begin,
95      f_24fc64_read_byte,
96      f_24fc64_read_done,
97      f_24fc64_write_begin,
98      f_24fc64_write_byte,
99      f_24fc64_write_done },
100   { "24LC64", MEMSIZE_24FC64,
101     I2C_BAUD_RATE_SSPxADD_400KHZ(_XTAL_FREQ),
102     "MICROCHIP TECHNOLOGY, I2C, 64KBIT (8KBYTE)",
103     f_24fc64_init,
104     f_24fc64_deinit,
105     f_24fc64_erase,
106     f_24fc64_read_begin,
107     f_24fc64_read_byte,
108     f_24fc64_read_done,
109     f_24fc64_write_begin,
110     f_24fc64_write_byte,
111     f_24fc64_write_done },
112   { NULL, 0, 0,
113     NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL }
114 };
```

ソースファイル cmd.h (コマンド関数: ヘッダ)

この後に紹介する cmd.c で実装する関数群のプロトタイプ宣言です.

ソースファイル番号 25.5: cmd.h

```
1  #if !defined(__CMD_H__)
2  #define __CMD_H__
3
4  #define CONS_WIDTH  80  /* コンソール幅 */
5
6  extern void  cmd_usage(void);
7  extern void  cmd_write(uint8_t dev);
8  extern void  cmd_read(uint8_t dev);
9  extern void  cmd_verify(uint8_t dev);
10 extern void  cmd_dump(uint8_t dev);
11 extern void  cmd_erase(uint8_t dev);
12 extern void  cmd_desc(uint8_t dev);
13 extern void  cmd_skip_line(void);
14 extern void  cmd_write_verify(uint8_t dev, enum ihex_cmd_e cmd);
15 extern void  erase_by_write(uint8_t dev);
16
17 #endif /*__CMD_H__*/
```

ソースファイル cmd.c (コマンド関数: 実装)

コマンドを実際に実行する関数群の実装です. このソースファイルではすべての EEPROM デバイスに共通の動作だけを記述しています. EEPROM デバイス固有の動作は EEPROM デバイスごとに用意された別の関数を呼び出します.

ソースファイル番号 25.6: cmd.c

```
1  /*-------------------------------------------------------
2   * 各種コマンド処理
3   *-------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include <ctype.h>
```

```
7  #include "mcr.h"
8  #include "io.h"
9  #include "ihex.h"
10 #include "devinfo.h"
11 #include "cmd.h"
12 #include "com.h"
13 #include "cons.h"
14
15 static void   skipline(void);
16 static int8_t confirm(const char* prompt, int8_t silent, int8_t nl);
17
18 /* コマンド: Usage */
19 void cmd_usage(void)
20 {
21   CLRWDT();
22   cons_print_newline();
23   cons_print_pgm("COMMAND LIST:");
24   cons_print_newline();
25   cons_print_pgm("  U - UNSELECT DEVICE");
26   cons_print_newline();
27   cons_print_pgm("  W - WRITE (UPLOAD INTEL-HEX FILE)");
28   cons_print_newline();
29   cons_print_pgm("  R - READ (DOWNLOAD INTEL-HEX FILE)");
30   cons_print_newline();
31   cons_print_pgm("  V - VERIFY (COMPARE MEMORY AND INTEL-HEX FILE)");
32   cons_print_newline();
33   cons_print_pgm("  D - DUMP (PRINT MEMORY CONTENTS)");
34   cons_print_newline();
35   cons_print_pgm("  E - ERASE (BLANK MEMORY CONTENTS)");
36   cons_print_newline();
37   cons_print_pgm("  P - PRINT DEVICE DESCRPTION");
38 }
39
40 /* コマンド: 行末まで読み飛ばし
41  * (間違って HEX ファイルをアップロードした時のため) */
42 void cmd_skip_line(void)
43 {
44   skipline();
45 }
46
47 static void skipline(void)
48 {
49   for (;;) {
50     CLRWDT();
51     int16_t ch = 0;
52     while ((ch = cons_getc()) < 0) {
53       CLRWDT();
54     }
55     if ((ch == CHAR_CR) || (ch == CHAR_LF)) {
56       break;
57     }
58   }
59 }
60
61 /* コマンド: 書込 */
62 void cmd_write(uint8_t dev)
63 {
64   cmd_write_verify(dev, IHEX_CMD_WRITE);
65 }
66
67 /* コマンド: ベリファイ */
68 void cmd_verify(uint8_t dev)
69 {
70   cmd_write_verify(dev, IHEX_CMD_VERIFY);
71 }
72
73 /* 書込/ベリファイ */
74 void cmd_write_verify(uint8_t dev, enum ihex_cmd_e cmd)
75 {
76   uint16_t nerrs = 0;
77   /* 確認 */
```

```
78    cons_print_newline();
79    if (cmd == IHEX_CMD_WRITE) {
80      cons_print_pgm("UPLOAD INTEL-HEX FILE TO WRITE ");
81    } else if (cmd == IHEX_CMD_VERIFY) {
82      cons_print_pgm("UPLOAD INTEL-HEX FILE TO VERIFY ");
83    }
84    /* EEPROMデバイス初期化 */
85    if (!(*devs[dev].f_init)(dev)) {
86      cons_print_newline();
87      cons_print_pgm("ERROR: MEMORY DEVICE NOT CONNECTED");
88      return;
89    }
90    OUT_LED_MEM_ON();
91    /* 書き込みループ */
92    ihex_init();
93    enum ihex_ret_e r = IHEX_OK;
94    uint16_t lineno = 0;
95    uint16_t progress = 0;
96    for (;;) {
97      CLRWDT();
98      /* HEXファイル1行読込(メモリ書込/ベリファイ) */
99      r = ihex_read_line(dev, cmd);
100     /* 進行状況を表示 */
101     lineno++;
102     if (progress == 0) {
103       progress++;
104       if (r == IHEX_OK) {
105         cons_print_newline();
106         if (cmd == IHEX_CMD_WRITE) {
107           cons_print_pgm("WRITING");
108         } else if (cmd == IHEX_CMD_VERIFY) {
109           cons_print_pgm("VERIFYING");
110         }
111       }
112     }
113     if ((lineno & 0x3F) == 1) {
114       cons_print_pgm(".");
115       progress++;
116       if (progress == 64) {
117         progress = 0;
118       }
119     }
120     /* ファイル終端ならば完了 */
121     if (r == IHEX_EOF) {
122       break;
123     }
124     /* エラー処理 */
125     if ((r != IHEX_OK) && (nerrs == 0)){
126       nerrs = 1;
127       cons_print_newline();
128       switch (r) {
129       case IHEX_ERR_CHECKSUM:
130         cons_print_pgm("CHECKSUM DOES NOT MATCH");
131         break;
132       case IHEX_ERR_FORMAT:
133         cons_print_pgm("BROKEN INTEL-HEX FILE");
134         break;
135       case IHEX_ERR_ADDR:
136         cons_print_pgm("ADDRESS OUT OF RANGE");
137         break;
138       case IHEX_ERR_VERIFY:
139         cons_print_pgm("UNVERIFIED");
140         break;
141       case IHEX_ERR_UNSUPP:
142         cons_print_pgm("UNSUPPORTED INTEL-HEX FORMAT");
143         break;
144       default:
145         cons_print_pgm("ERROR");
146         break;
147       }
148     }
```

```
149       if (r == IHEX_ERR_FORMAT) {
150         break;
151       }
152     }
153     /* 終了メッセージ */
154     cons_print_newline();
155     if (nerrs == 0) {
156       cons_print_pgm("COMPLETED");
157     }
158     /* EEPROMデバイス動作完了 */
159     (*devs[dev].f_deinit)(dev);
160     OUT_LED_MEM_OFF();
161   }
162
163   /* コマンド: メモリ内容読み出し(Intel HEXファイルとして) */
164   void cmd_read(uint8_t dev)
165   {
166   #define IHEX_LINE_NBYTES  16
167     uint32_t addr = 0;
168     uint8_t nbytes = 0;
169     uint8_t data[IHEX_LINE_NBYTES];
170     uint8_t done = 0;
171     uint8_t emit = 0;
172     uint8_t success = 0;
173     /* 確認 */
174     if (!confirm("READY?", 1, 1)){
175       return;
176     }
177     /* EEPROMデバイス初期化 */
178     if (!(*devs[dev].f_init)(dev)) {
179       cons_print_pgm("ERROR: MEMORY DEVICE NOT CONNECTED");
180       return;
181     }
182     OUT_LED_MEM_ON();
183     /* Intel HEX ファイルとして表示 */
184     (*devs[dev].f_read_begin)(dev, addr, devs[dev].size_in_byte);
185     for (;;) {
186       CLRWDT();
187       if (emit == 1) {
188         emit = 0;
189         if (cons_get_ready()) {
190           cons_print_newline();
191           cons_print_pgm("INTERRUPTED");
192           cons_print_newline();
193           while (cons_getc() >= 0) { /* 受信文字すべてを捨てる */
194             ;
195           }
196           break;
197         }
198         if (nbytes != 0) {
199           uint8_t chksum = 0;
200           /* レコードタイプ 4; 拡張線形アドレス(32-bit)出力 */
201           if ((addr & 0xFFFF) == 0x0000) {
202             uint8_t d2 = (addr >> 16) & 0xFF;
203             uint8_t d3 = (addr >> 24) & 0xFF;
204             chksum = (unsigned)0x02 + 0x00 + 0x00 + 0x04 + d3 + d2;
205             cons_print_pgm(":02000004");
206             cons_print_hex(d3);
207             cons_print_hex(d2);
208             cons_print_hex((unsigned)0 - chksum);
209             cons_print_newline();
210           }
211           /* レコードタイプ 0; データ出力(16-bitアドレッシング) */
212           uint8_t a0 =  addr       & 0xFF;
213           uint8_t a1 = (addr >> 8) & 0xFF;
214           chksum = (unsigned)0x00 + nbytes + a1 + a0 + 0x00;
215           cons_print_pgm(":");
216           cons_print_hex(nbytes);
217           cons_print_hex(a1);
218           cons_print_hex(a0);
219           cons_print_pgm("00");
```

```
220           uint8_t i = 0;
221           for (i = 0; i < nbytes; i++) {
222             addr++;
223             cons_print_hex(data[i]);
224             chksum += data[i];
225           }
226           nbytes = 0;
227           cons_print_hex((unsigned)0 - chksum);
228           cons_print_newline();
229         }
230       }
231       if (done == 1) {
232         /* レコードタイプ 2; EOF */
233         cons_print_pgm(":00000001FF");
234         cons_print_newline();
235         success = 1;
236         break;
237       }
238       if (addr + nbytes == devs[dev].size_in_byte) {
239         emit = 1;
240         done = 1;
241       } else {
242         data[nbytes] = (*devs[dev].f_read_byte)(dev);
243         nbytes++;
244         if (nbytes == IHEX_LINE_NBYTES) {
245           emit = 1;
246         }
247       }
248     }
249     (*devs[dev].f_read_done)(dev);
250     /* EEPROMデバイス動作完了 */
251     (*devs[dev].f_deinit)(dev);
252     /* 完了 */
253     OUT_LED_MEM_OFF();
254     if (success == 1) {
255       uint16_t t = 6000;
256       while (t != 0) {
257         CLRWDT();
258         t--;
259         __delay_ms(10);
260         if (cons_getc() >= 0) {
261           break;
262         }
263       }
264       cons_print_pgm("COMPLETED");
265     }
266   }
267
268   static int8_t confirm(const char* prompt, int8_t silent, int8_t nl)
269   {
270     cons_print_newline();
271     cons_print_pgm(prompt);
272     cons_print_pgm(" (Y/N) ");
273     if (nl) {
274       cons_print_newline();
275     }
276     int16_t c = 0;
277     while ((c = cons_getc()) < 0) {
278       CLRWDT();
279     }
280     char ch = (char)c;
281     if (!silent) {
282       cons_putc(ch);
283     }
284     if ((ch != 'Y') && (ch != 'y')) {
285       cons_print_newline();
286       cons_print_pgm("CANCELD");
287       return 0;
288     }
289     return 1;
290   }
291
```

```
292 /* コマンド: メモリ内容ダンプ */
293 void cmd_dump(uint8_t dev)
294 {
295   /* EEPROMデバイス初期化 */
296   if (!(*devs[dev].f_init)(dev)) {
297     cons_print_newline();
298     cons_print_pgm("ERROR: MEMORY DEVICE NOT CONNECTED");
299     return;
300   }
301   /* 256バイトごとに16進数ダンプ */
302   uint32_t addr;
303   for (addr = 0; addr < devs[dev].size_in_byte; addr++) {
304     CLRWDT();
305     uint8_t a8 = addr & 0xFF;
306     if (a8 == 0x00) {
307       (*devs[dev].f_read_begin)(dev, addr, 256);
308     }
309     if (a8 == 0x00) {
310       cons_print_newline();
311       cons_print_pgm("          ");
312       cons_print_pgm("+0 +1 +2 +3 +4 +5 +6 +7 ");
313       cons_print_pgm("+8 +9 +A +B +C +D +E +F");
314     }
315     if ((a8 & 0x0F) == 0x00) {
316       cons_print_newline();
317       cons_print_hex((addr >> 24) & 0xFF);
318       cons_print_hex((addr >> 16) & 0xFF);
319       cons_print_hex((addr >>  8) & 0xFF);
320       cons_print_hex( addr        & 0xFF);
321       cons_print_pgm(": ");
322     }
323     uint8_t d = (*devs[dev].f_read_byte)(dev);
324     cons_print_hex(d);
325     cons_putc(' ');
326     if (a8 == 0xFF) {
327       (*devs[dev].f_read_done)(dev);
328       /* 次の256バイトを表示する:
329        * (スペースキー押下)か終了するか(それ以外のキー押下) */
330       cons_print_newline();
331       cons_print_pgm("--MORE-- ");
332       cons_print_pgm("(HIT SPACE KEY TO CONTINUE; ");
333       cons_print_pgm("'Q' TO EXIT) ");
334       int16_t c = 0;
335       while ((c = cons_getc()) < 0) {
336         CLRWDT();
337       }
338       if (toupper((char)c) == 'Q') {
339         break;
340       }
341     }
342   }
343   /* EEPROMデバイス動作完了 */
344   (*devs[dev].f_deinit)(dev);
345 }
346
347 /* コマンド: 消去 */
348 void cmd_erase(uint8_t dev)
349 {
350   enum ihex_ret_e r = IHEX_OK;
351   /* 確認 */
352   if (!confirm("ERASE MEMORY?", 0, 0)){
353     return;
354   }
355   /* EEPROMデバイス初期化 */
356   if (!(*devs[dev].f_init)(dev)) {
357     cons_print_newline();
358     cons_print_pgm("ERROR: MEMORY DEVICE NOT CONNECTED");
359     return;
360   }
361   OUT_LED_MEM_ON();
362   /* 消去 */
```

```
363   cons_print_newline();
364   cons_print_pgm("ERASING MEMORY...");
365   (*devs[dev].f_erase)(dev);
366   cons_print_newline();
367   cons_print_pgm("COMPLETED");
368   /* EEPROMデバイス動作完了 */
369   (*devs[dev].f_deinit)(dev);
370   OUT_LED_MEM_OFF();
371 }
372
373 /* 書き込みによる消去 */
374 void erase_by_write(uint8_t dev)
375 {
376   uint32_t addr;
377   (*devs[dev].f_write_begin)(dev, 0, devs[dev].size_in_byte);
378   uint16_t xpos = CONS_WIDTH + 1;
379   for (addr = 0; addr < devs[dev].size_in_byte; addr++) {
380     CLRWDT();
381     if ((addr & 0x1FF) == 0x000) {
382       /* 作業進展の表示(512バイトごとアドレスを表示) */
383       uint16_t sl = 7;
384       if (xpos + sl >= CONS_WIDTH-1) {
385         cons_print_newline();
386         cons_print_pgm("  ");
387         xpos = 2;
388       }
389       cons_print_hex((addr >> 16) & 0xFF);
390       cons_print_hex((addr >>  8) & 0xFF);
391       cons_print_hex( addr        & 0xFF);
392       cons_print_pgm(" ");
393       xpos = xpos + sl;
394     }
395     (*devs[dev].f_write_byte)(dev, 0xFF);
396   }
397   (*devs[dev].f_write_done)(dev);
398 }
399
400 /* コマンド: EEPROMデバイス情報表示 */
401 void cmd_desc(uint8_t dev)
402 {
403   cons_print_newline();
404   cons_print_pgm(devs[dev].desc);
405 }
```

ソースファイル ihex.h (インテル HEX ファイル関数: ヘッダ)

この後に紹介する ihex.c で実装する関数群のプロトタイプ宣言です.

ソースファイル番号 25.7: ihex.h

```
1  #if !defined(__IHEX_H__)
2  #define __IHEX_H__
3
4  /* インテルHEXデコーダー返り値 */
5  enum ihex_ret_e {
6    IHEX_OK = 0,
7    IHEX_EOF,
8    IHEX_ERR_CHECKSUM,
9    IHEX_ERR_FORMAT,
10   IHEX_ERR_VERIFY,
11   IHEX_ERR_ADDR,
12   IHEX_ERR_UNSUPP,
13 };
14
15 /* インテルHEXコマンド */
16 enum ihex_cmd_e {
17   IHEX_CMD_WRITE,
18   IHEX_CMD_VERIFY,
19 };
20
```

```
21 extern void             ihex_init(void);
22 extern enum ihex_ret_e ihex_read_line(uint8_t dev,
23                                        enum ihex_cmd_e cmd);
24 extern int16_t          ihex_read_1byte(void);
25
26 #endif /*__IHEX_H__*/
```

ソースファイル ihex.c (インテル HEX ファイル関数: 実装)

EEPROM デバイスへの書き込みとベリファイの時に呼び出されるインテル HEX ファイルを復号する関数の実装です. 受信したインテル HEX ファイルを復号し, 書き込みコマンドの場合には EEPROM デバイスに書き込み, ベリファイコマンドの場合には EEPROM デバイスから読み出した内容と比較します.

ソースファイル番号 25.8: ihex.c

```
1  /*-------------------------------------------------------
2   * インテルHEXデコーダー
3   *-------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "cons.h"
7  #include "ihex.h"
8  #include "devinfo.h"
9
10 static uint32_t ihex_base = 0;
11
12 static enum ihex_ret_e ihex_read_line_main(uint8_t dev,
13                                            enum ihex_cmd_e cmd);
14 static void ihex_read_start(void);
15 static char ihex_read_char(void);
16 static void ihex_skip_to_newline(void);
17
18 /* デコーダー初期化 */
19 void ihex_init(void)
20 {
21   ihex_base = 0;
22 }
23
24 /* 1行デコード */
25 enum ihex_ret_e ihex_read_line(uint8_t dev,
26                                enum ihex_cmd_e cmd)
27 {
28   ihex_read_start();
29   enum ihex_ret_e r = ihex_read_line_main(dev, cmd);
30   ihex_skip_to_newline();
31   return r;
32 }
33
34 /* 1行デコード本体 */
35 static enum ihex_ret_e ihex_read_line_main(uint8_t dev,
36                                            enum ihex_cmd_e cmd)
37 {
38   uint16_t ihex_addr   = 0;
39   uint32_t ihex_exaddr = 0;
40   uint8_t  ihex_bytcnt = 0;
41   uint8_t  ihex_rectyp = 0;
42   uint8_t  ihex_data   = 0;
43   uint8_t  ihex_sum    = 0;
44   uint8_t  ihex_chksum = 0;
45   uint8_t  v0 = 0;
46   uint8_t  v1 = 0;
47   uint8_t  v2 = 0;
48   uint8_t  v3 = 0;
49   int16_t  v = 0;
50   uint8_t  i = 0;
51   char ch;
52   /* 行開始 */
53   ihex_sum = 0;
```

```
54    /* 開始コード */
55    if (ihex_read_char() != ':') {
56      return IHEX_ERR_FORMAT;
57    }
58    /* バイトカウント */
59    if ((v = ihex_read_1byte()) < 0) {
60      return IHEX_ERR_FORMAT;
61    }
62    v0 = (uint8_t) v;
63    ihex_sum += v0;
64    ihex_bytcnt = v0;
65    /* アドレス */
66    if ((v = ihex_read_1byte()) < 0) {
67      return IHEX_ERR_FORMAT;
68    }
69    v0 = (uint8_t) v;
70    ihex_sum += v0;
71    if ((v = ihex_read_1byte()) < 0) {
72      return IHEX_ERR_FORMAT;
73    }
74    v1 = (uint8_t) v;
75    ihex_sum += v1;
76    ihex_addr = ((uint16_t) v0 << 8) | ((uint16_t) v1);
77    /* レコード型 */
78    if ((v = ihex_read_1byte()) < 0) {
79      return IHEX_ERR_FORMAT;
80    }
81    ihex_sum += (uint8_t)v;
82    ihex_rectyp = (uint8_t)v;
83    enum ihex_ret_e r = IHEX_OK;
84    switch (ihex_rectyp) {
85    case 0: /* データ, 16ビットアドレッシング */
86      ihex_exaddr = ihex_addr + ihex_base;
87      if (cmd == IHEX_CMD_WRITE) {
88        (*devs[dev].f_write_begin)(dev, ihex_exaddr, ihex_bytcnt);
89      } else if (cmd == IHEX_CMD_VERIFY) {
90        (*devs[dev].f_read_begin)(dev, ihex_exaddr, ihex_bytcnt);
91      }
92      for ( ; ihex_bytcnt != 0; ihex_bytcnt--, ihex_exaddr++) {
93        if (ihex_exaddr >= devs[dev].size_in_byte) {
94          return IHEX_ERR_ADDR;
95        }
96        if ((v = ihex_read_1byte()) < 0) {
97          return IHEX_ERR_FORMAT;
98        }
99        ihex_sum += (uint8_t)v;
100       ihex_data = (uint8_t)v;
101       if (cmd == IHEX_CMD_WRITE) {
102         (*devs[dev].f_write_byte)(dev, ihex_data);
103       } else if (cmd == IHEX_CMD_VERIFY) {
104         uint8_t mem_data = (*devs[dev].f_read_byte)(dev);
105         if (mem_data != ihex_data) {
106           r = IHEX_ERR_VERIFY;
107         }
108       }
109     }
110     if (cmd == IHEX_CMD_WRITE) {
111       (*devs[dev].f_write_done)(dev);
112     } else if (cmd == IHEX_CMD_VERIFY) {
113       (*devs[dev].f_read_done)(dev);
114     }
115     if (r != IHEX_OK) {
116       return r;
117     }
118     break;
119   case 1: /* ファイル終端 */
120     if (ihex_bytcnt != 0) {
121       return IHEX_ERR_FORMAT;
122     }
123     break;
124   case 2: /* 20ビットアドレッシング */
```

```
125     case 4: /* 32ビットアドレッシング */
126       if (ihex_bytcnt != 2) {
127         return IHEX_ERR_FORMAT;
128       }
129       ihex_base = 0;
130       for (i = 0; i < 2; i++) {
131         if ((v = ihex_read_1byte()) < 0) {
132           return IHEX_ERR_FORMAT;
133         }
134         ihex_sum += (uint8_t)v;
135         ihex_base = (ihex_base << 8) | (uint8_t)v;
136       }
137       if (ihex_rectyp == 2) {
138         ihex_base = ihex_base <<  4;
139       } else {
140         ihex_base = ihex_base << 16;
141       }
142       break;
143     case 3: /*未対応*/
144     case 5: /*未対応*/
145       return IHEX_ERR_UNSUPP;
146     }
147     /* チェックサム */
148     if ((v = ihex_read_1byte()) < 0) {
149       return IHEX_ERR_FORMAT;
150     }
151     ihex_chksum = (uint8_t)v;
152     if (((ihex_chksum + ihex_sum) & 0xFF) != 0) {
153       return IHEX_ERR_CHECKSUM;
154     }
155     /* 改行文字 */
156     if (ihex_read_char() != CHAR_CR) {
157       return IHEX_ERR_FORMAT;
158     }
159     /* 1行処理終了 */
160     if (ihex_rectyp == 1) {
161       return IHEX_EOF;
162     }
163     return IHEX_OK;
164   }
165
166   /* 16進数1文字を数値へ */
167   int16_t ihex_read_1byte(void)
168   {
169     int16_t value = 0;
170     uint8_t n;
171     for  (n = 0; n < 2; n++) {
172       char ch = ihex_read_char();
173       int8_t v4 = 0;
174       if (('0' <= ch) && (ch <= '9')) {
175         v4 = ch - '0';
176       } else if (('A' <= ch) && (ch <= 'F')) {
177         v4 = ch - 'A' + 10;
178       } else if (('a' <= ch) && (ch <= 'f')) {
179         v4 = ch - 'a' + 10;
180       } else {
181         value = -1;
182         break;
183       }
184       value = (value << 4) + v4;
185     }
186     return value;
187   }
188
189   /* 1文字読み込み & 改行までの読み飛ばし */
190
191   enum ihex_char_pos_e {
192     IHEX_CHAR_POS_MOL = 0, /* 行中 */
193     IHEX_CHAR_POS_EOL      /* 行末 */
194   };
195
```

```
196 static uint8_t ihex_read_newline = IHEX_CHAR_POS_MOL;
197
198 static void ihex_read_start(void)
199 {
200   ihex_read_newline = IHEX_CHAR_POS_MOL;
201 }
202
203 static void ihex_skip_to_newline(void)
204 {
205   while (ihex_read_newline != IHEX_CHAR_POS_EOL) {
206     ihex_read_char();
207   }
208 }
209
210 static char ihex_read_char(void)
211 {
212   char ch = CHAR_CR;
213   if (ihex_read_newline == IHEX_CHAR_POS_MOL) {
214     int16_t c = 0;
215     while ((c = cons_getc()) < 0) {
216       CLRWDT();
217     }
218     ch = (char)c;
219     if ((ch == CHAR_CR) || (ch == CHAR_LF)) {
220       ihex_read_newline = IHEX_CHAR_POS_EOL;
221       ch = CHAR_CR;
222     }
223   }
224   return ch;
225 }
```

ソースファイル cons.h (コンソール関数: ヘッダ)

後述のソースファイル cons.c で実装する関数群のプロトタイプ宣言です.

ソースファイル番号 25.9: cons.h

```
1  #if !defined(__CONS_H__)
2  #define __CONS_H__
3
4  extern int16_t cons_getc(void);
5  extern int8_t  cons_get_ready(void);
6  extern void    cons_putc(char ch);
7  extern int8_t  cons_put_busy(void);
8  extern void    cons_print_newline(void);
9  extern void    cons_print_hex(uint8_t d);
10 extern void    cons_print(char *s);
11 extern void    cons_print_pgm(const char *s);
12
13 #define CHAR_LF     0x0A
14 #define CHAR_CR     0x0D
15
16 #endif /*__CONS_H__*/
```

ソースファイル cons.c (コンソール関数: 実装)

後述のソースファイル com.c による RS-232C 通信関数群を利用して, 送受信を便利にする関数群の実装です. 受信の際に使用されるされる改行コードが CRLF と CR のどちらでも正しく動作するようにしています. この動作は CR 直後の LF は捨てるという方法で実現しています.

ソースファイル番号 25.10: cons.c

```
1  /*-------------------------------------------------------
2   * コンソール
3   *   com インターフェースと通信
4   *-------------------------------------------------------*/
5  #include <xc.h>
```

```
6  #include <stdint.h>
7  #include "cons.h"
8  #include "com.h"
9
10 /* 前回に受信しした文字 (CRLF -> CR 変換用) */
11 static char cons_ch_prev = 0;
12
13 /* 1文字受信 (受信文字がなければ-1) : CRLF -> CR 変換*/
14 /*   注意: ノンブロッキング動作 */
15 int16_t cons_getc(void)
16 {
17   int16_t ch = com_rx_get();
18   if (ch >= 0) {
19     /* 受信文字あり */
20     if ((cons_ch_prev == CHAR_CR) &&
21         (ch == CHAR_LF)) { /* CRLFのLFは捨てる */
22       ch = com_rx_get();
23     }
24     if (ch >= 0) {
25       cons_ch_prev = (char)ch;
26     }
27   } else {
28     /* 受信文字なし */
29     ch = -1;
30   }
31   return ch;
32 }
33
34 /* 受信バッファに文字があるか否かを検査 */
35 int8_t cons_get_ready(void)
36 {
37   int16_t n = com_rx_nchars();
38   if ((cons_ch_prev == CHAR_CR) &&
39       (n > 0) && (com_rx_peek() == CHAR_LF)) {
40     /* CRLFのLFは捨てるので勘定外 */
41     return (n > 1);
42   }
43   return (n > 0);
44 }
45
46 /* 1文字送信 */
47 void cons_putc(char ch)
48 {
49   while (com_tx_busy()) {
50     CLRWDT();
51   }
52   com_tx_put(ch);
53 }
54
55 /* 送信中を検査 */
56 int8_t cons_put_busy(void)
57 {
58   return com_tx_busy();
59 }
60
61 /* 改行(CR+LF) */
62 void cons_print_newline(void)
63 {
64   cons_putc(CHAR_CR);
65   cons_putc(CHAR_LF);
66 }
67
68 /* 8ビットを16進数2ケタで表示 */
69 void cons_print_hex(uint8_t d)
70 {
71   const char hexchars[] = "0123456789ABCDEF";
72   cons_putc(hexchars[(unsigned)(d >> 4) & 0x0F]);
73   cons_putc(hexchars[(unsigned) d       & 0x0F]);
74 }
75
76 /* 文字列表示（文字列はデータ領域） */
```

```
77 void cons_print(char *s)
78 {
79   for (; *s != '\0'; s++) {
80     cons_putc(*s);
81   }
82 }
83
84 /* 文字列表示（文字列はプログラムコード領域） */
85 void cons_print_pgm(const char *s)
86 {
87   for (; *s != '\0'; s++) {
88     cons_putc(*s);
89   }
90 }
```

ソースファイル com.h (RS-232C 通信関数: ヘッダ)

後述のソースファイル com.c で実装する関数群のプロトタイプ宣言です. 送信と受信のバッファサイズを定義します. また受信バッファ内の文字数が 50% に相当する 40 文字を越えれば, Xoff を送信するように定義しています. 通信のボーレートを指定する記号定数も定義します.

ソースファイル番号 25.11: com.h

```
1  #if !defined(__COM_H__)
2  #define __COM_H__
3
4  #define COM_TX_BUFSIZ        80  /* 送信バッファサイズ */
5  #define COM_RX_BUFSIZ        80  /* 受信バッファサイズ */
6  #define COM_RX_BUFSIZ_XOFF   40  /* 受信時 XON/OFF しきい値 */
7
8  /* ボーレート */
9  enum com_baud_rate_e {
10   COM_BAUD_RATE_NULL,
11   COM_BAUD_RATE_1200,
12   COM_BAUD_RATE_2400,
13   COM_BAUD_RATE_4800,
14   COM_BAUD_RATE_9600,
15   COM_BAUD_RATE_19200,
16   COM_BAUD_RATE_38400
17 };
18
19 extern void    com_init(void);
20 extern void    com_tx_put(char ch);
21 extern int8_t  com_tx_busy(void);
22 extern void    com_tx_isr(void);
23 extern int16_t com_rx_nchars(void);
24 extern int16_t com_rx_get(void);
25 extern int16_t com_rx_peek(void);
26 extern void    com_rx_isr(void);
27 extern void    com_baud_rate_set(enum com_baud_rate_e com_baud_rate);
28
29 #endif /*__COM_H__*/
```

ソースファイル com.c (RS-232C 通信関数: 実装)

RS-232C 通信を実現する関数群の実装で, PIC マイコンの EUSART モジュールを使用して通信を行います. 送信と受信にはそれぞれバッファを用意して, 割り込み駆動により連続的に送信と受信を行います. 送信と受信それぞれに第 25.5 節で紹介した巡回バッファを使用します.

割り込み駆動による送信の実現方法の概略は以下の通りです.

- 送信手続き
  - 送信バッファが空でない場合

    送信バッファにデータを書き込みます.

  - バッファが空の場合
    送信データレジスタにデータを書き込んで送信を始動します. また送信完了時に割り込みが発生するように設定します.
- 送信完了割り込みハンドラ
  - 送信バッファにデータがない場合
    送信完了時の割り込みが発生しないように設定します.
  - 送信バッファにデータがある場合
    送信バッファからデータを取り出し, 送信データレジスタにデータを書き込んで送信を始動します.

割り込み駆動による受信の実現方法の概略は以下の通りです.

- 受信手続き
  受信バッファが空ならば –1 を返し, そうでなければ受信バッファから 1 文字を取り出して返します.
- 受信完了割り込みハンドラ
  受信データレジスタからデータを読み出して受信バッファに挿入します.

フロー制御 Xon/Xoff の実現をこのソースファイルで行っています. 受信バッファが溢れそうな場合には, 送信先に送信を一時停止を依頼するというのがこのフロー制御の基本的な考え方です. 送信先が送信を一時停止している間に受信バッファに溜まっているデータを受信して処理をし, それを終えたら送信を再開させます. フロー制御 Xon/Xoff は特殊な文字データを使用して通信の制御をソフトウエアで実現しますから, バイナリデータを直接送信することはできず, テキストファイルのみの送信しかできません. なぜなら Xon/Xoff の実現で送受信する特別な文字データと送信すべきバイナリデータとの判別がつかず, 正常にフロー制御を行えないからです.

- 送信の再開の要求 Xon
  制御文字 DC1 (Device Control 1; ASCII コード 0x11) を送信します.
- 送信の一時停止の要求 Xoff
  制御文字 DC3 (Device Control 3; ASCII コード 0x13) を送信します.

フロー制御 Xon/Xoff の実現は, 上述の送信と受信の動作に以下の動作を追加しています.

- データ受信
  受信バッファが溢れそうな場合は通信相手に Xoff (DC3) を送信して送信を一時停止させます. 受信バッファに空になれば通信相手に Xon (DC1) を送信して送信を再開させます.
- データ送信
  通信相手から Xoff (DC3) を受信した場合は送信を一時停止します. 通信相手から Xon (DC1) を受信した場合は送信を再開します.

Xoff の送信時にもし送信バッファがいっぱいなら送信バッファに Xoff を挿入できず Xoff の送信ができません. これは非常に困りますので, 通常の送信時には送信バッファいっぱいにデータを挿入せず, 容量には余裕を残すようにします. しかし Xon/Xoff を使用しても, 受信バッファがいっぱいの時にさらにデータの受信が発生する場合がまったくないとは言いきれません. その場合は仕方なく受信したデータは廃棄します. これが生じるのは, 送信相手であるターミナルソフトウエアでフロー制御の設定を Xon/Xoff にしていない場合, あるいは Xon/Xoff フロー制御の機能がない場合, Xoff を送信するタイミングが遅い場合などが考えられます.

ソースファイル番号 25.12: com.c

```
/*-------------------------------------------------------
 * RS232C送受信 (XON/XOFFによるフロー制御あり)
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "com.h"
#include "cons.h"

#define CHAR_XON    0x11
#define CHAR_XOFF   0x13

/* 送信バッファ */
static volatile char     com_tx_buff[COM_TX_BUFSIZ];
static volatile uint8_t  com_tx_nitems = 0;
static volatile uint8_t  com_tx_head   = 0;
static volatile uint8_t  com_tx_tail   = 0;
static volatile uint8_t  com_tx_paused = 0;
/* 受信バッファ */
static volatile char     com_rx_buff[COM_RX_BUFSIZ];
static volatile uint8_t  com_rx_nitems = 0;
static volatile uint8_t  com_rx_head   = 0;
static volatile uint8_t  com_rx_tail   = 0;
static volatile uint8_t  com_rx_paused = 0;

/* ボーレート */
enum com_baud_rate_e com_baud_rate = COM_BAUD_RATE_NULL;

static void com_tx_put_raw(char ch);

/* 初期化 */
void com_init(void)
{
  com_tx_nitems = 0;
  com_tx_head   = 0;
  com_tx_tail   = 0;
  com_tx_paused = 0;
  com_rx_nitems = 0;
  com_rx_head   = 0;
  com_rx_tail   = 0;
  com_rx_paused = 0;
  com_tx_put(CHAR_XON);
}

/* ボーレート設定 */
void com_baud_rate_set(enum com_baud_rate_e next_baud_rate)
{
  if (com_baud_rate != next_baud_rate) {
    com_baud_rate = next_baud_rate;
    switch (com_baud_rate) {
    default:
    case COM_BAUD_RATE_NULL:
      SP1BRGH = 0x00; SP1BRGL = 0x00; break;
    case COM_BAUD_RATE_4800:
      SP1BRGH = 0x06; SP1BRGL = 0x81; break;
    case COM_BAUD_RATE_9600:
      SP1BRGH = 0x03; SP1BRGL = 0x40; break;
    case COM_BAUD_RATE_19200:
      SP1BRGH = 0x01; SP1BRGL = 0xA0; break;
    case COM_BAUD_RATE_38400:
      SP1BRGH = 0x00; SP1BRGL = 0xCF; break;
    }
  }
}

/* 送信中か */
int8_t com_tx_busy(void)
{
  int8_t nchars = 0;
  int8_t ogie  = GIE;
  int8_t opeie = PEIE;
```

```
71    GIE  = 0;
72    PEIE = 0;
73    nchars = com_tx_nitems;
74    GIE  = ogie;
75    PEIE = opeie;
76    return (nchars != 0);
77  }
78
79  /* 送信 */
80  void com_tx_put(char ch)
81  {
82    int8_t ogie  = GIE;
83    int8_t opeie = PEIE;
84    GIE  = 0;
85    PEIE = 0;
86    if ((com_tx_paused == 0) &&
87        (com_tx_nitems < COM_TX_BUFSIZ-2)) {
88      /* XON/XOFF送信用に2文字分残しておく */
89      com_tx_put_raw(ch);
90    } /* 送信バッファ溢れ: 廃棄 */
91    GIE  = ogie;
92    PEIE = opeie;
93  }
94
95  /* 送信本体（割り込み禁止状態で呼び出すこと):
96   * 送信文字を送信バッファに挿入 */
97  static void com_tx_put_raw(char ch)
98  {
99    if ((ch == CHAR_XON) ||
100       (ch == CHAR_XOFF)) { /* 送信文字: XON または XOFF */
101     /* バッファの先頭へ挿入 */
102     if (com_tx_head == 0) {
103       com_tx_head = COM_TX_BUFSIZ - 1;
104     } else {
105       com_tx_head--;
106     }
107     com_tx_buff[com_tx_head] = ch;
108   } else { /* 送信文字: XON/XOFF以外（通常時) */
109     /* バッファの末尾へ挿入 */
110     com_tx_buff[com_tx_tail++] = ch;
111     if (com_tx_tail == COM_TX_BUFSIZ) {
112       com_tx_tail = 0;
113     }
114   }
115   com_tx_nitems++;
116   if (com_tx_nitems == 1) {
117     TXIE = 1;
118     TXIF = 1;  /* com_tx_isr()を呼び出す */
119   }
120 }
121
122 /* 送信割込ハンドラ:
123  * 送信バッファから次の文字を取出して送信レジスタへ */
124 void com_tx_isr(void)
125 {
126   if (com_tx_nitems == 0) {  /* 次の送信文字は無い */
127     TXIE = 0;
128   } else {                   /* 次の送信文字が有る */
129     if (com_tx_paused) {  /* XOFF中 */
130       TXIE = 0;
131     } else {              /* XOFF中ではない(通常時) */
132       com_tx_nitems--;
133       TX1REG = com_tx_buff[com_tx_head++];
134       if (com_tx_head == COM_TX_BUFSIZ) {
135         com_tx_head = 0;
136       }
137     }
138   }
139 }
140
```

```
141 /* 受信: 受信バッファに文字があるか否か */
142 int16_t com_rx_nchars(void)
143 {
144   int16_t nchars = 0;
145   int8_t ogie  = GIE;
146   int8_t opeie = PEIE;
147   GIE  = 0;
148   PEIE = 0;
149   nchars = (com_rx_nitems != 0);
150   GIE  = ogie;
151   PEIE = opeie;
152   return nchars;
153 }
154
155 /* 受信: 受信バッファから１文字取り出して返す (なければ-1) */
156 /*   注意: ノンブロッキング動作 */
157 int16_t com_rx_get(void)
158 {
159   int16_t ch = -1;
160   int8_t ogie  = GIE;
161   int8_t opeie = PEIE;
162   GIE  = 0;
163   PEIE = 0;
164   if (com_rx_nitems == 0) { /* 受信バッファは空 */
165     ch = -1;
166   } else {
167     com_rx_nitems--;
168     ch = com_rx_buff[com_rx_head++];
169     if (com_rx_head == COM_RX_BUFSIZ) {
170       com_rx_head = 0;
171     }
172     if ((com_rx_paused != 0) && (com_rx_nitems == 0)) {
173       /* 受信を再開: XON */
174       com_rx_paused = 0;
175       com_tx_put_raw(CHAR_XON);
176     }
177   }
178   GIE  = ogie;
179   PEIE = opeie;
180   return ch;
181 }
182
183 /* 受信バッファの読み出し(取り出しはしない) */
184 int16_t com_rx_peek(void)
185 {
186   int16_t ch = -1;
187   int8_t ogie  = GIE;
188   int8_t opeie = PEIE;
189   GIE  = 0;
190   PEIE = 0;
191   if (com_rx_nitems == 0) { /* 受信バッファは空 */
192     ch = -1;
193   } else {
194     ch = com_rx_buff[com_rx_head];
195   }
196   GIE  = ogie;
197   PEIE = opeie;
198   return ch;
199 }
200
201 /* 受信割込ハンドラ */
202 void com_rx_isr(void)
203 {
204   /* エラー時はモジュールをリセット */
205   if (RCSTAbits.OERR == 1) {
206     RCSTAbits.CREN = 0;
207     RCSTAbits.CREN = 1;
208   }
209   /* 受信文字を受信バッファへ挿入 */
210   char ch = RC1REG;
211   if (ch == CHAR_XOFF) {
```

```
212       com_tx_paused = 1; /* 送信を一時停止 */
213     } else if (ch == CHAR_XON) {
214       com_tx_paused = 0; /* 送信を再開 */
215       TXIE = 1;
216       TXIF = 1;  /* com_tx_isr()を呼び出す */
217     } else if (com_rx_nitems < COM_RX_BUFSIZ) {
218       /* バッファに受信文字を収容可能 */
219       com_rx_nitems++;
220       com_rx_buff[com_rx_tail++] = ch;
221       if (com_rx_tail == COM_RX_BUFSIZ) {
222         com_rx_tail = 0;
223       }
224       /* XOFF送信が必要? */
225       if ((com_rx_paused == 0) &&
226           (com_rx_nitems == COM_RX_BUFSIZ_XOFF)) {
227         /* 受信を停止: XOFF */
228         com_rx_paused = 1;
229         com_tx_put_raw(CHAR_XOFF);
230       }
231     } else {
232       /* 受信バッファ溢れ: 廃棄 */
233     }
234   }
```

ソースファイル f-24fc64.h (24FC64 インターフェース: ヘッダ)

この後に紹介する f-24fc64.c で実装する関数群のプロトタイプ宣言です. EEPROM デバイス 24FC64 にアクセスする際の汎用的な関数群として i2c-24fc64.h と i2c-24fc64.c を用意しており, これら関数群を本装置のファームウエアで利用可能とするための仲介役となる関数群です.

ソースファイル番号 25.13: f-24fc64.h

```
1  #if !defined(__F_24FC64_H__)
2  #define __F_24FC64_H__
3
4  extern int8_t  f_24fc64_init(uint8_t dev);
5  extern void    f_24fc64_deinit(uint8_t dev);
6  extern void    f_24fc64_erase(uint8_t dev);
7  extern void    f_24fc64_read_begin(uint8_t dev,
8                                     uint32_t addr,
9                                     uint32_t nbytes);
10 extern uint8_t f_24fc64_read_byte(uint8_t dev);
11 extern void    f_24fc64_read_done(uint8_t dev);
12 extern void    f_24fc64_write_begin(uint8_t dev,
13                                      uint32_t addr,
14                                      uint32_t nbytes);
15 extern void    f_24fc64_write_byte(uint8_t dev, uint8_t d);
16 extern void    f_24fc64_write_done(uint8_t dev);
17
18 #endif /*__F_24FC64_H__*/
```

ソースファイル f-24fc64.c (24FC64 インターフェース: 実装)

このソースファイルは EEPROM デバイス 24FC64 用の仲介役関数群の実装です. 変数 dev に保持されている 24FC64 に特有なパラメータ類を参照しながら, i2c-24fc64.c で実装されている汎用的な関数群を呼び出します. 24FC64 に加えて, ソースファイル devinfo.c 内の変数 devs[] で記述してある通り, パラメータの変更により 24LC64, 24FC256, 24LC256, 24FC512, 24LC512 もこのソースファイル中の関数群により読み書きが行われます.

ソースファイル番号 25.14: f-24fc64.c

```
1  #include <xc.h>
2  #include <stdint.h>
```

```
3  #include "mcr.h"
4  #include "io.h"
5  #include "devinfo.h"
6  #include "i2c.h"
7  #include "i2c-24fc64.h"
8  #include "f-24fc64.h"
9
10 extern void erase_by_write(uint8_t dev);
11
12 int8_t f_24fc64_init(uint8_t dev)
13 {
14   I2C_Init(devs[dev].ssp1add_value);
15   OUT_MEM_A0_LOW();
16   OUT_MEM_A1_LOW();
17   OUT_MEM_A2_LOW();
18   OUT_MEM_WP_LOW();
19   __delay_ms(100);
20   return I2C_Exist(I2C_24FC64_CTLBYTE, 0);
21 }
22
23 void f_24fc64_deinit(uint8_t dev)
24 {
25   I2C_Deinit();
26 }
27
28 void f_24fc64_erase(uint8_t dev)
29 {
30   erase_by_write(dev);
31 }
32
33 void f_24fc64_read_begin(uint8_t dev,
34                          uint32_t addr, uint32_t nbytes)
35 {
36   i2c_24fc64_read_begin(0, addr, nbytes);
37 }
38
39 uint8_t f_24fc64_read_byte(uint8_t dev)
40 {
41   return i2c_24fc64_read_byte();
42 }
43
44 void f_24fc64_read_done(uint8_t dev)
45 {
46   i2c_24fc64_read_done();
47 }
48
49 void f_24fc64_write_begin(uint8_t dev,
50                           uint32_t addr, uint32_t nbytes)
51 {
52   i2c_24fc64_write_begin(0, addr, nbytes);
53 }
54
55 void f_24fc64_write_byte(uint8_t dev, uint8_t d)
56 {
57   i2c_24fc64_write_byte(d);
58 }
59
60 void f_24fc64_write_done(uint8_t dev)
61 {
62   i2c_24fc64_write_done();
63 }
```

ソースファイル f-24fc1025.h (24FC1025 インターフェース: ヘッダ)

この後に紹介する f-24fc1025.c で実装する関数群のプロトタイプ宣言です.

ソースファイル番号 25.15: f-24fc1025.h

```
1  #if !defined(__F_24FC1015_H__)
2  #define __F_24FC1015_H__
3
```

```
4  extern int8_t  f_24fc1025_init(uint8_t dev);
5  extern void    f_24fc1025_deinit(uint8_t dev);
6  extern void    f_24fc1025_erase(uint8_t dev);
7  extern void    f_24fc1025_read_begin(uint8_t dev,
8                                       uint32_t addr,
9                                       uint32_t nbytes);
10 extern uint8_t f_24fc1025_read_byte(uint8_t dev);
11 extern void    f_24fc1025_read_done(uint8_t dev);
12 extern void    f_24fc1025_write_begin(uint8_t dev,
13                                        uint32_t addr,
14                                        uint32_t nbytes);
15 extern void    f_24fc1025_write_byte(uint8_t dev, uint8_t d);
16 extern void    f_24fc1025_write_done(uint8_t dev);
17
18 #endif /*__F_24FC1015_H__*/
```

ソースファイル f-24fc1025.c (24FC1025 インターフェース: 実装)

このソースファイルは EEPROM デバイス 24FC1025 用の仲介役関数群の実装です．EEPROM デバイス 24FC1025 にアクセスする際の汎用的な関数群として i2c-24fc1025.h と i2c-24fc1025.c を用意しており，これら関数群を本装置のファームウエアで利用可能とするための仲介役を本ソースファイルの関数が担います．具体的には，変数 dev に保持されている 24FC1025 に特有なパラメータ類を参照しながら i2c-24fc1025.c で実装されている汎用的な関数群を呼び出します．24FC1025 に加えてソースファイル devinfo.c 内の変数 devs[] で記述してある通り，パラメータの変更により 24LC1025 もこのソースファイル中の関数群を使ってアクセスされます．

ソースファイル番号 25.16: f-24fc1025.c

```
1  #include <xc.h>
2  #include <stdint.h>
3  #include "mcr.h"
4  #include "io.h"
5  #include "devinfo.h"
6  #include "i2c.h"
7  #include "i2c-24fc1025.h"
8  #include "f-24fc1025.h"
9
10 extern void erase_by_write(uint8_t dev);
11
12 int8_t f_24fc1025_init(uint8_t dev)
13 {
14   I2C_Init(devs[dev].ssp1add_value);
15   OUT_MEM_A0_LOW();
16   OUT_MEM_A1_LOW();
17   OUT_MEM_A2_HIGH();
18   OUT_MEM_WP_LOW();
19   __delay_ms(100);
20   return I2C_Exist(I2C_24FC1025_CTLBYTE, 0);
21 }
22
23 void f_24fc1025_deinit(uint8_t dev)
24 {
25   I2C_Deinit();
26 }
27
28 void f_24fc1025_erase(uint8_t dev)
29 {
30   erase_by_write(dev);
31 }
32
33 void f_24fc1025_read_begin(uint8_t dev,
34                            uint32_t addr, uint32_t nbytes)
35 {
36   i2c_24fc1025_read_begin(0, addr, nbytes);
37 }
38
```

```
39 uint8_t f_24fc1025_read_byte(uint8_t dev)
40 {
41   return i2c_24fc1025_read_byte();
42 }
43
44 void f_24fc1025_read_done(uint8_t dev)
45 {
46   i2c_24fc1025_read_done();
47 }
48
49 void f_24fc1025_write_begin(uint8_t dev,
50                             uint32_t addr, uint32_t nbytes)
51 {
52   i2c_24fc1025_write_begin(0, addr, nbytes);
53 }
54
55 void f_24fc1025_write_byte(uint8_t dev, uint8_t d)
56 {
57   i2c_24fc1025_write_byte(d);
58 }
59
60 void f_24fc1025_write_done(uint8_t dev)
61 {
62   i2c_24fc1025_write_done();
63 }
```

ソースファイル i2c-24fc64.h (24FC64 アクセス関数: ヘッダ)

この後に紹介する i2c-24fc64.c で実装する関数群のプロトタイプ宣言です.

ソースファイル番号 25.17: i2c-24fc64.h

```
1  #if !defined(__I2C_24FC64_H__)
2  #define __I2C_24FC64_H__
3
4  #define I2C_24FC64_CTLBYTE  0xA0  /* 制御バイト */
5  #define MEMSIZE_24FC64    0x2000  /* メモリ容量  8K [Byte] */
6  #define MEMSIZE_24FC256   0x8000  /* メモリ容量 32K [Byte] */
7  #define MEMSIZE_24FC512  0x10000  /* メモリ容量 64K [Byte] */
8
9  extern void    i2c_24fc64_read_begin(uint8_t caddr,
10                                       uint32_t maddr,
11                                       uint32_t nbytes);
12 extern uint8_t i2c_24fc64_read_byte(void);
13 extern void    i2c_24fc64_read_done(void);
14 extern void    i2c_24fc64_write_begin(uint8_t caddr,
15                                        uint32_t maddr,
16                                        uint32_t nbytes);
17 extern void    i2c_24fc64_write_byte(uint8_t d);
18 extern void    i2c_24fc64_write_done(void);
19
20 #endif /*__I2C_24FC64_H__*/
```

ソースファイル i2c-24fc64.c (24FC64 アクセス関数: 実装)

上述の i2c.c を利用して 24FC64 等にアクセスする関数群の実装です. これら関数では, 図 25.11a に示した制御バイト, 図 25.11b, に示したアドレス指定 3 バイト, 図 25.13 に示したプロトコル, それぞれに従って I^2C バスを制御して EEPROM の読み書きを行います.

EEPROM の書き込み完了の待機はアクノリッジポーリングで行います. ただし書き込みを行うと直ちに書き込み完了の待機はせず, 次のメモリアクセスの時になってアクノリッジポーリングにより書き込みの完了を待機します. 具体的な実装方法を以下に説明します. アクノリッジポーリングは関数 i2c_24fc64_set_addr() の先頭部分で, 制御バイトの送信をし, それに対する ACK が返るまで繰り返す形で行っています. この関数は EEPROM にアドレス設定を行う関数なので, EEPROM にアドレス設定を行う度にアクノリッジポーリングが

行われます. つまり EEPROM へ書き込みした後に直ちに書き込み完了を待機するのではないということです. マイコンが EEPROM に書き込みをした後, 別の処理をし, 次に EEPROM に書き込みや読み出しをする時には, EEPROM にアドレス設定を行うためにマイコンはこの関数を呼び出します. そしてこの時点でアクノリッジポーリングによる書き込み完了の待機をします. この方法は, マイコンが別の処理をしている間に同時並行で EEPROM が内部での書き込み処理を行いますから, マイコンの待機時間を少ないという利点があります.

ソースファイル番号 25.18: i2c-24fc64.c

```
/*********************************************
 * I2C メモリドライバインターフェース:
 *   利用可能デバイス:
 *     24FC64, 24LC64, 24FC256, 24LC256, 24FC512, 24LC512
 *********************************************/
#include <xc.h>
#include <stdint.h>
#include "i2c.h"
#include "i2c-24fc64.h"

/* 書込ページサイズ */
#define I2C_24FC64_PAGESIZE_BITS    (32-1)  /*  32 Byte / Page */
#define I2C_24FC256_PAGESIZE_BITS   (64-1)  /*  64 Byte / Page */
#define I2C_24FC512_PAGESIZE_BITS  (128-1)  /* 128 Byte / Page */

/* Read/Write */
enum i2c_24fc64_rw_e {
  I2C_24FC64_WR = 0,
  I2C_24FC64_RD = 1,
};

static uint8_t  i2c_24fc64_caddr  = 0;
static uint32_t i2c_24fc64_maddr  = 0;
static uint32_t i2c_24fc64_nbytes = 0;

/*-------------------------------------------*/

/* 制御バイト送出 */
static enum i2c_ack_type_e
i2c_24fc64_send_ctl_byte(enum i2c_24fc64_rw_e r_w,
                         uint32_t maddr)
{
  return
    I2C_Write(/* Control Byte */
              I2C_24FC64_CTLBYTE
              /* A0,A1,A2: Chip Select Bits */
              | ((unsigned)i2c_24fc64_caddr << 1)
              /* R/nW */
              | ((r_w == I2C_24FC64_RD) ? 0x01 : 0x00));
}

/* メモリアドレスを設定 */
static void i2c_24fc64_set_addr(uint32_t maddr)
{
 polling_ack_retry:
  I2C_Start();
  if (i2c_24fc64_send_ctl_byte(I2C_24FC64_WR, maddr)
      != I2C_ACKTYPE_ACK) {
    goto polling_ack_retry;
  }
  I2C_Write((unsigned)(maddr >> 8));   /* Addr High Byte */
  I2C_Write((unsigned)(maddr) & 0xFF); /* Addr Low Byte  */
}

/* 読出し開始 */
void i2c_24fc64_read_begin(uint8_t caddr,
                           uint32_t maddr,
                           uint32_t nbytes)
{
  i2c_24fc64_caddr  = (unsigned)caddr & 0x07;
```

```
61    i2c_24fc64_maddr  = maddr;
62    i2c_24fc64_nbytes = nbytes;
63    i2c_24fc64_set_addr(i2c_24fc64_maddr);
64    I2C_Repstart();
65    i2c_24fc64_send_ctl_byte(I2C_24FC64_RD, i2c_24fc64_maddr);
66  }
67
68  /* 1バイト読み出し */
69  uint8_t i2c_24fc64_read_byte(void)
70  {
71    uint8_t d = 0;
72    if (i2c_24fc64_nbytes == 0) { /* 設定バイト数を読出し済み */
73      return 0;
74    }
75    if (i2c_24fc64_nbytes == 1) { /* 最後の１バイト */
76      d = I2C_Read(I2C_ACKTYPE_NOACK);
77    } else {
78      d = I2C_Read(I2C_ACKTYPE_ACK);
79    }
80    i2c_24fc64_maddr++;
81    i2c_24fc64_nbytes--;
82    return d;
83  }
84
85  /* 読出し途中終了 */
86  void i2c_24fc64_read_done(void)
87  {
88    if (i2c_24fc64_nbytes != 0) {
89      I2C_Read(I2C_ACKTYPE_NOACK);
90      i2c_24fc64_nbytes = 0;
91    }
92  }
93
94  /* 書込み開始 */
95  void i2c_24fc64_write_begin(uint8_t caddr,
96                              uint32_t maddr,
97                              uint32_t nbytes)
98  {
99    i2c_24fc64_caddr  = (unsigned)caddr & 0x07;
100   i2c_24fc64_maddr  = maddr;
101   i2c_24fc64_nbytes = nbytes;
102   i2c_24fc64_set_addr(i2c_24fc64_maddr);
103 }
104
105 /* 1バイト書込み */
106 void i2c_24fc64_write_byte(uint8_t d)
107 {
108   if (i2c_24fc64_nbytes == 0) { /* 設定バイト数を書込み済み */
109     return;
110   }
111   I2C_Write(d);
112   i2c_24fc64_maddr++;
113   i2c_24fc64_nbytes--;
114   if (i2c_24fc64_nbytes == 0) { /* 最後の1バイト */
115     I2C_Stop();
116   } else if ((i2c_24fc64_maddr & I2C_24FC64_PAGESIZE_BITS)
117              == 0x0000) {
118     /* ページ境界 */
119     I2C_Stop();
120     i2c_24fc64_set_addr(i2c_24fc64_maddr);
121   }
122   /* 注意: プログラム簡略化のため，どのデバイスでも
123    *       連続書込は32バイトで行っている */
124 }
125
126 /* 書込み途中終了 */
127 void i2c_24fc64_write_done(void)
128 {
129   if (i2c_24fc64_nbytes != 0) {
130     I2C_Stop();
131     i2c_24fc64_nbytes = 0;
```

```
132     }
133 }
```

ソースファイル i2c-24fc1025.h (24FC1025 アクセス関数: ヘッダ)

この後に紹介する i2c-24fc1025.c で実装する関数群のプロトタイプ宣言です.

ソースファイル番号 25.19: i2c-24fc1025.h

```
1  #if !defined(__I2C_24FC1025_H__)
2  #define __I2C_24FC1025_H__
3
4  #define I2C_24FC1025_CTLBYTE  0xA0  /* 制御バイト */
5  #define MEMSIZE_24FC1025   0x20000  /* メモリ容量 128K [Byte] */
6
7  extern void    i2c_24fc1025_read_begin(uint8_t caddr,
8                                         uint32_t maddr,
9                                         uint32_t nbytes);
10 extern uint8_t i2c_24fc1025_read_byte(void);
11 extern void    i2c_24fc1025_read_done(void);
12 extern void    i2c_24fc1025_write_begin(uint8_t caddr,
13                                          uint32_t maddr,
14                                          uint32_t nbytes);
15 extern void    i2c_24fc1025_write_byte(uint8_t d);
16 extern void    i2c_24fc1025_write_done(void);
17
18 #endif /*__I2C_24FC1025_H__*/
```

ソースファイル i2c-24fc1025.c (24FC1025 アクセス関数: 実装)

上述の i2c.c を利用して 24FC1025 にアクセスする関数群の実装です. これら関数では, 図 25.12a に示した制御バイト, 図 25.12b, に示したアドレス指定 3 バイト, 図 25.13 に示したプロトコル, それぞれに従って I^2C バスを制御して EEPROM の読み書きを行います. このソースファイルでもアクノリッジポーリングを実装していますが, その内容は先述の通りです.

ソースファイル番号 25.20: i2c-24fc1025.c

```
1  /*********************************************
2   * I2C メモリドライバインターフェース:
3   *   24FC1025/24LC1025/24AA1025
4   *********************************************/
5  #include <xc.h>
6  #include <stdint.h>
7  #include "i2c.h"
8  #include "i2c-24fc1025.h"
9
10 /*-------------------------------------------*/
11 #define I2C_24FC1025_PAGESIZE_BITS   (128-1)
12
13 enum i2c_24fc1025_rw_e {
14   I2C_24FC1025_WR = 0,
15   I2C_24FC1025_RD = 1,
16 };
17
18 static uint8_t  i2c_24fc1025_caddr  = 0;
19 static uint32_t i2c_24fc1025_maddr  = 0;
20 static uint32_t i2c_24fc1025_nbytes = 0;
21
22 /* 制御バイト送出 */
23 static enum i2c_ack_type_e
24 i2c_24fc1025_send_ctl_byte(enum i2c_24fc1025_rw_e r_w,
25                            uint32_t maddr)
26 {
27   return
28     I2C_Write(/* Control Byte */
29               I2C_24FC1025_CTLBYTE
```

```
30                /* B0:    Block Select Bit */
31                | ((maddr >> 13) & 0x08)
32                /* A0,A1: Chip Select Bits */
33                | ((unsigned)i2c_24fc1025_caddr << 1)
34                /* R/nW */
35                | ((r_w == I2C_24FC1025_RD) ? 0x01 : 0x00));
36  }
37
38  /* メモリアドレスを設定 */
39  static void i2c_24fc1025_set_addr(uint32_t maddr)
40  {
41   polling_ack_retry:
42    I2C_Start();
43    if (i2c_24fc1025_send_ctl_byte(I2C_24FC1025_WR, maddr)
44        != I2C_ACKTYPE_ACK) {
45      goto polling_ack_retry;
46    }
47    I2C_Write((unsigned)(maddr >> 8));   /* Addr High Byte */
48    I2C_Write((unsigned)(maddr & 0xFF)); /* Addr Low Byte  */
49  }
50
51  /* 読出し開始 */
52  void i2c_24fc1025_read_begin(uint8_t  caddr,
53                               uint32_t maddr,
54                               uint32_t nbytes)
55  {
56    i2c_24fc1025_caddr  = (unsigned)caddr & 0x03;
57    i2c_24fc1025_maddr  = maddr;
58    i2c_24fc1025_nbytes = nbytes;
59    i2c_24fc1025_set_addr(i2c_24fc1025_maddr);
60    I2C_Repstart();
61    i2c_24fc1025_send_ctl_byte(I2C_24FC1025_RD, i2c_24fc1025_maddr);
62  }
63
64  /* １バイト読出し */
65  uint8_t i2c_24fc1025_read_byte(void)
66  {
67    uint8_t d = 0;
68    if (i2c_24fc1025_nbytes == 0) { /* 設定バイト数を読出し済み */
69      return 0;
70    }
71    if (i2c_24fc1025_nbytes == 1) { /* 最後の１バイト */
72      d = I2C_Read(I2C_ACKTYPE_NOACK);
73    } else if ((i2c_24fc1025_maddr & 0x00FFFF)
74                == 0x00FFFF) {       /* ブロック境界 */
75      /* ブロック境界を越えた連続読み取り: アドレス再設定 */
76      d = I2C_Read(I2C_ACKTYPE_NOACK);
77      i2c_24fc1025_set_addr(i2c_24fc1025_maddr + 1);
78      I2C_Repstart();
79      i2c_24fc1025_send_ctl_byte(I2C_24FC1025_RD,
80                                 i2c_24fc1025_maddr + 1);
81    } else {                         /* 連続読出し */
82      d = I2C_Read(I2C_ACKTYPE_ACK);
83    }
84    i2c_24fc1025_maddr++;
85    i2c_24fc1025_nbytes--;
86    return d;
87  }
88
89  /* 読出し途中終了 */
90  void i2c_24fc1025_read_done(void)
91  {
92    if (i2c_24fc1025_nbytes != 0) {
93      I2C_Read(I2C_ACKTYPE_NOACK);
94      i2c_24fc1025_nbytes = 0;
95    }
96  }
97
98  /* 書込み開始 */
99  void i2c_24fc1025_write_begin(uint8_t  caddr,
100                               uint32_t maddr,
```

```
                                uint32_t nbytes)
{
  i2c_24fc1025_caddr  = (unsigned)caddr & 0x03;
  i2c_24fc1025_maddr  = maddr;
  i2c_24fc1025_nbytes = nbytes;
  i2c_24fc1025_set_addr(i2c_24fc1025_maddr);
}

/* 1バイト書込み */
void i2c_24fc1025_write_byte(uint8_t d)
{
  if (i2c_24fc1025_nbytes == 0) { /* 設定バイト数を書込み済み */
    return;
  }
  I2C_Write(d);
  i2c_24fc1025_maddr++;
  i2c_24fc1025_nbytes--;
  if (i2c_24fc1025_nbytes == 0) { /* 最後の1バイト */
    I2C_Stop();
  } else if ((i2c_24fc1025_maddr & I2C_24FC1025_PAGESIZE_BITS)
             == 0x0000) {
    /* ページ境界 */
    I2C_Stop();
    i2c_24fc1025_set_addr(i2c_24fc1025_maddr);
  }
}

/* 書込み途中終了 */
void i2c_24fc1025_write_done(void)
{
  if (i2c_24fc1025_nbytes != 0) {
    I2C_Stop();
    i2c_24fc1025_nbytes = 0;
  }
}
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 25.21: mcr.h

```
#if !defined(__MCR_H__)
#define __MCR_H__

#define _XTAL_FREQ 32000000 /* OSC: 8MHz, PLL => 32MHz */

void System_Init(void);
void UART_Init(void);
void I2C_Init(uint8_t ssp1add_value);

#endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います.

ソースファイル番号 25.22: mcr.c

```
/*-------------------------------------------------------
 * MCR設定&周辺モジュール (PIC16F18326)
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "i2c.h"
#include "com.h"
```

```
10
11  /*-------------------------------------------*/
12  /* MCR初期化 */
13  void System_Init(void)
14  {
15    /* 発振器 (FOSC=32MHz) */
16    OSCCON1 = 0x00; /* HFINTOSC with 2x PLL */
17    OSCCON3 = 0x00;
18    OSCEN   = 0x00;
19    OSCFRQ  = 0x06; /* 32MHz */
20    OSCTUNE = 0x00;
21    while (PLLR == 0) {
22      ;
23    }
24    /* I/Oピン */
25    LATA    = 0x00;
26    LATC    = 0x00;
27    ANSELA  = 0x00;
28    ANSELC  = 0x00;
29    WPUA    = 0x00;
30    WPUC    = 0x00;
31    TRISA   = 0x28; /* RA5, RA3 */
32    TRISC   = 0x23; /* RC5:RX, RC1:SDA, RC0:SCL */
33    ODCONA  = 0x00;
34    ODCONC  = 0x00;
35    int8_t ogie = GIE;
36    GIE = 0;
37    PPSLOCK = 0x55;
38    PPSLOCK = 0xAA;
39    PPSLOCKbits.PPSLOCKED = 0;
40    RC0PPS     = 0x18; /* RC0 <- SCL1 */
41    RC1PPS     = 0x19; /* RC1 <- SDA1 */
42    SSP1CLKPPS = 0x10; /* RC0 -> SCL1 */
43    SSP1DATPPS = 0x11; /* RC1 -> SDA1 */
44    RC4PPS     = 0x14; /* RC4 <- TX   */
45    RXPPS      = 0x15; /* RC5 -> RX   */
46    PPSLOCK = 0x55;
47    PPSLOCK = 0xAA;
48    PPSLOCKbits.PPSLOCKED = 1;
49    GIE = ogie;
50    /* 周辺モジュール */
51    UART_Init();
52  }
53
54  /*-------------------------------------------*/
55  /* UART */
56  void UART_Init(void)
57  {
58    RCIE = 0;
59    TXIE = 0;
60    BAUD1CONbits.BRG16 = 1; /* BAUD1CON = 0x08 */
61    RC1STAbits.SPEN    = 1; /* RC1STA = 0x90 */
62    RC1STAbits.CREN    = 1;
63    TX1STAbits.TXEN    = 1; /* TX1STA = 0x24 */
64    TX1STAbits.SYNC    = 0;
65    TX1STAbits.BRGH    = 1;
66    RCIE = 1;
67  }
68
69  /*-------------------------------------------*/
70  /* 割り込みルーチン */
71  __interrupt() void isr(void)
72  {
73    if ((TXIE == 1) && (TXIF == 1)) {
74      TXIF = 0;
75      com_tx_isr();
76    }
77    if ((RCIE == 1) && (RCIF == 1)) {
78      RCIF = 0;
79      com_rx_isr();
80    }
81  }
```

```
82
83  /*-------------------------------------------*/
84  /* CONFIG1 */
85  #pragma config FEXTOSC  = OFF
86  #pragma config RSTOSC   = HFINT32
87  #pragma config CLKOUTEN = OFF
88  #pragma config CSWEN    = ON
89  #pragma config FCMEN    = ON
90  /* CONFIG2 */
91  #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
92  #pragma config PWRTE    = OFF
93  #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
94  #pragma config LPBOREN  = OFF
95  #pragma config BOREN    = ON
96  #pragma config BORV     = LOW
97  #pragma config PPS1WAY  = ON
98  #pragma config STVREN   = ON
99  #pragma config DEBUG    = OFF
100 /* CONFIG3 */
101 #pragma config WRT      = OFF
102 #pragma config LVP      = OFF /* 低電圧書込不可 */
103 /* CONFIG4 */
104 #pragma config CP       = OFF
105 #pragma config CPD      = OFF
```

課題

1. Microchip Technology 社の I^2C EEPROM 24FC128, 24FC32, 24FC16 が利用可能となるようにファームウエアを拡張して下さい.
2. Atmel 社の I^2C EEPROM を書き込めるようにファームウエアを拡張して下さい.
3. SPI EEPROM の書き込みも行えるように書き込みアダプタを設計して下さい.
4. 上の課題での SPI EEPROM 書き込みアダプタを使用して, SPI EEPROM の書き込みも行えるようにファームウエアを拡張して下さい.
5. インテル HEX ファイルの代わりに Kermit プロトコルを用いたアップロードとダウンロードができるようにファームウエアを変更して下さい.

第 26 章

サウンド EEPROM データの生成

本章では EEPROM メモリ型 PCM サウンド再生器 (第 27 章) で使用するサウンドデータを作成するソフトウエアを紹介します. 詳しくは後述しますが, EEPROM メモリ型 PCM サウンド再生器では表 26.1 に示す通りの独自のデータ形式のサウンドデータを EEPROM に書き込んで再生します. 一連の作業の全体像の中での本章の位置づけを図 26.1 に示します.

本章で紹介するソフトウエアは図 26.2 に示すように Unix 系オペレーティングシステム (Linux や FreeBSD など) あるいは Microsoft 社 Windows オペレーティングシステムなどのパーソナルコンピュータ上で実行するものであり, PIC マイコン用のファームウエアではありません. 従って本ソフトウエアは開発時に使用するものであり, 開発環境の一部をなすものです.

本章で紹介するソフトウエアの特徴は以下の通りです.

- WAV ファイルからサウンド EEPROM データを生成
  広く一般的に使われている WAV ファイル形式のオーディオファイルから本書の独自形式へと変換します.
- 標準 C 言語で記述した可搬性の高いコンソールアプリケーション
  特定のオペレーティングシステムや GUI に依存することなく様々なコンピュータ環境で永きに渡って動作可能です.

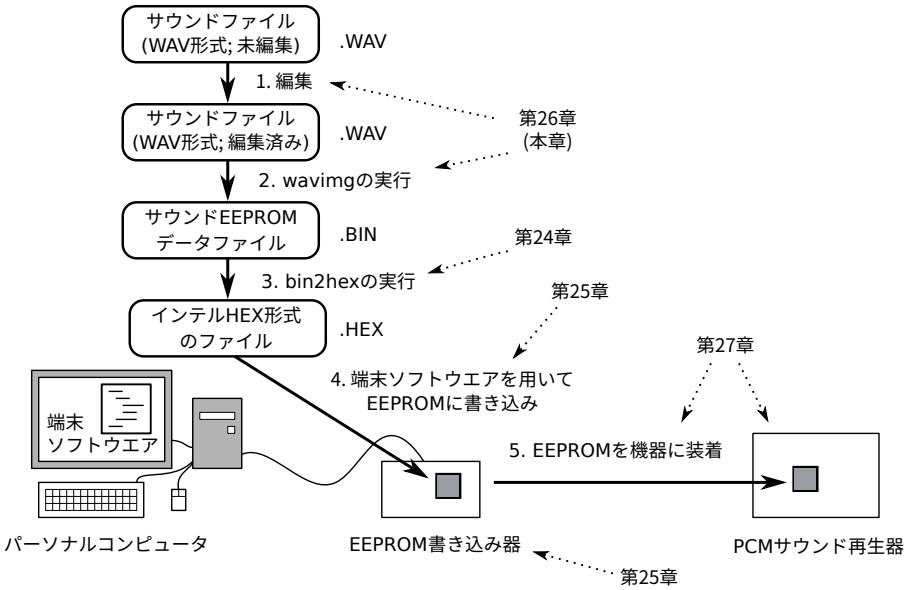


図 26.1: PCM サウンド再生器で使用するサウンド EEPROM データの生成の作業手順

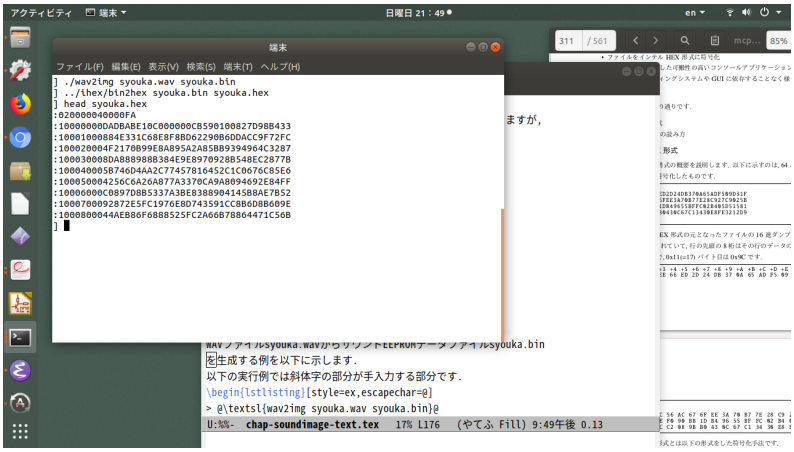


図 26.2: wav2img 使用中の様子 (Ubuntu 上)

表 26.1: PCM サウンドデータ仕様

パラメータ項目	値
サンプリング周波数	8kHz
量子化ビット数	8 ビット
チャンネル数	1 (モノラル)

本章で学ぶ内容は以下の通りです.

- WAV ファイル形式

本装置ではメモリ容量 1M ビットの EEPROM 24FC1025 を使用し, 表 26.1 に示す通り PCM サウンドのパラメータはサンプリング周波数 8kHz, 量子化ビット数 8 ビット (1 バイト), チャンネル数 1 (モノラル) しています. メモリ容量 1M ビットをバイト数に換算すると $1024 \times 1024/8 = 131072 = 128$K バイトで, 1 秒間のデータ量は $8000 \times 1 \times 1 = 8000$ バイトなので, 全体で記録できる音声は最大で $131072/8000 = 16.384$ 秒です.

26.1　使用方法

作業全体の流れは以下の通りで, 図 26.3 に概要を図示しています.

1. 元となる音源ファイルを用意
   PCM レコーダで録音するなどの方法でデジタル化された音源を用意します. 動画サイトなどでの音源を利用する場合にはライセンスや著作権を守った利用をして下さい.
2. 音源ファイルの編集
   オーディオエディタを使用して音源ファイルを編集して WAV ファイルを作成します. オーディオエディタには様々なフリーソフトウエアがありますから各自で使いやすいものを選んでください. 例えば Linux, MacOS, Windows のいずれでも利用可能な Audacity が挙げられます.

   Audacity

   `https://www.audacityteam.org/` (2019 年 6 月 23 日閲覧)

   第 26.2 節では Audacity を使った編集作業の例を説明しています. なお WAV ファイル保存の際のパラメータは, サンプリング周波数 8kHz, 量子化ビット数 8 ビット, チャンネル数 1 のモノラルとして下さ

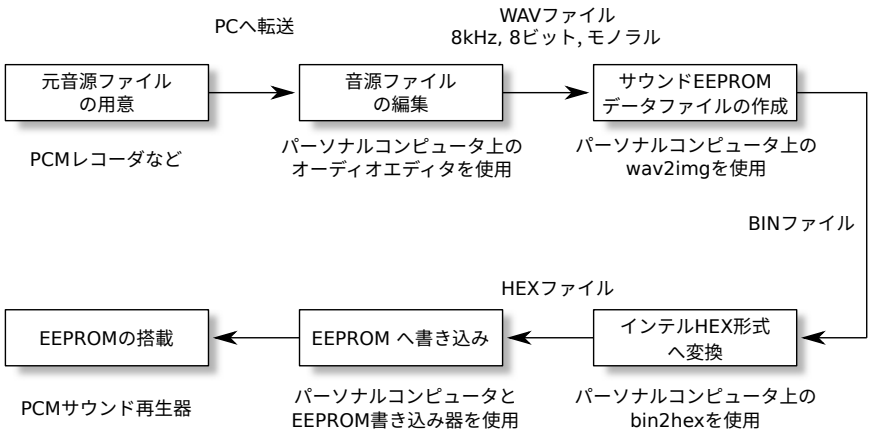


図 26.3: サウンド EEPROM データ作成の作業手順の詳細

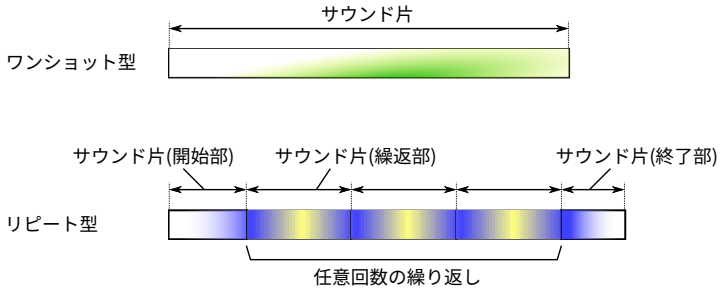


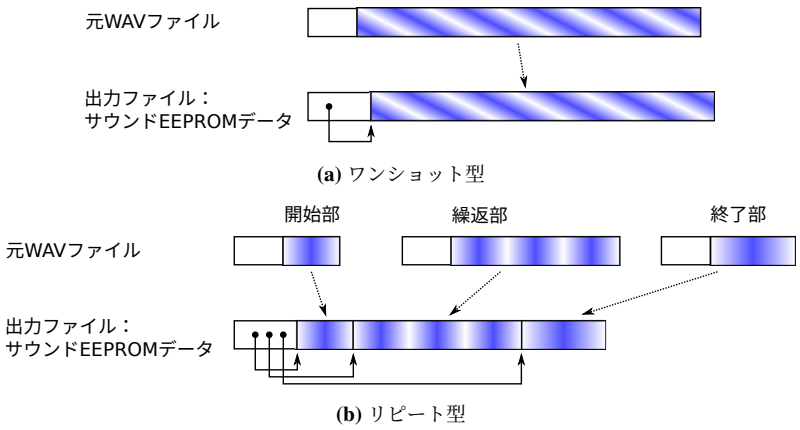
図 26.4: サウンド再生の 2 つの種類: ワンショット型とリピート型

い. wav2img はこれらと違うパラメータの WAV ファイルを受け付けません.

3. サウンド EEPROM データの作成
   本章で紹介するソフトウエア wav2img を使用して, WAV ファイルからサウンド EEPROM データ形式のファイルを作成します.
4. インテル HEX ファイルの作成
   第 24 章で紹介したソフトウエア bin2hex を使用して, 作成したサウンド EEPROM データ形式のファイルをインテル HEX 形式へ変換します.
5. EEPROM への書き込み
   第 25 章で紹介した EEPROM 書き込み器を使用して, インテル HEX 形式のサウンド EEPROM データ形式のファイルを EEPROM に書き込みます.
6. EEPROM の装着
   第 27 章で紹介する EEPROM メモリ型 PCM サウンド再生器に EEPROM を装着すれば準備はすべて完了です.

第 27 章で紹介する EEPROM メモリ型 PCM サウンド再生器のサウンド EEPROM データ形式には以下の 2 種類があり, 本章のソフトウエアを使用して生成を行います. 図 26.4 にその 2 種類のイメージを示します.

- ワンショット型
  PCM サウンドデータは 1 部だけで構成され, 初めから終わりまでを再生して終了します. 例えば発車メロディ, 鉄道唱歌やハイケンスのセレナーデなどのオルゴール曲などがこの型に該当します.



(a) ワンショット型

(b) リピート型

図 26.5: サウンド EEPROM データのファイル構造の模式図

- リピート型
  PCM サウンドデータは開始部, 繰返部, 終了部の 3 部で構成されます. 最初に開始部を再生し, 繰返部を 0 回以上の任意の回数繰り返して再生し, そして終了部を再生して終了します. 例えば踏切の音, 汽笛の音などがこの型に該当します.

本書で独自のサウンド EEPROM データ形式を使用する理由はリピート型を実現するためです. DCC サウンドデコーダでは任意に長く汽笛を吹鳴する機能がありますが, それと同じ機能をリピート型で実現します.

本章で紹介するサウンド EEPROM データ生成ソフトウエア wav2img の使い方を説明します. このプログラムはコマンドライン上で動作しますので, 実行の際には wav2img へ実行パスが通るようにして下さい. 指定された元 WAV ファイルの内容と出力するサウンド EEPROM データの関係の模式図を図 26.5 に示します.

ワンショット型

WAV ファイル syouka.wav からサウンド EEPROM データファイル syouka.bin を生成する例を以下に示します. 以下の実行例では斜体字の部分が手入力する部分です.

```
> wav2img syouka.wav syouka.bin
```

コマンドラインの第 1 引数は WAV ファイルで, 第 2 引数は出力となるサウンド EEPROM データファイルです. なお第 2 引数で指定されたファイルがすでに存在していればエラーとなります. これはファイル名の指定を間違って重要な PCM サウンドファイルを上書きしてしまわないための機能です. 出力ファイルがすでに存在する場合にはファイルを削除したうえで実行して下さい.

リピート型

3 つの WAV ファイル bell1.wav, bell2.wav, bell3.wav からサウンド EEPROM データファイル bell.bin を生成する例を以下に示します.

```
> wav2img bell1.wav bell2.wav bell3.wav bell.bin
```

コマンドラインの第 1 引数は開始部の WAV ファイル, 第 2 引数は繰返部の WAV ファイル, 第 3 引数は終了部の WAV ファイル, 第 4 引数は出力となるサウンド EEPROM データファイルです. なお第 4 引数で指定されたファイルがすでに存在していればエラーとなるのはワンショット型の場合と同じです.

wav2img の大まかな構造を以下に示します. コマンドラインで指定された WAV ファイルから PCM サウンドデータを抜き出して, ファイルに出力するだけの単純な構造です.

```
1  main()
2  {
3    if (リピート型の生成) {
4      リピート型のサウンドEEPROMデータのヘッダを出力;
5      WAVファイル(開始部)のPCMサウンドデータを抜き出して出力;
6      WAVファイル(繰返部)のPCMサウンドデータを抜き出して出力;
7      WAVファイル(終了部)のPCMサウンドデータを抜き出して出力;
8    } else {
9      ワンショット型のサウンドEEPROMデータのヘッダを出力;
10     WAVファイルのPCMサウンドデータを抜き出して出力;
11   }
12 }
```

26.2　音源ファイルの編集

サウンドエディタで音源ファイルを編集する際の注意点を説明します. ワンショット型の場合は音源ファイルの編集は前後の不要な無音部分の切り取り程度ですから編集作業は比較的簡単です. しかしリピート型の場合は音源ファイルの編集の際に注意を要する点があるので以下で説明します.

リピート型ではオーディオエディタを使用してひとつの音源を開始部, 繰返部, 終了部の 3 つに分解・編集して, 音源ファイルを切り出します. この時に重要なのが, 開始部, 繰返部, 終了部を繋げたときに自然な音声波形となるように切り出しをしなくてはならないという点です. もし切り出しが不適切だと再生した時に著しく不快な音が含まれてしまいます. その理由を図 26.6 で使って説明します. 図 26.6 の上の部分が元の波形です. 図 26.6 の中の部分のように切り出し区間がちょうど波形の周期の倍数であれば, 切り出しを部分を繰り返したときの波形はきれいに繋がるため自然な音となります. しかし図 26.6 の下の部分のように切り出し区間が波形の周期でなければ, 繰り返し波形はきれいに繋がらずに元の音とはずいぶん異なる不快な音になってしまいます. この理由は元の波形には含まれない周波数成分が繰り返し波形に含まれてしまうことに起因します. そこでオーディオエディタでの編集では波形の周期を単位として切り出すようにします. オーディオエディタで時間軸を拡大して波形を表示し, 波形の 1 周期の始まりと終わりで音圧レベルが 0 になるところで切り出すのが確実な方法です.

繰返部の繰り返しの繋がりだけでなく他の繋がりも考慮する必要があります. リピート型での再生では繰返部の再生が 0 回あるいは 1 回以上の場合がありますから, 考慮すべき繋がりの組み合わせは開始部 – 繰返部, 開始部 – 終了部, 繰返部 – 繰返部, 繰返部 – 終了部の 4 通りです.

実際の音源ファイルの編集例

音源ファイルの編集の様子を, オーディオエディタ Audacity を使用した場合での概要を示します. 細かな操作のすべては説明しきれませんので各自で調べて下さい. ここでは例としてスマートホンで録音した踏切の音の音源ファイルを基にしてリピート型の WAV ファイル 3 部を作る場合を示します. なお使用機材は iPhone6 と外付けマイクロフォンで, 京福電鉄嵐山本線西院駅前 (京都市中京区) の電鐘式踏切警報機の音を録音しました.

1. 音源ファイルはパーソナルコンピュータ上にあらかじめコピーしておきます.
2. オーディオエディタ Audacity を起動して, 音源ファイルを開いたときの画面が図 26.7 です. このように画面中央に波形が表示されますので, 時間軸の縮尺を拡大・縮小しながら波形の切り出す箇所を見つけ

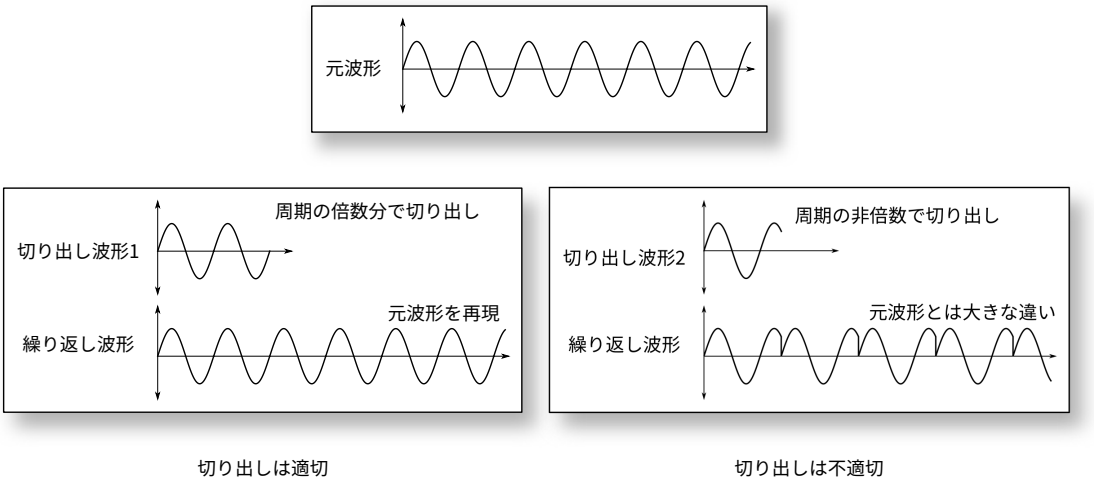


図 26.6: 音源ファイルの切り出しとその繰り返し

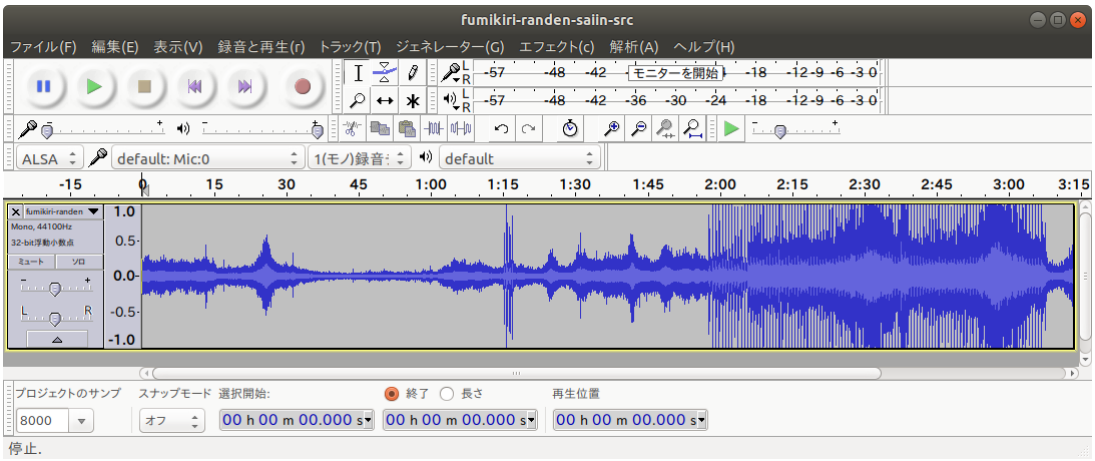


図 26.7: Audacity 起動時の画面

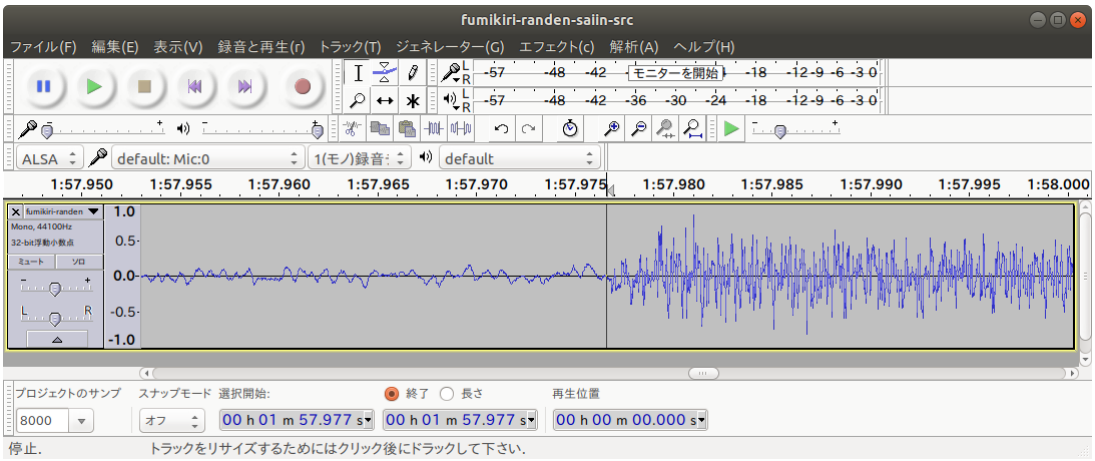


図 26.8: 切り分けの箇所 (時間軸の拡大による表示)

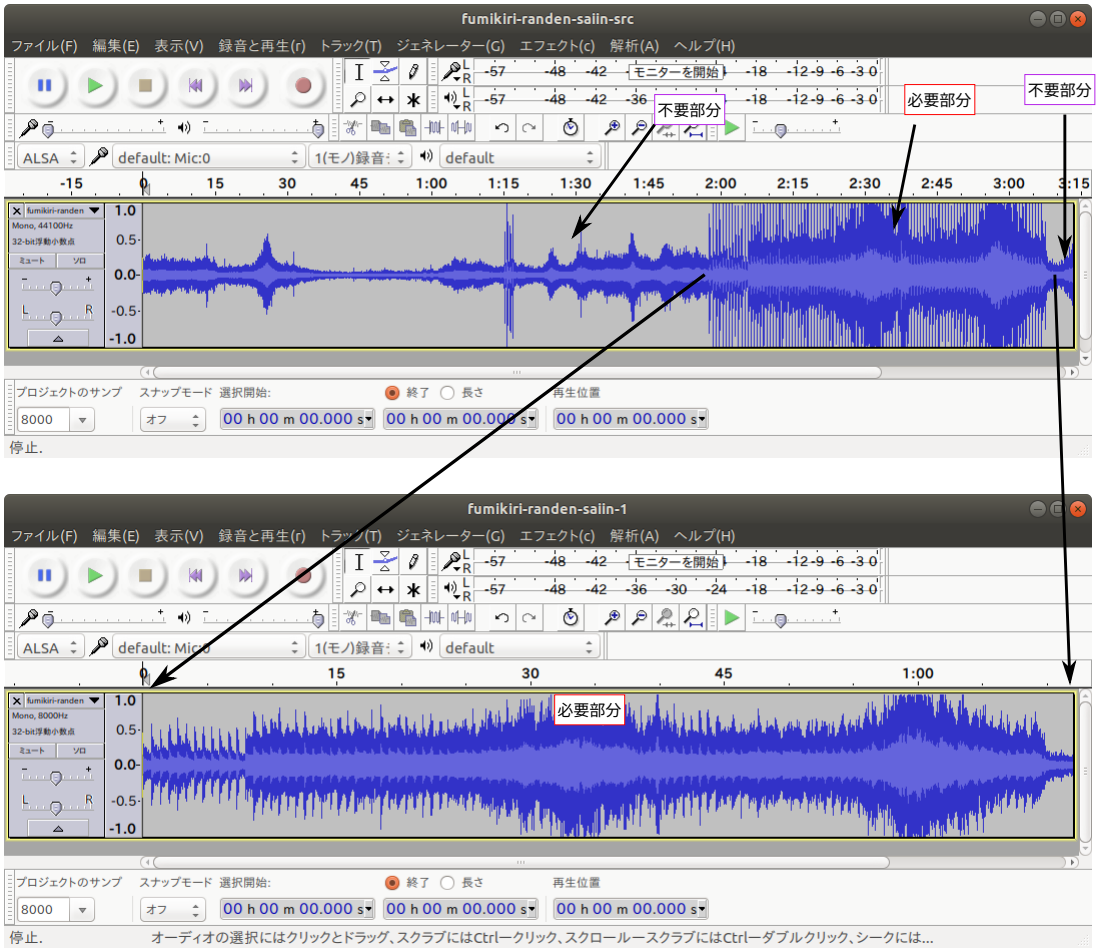


図 26.9: 前後の不要部分を切り捨てて必要部分だけを残す

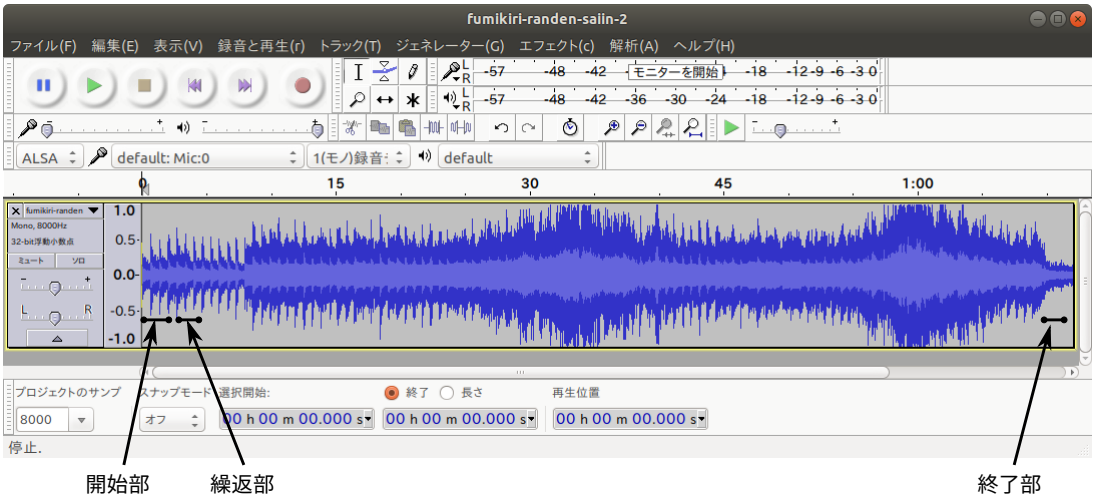


図 26.10: 編集箇所 (切り出す場所の目星をつける)

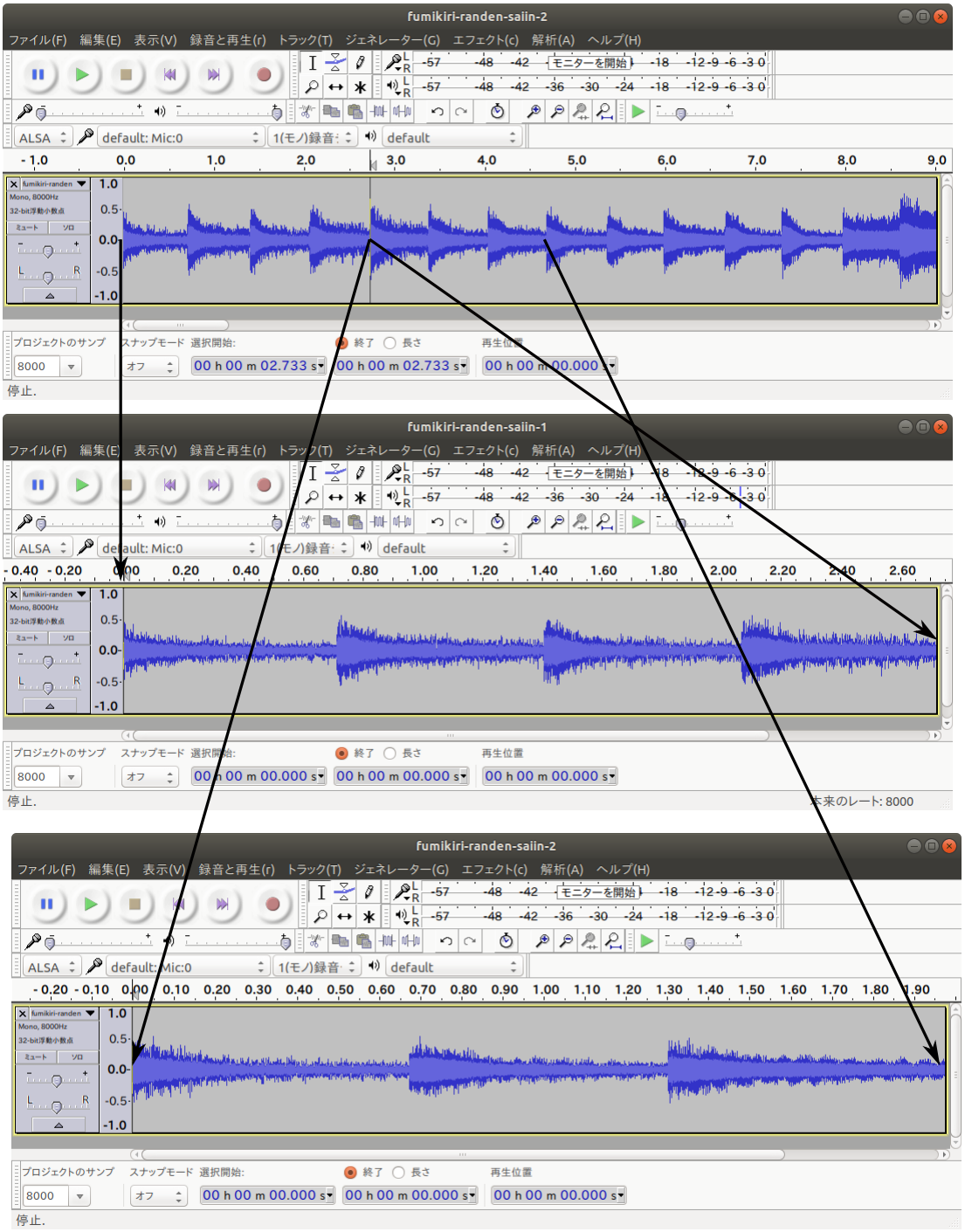


図 26.11: 開始部と繰返部の切り出し (上: 元, 中: 開始部, 下: 繰返部)

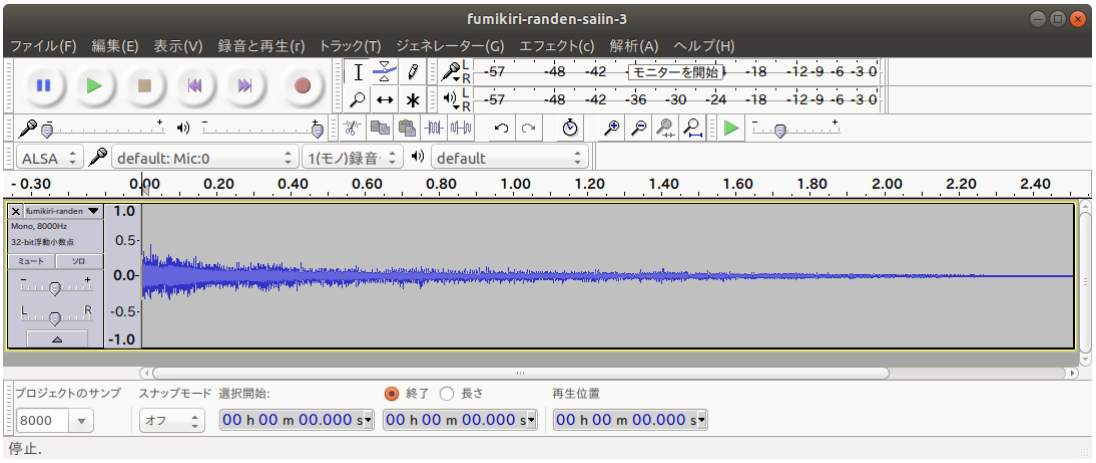


図 26.12: 終了部の切り出し

て編集していきます. この画面での時間軸の縮尺は音源の全体を見るには良いのですが, 波形を切り出し箇所を精密に決めるには更に時間軸の拡大表示が必要です.

3. 図 26.8 が時間軸の拡大表示した例で, 無音から有音に切り替わる箇所が見て取れます. 他の波形と綺麗に繋げるために, 音圧レベルがちょうどゼロのところで切り出しを行います.
4. 編集作業の最初には, 全体での前後にある不要部分を切り捨てます. 踏切が鳴り始めるタイミングはあらかじめ分からず早めに録音を開始したため, 最初の部分にはずいぶんと不要な音が入っています. また踏切が鳴り終えた後もしばらくの間は録音を続けていましたので, 最後の部分にも不要な音が入っています.
5. 踏切の音が鳴っている部分以外を切り捨てたものを図 26.9 に示します. この状態で音源データをファイルに保存して, そのファイルを 3 つにコピーします.
6. 3 つのコピーそれぞれがリピート型の WAV ファイルの開始部, 繰返部, 終了部となるように, 不要箇所を切り捨てて必要箇所のみを残すようにします. 繰返部は何度も繰返し再生されることから, 音源のすべてを使用する必要はなく, 一部を切り出すだけで充分です.
   例として用いた電鐘式踏切警報機の音源ファイルでは, 切り出す場所を図 26.10 に示す通りの 3 箇所としました. 具体的には, 開始部には鳴りはじめの鐘の音を数回, 繰返部には繰り返されている途中での鐘の音を数回, 終了部には鳴り終わりの最後の鐘の音を 1 回, としました. 電子音式踏切警報機の場合は鐘の音を 1 回だけを切り出して繰り返し再生しても問題はないのですが, それではリアルさを失うように思います. 電鐘式踏切警報機では鐘を叩く強さに若干のばらつきがありますが, 鐘の音を 1 回だけを切り出して繰り返し再生するのではそのばらつきが失われ均一な鐘の音になってしまうからです. 音源の特徴が残るように編集すると良いでしょう.
7. 図 26.11 に開始部と繰返部の切り出し箇所を示します. 鐘の音のひとつひとつに若干のばらつきがあることがこの図の波形からも見て取れます.
8. 図 26.12 に終了部の切り出し結果を示します. 元の音源には周辺の自動車の音など不要な雑音が多く含まれていたので, Audacity の編集機能のひとつであるフェードアウト機能を使用して徐々に音量が小さくなるようにしました.

上記の例では音源ファイルのパラメータを, サンプリング周波数を 8000Hz に, 量子化ビット数を 8 ビットに

変換してから編集を行っていました. しかし音源によっては, 図 26.6 で説明した音の周期を見つけ出すことが困難な場合もあります. その場合には, 音源ファイルは録音時のままのパラメータで編集するといいでしょう. 例えば, サンプリング周波数を 44.1kHz, 量子化ビット数を 16 ビット, などの情報量を保てば波形を詳細に観察できるので, 切り出し場所を見つけ出すのが楽になります.

編集した音源ファイルの確認方法

上述の方法で編集した音源ファイルが適切に切り出されているかを確認する方法を説明します. ひとつの方法はサウンド EEPROM に書き込んで, PCM サウンド再生器で実際に再生してみることです. しかし編集に慣れていないうちは適切な切り出し箇所がなかなか分からずに, 何度もやり直しになってしまうものです. 最終的にはサウンド EEPROM への書き込みが必要であるとはいえ, 何度も書き込み作業を行うのは実に手間がかかり非効率的です.

そこで作業の効率化の方法のひとつが, 作成した WAV ファイルをパーソナルコンピュータ上で再生してみることです. 具体的には, コマンドライン上で WAV ファイルを再生するコマンドを利用します. この目的に最適なソフトウエアが SoX (Sound eXchange) で, 本来はフォーマット変換やサウンド効果機能を持つソフトウエアですが, 再生機能も有しています.

> SoX - Sound eXchange.
> `http://sox.sourceforge.net/` (2019 年 6 月 23 日閲覧)

SoX は Windows/MacOS/Linux いずれでも利用可能なフリーソフトウエアで, Linux では標準で利用可能となっている場合もあります. もしあらかじめインストールされていなければ, 各自でダウンロードしてインストールして下さい.

SoX には音声ファイルを再生する play コマンドが含まれているのでこれを利用します. このコマンドの便利な点は, コマンドライン引数に再生する WAV ファイルを 1 つ以上の任意の数を指定でき, それらを切れ目なく続けて再生する点です. この機能のおかげで, 切り出した WAV ファイルのつながり具合を簡単に確認することができます.

Linux 上で play コマンドを使用する例を以下に示します. ひとつの WAV ファイル syouka.wav を再生する場合は次のようにします.

```
> play syouka.wav
```

3 つの WAV ファイルを再生する場合は次のようにします. この例では, 開始部 bell-1.wav, 繰返部 bell-2.wav, 終了部 bell-3.wav の WAV ファイルを想定しています.

```
> play bell-1.wav bell-2.wav bell-3.wav
```

切り出しの適切さを確認するには以下の組み合わせで再生を行います.

- 開始部, 終了部 (繰返部がない場合)
- 開始部, 繰返部, 終了部 (繰返部が 1 回の場合)
- 開始部, 繰返部, 繰返部, 終了部 (繰返部が 2 回の場合)

それぞれの場合のコマンドラインは以下のようになります.

```
> play bell-1.wav bell-3.wav
> play bell-1.wav bell-2.wav bell-3.wav
```

表 26.2: ワンショット型サウンド EEPROM データ形式

データ型	フィールド名	バイト数	説明
uint32_t	magic	4	マジックナンバ 0xDADBABE1
uint32_t	addr1	4	PCM サウンドデータ開始アドレス 12
uint32_t	size1	4	PCM サウンドデータサイズ n
uint8_t	data1	n	PCM サウンドデータ本体

表 26.3: リピート型サウンド EEPROM データ形式

データ型	フィールド名	バイト数	説明
uint32_t	magic	4	マジックナンバ 0xDADBABE0
uint32_t	addr1	4	開始部 PCM サウンドデータ開始アドレス 28
uint32_t	size1	4	開始部 PCM サウンドデータサイズ n_1
uint32_t	addr2	4	繰返部 PCM サウンドデータ開始アドレス
uint32_t	size2	4	繰返部 PCM サウンドデータサイズ n_2
uint32_t	addr3	4	終了部 PCM サウンドデータ開始アドレス
uint32_t	size3	4	終了部 PCM サウンドデータサイズ n_3
uint8_t	data1	n_1	開始部 PCM サウンドデータ本体
uint8_t	data2	n_2	繰返部 PCM サウンドデータ本体
uint8_t	data3	n_3	終了部 PCM サウンドデータ本体

```
> play bell-1.wav bell-2.wav bell-2.wav bell-3.wav
```

実際に耳で聞いてみて不満足なら編集をやり直します．十分に満足できれば図 26.3 の手順に従って，EEPROM への書き込みと PCM サウンド再生器への装着を行います．

26.3　サウンド EEPROM データ形式

第 27 章で紹介する EEPROM メモリ型 PCM サウンド再生器で使用する EEPROM 用の PCM サウンドデータのための独自のデータ形式を紹介します．

- ワンショット型
  表 26.2 にワンショット型のデータ構造を示します．最初の 4 バイトはデータ形式がワンショット型であることを示すマジックナンバーです．続いて PCM サウンドデータ開始アドレスを配置しますがワンショット型の場合にはアドレス 12 となります．その後には PCM サウンドデータのサイズをバイト数が配置されます．そして最後には PCM サウンドデータ本体が配置されます．PCM サウンドデータは 1 サンプル 1 バイト (8 ビット) で，サンプリング周波数は 8kHz を想定しています．
- リピート型
  表 26.3 にリピート型のデータ構造を示します．最初の 4 バイトはデータ形式がリピート型であることを示すマジックナンバーで，この値を見ることでワンショット型かリピート型かを区別できます．続いて開始部，繰返部，終了部それぞれに対するデータ開始アドレスとデータサイズ情報が配置され，その後にそれぞれの PCM サウンドデータ本体が配置されます．

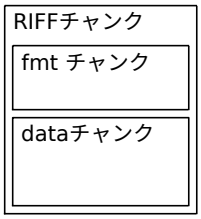


図 26.13: WAV ファイルの内部構造 (概略)

表 26.4: RIFF チャンク

データ型	フィールド名	バイト数	説明
`uint32_t`	`ckID`	4	チャンク識別子
	0x52494646 (“`RIFF`”)		
`int32_t`	`ckSize`	4	チャンクサイズ
	チャンク全体のサイズから 8 を減じた値		
`uint32_t`	`formType`	4	データ型識別子.
	音声の場合は 0x57415645 (“`WAVE`”)		

26.4　WAV ファイル形式

WAV ファイル形式 (RIFF waveform audio format) とは, サウンドデータをコンピュータ上のファイルとして表現する方法のひとつです. この形式のファイルの内部構造を説明します. WAV ファイル形式には多彩なデータの格納や拡張機能がありますが, 本章では最小限の基本的な部分のみを概説します.

26.4.1　WAV ファイル構造

WAV ファイル形式の全体像は, 図 26.13 に示す通りで, 詳しい説明は以下の通りです. ただし本書では扱わないものは説明を省略します.

- ファイルはチャンク (chunk) と呼ばれる一塊のデータの集合として構成されます.
- ファイル全体はひとつの RIFF チャンクです.
- RIFF チャンクはヘッダ (header) で始まり, 複数のチャンクの並びで構成されます. つまり RIFF チャンク内に複数のチャンクが格納されています. サウンドデータとして必須のチャンクは fmt チャンクと data チャンクです.

WAV ファイルの構成要素であるチャンクを以下に説明します.

RIFF チャンク

RIFF チャンクは WAV ファイル全体の構造を表すチャンクで, ファイルの先頭に配置されます. RIFF チャンクのヘッダ部分の構造を表 26.4 に示します. なおフィールド `ckSize` でのチャンクサイズとは RIFF チャンク全体での数値を指します.

fmt チャンク

fmt チャンクはサウンドデータに関する各種パラメータを記録するチャンクで, その構造を表 26.5 に示します.

表 26.5: fmt チャンク

データ型	フィールド名	バイト数	説明
uint32_t	ckID	4	チャンク識別子
	fmt チャンクは 0x666D7420 (“fmt ”)		
int32_t	ckSize	4	チャンクサイズ
int16_t	wFormatTag	2	サウンドデータの形式
	PCM 形式の場合は 1		
uint16_t	wChannels	2	チャンネル数
	モノラルの場合は 1, ステレオの場合は 2		
uint16_t	dwSamplesPerSec	4	サンプリング周波数 (Hz)
	8kHz の場合は 8000		
uint32_t	dwAvgBytesPerSec	4	平均データ数
	1 秒あたりのデータのバイト数		
uint16_t	wBlockAlign	2	データのブロックサイズ
	1 秒あたりのデータのバイト数		
uint16_t	bitspersample	2	1 サンプルあたりのビット数
	モノラル 8 ビットのリニア PCM の場合は 8		
uint16_t	extsize	2	拡張データサイズ m
uint16_t	extdata	m	拡張データ (m バイト)

表 26.6: data チャンク

データ型	フィールド名	バイト数	説明
uint32_t	ckID	4	チャンク識別子
	data チャンクは 0x666D7420 (“data”)		
int32_t	ckSize	4	n
int8_t	wavedata	n	サウンドデータ (n バイト)

data チャンク

data チャンクはサウンドデータ本体を記録するチャンクで, その構造を表 26.6 に示します. なお fmt チャンクは data チャンクの前に配置されなければなりません.

26.4.2　WAV ファイルの例

具体例を通してファイル構造を見てゆきます. ある WAV ファイルの 16 進ダンプの先頭部分を図 26.14 に示します.

- 第 0x0000 バイト目より: 0x52, 0x49, 0x46, 0x46
  RIFF チャンクに対するチャンク識別子です.
- 第 0x0004 バイト目より: 0x09, 0x27, 0x00, 0x00
  ファイル全体のサイズから 8 を減じた値 0x00002709 です. 複数バイトで表現される整数のバイトオーダはリトルエンディアン (litlle endian), すなわち桁の小さい順で並べられます. ファイル全体のサイズは

```
          +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
00000000: 52 49 46 46 09 27 00 00 57 41 56 45 66 6D 74 20
00000010: 12 00 00 00 01 00 01 00 40 1F 00 00 40 1F 00 00
00000020: 01 00 08 00 00 00 64 61 74 61 5D 1C 00 00 88 7D
00000030: 6E 72 76 82 8C 93 97 89 73 81 83 6C 75 81 7B 7D
00000040: 89 8D 82 73 83 90 7A 71 79 6F 6C 7C 84 7B 7E 89
    ⋮                             ⋮
00002700: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00002710: 00
```

図 26.14: WAV ファイルの例 (16 進ダンプ)

0x0002711 バイトですから, これから 8 を引くと 0x0002719 となります.

- 第 0x0008 バイト目より: 0x57, 0x41, 0x56, 0x45
  この RIFF チャンクは音声データを記録していることを示しています.
- 第 0x000C バイト目より: 0x66, 0x6D, 0x74, 0x20
  チャンク識別子です. fmt チャンクの開始を示します.
- 第 0x0010 バイト目より: 0x12, 0x00, 0x00, 0x00
  fmt チャンクのサイズ 0x12(=18) の値です. fmt チャンクは第 0x000C バイト目から始まり第 0x0025 バイト目で終わりますから, サイズは 18 バイトです.
- 第 0x0014 バイト目より: 0x01, 0x00
  サウンドデータの形式を示します. 値は 0x0001 なのでこのファイルに記録されているデータは PCM 形式です.
- 第 0x0016 バイト目より: 0x01, 0x00
  チャネル数を示します. 値は 0x0001 なのでチャンネル数は 1, すなわちモノラルのサウンドデータです.
- 第 0x0018 バイト目より: 0x40, 0x1F, 0x00, 0x00
  サンプリングレートを示します. 値 0x00001F40 を 10 進数表現すると 8000 ですから, サンプリングレートは 8kHz のサウンドデータです.
- 第 0x001C バイト目より: 0x40, 0x1F, 0x00, 0x00
  平均データ数, すなわち 1 秒あたりのデータのバイト数を示します. 値 0x00001F40 を 10 進数表現すると 8000, つまり 1 秒あたりのサウンドデータは 8K バイトであることを示します.
- 第 0x0020 バイト目より: 0x01, 0x00
  データのブロックサイズを示します. 8 ビットモノラルですので, 値は 1 バイトです.
- 第 0x0022 バイト目より: 0x08, 0x00
  1 サンプルあたりのビット数を示します. 値は 0x0008 ですので 1 サンプルあたり 8 ビット (1 バイト) のサウンドデータです.
- 第 0x0024 バイト目より: 0x00, 0x00
  拡張部分のサイズを示します. 値は 0x0000 ですので拡張部分は含まれません. なお fmt チャンクサイズが 0x10 の場合にはこの部分も含まれません.
- 第 0x0026 バイト目より: 0x64, 0x61, 0x74, 0x61
  チャンク識別子です. data チャンクの開始を示します.
- 第 0x002A バイト目より: 0x5D, 0x1C, 0x00, 0x00
  サウンドデータのサイズを示します. 値は 0x00001C5D です.

- 第 0x002E バイト目より: 0x88, 0x7D, 0x6E, 0x72, ... 0x00
  サウンドデータです.

26.5　サウンド EEPROM データファイル生成プログラム wav2img

WAV ファイルからサウンド EEPROM データファイルを生成するプログラム wav2img を紹介します. 本プログラムを構成するソースファイル一覧を以下に示します.

表 26.7: 【ソースファイル一覧】 サウンド EEPROM データ生成 wav2img

初	頁	番号	ファイル名	内容	行数
✓	478	26.1	defs.h	諸定義: ヘッダ	19
✓	478	26.2	snd-eeprom.h	サウンド EEPROM アクセス関数: ヘッダ	54
✓	479	26.3	misc.c	補助関数群: 実装	39
✓	480	26.4	wav2img.c	主ソースコード	234

計 346 行 (うち初出 346 行, 既出 0 行)

ソースファイル defs.h (諸定義: ヘッダ)

定数の定義, 関数のプロトタイプ宣言を行います.

ソースファイル番号 26.1: defs.h

```
1  #if !defined(__DEFS_H__)
2  #define __DEFS_H__
3
4  /* サンプリングレート: 8kHz */
5  #define WAV_SAMPRATE        8000
6
7  extern void emit_32bit(FILE *fp, uint32_t d);
8  extern void emit_16bit(FILE *fp, uint16_t d);
9  extern void emit_8bit(FILE *fp, uint8_t d);
10 extern uint32_t get_32bit(uint8_t *p, uint8_t i);
11 extern uint16_t get_16bit(uint8_t *p, uint8_t i);
12
13 extern  void emit_wave_header(FILE *fp_wav, uint32_t size);
14 extern void emit_wave_data_chunk(FILE *fp_wav, uint32_t size);
15 extern void emit_wave_data_pcm(char *f_img,
16                                FILE *fp_img, FILE *fp_wav,
17                                uint32_t addr, uint32_t size);
18
19 #endif /*__DEFS_H__*/
```

ソースファイル snd-eeprom.h (サウンド EEPROM アクセス関数: ヘッダ)

マジックナンバなどを定義します. このソースファイルは第 27 章の EEPROM メモリ型 PCM サウンド再生器のファームウエアと共用しているため, 本章のプログラムでは使用しない関数のプロトタイプ宣言がありますが気にしないで下さい.

ソースファイル番号 26.2: snd-eeprom.h

```
1  #if !defined(__SND_EEPROM_H__)
2  #define __SND_EEPROM_H__
3
4  /*
5   * EEPROMの先頭に書き込むマジックナンバー
6   *（正しいEEPROMであるかどうかの判定に使用）
```

```
 *    繰り返し再生するサウンドデータ：       DADBABE0
 *    一度だけ再生して終了するサウンドデータ: DADBABE1
 */
#define EEPROM_MAGIC_NUMBER_0         0xDA
#define EEPROM_MAGIC_NUMBER_1         0xDB
#define EEPROM_MAGIC_NUMBER_2         0xAB
#define EEPROM_MAGIC_NUMBER_3_REPEAT  0xE0 /* リピート型 */
#define EEPROM_MAGIC_NUMBER_3_ONESHOT 0xE1 /* ワンショット型 */

/* リピート型 EEPROMデータフォーマット１
 *   0x00-0x03 (4 byte) : マジックナンバー(0xDA,0xDB,0xAB,0xE0)
 *   0x04-0x07 (4 byte) : 開始部サウンドデータ配置アドレス
 *   0x08-0x0B (4 byte) : 開始部サウンドデータサイズ
 *   0x0C-0x0F (4 byte) : 繰返部サウンドデータ配置アドレス
 *   0x10-0x13 (4 byte) : 繰返部サウンドデータサイズ
 *   0x14-0x17 (4 byte) : 終了部サウンドデータ配置: アドレス
 *   0x18-0x1B (4 byte) : 終了部サウンドデータサイズ
 *   0x1C-              : サウンドデータ本体
 * ワンショット型 EEPROMデータフォーマット２
 *   0x00-0x03 (4 byte) : マジックナンバー(0xDA,0xDB,0xAB,0xE1)
 *   0x04-0x07 (4 byte) : サウンドデータ配置アドレス
 *   0x08-0x0B (4 byte) : サウンドデータサイズ
 *   0x0C-              : サウンドデータ本体
 */

#define INDEX_0   0  /* 開始(start)部  */
#define INDEX_1   1  /* 繰返(repeat)部 */
#define INDEX_2   2  /* 終了(end)部    */

enum sound_type_e {
  SOUND_TYPE_UNKNOWN = 0,
  SOUND_TYPE_REPEAT,  /* リピート型 */
  SOUND_TYPE_ONESHOT, /* ワンショット型 */
};

struct eeprom_index_s {
  uint32_t  addr;
  uint32_t  size;
};

/* インデックス情報 */
extern struct eeprom_index_s eeprom_index_0; /* 開始(start)部  */
extern struct eeprom_index_s eeprom_index_1; /* 繰返(repeat)部 */
extern struct eeprom_index_s eeprom_index_2; /* 終了(end)部    */

extern enum sound_type_e eeprom_init(void);

#endif /*__SND_EEPROM_H__*/
```

ソースファイル misc.c (補助関数群: 実装)

32 ビット整数や 16 ビット整数を複数バイトで符号化/復号化する関数群です. 関数として定義しておき, バイトオーダをうっかり間違えないようにします.

ソースファイル番号 26.3: misc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "defs.h"

void emit_32bit(FILE *fp, uint32_t d)
{
  fprintf(fp, "%c", (d & 0x000000FF));
  fprintf(fp, "%c", (d & 0x0000FF00) >>  8);
  fprintf(fp, "%c", (d & 0x00FF0000) >> 16);
  fprintf(fp, "%c", (d & 0xFF000000) >> 24);
```

```
12  }
13
14  void emit_16bit(FILE *fp, uint16_t d)
15  {
16    fprintf(fp, "%c", (d & 0x00FF));
17    fprintf(fp, "%c", (d & 0xFF00) >>  8);
18  }
19
20  void emit_8bit(FILE *fp, uint8_t d)
21  {
22    fprintf(fp, "%c", d);
23  }
24
25  uint32_t get_32bit(uint8_t *p, uint8_t i)
26  {
27    return
28      ((uint32_t) p[i+0]) +
29      ((uint32_t) p[i+1] <<  8) +
30      ((uint32_t) p[i+2] << 16) +
31      ((uint32_t) p[i+3] << 24);
32  }
33
34  uint16_t get_16bit(uint8_t *p, uint8_t i)
35  {
36    return
37      ((uint32_t) p[i+0]) +
38      ((uint32_t) p[i+1] <<  8);
39  }
```

ソースファイル wav2img.c (主ソースコード)

WAV ファイルからサウンド EEPROM データファイルを生成するプログラム wav2img の主たる部分です. このソースファイルで行っていることは実に単純で, WAV ファイル中の PCM サウンドデータを単に抜き出して, サウンド EEPROM データファイルのデータ構造に合わせて出力しているだけです. WAV ファイルの構造とサウンド EEPROM データファイルのデータ構造を参照しながら以下のソースファイルを読めば理解できると思います.

ソースファイル番号 26.4: wav2img.c

```
1   /*
2    * wav2img
3    * - Generate a sound EEPROM image file from WAVE files
4    */
5   #include <stdio.h>
6   #include <stdlib.h>
7   #include <stdint.h>
8   #include <string.h>
9   #include <sys/stat.h>
10  #include "defs.h"
11  #include "snd-eeprom.h"
12
13  char *f_wav[3];
14  int opt_verbose = 0;
15
16  static void usage(void);
17  static void     wav2img(enum sound_type_e st, FILE *fp);
18  static void     wav_pcm_emit(FILE *fp, char *f);
19  static uint32_t wav_pcm_size(char *f);
20  static uint32_t wav_pcm_data(FILE *fp_emit, char *f);
21
22  #if defined(_MSC_VER)
23  #  define FOPEN(p,f,m)  fopen_s(p, f, m)
24  #else
25  #  define FOPEN(p,f,m)  (((*(p)=fopen(f, m)) != NULL) ? 0 : 1)
26  #endif
27
28  int main(int argc, char ** argv)
```

```
{
  enum sound_type_e st = SOUND_TYPE_UNKNOWN;
  char *f_out = NULL;
  /* Command line arguments */
  for (argc--, argv++; argc > 0; argc--, argv++) {
    if (argv[0][0] == '-') {
      if (strcmp(argv[0], "-v") == 0)  {
        opt_verbose = 1;
      } else {
        usage();
      }
    } else {
      break;
    }
  }
  if (argc == 4) {
    st = SOUND_TYPE_REPEAT;
    f_wav[0] = argv[0];
    f_wav[1] = argv[1];
    f_wav[2] = argv[2];
    f_out    = argv[3];
  } else if (argc == 2) {
    st = SOUND_TYPE_ONESHOT;
    f_wav[0] = argv[0];
    f_wav[1] = NULL;
    f_wav[2] = NULL;
    f_out    = argv[1];
  } else {
    usage();
  }
  /* Output file check */
  struct stat fs;
  if (stat(f_out, &fs) == 0) {
    printf("ERROR - FILE EXISTS: %s\n", f_out);
    exit(1);
  }
  /* Emit EEPROM binary image */
  FILE *fp_out = NULL;
  if (FOPEN(&fp_out, f_out, "wb") != 0) {
    printf("ERROR - CANNOT WRITE TO FILE: %s\n", f_out);
    exit(1);
  }
  wav2img(st, fp_out);
  fclose(fp_out);
  fp_out = NULL;
  /* Done */
  return 0;
}

static void usage(void)
{
  printf(
"wav2img - Generates a sound EEPROM image file\n\
Usage (Oneshot type): wav2img WAV FILE\n\
Usage (Repeart type): wav2img WAV-START WAV-REPEAT WAV-END FILE\n\
Example 1: wav2img chime.wav chime.bin\n\
Example 2: wav2img bell-1.wav bell-2.wav bell-3.wav bell.bin\n\
");
  exit(1);
}

static void wav2img(enum sound_type_e st, FILE *fp)
{
  switch (st) {
  default:
  case SOUND_TYPE_UNKNOWN:
    break;
  case SOUND_TYPE_REPEAT:
    /* File magic number */
    emit_8bit(fp, EEPROM_MAGIC_NUMBER_0);
    emit_8bit(fp, EEPROM_MAGIC_NUMBER_1);
    emit_8bit(fp, EEPROM_MAGIC_NUMBER_2);
```

```
101       emit_8bit(fp, EEPROM_MAGIC_NUMBER_3_REPEAT);
102       /* Index */
103       uint32_t addr_start = 4 + 8 + 8 + 8;
104       uint32_t size_start = wav_pcm_size(f_wav[0]);
105       uint32_t addr_repeat = addr_start + size_start;
106       uint32_t size_repeat = wav_pcm_size(f_wav[1]);
107       uint32_t addr_end = addr_repeat + size_repeat;
108       uint32_t size_end = wav_pcm_size(f_wav[2]);
109       if (opt_verbose == 1) {
110         printf("TYPE: REPEAT\n");
111         printf("ADDR-START:  0x%06X\n", addr_start);
112         printf("SIZE-START:  0x%06X\n", size_start);
113         printf("ADDR-REPEAT: 0x%06X\n", addr_repeat);
114         printf("SIZE-REPEAT: 0x%06X\n", size_repeat);
115         printf("ADDR-END:    0x%06X\n", addr_end);
116         printf("SIZE-END:    0x%06X\n", size_end);
117       }
118       emit_32bit(fp, addr_start);
119       emit_32bit(fp, size_start);
120       emit_32bit(fp, addr_repeat);
121       emit_32bit(fp, size_repeat);
122       emit_32bit(fp, addr_end);
123       emit_32bit(fp, size_end);
124       /* PCM data */
125       wav_pcm_emit(fp, f_wav[0]);
126       wav_pcm_emit(fp, f_wav[1]);
127       wav_pcm_emit(fp, f_wav[2]);
128       break;
129     case SOUND_TYPE_ONESHOT:
130       /* File magic number */
131       emit_8bit(fp, EEPROM_MAGIC_NUMBER_0);
132       emit_8bit(fp, EEPROM_MAGIC_NUMBER_1);
133       emit_8bit(fp, EEPROM_MAGIC_NUMBER_2);
134       emit_8bit(fp, EEPROM_MAGIC_NUMBER_3_ONESHOT);
135       /* Index */
136       uint32_t addr = 4 + 8;
137       uint32_t size = wav_pcm_size(f_wav[0]);
138       if (opt_verbose == 1) {
139         printf("TYPE: ONESHOT\n");
140         printf("ADDR: 0x%06X\n", addr);
141         printf("SIZE: 0x%06X\n", size);
142       }
143       emit_32bit(fp, addr);
144       emit_32bit(fp, size);
145       addr += size;
146       /* PCM data */
147       wav_pcm_emit(fp, f_wav[0]);
148       break;
149     }
150   }
151
152   static uint32_t wav_pcm_size(char *f)
153   {
154     return wav_pcm_data(NULL, f);
155   }
156
157   static void wav_pcm_emit(FILE *fp, char *f)
158   {
159     wav_pcm_data(fp, f);
160   }
161
162   static uint32_t wav_pcm_data(FILE *fp_out, char *f)
163   {
164     uint32_t wav_datsiz;
165     uint32_t chunk_addr;
166     uint32_t chunk_size;
167     FILE  *fp_wav = NULL;
168     if (f == NULL) {
169       printf("INTERNAL ERROR\n");
170       exit(1);
171     }
172     if (FOPEN(&fp_wav, f, "rb") != 0) {
```

```
173       printf("ERROR - FILE NOT FOUND: %s\n", f);
174       exit(1);
175     }
176     uint8_t  wavheader[36];
177     fread(wavheader, 1, sizeof(wavheader), fp_wav);
178     if ((wavheader[ 0] != 'R') || (wavheader[ 1] != 'I') ||
179         (wavheader[ 2] != 'F') || (wavheader[ 3] != 'F') ||
180         (wavheader[ 8] != 'W') || (wavheader[ 9] != 'A') ||
181         (wavheader[10] != 'V') || (wavheader[11] != 'E') ||
182         (wavheader[12] != 'f') || (wavheader[13] != 'm') ||
183         (wavheader[14] != 't') || (wavheader[15] != ' ')) {
184       printf("ERROR - NOT A WAVE FILE: %s\n", f);
185       exit(1);
186     }
187     if (get_16bit(wavheader, 20)  != 1) { /* format id */
188       printf("ERROR - NOT A PCM WAVE FILE: %s\n", f);
189       exit(1);
190     }
191     if (get_16bit(wavheader, 22) != 1) { /* #channels */
192       printf("ERROR - NOT MONORAL: %s\n", f);
193       exit(1);
194     }
195     if (get_32bit(wavheader, 24) != WAV_SAMPRATE) { /* sampling rate */
196       printf("ERROR - SAMPLING RATE != %dHz: %s\n",
197              WAV_SAMPRATE, f);
198       exit(1);
199     }
200     if (get_32bit(wavheader, 28) != WAV_SAMPRATE) { /* data speed */
201       printf("ERROR - DATA SPEED != %dByte/Sec: %s\n",
202              WAV_SAMPRATE, f);
203       exit(1);
204     }
205     if (get_16bit(wavheader, 34) != 8) { /* #bits/sample */
206       printf("ERROR - BIT/SAMPLE != 8BIT: %s\n", f);
207       exit(1);
208     }
209     chunk_size = get_32bit(wavheader, 16);
210     chunk_addr = 20 + chunk_size;
211     while (!feof(fp_wav)) {
212       fseek(fp_wav, chunk_addr, SEEK_SET);
213       fread(wavheader, 1, 8, fp_wav);
214       if ((wavheader[0] == 'd') && (wavheader[1] == 'a') &&
215           (wavheader[2] == 't') && (wavheader[3] == 'a')){
216         wav_datsiz = get_32bit(wavheader, 4);
217         break;
218       }
219       chunk_size = get_16bit(wavheader, 4);
220       chunk_addr += chunk_size;
221     }
222     if (feof(fp_wav)) {
223       printf("ERROR - UNSUPPORTED FORMAT: %s\n", f);
224       exit(1);
225     }
226     if (fp_out != NULL) {
227       uint32_t nbytes;
228       for (nbytes = wav_datsiz; nbytes > 0; nbytes--) {
229         fputc(fgetc(fp_wav), fp_out);
230       }
231     }
232     fclose(fp_wav);
233     return wav_datsiz;
234   }
```

26.6　各種オペレーティングシステムでのビルト方法

Linux や FreeBSD などの Unix 系オペレーティングシステムおよび Windows オペレーティングシステムにおいて wav2img を GCC (GNU Compiler Collection) と make を用いた方法を説明します. コンパイラ等の開発環境は第 24.4 節 (411 ページ) を参照して下さい.

Makefile の具体的な内容はすぐこの後に示しますが, いまは仮に Makefile が用意できたものとして, 以下の通り斜体字の部分を手入力して make コマンドを実行すると, 自動的にビルドが行われます. なお Windows オペレーティングシステムの場合には, コマンド名は make ではなく mingw32-make となります.

```
> make                          — make コマンドを実行
gcc  -c -I. -I../../firmware/SoundPlayer-EEPROM-A.X/ wav2img.c
gcc  -c -I. -I../../firmware/SoundPlayer-EEPROM-A.X/ misc.c
gcc  -I. -I../../firmware/SoundPlayer-EEPROM-A.X/ -o wav2img wav2img.o misc.o
```

ビルドのための Makefile の内容を以下に示します. この Makefile は Unix 系オペレーティングシステムと Windows オペレーティングシステムのどちらでも make によるビルドが可能なように書き方を工夫しています. なおソースファイルのディレクトリの設定が Makefile の先頭にありますから, 各自の状況に合わせて適宜修正を行って下さい.

ソースファイル番号 26.5: Makefile

```
1  SRCDIR=../../firmware/SoundPlayer-EEPROM-A.X/
2  CFLAGS=-I. -I$(SRCDIR)
3
4  ifeq ($(OS), Windows_NT)
5    EXEEXT = .exe
6    CC = gcc
7    RM = cmd.exe /C del
8  else
9    EXEEXT =
10   CC = gcc
11   RM = rm -f
12 endif
13
14 PROGRAM=wav2img$(EXEEXT)
15
16 .PHONY: all clean distclean
17 all: $(PROGRAM)
18 clean:
19   -$(RM) $(PROGRAM)
20 distclean:
21   -$(RM) *.o core *.core *~
22
23 $(PROGRAM): wav2img.o misc.o
24   $(CC) $(CFLAGS) -o $(PROGRAM) wav2img.o misc.o
25 wav2img.o:   wav2img.c defs.h $(SRCDIR)/snd-eeprom.h
26   $(CC) -c $(CFLAGS) wav2img.c
27 misc.o:      misc.c defs.h
28   $(CC) -c $(CFLAGS) misc.c
```

課題

1. WAV ファイルをひとつ任意に選んで, そのファイル構造を第 26.4.2 節での説明と同様な形で説明して下さい.
2. 以下の条件を満たす WAV ファイルはどのようなデータ列となるか示して下さい. つまり手作業で WAV ファイルを構成してみて下さい. サウンドデータ形式: PCM, チャンネル数: 2, サンプリング周波数: 16kHz, 量子化ビット数: 16. 音声部分は 8 秒の無音としますが, データ量が多いのでサウンド EEPROM データファイルの先頭部分だけを示して下さい.
3. 以下の WAV ファイルから本章で紹介したワンショット型の EEPROM データ形式のサウンドデータを手作業で構成して下さい. 音声部分はすべて無音としますが, データ量が多いのでサウンド EEPROM データファイルの先頭部分だけを示して下さい.
   - サイズ 2K バイト

4. 以下の 3 つの WAV ファイルから本章で紹介したリピート型の EEPROM データ形式のサウンドデータを手作業で構成して下さい. 音声部分はすべて無音としますが, データ量が多いのでサウンド EEPROM データファイルの先頭部分だけを示して下さい.
   - 開始部: サイズ 2K バイト
   - 繰返部: サイズ 8K バイト
   - 終了部: サイズ 3K バイト
5. wav2img は, パラメータがサンプリング周波数 8kHz, 量子化ビット数 8 ビット, チャンネル数 1 のモノラルの WAV ファイルしか受け付けませんでした. パラメータが異なる WAV ファイル, あるいは WAV 以外の AIFF や AU フォーマットのファイルが与えられた場合でも, 必要に応じた変換をする機能を追加して下さい. 自らパラメータ変換をするプログラムコードを書くのは大変ですから, SoX などの既存の変換ソフトウエアを呼び出すようにすると良いでしょう.
6. サウンドデータの HEX あるいは BIN ファイルから, WAV ファイルに変換するプログラムを作成して下さい. 元の WAV ファイルを失った時に役に立ちます.

第 27 章

EEPROM メモリ型
PCM サウンド再生器

本章では PCM サウンドを再生する装置を紹介します. 本装置を組み立てただけでは PCM サウンドの再生はできず, サウンドデータを書き込んだ EEPROM の用意が必要であることに注意して下さい. デジタルオーディオの仕様は第 26 章で説明した通り, サンプリング周波数 8kHz, 量子化ビット数 8, チャンネル数 1 のモノラルですが, 鉄道のサウンドにはノイズが含まれているものが多いため, このような貧弱な仕様でも実際のところあまり気にはなりません.

本章で紹介する回路とファームウエアの特色は以下の通りです.

- 実物の音を再生
  実物から録音した音源を PCM サウンドデータにして再生します. 例えば駅の発車メロディ, 踏切, 汽笛, 鉄道唱歌のオルゴール, 到着アナウンスなど, 様々な音の再生が可能です.
- リピート型再生機能
  汽笛や踏切などの音に対して再生時間を任意に長く伸ばすことが可能です. DCC サウンドデコーダでは任意に長く汽笛を吹鳴する機能がありますが, それと同じ機能をリピート再生機能で実現します.
- 最大約 16 秒までのサウンドを記憶可能
  容量 1M ビットの EEPROM を使用して最大約 16 秒までの記憶が可能です.
- 特殊なデジタルオーディオ部品は不使用
  本装置は PCM デジタルオーディオ再生装置の一種ですがデジタルオーディオ用の高価な部品は使用しません PIC マイコン, EEPROM, アナログオーディオアンプ IC を主要な部品として安価に実現しています. デジタル信号からアナログ信号への変換は PIC マイコンによる PWM 波の発生により行います.

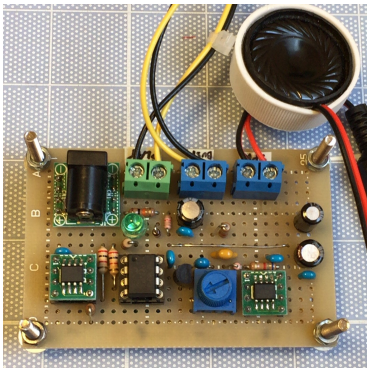


図 27.1: EEPROM メモリ型 PCM サウンド再生器の試作例

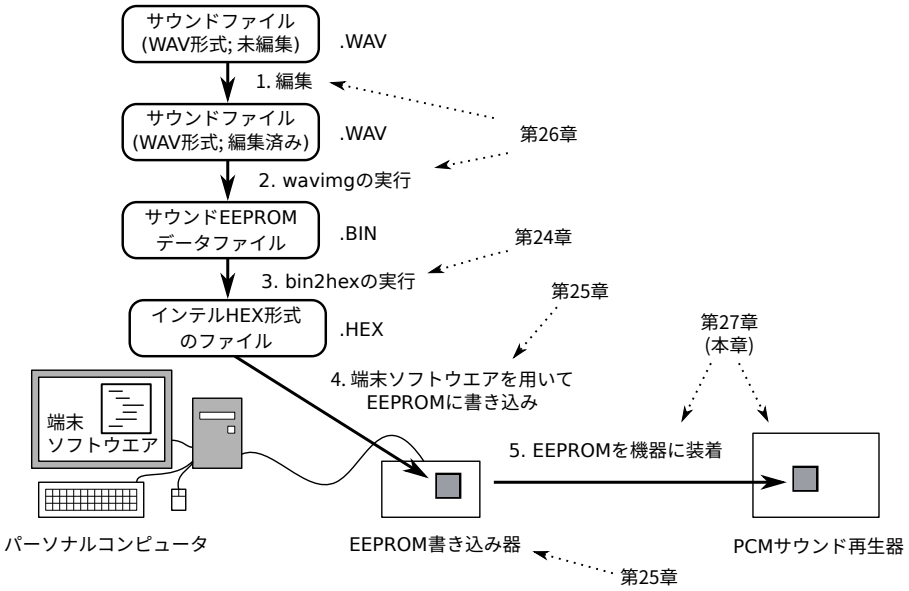


図 27.2: PCM サウンド再生器の利用のための作業手順

```
1  main()
2  {
3    for (;;) {
4      再生開始ボタンのオン(再生指令)を待機;
5      if (EEPROM内容 == リピート型) {
6        開始部を再生;
7        while (再生開始ボタンがオン) {
8          繰返部を再生;
9        }
10       終了部を再生;
11     } else if (EEPROM内容 == ワンショット型) {
12       再生;
13     }
14     再生開始ボタンのオフを待機;
15   }
16 }
```

図 27.3: ファームウエア構造 (概要)

本章で学ぶ内容は以下の通りです.

- PIC マイコンのタイマモジュール TMR0
- スイッチ入力におけるチャタリングの除去

一連の作業の全体像の中での本章の位置づけは図 27.2 に示します.

本章のファームウエアの大まかな構造を図 27.3 に示します. EEPROM 内にはリピート型かワンショット型かを識別するデータが書いてあり, それに従って再生方法を決定します.

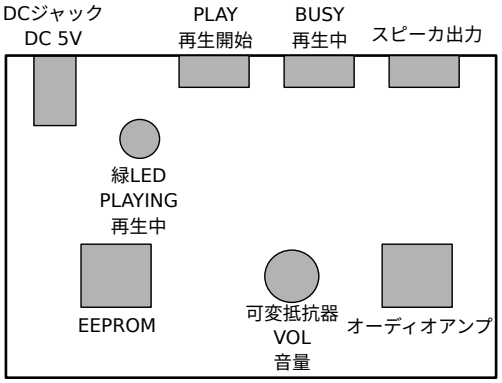


図 27.4: EEPROM メモリ型 PCM サウンド器の接続表示部品の配置例と名称

27.1　使用方法

本装置の使用方法を以下に説明します. 以下の説明で使用する部品名称は図 27.4 に配置例とともに示しています. 再生開始の入力はコネクタ PLAY を通じて行います. この入力はアクセサリ制御入力の仕様に従っていますから, 手動でオン・オフするスイッチに限らず様々な装置から再生開始の制御が可能です. 手動で再生する場合あるいは動作テストを行う際には押しボタンスイッチをコネクタに接続したものを別途用意すると良いでしょう. ただしそれらの部品は部品表には含まれていませんので注意して下さい. 再生中の状態は LED の点灯による表示に加えてコネクタ BUSY を通じて出力されます. この出力はアクセサリ制御出力の仕様に従っていますから, 再生終了を他の装置に知らせることが可能です.

本装置の PCM サウンドデータには以下の 2 種類があります (466 ページ図 26.4 参照).

- ワンショット型
  EEPROM 内の PCM サウンドデータの初めから終わりまでを再生して終了します.
- リピート型
  PCM サウンドデータは開始部, 繰返部, 終了部の 3 部で構成されます. 最初に開始部を再生し, 繰返部を繰り返して再生して, 終了部を再生して終了します.

準備は以下の通りです.

1. PCM サウンドデータの作成
   wav2img を使用します. (第 26 章)
2. PCM サウンドデータのインテル HEX ファイル化
   bin2hex を使用します. (第 24 章)
3. EEPROM への PCM サウンドデータの書き込み
   EEPROM 書き込み器を使用します. (第 25 章)
4. スピーカの接続
5. 再生開始スイッチの接続
   コネクタ PLAY に接続します.
6. AC アダプタの接続
   5V の AC アダプタを DC ジャックに接続します. **間違って 12V 出力の AC アダプタを使用すると回路が破損する場合がありますので, 十分に注意して下さい.**

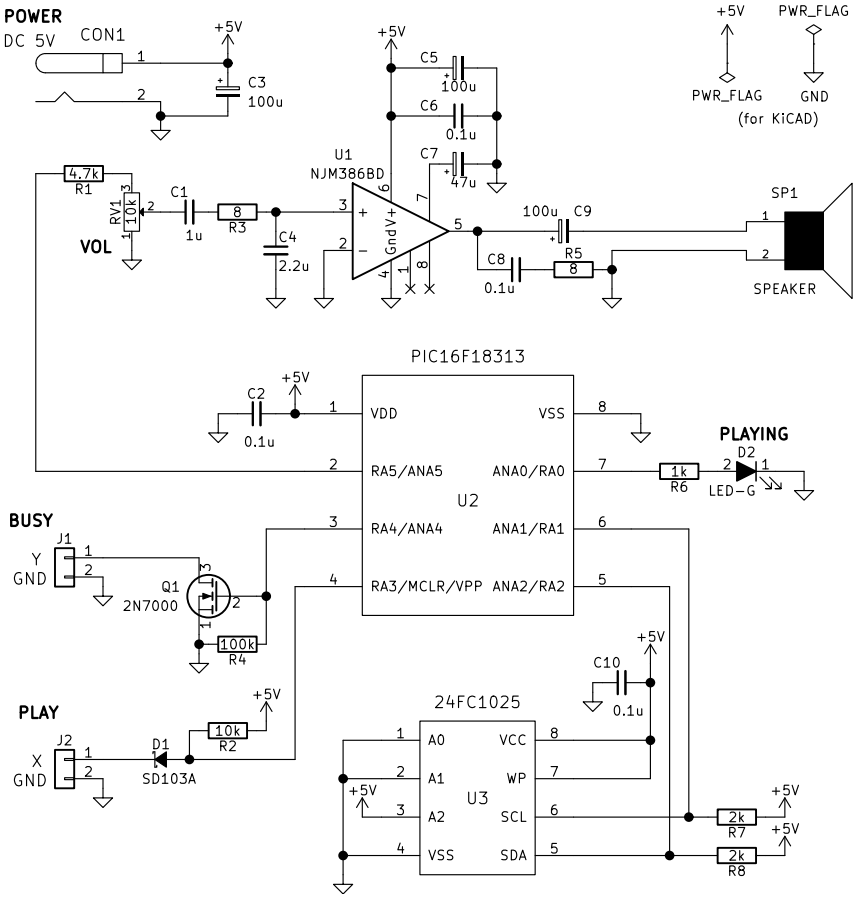


図 27.5: 【回路図】 EEPROM メモリ型 PCM サウンド器

サウンドの再生方法は以下の通りです.

- 再生開始
  PLAY スイッチをオン (L) にします. LED が点灯して出力 BUSY は L になり, 再生が始まります. このとき直ちに PLAY スイッチをオフにしても構いません.
  - ワンショット型
    再生が終了すると LED は消灯して出力 BUSY は HiZ になります. 次の再生を行うにはいったん PLAY スイッチをオフにして下さい.
  - リピート型
    まず開始部が再生されます. 開始部の再生が終了すると出力 BUSY は HiZ になります. PLAY スイッチがオンである間は反復部が繰り返し再生され, PLAY スイッチがオフになると反復部の繰り返しを終えて終了部が再生されます. 終了部の再生が終了すると LED は消灯します.
- 再生強制終了 (ワンショット型)
  いったん PLAY スイッチをオフにして再び PLAY スイッチをオンにすると再生は強制終了します.

27.2　回路図

表 27.1: 【部品表】 EEPROM メモリ型 PCM サウンド器

記号	数量	部品名
U2	1	PIC 16F18313-I/P [I-10889]
U1	1	NJM386BD　互換品でも可 [I-00022]
U3	1	I^2C EEPROM 24FC1025-I/P [I-03570]
Q1	1	FET 2N7000 [I-09723]
D1	1	ショットキーバリアダイオード SD103A [I-04271]
D2	1	LED 緑 [I-06413]
C3, C5, C9	3	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C7	1	電解コンデンサ 47μF　耐圧 16V 以上 [P-10270]
C1	1	セラミックコンデンサ 1μF [P-08150]
C2, C6, C8, C10	4	セラミックコンデンサ 0.1μF [P-00090]
C4	1	セラミックコンデンサ 2.2μF [P-08152]
R3, R5	2	抵抗 8Ω　1/4W [R-25100]
R6	1	抵抗 1kΩ　1/4W [R-25102]
R7, R8	2	抵抗 2kΩ　1/4W [R-25202]
R1	1	抵抗 4.7kΩ　1/4W [R-25472]
R2	1	抵抗 10kΩ　1/4W [R-25103]
R4	1	抵抗 100kΩ　1/4W [R-25104]
RV1	1	可変抵抗器 10kΩ　B カーブ, 半固定型 [P-08012]
	2	IC ソケット 8 ピン [P-00035]
J1, J2	2	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
SP1	1	スピーカー　8Ω, 入力 1W 程度
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか基板・配線材等

本装置の回路図を図 27.5 に, 部品リストを表 27.1 に, それぞれ示します. オーディオアンプには NJM386BD を使用していますが, いわゆる 386 と呼ばれている互換品であれば大丈夫と思います. オーディオアンプの出力は 1W 程度ですから, スピーカもそれに合わせたものを使用します. スピーカにはバッフルを付けて音が響くようにすると良いでしょう. 直径 28mm のスピーカはペットボトルのキャップをバッフルとして流用できて便利です.

27.3　PWM 波出力による PCM サウンド再生

デジタルアナログ変換器 DAC を使用することなく PCM サウンド再生を実現する原理を説明します. 図 27.6 に PCM サウンドデータと信号の流れを全体像を示します. 図 27.7 に PWM 波出力を利用した PCM サウンド再生の原理を示し, 以下ではこの図に基づいて説明をします.

音源の音声信号を図 27.7 の上から 1 番目のグラフで示し, これが PCM サウンドデータとして図 27.7 の上から 2 番目のグラフで示すようにデジタル化されて EEPROM に記憶されます. 本装置では量子化ビット数が 8

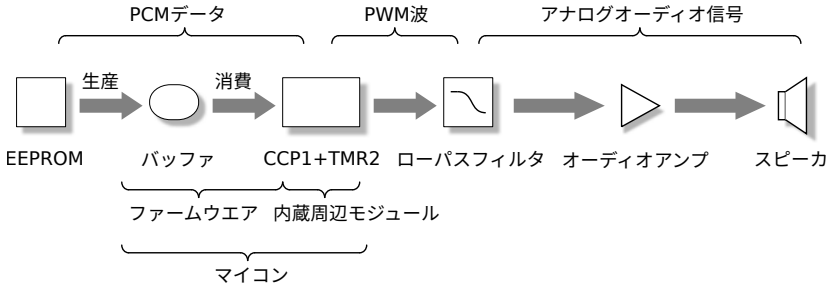


図 27.6: データと信号の流れ

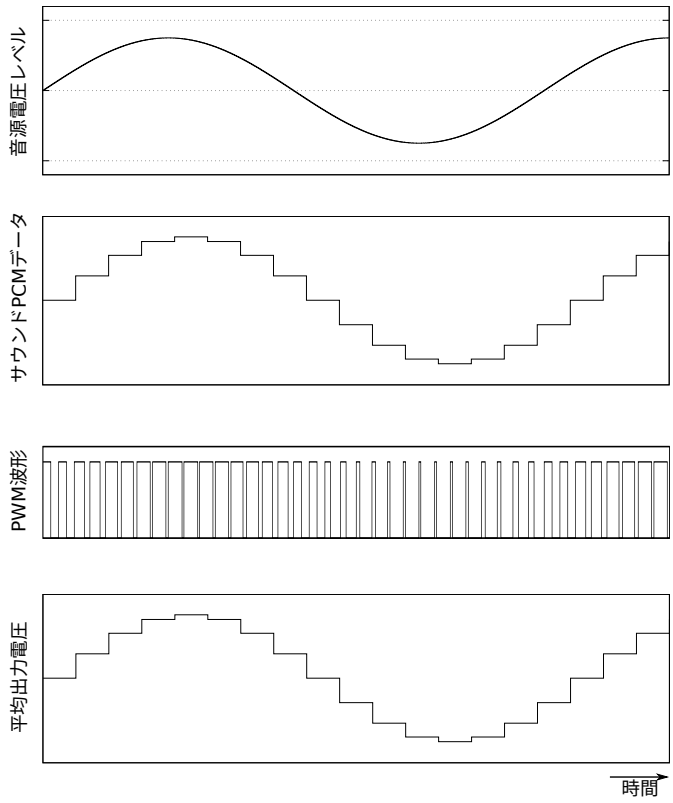


図 27.7: PWM 波出力を利用した PCM サウンド再生の原理

ですからひとつひとつの PCM サウンドデータは 0 から 255 までの範囲の整数で表現されます. 音声再生時には PCM サウンドデータの値から PWM デューティ比を決定して図 27.7 の上から 3 番目のグラフで示す PWM 波を出力します. 本装置ではサンプリング周波数が 8kHz ですから, 1 秒間に 8000 回の頻度で PCM サウンドデータを読み出して PWM デューティ比を更新します. PCM サウンドデータの値が 0 の場合は PWM デューティ比を 0%, 255 の場合は 100% とします. すると PCM サウンドデータの値の大小が PWM デューティ比の大小となりますので, 電圧レベルの大小が PWM 波の平均電圧の大小となります. すると PWM 波の出力の平均電圧は図 27.7 の上から 4 番目のグラフで示すように, 元の音声信号に近い波形に再生されます.

本回路ではマイコンからの PWM 波の出力は 0V/5V のオン・オフなので直流成分を含んでいますから, 100μF の電解コンデンサを通すことで直流成分を除去し交流成分だけを通過させます. さらにローパスフィルタを通すことでサンプリング周波数以上の周波数成分を除去し可聴域成分だけを通過させて, オーディオア

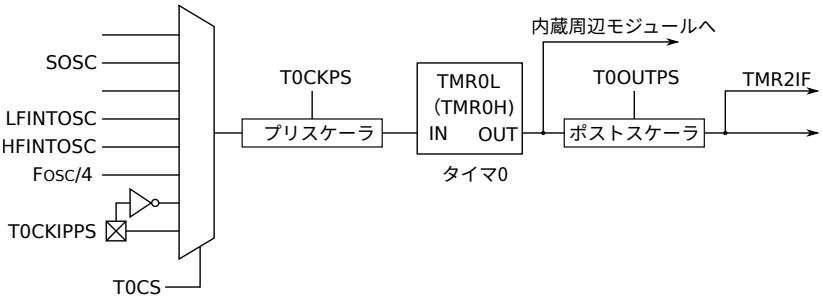


図 27.8: TMR0 の内部構造の概略

ンプの入力とします. ローパスフィルタで除去すべきは PWM 周波数以上の周波数成分です. ローパスフィルタにはコンデンサと抵抗器によるローパスフィルタ回路を使用します. このローパスフィルタ回路の遮断周波数 f_c は以下の式で求まることが知られています.

$$f_c = \frac{1}{2\pi RC}$$

本装置では C=2.2μF, R=6Ω ですから, 上の式に当てはめて計算すると以下の通り遮断周波数は 12.1kHz となります. するとサンプリング周波数 8kHz より低い音声成分は通過し, それより高い周波数の成分は遮断されます.

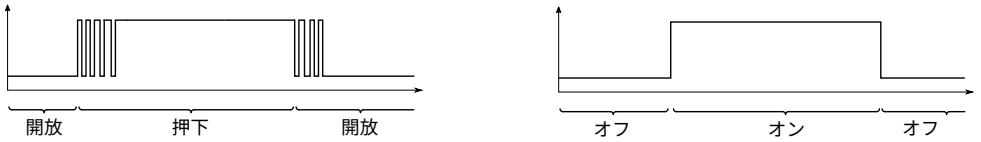
$$\begin{aligned} f_c &= \frac{1}{2\pi \times 6 \times (2.2 \times 10^{-6})} \\ &= 1/8.29 \times 10^{-5} \\ &= 12.1\text{kHz} \end{aligned}$$

27.4　TMR0: 8 ビットタイマモジュール

PIC マイコンの内蔵周辺モジュール TMR0 (Timer0; タイマ 0) を使用して時間の計測を行います. 図 27.8 に TMR0 の内部構造の概略を示します. TMR0 モジュールはクロックから受けた信号を分周するプリスケーラを通り, 一定周期で TMR0 レジスタのカウントアップを行います. 出力はポストスケーラにより分周されます.

TMR0 タイマで使用する主なレジスタやビットを以下に概説します.

- `TMR0L`
  8 ビットのレジスタで, タイマの値を保持します.
- `T0CON0.T0EN`
  TMR0 のオン・オフを設定するビットです. 1 の場合はオンで, 0 の場合はオフです.
- `T0CON0.T016BIT`
  16 ビットモードを設定するビットです. 1 の場合は 16 ビットモードで, 0 の場合は 8 ビットモードです. 16 ビットモードの場合には, `TMR0H` を上位 8 ビット, `TMR0L` を下位 8 ビットとする 16 ビットタイマとして使用可能です.
- `T0CON0.T0OUTPS`
  出力ポストスケーラを指定します. 1:1 から 1:16 までの比率を指定可能で, 指定した比率にタイマ出力を分周します.



(a) 入力　　**(b)** 出力

図 27.9: チャタリングの除去

- `T0CON1.T0CS`
  クロック源を設定します.
- `T0CON1.T0CKPS`
  プリスケーラを指定します. 1/1, 1/2, 1/4, ..., 1/32768 の比率を指定可能で, 指定した比率にクロック周波数を分周します.
- `PIR0.TMR0IF`
  TMR0 割り込み発生フラグです. オーバーフローが発生してタイマの値が 0 に戻ったときにこのビットが 1 になります.
- `PIE0.TMR0IE`
  TMR0 割り込み許可ビットです.

27.5　チャタリングの除去

チャタリング (chattering) とはスイッチのオン・オフを切り替える際に図 27.9 で示すように短時間に何度も開閉が発生する現象です. これはスイッチの可動接点部分で機械的に細かな振動が発生して, 接点の接触・非接触が頻繁に生じることに起因します. いちどスイッチを切り替えただけのつもりなのに短時間に何度もスイッチのオン・オフが発生して, 回路やファームウエアが期待外れの動作となることがあります. 本ファームウエアでワンショット型の再生を行う場合には, 押しボタンスイッチを押して再生を開始した後に再び押しボタンスイッチを押すと再生を中断する動作仕様としています. この場合でのスイッチによる入力はオン, オフ, オンとなっています. この仕様は再生途中に強制終了ができて便利なのですが, チャタリングが発生すると誤動作が起きてしまいます.

チャタリングを解消する方法のいくつかを紹介します.

- ハードウエア的解法
  CR 積分回路を用いて短時間の頻繁なオン・オフによるパルスを吸収する手法です. さらに CR 積分回路の出力電圧の緩やかな上昇をヒステリシス特性を持ったシュミットトリガ回路を通して波形をオン・オフの二値に整形します.
- ソフトウエア的解法
  スイッチの状態が一定時間以上一定したらスイッチの読み出し値を確定する手法です. チャタリング期間中はスイッチの状態は何度も変化しますが, チャタリング期間終了後は変化が生じないことに基づいた手法です.

本章では後者の方法であるソフトウエア的解法を用いてチャタリング除去をします.

27.6　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

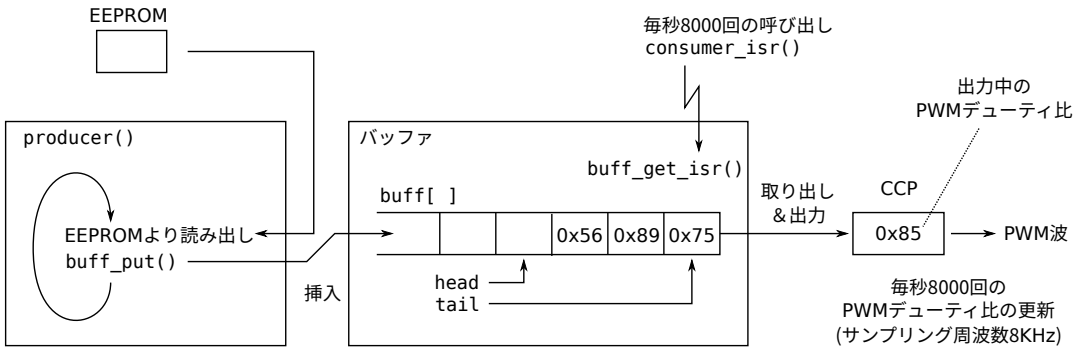


図 27.10: サウンドデータの読み出しから PWM 波の出力までの流れ

表 27.2: 【ソースファイル一覧】 EEPROM メモリ型 PCM サウンド器

初	頁	番号	ファイル名	内容	行数
✓	496	27.2	SoundPlayer-EEPROM.c	主ソースコード	105
✓	498	27.3	sw.h	再生ボタン読み込み: ヘッダ	8
✓	498	27.4	sw.c	再生ボタン読み込み: 実装	53
✓	501	27.6	buff.h	PCM データバッファ部: ヘッダ	12
✓	501	27.7	buff.c	PCM データバッファ部: 実装	80
✓	503	27.8	producer.h	PCM データ生産部: ヘッダ	6
✓	503	27.9	producer.c	PCM データ生産部: 実装	43
✓	504	27.10	consumer.h	PCM データ消費部: ヘッダ	6
✓	504	27.11	consumer.c	PCM データ消費部: 実装	17
	478	26.2	snd-eeprom.h	サウンド EEPROM アクセス関数: ヘッダ	54
✓	499	27.5	snd-eeprom.c	サウンド EEPROM アクセス関数: 実装	100
	394	23.3	i2c.h	I^2C インターフェース関数: ヘッダ	25
	394	23.4	i2c.c	I^2C インターフェース関数: 実装	111
	459	25.19	i2c-24fc1025.h	24FC1025 アクセス関数: ヘッダ	18
	459	25.20	i2c-24fc1025.c	24FC1025 アクセス関数: 実装	135
✓	494	27.1	io.h	I/O ピン割付: ヘッダ	16
✓	505	27.12	mcr.h	MCR・周辺モジュール: ヘッダ	18
✓	505	27.13	mcr.c	MCR・周辺モジュール: 実装	174
				計 981 行 (うち初出 638 行, 既出 343 行)	

本ファームウエアの骨子を図 27.11 で少し詳しく示しています. EEPROM からサウンドデータを 1 バイト読み出し, その 1 バイトの値をデューティ比とする PWM 波を出力するという単純なプログラム構造です. これを図示したのが図 27.10 です.

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

```
【ソースファイル SoundPlayer-EEPROM.c】

void main(void)
{
  初期化;
  for (;;) {
    再生開始ボタンのオン(再生指令)を待機;
    if (リピート型) {
      producer();   /* 開始部の再生 */
      while (再生開始ボタンがオン) {
        producer(); /* 繰返部の再生 */
      }
      producer();   /* 終了部の再生 */
    } else if (ワンショット型) {
      producer();   /* 再生 */
    }
    バッファが空になるまで待機;
    再生開始ボタンのオフを待機;
  }
}

【ソースファイル producer.c】

/* main関数からの呼び出し */
void producer()
{
  for (;;) {
    while (バッファに空きが全くない) {
      ;
    }
    EEPROMより1バイト読み出してバッファに挿入;
  }
}

【ソースファイル consumer.c】

/* 毎秒8000回のタイマ割り込みによる呼び出し */
void consuer_isr()
{
  バッファより1バイト読み出してPWM波デューティー比を設定;
}
```

図 27.11: ファームウエア構造 (詳細)

ソースファイル番号 27.1: io.h

```
#if !defined(__IO_H__)
#define __IO_H__

/* サウンド再生中をLED表示(再生中はHi): RA0 */
#define LED_MONITOR_ON()          (LATA0 = 1)
#define LED_MONITOR_OFF()         (LATA0 = 0)
#define LED_MONITOR_TOGGLE()      (LATA0 = ~LATA0)
/* サウンド再生開始の制御入力(Low入力で再生開始): RA3 */
#define SW_PLAY()                 (RA3 == 0)
/* サウンド再生中の制御出力(再生時にLow): RA4 */
#define OUT_PLAYING_NOW()         (LATA4 = 1)
#define OUT_PLAYING_NONE()        (LATA4 = 0)
/* I2C SCLピン: RA1 */
/* I2C SDAピン: RA2 */

#endif /*__IO_H__*/
```

ソースファイル SoundPlayer-EEPROM.c (主ソースコード)

以下の動作を繰り返すことでサウンドの再生を制御します. 繰り返し一度につき一度の再生となっていることに注意して下さい.

1. PCM サウンドデータの再生で使用するバッファを初期化します.
2. 制御入力から再生開始指令が出るまで待機します.
3. EEPROM のインデックス情報を読み出します. リピート型とワンショット型の区別も読み出します.
4. EEPROM 内の PCM サウンドデータを再生します. リピート型かワンショット型であるかに従い, 繰り返し再生の制御を行います. 実際のサウンドの再生の動作は関数 producer() の中で行います.
5. バッファが空になるまで, すなわちすべての PCM データの再生が実際に完了するまで待機します.
6. 制御入力からの再生開始指令が終了するまで待機します. これは最初の再生開始指令が出てから, 再生中もずっと出されたままの場合への対処です. もしこの対処がなければすぐさま次の再生が始まってしまい具合がよろしくありません.

ソースファイル番号 27.2: SoundPlayer-EEPROM.c

```
/*************************************************
 * サウンドファイル再生器 EEPROM(24FC1025)版 / PIC16F18313
 * by Hirotsugu Kakugawa
 *************************************************/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "i2c.h"
#include "i2c-24fc1025.h"
#include "snd-eeprom.h"
#include "producer.h"
#include "consumer.h"
#include "buff.h"
#include "sw.h"

/*-------------------------------------------------------
 * MAIN
 *-------------------------------------------------------*/
void main(void)
{
  /* システム初期化 */
  System_Init();
  /* ウオッチドッグタイマ初期化: 2 [Sec] */
  WDTCONbits.WDTPS = 0x0B; /* 0b01011 */
```

```
26    CLRWDT();
27    /* 出力初期化 */
28    LED_MONITOR_OFF();
29    OUT_PLAYING_NONE();
30    /* 再生SW状態読み取り初期化 */
31    sw_reader_init();
32    /* 主ループ */
33    for (;;) {
34      CLRWDT();
35      /* PWMバッファ初期化 */
36      buff_init();
37      /* 割込み許可 */
38      INTERRUPT_ENABLE();
39      /* 制御入力から再生開始指令 (SW ON) を待機 */
40      while (!sw_reader_get_play()) {
41        CLRWDT();
42      }
43      /* EEPROMチェック& インデックス情報の読込み */
44      __delay_ms(100);
45      CLRWDT();
46      enum sound_type_e st = eeprom_init();
47      /* LED表示(再生中) */
48      OUT_PLAYING_NOW();
49      LED_MONITOR_ON();
50      __delay_ms(1);
51      /* 再生開始 */
52      if (st == SOUND_TYPE_REPEAT) { /* リピート型 */
53        /* 開始部 */
54        if (eeprom_index_0.size != 0) {
55          producer(&eeprom_index_0, 0);
56        }
57        /* 繰返部 */
58        if (eeprom_index_1.size != 0) {
59          while (sw_reader_get_play()) {
60            producer(&eeprom_index_1, 0);
61            OUT_PLAYING_NONE();
62          }
63        }
64        OUT_PLAYING_NONE();
65        /* 終了部 */
66        if (eeprom_index_2.size != 0) {
67          producer(&eeprom_index_2, 0);
68        }
69      } else if (st == SOUND_TYPE_ONESHOT) { /* ワンショット型 */
70        if (eeprom_index_0.size != 0) {
71          producer(&eeprom_index_0, 1);
72        }
73        OUT_PLAYING_NONE();
74      }
75      /* バッファ中の全データの出力完了を待機 */
76      while (!buff_is_empty()) {
77        CLRWDT();
78      }
79      /* 終了 */
80      LED_MONITOR_OFF();
81      __delay_ms(1);
82      /* 再生解除 (SW OFF) を待機 */
83      while (sw_reader_get_play()) {
84        CLRWDT();
85      }
86      __delay_ms(100);
87      /* 割込み禁止 */
88      INTERRUPT_DISABLE();
89    }
90  }
91
92  /* LED点滅によるエラー通知 */
93  void fatal(void)
94  {
95    CLRWDT();
```

```
96      uint8_t t = 0;
97      for (t = 0; t < 25; t++) {
98        CLRWDT();
99        LED_MONITOR_TOGGLE();
100       __delay_ms(100);
101     }
102     CLRWDT();
103     LED_MONITOR_OFF();
104     __delay_ms(500);
105   }
```

ソースファイル sw.h (再生ボタン読み込み: ヘッダ)

再生開始ボタンの押下を判定する関数群のプロトタイプ宣言です.

ソースファイル番号 27.3: sw.h

```
1   #if !defined(__SW_H__)
2   #define __SW_H__
3
4   extern void    sw_reader_init(void);
5   extern int8_t  sw_reader_get_play(void);
6   extern void    sw_reader_isr(void);
7
8   #endif /*__SW_H__*/
```

ソースファイル sw.c (再生ボタン読み込み: 実装)

再生開始ボタンの押下を判定する関数群の実装を行います. このソースファイルの目的はチャタリング除去をソフトウエア的に行うことであり, その原理は以下の通りです.

- タイマ割り込みを使用して一定時間間隔で再生開始ボタンの状態を読みます.
- 再生開始ボタンが押下の状態が一定回数連続すれば押下にあると判断します. 再生開始ボタンが開放の状態の時も同様で, 開放の状態が一定回数連続すれば開放にあると判断します.
- 押下と開放の状態が頻繁に変われば判断は前回のままとします.

なおソースファイル中では再生開始ボタンを記号 sw あるいは SW で表しています.

具体的には以下の有限状態機械によりチャタリングによる不安定な入力を安定化します.

- 状態 0: 再生開始ボタンがオフ
  20 回連続して再生開始ボタンがオンであれば状態 1 に遷移します. ひとたび再生開始ボタンがオフになればゼロから勘定し直します.
- 状態 1: 再生開始ボタンがオン
  20 回連続して再生開始ボタンがオフであれば状態 0 に遷移します. ひとたび再生開始ボタンがオンになればゼロから勘定し直します.

ソースファイル番号 27.4: sw.c

```
1   /*-------------------------------------------------------
2    * SW状態読み取り
3    *   タイマ割り込みで定期的に読み取り、チャタリング除去する
4    *-------------------------------------------------------*/
5   #include <xc.h>
6   #include <stdint.h>
7   #include "mcr.h"
8   #include "io.h"
9   #include "sw.h"
```

```
10
11  #define SW_CNT_THREASHOULD  20
12  volatile int8_t  sw_state_play_val = 0;
13  volatile uint8_t sw_state_play_cnt = 0;
14
15  void sw_reader_init(void)
16  {
17    sw_state_play_val = SW_PLAY();
18    sw_state_play_cnt = SW_CNT_THREASHOULD;
19  }
20
21  int8_t sw_reader_get_play(void)
22  {
23    return sw_state_play_val;
24  }
25
26  void sw_reader_isr(void)
27  {
28    switch (sw_state_play_val) {
29    default:
30    case 0: /* SW OFF */
31      if (!SW_PLAY()) {
32        sw_state_play_cnt = SW_CNT_THREASHOULD;
33      } else {
34        if (sw_state_play_cnt != 0) {
35          sw_state_play_cnt--;
36        } else {
37          sw_state_play_val = 1;
38        }
39      }
40    break;
41    case 1: /* SW ON */
42      if (SW_PLAY()) {
43        sw_state_play_cnt = SW_CNT_THREASHOULD;
44      } else {
45        if (sw_state_play_cnt != 0) {
46          sw_state_play_cnt--;
47        } else {
48          sw_state_play_val = 0;
49        }
50      }
51      break;
52    }
53  }
```

ソースファイル snd-eeprom.c (サウンド EEPROM アクセス関数: 実装)

EEPROM 内の PCM サウンドデータ構造を読み取ります. このソースファイルの主な目的は EEPROM 内にある PCM サウンドデータのバイト数と配置アドレスの読み出しで, 以下の内容を実現します.

- EEPROM のハードウエア的な接続の有無の検査
  I^2C バスに EEPROM が接続されているか否かを検査します.
- PCM サウンドデータの種別の読み出し
  EEPROM 内のマジックナンバを参照してワンショット型かリピート型かを判定します.
- PCM サウンドデータのバイト数と配置アドレスの読み出し
  ワンショット型, リピート型それぞれのサウンド EEPROM データファイルのデータ構造に合わせて読み出します.

ソースファイル番号 27.5: snd-eeprom.c

```
1  /*-------------------------------------------------------
2   * サウンドEEPROMへのアクセス
3   *-------------------------------------------------------*/
```

```
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "io.h"
8  #include "snd-eeprom.h"
9  #include "i2c.h"
10 #include "i2c-24fc1025.h"
11
12 /* EEPROMインデックス情報 */
13 struct eeprom_index_s eeprom_index_0;
14 struct eeprom_index_s eeprom_index_1;
15 struct eeprom_index_s eeprom_index_2;
16
17 static enum sound_type_e eeprom_check(void);
18 static void eeprom_read_index(uint8_t sndidx,
19                               struct eeprom_index_s *h);
20 extern void fatal(void);
21
22 /* EEPROM接続を確認しサウンド種別を返す */
23 enum sound_type_e eeprom_init(void)
24 {
25   /* I2C EEPROM の接続を検査 */
26   while (!I2C_Exist(I2C_24FC1025_CTLBYTE, 0)) { /* 無接続 */
27     CLRWDT();
28     fatal();
29   }
30   /* サウンド種別を判定 */
31   enum sound_type_e st = SOUND_TYPE_UNKNOWN;
32   while (st == SOUND_TYPE_UNKNOWN) {
33     st = eeprom_check();
34     switch (st) {
35     case SOUND_TYPE_REPEAT:  /* リピート型 */
36       eeprom_read_index(INDEX_0, &eeprom_index_0);
37       eeprom_read_index(INDEX_1, &eeprom_index_1);
38       eeprom_read_index(INDEX_2, &eeprom_index_2);
39       break;
40     case SOUND_TYPE_ONESHOT: /* ワンショット型 */
41       eeprom_read_index(INDEX_0, &eeprom_index_0);
42       break;
43     default:
44       fatal();
45     }
46   }
47   return st;
48 }
49
50 /* 接続EEPROMのサウンド種別を返す */
51 static enum sound_type_e eeprom_check(void)
52 {
53   /* I2C EEPROM データ内容の種別を検査 */
54   enum sound_type_e st = SOUND_TYPE_UNKNOWN;
55   uint8_t b0, b1, b2, b3;
56   /* マジックナンバーを読んで種別を判定 */
57   i2c_24fc1025_read_begin(0, 0, 4);
58   b0 = i2c_24fc1025_read_byte();
59   b1 = i2c_24fc1025_read_byte();
60   b2 = i2c_24fc1025_read_byte();
61   b3 = i2c_24fc1025_read_byte();
62   if        ((b0 == EEPROM_MAGIC_NUMBER_0) &&
63              (b1 == EEPROM_MAGIC_NUMBER_1) &&
64              (b2 == EEPROM_MAGIC_NUMBER_2) &&
65              (b3 == EEPROM_MAGIC_NUMBER_3_REPEAT)) {
66     st = SOUND_TYPE_REPEAT;  /* リピート型 */
67   } else if ((b0 == EEPROM_MAGIC_NUMBER_0) &&
68              (b1 == EEPROM_MAGIC_NUMBER_1) &&
69              (b2 == EEPROM_MAGIC_NUMBER_2) &&
70              (b3 == EEPROM_MAGIC_NUMBER_3_ONESHOT)) {
71     st = SOUND_TYPE_ONESHOT; /* ワンショット型 */
72   } else { /* これはサウンドデータではない */
73     fatal();
74   }
```

```
75    i2c_24fc1025_read_done();
76    return st;
77  }
78
79  /* サウンドEEPROMのヘッダ情報の読出 */
80  static void eeprom_read_index(uint8_t sndidx,
81                                struct eeprom_index_s *h)
82  {
83    uint8_t i;
84    uint32_t d0, d1, d2, d3;
85    /* 読出アドレスの設定 */
86    i2c_24fc1025_read_begin(0, 4 + (unsigned)sndidx * 8, 8);
87    /* データ開始アドレスの読出 */
88    d0 = i2c_24fc1025_read_byte();
89    d1 = i2c_24fc1025_read_byte();
90    d2 = i2c_24fc1025_read_byte();
91    d3 = i2c_24fc1025_read_byte();
92    h->addr = (d0) + (d1 << 8) + (d2 << 16) + (d3 << 24);
93    /* データ長の読出 */
94    d0 = i2c_24fc1025_read_byte();
95    d1 = i2c_24fc1025_read_byte();
96    d2 = i2c_24fc1025_read_byte();
97    d3 = i2c_24fc1025_read_byte();
98    h->size = (d0) + (d1 << 8) + (d2 << 16) + (d3 << 24);
99    i2c_24fc1025_read_done();
100 }
```

ソースファイル buff.h (PCM データバッファ部: ヘッダ)

PCM データバッファへのアクセスを行う関数群のプロトタイプ宣言を行います.

ソースファイル番号 27.6: buff.h

```
1   #if !defined(__BUFF_H__)
2   #define __BUFF_H__
3
4   #define BUFF_SIZE   0x40  /* バッファ容量 */
5
6   extern void     buff_init(void);
7   extern void     buff_put(uint8_t code);
8   extern uint8_t  buff_get_isr(void);
9   extern int8_t   buff_is_empty(void);
10  extern int8_t   buff_is_full(void);
11
12  #endif /*__BUFF_H__*/
```

ソースファイル buff.c (PCM データバッファ部: 実装)

PCM データバッファへのアクセスを行う関数群の実装を行います. データ構造には第 25.5 節で紹介した巡回バッファを使用します.

ソースファイル番号 27.7: buff.c

```
1   /*-------------------------------------------------------
2    * PCMデータバッファ
3    *   巡回バッファ法でデータ管理
4    *------------------------------------------------------- */
5   #include <xc.h>
6   #include <stdint.h>
7   #include "mcr.h"
8   #include "io.h"
9   #include "i2c-24fc1025.h"
10  #include "buff.h"
11  #include "snd-eeprom.h"
12  #include "sw.h"
13
14  static volatile uint8_t  buff[BUFF_SIZE];
```

```
15  static volatile uint8_t  nitems;
16  static volatile uint8_t  head;
17  static volatile uint8_t  tail;
18
19  /* バッファを初期化 */
20  void buff_init(void)
21  {
22    nitems = 0;
23    head = 0;
24    tail = 0;
25  }
26
27  /* バッファが空か否か検査 */
28  int8_t buff_is_empty(void)
29  {
30    INTERRUPT_DISABLE();
31    int8_t is_empty = (nitems == 0);
32    INTERRUPT_ENABLE();
33    return is_empty;
34  }
35
36  /* バッファが満杯か否か検査 */
37  int8_t buff_is_full(void)
38  {
39    INTERRUPT_DISABLE();
40    int8_t is_full = (nitems == BUFF_SIZE);
41    INTERRUPT_ENABLE();
42    return is_full;
43  }
44
45  /* バッファにデータを挿入 */
46  void buff_put(uint8_t code)
47  {
48    /* バッファに空が出るのを待つ */
49    for (;;) {
50      INTERRUPT_DISABLE();
51      int8_t is_full = (nitems == BUFF_SIZE);
52      INTERRUPT_ENABLE();
53      if (!is_full) { /* バッファに空あり */
54        break;
55      }
56    }
57    /* バッファに挿入 */
58    buff[head++] = code;
59    if (head == BUFF_SIZE) {
60      head = 0;
61    }
62    INTERRUPT_DISABLE();
63    nitems++;
64    INTERRUPT_ENABLE();
65  }
66
67  /* バッファよりデータを削除
68   *  (呼び出しは割込みハンドラからに限定: 割り込み禁止状態で実行)  */
69  uint8_t buff_get_isr(void)
70  {
71    uint8_t d = 0x80;
72    if (nitems != 0) { /* バッファにデータあり */
73      nitems--;
74      d = buff[tail++];
75      if (tail == BUFF_SIZE) {
76        tail = 0;
77      }
78    }
79    return d;
80  }
```

ソースファイル producer.h (PCM データ生産部: ヘッダ)

PCM データ生産部のプロトタイプ宣言を行います.

ソースファイル番号 27.8: producer.h

```
#if !defined(__PRODUCER_H__)
#define __PRODUCER_H__

extern void producer(struct eeprom_index_s *h, uint8_t stop_by_sw);

#endif /*__PRODUCER_H__*/
```

ソースファイル producer.c (PCM データ生産部: 実装)

PCM データ生産部の実装を行います．ここでの具体的な動作は EEPROM から PCM データを読み出してバッファにひとつでも空きが出れば即座に挿入をすることです．これに加えてワンショット型での再生強制終了はここで実現します. 再生開始ボタンがオフになった後に再びオンになると再生を中断します.

ここでの動作の上で重要な点は, 消費部がデータを読み尽くしてバッファが完全に空にならないように生産部はデータをバッファに挿入し続けなければならないということです．そのため再生中は EEPROM からのデータの読出しバッファへ挿入する以外の動作は極力控えるようにします. 再生開始ボタンのチャタリング処理と再生強制終了の判定だけが再生以外に関する動作です. 再生に関する動作であっても処理にかかる時間は極力少なくしています．EEPROM のインデックス情報は起動時の初期化の際に読み出していますので, インデックス情報の読み出しに要する時間は再生中には発生しません. EEPROM からのデータの読み出しには連続読み出しを行う I^2C プロトコルを用いていますので, 読み出しアドレスの設定に要する時間は 1 バイト読み出す度には発生しません.

時間間隔のばらつきはジッターと呼ばれている音質低下の原因ですから, PCM データの出力は正確な時間間隔で行わないといけません．しかしデータを EEPROM から読み出す時間間隔を正確に保つのは非常に困難です. なぜならマイコンの実行命令数を絶妙に調整しようとしても, EEPROM との I^2C 通信の時間の不確定性ために正確な時間間隔の実現は困難です．しかしタイマ割り込みを用いれば正確に時間間隔を保つことが可能で, 割り込みハンドラ内でバッファの内容を順に出力すれば正確な時間間隔で PCM データの出力が行えます．これが実時間処理における割り込みとバッファリングの重要性です.

ソースファイル番号 27.9: producer.c

```
/*------------------------------------------------------
 * PCMデータ生産者
 *   EEPROMからPCMデータを読み込む（一時的にバッファリング）
 *   順次PCMバッファに挿入
 *------------------------------------------------------ */
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "i2c-24fc1025.h"
#include "producer.h"
#include "buff.h"
#include "snd-eeprom.h"
#include "sw.h"

void producer(struct eeprom_index_s *h, uint8_t stop_by_sw)
{
  int8_t sw_state_prev = 1;
  int8_t sw_state_curr = sw_state_prev;
  /* サウンドデータを1バイトづつバッファに入れてゆく */
  uint32_t left_nbytes = left_nbytes = h->size;
  i2c_24fc1025_read_begin(0, h->addr, left_nbytes);
  for ( ; left_nbytes != 0; left_nbytes--) {
    CLRWDT();
    /* バッファに空きが出るのを待つ */
    while (buff_is_full()) {
      CLRWDT();
```

```
28      }
29      /* EEPROMより１バイト読んでデータをバッファに挿入 */
30      buff_put(i2c_24fc1025_read_byte());
31      /* SWによる再生中断(ONESHOT型再生時) */
32      if (stop_by_sw == 1) {
33        /* 一旦再生 SW が OFF となったのち、
34         * 次に再生 SW が ON になれば，再生を中断. */
35        if (!sw_state_prev && sw_state_curr) {
36          break;
37        }
38        sw_state_prev = sw_state_curr;
39        sw_state_curr = sw_reader_get_play();
40      }
41    }
42    i2c_24fc1025_read_done();
43  }
```

ソースファイル consumer.h (PCM データ消費部: ヘッダ)

PCM データ消費部のプロトタイプ宣言を行います.

ソースファイル番号 27.10: consumer.h

```
1  #if !defined(__CONSUMER_H__)
2  #define __CONSUMER_H__
3
4  extern void consumer_isr(void);
5
6  #endif /*__CONSUMER_H__*/
```

ソースファイル consumer.h (PCM データ消費部: 実装)

PCM データ消費部の実装を行います. 消費部は毎秒 8000 回の頻度のタイマ割り込みで呼び出され, PCM データの値に相当するデューティ比の PWM 波を発生します. これによりサンプリング周波数 8kHz の PCM サウンドデータの再生が行われます. PCM サウンドデータは 8 ビットですがデューティ比の指定は 10 ビットのため, PCM サウンドデータを 2 ビット左へシフトして 10 ビットに拡張します. ソフトウエアは PWM 波のデューティ比を決めるだけで, ハードウエアがアナログサウンドに変換します.

ソースファイル番号 27.11: consumer.c

```
1  /*-----------------------------------------------------
2   * データ消費者
3   *   PWMモジュールにPCMデータを出力
4   *   (8kHz周波数のタイマ割り込みで呼び出される)
5   *-----------------------------------------------------*/
6  #include <xc.h>
7  #include <stdint.h>
8  #include "mcr.h"
9  #include "io.h"
10 #include "consumer.h"
11 #include "buff.h"
12
13 void consumer_isr(void)
14 {
15   /* PCMデータは8ビット: PWMデューティー値10ビットに変換 */
16   CCP1_SetDutyCycle((uint16_t)buff_get_isr() << 2);
17 }
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 27.12: mcr.h

```
#if !defined(__MCR_H__)
#define __MCR_H__

#define _XTAL_FREQ 32000000  /* 32MHz */

extern void System_Init (void);
extern void CCP1_Init (void);
extern void CCP1_SetDutyCycle(uint16_t duty);
extern void TMR0_Init(void);
extern void TMR1_Init(void);
extern void TMR2_Init(void);

#define INTERRUPT_ENABLE()                          \
  { INTCONbits.GIE = 1; INTCONbits.PEIE = 1; }
#define INTERRUPT_DISABLE()                         \
  { INTCONbits.GIE = 0; INTCONbits.PEIE = 0; }

#endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います. 再生開始ボタンのチャタリング除去のために TMR0 による 5.12 ミリ秒間隔の割り込みを発生させます. サウンドの再生用のために TMR1 による 8kHz すなわち 125 マイクロ秒間隔の割り込みを発生させます. 音質の観点からは PWM 波は可能な限り高い周波数を用いるのが良いですから, TMR2 は設定可能な最高の周波数の 31.25kHz で動作させます.

ソースファイル番号 27.13: mcr.c

```
/*-----------------------------------------------------
 * MCR設定&周辺モジュール (PIC16F18313)
 *-----------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "i2c.h"
#include "consumer.h"
#include "sw.h"

/*-------------------------------------------*/
/* MCR初期化 */
void System_Init(void)
{
  /* 発振器 (FOSC=32MHz) */
  OSCCON1 = 0x00; /* HFINTOSC with 2x PLL */
  OSCCON3 = 0x00;
  OSCEN   = 0x00;
  OSCFRQ  = 0x06; /* 32MHz */
  OSCTUNE = 0x00;
  while (PLLR == 0) {
    ;
  }
  /* I/Oピン */
  LATA   = 0x00;
  TRISA  = 0x0E; /* A3,A2,A1 */
  ANSELA = 0x00;
  WPUA   = 0x00;
  ODCONA = 0x00;
  int8_t ogie = GIE;
  GIE = 0;
  PPSLOCK = 0x55;
  PPSLOCK = 0xAA;
  PPSLOCKbits.PPSLOCKED = 0x00;
  RA1PPS     = 0x18; /* RA1 <- SCL1 */
  RA2PPS     = 0x19; /* RA2 <- SDA1 */
  SSP1CLKPPS = 0x01; /* RA1 -> SCL1 */
```

```
39    SSP1DATPPS = 0x02; /* RA2 -> SDA1 */
40    RA5PPS     = 0x0C; /* RA5 <- CCP1 */
41    PPSLOCK = 0x55;
42    PPSLOCK = 0xAA;
43    PPSLOCKbits.PPSLOCKED = 0x01;
44    GIE = ogie;
45    /* 周辺モジュール */
46    I2C_Init(I2C_BAUD_RATE_SSPxADD_1MHZ(_XTAL_FREQ));
47    TMR0_Init();
48    TMR1_Init();
49    TMR2_Init();
50    CCP1_Init();
51  }
52
53  /*-------------------------------------------*/
54  /* TMR0 */
55  /* スイッチ入力チャタリング除去用タイマ
56   * -  5.12mSec 間隔で割り込み発生
57   */
58  void TMR0_Init(void)
59  {
60    T0CON0bits.T0EN    = 1;   /* Enabled */
61    T0CON0bits.T016BIT = 0;   /* TMR0 is 8-bit Timer */
62    T0CON1bits.T0CS    = 2;   /* Clock Source: FOCS/4  */
63    T0CON1bits.T0CKPS  = 0x8; /* Prescaler: 1:256   */
64    TMR0L = 0;
65    TMR0IF = 0;
66    TMR0IE = 1;
67  }
68
69  void TMR0_isr(void)
70  {
71    TMR0L = 96;
72    sw_reader_isr();
73  }
74
75  /*-------------------------------------------*/
76  /* TMR1 */
77  /* サウンド再生用タイマ
78   * - サウンドのサンプリング周波数が 8kHz のため
79   *   8kHz (125uSec) 周期でタイマ割込
80   * - Clock Source: FOSC (= 32MHz), Prescale 1:1 で設定
81   * - 32MHz/8kHz = 4000カウントごとに割込みを発生させればよい.
82   * - タイマの初期値を 0x10000-4000=61536=0xF060 と設定.
83   *   (4000回カウントするとオーバーフローが発生し、割込み発生)
84   */
85  void TMR1_Init(void)
86  {
87    T1CONbits.TMR1ON = 0; /* Disabled */
88    T1CONbits.TMR1CS = 0x0; /* Clock Sorce: FOCS/4 */
89    T1CONbits.T1CKPS = 0x0; /* Prescale: 1:1 */
90    TMR1H = 0;
91    TMR1L = 0;
92    TMR1IF = 0;
93    TMR1IE = 1;
94    T1CONbits.TMR1ON = 1; /* Enabled */
95  }
96
97  void TMR1_isr(void)
98  {
99    TMR1H = 0xFC;
100   TMR1L = 0x18;
101   consumer_isr();
102 }
103
104 /*-------------------------------------------*/
105 /* TMR2 */
106 /* PWM出力用タイマ: PWM周波数 = 31.25kHz
107  *      (32MHz/(31.25kHz*4*1)) - 1 = 255 = PR2
108  */
109 void TMR2_Init(void)
```

```
{
  T2CONbits.TMR2ON = 0;
  T2CONbits.T2OUTPS = 0; /* Postscaler: 1:1 */
  T2CONbits.T2CKPS  = 0; /* Prescaler:  1:1 */
  PR2  = 0xFF;
  TMR2 = 0;
  TMR2IF = 0;
  TMR2IE = 0;
  T2CONbits.TMR2ON = 1;
}

/*-------------------------------------------*/
/* CCP1 */
void CCP1_Init (void)
{
  CCP1CONbits.CCP1EN   = 1;   /* Enabled */
  CCP1CONbits.CCP1FMT  = 0;   /* 右寄せ形式 */
  CCP1CONbits.CCP1MODE = 0xF; /* PWMモード */
  CCPR1L = 0x00;
  CCPR1H = 0x00;
}

void CCP1_SetDutyCycle(uint16_t duty)
{
  CCPR1H = duty >> 8; /* 2 MSBs */
  CCPR1L = duty;      /* 8 LSBs */
}

/*-------------------------------------------*/
/* 割り込みハンドラ */
__interrupt() void isr(void)
{
  if ((TMR0IE == 1) && (TMR0IF == 1)) {
    TMR0IF = 0;
    TMR0_isr();
  }
  if ((TMR1IE == 1) && (TMR1IF == 1)) {
    TMR1IF = 0;
    TMR1_isr();
  }
}

/*-------------------------------------------*/
/* CONFIG1 */
#pragma config FEXTOSC  = OFF
#pragma config RSTOSC   = HFINT32
#pragma config CLKOUTEN = OFF
#pragma config CSWEN    = ON
#pragma config FCMEN    = ON
/* CONFIG2 */
#pragma config MCLRE    = OFF /* MCLRをI/Oピン */
#pragma config PWRTE    = OFF
#pragma config WDTE     = ON /* WDT有効, SWDTEN無視 */
#pragma config LPBOREN  = OFF
#pragma config BOREN    = ON
#pragma config BORV     = LOW
#pragma config PPS1WAY  = ON
#pragma config STVREN   = ON
#pragma config DEBUG    = OFF
/* CONFIG3 */
#pragma config WRT      = OFF
#pragma config LVP      = OFF /* 低電圧書込不可 */
/* CONFIG4 */
#pragma config CP       = OFF
#pragma config CPD      = OFF
```

課題

1. 本装置の回路で用いているローパスフィルタ回路の遮断周波数 f_c の式を導出して下さい.
2. 本装置では搭載する EEPROM はひとつだけですが, 24FC1025 を最大 4 個まで装着可能とするように

ハードウエアとファームウエアの両方を拡張して下さい.

3. ランダムな時間間隔で再生を行う制御装置を設計・製作して下さい. 本章の装置とは別個の装置とし, 本章の装置へのアクセサリ制御入力に信号を発生させる形で実現して下さい.
4. 記憶媒体に SD カードを使用した PCM サウンド再生器を設計・製作して下さい.

第 28 章

EEPROM プログラム型
ポイント一括切替器

本章ではポイントの一括切替を行う装置を紹介します. この装置は一度の操作で複数のポイントを一括して切り替える機能を有しており, 駅やヤードでのポイント切り替え操作を大幅に簡略化します. この種の装置は経路設定装置とも呼ばれており, ダイオードマトリクスによる回路がこれまで発表されています. 本章ではマイコンを使用して, カスタマイズ性と拡張可能性を備えた装置を紹介します.

本章の回路とファームウエアの特徴は以下の通りです.

- 順次切替機能
  本装置では複数のポイントひとつひとつ順に切り替えます. もし複数のポイントの切り替えを同時に行えば一度に大電流が流れてブレーカが落ちてしまいますが, 本装置はこの問題点を解決します.
- カスタマイズ性
  本装置では経路設定データを EEPROM に記憶することで経路設定の変更が容易です. 従来の同種の装置はダイオードマトリクスを使用しており, 経路設定の変更には回路の変更が伴っていました.
- 拡張可能性
  ポイントインターフェースボードは最大 8 枚までの任意の枚数を接続可能です. 接続枚数を変えてもファームウエアの変更は一切不要です.
- 最大 64 のポイントマシンを制御可能
  各ポイントインターフェースボードでは 8 つのポイントマシンが接続可能で, 本装置全体では最大 64 の

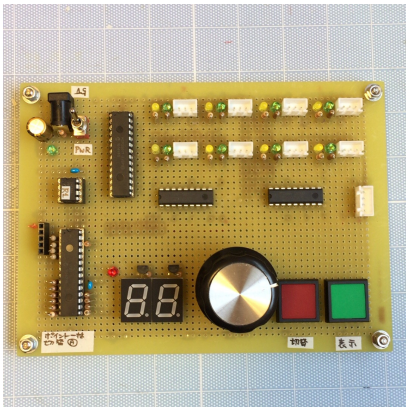


図 28.1: ポイント一括切替器の試作例 (メインボードとポイントインターフェースボード 1 つ分の機能を 1 枚の基板上に実装)

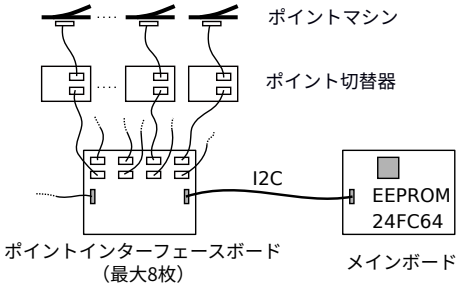


図 28.2: 接続方法

```
1  main()
2  {
3    route_idを選択中の経路番号とする;
4    for (;;) {
5      if (ロータリーエンコーダが右に回転) {
6        route_idを増加;
7      } else if (ロータリーエンコーダが左に回転) {
8        route_idを減少;
9      }
10     route_idを7セグメントLEDに表示;
11     if (経路内容の表示スイッチが押された) {
12       route_idに対する定義内容をEEPROMから読み，7セグメントLEDに表示;
13     } else if (経路の切り替えスイッチが押された) {
14       route_idに対する定義内容をEEPROMから読み，ポイントの切替を実行;
15     }
16   }
17 }
```

図 28.3: ファームウエア構造 (概要)

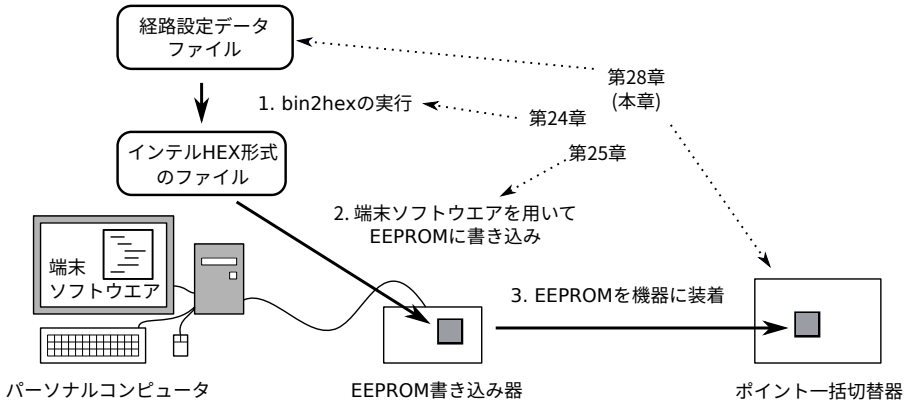


図 28.4: 経路一括切替器の利用のための作業手順

ポイントマシンを接続可能です.

本章で学ぶ内容は以下の通りです.

- ロータリエンコーダの使用方法

この装置のファームウエアの大まかな構造を図 28.3 に示します. ロータリエンコーダにより経路番号を選択して, 切り替えスイッチが押されれば EEPROM 中の経路設定データに従ってポイントを切り替えてゆく, というのが基本構造です.

28.1　使用方法

本装置の使用方法を以下に説明します. 以下の説明で使用する部品名称は図 28.5 に配置例とともに示しています. 一連の作業の全体像は図 28.4 で示す通りです.

まず最初に以下の準備をします.

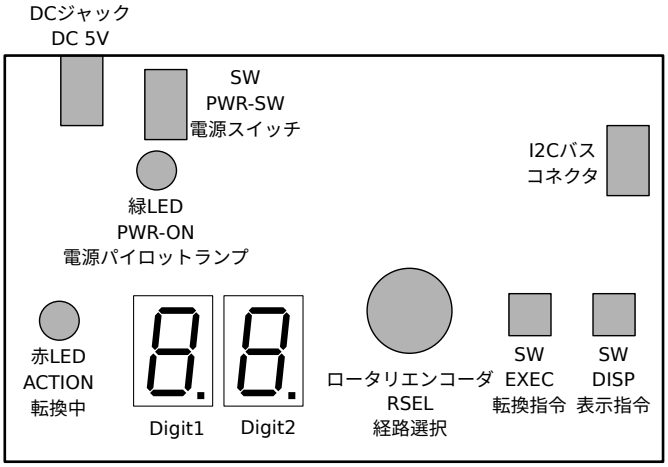


図 28.5: 操作表示部品の配置例と名称

1. 経路設定データを後述の形式に従って作成します. 作成した経路設定データは第 24 章で紹介したインテル HEX 形式変換ソフトウエア bin2hex を使用してインテル HEX 形式に変換し, 第 25 章で紹介した EEPROM 書き込み器を使用して EEPROM 24FC64 に書き込みます.
2. 経路設定データを書き込んだ EEPROM 24FC64 をメインボードに装着します.
3. 図 28.2 に示すようにメインボードとポイントインターフェースボードを接続します. ポイントインターフェースボードには第 12 章などで紹介したポイント切替器を接続し, 各ポイント切替器とポイントマシンを接続します. ポイントインターフェースボードには 8 つのポイント制御出力がありますが, 必ずしもそれらすべてにポイント切替器を接続する必要はありません.
4. ポイントインターフェースボードの DIP スイッチを設定します. ポイント番号とポイントインターフェースボード番号の関係と DIP スイッチの設定を表 28.1 に示します. で示した通りです. ポイントインターフェースボード番号 1 ではポイント番号 1 から 8, ポイントインターフェースボード番号 2 ではポイント番号 9 から 16 となっており, 以降同様です.
5. DC 5V の AC アダプタを本装置に接続します. **間違って 12V 出力の AC アダプタを使用すると回路が破損する場合がありますので, 十分に注意して下さい.**
6. 本装置の電源スイッチを入れます. このときファームウエアはハードウエア接続や EEPROM 内の経路設定データが正しいか否かを検査し, エラーがなければ 7 セグメント LED には **0** が表示されます. もし何らかのエラーが見つかれば 7 セグメント LED にエラーコードが表示されます. このときのエラーコードは表 28.2 に示す通りで, 経路設定データに関するエラー原因の詳細は後述の図 28.11 で説明します.

ポイント切り替えは以下の操作で行います.

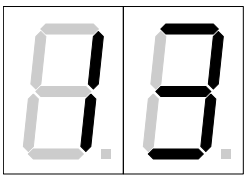
1. 経路番号の選択
   ロータリエンコーダを回して経路番号を選択します. このときの経路番号は 7 セグメント LED に表示されます. なお経路番号 0 は特別で, EEPROM 内の経路設定データとは無関係に全ポイントを定位に切り替えます.
2. ポイント切替内容の表示
   押しボタンスイッチ DISP の押下により選択中の経路番号での切り替え内容を 7 セグメント LED に表示します. ポイント番号は数字 2 桁で, 切替方向は小数点で, それぞれ表します. 小数点が消灯の場合に

表 28.1: ポイントインターフェースボードの DIP スイッチ設定

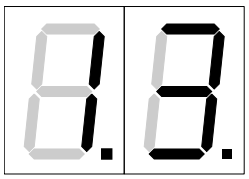
インターフェースボード番号	ポイント番号	DIP スイッチ設定		
		1	2	3
1	1 ~ 8	OFF	OFF	OFF
2	9 ~ 16	ON	OFF	OFF
3	17 ~ 24	OFF	ON	OFF
4	25 ~ 32	ON	ON	OFF
5	33 ~ 40	OFF	OFF	ON
6	41 ~ 48	ON	OFF	ON
7	49 ~ 56	OFF	ON	ON
8	57 ~ 64	ON	ON	ON

表 28.2: エラーコード表

表示	エラー原因
E1	EEPROM 無接続
E2	ポイントインターフェースボード無接続
E3	(経路設定データ) コマンドエラー
E4	(経路設定データ) 書式エラー
E5	(経路設定データ) 順序エラー
E6	(経路設定データ) 欠落エラー
E7	(経路設定データ) 行終端エラー
E8	(経路設定データ) ポイント番号範囲外



(a) ポイント番号 13: 定位 (小数点は消灯)



(b) ポイント番号 13: 反位 (小数点は点灯)

図 28.6: 7 セグメント LED によるポイント切替内容の表示例

は定位を, 点灯の場合には反位をそれぞれ表します. 図 28.6 に表示の例を示します. なおこの操作は表示をするだけで実際のポイント切替は行いません. また必ずしもこの操作を行う必要はありません.

3. ポイント切替の実行

   押しボタンスイッチ EXEC の押下により選択中の経路番号での切り替えを行います. 切り替え実行中にはポイント番号と切替方向を 7 セグメント LED に表示します. 図 28.6 に表示の例を示します. 経路番号 0 の場合にはすべてのポイントを定位に切り替えますが, このときにはいったん反位に切り替えた後に定位に切り替えます. これは第 14 章で紹介した簡易型ポイント切替器で切り替えを確実に行うための機能です.

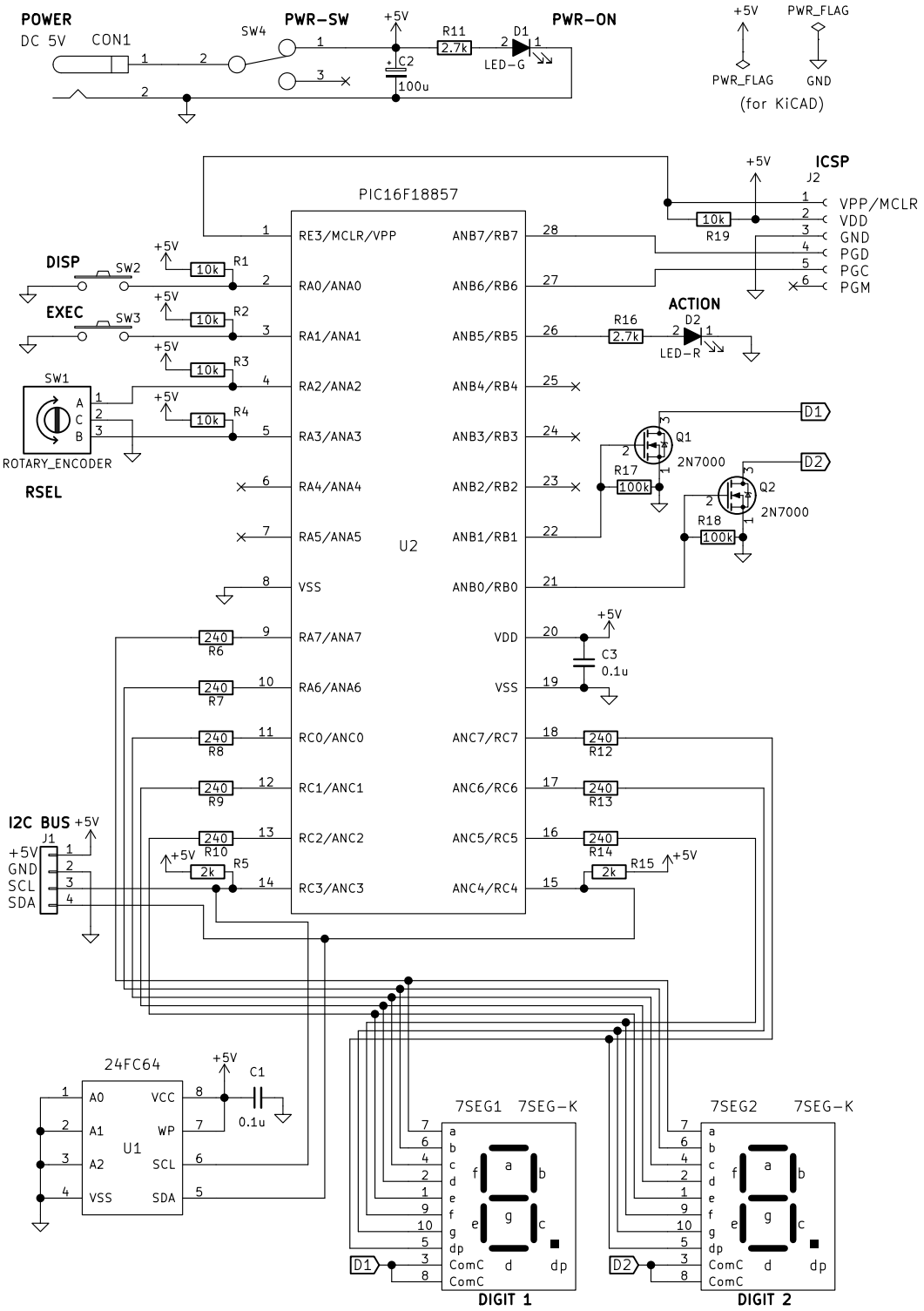


図 28.7: 【回路図】 プログラマブル式ポイント一括切替器/メインボード

28.2　回路図

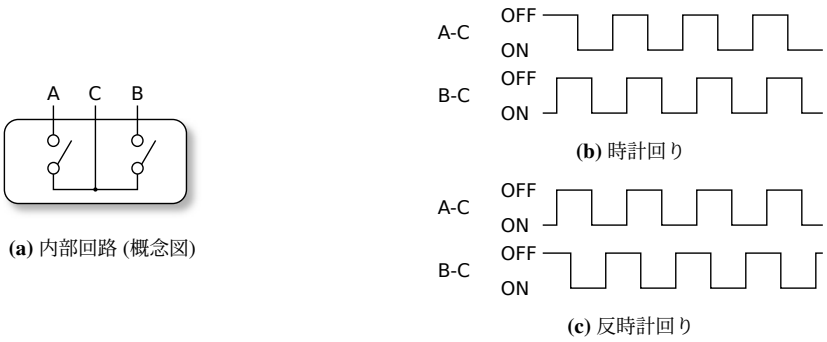
プログラマブル式ポイント一括切替器はメインボードとポイントインターフェースボードの 2 種のボードで構成されます. メインボードの回路図を図 28.7 に, 部品リストを表 28.3 に, それぞれ示します. ポイントインターフェースボードの回路図と部品リストは, 第 23 章の図 23.3 (381 ページ) と表 23.1 (382 ページ) で示したものをそのまま使用します. 部品表には以下のものは含まれていないので別途用意をして下さい.

- メインボードとポイントインターフェースボードを I^2C 接続するためのコネクタハウジングやケーブル
- ポイントインターフェースボードとポイント切替器を接続するためのコネクタハウジングやケーブル
- ポイント切替器

I^2C バスには EEPROM 24FC64 と I/O エクスパンダ MCP23017 の 2 種類のデバイスが接続されますが, それぞれの制御バイトの値が異なるために I^2C アドレスが重複することはありません.

表 28.3: 【部品表】 プログラマブル式ポイント一括切替器/メインボード

記号	数量	部品名
U2	1	PIC 16F18857-I/SP [I-11888]
U1	1	I^2C EEPROM 24FC64-I/P [I-03567]
Q1, Q2	2	FET 2N7000 [I-09723]
7SEG1, 7SEG2	2	7 セグメント LED (カソードコモン) [I-04115]
D1	1	LED 緑 [I-06413]
D2	1	LED 赤 [I-04780]
C2	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C1, C3	2	セラミックコンデンサ 0.1μF [P-00090]
R6, R7, R8, R9, R10, R12, R13, R14	8	抵抗 240Ω　1/4W [R-25241]
R5, R15	2	抵抗 2kΩ　1/4W [R-25202]
R11, R16	2	抵抗 2.7kΩ　1/4W [R-25272]
R1, R2, R3, R4, R19	5	抵抗 10kΩ　1/4W [R-25103]
R17, R18	2	抵抗 100kΩ　1/4W [R-25104]
	1	IC ソケット 28 ピン [P-01339]
	1	IC ソケット 8 ピン [P-00035]
SW1	1	ロータリエンコーダ　アルプス電気 EC12E2420801 [P-06357]
SW2, SW3	2	プッシュ SW [P-03651]
SW4	1	単極双投トグル SW　On-On 型 [P-02399]
J1	1	基板用コネクタ 4 ピン　JST XH コネクタ B4B-XH-A(LF) [C-12249]
J2	1	ピンソケット 6 ピン　ICSP 用 (無くても可) [C-03784]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 5V　出力 1A [M-06096]
		ほか基板・配線材等



(a) 内部回路 (概念図)

(b) 時計回り

(c) 反時計回り

図 28.8: インクリメンタル式ロータリエンコーダ

28.3　ロータリエンコーダ

ロータリエンコーダ (rotery encoder) とは回転軸の回転を検出する電子部品で, 以下の 2 種に大別されます.

- インクリメンタル方式
  回転軸の角度の増加・減少を出力します. すなわち出力は増加・減少ですので, 回転軸の現在の角度は分かりません.
- アブソリュート方式
  回転軸の角度を出力します. すなわち出力は回転軸の現在の角度です.

インクリメンタル方式は 2 つのスイッチで構成できることから簡単な機構で済みます. アブソリュート方式は回転軸の現在の角度を検出する機構が必要なので機構は複雑です. 以降では本書で使用するインクリメンタル方式のロータリエンコーダについて説明します.

インクリメンタル式ロータリエンコーダの内部回路の概念図を図 28.8a に示します. この図のように 2 つのスイッチの片方が出力 A と B であり, 他方が共通線 C で接続されています. ロータリエンコーダの回転軸の回転に従って内部の 2 つのスイッチがオン・オフします. 2 つのスイッチのオン・オフの関係は回転方向が時計回りか反時計回りかで異なり, 図 28.8b に時計回りの場合を, 図 28.8c に反時計回りの場合を, それぞれ示します. スイッチ出力の観察から回転方向の検出方法の論理は, 以下のように何通りかが挙げられます.

- A-C 間がオフからオンに変化したとき
  B-C 間がオンならば反時計回り. B-C 間がオフならば時計回り.
- A-C 間がオンからオフに変化したとき
  B-C 間がオンならば時計回り. B-C 間がオフならば反時計回り.
- B-C 間がオフからオンに変化したとき
  A-C 間がオンならば時計回り. A-C 間がオフならば反時計回り.
- B-C 間がオンからオフに変化したとき
  A-C 間がオンならば反時計回り. A-C 間がオフならば時計回り.

回転方向の検出には上記を実現するソフトウエア的な工夫が必要です. ロータリエンコーダの内部が機械的なスイッチで構成されている場合にはチャタリングが発生しますのでチャタリング対策も必須です. なおロータリエンコーダ部品によっては上記の論理のいずれもで確実に検出できるとは限りませんので, 現物を用いた試行錯誤が必要かもしれません.

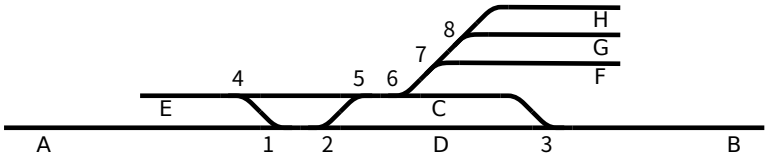


図 28.9: ある架空の駅の線路配線図 (数字: ポイント番号, ローマ字: 場所)

表 28.4: 図 28.9 に対する経路表の例

番号	経路	ポイント切替方向							
		1	2	3	4	5	6	7	8
1	A ↔ C	定位	反位	–	–	反位	定位	–	–
2	C ↔ B	–	–	反位	–	–	–	–	–
3	B ↔ D	–	–	定位	–	–	–	–	–
4	D ↔ A	定位	定位	–	–	–	–	–	–
5	C ↔ E	–	–	–	定位	定位	定位	–	–
6	D ↔ E	反位	定位	–	反位	–	–	–	–
7	E ↔ F	–	–	–	定位	定位	反位	反位	–
8	E ↔ G	–	–	–	定位	定位	反位	定位	反位
9	E ↔ H	–	–	–	定位	定位	反位	定位	定位

```
#-------------------------
# ROUTE DATA
#-------------------------
# THE NUMBER OF TOTAL ROUTES
= 9;
# ROUTE INFORMATION
* 1  +1 -2 -5 +6;
* 2  -3;
* 3  +3;
* 4  +1 +2;
* 5  +4 +5 +6;
* 6  -1 -2 -4;
* 7  +4 +5 -6 -7;
* 8  +4 +5 -6 +7 -8;
* 9  +4 +5 -6 +7 +8;
# END
/
#-------------------------
```

図 28.10: EEPROM に書き込む経路設定データの例

28.4　経路設定データの書式

経路設定データの作成方法と書式を説明します. 最初に図 28.9 に示す線路配線を例にして経路表の具体例を示します. なおこの図中では, 数字でポイント番号を, ローマ字で線路上の場所を, それぞれ表します. この線路配線は 1 面 2 線のホームを有する単線区間の駅を想定したもので, C 地点と D 地点にホームがあると考えて下さい. この駅において本線から駅に列車が到着したとき, 駅から列車が発車して本線に出るとき, あるいは, ヤードに停泊中の車両をホームに回送するときの一連のポイント切替を考えると, 経路として表 28.4 に示すものが考えられます. 表 28.4 の経路表に対する経路設定データの例を図 28.9 に示します. 経路設定データの書式の詳細は図 28.11 で説明しています.

経路設定データ書式

- テキストファイルで記述します. 改行は CR と CRLF のどちらでも構いません. 使用する文字は ASCII 文字 (いわゆる半角文字) ですが, コメント部分にのみ例外として日本語 (いわゆる全角文字) も使用できます.
- 1 行に 1 つのコマンドを記述します.
- 行の 1 文字目はコマンドを表し, #, =, *, / の 4 種の文字のいずれかでなければなりません. もしこれに違反している場合はコマンドエラーとなります. スペースやタブでもコマンドエラーとなります.
- コマンドは=, *, / の順序で出現しなくてはなりません. もしこれに違反している場合は順序エラーとなります. #の出現場所には制約はありません.
- コメントコマンド #

  構文: #〈 コメント 〉

  行末までコメントを書き入れることができます.
- 経路総数コマンド =

  構文: =〈 空白 〉〈 数 〉;

  この経路設定データ中に含まれる経路の総数を記述します.
  - 〈 空白 〉はスペースあるいはタブで, 1 文字以上であれば何文字あっても構いません.
  - 〈 数 〉は 1 以上 99 以下の整数値です.
  - ;は行の終端を表します. もしこの文字がない場合は行終端エラーとなります.

  このコマンドは 1 回だけしか使用できません. もしこれに違反している場合は書式エラーとなります.
- 経路仕様コマンド *

  構文: *〈 空白 〉〈 経路番号 〉〈 空白 〉〈 ポイント記述子 〉[〈 空白 〉〈 ポイント記述子 〉...];

  各経路におけるポイントの切替方向と番号を記述します.
  - 〈 空白 〉はスペースあるいはタブで, 1 文字以上であれば何文字あっても構いません.
  - 〈 経路番号 〉は整数値で記述します. 経路設定データ中では 1, 2, 3, ... の順番で〈 経路番号 〉は出現しなくてはなりません. もし順番に合致していない場合は順序エラーとなります. また経路総数コマンドで指定した数の経路仕様コマンドの出現が必要です. もしこれに違反している場合は欠落エラーとなります.
  - 〈 ポイント記述子 〉は切り替えを行うポイントの切替方向と番号を表します. 切替方向は + で定位, -で反位をそれぞれ表し, ポイントの番号との間には空白文字は入れずに続けて書きます. 〈 ポイント記述子 〉は〈 空白 〉を挟むことで複数の記述ができます. ポイント番号の範囲は 1 以上 64 以下です. もしこれに違反している場合はポイント番号範囲外エラーとなります.
  - ;は行の終端を表します. もしこの文字がない場合は行終端エラーとなります.
- 終了コマンド /

  構文: /

  経路設定データの終端を表します. もし終了コマンドがない場合は行終端エラーとなります.

図 28.11: 経路設定データの書式

経路設定データはこのようにプレインテキストのファイルの形ですので, パーソナルコンピュータ上で普通に利用可能なテキストエディタを使って経路設定データの作成ができます. 使用できる文字は ASCII 文字だけで, いわゆる全角文字は一切使えませんから十分に注意して下さい. なお#以降のコメント部分には全角文字は使用できますので, 内容が分かるコメントを書くと良いでしょう.

28.5　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します.

表 28.5: 【ソースファイル一覧】 EEPROM プログラム型ポイント一括切替器

初	頁	番号	ファイル名	内容	行数
✓	521	28.4	RouteCtr.c	主ソースコード	150
✓	519	28.3	defs.h	諸定義: ヘッダ	43
✓	523	28.5	fatal.h	エラー表示: ヘッダ	13
✓	523	28.6	fatal.c	エラー表示: 実装	41
✓	524	28.7	rotenc.h	ロータリエンコーダ読み取り: ヘッダ	7
✓	525	28.8	rotenc.c	ロータリエンコーダ読み取り: 実装	53
✓	528	28.11	route.h	経路一括制御: ヘッダ	17
✓	529	28.12	route.c	経路一括制御: 実装	148
✓	532	28.13	action.h	切替操作実行: ヘッダ	14
✓	533	28.14	action.c	切替操作実行: 実装	123
	282	17.8	led7seg.h	7 セグメント LED 点灯: ヘッダ	11
	283	17.9	led7seg.c	7 セグメント LED 点灯: 実装	321
✓	525	28.9	mem.h	EEPROM メモリアクセス: ヘッダ	24
✓	526	28.10	mem.c	EEPROM メモリアクセス: 実装	133
	394	23.3	i2c.h	I^2C インターフェース関数: ヘッダ	25
	394	23.4	i2c.c	I^2C インターフェース関数: 実装	111
	456	25.17	i2c-24fc64.h	24FC64 アクセス関数: ヘッダ	20
	457	25.18	i2c-24fc64.c	24FC64 アクセス関数: 実装	133
	396	23.5	i2c-mcp23017.h	MCP23017 アクセス関数: ヘッダ	53
	397	23.6	i2c-mcp23017.c	MCP23017 アクセス関数: 実装	109
✓	518	28.2	io.h	I/O ピン割付: ヘッダ	32
✓	534	28.15	mcr.h	MCR・周辺モジュール: ヘッダ	8
✓	535	28.16	mcr.c	MCR・周辺モジュール: 実装	92

計 1681 行 (うち初出 898 行, 既出 783 行)

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

ソースファイル番号 28.2: io.h

```
1 #if !defined(__IO_H__)
2 #define __IO_H__
3
```

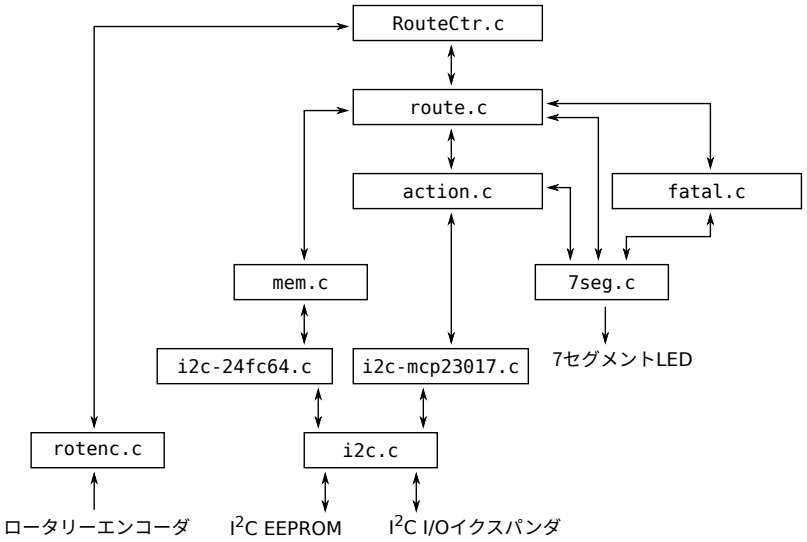


図 28.12: ファームウエアにおける主要ソースファイル間の関係

```
4  /* 経路表示SW DISP: RA0 */
5  #define SW_ROUTE_DISP()          (RA0 == 0)
6  /* 転換動作SW EXEC: RA1 */
7  #define SW_ROUTE_EXEC()          (RA1 == 0)
8  /* ロータリーエンコーダA: RA2 */
9  #define SW_ROTENC_A()            RA2
10 /* ロータリーエンコーダB: RA3 */
11 #define SW_ROTENC_B()            RA3
12 /* 転換動作中表示LED: RB5 */
13 #define LED_SWITCHING_ON()       (LATB5 = 1)
14 #define LED_SWITCHING_OFF()      (LATB5 = 0)
15 #define LED_SWITCHING_TOGGLE()   (LATB5 = ~LATB5)
16 /* I2C SCLピン: RC3 */
17 /* I2C SDAピン: RC4 */
18 /* 7-SEG LED ダイナミック点灯用の桁選択: RB1, RB0 */
19 #define LED7SEG_SEL_D1_PORT      LATB1 /* 左桁 */
20 #define LED7SEG_SEL_D2_PORT      LATB0 /* 中桁 */
21 #undef  LED7SEG_SEL_D3_PORT           /* 右桁(未実装) */
22 /* 7-SEG LED セグメント */
23 #define LED7SEG_A_PORT           LATA7
24 #define LED7SEG_B_PORT           LATA6
25 #define LED7SEG_C_PORT           LATC0
26 #define LED7SEG_D_PORT           LATC1
27 #define LED7SEG_E_PORT           LATC2
28 #define LED7SEG_F_PORT           LATC5
29 #define LED7SEG_G_PORT           LATC6
30 #define LED7SEG_DP_PORT          LATC7
31
32 #endif /*__IO_H__*/
```

ソースファイル defs.h (諸定義: ヘッダ)

各種定数, エラーコード, 動作種別やポイントの切替方向の記号定数などを定義します.

ソースファイル番号 28.3: defs.h

```
1  #if !defined(__DEFS_H__)
2  #define __DEFS_H__
3
4  /* 最大8つのMCP23017を接続可能 */
5  #define MAX_IO_MODULE             8
```

```
6
7   /* 1つのMCP23017あたりの接続ポイント数 */
8   #define N_TURNOUTS_PER_IO_MODULE  8
9
10  /* 接続可能なポイントの総数 */
11  #define MAX_N_TURNOUTS   (MAX_IO_MODULE * N_TURNOUTS_PER_IO_MODULE)
12
13  /* 7セグメントLED表示用エラーコード */
14  #define ERR_NO_MEMORY   "E1" /* EEPROM接続無し */
15  #define ERR_NO_MCP20317 "E2" /* MCP23017接続無し */
16  #define ERR_R_ILLCMD    "E3" /* 経路データ:コマンドエラー */
17  #define ERR_R_SYNTAX    "E4" /* 経路データ:書式エラー */
18  #define ERR_R_ORDERING  "E5" /* 経路データ:順序エラー */
19  #define ERR_R_MISSING   "E6" /* 経路データ:欠落エラー */
20  #define ERR_R_NONTERM   "E7" /* 経路データ:行終端エラー */
21  #define ERR_R_TURNOUT   "E8" /* 経路データ:ポイント番号範囲外 */
22
23  /* 動作種別 */
24  enum action_e {
25    ACTION_VALIDATE = 0,
26    ACTION_DISPLAY,
27    ACTION_OPERATION,
28  };
29
30  /* 開通方向 */
31  enum turnout_dir_e {
32    TURNOUT_DIR_NORMAL = 0, /* 定位 */
33    TURNOUT_DIR_REVERSE     /* 反位 */
34  };
35
36  extern uint8_t         io_module_table[];
37  extern uint16_t        n_routes;
38
39  extern void display_number(uint16_t n);
40  extern void display_turnout_and_dir(uint16_t p,
41                                      enum turnout_dir_e tdir);
42
43  #endif /*__DEFS_H__*/
```

ソースファイル RouteCtr.c (主ソースコード)

動作の主となる部分です. 初期化の際には以下の動作を行います.

1. 7 セグメント LED による表示の初期化を行います.
2. EEPROM 24FC64 の接続を確認してメモリアクセス関数の初期化を行います. 接続がなければエラーとして実行を中断します.
3. EEPROM 中の経路仕様の書式が正しいか検査します.
4. ポイントインターフェースボードの接続数を数えます. これは I^2C バスへの MCP23017 の接続の有無の検査で行います. 接続がなければエラーとして実行を中断します.
5. ポイント切替制御部を初期化します.
6. 初期選択の経路番号を 0 とし, 7 セグメント LED により経路番号を表示します.

初期化が終われば, 以下の動作を繰り返します.

1. ロータリエンコーダの回転を検出して経路番号の選択をします.
2. 押しボタンスイッチ DISP の押下があれば, 現在選択している経路番号に対する登録内容を 7 セグメント LED に表示します.
3. 押しボタンスイッチ EXEC の押下があれば, 現在選択している経路番号に対する登録内容に従ってポイ

ントを切り替えます. ただし経路番号が 0 の場合は特別で, 全ポイントを定位に切り替えます.

1 回の繰返しは 200 マイクロ秒間隔としている点に注意して下さい. ロータリエンコーダの出力の変化に追従するためには短い時間間隔でロータリエンコーダの出力を読み取らないといけないからです.

ソースファイル番号 28.4: RouteCtr.c

```
/*************************************************************
 * プログラマブル式ポイント一括切替器 / PIC16F18857
 * by Hirotsugu Kakugawa
 *************************************************************/
#include <xc.h>
#include <stdint.h>
#include <stdio.h>
#include "mcr.h"
#include "io.h"
#include "defs.h"
#include "route.h"
#include "action.h"
#include "led7seg.h"
#include "rotenc.h"
#include "fatal.h"
#include "i2c.h"
#include "i2c-mcp23017.h"
#include "mem.h"

/*-----------------------------------------------------------
 * MAIN
 *-----------------------------------------------------------*/
void main(void)
{
  /* システム初期化 */
  System_Init();
  /* 周辺モジュール初期化 */
  I2C_Init(I2C_BAUD_RATE_SSPxADD_1MHZ(_XTAL_FREQ));
  /* ウオッチドッグタイマ初期化: 2 [Sec] */
  WDTCON0bits.WDTPS = 0x0B; /* 0b01011 */
  CLRWDT();
  /* LED表示初期化 */
  led7seg_init();
  /* 割込許可 */
  GIE = 1;
  PEIE = 1;
  /* I2C EEPROMメモリの接続を確認 */
  __delay_ms(100);
  CLRWDT();
  if (!mem_probe_device()) {
    fatal(ERR_NO_MEMORY); /* EEPROM非存在 */
  }
  mem_init(MEM_TYPE_24FC64);
  /* EEPROM中の経路登録情報の検査 */
  route_read(ACTION_VALIDATE, ROUTE_ID_NULL);
  /* ポイントインターフェースボードを探索 */
  if (action_probe_device() == 0) {
    fatal(ERR_NO_MCP20317); /* ポイントインターフェースボード非存在 */
  }
  /* 切替制御部初期化 */
  action_init();
  /* 経路番号初期化 */
  uint16_t route_id = 0;
  display_number(route_id);
  /* 主ループ */
  for (;;) {
    CLRWDT();
    /* ロータリーエンコーダ入力(経路番号選択) */
    __delay_us(200);
    int8_t d = rotenc_scan();
    if (d != 0) {
      /* 入力あり */
```

```
63        if (d > 0) { /* 1増加 */
64          if (route_id >= n_routes) {
65            route_id = 0;
66          } else {
67            route_id++;
68          }
69        } else if (d < 0) { /* 1減少 */
70          if (route_id == 0) {
71            route_id = n_routes;
72          } else {
73            route_id--;
74          }
75        }
76        /* 表示更新 */
77        display_number(route_id);
78      }
79      /* 動作: 切替を行うポイント番号を表示 (切替は実行しない) */
80      if (SW_ROUTE_DISP()) {
81        if (route_id != 0) {
82          led7seg_display_const("  ");
83          CLRWDT();
84          __delay_ms(1000);
85          route_read(ACTION_DISPLAY, route_id);
86          led7seg_display_const("  ");
87          CLRWDT();
88          __delay_ms(1000);
89          display_number(route_id);
90        }
91        while (SW_ROUTE_DISP()) {
92          CLRWDT();
93        }
94      }
95      /* 動作: ポイント切替を実行 */
96      if (SW_ROUTE_EXEC()) {
97        CLRWDT();
98        led7seg_display_const("  ");
99        __delay_ms(500);
100       if (route_id != 0) {
101         /* 指定経路番号のポイント切替 */
102         route_read(ACTION_OPERATION, route_id);
103       } else {
104         /* 全ポイントを定位へ */
105         action_exec_all_normal();
106       }
107       CLRWDT();
108       led7seg_display_const("  ");
109       __delay_ms(500);
110       display_number(route_id);
111       while (SW_ROUTE_EXEC()) {
112         CLRWDT();
113       }
114     }
115   }
116 }
117
118 /* 7セグメントLEDに数値を表示 */
119 void display_number(uint16_t n)
120 {
121   char str[8];
122   sprintf(str, "%02d", n);
123   led7seg_display(str);
124 }
125
126 /* 7セグメントLEDに経路番号と分岐方向を表示 */
127 void display_turnout_and_dir(uint16_t p,
128                              enum turnout_dir_e tdir)
129 {
130   char strp[8];
131   char strx[8];
132   uint8_t i = 0;
133   uint8_t j = 0;
```

```
134     switch (tdir) {
135     default:
136     case TURNOUT_DIR_NORMAL:
137       /* 定位は小数点の消灯により表す*/
138       sprintf(strx, "%02d", p);
139       break;
140     case TURNOUT_DIR_REVERSE:
141       /* 反位は小数点の点灯により表す*/
142       sprintf(strp, "%02d", p);
143       for (i = 0; strp[i] != '\0'; i++) {
144         strx[((unsigned)i << 1) + 0] = strp[i];
145         strx[((unsigned)i << 1) + 1] = '.';
146       }
147       break;
148     }
149     led7seg_display(strx);
150   }
```

ソースファイル fatal.h (エラー表示: ヘッダ)

EEPROM 内の経路設定データに誤りがあるなど, 致命的エラーのために実行の継続が不可能な場合は実行を中止します. このソースファイルでは実行の中止をする関数のプロトタイプ宣言を行います.

ソースファイル番号 28.5: fatal.h

```
1   #if !defined(__FATAL_H__)
2   #define __FATAL_H__
3
4   extern void fatal(const char *p);
5   extern void fatal_s(const char *p, uint16_t lineno,
6           const char *s);
7   extern void fatal_sc(const char *p, uint16_t lineno,
8           const char *s, char ch);
9   extern void fatal_si(const char *p, uint16_t lineno,
10          const char *s, uint16_t i);
11
12  #endif /*__FATAL_H__*/
```

ソースファイル fatal.c (エラー表示: 実装)

致命的エラーによる実行中止をする関数の実装を行います. 具体的には, 表 28.2 に示すエラーコードを 7 セグメント LED に表示して無限ループに入ります. なお様々な引数の関数が宣言されていますが, 7 セグメント LED で実際に表示されるのはエラーコードだけでエラー要因の文字列情報などは表示されません.

ソースファイル番号 28.6: fatal.c

```
1   /*------------------------------------------------------
2    * エラーメッセージ
3    *------------------------------------------------------*/
4   #include <xc.h>
5   #include <stdint.h>
6   #include <stdio.h>
7   #include "mcr.h"
8   #include "io.h"
9   #include "fatal.h"
10  #include "led7seg.h"
11
12  void fatal(const char *p)
13  {
14    for (;;) {
15      CLRWDT();
16      LED_SWITCHING_ON();
17      led7seg_display_const(p);
18      __delay_ms(300);
19      LED_SWITCHING_OFF();
20      led7seg_display_const("  ");
```

```
21      __delay_ms(300);
22    }
23  }
24
25  void fatal_s(const char *p, uint16_t lineno,
26               const char *s)
27  {
28    fatal(p);
29  }
30
31  void fatal_sc(const char *p, uint16_t lineno,
32                const char *s, char ch)
33  {
34    fatal(p);
35  }
36
37  void fatal_si(const char *p, uint16_t lineno,
38                const char *s, uint16_t i)
39  {
40    fatal(p);
41  }
```

ソースファイル rotenc.h (ロータリエンコーダ読み取り: ヘッダ)

ロータリエンコーダの読み取りを行う関数のプロトタイプ宣言を行います.

ソースファイル番号 28.7: rotenc.h

```
1  #if !defined(__ROTENC_H__)
2  #define __ROTENC_H__
3
4  #define ROTENC_STAB_TIME  4
5  extern int8_t  rotenc_scan(void);
6
7  #endif /*__ROTENC_H__*/
```

ソースファイル rotenc.c (ロータリエンコーダ読み取り: 実装)

ロータリエンコーダの読み取りを行う関数 rotenc_scan() の実装を行います. 先述の通りこの関数は 200 マイクロ秒間隔で呼び出され, 以下の 2 つの処理を行います.

- チャタリングの処理
  ロータリエンコーダの前回の出力と今回の出力を比較し, 同じ出力が十分な回数継続すれば出力は安定したと判定します. この処理によりチャタリングに起因する短時間のパルスは除去されます.
- ロータリエンコーダ回転方向の検出
  チャタリング処理により得られた安定した出力をもとに, ロータリエンコーダの回転方向を検出します.

この関数が返す値は以下の通りです.

- +1: 正方向すなわち時計回りの回転を検出した場合
- −1: 逆方向すなわち反時計回りの回転を検出した場合
- 0: 回転を検出しなかった場合

回転方向検出の論理には, 実際に使用したロータリエンコーダで試行錯誤して以下の通りとしました. 別のロータリエンコーダ部品を使用する場合にはソースファイルの変更が必要かもしれません.

- B-C がオフからオンに変化したときに A-C がオン: 正方向
- A-C がオフからオンに変化したときに B-C がオン: 逆方向

ソースファイル番号 28.8: rotenc.c

```
/*---------------------------------------------------------
 * ロータリーエンコーダ
 *---------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "io.h"
#include "rotenc.h"

int8_t rotenc_scan(void)
{
  /* 回転方向: +1 (正) / 0 (無) / -1 (逆) */
  int8_t value = 0;
  /* 現在入力値 */
  int8_t curr_a = SW_ROTENC_A(); /* 0:ON, 1: OFF */
  int8_t curr_b = SW_ROTENC_B(); /* 0:ON, 1: OFF */
  /* チャタリング除去処理 */
  uint8_t stab_done = 0;
  static uint8_t chat_stab_t = 0;
  static int8_t  chat_prev_a = 0;
  static int8_t  chat_prev_b = 0;
  if ((chat_prev_a == curr_a) &&
      (chat_prev_b == curr_b)) {
    chat_stab_t++;
    if (chat_stab_t == ROTENC_STAB_TIME) { /* 収束 */
      stab_done = 1; /* チャタリング除去完了 */
      chat_stab_t = 0;
    }
  } else { /* 進行中 */
    chat_stab_t = 0; /* チャタリング除去未完了 */
    chat_prev_a = curr_a;
    chat_prev_b = curr_b;
  }
  /* 回転方向検出 */
  static int8_t prev_a = 0;
  static int8_t prev_b = 0;
  if ((stab_done == 1) &&
      ((prev_a != curr_a) || (prev_b != curr_b))) {
    /* チャタリング除去完了 かつ ロータリエンコーダ出力変化あり */
    if ((prev_b != 0) && (curr_b == 0) && /* B: OFF -> ON */
        (curr_a == 0)) { /* A: ON */
      /* 正方向の回転 */
      value = +1;
    }
    if ((prev_a != 0) && (curr_a == 0) && /* A: OFF -> ON */
        (curr_b == 0)) { /* B: ON */
      /* 逆方向の回転 */
      value = -1;
    }
    prev_a = curr_a;
    prev_b = curr_b;
  }
  return value;
}
```

ソースファイル mem.h (EEPROM メモリアクセス: ヘッダ)

EEPROM へのアクセスを行う関数群のプロトタイプ宣言を行います.

ソースファイル番号 28.9: mem.h

```
#if !defined(__MEM_H__)
#define __MEM_H__

#include "i2c-24fc64.h"

enum mem_type_e {
  MEM_TYPE_24FC64  = 0,
  MEM_TYPE_24FC256,
```

```
9     MEM_TYPE_24FC512,
10  };
11
12  extern int8_t   mem_probe_device(void);
13  extern void     mem_init(enum mem_type_e mt);
14  extern void     mem_addr_set(uint16_t addr);
15  extern uint16_t mem_addr_get(void);
16  extern int8_t   mem_is_end(void);
17  extern uint8_t  mem_read(void);
18  extern void     mem_unread(void);
19  extern uint8_t  mem_read_number(void);
20  extern int8_t   mem_is_space(char ch);
21  extern void     mem_skip_space(void);
22  extern void     mem_skip_to_eol(void);
23
24  #endif /*__MEM_H__*/
```

ソースファイル mem.c (EEPROM メモリアクセス: 実装)

EEPROM へのアクセスを行う関数群の実装を行います. このソースファイルの目的には以下の 2 通りが挙げられます.

- 経路設定データの書式の解析に便利な関数の提供
  数字の列を整数値に変換する関数, 空白文字の判別, 空白文字を読み飛ばす関数, 行末まで読み飛ばす関数などです.
- I^2C バスを占有しないメモリアクセス関数の提供
  ソースファイル i2c-24fc64.c により EEPROM へのアクセスを行う関数群が用意されていますが, それらを本ファームウエアで使用しやすくした関数群をこのソースファイルで実現します. 具体的には, EEPROM から複数のバイトを続けて読み出すときに, 1 バイトごとにひとつの I^2C トランザクションを実行して読み出します.
  ソースファイル i2c-24fc64.c では連続する複数バイトをひとつの I^2C トランザクションで連続して読み出す実装としています. この実装は最高速度で読み出しができて実時間要求の厳しい PCM サウンド再生では好都合なのですが, 本章のファームウエアでは以下の理由により不都合になります. その理由は本装置の I^2C バス上に 24FC64 と MCP23017 の 2 種類の I^2C デバイスが接続されていて, 24FC64 と MCP23017 の双方と交互に I^2C 通信を行うことに起因します. つまり本ファームウエアでは EEPROM から経路設定データを読みながら順次 MCP23017 に対してポイント切替を指示しています. しかし EEPROM に対する I^2C トランザクション実行中に MCP23017 に対する I^2C トランザクションを入れ子にはできません. この理由により, EEPROM の読み出しを 1 バイトにつき 1 つの I^2C トランザクションとすることで, どのタイミングにおいても MCP23017 に対する I^2C トランザクションを実行可能とします.

ソースファイル番号 28.10: mem.c

```
1   /*-------------------------------------------------------
2    * EEPROMメモリアクセス
3    * - 1バイトづつアドレスを設定して読み出す.
4    * i2c_24fc64.c は EEPROMのみがI2Cバスを連続的に使用することを前提とした
5    * 実装となっているため, EEPROM と MCP23018 へのアクセスが混在する場合では
6    * 本ファイルでの実装の通り, EEPROM読み出しの都度, アドレス設定を行う.
7    *-------------------------------------------------------*/
8   #include <xc.h>
9   #include <stdint.h>
10  #include <stdio.h>
11  #include <ctype.h>
```

```
#include "mcr.h"
#include "i2c.h"
#include "i2c-24fc64.h"
#include "mem.h"

static enum mem_type_e memtype = MEM_TYPE_24FC64;
static uint32_t        memsize = MEMSIZE_24FC64;
static uint32_t        memaddr = 0;

#define CHAR_LF     0x0A
#define CHAR_CR     0x0D

/* I2C EEPROMメモリの接続を確認 */
int8_t mem_probe_device(void)
{
  return I2C_Exist(I2C_24FC64_CTLBYTE, 0);
}

/* メモリデバイス設定 */
void mem_init(enum mem_type_e mt)
{
  memtype = mt;
  switch (mt) {
  case MEM_TYPE_24FC64:  memsize = MEMSIZE_24FC64;  break;
  case MEM_TYPE_24FC256: memsize = MEMSIZE_24FC256; break;
  case MEM_TYPE_24FC512: memsize = MEMSIZE_24FC512; break;
  default: break;
  }
}

/* 読み出しアドレス設定 */
void mem_addr_set(uint16_t addr)
{
  memaddr = addr;
}

/* 読み出しアドレス値 */
uint16_t mem_addr_get(void)
{
  return memaddr;
}

/* メモリ終端か否か */
int8_t mem_is_end(void)
{
  return (memaddr >= memsize);
}

/* 1バイト読み出し(次のアドレス) */
uint8_t mem_read(void)
{
  if (mem_is_end()) {
    return 0;
  }
  i2c_24fc64_read_begin(0, memaddr, 1);
  uint8_t ch = i2c_24fc64_read_byte();
  i2c_24fc64_read_done();
  memaddr++;
  return ch;
}

/* 読み出しアドレスを1つ戻す */
void mem_unread(void)
{
  if (memaddr != 0) {
    memaddr--;
  }
}

/* 整数値を読み取る */
uint8_t mem_read_number(void)
```

```
{
  uint8_t v = 0;
  while (!mem_is_end()) {
    char ch = mem_read();
    if (isdigit(ch)) {
      uint8_t v10 = (unsigned)((v << 3) + (v << 1)); /* = 10*v */
      uint8_t d = ch - '0';
      v = (unsigned) v10 + d;
    } else {
      mem_unread();
      break;
    }
  }
  return v;
}

/* 空白文字か否か */
int8_t mem_is_space(char ch)
{
  return (isspace(ch) && (ch != CHAR_CR) && (ch != CHAR_LF));
}

/* 空白文字を読み飛ばす(現在の行の範囲内で) */
void mem_skip_space(void)
{
  while (!mem_is_end()) {
    char ch = mem_read();
    if (!isspace(ch) || (ch == CHAR_CR) || (ch == CHAR_LF)) {
      mem_unread();
      break;
    }
  }
}

/* 行末まで読み飛ばす */
void mem_skip_to_eol(void)
{
  while (!mem_is_end()) {
    char ch = mem_read();
    if (ch == CHAR_CR) {
      if (mem_read() == CHAR_LF) {
        ;               /* CR LF */
      } else {
        mem_unread(); /* CR */
      }
      break;
    } else if (ch == CHAR_LF) {
      break;            /* LF */
    }
  }
}
```

ソースファイル route.h (経路一括制御: ヘッダ)

経路一括制御を行う関数群のプロトタイプ宣言を行います. また経路設定データでのコマンド文字のマクロ定義も行います.

ソースファイル番号 28.11: route.h

```
#if !defined(__ROUTE_H__)
#define __ROUTE_H__

#define ROUTE_ID_NULL  0

/* I2C EEPROM データでの指示子 */
#define CMD_CH_COMMENT    '#'
#define CMD_CH_END        '/'
#define CMD_CH_NROUTES    '='
#define CMD_CH_ROUTESPEC  '*'
```

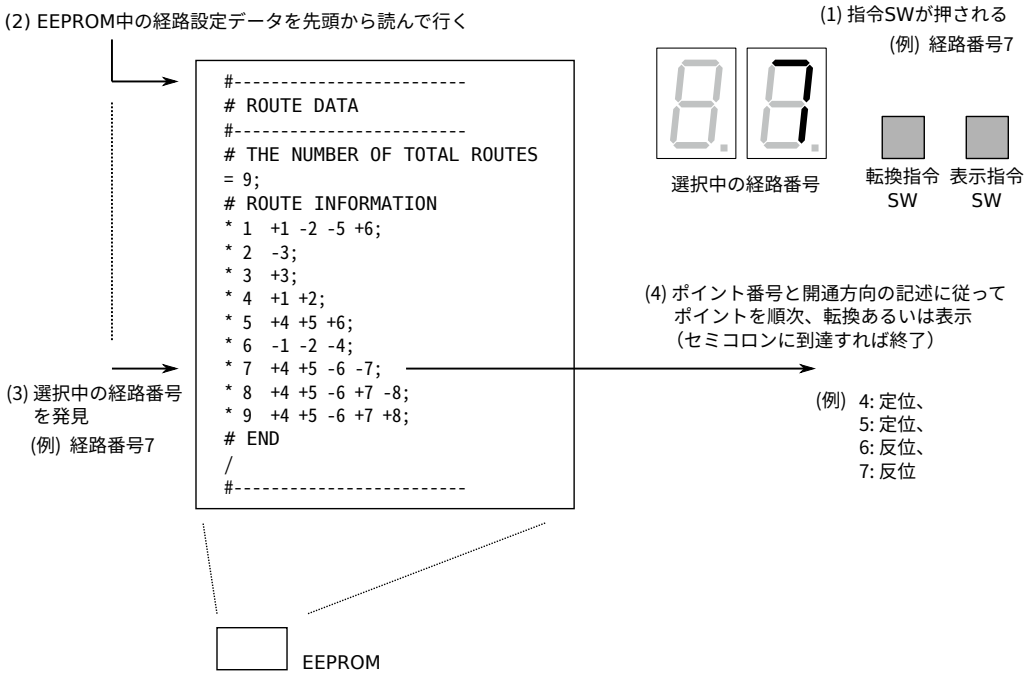


図 28.13: 動作の流れの概要

```
11 #define CMD_CH_TERMINATOR ';'
12
13 extern uint16_t n_routes;
14 extern void route_read(enum action_e act,
15                        uint8_t rid_action);
16
17 #endif /*__ROUTE_H__*/
```

ソースファイル route.c (経路一括制御: 実装)

経路一括制御を行う関数群の実装を行います. 具体的には, EEPROM 内の経路設定データを順次読み込み, 書式の検査と解析を行い, 指定された動作を実行します. 一連の動作の流れの概要を図 28.13 に示します.

このソースファイルでは read_route_spec() が主たる関数で, ファームウエア初期化の際に呼び出されて EEPROM 内の経路設定データの検査を行います. また, 押しボタンスイッチ DISP や EXEC が押されるときにもこの関数は呼び出され, その都度 EEPROM を先頭から順次読み出し, 対応する動作を行います. このときに EEPROM 内のデータ全体はおろか 1 行全体すら PIC マイコンのメモリには全て記憶はしていないことに注意して下さい. 先頭から順次読み出すことで処理が可能なように経路設定データの書式を注意深く設計してあり, PIC マイコンでの使用メモリ量はごく僅かです.

実際のポイント切替の制御は関数 read_route_spec() の中で行います. ロータリエンコーダで選択された経路番号が引数で与えられ, 合致する EEPROM 内の経路仕様の記述に対して関数 action_exec() を呼び出します. この関数 action_exec() はソースファイル action.c の中で実装されており, 詳しくは後述します.

ソースファイル番号 28.12: route.c

```
1 /*-------------------------------------------------------
2  * 経路設定
3  *-------------------------------------------------------*/
4 #include <xc.h>
5 #include <stdint.h>
```

```
6  #include <stdio.h>
7  #include <ctype.h>
8  #include "mcr.h"
9  #include "io.h"
10 #include "defs.h"
11 #include "route.h"
12 #include "action.h"
13 #include "fatal.h"
14 #include "mem.h"
15
16 /* 接続可能なポイントの最大数 */
17 uint16_t       max_turnouts = 0;
18 /* 経路総数 */
19 uint16_t       n_routes     = 0;
20
21 static void read_route_spec(enum action_e act,
22                             uint8_t rid_action,
23                             uint8_t rid_next,
24                             uint16_t lineno);
25
26 /* 経路情報の各行を読み順次処理を行う */
27 void route_read(enum action_e act,
28                 uint8_t rid_action)
29 {
30   n_routes = 0;
31   uint8_t occur_end     = 0;
32   uint8_t occur_nroutes = 0;
33   uint16_t rid_next = 0;
34   uint16_t lineno = 1;
35   mem_addr_set(0);
36   while (!mem_is_end()) {
37     CLRWDT();
38     uint8_t ch = mem_read(); /* 行頭の指示子文字 */
39     switch (ch) {
40     default:
41       fatal_sc(ERR_R_ILLCMD, lineno,
42                "ILLEGAL COMMAND", ch);
43       break;
44     case CMD_CH_END:     /* データ終了記号 */
45       occur_end = 1;
46       break;
47     case CMD_CH_COMMENT: /* コメント行 */
48       break;
49     case CMD_CH_NROUTES: /* 経路総数記述 */
50       if (occur_nroutes != 0) { /* 二重出現 */
51         fatal_s(ERR_R_SYNTAX, lineno,
52                 "COMMAND '=' APPEARS MULTIPLY");
53       }
54       occur_nroutes = 1;
55       /* 空白文字を読み飛ばす */
56       if (!mem_is_space(mem_read())) {
57         fatal_s(ERR_R_SYNTAX, lineno,
58                 "SPACE EXPECTED");
59       }
60       mem_skip_space();
61       /* 経路数を読む */
62       n_routes = mem_read_number();
63       /* 文終端チェック*/
64       if (mem_read() != CMD_CH_TERMINATOR) {
65         fatal_s(ERR_R_SYNTAX, lineno,
66                 "LINE TERMINATOR EXPECTED");
67       }
68       break;
69     case CMD_CH_ROUTESPEC: /* 経路仕様記述 */
70       rid_next++; /* 記述されているべき経路番号 */
71       /* 空白文字を読み飛ばす */
72       if (!mem_is_space(mem_read())) {
73         fatal_s(ERR_R_SYNTAX, lineno,
74                 "SPACE CHARACTER EXPECTED");
75       }
```

```
76        mem_skip_space();
77        /* 経路仕様記述を読む */
78        read_route_spec(act, rid_action, rid_next, lineno);
79        break;
80      }
81      /* データ終了チェック */
82      if (occur_end == 1) {
83        break;
84      }
85      /* 行末まで読み飛ばし */
86      mem_skip_to_eol();
87      lineno++;
88    }
89    /* データ統合性チェック */
90    if (n_routes != rid_next) { /* 記述されていない経路あり */
91      fatal_s(ERR_R_MISSING, lineno,
92              "NOT ENOUGH NUMBER OF ROUTES");
93    }
94    if (occur_end == 0) { /* データ終了記号が欠落 */
95      fatal_s(ERR_R_NONTERM, lineno,
96              "COMMAND '/' NOT FOUND");
97    }
98  }
99
100 /* 経路仕様記述を1行読む */
101 static void read_route_spec(enum action_e act,
102                             uint8_t rid_action,
103                             uint8_t rid_next,
104                             uint16_t lineno)
105 {
106   char ch = 0;
107   /* 経路番号（経路番号は1,2,3,...の順序で出現する事） */
108   uint8_t rid = mem_read_number();
109   if (rid != rid_next) { /* 経路番号順序違反 */
110     fatal_si(ERR_R_ORDERING, lineno,
111              "UNEXPECTED ROUTE ID", rid);
112   }
113   /* ポイント番号リスト */
114   while (!mem_is_end()) {
115     /* 空白文字を読み飛ばす */
116     ch = mem_read();
117     if (mem_is_space(ch)) {
118       mem_skip_space();
119     } else if (ch == CMD_CH_TERMINATOR) { /* リスト終了 */
120       mem_unread();
121       break;
122     }
123     /* 記述解析 */
124     ch = mem_read();
125     if ((ch == '+') ||
126         (ch == '-')) { /* 切替方向＆ポイント番号 */
127       /* 切替方向 */
128       enum turnout_dir_e tdir = TURNOUT_DIR_NORMAL;
129       switch (ch) {
130       case '+': tdir = TURNOUT_DIR_NORMAL;  break; /* 定位 */
131       case '-': tdir = TURNOUT_DIR_REVERSE; break; /* 反位 */
132       }
133       /* ポイント番号 p */
134       uint8_t p = mem_read_number();
135       if ((p == 0) || (p > MAX_N_TURNOUTS)) { /* 範囲外 */
136         fatal_si(ERR_R_TURNOUT, lineno,
137                  "UNEXPECTED TURNOUT NUMBER", p);
138       }
139       /* 現在行の経路番号が動作対象の経路番号なら実行 */
140       if (rid == rid_action) {
141         action_exec(act, p, tdir);
142       }
143     } else { /* 不正文字 */
144       fatal_sc(ERR_R_SYNTAX, lineno,
145                "ILLEGAL CHARACTER", ch);
```

```
146       }
147     }
148   }
```

ソースファイル action.h (切替操作実行: ヘッダ)

実際にポイント切替を行う関数群のプロトタイプ宣言を行います. 複数のポイントを順次切り替える際の時間間隔は記号定数 ACTION_OP_TIME で定義します.

ソースファイル番号 28.13: action.h

```
1  #if !defined(__ACTION_H__)
2  #define __ACTION_H__
3
4  /* 転換時間間隔 */
5  #define ACTION_OP_TIME   150  /* [mSec] */
6
7  extern uint8_t action_probe_device(void);
8  extern void    action_init(void);
9  extern void    action_exec_all_normal(void);
10 extern void    action_exec(enum action_e act,
11                            uint16_t p,
12                            enum turnout_dir_e tdir);
13
14 #endif /*__ACTION_H__*/
```

ソースファイル action.c (切替操作実行: 実装)

実際にポイント切替を行う関数群の実装を行います. 各関数の概要は以下の通りです.

- action_probe_device()
  I^2C バスへの MCP23017 の接続の有無を検出します. 8 つのアドレスに対して接続検出を行い, ポイントインターフェースボードの接続の有無は MCP23017 の検出の有無で判断します. この関数の返り値は接続されているポイントインターフェースボードの数です.
- action_init()
  切替状態の初期化を行います. 具体的には, 接続されている MCP23017 に対して GIPO を出力モードにし, 出力値を 0 に設定します.
- action_exec_all_normal()
  全ポイントを定位に切り替えます. ただしいったん反位に切り替えた後に定位に切り替えます. この動作により本書で紹介するどのポイント切替器でも確実に定位に切り替えます.
- action_exec()
  ポイント切替を行います. この関数の引数は動作種別, ポイント番号, 切替方向の 3 つです. ただし引数の動作種別によっては, 何もしない, あるいは切替内容を 7 セグメント LED に表示するだけです. 動作種別が切替実行の場合には, MCP23017 を操作してポイント切替器へのポイント制御出力を設定します. ポイント番号と各ポイントインターフェースボードの GPIO ポートとの対応を説明します. ポイント番号は 1, 2, 3, ... と番号付けされています. ひとつのポイントインターフェースボードには 8 つのポイント制御出力がありますから, ポイント番号 1 から 8 はポイントインターフェースボード番号 1 に対応し, ポイント番号 9 から 16 はポイントインターフェースボード番号 2 に対応し, 以下同様となります. 各ポイントインターフェースボードでは, 1 つ目のポイント制御出力に GPA0 と GPA1 を使用し, 2 つ目のポイント制御出力に GPA2 と GPA3 を使用し, 以下同様となります. 更に 4 つ目のポイント制御出力に GPB0 と GPB1 を使用し, 5 つ目のポイント制御出力に GPB2 と GPB3 を使用し, 以下同様となります.

ポイント切替は, ポイント番号に該当するポイントインターフェースボードの GPIO ポートを操作することでポイント制御出力を制御します. プログラムコードで行われている各種ビット演算はこの操作を行うものです.

ソースファイル番号 28.14: action.c

```
/*------------------------------------------------------
 * 切替制御部
 *------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include <stdio.h>
#include "mcr.h"
#include "io.h"
#include "defs.h"
#include "action.h"
#include "route.h"
#include "i2c.h"
#include "i2c-mcp23017.h"
#include "led7seg.h"

/* ポイントインターフェースボード(MCP23017)の接続の有無 */
uint8_t io_module_table[MAX_IO_MODULE];
/* ポイントインターフェースボード接続数 */
uint8_t io_nmodules = 0;

/* MCP23017のポートの出力値 */
static uint8_t gpa[MAX_IO_MODULE];
static uint8_t gpb[MAX_IO_MODULE];

/* ポイントインターフェースボードの接続を確認 */
uint8_t action_probe_device(void)
{
  io_nmodules = 0;
  /* 各アドレスに対して接続を確認 */
  uint8_t caddr = 0;
  for (caddr = 0; caddr < MAX_IO_MODULE; caddr++) {
    CLRWDT();
    io_module_table[caddr] = 0;
    if (I2C_Exist(I2C_MCP23017_CTLBYTE, caddr)) { /* 存在確認 */
      io_module_table[caddr] = 1;
      io_nmodules++;
    }
  }
  return io_nmodules;
}

/* 切替状態の初期化 */
void action_init(void)
{
  uint8_t caddr = 0;
  for (caddr = 0; caddr < MAX_IO_MODULE; caddr++) {
    CLRWDT();
    gpa[caddr] = 0x00;
    gpb[caddr] = 0x00;
    if (io_module_table[caddr] != 0) {
      /* MCP23017初期化 */
      i2c_mcp23017_init(caddr);
      i2c_mcp23017_a_write(caddr, gpa[caddr]);
      i2c_mcp23017_b_write(caddr, gpb[caddr]);
      i2c_mcp23017_a_dir_set(caddr, 0x00);
      i2c_mcp23017_b_dir_set(caddr, 0x00);
    }
  }
}

/* 全ポイントを定位へ切替 */
void action_exec_all_normal(void)
```

```
63  {
64    uint8_t p = 1;
65    for (p = 1; p <= MAX_N_TURNOUTS; p++) {
66      /* まず反位に切替し，次に定位に切替 */
67      action_exec(ACTION_OPERATION, p, TURNOUT_DIR_REVERSE);
68      action_exec(ACTION_OPERATION, p, TURNOUT_DIR_NORMAL);
69    }
70  }
71
72  /* 切替実行（実際の動作は引数 act で選択） */
73  void action_exec(enum action_e act,
74                   uint16_t p,
75                   enum turnout_dir_e tdir)
76  {
77    uint8_t g = (unsigned)p - 1;  /* p=1,2,3,... => g=0,1,2,... */
78    uint8_t caddr = ((unsigned)g >> 3) & 0x07;
79    uint8_t gpio  = (unsigned)g & 0x07;
80    const uint8_t bit_mask[4] =    { 0x03, 0x0C, 0x30, 0xC0 };
81    const uint8_t bit_normal[4] =  { 0x01, 0x04, 0x10, 0x40 };
82    const uint8_t bit_reverse[4] = { 0x02, 0x08, 0x20, 0x80 };
83    CLRWDT();
84    switch(act) {
85    default:
86    case ACTION_VALIDATE: /* 検証 */
87      break; /* 特に何もしない（常にデータチェックしているため） */
88    case ACTION_DISPLAY: /* 表示のみ */
89      /* MCP23017の接続の有無に関わらず表示 */
90      display_turnout_and_dir(p, tdir);
91      __delay_ms(1000);
92      break;
93    case ACTION_OPERATION: /* 切替実行 */
94      /* MCP23017の出力ビットパターンを求める */
95      if (gpio < 4) { /* GPIO A */
96        uint8_t i = gpio;
97        gpa[caddr] &= (unsigned)~bit_mask[i];
98        if (tdir == TURNOUT_DIR_NORMAL) {
99          gpa[caddr] |= bit_normal[i];
100       } else {
101         gpa[caddr] |= bit_reverse[i];
102       }
103     } else {        /* GPIO B */
104       uint8_t i = (unsigned)gpio - 4;
105       gpb[caddr] &= (unsigned)~bit_mask[i];
106       if (tdir == TURNOUT_DIR_NORMAL) {
107         gpb[caddr] |= bit_normal[i];
108       } else {
109         gpb[caddr] |= bit_reverse[i];
110       }
111     }
112     /* MCP23017に出力(該当MCP23017の接続がある場合のみ) */
113     if (io_module_table[caddr] != 0) {
114       LED_SWITCHING_ON();
115       display_turnout_and_dir(p, tdir);
116       i2c_mcp23017_a_write(caddr, gpa[caddr]);
117       i2c_mcp23017_b_write(caddr, gpb[caddr]);
118       __delay_ms(ACTION_OP_TIME);
119       LED_SWITCHING_OFF();
120     }
121     break;
122   }
123 }
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 28.15: mcr.h

```
1  #if !defined(__MCR_H__)
```

```
#define __MCR_H__

#define _XTAL_FREQ  32000000 /* 32MHz */

extern void System_Init(void);

#endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います. TMR1 による割り込みは 7 セグメント LED のダイナミック点灯で使用します.

ソースファイル番号 28.16: mcr.c

```
/*-------------------------------------------------------
 * MCR設定&周辺モジュール (PIC16F18857)
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "i2c.h"

/*-----------------------------------------------*/
/* MCR初期化 */
void System_Init(void)
{
  /* 発振器 (FOSC=32MHz) */
  OSCCON1 = 0x10; /* HFINTOSC with 2x PLL */
  OSCCON3 = 0x00;
  OSCEN   = 0x00;
  OSCFRQ  = 0x05; /* 16MHz */
  OSCTUNE = 0x00;
  while (PLLR == 0) {
    ;
  }
  /* I/Oピン */
  LATA   = 0x00;
  LATB   = 0x00;
  LATC   = 0x00;
  TRISA  = 0x3F; /* (RA5,RA4),RA3,RA2,RA1,RA0 */
  TRISB  = 0xDC; /* (RB7,RB6,RB4,RB3,RB2) */
  TRISC  = 0x18; /* RC3,RC4 */
  ANSELA = 0x00;
  ANSELB = 0x00;
  ANSELC = 0x00;
  WPUA   = 0x30; /* (RA5,RA4) */
  WPUB   = 0xDC; /* (RB7,RB6,RB4,RB3,RB2) */
  WPUC   = 0x00;
  ODCONA = 0x00;
  ODCONB = 0x00;
  ODCONC = 0x00;
  int8_t ogie = GIE;
  GIE = 0;
  PPSLOCK = 0x55;
  PPSLOCK = 0xAA;
  PPSLOCKbits.PPSLOCKED = 0;
  RC3PPS = 0x14;     /* RC3 <- SCL1 */
  RC4PPS = 0x15;     /* RC4 <- SDA1 */
  SSP1CLKPPS = 0x13; /* RC3 -> SCL1 */
  SSP1DATPPS = 0x14; /* RC4 -> SDA1 */
  PPSLOCK = 0x55;
  PPSLOCK = 0xAA;
  PPSLOCKbits.PPSLOCKED = 1;
  GIE = ogie;
}

/*-----------------------------------------------*/
/* 割り込みハンドラ */
```

```
55 __interrupt() void isr(void)
56 {
57   if ((TMR1IE == 1) && (TMR1IF == 1)) {
58     extern void TMR1_isr(void);
59     TMR1IF = 0;
60     TMR1_isr();
61   }
62 }
63
64 /*-----------------------------------------*/
65 /* CONFIG1 */
66 #pragma config FEXTOSC  = OFF
67 #pragma config RSTOSC   = HFINTPLL
68 #pragma config CLKOUTEN = OFF
69 #pragma config CSWEN    = ON
70 #pragma config FCMEN    = ON
71 /* CONFIG2 */
72 #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
73 #pragma config PWRTE    = OFF
74 #pragma config LPBOREN  = OFF
75 #pragma config BOREN    = ON
76 #pragma config BORV     = LO
77 #pragma config ZCD      = OFF
78 #pragma config PPS1WAY  = ON
79 #pragma config STVREN   = ON
80 #pragma config DEBUG    = OFF
81 /* CONFIG3 */
82 #pragma config WDTCPS   = WDTCPS_31
83 #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
84 #pragma config WDTCWS   = WDTCWS_7
85 #pragma config WDTCCS   = SC
86 /* CONFIG4 */
87 #pragma config WRT      = OFF
88 #pragma config SCANE    = available
89 #pragma config LVP      = OFF /* 低電圧書込不可 */
90 /* CONFIG5 */
91 #pragma config CP       = OFF
92 #pragma config CPD      = OFF
```

28.6　経路設定データ書式チェッカ chkroute

経路設定データの開発の際に便利なソフトウエア chkroute を紹介します. これは作製した経路設定データの書式をチェックするソフトウエアで, パーソナルコンピュータ上で実行します. 経路設定データ書式チェッカ chkroute を使用した経路一括切替器の利用のための作業手順を図 28.14 に示します.

もしこのソフトウエアがなければ, 経路設定データの書式上のミスを見つける方法は 2 通りで, 人間が目視でチェックする, あるいはポイント一括切替器に装着してファームウエアが書式上のミスを発見してエラー表示をする, に限られています. つまり経路設定データの完成までには図 28.4 に示されているように, 以下の手順を踏むことになります.

1. パーソナルコンピュータ上で経路設定データを作製する.
2. 経路設定データを EEPROM に書き込む.
3. ポイント一括切替器に装着して実際に動作させる.
4. もし間違いがあれば修正して, 手順 2 に戻る.

しかしこれでは間違いがある度に EEPROM の抜き差し・書き込み作業を繰り返さなければならず, 実に不便です.

そこで chkroute を使用すれば, 経路設定データの完成までには図 28.14 に示されているように, 以下の手順を踏むことになります.

1. パーソナルコンピュータ上で経路設定データを作製する.
2. パーソナルコンピュータ上で chkroute を動作させる.
3. もし間違いがあれば修正して, 手順 2 に戻る.
4. 経路設定データを EEPROM に書き込む.
5. ポイント一括切替器に装着して実際に動作させる.

chkroute を使用するぶん作業手順が増えたようにも見えます. 書式チェックをパーソナルコンピュータ上で行えますから, 何度もの EEPROM の抜き差し・書き込み作業から開放されて実際の開発効率は劇的に向上します.

28.6.1　使い方

chkroute を使用した経路設定データのチェックはパーソナルコンピュータ上で実行します. コマンドライン引数にはチェック対象の経路設定データのファイル名を指定します.

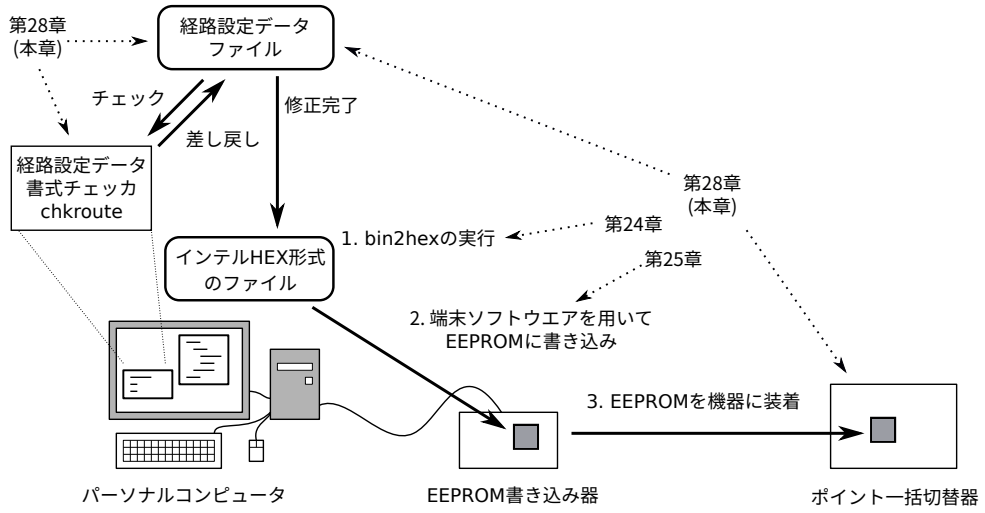


図 28.14: 経路設定データ書式チェッカ chkroute を使用した経路一括切替器の利用のための作業手順

```
1  = 4;
2  * 1   +1 +2 +3 +4;
3  * 2   -1 -2 -3 -4;
4  * 3   +1 -2;
5  * 4   -1 +2;
6  /
```

(a) test-route-ok.txt

```
1  = 4;
2  + 1   +1 +2 +3 +4;
3  * 2   -1 -2 -3 -4;
4  * 3   +1 -2;
5  * 4   -1 +2;
6  /
```

(b) test-route-e3.txt

```
1  = 4;
2  * 1   +1 +2 +3 +4
3  * 2   -1 -2 -3 -4
4  * 3   +1 -2
5  * 4   -1 +2
6  /
```

(c) test-route-e4.txt

```
1  = 4;
2  * 1   +1 +2 +3 +4;
3  * 2   -1 -2 -3 -4;
4  * 4   -1 +2;
5  * 3   +1 -2;
6  /
```

(d) test-route-e5.txt

```
1  = 4;
2  * 1   +1 +2 +3 +4;
3  * 2   -1 -2 -3 -4;
4  * 3   -1 +2;
5  /
```

(e) test-route-e6.txt

```
1  = 4;
2  * 1   +1 +2 +3 +4;
3  * 2   -1 -2 -3 -4;
4  * 3   +1 -2;
5  * 4   -1 +2 +91;
6  /
```

(f) test-route-e8.txt

図 28.15: 誤りなしと誤りありの経路データの例

誤りのないものや誤りのある経路設定データを各種用意して図 28.15 に示します. これらに対してチェックを行う例を示します.

まず最初に, 書式エラーのない経路設定データ (図 28.15a) の場合には以下のように “VERIFIED” と表示されます. この表示が出れば書式にエラーはありませんから, インテル HEX 形式に変換し, EEPROM に書き込む作業に進みます.

```
> ./chkroute test-route-ok.txt
VERIFIED
```

次に, 書式エラーがある経路設定データの場合には以下のようになんらかのエラー表示があります. なお斜体字の部分が入力部分です.

```
> ./chkroute test-route-e3.txt
FATAL ERROR (E3) AT LINE 2. ILLEGAL COMMAND: '+'
> ./chkroute test-route-e4.txt
FATAL ERROR (E4) AT LINE 2. ILLEGAL CHARACTER: '*'
> ./chkroute test-route-e5.txt
FATAL ERROR (E5) AT LINE 4. UNEXPECTED ROUTE ID: 4
> ./chkroute test-route-e6.txt
FATAL ERROR (E6) AT LINE 5. NOT ENOUGH NUMBER OF ROUTES
> ./chkroute test-route-e8.txt
FATAL ERROR (E8) AT LINE 5. UNEXPECTED TURNOUT NUMBER: 91
```

それぞれのエラーメッセージを簡単に説明しますので, 修正作業の際の参考にして下さい.

- 1 つ目: ファイル test-route-e3.txt (図 28.15b)
  2 行目にエラーがあります (At LINE 2). * とすべきところを + と誤って書いています.
- 2 つ目: ファイル test-route-e4.txt (図 28.15c)
  2 行目にエラーがあります (At LINE 2). 行末の ; を書き忘れています.
- 3 つ目: ファイル test-route-e5.txt (図 28.15d)
  4 行目にエラーがあります (At LINE 4). 経路番号 3 を書くべきなのに誤って経路 4 を書いています.
- 4 つ目: ファイル test-route-e6.txt (図 28.15e)
  5 行目にエラーがあります (At LINE 5). 経路総数は 4 なのに, 全ての経路を記述しないままデータが終わっています.
- 5 つ目: ファイル test-route-e8.txt (図 28.15f)
  5 行目にエラーがあります (At LINE 5). 使用可能なポイント番号の範囲を超えたポイント番号を書いています.

経路設定データの誤りをひとつづつ修正してゆき, エラーがなくなるまで繰り返します. 上記の作業はすべてパーソナルコンピュータ上で行えるので, 修正作業は実に簡単に完了します.

28.6.2　ソースファイル

経路設定データ書式チェッカ chkroute を構成するソースファイル一覧を以下に示します.

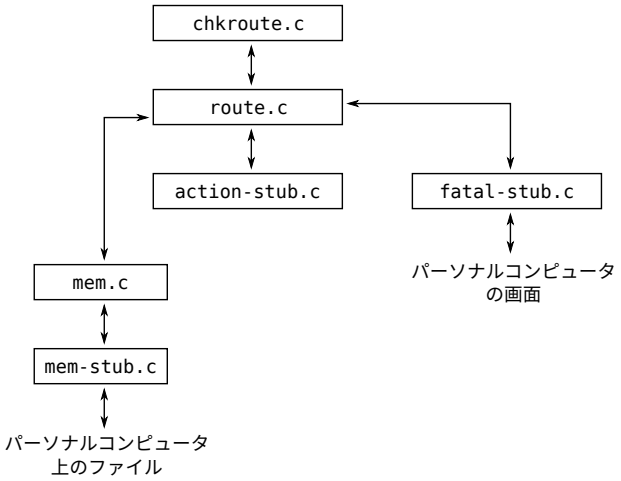


図 28.16: 経路設定データ書式チェッカ chkroute における主要ソースファイル間の関係

表 28.6: 【ソースファイル一覧】 経路設定データ書式チェッカ chkroute

初	頁	番号	ファイル名	内容	行数
✓	540	28.23	`chkroute.c`	主ソースコード	47
	519	28.3	`defs.h`	諸定義: ヘッダ	43
	528	28.11	`route.h`	経路一括制御: ヘッダ	17
	529	28.12	`route.c`	経路一括制御: 実装	148
	532	28.13	`action.h`	切替操作実行: ヘッダ	14
	523	28.5	`fatal.h`	エラー表示: ヘッダ	13
✓	541	28.25	`fatal-stub.c`	エラー表示 (代替): 実装	38
	525	28.9	`mem.h`	EEPROM メモリアクセス: ヘッダ	24
	526	28.10	`mem.c`	EEPROM メモリアクセス: 実装	133
✓	542	28.26	`mem-stub.c`	EEPROM エミュレーション: 実装	41
	394	23.3	`i2c.h`	I^2C インターフェース関数: ヘッダ	25
	456	25.17	`i2c-24fc64.h`	24FC64 アクセス関数: ヘッダ	20
✓	543	28.27	`xc.h`	XC8 代替ヘッダ	2
	518	28.2	`io.h`	I/O ピン割付: ヘッダ	32
	534	28.15	`mcr.h`	MCR・周辺モジュール: ヘッダ	8

計 605 行 (うち初出 128 行, 既出 477 行)

ほとんどのソースファイルはファームウエアのものを流用しています. 特に経路設定データを読み込む部分 (`route.c`) を流用していることが重要です. つまり経路設定データのチェックを行うプログラム部分が, パーソナルコンピュータ上で動作する chkroute と組み込み機器で動作するファームウエアとで同じです. 従って chkroute でエラーがないことを確認しておけばファームウエアの実行時にエラーが発生することは原理上起き得ません.

パーソナルコンピュータ上で経路設定データ書式のチェックを行う仕組みを概説します. ファームウエアはハードウエアと連携した動作をしていますが, パーソナルコンピュータ上ではハードウエアのふりをした動作

をさせる部分を用意することで, ファームウエアのソースコードをそのままコンパイル・動作できるようにします. 図 28.16 に chkroute における主要ソースファイル間の関係を示します. ファームウエアにおける主要ソースファイル間の関係を示した図 28.12 と比較してみて下さい.

- EEPROM
  ファームウエアでは経路設定データを EEPROM を読み出して動作しますが, パーソナルコンピュータには EEPROM はありません. そこで chkroute では, 経路設定データをファイルから読み出すことで EEPROM から読み出したことにします. ソースファイル `mem-stub.c` がこの役割を行います.
- I^2C アクセス
  ハードウエアとしての EEPROM へのアクセスはなくなりますから, I^2C へのアクセスは不要になります. そこで chkroute では, ファームウエアでのポイント切り替えの指令に対して何もせず, ポイントを切り替えたことにします. ソースファイル `action-stub.c` がこの役割を行います.
- 7 セグメント LED による表示
  ファームウエアではエラー番号の表示を行う重要な役割を担っています. 経路設定データのチェックの際にはエラー表示を充実させることが重要です. そこで chkroute では, ファームウエアでの 2 桁の 7 セグメント LED へ表示する代わりに, エラーの原因などできるだけ様々な情報を画面に表示するようにします. ソースファイル `fatal-stub.c` がこの役割を行います.
- 押しボタンスイッチ
  経路切り替え実行や経路表示のための押ボタンスイッチ DISP および EXEC は chkroute で利用できません. chkroute は経路設定データをチェックするだけですから本来これらは不要です. ファームウエアで用意してある経路設定データを読み込む関数に書式チェックが含まれていることから, chkroute はこの関数を呼び出して終了するだけで目的が達成できます.

ソースファイル chkroute.c (主ソースコード)

コマンドライン引数に与えられた経路設定データが書かれているファイルを EEPROM の代わりに読み出すようにします. そして経路設定データを読み込む関数を呼び出して書式チェックを行います.

ソースファイル番号 28.23: chkroute.c

```
1  /*
2   * 経路設定データ書式チェッカ chkroute
3   */
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <stdint.h>
7  #include "mem.h"
8  #include "defs.h"
9  #include "route.h"
10
11 static void usage(void);
12 extern void mem_emu_init(char *f);
13
14 int main(int argc, char **argv)
15 {
16   char *script = NULL;
17   int i;
18   /* Command line args */
19   for (argc--, argv++; argc > 0; argc--, argv++) {
20     if (argv[0][0] != '-') {
21       script = argv[0];
22     } else {
23       usage();
24     }
```

```
  }
  if (script == NULL) {
    usage();
  }
  /* EEPROM EMULATION */
  mem_emu_init(script);
  /* Check route data */
  route_read(ACTION_VALIDATE, ROUTE_ID_NULL);
  /* Done */
  printf("VERIFIED\n");
  return 0;
}

static void usage(void)
{
  printf("chkroute - Check ROUTECTR EEPROM data\n");
  printf("Usage: chkroute FILE\n");
  printf("Argument: \n");
  printf("  FILE      EEPROM Route data file (plain text)\n");
  printf("Examples:\n");
  printf("  chkroute RouteROM-1.txt\n");
  exit(0);
}
```

ソースファイル action-stub.c (切替操作実行 (代替): 実装)

ポイントの切り替えをする関数を用意しますが, 実際には何もせずに切り替えたことにするだけです. わざわざ用意する必要はなさそうなのですが, 流用しているファームウエアのソースファイルが関数 action_exec() を呼び出しているために, 中身が空の関数を用意する必要があります.

ソースファイル番号 28.24: action-stub.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "defs.h"

void action_exec(enum action_e act,
                 uint16_t p,
                 enum turnout_dir_e tdir)
{
  ;
}
```

ソースファイル fatal-stub.c (エラー表示 (代替): 実装)

エラーメッセージを画面上に表示します. ファームウエアでは不要な引数を用意していたのは, 実は chkroute を作りやすくするための事前の準備なのでした.

ソースファイル番号 28.25: fatal-stub.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

void fatal(const char *p)
{
  printf("FATAL ERROR (%s)\n", p);
  exit(0);
}

void fatal_s(const char *p, uint16_t lineno,
             const char *s)
{
  printf("FATAL ERROR (%s) AT LINE %d. %s\n",
         p, lineno, s);
```

```
16    exit(0);
17  }
18
19  void fatal_sc(const char *p, uint16_t lineno,
20                const char *s, char ch)
21  {
22    if ((0x20 <= ch) && (ch <= 0x7E)) {
23      printf("FATAL ERROR (%s) AT LINE %d. %s: '%c'\n",
24        p, lineno, s, ch);
25    } else {
26      printf("FATAL ERROR (%s) AT LINE %d. %s: CHARCODE=0x%02X\n",
27        p, lineno, s, ch);
28    }
29    exit(0);
30  }
31
32  void fatal_si(const char *p, uint16_t lineno,
33                const char *s, uint16_t i)
34  {
35    printf("FATAL ERROR (%s) AT LINE %d. %s: %d\n",
36           p, lineno, s, i);
37    exit(0);
38  }
```

ソースファイル mem-stub.c (EEPROM エミュレーション: 実装)

EEPROM へのアクセスの代わりにパーソナルコンピュータのファイルへのアクセスを行う機能を提供します. 526 ページで紹介したソースファイル mem.c による EEPROM アクセス関数に対する代替を行います.

ソースファイル番号 28.26: mem-stub.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <stdint.h>
4
5   static FILE *mem_fp = NULL;
6
7   #if defined(_MSC_VER)
8   #define FOPEN(p,f,m)    fopen_s(p, f, m)
9   #else
10  #define FOPEN(p,f,m)    (((*(p)=fopen(f,m))!=NULL)?0:1)
11  #endif
12
13  void mem_emu_init(char *f)
14  {
15    if (FOPEN(&mem_fp, f, "rb") != 0) {
16      fprintf(stderr, "CANNOT OPEN: %s\n", f);
17      exit(0);
18    }
19  }
20
21  void i2c_24fc64_read_begin(uint8_t caddr,
22                             uint32_t maddr,
23                             uint32_t nbytes)
24  {
25    fseek(mem_fp, maddr, SEEK_SET);
26  }
27
28  uint8_t i2c_24fc64_read_byte(void)
29  {
30    return fgetc(mem_fp);
31  }
32
33  void i2c_24fc64_read_done(void)
34  {
35    ;
36  }
37
38  int8_t I2C_Exist(uint8_t ctlbyte, uint8_t addr)
39  {
```

```
40    return 1;
41  }
```

ソースファイル xc.h (XC8 代替ヘッダ)

本エミュレータプログラムを構成するソースファイルの多くはファームウエアのソースファイルを再利用しています. しかしファームウエアのソースファイルでは PIC マイコン用 XC8 コンパイラ特有のヘッダファイル xc.h をインクルードしていますから, この代替となるものを GCC 等の C コンパイラ用に用意します. つまりファームウエアのコンパイルの際には XC8 コンパイラ用の xc.h がインクルードされ, エミュレーションプログラム emuscript のコンパイルの際には以下に示す xc.h がインクルードされるようにします. これにより無修正でファームウエアのソースファイルを再利用できます.

ソースファイル番号 28.27: xc.h

```
1  /* ダミーの xc.h */
2  #define CLRWDT()   /*CLRWDT*/
```

ソースファイル Makefile (ビルド用設定ファイル)

Linux や FreeBSD などの Unix 系オペレーティングシステムおよび Windows オペレーティングシステムにおいて chkroute をビルトする方法を説明します. コンパイラ等の開発環境は第 24.4 節 (411 ページ) を参照して下さい.

ビルドのための Makefile の内容を以下に示します. この Makefile は Unix 系オペレーティングシステムと Windows オペレーティングシステムのどちらでも make によるビルドが可能なように書き方を工夫しています. なおソースファイルのディレクトリの設定が Makefile の先頭にありますから, 各自の状況に合わせて適宜修正を行って下さい.

ソースファイル番号 28.28: Makefile

```
1  ifeq ($(OS), Windows_NT)
2    EXEEXT = .exe
3    CC = gcc
4    RM = cmd.exe /C del
5  else
6    EXEEXT =
7    CC = gcc
8    RM = rm -f
9  endif
10
11 SRC1DIR=../stub
12 SRC2DIR=../../firmware/RouteCtr-B.X
13 SRC3DIR=../../firmware/lib
14
15 PROGRAM=chkroute$(EXEEXT)
16 OBJS_CHK=chkroute.o action-stub.o fatal-stub.o
17 OBJS_FAR=route.o mem.o mem-stub.o
18 OBJS=$(OBJS_CHK) $(OBJS_FAR)
19 VPATH=$(SRC1DIR) $(SRC2DIR) $(SRC3DIR)
20 CFLAGS=-I. -I$(SRC1DIR) -I$(SRC2DIR) -I$(SRC3DIR)
21
22 .PHONY: all clean distclean
23 all: $(PROGRAM)
24 clean:
25   -$(RM) $(PROGRAM) *.o *.OBJ
26 distclean:
27   -$(RM) *.o *.OBJ core *.core *~
28
29 $(PROGRAM):   $(OBJS)
30   $(CC) -o $(PROGRAM) $(OBJS)
31 %.o: %.c xc.h
```

```
32 |   $(CC) -c $(CFLAGS) -o $@ $<
```

make コマンドを実行すると自動的にビルドが行われます. なお Windows オペレーティングシステムの場合には, コマンド名は make ではなく mingw32-make となりますので注意して下さい.

課題

1. 図 28.9 における 3 つの経路 F ↔ A, G ↔ A, H ↔ A を表 28.4 に追加して下さい.
2. 上問で追加した経路を, 本文中で示した経路設定データに追加して下さい.
3. アブソリュート型ロータリエンコーダはどのような内部構造をしているか調査して下さい.
4. アブソリュート型ロータリエンコーダの出力にはグレイコードが使用されているものが多くあります. グレイコードを使用する利点は何か説明して下さい.
5. 本章で紹介した装置では, EEPROM に記憶した経路設定データに基づくポイントの切り替えもしくは全ポイントの定位への切り替えしかできません. すなわち任意にひとつのポイントを指定して定位・反位に切り替えはできません. そこで, 任意にひとつのポイントを指定して切り替える機能を追加して下さい.
6. 本章で紹介した装置では, 切り替える経路番号はロータリーエンコーダを手回しで選択しています. ロータリーエンコーダと押しボタンスイッチの代わりに赤外線テレビリモコンで使い, 離れた場所からポインと切り替えを操作できる装置を開発して下さい.
7. 本章の装置と第 23 章で紹介したポイント制御機能テレビリモコン式パワーパックを統合して下さい.
8. 本章で紹介した装置は EEPROM に経路設定データを記憶しています. しかし EEPROM への書き込み作業は煩わしく, 昔ながらのダイオードマトリクス回路で経路を設定するほうが楽だと感じることもあります. そこで EEPROM の代わりにダイオードマトリクス回路で経路を設定するよう, 回路とファームウエアを改造して下さい. メインボードとポイントインターフェースボードは本章で紹介したものをそのまま使用します. ダイオードマトリクスボードを新たに設計し, ダイオードマトリクス回路を MCP23017 を通じて I^2C バスから読み出しできるようにします. マイコンは MCP23017 を通じてダイオードマトリクス回路の入力を読み取り, ポイント切り替えを行います. ポイントインターフェースボードの接続枚数は最大 4 枚とし, ダイオードマトリクスボードの接続枚数は最大 4 枚とします. この改造により, 昔から慣れ親しんだダイオードマトリクス回路で経路を設定でき, しかも欠点であった一斉同時切替による大電流問題が解消できます.

第 VII 部

鉄道模型の自動運転システムの製作

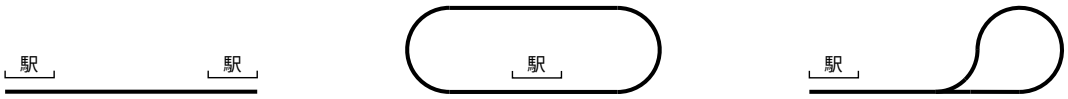
第 29 章

自動往復運転システム

本章では模型車両の自動運転を実現する装置を紹介します. 本装置では単線での自動往復運転を実現しますが, エンドレスやリバースでの自動運転も加速・減速・停車を繰り返すだけでほとんど同じですから, これらの実現もあわせて行います. 第 30 章では本章のファームウエアを拡張して, ユーザ作成の自動運転スクリプトによる自動運転システムを紹介します. これまでに発表されている自動往復運転システムと比較すると少し複雑ですが, 第 30 章の基礎ですから十分な内容の理解に努めて下さい.

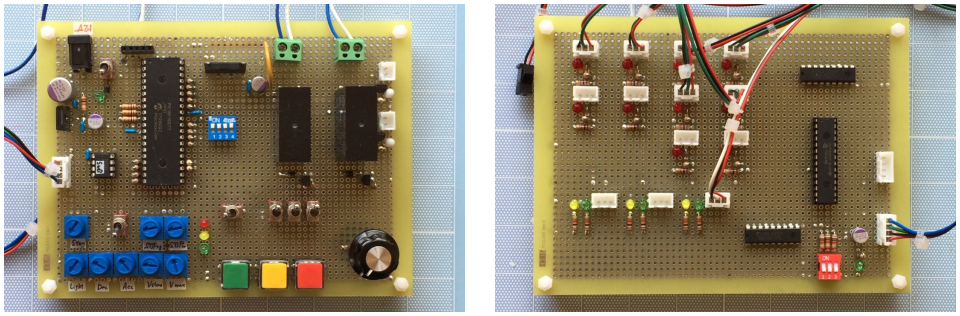
本章で紹介する回路とファームウエアの特徴は以下の通りです.

- 減速度パラメータの自動学習機能
  センサ位置に正確にしかも滑らかに停車を実現する自動学習機能を備えており, 本装置が自動的に減速度パラメータの調整を行います. 模型車両やレイアウトごとにパラメータを微調整する面倒な作業は, この機能により一切不要です.
- リバース出力つき
  リバース線を有するレイアウトに対しても自動運転制御が可能です.
- 3 種のレイアウトに対する自動運転制御プログラムを事前組み込み済み
  図 29.1 に示すように単線, エンドレス, リバースの 3 種のレイアウトに対する自動運転制御プログラムをファームウエアに組み込んでおり, DIP スイッチの切り替えで選択します. レイアウト上での車両セン



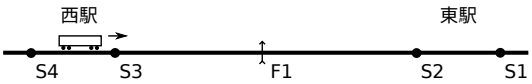
(a) 単線　**(b)** エンドレス　**(c)** リバース

図 29.1: 事前組み込み済みレイアウト 3 種



(a) メインボード　**(b)** センサ・ポイントインターフェースボード

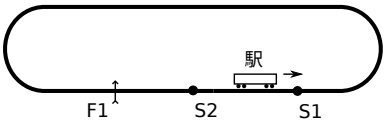
図 29.2: 自動往復運転システムの試作例



西駅と東駅の 2 つ駅があり, 模型列車は西駅から走行を開始します.

1. 模型列車は東駅に向かって走行を開始します.
2. 模型列車が車両センサ S2 に到達すると減速を開始し, 車両センサ S1 で停車します.
3. 模型列車は東駅にしばらく停車した後に, 西駅に向かって走行をします. 模型列車が車両センサ S3 に到達すると減速を開始し, 車両センサ S4 で停車します.
4. 模型列車は西駅にしばらく停車します. 以上を繰り返します.

(a) 単線トラックプラン

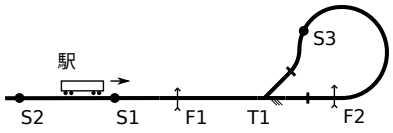


駅は 1 つで, この駅から模型列車は走行を開始します.

1. 模型列車は東に向かって走行を開始します.
2. 模型列車が車両センサ S2 に到達すると減速を開始し, 車両センサ S1 で停車します.
3. 模型列車は駅にしばらく停車します. 以上を繰り返します.

※ 模型列車はふたつのセンサの間に収まるようにして下さい. また模型列車をレイアウトに置くときは, センサ S2 で検出されない位置にして下さい.

(b) エンドレストラックプラン



駅は 1 つで, この駅から模型列車は走行を開始します.

1. ポイント T1 を定位に, フィーダ F1 と F2 を正極性に切り替えます. 模型列車は東に向かって走行を開始します.
2. 模型列車が車両センサ S3 に到達するとポイント T1 を反位に, フィーダ F1 を逆極性に切り替えます.
3. 模型列車が車両センサ S1 に到達すると減速を開始し, 車両センサ S1 で停車します.
4. 模型列車は駅にしばらく停車します. 以上を繰り返します.

※ リバース区間の長さは模型列車の編成長よりも長くして下さい.

(c) リバーストラックプラン

図 29.3: 事前組み込み済みのトラックプラン 3 種

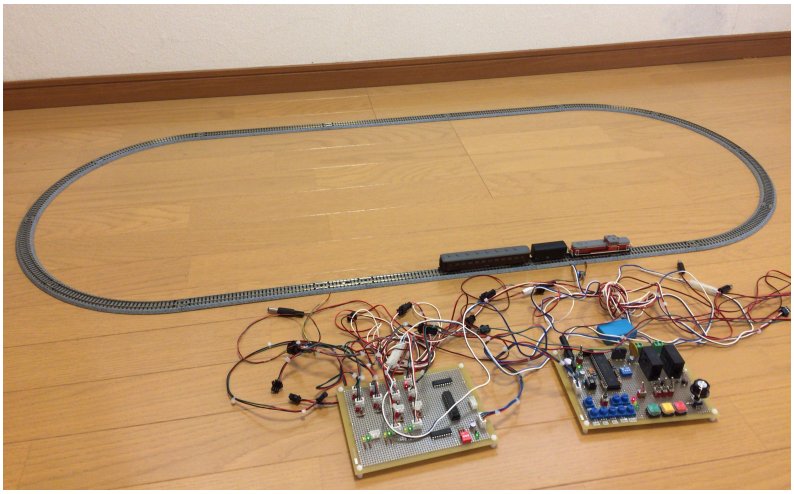


図 29.4: エンドレストラックプランの実施例 (KATO ユニトラック)

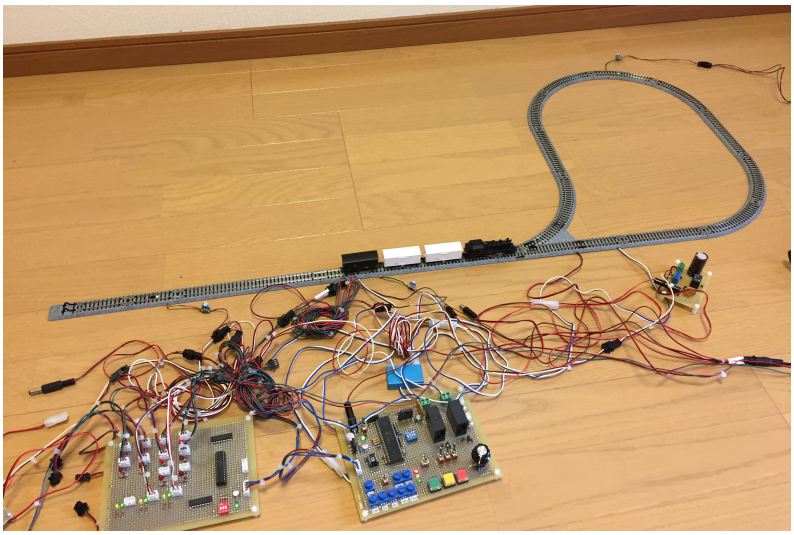


図 29.5: リバーストラックプランの実施例 (KATO ユニトラック)

サやフィーダの位置と自動運転パターンを図 29.3 に示します. 以下ではこのような図を「トラックプラン」と呼びます.

- 滑らかな加減速と低速走行
  第 6 章で学んだブレゼンハムアルゴリズムと第 10 章で学んだ低速ブースト機能を使用して, 滑らかな加減速と低速走行を実現します.
- 増設可能なインターフェースボード
  インターフェースボードにはセンサ・ポイントインターフェースボードとアクセサリインターフェースボードの二種類があり, 前者は最大 7 枚, 後者は最大 1 枚接続できます. これらはレイアウトの規模に応じて随時増設が可能で, ファームウエアの変更は一切不要です. 1 枚のセンサ・ポイントインターフェースボードにはポイント制御出力 3 つと車両センサ入力 10 が装備されていますから, ポイントは最大 21 個, 車両センサは最大 70 個の接続が可能です. 第 30 章のファームウエアを使用することで, 立山砂防工事専用軌道の連続 18 段スイッチバックが実現可能です.

```
main()
{
  for(;;) {
    if (操作モードスイッチ == 自動運転モード) {
      ar_run_start();
        (ただし操作モードスイッチが手動運転モードになれば終了);
    } else {
      手動運転を行う
        (ただし操作モードスイッチが自動運転モードになれば終了);
    }
  }
}

ar_run_start() {
  自動運転開始のボタンが押されるのを待機;
  for (;;) {
    自動運転の制御を行う有限状態機械を1ステップ動作させ,
        走行速度等を変化させる;
    現在のPWMデューティー比をモータドライバに出力;
  }
}
```

図 29.6: ファームウエア構造 (概要)

- 手動運転機能つき
  走行を開始する定位置へ模型車両を回送するために, 手動による走行とポイント切替の操作が可能です.

本章で学ぶ内容は以下の通りです. 新たに学ぶことは意外と少なく, これまでに学んできた内容を基礎にして新たに少し学ぶことで本装置のすべてが理解可能です.

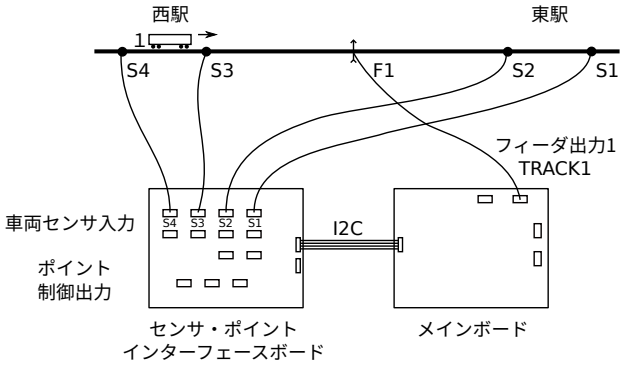
- 有限状態機械によるシーケンス制御
- 減速度パラメータの自動学習の実現手法

本章では以下のものを使用します.

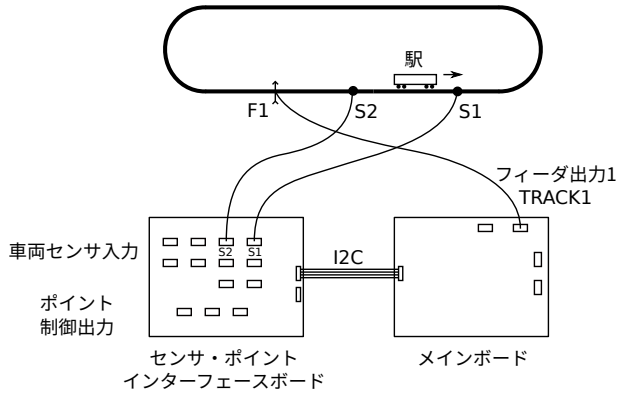
- メインボード: 1 枚 (本章)
- センサ・ポイントインターフェースボード: 1 枚 (本章)
- 車両センサ: トラックプランにより 2 個から 4 個 (第 16 章)
- ポイント切替装置: トラックプランにより 1 個 (第 12 章および第 14 章)

本装置で運転できる範囲は, リバース線出力用の逆転スイッチを装備したパワーパックとポイント切替コントローラを有するアナログ運転システムとほぼ同等です. 機能を限定することで回路とファームウエアを簡潔にし, 容易に理解が可能なものにします. 本装置が持たない機能は, ブロックコントロール, 閉塞制御, 複数列車同時制御などが挙げられます. これらの機能をお望みの方は別の自動運転システムの導入をご検討下さい.

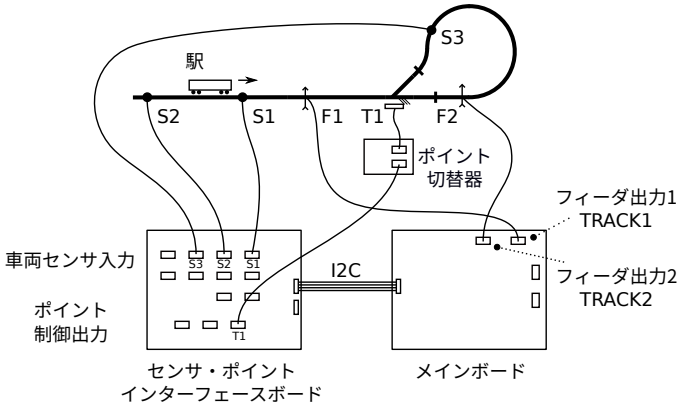
この装置のファームウエアの大まかな構造を図 29.6 に示します. 手動運転モードはパワーパックのファームウエアと同じ構造をしているので説明は省略します.



(a) 単線



(b) エンドレス

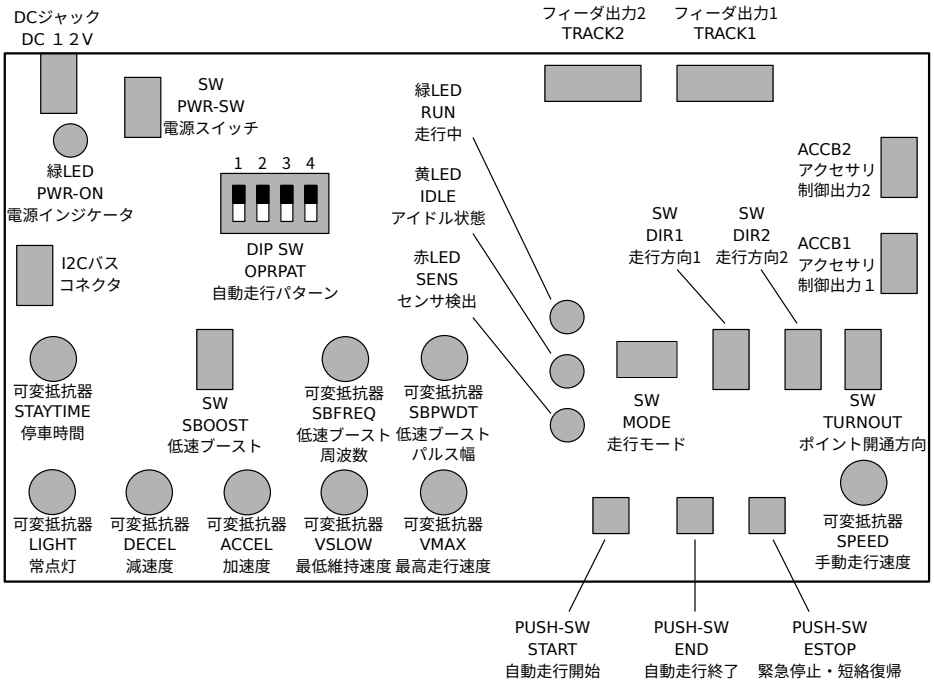


(c) リバース

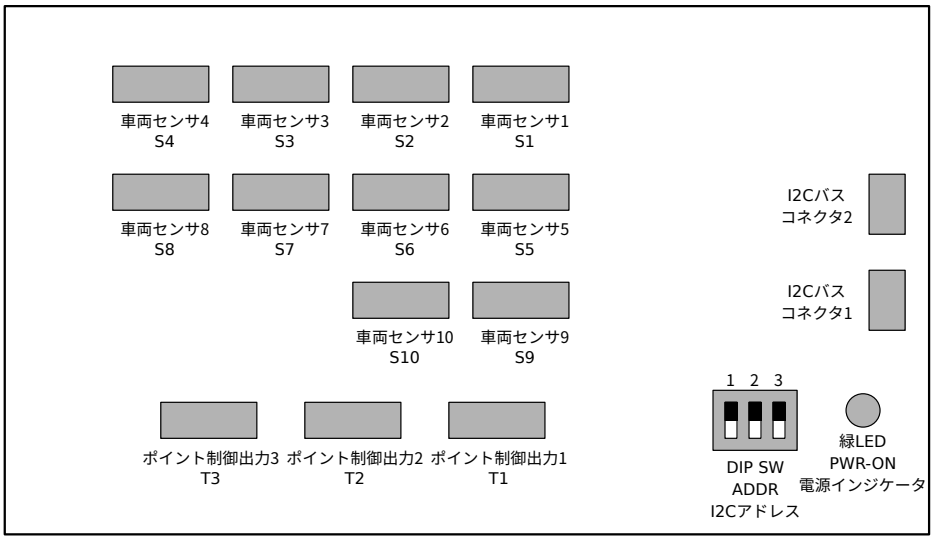
図 29.7: 接続方法

29.1　使用方法

本装置の使用方法を以下に説明します. 以下の説明で使用する部品名称は図 29.8 に配置例とともに示しています. 3 つのトラックプランそれぞれに対する配線は図 29.7 に示す通りです.



(a) メインボード



(b) センサ・ポイントインターフェースボード

図 29.8: 自動往復運転システムの操作表示部品の配置例と名称

ボードの種類

本装置は以下の 3 種類のボードより構成されます. なお本章の範囲ではメインボード 1 枚とセンサ・ポイントインターフェースボード 1 枚を使用します. アクセサリインターフェースボードは第 30 章で紹介する自動

表 29.1: メインボード DIP スイッチ OPRPAT 設定

DIP スイッチ 1	2	3	4	動作
OFF	OFF	OFF	OFF	(事前組込) 単線トラックプラン
ON	OFF	OFF	OFF	(事前組込) エンドレストラックプラン
OFF	ON	OFF	OFF	(事前組込) リバーストラックプラン
OFF	ON	ON	ON	インターフェースボード動作テスト
ON	ON	ON	ON	EEPROM 自動運転スクリプト (第 30 章)
その他の組み合わせは未定義				

表 29.2: インターフェースボード DIP スイッチ設定

DIP スイッチ 1	2	3	種別	備考
OFF	OFF	OFF	センサ・ポイント (1 番)	センサ番号 1~10, ポイント番号 1~3
ON	OFF	OFF	センサ・ポイント (2 番)	センサ番号 11~20, ポイント番号 4~6
OFF	ON	OFF	センサ・ポイント (3 番)	センサ番号 21~30, ポイント番号 7~9
ON	ON	OFF	センサ・ポイント (4 番)	センサ番号 31~40, ポイント番号 10~12
OFF	OFF	ON	センサ・ポイント (5 番)	センサ番号 41~50, ポイント番号 13~15
ON	OFF	ON	センサ・ポイント (6 番)	センサ番号 51~60, ポイント番号 16~18
OFF	ON	ON	センサ・ポイント (7 番)	センサ番号 61~70, ポイント番号 19~21
ON	ON	ON	アクセサリ (1 番)	アクセサリ番号 17~32
· センサ・ポイントインターフェースボードでは 3 つすべてを ON にしないこと				
· アクセサリインターフェースボードでは 3 つすべてを ON にすること				
· アクセサリ番号 1~2 はメインボードにて実装済 (3~16 は未実装)				

運転システムで使用します.

- メインボード
  マイコン, モータドライバ, フィーダ出力などを装備しており, 可変抵抗器により様々なパラメータの調整を行います. フィーダ出力は 2 系統あり, フィーダ出力 TRACK1 は通常の出力でフィーダ出力 TRACK2 はリバース線用の出力です. アクセサリ制御出力は 2 系統を装備しています.
- センサ・ポイントインターフェースボード
  車両センサ入力 10 系統, ポイント制御出力 3 系統を装備しています.
- アクセサリインターフェースボード
  アクセサリ制御出力 16 系統を装備しています.

接続

接続手順は以下の通りです.

1. メインボード 1 枚とセンサ・ポイントインターフェースボード 1 枚を用意します. なお本章では EEPROM 24FC64 の装着は不要です.

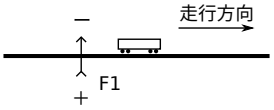


図 29.9: トラックプランの記号: フィーダ

2. メインボードとセンサ・ポイントインターフェースボードを接続します. なお接続ケーブルについては回路の説明の際に説明します.
3. メインボードの DIP スイッチ OPRPAT をトラックプランに合わせて表 29.1 に従って設定します.
4. センサ・ポイントインターフェースボードの DIP スイッチを表 29.2 に従って設定します. 本章ではセンサ・ポイントインターフェースボードの DIP スイッチは 1 番に設定して使用します. (なおセンサ・ポイントインターフェースボードでは 3 つすべてを ON にしないで下さい. ショートを起こして故障する場合があります.)
5. 次にメインボードおよびセンサ・ポイントインターフェースボードと線路の間の接続を行います. 各トラックプランに対するの電気配線の概略を図 29.7 に示しますから, この図に従って接続をして下さい. トラックプランには以下の内容が記載されています.
   - フィーダ: F1, F2
     模型車両走行用の電力を供給する場所です. F1 はメインボードのフィーダ出力 TRACK1 へ, F2 は TRACK2 へ, それぞれ接続します. フィーダ記号は図 29.9 に示す通り, 模型車両の走行方向が右の場合にはフィーダ記号の矢尻側がマイナス, 矢羽側がプラスになるように接続します.
   - センサ: S1, S2, S3, ...
     車両センサを設置する場所です.
   - ポイント: T1, T2, T3, ...
     ポイントを表します. 本装置のポイント制御出力とポイント切替器を接続し, ポイント切替器の出力をポイントマシンに接続します. 定位・反位も合わせて下さい.

状態表示 LED

本装置には電源ランプのほかに 3 つの LED があり, 自動運転の状態を表示します.

- RUN (緑 LED)
  走行中に点灯します.
- IDLE (黄 LED)
  停車中に点灯します. オートオフ機能による自動運転の終了準備中には点滅します.
- SENS (赤 LED)
  車両センサによる検出時に点灯します. 車両センサの調整や接続の確認の際に便利です.

手動運転モードと自動運転モード

本装置には手動運転モードと自動運転モードの 2 つのモードがあり, トグルスイッチ MODE で切り替えます. 自動運転においてはトラックプランにおける模型列車の定位置を考える必要があります. ここで定位置とは, 自動運転の開始時に模型車両が置かれるべき位置のことです. 定位置に模型車両が置かれていることを前提にして自動運転は動作を開始しますから, 異なる位置に置かれていると正しく自動運転はできません. 定位置への回送を自動化できれば便利なのですが, その仕組みの実現は大変面倒なので手動運転で行うことにします.

表 29.3: パラメータの初期設定の参考値

パラメータ名	設定値
ACCEL (加速度)	25%
DECEL (減速度)	25%
VMAX (最高走行速度)	50%
VSLOW (最低維持速度)	25%
SBOOST (低速ブースト有効)	オン
SBPWDT (低速ブーストパルス幅)	40%
SBFREQ (低速ブースト周波数)	40%
LIGHT (常点灯)	0%
STAYTIME (停車時間)	10%

- 手動運転モード
  人間が手動で模型車両の走行を行うモードで, 定位置へ模型車両を回送する場合に使用します.
  - 可変抵抗器 SPEED : 手動運転速度
    手動運転時での走行速度を操作します. 模型車両の急発進を防止するラッシュ防止機能を持たせています.
  - トグルスイッチ DIR1 : 走行方向 1
    フィーダ出力 TRACK1 での走行方向を操作します. このスイッチは中点を有する On-Off-On 型であり, 中点の位置の場合には常点灯はオフになります.
  - トグルスイッチ DIR2 : 走行方向 2
    フィーダ出力 TRACK2 での走行方向を操作します. リバース線での利用を想定しています.
  - トグルスイッチ TURNOUT : ポイント切替
    ポイント切替を操作します. このスイッチひとつでポイント全てを切り替えますので, トラックプランによっては模型車両を定位置へ回送するための経路の構成が不可能かもしれません. その場合には面倒ですが手づかみで模型車両を定位置へと運んで下さい.
- 自動運転モード
  自動的に模型車両の走行を行うモードです.
  - 押しボタンスイッチ START
    自動運転を開始します.
  - 押しボタンスイッチ END
    自動運転を終了します. ただし直ちに自動運転を終了とせず, 模型車両が定位置に戻ってから終了とします.
  - 押しボタンスイッチ ESTOP
    自動運転を直ちに終了します. 事故発生が目前に迫っている場合に使用します.

設定パラメータ

メインボードの設定パラメータを説明します. パラメータの初期設定の参考値も記しますので, パラメータ調整の際の参考にして下さい. これらを一覧表にまとめたものを表 29.3 に示します.

- ACCEL : 加速度
  模型車両が加速する際に使用するパラメータです. 0% の場合は即座に加速され, 100% の場合は最も時間をかけてゆっくりと加速されます. 初期設定は 25% 程度が良いでしょう. 手動運転モードではこのパラメータは使用しません.
- DECEL : 減速度
  模型車両が減速する際に使用するパラメータです. 0% の場合は即座に減速され, 100% の場合は最も時間をかけてゆっくりと減速されます. 初期設定は 25% 程度が良いでしょう. 手動運転モードではこのパラメータは使用しません. 本装置にはこの減速度パラメータの自動学習機能があり, ここでの設定はパラメータの初期値として使用されます. 初期値の設定が著しく外れていれば学習に要する走行回数は多くなりますが, 設定がおおよそ近い値ならば数回の走行で十分な学習が行えます.
- VMAX : 最高走行速度
  模型車両が走行する際の最高速度を設定します. 0% の場合は最も低速で, 100% の場合は最も高速です. 初期設定は 50% 程度が良いでしょう.
- VSLOW : 最低維持速度
  模型列車が停止することなく走行できる速度に設定します. 0% の場合は最も低速で, 100% の場合は最も高速です. 初期設定は 25% 程度が良いでしょう. 詳しくは後述します. 手動運転モードではこのパラメータは使用しません.
- トグルスイッチ SBOOST : 低速ブースト有効
  低速ブースト機能のオン・オフを設定します. オンの場合には以下の 2 つのパラメータ SBFREQ と SBPWDT が有効になります. なお低速ブースト機能については第 10 章を参照して下さい. 特に図 10.9 (186 ページ) では低速ブーストパルス幅と低速ブースト周波数を図示して説明しているので参考にして下さい. なおコアレスモータを搭載している模型車両の場合には, 低速ブースト機能はオフにするほうが低速走行できる場合があります. これはコアレスモータはわずかの電圧でも強いトルクで低速回転できる優れた性能を持っているためです.
- SBPWDT : 低速ブーストパルス幅
  低速ブーストパルス幅を設定します. 0% の場合はパルス幅が小さくてブーストは最も弱く, 100% の場合はその逆でブーストは最も強くなります. 初期設定は 40% 程度が良いでしょう.
- SBFREQ : 低速ブースト周波数
  低速ブースト周波数を設定します. 0% の場合はパルスの発生回数が少なくブーストは最も弱く, 100% の場合はその逆でブーストは最も強くなります. 初期設定は 40% 程度が良いでしょう.
- LIGHT : 常点灯機能
  常点灯機能での輝度を設定します. 0% の場合は輝度は 0 で消灯状態で, 100% の場合はその逆で輝度は最大になります. 初期設定は 0% が良いでしょう. 設定が大きいと模型車両が走り出しますし, 発熱量の大きな電球の場合は加熱により模型車両を痛める場合がありますから, 設定は各自で適切に行って下さい.
- STAYTIME : 停車時間
  駅などでの停車時間を設定します. 0% のときに停車時間は最小, 100% のときに停車時間は最大となります. 初期設定は 10% 程度が良いでしょう.

VMAX と VSLOW の調整にはスイッチ操作が必要なので以下で説明します.

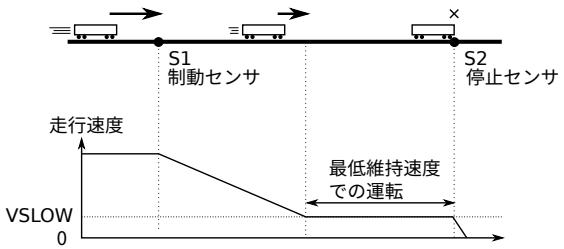
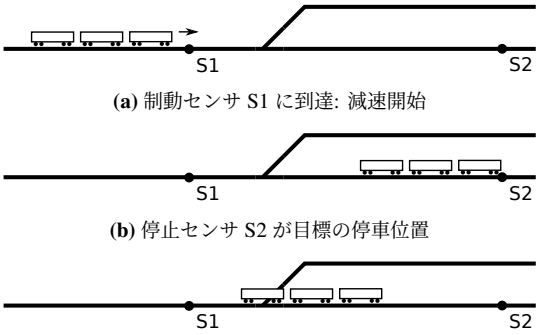


図 29.10: VSLOW と最低維持速度での走行



(a) 制動センサ S1 に到達: 減速開始

(b) 停止センサ S2 が目標の停車位置

(c) 停止センサ S2 の前で停車すればポイント封鎖

図 29.11: 徐行速度の設定の重要性

- 最高走行速度 VMAX の調整
  トグルスイッチ MODE を自動運転モードから手動運転モードに切り替えます. ただし他のスイッチは同時に押さずに行います. 調整手順は以下の通りです. (1) 可変抵抗器 VMAX をいったん 0% に設定します. (2) 可変抵抗器 SPEED を 100% いっぱいにします. (3) そして可変抵抗器 VMAX を調整して望みの速度で模型車両が走行するようにします. つまり可変抵抗器 SPEED を 100% いっぱいにしたときの走行速度が自動運転時の最高速度で, これが通常の走行速度となります. 自動運転のときにはこの速度まで加速しますから, 事故防止の為にも最初のうちは控えめな最高速度から始めて下さい.
- 最低維持速度 VSLOW の調整
  END スイッチを押しながらトグルスイッチ MODE を自動運転モードから手動運転モードに切り替えます. 調整手順は以下の通りです. (1) 可変抵抗器 VSLOW をいったん 0% に設定します. (2) 可変抵抗器 SPEED を 100% いっぱいにします. (3) そして模型車両が低速走行する速度に VSLOW を調整します. このときの走行速度が最低維持速度です.

最低維持速度の調整では停止寸前の超低速にはせず, ある程度のゆっくり走行する速度にして下さい. 減速度パラメータの自動学習において測定誤差が大きくなり, 的確な自動学習を速やかに行えなくなるためです. 本ファームウエアでは, 低速ブーストや常点灯の機能で線路に加えている電圧は測定していません. 最低維持速度の設定が小さい場合にはこれらの要因が無視できない大きさとなって測定誤差が大きくなるからです.

目標の停車位置で確実な停車を実現する仕組み

模型車両を目標の停車位置へ停車させるために 2 つの車両センサを組にして使用します.

- 制動センサ
  制動の開始位置に設置する車両センサです.
- 停止センサ
  停車の目標位置に設置する車両センサです.

最低維持速度の設定は停車目標位置へ確実に模型車両の到達を実現するための機構です. 図 29.10 に示す通り, 模型列車が制動センサを通過すれば減速を開始します. しかし停止センサに到達する前に最低維持速度まで減速されれば最低維持速度を保って走行し, 停止センサに到達すれば減速を再開して停車します. 図 29.11 を用いてこの機能の重要性を説明します.

- 図 29.11a: 模型列車は制動センサ S1 を通過して減速を開始します.
- 図 29.11b: 停止センサ S2 が模型列車が目標とする停車位置です.
- 図 29.11c: もし減速しすぎて目標の停車位置よりも前で模型車両が止まった場合には, 模型列車がポイントを塞いでしまいます. この事態の回避には停止センサ S2 の位置まで確実に模型列車を走行させねばなりません. 本章ではこの例が発生するトラックプランはありませんが, 第 30 章で紹介するトラックプランではこの事態の回避は必須です.

本装置では, 目標の停車位置まで停止することなく低速走行を維持するために, 最低維持速度の設定機能と低速ブースト機能を使用します. この 2 つの機能により模型車両は確実に前進し, 目標の停車位置へ確実に到達させます. もし最低維持速度の設定値が小さいと摩擦等の理由で模型車両が停止してしまう場合がありますから, 最低維持速度の設定値は走行可能ギリギリな値にはせず少し余裕を持たせたほうが良いでしょう. 適切な最低維持速度は模型車両ごとに異なりますから, ハードウエアあるいはファームウエアで走行速度を固定的な値とするわけにはゆきません. そこで模型車両ごとに簡単に設定・調節できるように, 本装置では可変抵抗器ひとつで設定できるようにしています.

29.2　回路図

回路図と部品表は以下の通りです. 本章で紹介する自動運転を行うには, 先述の通りメインボード 1 枚とセンサ・ポイントインターフェースボード 1 枚を作成します. リバース線での自動運転を行う場合にはポイント切替器が 1 つ必要です.

- メインボード
  回路図を図 29.12 と図 29.13 に, 部品表を表 29.4 に, それぞれ示します.
- センサ・ポイントインターフェースボード
  回路図を図 29.14 に, 部品表を表 29.5 に, それぞれ示します.

本章の回路は部品数が多いですが, PIC マイコンに接続された TB6643KQ, 24FC64, MCP23017 はこれまでに学んだ回路を組み合わせたものです. モータドライバ TB6643KQ の出力をそのままフィーダ出力とはしていません. ふたつの 2 回路 C 接点のリレーを介して 2 系統のフィーダ出力として, リレーを電子的に制御可能な逆転スイッチとして使用しています. これまでに紹介したパワーパックではモータドライバ TB6643KQ の入力端子 IN1 と IN2 への入力信号により前進と後進の逆転を行っていましたが, 本回路ではリレーの切り替えによって逆転を行います. この点はこれまでに紹介したパワーパックとは大きく異なるところです. 従って本装置の走行制御の能力はリバース出力付きのパワーパックと同等です. 従って複数の模型列車を同時に走行できません. なぜならモータドライバはひとつだけで, その出力をリレーに分岐しているからです.

リレーの駆動に要する電流は PIC マイコンのポートの駆動能力を越える電流の可能性がありますので, FET を通じてリレーのコイルを駆動します. またリレーのコイル部分にはオン・オフの際に大きな逆起電力が発生しますので, 逆起電力で FET が破壊されないようにダイオードを接続して逆起電力を吸収します. ダイオードの設置場所はリレーのすぐそばとし, 配線長も極力短くして下さい.

センサ・ポイントインターフェースボードは I/O エクスパンダ MCP23017 のポートを車両センサ入力やポイント制御出力としています. 車両センサの感度調整や動作確認が楽になるよう, 車両センサの検出を知らせる LED を備えています. ポイント制御出力は十分な電流の駆動ができるようにトランジスタアレイを介して出力します. ポイント制御出力には開通方向の状態が分かるよう, 動作モニタ用の LED を備えています. ボードの増設のために MCP23017 の I^2C アドレスを設定する DIP スイッチを取り付けています.

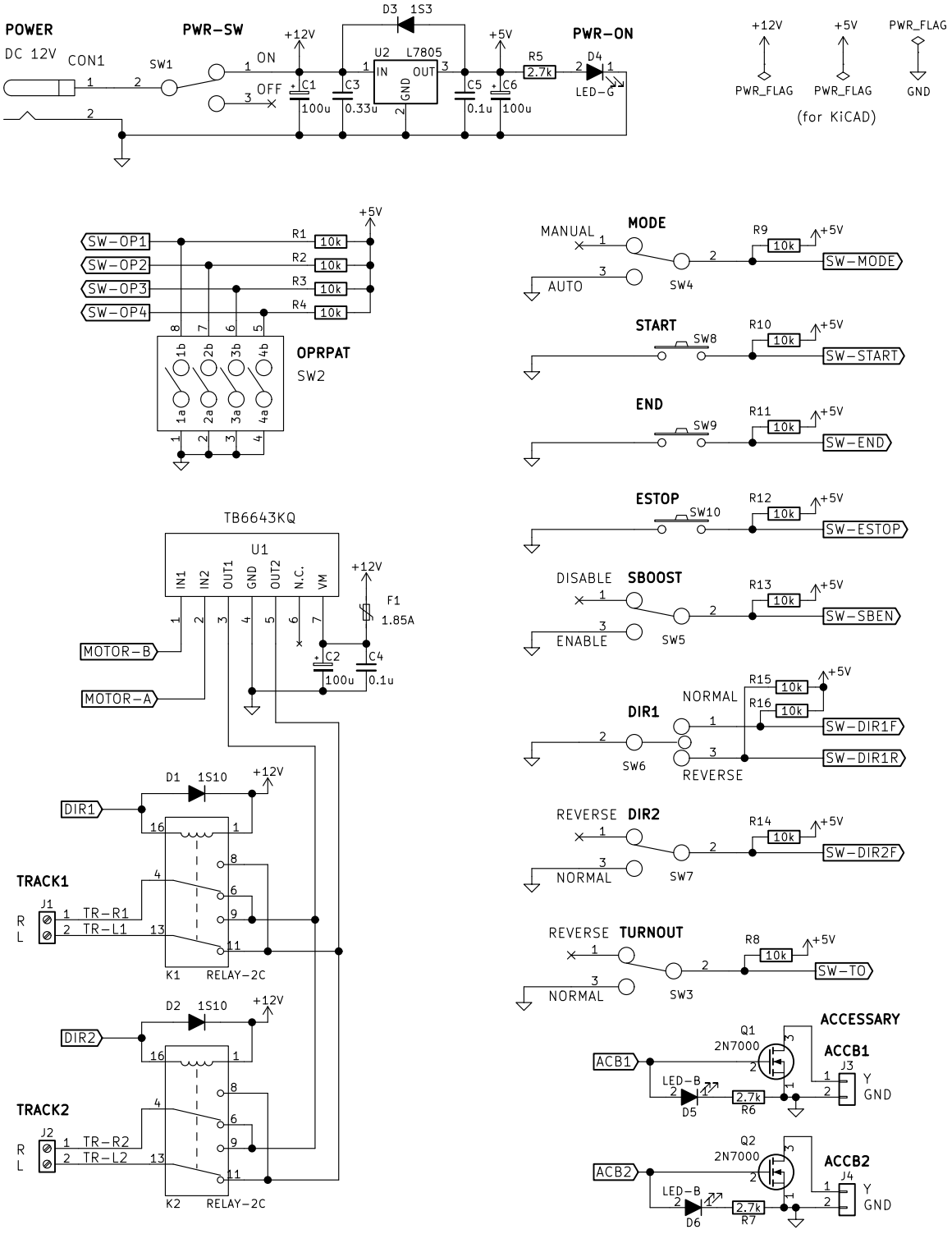


図 29.12: 【回路図】 自動往復運転システム / メインボード (1/2)

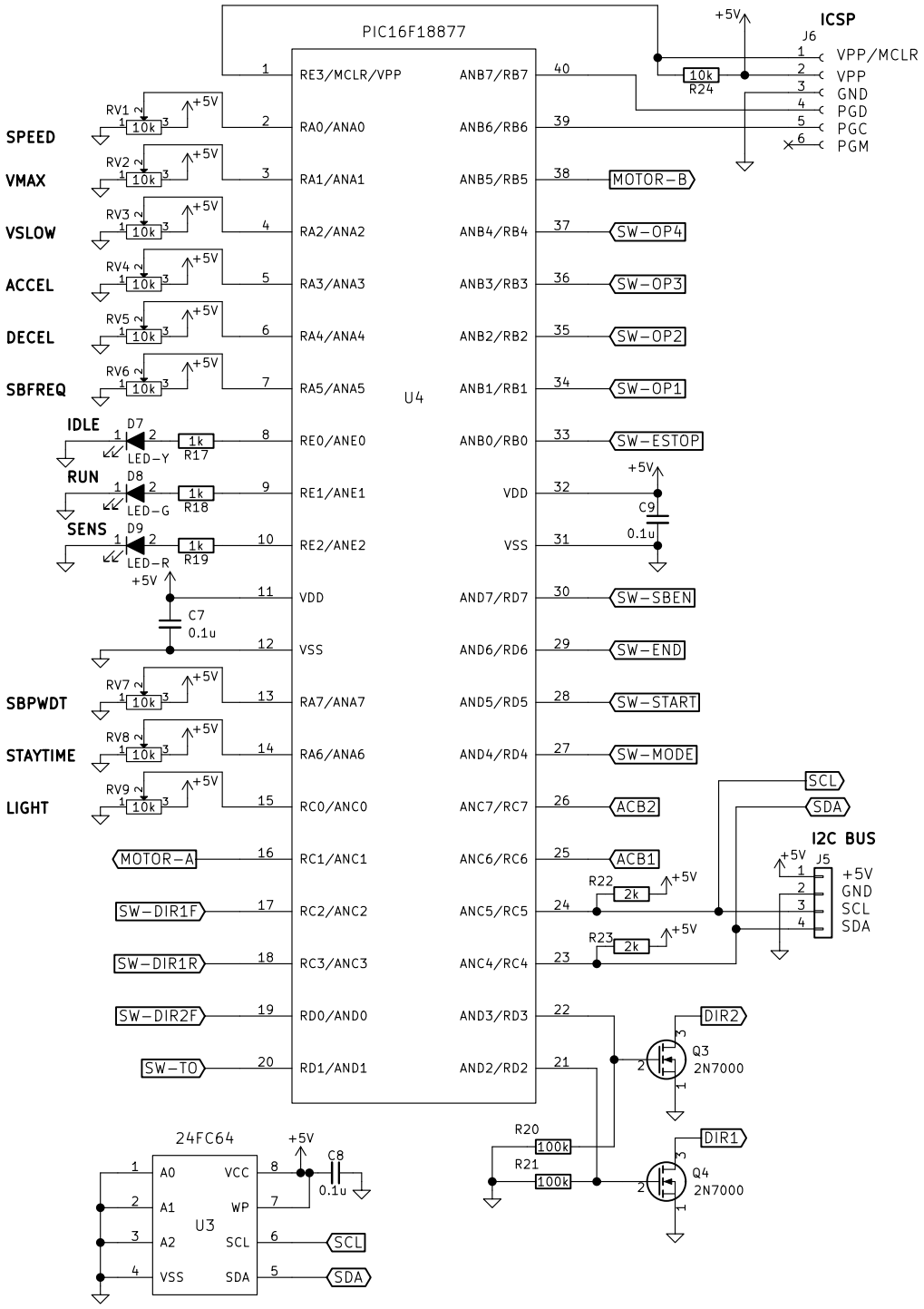


図 29.13: 【回路図】 自動往復運転システム / メインボード (2/2)

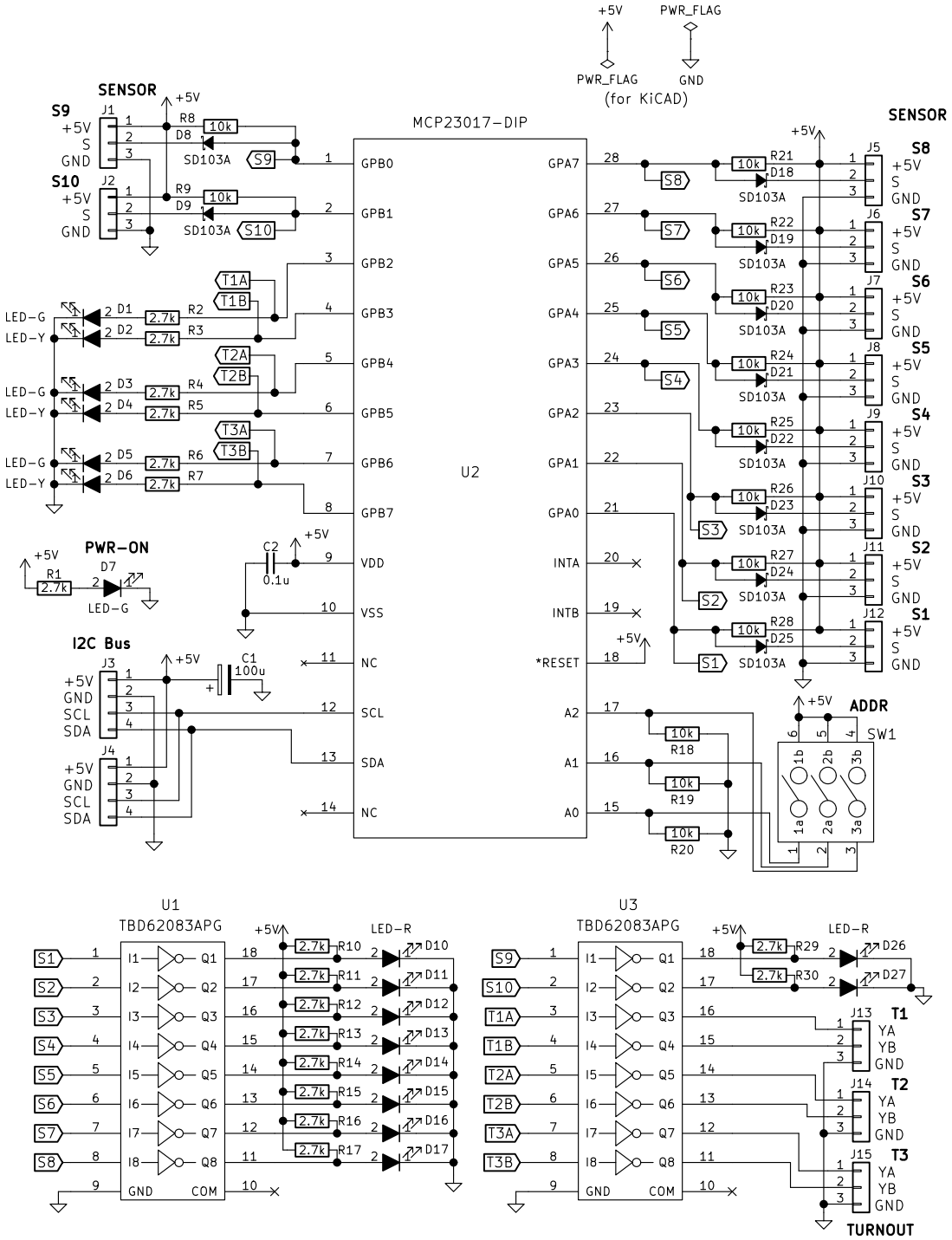


図 29.14: 【回路図】 自動往復運転システム / センサ・ポイントインターフェースボード

表 29.4: 【部品表】 自動往復運転システム / メインボード

記号	数量	部品名
U4	1	PIC 16F18877-I/P [I-11901]
U3	1	I^2C EEPROM 24FC64-I/P [I-03567]
U1	1	モータドライバ TB6643KQ [I-07688]
Q1, Q2, Q3, Q4	4	FET 2N7000 [I-09723]
U2	1	3 端子レギュレータ L7805CV-DG　1.5A (又は同等品) [I-07903]
D1, D2	2	ダイオード 1S10　100V 1A [I-00126]
D3	1	ダイオード 1S3　30V 1A [I-01707]
D4, D8	2	LED 緑 [I-06413]
D7	1	LED 黄 [I-11639]
D9	1	LED 赤 [I-04780]
D5, D6	2	LED 青 [I-06413]
F1	1	ポリスイッチ MF-R185　1.85A [P-12629]
C2	1	電解コンデンサ 100μF　耐圧 25V 以上 [P-07760]
C1, C6	2	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C3	1	セラミックコンデンサ 0.33μF [P-08147]
C4, C5, C7, C8, C9	5	セラミックコンデンサ 0.1μF [P-00090]
RV1	1	可変抵抗器 10kΩ　B カーブ, スピード制御用 [P-12576]
RV2, RV3, RV4, RV5, RV6, RV7, RV8, RV9	8	可変抵抗器 10kΩ　B カーブ, 半固定型 [P-08012]
		可変抵抗器用ツマミ　使用する可変抵抗器に合わせて適宜
R17, R18, R19	3	抵抗 1kΩ　1/4W [R-25102]
R22, R23	2	抵抗 2kΩ　1/4W [R-25202]
R5, R6, R7	3	抵抗 2.7kΩ　1/4W [R-25272]
R1, R2, R3, R4, R8, R9, R10, R11, R12, R13, R14, R15, R16, R24	14	抵抗 10kΩ　1/4W [R-25103]
R20, R21	2	抵抗 100kΩ　1/4W [R-25104]
	1	IC ソケット 40 ピン [P-00034]
	1	IC ソケット 8 ピン [P-00035]
K1, K2	2	リレー 2 回路 C 接点　12V 2A [P-01228]
SW1, SW3, SW4, SW5, SW7	5	単極双投トグル SW　On-On 型 [P-02399]
SW6	1	単極双投トグル SW・中点付　On-Off-On 型 [P-02400]
SW8, SW9, SW10	3	プッシュ SW [P-03651]
SW2	1	DIP SW 4P [P-08923]

(つづく)

(つづき)

記号	数量	部品名
J1, J2	2	ターミナルブロック 2 ピン [P-02333]
J3, J4	2	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
J5	1	基板用コネクタ 4 ピン　JST XH コネクタ B4B-XH-A(LF) [C-12249]
J6	1	ピンソケット 6 ピン　ICSP 用 (無くても可) [C-03784]
CON1	1	DC ジャック [C-09408]
	1	AC アダプタ 12V　出力 3A [M-10661]
		ほか基板・配線材等

表 29.5: 【部品表】 自動往復運転システム / センサ・ポイントインターフェースボード

記号	数量	部品名
U2	1	MCP23017-E/SP　I/O エキスパンダ [I-09486]
U1, U3	2	TBD62083APG　トランジスタアレイ [I-10669]
D8, D9, D18, D19, D20, D21, D22, D23, D24, D25	10	ショットキーバリアダイオード SD103A [I-04271]
D1, D3, D5, D7	4	LED 緑 [I-06413]
D2, D4, D6	3	LED 黄 [I-11639]
D10, D11, D12, D13, D14, D15, D16, D17, D26, D27	10	LED 赤 [I-04780]
C1	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C2	1	セラミックコンデンサ 0.1μF [P-00090]
R1, R2, R3, R4, R5, R6, R7, R10, R11, R12, R13, R14, R15, R16, R17, R29, R30	17	抵抗 2.7kΩ　1/4W [R-25272]
R8, R9, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27, R28	13	抵抗 10kΩ　1/4W [R-25103]
	1	IC ソケット 28 ピン [P-01339]
SW1	1	DIP SW 3P [P-09992]
J13, J14, J15	3	基板用コネクタ 3 ピン　JST XH コネクタ B3B-XH-A(LF) [C-12248]
J1, J2, J5, J6, J7, J8, J9, J10, J11, J12	10	基板用コネクタ 3 ピン　JST ZH コネクタ B3B-ZR-3.4(LF)(SN) [C-14156]
J3, J4	2	基板用コネクタ 4 ピン　JST XH コネクタ B4B-XH-A(LF) [C-12249]
	2	コネクタハウジング 4 ピン　JST XH コネクタ XHP-4 [C-12257]
	8	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
		ほか接続用ケーブル・基板・配線材等

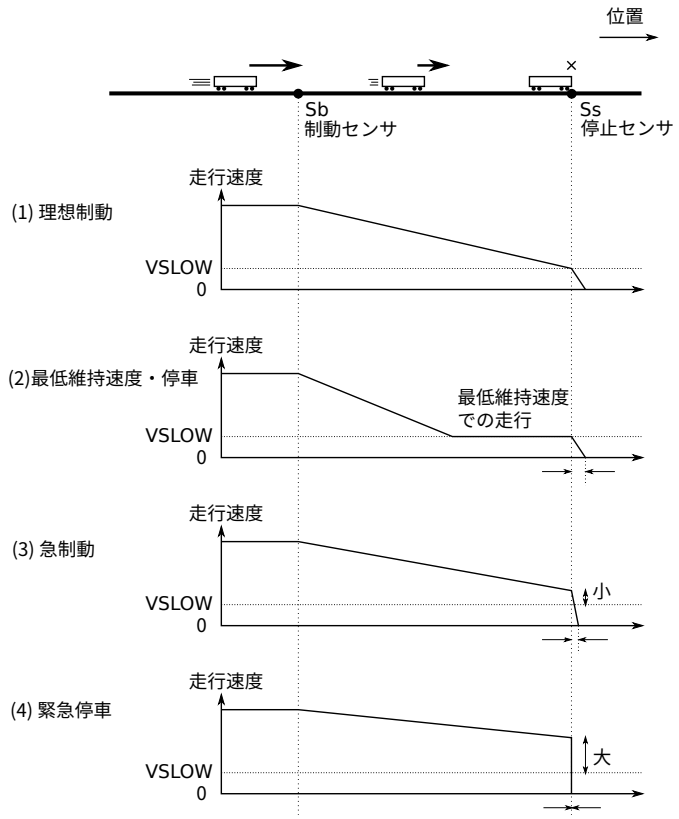


図 29.15: 制動の開始から停車までの走行速度の変化

29.3　減速度パラメータの自動学習

本節では本ファームウエアの最大の特色である減速度パラメータの自動学習の方法を説明します. 自動学習の基本的な考え方は以下の 3 点に集約されます.

1. 制動センサから停止センサまでの距離 L を測定します.
2. 測定した距離 L に対し, 走行する距離が L の時になめらかに停車するような減速度パラメータを計算して更新します.
3. 誤差のため一度では正確な値は求まらないので, 上記の測定と計算を繰り返し行います.

以下ではこの考え方の具体的な実現方法を説明します.

減速と停車の速度変化

車両センサを使用した減速と停車の速度変化を概説します. 模型車両の停車には 2 つの車両センサを組にして使用し, 一方を制動センサ, 他方を停止センサと呼びます. 制動の開始から停車までの走行速度の変化は図 29.15 に示す通りで, 停車するまでの走行速度の変化には以下の通りいくつかの場合があります.

1. 理想制動
   理想的な制動パターンです. 停止センサに到達した時に走行速度がちょうど最低維持速度 VSLOW に減速される場合です. このパターン以外の場合では, 減速度の調整を行うことでこの制動パターンとなるようにします.

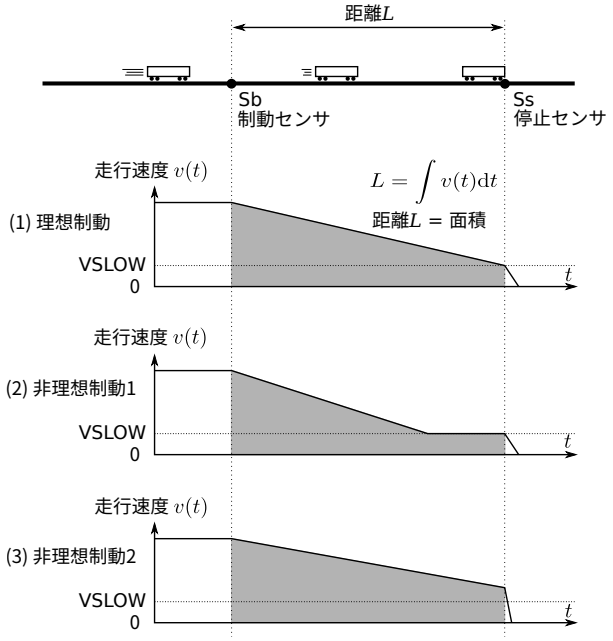


図 29.16: 制動センサと停車センサの間の距離 L と走行距離の関係

2. 最低維持速度運転・停車
   停止センサに到達前に走行速度が最低維持速度 VSLOW にまで下がる場合です. 最低維持速度を下回らないように走行速度を維持して, 停止センサまで走行したら減速して停車します. このパターンでは減速度が大きいために早く速度が低下していますから, 減速度を小さくして理想制動に近づける必要があります.
3. 急制動
   停止センサに到達時に走行速度が最低維持速度 VSLOW を少し越える場合です. この場合は急制動をかけて停車します. このパターンでは減速度が小さいために速度が低下していませんから, 減速度を大きくして理想制動に近づける必要があります.
4. 緊急停車
   停止センサに到達時に走行速度が最低維持速度 VSLOW を大きく越える場合です. この場合は瞬時に走行速度を 0 にして緊急停車を行います. このパターンでも減速度が小さいために速度が低下していませんから, 減速度を大きくして理想制動に近づける必要があります.

停止センサまで走行したら模型車両は停車しますが, 図 29.15 に示すように瞬時に走行速度を 0 にするのではなく, 減速度を大きくして滑らかに停車します. ただし走行速度が最低維持速度 VSLOW より大きく外れていると大きく過走しますから, 外れ度合いの大小により減速度の大小を決めて滑らかな停車と過走防止の両立を行います.

減速度パラメータの理想値

理想的な減速度パラメータについて説明します. 表 29.6 は以降で使用する主な記号の一覧ですので参考にして下さい. 図 29.15 で示したように, 理想制動となるのは停止センサに到達した時に走行速度がちょうど最低維持速度 VSLOW になる減速度パラメータの値の場合です. 図 29.16 に示すように制動センサと停車センサの間の距離を L とし, 時刻 t での走行速度を $v(t)$ とします. 走行速度 $v(t)$ は減速度パラメータに従って時々刻々減

表 29.6: 主な記号

記号	意味	備考
L	制動センサと停車センサの距離 (単位: メートル)	
D^*	減速度パラメータ (理想時)	
t	時刻 (単位: 秒)	
Δ_T	繰り返し 1 ステップの時間間隔 (単位: 秒)	
s	繰り返しステップ番号	$t = \Delta_T s$ の関係
K	比例定数 $v = Kw$	
w_1	制動センサ通過時の PWM デューティ比 (理想時)	惰行時の速度で決まる
w_2	停止センサ通過時の PWM デューティ比 (理想時)	最低維持速度 VSLOW で決まる
$v(t)$	時刻 t での走行速度 (単位: メートル/秒)	
$w(s)$	第 s ステップでの PWM デューティ比	測定可能

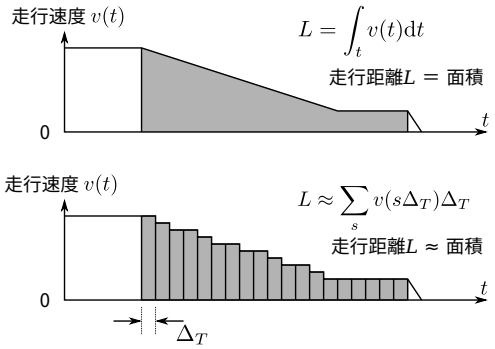


図 29.17: 走行距離の近似算出法

少します. 制動センサが模型車両を検出した時刻を $t = 0$ とし, 停止センサが模型車両を検出した時刻を $t = \tau$ とすると, 理論的には制動パターンに関わらず以下の式が成立します. なおこの式での積分値は図 29.16 での網掛けの部分の面積に等しいことに注意して下さい.

$$L = \int_{t=0}^{\tau} v(t)\mathrm{d}t \tag{29.1}$$

以上のことから理想制動を実現する理想の減速度パラメータは以下の条件を満たすものです.

- 模型車両が距離 L を走行すると, 走行速度はちょうど最低維持速度 VSLOW に減速される.

理想的な減速度パラメータを求める方法

上述の条件に基づいて理想的な減速度パラメータを求める方法を説明します. モータの PWM デューティ比 w と走行速度 v の間には比例関係, すなわちある比例定数 K に対して以下の関係式が成立すると仮定します.

$$v = Kw \tag{29.2}$$

長大編成や急勾配でない限りはこの関係は近似として妥当と思われますし, ファームウエアではモータの PWM デューティ比 w は分かりますが, 走行速度 v は分かりません. この理由により以降ではこの比例関係を仮定して計算の簡便化を行います. なお以降では, 最低維持速度 VSLOW の値は v_2 (走行速度) ではなく w_2 (PWM

デューティ比) を表すものとします.

式 29.1 による走行距離の計算式は積分で与えられていますが, ファームウエアでは数値積分により近似的に行わざるをえません. そこで図 29.17 に示す方法を使用して走行距離の近似値を求めます. 具体的には, 図の灰色の部分の面積が走行距離に一致しますから, 階段状に近似した形状の面積を走行距離の近似値とします. ファームウエアの繰り返し動作の時間間隔を Δ_T とすれば, ファームウエアの繰り返し動作の第 s ステップでの時刻は $t = s\Delta_T$ となります. 第 s ステップでのモータの PWM デューティ比の値を $w(s)$ とすれば走行距離に関する以下の近似式を得ます. なおこの式は最低維持速度での運転となる場合や急制動となる場合のいずれにも当てはまりますから, 制動パターンの違いには関係なく成立する点に注意して下さい.

$$\begin{aligned} L &= \int_t v(t)\mathrm{d}t \\ &\approx \sum_s v(s\Delta_T)\Delta_T = \Delta_T K \sum_s w(s) \end{aligned} \tag{29.3}$$

理想的な減速度パラメータ D^* の場合での走行を考えます. 制動センサ通過時の時刻 $t = 0$ での走行速度を $v_1 = Kw_1$, 停車センサ通過時の時刻 $t = \tau$ での走行速度を $v_2 = Kw_2$ とします. すると 2 センサ間の走行に要するステップ数を S は以下の通り求められます.

$$\begin{aligned} w(s) &= w_1 - D^* s \\ w_2 &= w(S) = w_1 - D^* S \\ \therefore S &= (w_1 - w_2)/D^* \end{aligned}$$

この理想的な減速度パラメータの場合における走行距離 L の近似値は以下の通り求められます. なお第 S ステップの時刻は $\tau = S\Delta_T$ となることに注意して下さい.

$$\begin{aligned} L &= \int_0^\tau v(t)\mathrm{d}t \\ &\approx \sum_{s=0}^{S-1} v(s\Delta_T)\Delta_T = \sum_{s=0}^{S-1} Kw(s)\Delta_T = \Delta_T K \sum_{s=0}^{S-1} (w_1 - D^* s) \\ &= \Delta_T K\left\{w_1 S - D^* S(S-1)/2\right\} \\ &\approx \Delta_T K\left\{w_1 S - D^* S^2/2\right\} = \Delta_T K\left\{2w_1(w_1 - w_2) - (w_1 - w_2)^2\right\}/(2D^*) \\ &= \Delta_T K(w_1^2 - w_2^2)/(2D^*) \end{aligned} \tag{29.4}$$

式 29.3 と式 29.4 はともに同じ走行距離 L の走行であることをから, 以下の近似式を得ます.

$$L \approx \Delta_T K(w_1^2 - w_2^2)/(2D^*) \approx \Delta_T K \sum_{s=0}^{S-1} w(s)$$

以上より理想的な減速度パラメータ D^* は, 理想的ではない場合での走行時での数値を基にして次の式で計算されます.

$$D^* \approx \frac{w_1^2 - w_2^2}{2\sum_{s=0}^{S-1} w(s)} \tag{29.5}$$

ここまでの結果をまとめます. 理想的な減速度パラメータ D^* は式 29.5 を用いて算出可能で, この式で使用する $w_1, w_2, \sum_s w(s)$ はいずれもファームウエア内で値を得ることができます. 2 センサ間の物理的な距離 L や比例係数 K の値を得るにはファームウエア外で物理的な測定が必要ですが, それらが全く不要である点が式 29.5 の大変重要な特長です.

測定値に基づく減速度パラメータの計算

減速度パラメータを測定値から計算する仕組みを説明します. 実際の回路とファームウエアでは, 理想的な減速度パラメータ D^* を以下の通り自動的に算出して次回の走行で使用します.

1. 実際に使用する減速度パラーメータを d とし, 初期値は可変抵抗器 DECEL による設定値とします.
2. 制動センサが模型車両を検出すれば, そのとき時のモータの PWM デューティ比の値 $w(s)$ を w_1 として記憶します. そして PWM デューティ比の積算 $\sum_s w(s)$ を初期値 0 から開始します.
3. ファームウエアが繰り返しを行う度に PWM デューティ比の積算 $\sum_s w(s)$ を更新します. 走行速度の減速に際しては PWM デューティ比 $w(s)$ が最低維持速度 VSLOW による設定値を下回らないようにします.
4. 停止センサが模型車両を検出すれば最低維持速度 VSLOW の値を w_2 として記憶します. 以上で $w_1, w_2, \sum_s w(s)$ の値が求まりました. 式 29.5 に基づいて理想的な減速度パラメータ D^* の推測値を算出し, 減速度パラメータ d をこの値に更新して次回の走行で使用します.

原理上は一度の減速度パラーメータの計算で理想的な減速度パラーメータの近似値が得られますが, 実際上は測定誤差などのために一度では理想的な減速度パラーメータは得られません. 従って実際には走行を行う度に計算を行い, 理想的な減速度パラーメータへと近づけてゆきます.

29.4　有限状態機械による自動運転の実現

有限状態機械による自動運転の実現の手法を説明します. ここで紹介する手法はごく単純なシーケンス制御であり, 模型車両等の走行制御の手順を順序的に実行します. 自動運転における制御の対象は以下のものです.

- 模型車両
  線路電圧の制御により走行速度の制御を行います.
- ポイント
  ポイント制御出力によりポイント切替を行います.
- アクセサリ
  アクセサリ制御出力により各種装置の動作を行います.

様々な自動運転パターンの設計・理解・実装を容易にするには系統的な手法の確立が重要です.

- 設計の容易さ
  もし自動運転パターンの設計が容易でなければ設計の正しさの確認が困難となります.
- 理解の容易さ
  もし既存の自動運転パターンの理解が容易でなければ機能の拡張やデバッグなどの保守, 他のトラックプランへの転用等が困難となります.
- 実装の容易さ
  もし自動運転パターンの実装が容易でなければバグが混入する可能性が高まります.

そこで設計・理解・実装を容易にするために有限状態機械によりモデル化を行います. 有限状態機械の構成は以下の方針で行います.

- 状態
  模型車両の位置, フィーダの極性, ポイントの開通方向等, 自動運転システムの現在の状態を表現します.

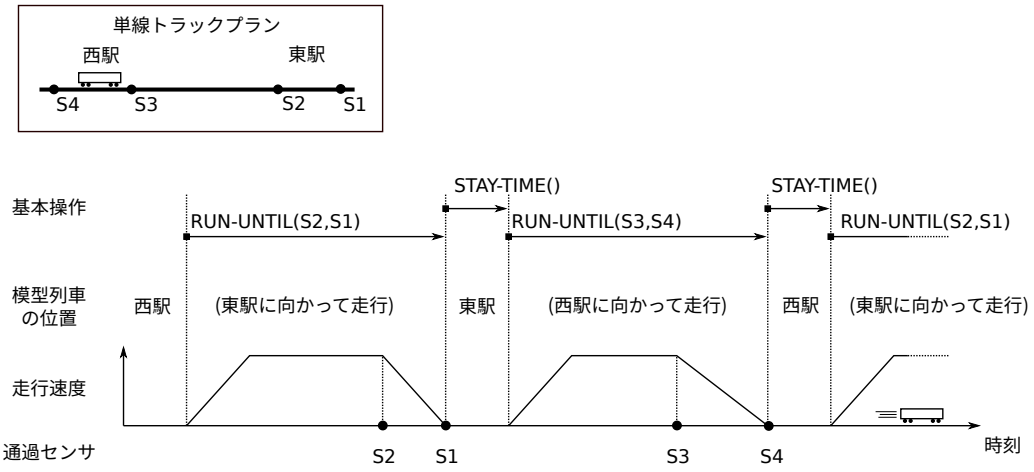


図 29.18: 単線トラックプランでの自動運転を実現する基本操作の列: 模型列車の位置, 走行速度, 通過する車両センサの関係

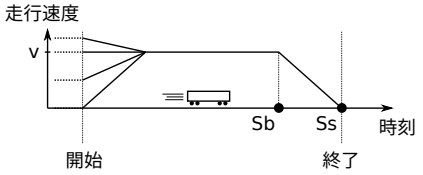
- 状態遷移規則
  以下の事象の発生により状態遷移を行い, 次の動作に移ります.
  - 押しボタンスイッチ START, END, ESTOP の押下
    人による自動運転の開始や終了の指示です.
  - 車両センサによる模型車両の検出
    模型車両の減速の開始, 停止, あるいは通過の検出です.
  - タイマによる時間経過の検出
    駅での停車などで使用する, 一定時間の待機の終了の検出です.

基本操作と状態遷移

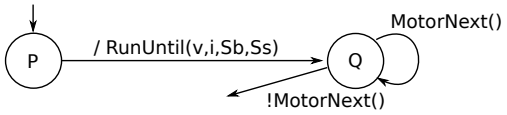
模型車両の自動運転を実現するには, 力行・惰行・制動の制御を行い目的の停車位置への正確な停車が必要です. トラックプランごとに細かな記述をするのは面倒であり, また間違いの生じる元ですから, 自動走行の記述を簡便に行う方法を定めることは重要なことです. そこで本書では, 走行の制御の基本的な動作を 3 つの基本操作 RUN-UNTIL, RUN-TO, STAY-TIME に抽象化し, 自動運転の仕様を記述は基本操作の列の記述で行うことにします. 2 つの駅を往復する単線トラックプランを図 29.3a で示しましたが, この自動運転で呼び出す基本操作の列と模型列車の位置, 走行速度, 通過する車両センサの関係を図 29.18 に示します. この図と図 29.21 およびソースファイル 29.8 を参照・比較することで, 模型車両がどのような走行をするかを理解できるようになって下さい.

3 つの基本操作 RUN-UNTIL, RUN-TO, STAY-TIME の詳細は以下の通りです.

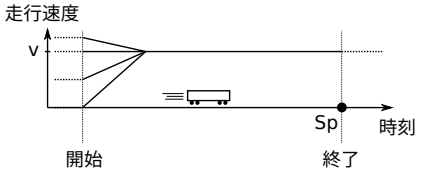
- センサ停止走行 RUN-UNTIL(v,i,Sb,Ss)
  模型車両の走行を開始して, 車両センサ Ss まで走行した後に停車します. この基本操作には 4 つの引数があります. v は惰行速度, i は減速度番号, Sb は制動センサとして使用する車両センサ番号で, Ss は停止センサとして使用する車両センサ番号です. ここで i は減速度の値そのものではなく, 番号で間接的に指定している点に注意して下さい. 詳しくは後述します. この基本操作の詳細は以下の通りで, 図 29.19a でその動作を図示しています.
  - その時点での位置と走行速度から走行を開始します. 多くの場合では走行速度 0 からの力行となり



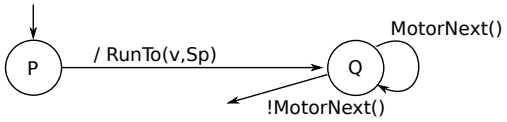
(a) RUN-UNTIL(v,i,Sb,Ss) による走行と停車



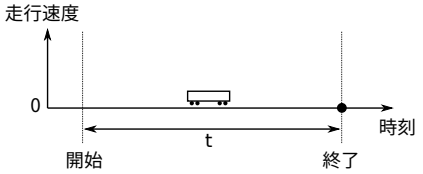
(b) センサ停止走行 RUN-UNTIL(v,i,Sb,Ss) に対する状態遷移図の断片



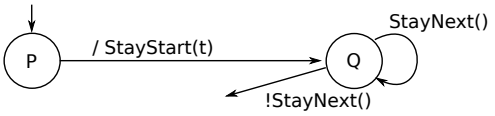
(c) RUN-TO(v,Sp) による走行



(d) センサ通過走行 RUN-TO(v,Sp) に対する状態遷移図の断片



(e) STAY-TIME(t) による時間待機



(f) 時間待機 STAY-TIME(t) に対する状態遷移図の断片

図 29.19: 基本操作に対する状態遷移図の断片と動作

ます.

- 走行速度が v となるように力行あるいは制動を行い, その後は惰行します.
- 模型車両がセンサ Sb に到達すると減速を開始し, センサ Ss に到達すると停車してこの基本操作を終えます.

状態遷移図における表現は図 29.19b の状態遷移図断片で示す通り, ふたつの状態を使用して記述します. なお状態遷移図では記法 RunUntil(v,i,Sb,Ss) を使用します. 以下での説明のために, ふたつの状態を仮に P と Q と呼ぶことにします. まず状態 P において走行の引数を指定して状態 Q に遷移します. 次に状態 Q で模型車両は停車センサ Ss に向けて走行します. 模型列車が停車センサ Ss に到達し停車するまで MotorNext() の値は真です. 停車すると MotorNext() の値は偽となり, 状態 Q から次の状態へと遷移します.

- センサ通過走行 RUN-TO(v,Sp)

  模型車両の走行を開始して車両センサ Sp まで走行しますが停車はしません. この基本操作には 2 つの引数があります. v は惰行速度, Sp は通過センサとして使用する車両センサ番号です. この基本操作の詳細は以下の通りで, 図 29.19e でその動作を図示しています.

  - その時点での位置と走行速度から走行を開始します. 多くの場合では走行速度 0 からの力行となります.
  - 走行速度が v となるように力行あるいは制動を行い, その後は惰行します.
  - 模型車両がセンサ Sp に到達するとこの基本操作を終えます. このとき模型列車は走行速度 v で走行していることに注意して下さい. 多くの場合ではこの後に, センサ停止走行 RUN-UNTIL(v,i,Sb,Ss) により停車します.

  状態遷移図における表現は図 29.19d の状態遷移図断片で示す通り, ふたつの状態を使用して記述します. なお状態遷移図においては記法 RunTo(v,Sp) を使用します. 以下での説明のために, ふたつの状態を

仮に P と Q と呼ぶことにします. まず状態 P において走行の引数を指定して状態 Q に遷移します. 次に状態 Q で模型車両は通過センサ Sp に向けて走行します. 模型列車が通過センサ Sp を通過するまで MotorNext() の値は真です. 通過すると MotorNext() の値は偽となり, 状態 Q から次の状態へと遷移します.

- 時間待機 STAY-TIME(t)
  指定した時間が経過するまで待機します. この基本操作の引数 t は待機時間です. この基本操作の詳細は以下の通りで, 図 29.19e でその動作を図示しています.
  - 時刻の経過の測定を開始します.
  - 時刻 t が経過するとこの基本操作を終えます.

  状態遷移図における表現は図 29.19f の状態遷移図断片で示す通りであり, ふたつの状態を使用して記述します. なお状態遷移図においては記法 StayStart(t) を使用します. 以下での説明のために, ふたつの状態を仮に P と Q と呼ぶことにします. まず状態 P において時間待機の引数を指定して状態 Q に遷移します. 次に状態 Q で一定時間の経過を待機します. 一定時間が経過していない間は StayNext() の値は真です. 経過すると StayNext() の値は偽となり, 状態 Q から次の状態へと遷移します.

センサ停止走行操作 RUN-UNTIL(v,i,Sb,Ss) の引数 i は減速度パラメータの値そのものではなく, 減速度番号としている理由を説明します. 本装置では減速度パラメータを自動的に調整をしますから, 減速度パラメータの値そのものを固定的な値として与えるわけにはゆきません. またトラックプランにおいて停車する位置は複数ありますから, それぞれに異なる減速度パラメータを用意して管理する必要がありますし, 減速度パラメータの自動学習で使用する各種データも記憶する必要があります. そこで減速度パラメータや自動学習用の各種データを減速度番号により抽象化します.

減速度番号に関する注意点は, ひとつの減速度番号に対して異なる減速センサ Sb と停止センサ Ss の組を指定してはならないということです. 例えば, RUN-UNTIL(32,2,1,3) と RUN-UNTIL(32,2,1,4) のように, ひとつの減速度番号に対して異なる車両センサの対を指定してはならないということです. この例では減速度番号 2 に対して車両センサの組 1 と 3 および組 1 と 4 を指定しています. これでは減速度パラメータの学習はできません. 車両センサの組 1 と 3 および組 1 と 4 で交互にひとつの減速度パラメータを学習することになりますから, これでは適切な減速パラメータはいつまでも得られません.

基本状態

状態の定義はトラックプランや自動運転パターンごとに異なりますが, 自動運転システムで統一的な取扱いを可能とするための共通かつ最低限の基本的な状態を以下の通り定めます.

- 初期状態 INIT
  自動運転が開始された時の最初の状態で, このとき模型列車は定位置に置かれているものとします. 状態遷移図においてはこの状態を図 29.20a の形で表現します.
- 定位置状態 HOME
  模型列車が定位置に停車している状態で, 自動運転の終了をこの状態のときに限定します. 次回の自動運転を行う際には模型車両を移動させる必要はありません. 状態遷移図においてはこの状態を図 29.20b の形で表現します.



(a) 初期状態 INIT



(b) 定位置状態 HOME

図 29.20: 自動運転における基本状態

有限状態機械の構成方法

自動運転パターンを記述する有限状態機械の構成方法を説明します. 構成方法は上記の 3 つの基本操作の合成を行い状態遷移図を描くことで行います. なおフィーダの極性の逆転, ポイント切替, アクセサリ制御出力のオン・オフ等も必要ですが, これらは状態遷移図に動作の記述を加えるだけです.

遷移の動作部の記述には以下の記法を使用します.

- RunUntil(v,i,Sb,Ss)
  センサ停止走行基本操作の引数を設定します.
- RunTo(v,Sp)
  センサ通過走行基本操作の引数を設定します.
- StayStart(t)
  時間待機基本操作の引数を設定します.
- M(x)
  フィーダ出力すべてのオン・オフを設定します. 引数 x はオン・オフを表し, 値は On または Off です. 常点灯のオン・オフの制御が主な目的ですが, オンにしないと走行はできませんので注意が必要です.
- F(f,d)
  フィーダ出力の極性の設定により模型車両の走行方向を設定します. 引数 f はフィーダ番号で, 値は 1 または 2 です. 引数 d は走行方向で, 値は Normal(前進) または Reverse(後進) です.
- T(p,d)
  ポイント切替を行います. 引数 p はポイント番号です. 引数 d は開通方向で, 値は Normal(定位) または Reverse(反位) です.
- A(x)
  アクセサリ制御出力のオン・オフを設定します. 引数 x はオン・オフを表し, 値は On または Off です.

遷移の条件部の記述には以下の記法を使用します.

- MotorNext()
  センサ停止走行あるいはセンサ通過走行において, 目標センサへ未到達か否かを判定します.
- StayNext()
  時間待機が未完了か否かを判定します.

自動運転パターンを実現する有限状態機械の例を以下に示します. 図 29.3 で示した自動運転用のトラックプラン 3 種に対する自動運転パターンを実現する有限状態機械を状態遷移図の形で構成したものを図 29.21 (往復トラックプラン), 図 29.22 (エンドレストラックプラン), 図 29.23 (リバーストラックプラン) に示します. これらは図 29.19 に示した基本状態と基本状態操作に対する状態遷移図断片を機械的に合成して得られている点

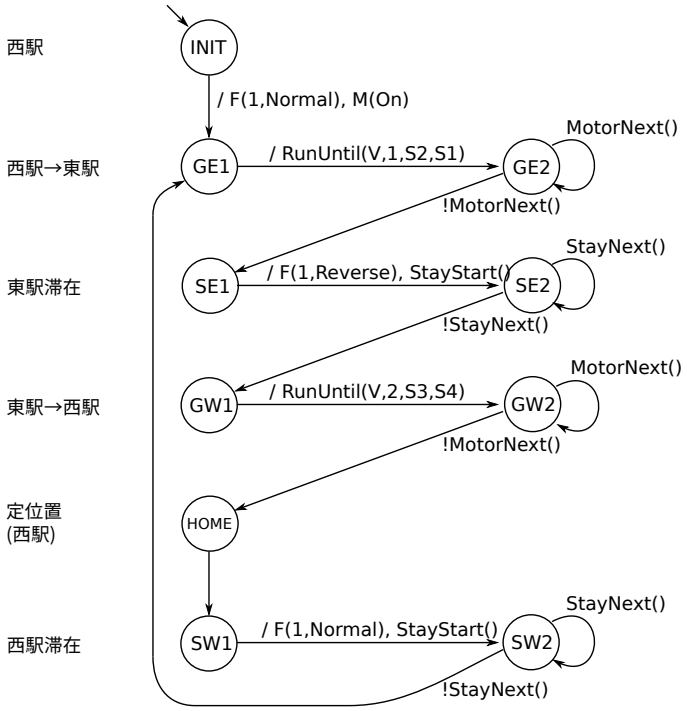


図 29.21: 単線トラックプランに対する自動運転有限状態機械の状態遷移図

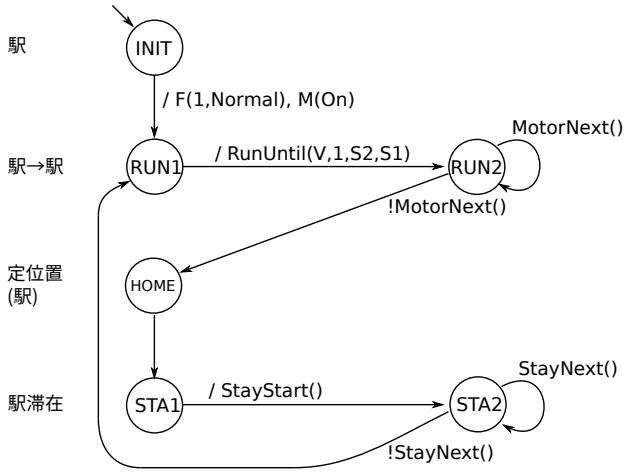


図 29.22: エンドレストラックプランに対する自動運転有限状態機械の状態遷移図

に注目して下さい.

以上で説明した通り, 状態遷移図の構成方法と条件・動作の記法により自動運転パターンの設計・理解が容易化ができました. 実装の容易化は記法それぞれに対応する C 言語の関数をファームウエアで用意することで実現します.

29.5　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します. 図 29.24 に主要なソースファイル間の関係を示します.

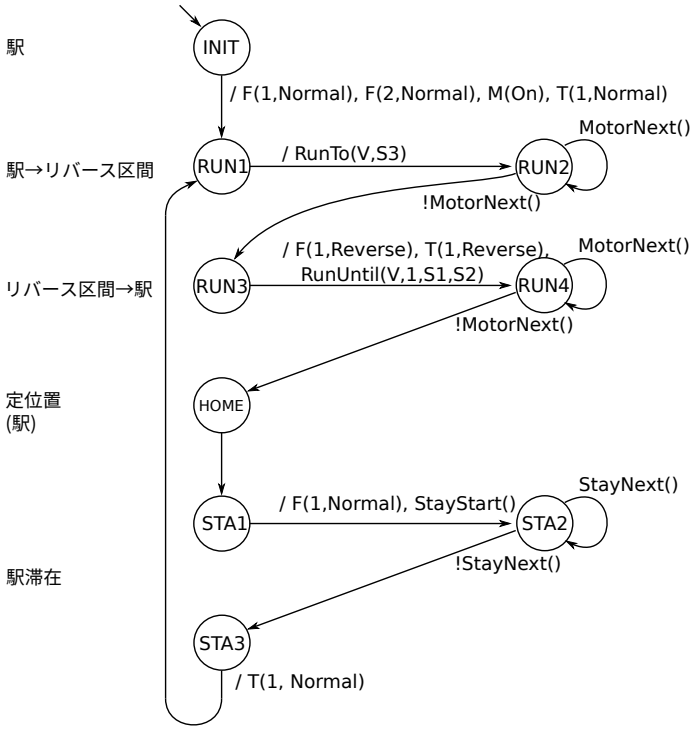


図 29.23: リバーストラックプランに対する自動運転有限状態機械の状態遷移図

表 29.7: 【ソースファイル一覧】 自動往復運転システム

初	頁	番号	ファイル名	内容	行数
✓	577	29.2	AutoTrainOp.c	主ソースコード	175
✓	580	29.3	fatal.h	エラー表示: ヘッダ	24
✓	581	29.4	fatal.c	エラー表示: 実装	48
✓	582	29.5	manual-run.c	手動運転: 実装	120
✓	584	29.6	fsm.h	自動運転有限状態機械: ヘッダ	10
✓	584	29.7	fsm.c	自動運転有限状態機械: 実装	30
✓	585	29.8	fsm-shuttle.c	自動運転有限状態機械 (単線): 実装	105
✓	587	29.9	fsm-endless.c	自動運転有限状態機械 (エンドレス): 実装	76
✓	588	29.10	fsm-reverse.c	自動運転有限状態機械 (リバース): 実装	101
✓	590	29.11	script-dummy.c	スクリプト言語: ダミー実装	25
✓	591	29.12	if.h	インターフェースボード: ヘッダ	99
✓	592	29.13	if.c	インターフェースボード: 実装	90
✓	374	22.16	if-toq.h	ポイント切替動作キュー: ヘッダ	12
✓	392	23.2	if-toq.c	ポイント切替動作キュー: 実装	122
✓	594	29.14	if-sensor.c	インターフェースボード・センサ部: 実装	113
✓	596	29.15	if-turnout.c	インターフェースボード・ポイント部: 実装	61
✓	597	29.16	if-accessary.c	インターフェースボード・アクセサリ部: 実装	128

(つづく)

(つづき)

初	頁	番号	ファイル名	内容	行数
✓	599	29.17	ar.h	自動運転: ヘッダ	47
✓	600	29.18	ar.c	自動運転: 実装	226
✓	605	29.19	ar-decp.h	自動運転減速度パラメータ: ヘッダ	21
✓	605	29.20	ar-decp.c	自動運転減速度パラメータ: 実装	18
✓	606	29.21	ar-motor.h	自動運転モーター制御: ヘッダ	62
✓	609	29.22	ar-motor.c	自動運転モーター制御: 実装	224
✓	613	29.23	ar-est.h	自動運転減速度予測: ヘッダ	7
✓	614	29.24	ar-est.c	自動運転減速度予測: 実装	114
✓	615	29.25	ar-stay.h	自動運転停車関数: ヘッダ	13
✓	616	29.26	ar-stay.c	自動運転停車関数: 実装	40
	189	10.1	pp.h	パワーパック関数宣言: ヘッダ	46
	172	9.2	pp-const.h	パワーパック定数: ヘッダ	17
	191	10.3	pp-misc.c	自動加減速制御: 実装	111
	196	10.6	pp-sb.h	低速ブースト機能: ヘッダ	8
	198	10.7	pp-sb.c	低速ブースト機能: 実装	171
	129	6.1	incr-lin.h	線形増加アルゴリズム: ヘッダ	9
	130	6.2	incr-lin.c	線形増加アルゴリズム: 実装	78
	394	23.3	i2c.h	I^2C インターフェース関数: ヘッダ	25
	394	23.4	i2c.c	I^2C インターフェース関数: 実装	111
	396	23.5	i2c-mcp23017.h	MCP23017 アクセス関数: ヘッダ	53
	397	23.6	i2c-mcp23017.c	MCP23017 アクセス関数: 実装	109
✓	574	29.1	io.h	I/O ピン割付: ヘッダ	81
✓	617	29.27	mcr.h	MCR・周辺モジュール: ヘッダ	8
✓	617	29.28	mcr.c	MCR・周辺モジュール: 実装	110
	175	9.6	mcr-motor.h	モーター制御周辺モジュール: ヘッダ	25
	177	9.7	mcr-motor.c	モーター制御周辺モジュール: 実装	98
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76

計 3256 行 (うち初出 2310 行, 既出 946 行)

ソースファイル io.h (I/O ピン割付: ヘッダ)

回路図に合わせて各種マクロ定義を行います.

ソースファイル番号 29.1: io.h

```
#if !defined(__IO_H__)
#define __IO_H__

/* 設定VR ADC: 手動走行速度: ANA0 */
#define ADC_CH_SPEED              0x00
/* 設定VR ADC - 走行最大電圧設定: ANA1 */
#define ADC_CH_VMAX               0x01
```

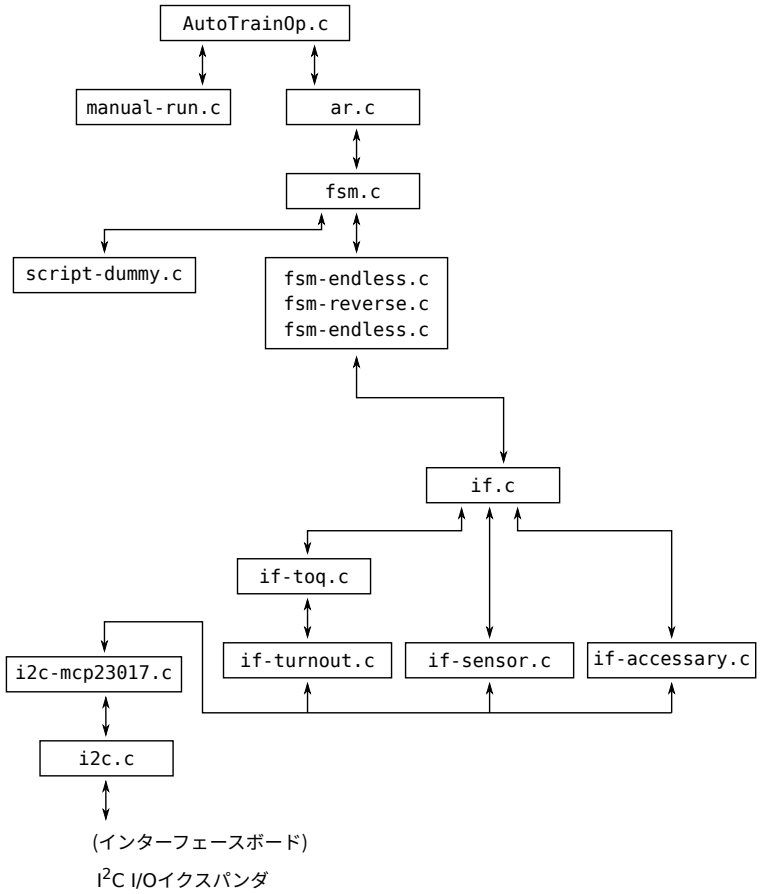


図 29.24: ファームウエアにおける主要ソースファイル間の関係

```
8  | /* 設定VR ADC - 走行最小電圧設定: ANA2 */
9  | #define ADC_CH_VSLOW              0x02
10 | /* 設定VR ADC - 加速度設定: ANA3 */
11 | #define ADC_CH_ACCEL              0x03
12 | /* 設定VR ADC - 減速度設定: ANA4 */
13 | #define ADC_CH_DECEL              0x04
14 | /* 設定VR ADC - 低速ブースト周波数設定: ANA5 */
15 | #define ADC_CH_SBFREQ             0x05
16 | /* 設定VR ADC - 低速ブーストパルス幅設定: ANA7 */
17 | #define ADC_CH_SBPWDT             0x07
18 | /* 設定VR ADC - 停車時間設定: ANA6 */
19 | #define ADC_CH_STAYTIM            0x06
20 | /* 設定VR ADC - 常点灯設定: ANC0 */
21 | #define ADC_CH_LIGHT              0x10
22 |
23 | /* 走行パターン設定DIPスイッチ1: RB1 */
24 | #define SW_OPPAT_1()              (RB1 == 0)
25 | /* 走行パターン設定DIPスイッチ2: RB2 */
26 | #define SW_OPPAT_2()              (RB2 == 0)
27 | /* 走行パターン設定DIPスイッチ3: RB3 */
28 | #define SW_OPPAT_3()              (RB3 == 0)
29 | /* 走行パターン設定DIPスイッチ4: RB4 */
30 | #define SW_OPPAT_4()              (RB4 == 0)
31 |
32 | /* LED IDLE (アイドル時): RE0 */
33 | #define LED_IDLE_ON()             (LATE0 = 1)
34 | #define LED_IDLE_OFF()            (LATE0 = 0)
35 | #define LED_IDLE_TOGGLE()         (LATE0 = ~LATE0)
```

```
36 /* LED RUN (動作時): RE1 */
37 #define LED_RUN_ON()              (LATE1 = 1)
38 #define LED_RUN_OFF()             (LATE1 = 0)
39 #define LED_RUN_TOGGLE()          (LATE1 = ~LATE1)
40 /* LED センサ検出表示 RE2 */
41 #define LED_SENS_ON()             (LATE2 = 1)
42 #define LED_SENS_OFF()            (LATE2 = 0)
43 #define LED_SENS_TOGGLE()         (LATE2 = ~LATE2)
44
45 /* 操作モードスイッチ(自動運転/手動運転): RD4 */
46 #define SW_MODE_AUTO()            (RD4 == 0)
47 /* 緊急停止ボタン: RB0 */
48 #define SW_EMG_STOP()             (RB0 == 0)
49 /* 自動運転開始スイッチ: RD5 */
50 #define SW_START()                (RD5 == 0)
51 /* 自動運転終了スイッチ: RD6 */
52 #define SW_END()                  (RD6 == 0)
53 /* 低速ブースト有効/禁止選択スイッチ: RD7 */
54 #define SW_SBEN()                 (RD7 == 0)
55
56 /* (手動運転時)走行方向スイッチ 通常 RC2 */
57 #define SW_DIR1F()                (RC2 == 0)
58 /* (手動運転時)走行方向スイッチ 反転 RC3 */
59 #define SW_DIR1R()                (RC3 == 0)
60 /* (手動運転時)走行方向スイッチ 通常 RD0 */
61 #define SW_DIR2F()                (RD0 == 0)
62 /* (手動運転時)ポイント 定位 RD1 */
63 #define SW_TURNOUT()              (RD1 == 0)
64
65 /* 走行方向出力1 通常 RD2 */
66 #define TDIR1_NORM()              (LATD2 = 0)
67 /* 走行方向出力1 反転 RD2 */
68 #define TDIR1_REV()               (LATD2 = 1)
69 /* 走行方向出力2 通常 RD3 */
70 #define TDIR2_NORM()              (LATD3 = 0)
71 /* 走行方向出力2 反転 RD3 */
72 #define TDIR2_REV()               (LATD3 = 1)
73
74 /* 内蔵アクセサリ出力1 RC6 */
75 #define ACC_1_ON()                (LATC6 = 1)
76 #define ACC_1_OFF()               (LATC6 = 0)
77 /* 内蔵アクセサリ出力2 RC7 */
78 #define ACC_2_ON()                (LATC7 = 1)
79 #define ACC_2_OFF()               (LATC7 = 0)
80
81 #endif /*__IO_H__*/
```

ソースファイル AutoTrainOp.c (主ソースコード)

関数 main() が動作の主となる部分です. 初期化を終えた後は DIP スイッチを読み, 表 29.1 で示す動作を行います.

- 事前組み込み済み自動運転制御プログラムの場合
  主ループに入り, 自動走行・手動走行を行います.
- インターフェースボードのテストの場合
  このソースファイル中に定義されている関数 test_if_board() を呼び出します. これは後で説明します.
- EEPROM 自動運転スクリプトの場合
  関数 script_init() を呼び出してスクリプト実行機構を初期化しますが, 本章ではスクリプト実行機構はまだ実現せず第 30 章で実現します. 本章ではエラー表示を行うダミーの関数をソースファイル script-dummy.c の中で定義します.

関数 main() のループ部分の動作は単純で, トグルスイッチ MODE の状態に応じて手動運転モードと自動運転モードそれぞれに対応する関数を呼び出すだけです. トグルスイッチ MODE が切り替えられてモード変更が行われるとそれら関数は実行を終えて関数 main() に戻ってきます.

- 関数 pp_home_position()
  手動運転モード時に切り替わったときのラッシュ防止機能を実現します. すでに紹介したパワーパックでの場合と同様です.
- 関数 op_pattern_read()
  DIP スイッチの設定を読んで変数 op_pattern に設定内容を書き込みます.
- 関数 test_if_board()
  インターフェースボードの動作テストを行う関数です. インターフェースボードを製作したらこの機能を利用して動作確認をして下さい. この関数は以下の動作を永久に繰り返します.
  - すべてのポイント切替: ポイント番号 1 から順に反位・定位の切り替えを行います.
  - すべてのアクセサリ制御出力のオン・オフ: アクセサリ番号 1 から順に 200 ミリ秒間隔でオン・オフを行います.

ソースファイル番号 29.2: AutoTrainOp.c

```
1  /*************************************************
2   * プログラマブル自動列車運転システム / PIC16F18877
3   * by Hirotsugu Kakugawa
4   *************************************************/
5  #include <xc.h>
6  #include <stdint.h>
7  #include "mcr.h"
8  #include "mcr-motor.h"
9  #include "io.h"
10 #include "pp.h"
11 #include "pp-const.h"
12 #include "pp-sb.h"
13 #include "ar.h"
14
15 enum op_pattern_e op_pattern = OPPAT_UNDEF;
16 extern void script_init(void);
17 extern void manual_run_start(void);
18 static void op_pattern_read(void);
19 static void test_if_board(void);
20
21 /*-----------------------------------------------------
22  * MAIN
23  *-----------------------------------------------------*/
24 void main(void)
25 {
26   /* システム初期化 */
27   System_Init();
28   /* ウオッチドッグタイマ: 256 [mSec] */
29   WDTCON0bits.WDTPS = 0x08; /* 0b01000 */
30   CLRWDT();
31   /* モータ出力初期化, PWM動作開始 */
32   MOTOR_DIR_NONE();
33   MOTOR_PWM_START();
34   /* 低速ブースト初期化 */
35   pp_sb_init();
36   pp_sb_duty_set(0);
37   /* インターフェースモジュール初期化 */
38   pp_delay_ms(100);
39   CLRWDT();
40   if_init();
41   CLRWDT();
42   /* 走行パターン */
```

```
43    op_pattern_read();
44    /* 自動運転初期化 */
45    auto_run_init();
46    /* 割り込み許可 */
47    GIE = 1;
48    PEIE = 1;
49    /* LED */
50    LED_RUN_OFF();
51    LED_IDLE_OFF();
52    LED_SENS_OFF();
53    /* DIP SW */
54    switch (op_pattern) {
55    default:
56      break;
57    case OPPAT_TEST_IF_BOARD: /* インターフェースボードのテスト */
58      test_if_board();
59      break;
60    case OPPAT_EEPROMSCRIPT:
61      script_init(); /* スクリプト実行機構初期化 */
62      break;
63    }
64    /* 主ループ */
65    for (;;) {
66      CLRWDT();
67      /* 停止 */
68      pp_sb_duty_set(0);
69      /* モーター＆常点灯Off */
70      ar_motor_off();
71      /* アクセサリ制御出力を全てオフ */
72      if_accessary_off_all();
73      /* 全ポイント切替完了待ち */
74      while (if_toq_next(RUN_INTERVAL)) {
75        pp_delay_ms(RUN_INTERVAL);
76        CLRWDT();
77      }
78      /* 操作モード選択 */
79      if (SW_MODE_AUTO()) { /* 自動運転 */
80        auto_run_start();
81      } else {              /* 手動運転 */
82        manual_run_start();
83      }
84      pp_delay_ms(100);
85    }
86  }
87
88  /*-----------------------------------------------------
89   * 速度ボリュームの定位置復帰を待機
90   *-----------------------------------------------------*/
91  void pp_home_position(void)
92  {
93    while (!SW_MODE_AUTO()) { /* 手動 */
94      CLRWDT();
95      /* ボリューム読み込み */
96      pp_param_read();
97      if (p_speed == 0) { /* 定位置へ復帰 */
98        break;
99      }
100     /* 未復帰: LEDを点滅 */
101     LED_RUN_TOGGLE();
102     pp_delay_ms(T_LED_WARNING);
103   }
104   LED_RUN_OFF();
105 }
106
107 /*-----------------------------------------------------
108  * 走行パターン（設定 DIP-SW より）
109  *-----------------------------------------------------*/
110 static void op_pattern_read(void)
111 {
112   static const enum op_pattern_e oppat_table[16] = {
```

```
113       /* DIP-SW設定                  (SW1,SW2,SW3,SW4) */
114       OPPAT_SHUTTLE,        /*  0 (OFF,OFF,OFF,OFF) */
115       OPPAT_ENDLESS,        /*  1 (ON, OFF,OFF,OFF) */
116       OPPAT_REVERSE,        /*  2 (OFF,ON, OFF,OFF) */
117       OPPAT_UNDEF,          /*  3 (ON, ON, OFF,OFF) */
118       OPPAT_UNDEF,          /*  4 (OFF,OFF,ON, OFF) */
119       OPPAT_UNDEF,          /*  5 (ON, OFF,ON, OFF) */
120       OPPAT_UNDEF,          /*  6 (OFF,ON, ON, OFF) */
121       OPPAT_UNDEF,          /*  7 (ON, ON, ON, OFF) */
122       OPPAT_UNDEF,          /*  8 (OFF,OFF,OFF,ON ) */
123       OPPAT_UNDEF,          /*  9 (ON, OFF,OFF,ON ) */
124       OPPAT_UNDEF,          /* 10 (OFF,ON, OFF,ON ) */
125       OPPAT_UNDEF,          /* 11 (ON, ON, OFF,ON ) */
126       OPPAT_UNDEF,          /* 12 (OFF,OFF,ON, ON ) */
127       OPPAT_UNDEF,          /* 13 (ON, OFF,ON, ON ) */
128       OPPAT_TEST_IF_BOARD, /* 14 (OFF,ON, ON, ON ) */
129       OPPAT_EEPROMSCRIPT    /* 15 (ON, ON, ON, ON ) */
130     };
131     int8_t s1 = SW_OPPAT_1();
132     int8_t s2 = SW_OPPAT_2();
133     int8_t s3 = SW_OPPAT_3();
134     int8_t s4 = SW_OPPAT_4();
135     int8_t n = (s1 + (s2 << 1) + (s3 << 2) + (s4 << 3)) & 0x0F;
136     op_pattern = oppat_table[(unsigned)n];
137   }
138
139   /* インターフェースボードのテスト */
140   static void test_if_board(void)
141   {
142     for (;;) {
143       CLRWDT();
144       /* ポイント制御出力 */
145       uint8_t p = 0;
146       for (p = 1; p <= IF_MAX_TURNOUTS; p++) {
147         /* 反位 */
148         if_toq_set_reverse(p);
149         while (!if_toq_is_done()) {
150           pp_delay_ms(RUN_INTERVAL);
151           CLRWDT();
152           if_toq_next(RUN_INTERVAL);
153         }
154         /* 定位 */
155         if_toq_set_normal(p);
156         while (!if_toq_is_done()) {
157           pp_delay_ms(RUN_INTERVAL);
158           CLRWDT();
159           if_toq_next(RUN_INTERVAL);
160         }
161       }
162       /* アクセサリ制御出力 */
163       uint8_t ac = 0;
164       for (ac = 1; ac <= IF_MAX_ACCESSARIES; ac++) {
165         /* オン */
166         if (if_accessary_on(ac)) {
167           pp_delay_ms(200);
168         }
169         /* オフ */
170         if (if_accessary_off(ac)) {
171           pp_delay_ms(200);
172         }
173       }
174     }
175   }
```

関数 test_if_board() を読むことで, 本ファームウエアにおいてポイント切替やアクセサリ制御出力のオン・オフを行うプログラムの書き方が分かります. ポイント切替は以下の関数呼び出しと手順で行います.

- 関数 if_toq_normal(p), 関数 if_toq_reverse(p)
  この関数呼び出しにより, 指定したポイントを定位 (normal) あるいは反位 (reverse) に切り替える要求を

表 29.8: 致命エラー時の LED の点滅パターン (本章の範囲のみ)

点滅回数	エラー原因	備考
7	未実装機能	DIP スイッチ設定誤り
9	インターフェースボード無装着	インターフェースボードが全く接続されていない
10	内部エラー	ファームウエアバグの検出

行います. 引数の p はポイント番号です. ここで注意が必要なのは, この関数では切り替えの要求をするだけで, 実際に切り替えを開始したり完了を待機したりはしないという点です. これは動作の完了を待たずに複数の切り替え要求を連続して行えるようにするためです.

- 関数 `if_toq_is_done()`
  ポイント切替の要求がすべて動作完了したら真を返し, そうでなければ偽を返します.
- 関数 `if_toq_next(t)`
  要求された切替動作を実際に実行します. 引数により前回の呼び出しから時刻は t ミリ秒経過したことを知らせて, 切替を実行する時間間隔を確保します.

これらの関数を使用する目的は, 全ポイントが一斉切替とならないようにひとつひとつ順番に切替を実行することです. 第 28 章で紹介した EEPROM プログラム型ポイント一括切替器でのファームウエアでは, ソースファイル `action.c` (532 ページ) にて `__delay_ms()` 関数を使用して切替時間の間隔を確保していました. しかし本章のファームウエアでは同じ方法をとるわけにはいきません. なぜなら自動運転による模型車両の走行をしながらポイント切替をする必要があり, もし `__delay_ms()` 関数を使用すればその間は走行速度の更新や車両センサの監視が行えず, 正常に自動運転が行えないからです.

アクセサリ制御出力のオン・オフは以下の関数呼び出しと手順で行います. ポイント切替の場合と異なり, オン・オフの動作は即座に完了します.

- `if_accessary_on(ac)`
  指定したアクセサリ制御出力をオンにします. 引数の ac はアクセサリ番号です.
- `if_accessary_off(ac)`
  指定したアクセサリ制御出力をオフにします. 引数の ac はアクセサリ番号です.

ソースファイル fatal.h (エラー表示: ヘッダ)

このファイルではエラーコードの定義と関数のプロトタイプ宣言を行います.

ソースファイル番号 29.3: fatal.h

```
1  #if !defined(__FATAL_H__)
2  #define __FATAL_H__
3
4  /* エラーコード */
5  #define AR_ERROR_OUT_OF_RANGE      1  /* 値が範囲外 */
6  #define AR_ERROR_UNKNOW_DIRECTIVE  2  /* 存在しない指示子 */
7  #define AR_ERROR_UNKNOW_STATEMENT  3  /* 存在しない文 */
8  #define AR_ERROR_SYNTAX            4  /* 構文エラー */
9  #define AR_ERROR_UNDEFINED_LABEL   5  /* 未定義ラベルの使用 */
10 #define AR_ERROR_UNEXPECTED_END    6  /* 実行中にEND文へ到達 */
11 #define AR_ERROR_UNSUPPORTED       7  /* 未実装機能 */
12 #define AR_ERROR_NO_EEPROM         8  /* EEPROM 無 */
13 #define AR_ERROR_NO_IFBOARD        9  /* インターフェースボード無 */
```

```
14 #define AR_ERROR_INTERNAL        10  /* 内部エラー */
15
16 extern void fatal(uint8_t err);
17 extern void fatal_s(uint8_t err, uint16_t lineno,
18                     const char *s);
19 extern void fatal_sc(uint8_t err, uint16_t lineno,
20                      const char *s, char ch);
21 extern void fatal_si(uint8_t err, uint16_t lineno,
22                      const char *s, uint16_t i);
23
24 #endif /*__FATAL_H__*/
```

ソースファイル fatal.c (エラー表示: 実装)

致命的なエラーが生じた場合には自動運転の動作は一切停止してエラー表示を行います. 具体的には 3 つの LED (RUN, IDLE, SENS) の点滅を繰り返します. 本章のファームウエアにおいて発生しうるエラーの一覧表を表 29.8 に示します. エラーが発生した時は 3 つの LED が同時に短時間の点滅をします. 点滅の回数はエラー原因によって異なり, その回数は表で示す通りです. 短時間の点滅を繰り返した後は 2 秒間の空白時間をおいて, 再び短時間の点滅を繰り返します.

関数 fatal() でエラー表示を行います. これ以外にも関数 fatal_s(), fatal_sc(), fatal_si() が定義されていますが, これらの中では関数 fatal() を呼び出すだけとしています. これらの関数への引数にはエラーコード以外にもエラー原因に関する付加的な情報を持たせていますが, 本装置では表示能力がないために付加的な情報は使用せずにエラーコードだけを LED の点滅に使用します. これらの関数は第 32 章で紹介する自動運転スクリプトのエミュレータソフトウエア emuscript のための準備で, 詳しくは後述します.

ソースファイル番号 29.4: fatal.c

```
1  #include <xc.h>
2  #include <stdint.h>
3  #include "mcr.h"
4  #include "io.h"
5  #include "pp.h"
6  #include "fatal.h"
7
8  /*-------------------------------------------------------
9   * エラー表示
10  *   短時間点灯をエラーコード値の回数行い, 長時間消灯を行う
11  *-------------------------------------------------------*/
12 void fatal(uint8_t err)
13 {
14   uint8_t i = 0;
15   for (;;) { /* 繰返し */
16     /* エラーコード値の回数だけ短時間点灯 */
17     for (i = err; i != 0; i--) {
18       LED_RUN_ON();
19       LED_IDLE_ON();
20       LED_SENS_ON();
21       pp_delay_ms(100);
22       LED_RUN_OFF();
23       LED_IDLE_OFF();
24       LED_SENS_OFF();
25       pp_delay_ms(300);
26     }
27     /* 消灯 (2秒) */
28     pp_delay_ms(2000);
29   }
30 }
31
32 void fatal_s(uint8_t err, uint16_t lineno,
33              const char *s)
34 {
35   fatal(err);
36 }
```

```
37
38 void fatal_sc(uint8_t err, uint16_t lineno,
39                const char *s, char ch)
40 {
41   fatal(err);
42 }
43
44 void fatal_si(uint8_t err, uint16_t lineno,
45                const char *s, uint16_t i)
46 {
47   fatal(err);
48 }
```

ソースファイル manual-run.c (手動運転: 実装)

手動運転モードの場合の動作を実現します. 基本的な構造はパワーパックと同じですが以下の点で異なります.

- 最高速度
  スイッチ MODE により手動運転モードへと切り替えられた時に同時に END スイッチが押されていれば, 最高速度は可変抵抗器 VSLOW による設定を使用します. そうでなければ最高速度は可変抵抗器 VMAX による設定を使用します.
- 手動運転モードの終了
  スイッチ MODE が手動運転モードから自動運転モードへと切り替えられると手動運転モードの終了します.
- 車両センサ検出モニタ
  車両センサのうちひとつでも検出があれば LED を点灯します.
- 走行方向
  これまでのパワーパックではモータドライバ IC への信号により走行方向の逆転を行っていましたが, 本章のファームウエアではリレーの切り替えにより走行方向の逆転を行います. 2 系統のフィーダ出力 F1 と F2 があり, スイッチ DIR1 と DIR2 の切り替えに従って対応するリレーを切り替えます.
- ポイント切替
  ひとつのスイッチの操作ですべてのポイント切替をひとつひとつ順番に行います.

ソースファイル番号 29.5: manual-run.c

```
1  /*-------------------------------------------------------
2   * 手動運転モード
3   *-------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "mcr-motor.h"
8  #include "io.h"
9  #include "pp.h"
10 #include "pp-const.h"
11 #include "pp-sb.h"
12 #include "ar.h"
13
14 /* 最高速度として用いる可変抵抗 */
15 enum max_src_e {
16   MAX_SRC_VMAX = 0,
17   MAX_SRC_VSLOW,
18 };
19
20 /* 手動運転(自動加減速なし) */
21 void manual_run_start(void)
```

```
{
  /* 最高速度として用いる可変抵抗を選択 */
  enum max_src_e max_src = MAX_SRC_VMAX;
  if (!SW_END()) { /* 切替時に END SW 解放: VMAX を使用 */
    max_src = MAX_SRC_VMAX;
  } else {         /* 切替時に END SW 押下: VSLOW を使用 */
    max_src = MAX_SRC_VSLOW;
  }
  /* スピードボリュームの定位置復帰を待機 */
  pp_home_position();
  /* LED */
  LED_RUN_ON();
  if (max_src == MAX_SRC_VMAX) {
    LED_IDLE_OFF();
  } else {
    LED_IDLE_ON();
  }
  LED_SENS_OFF();
  /* ポイント切替方向 */
  enum if_turnout_dir_e d_curr = TURNOUT_DIR_NONE;
  enum if_turnout_dir_e d_next = TURNOUT_DIR_NONE;
  /* 停止 */
  MOTOR_DIR_NONE();
  pp_sb_duty_set(0);
  /* 主ループ */
  for (;;) {
    pp_delay_ms(RUN_INTERVAL);
    CLRWDT();
    /* モータードライバ LSI TB6643KQ 制御 (過電流検出の解除) */
    if (SW_EMG_STOP()) { /* 解除要求の発生 */
      extern void MotorDriver_TB6643KQ_Reset(void);
      MotorDriver_TB6643KQ_Reset();
    }
    /* 非常停止 */
    if (SW_EMG_STOP()) {
      break;
    }
    /* 現在モード終了 */
    if (SW_MODE_AUTO()) {
      break;
    }
    /* センサ検出 LED 表示 */
    if_sensor_read();
    if (if_sensor_detected()) {
      LED_SENS_ON();
    } else {
      LED_SENS_OFF();
    }
    /* 走行方向を出力(フィーダ1) */
    switch (pp_dir_read()) {
    case DIR_FWD: /* 順方向 */
      ar_motor_on(); /* モータ＆常点灯On  */
      ar_track1_dir_set(AR_TRACK_DIR_NORMAL);
      break;
    case DIR_REV: /* 逆方向 */
      ar_motor_on(); /* モータ＆常点灯On  */
      ar_track1_dir_set(AR_TRACK_DIR_REVERSE);
      break;
    default:
      ar_motor_off(); /* モータ＆常点灯Off  */
      break;
    }
    /* 走行方向を出力(フィーダ2) */
    if (SW_DIR2F()) { /* 順方向 */
      ar_track2_dir_set(AR_TRACK_DIR_NORMAL);
    } else {          /* 逆方向 */
      ar_track2_dir_set(AR_TRACK_DIR_REVERSE);
    }
    /* ポイント切替 */
```

```
91      if (!if_toq_next(RUN_INTERVAL)) {
92        if (SW_TURNOUT()) {
93          d_next = TURNOUT_DIR_NORMAL;  /* 定位 */
94        } else {
95          d_next = TURNOUT_DIR_REVERSE; /* 反位 */
96        }
97        if (d_curr != d_next) { /* 切替 */
98          d_curr = d_next;
99          if (d_curr == TURNOUT_DIR_NORMAL) {
100           if_toq_set_all_normal();
101         } else {
102           if_toq_set_all_reverse();
103         }
104       }
105     }
106     /* パラメータの読込 */
107     ar_param_read();
108     /* 現在走行速度 */
109     uint16_t speed = p_speed;
110     /* 走行速度を変換して，モーターPWMデューティー比を得る */
111     uint16_t md = 0;
112     if (max_src == MAX_SRC_VMAX) {
113       md = PP_CONV_SPEED_TO_MDUTY(p_speed, p_vmax);
114     } else {
115       md = PP_CONV_SPEED_TO_MDUTY(p_speed, p_vslow);
116     }
117     /* モーターPWMデューティー比を出力 */
118     pp_sb_duty_set(md);
119   }
120 }
```

ソースファイル fsm.h (自動運転有限状態機械: ヘッダ)

自動運転のための有限状態機械を実現する関数のプロトタイプ宣言を行います.

ソースファイル番号 29.6: fsm.h

```
1  #if !defined(__AR_FSM_H__)
2  #define __AR_FSM_H__
3
4  extern enum ar_state_e fsm_next(enum ar_state_e ar_state);
5  extern enum ar_state_e fsm_shuttle_next(enum ar_state_e ar_state);
6  extern enum ar_state_e fsm_endless_next(enum ar_state_e ar_state);
7  extern enum ar_state_e fsm_reverse_next(enum ar_state_e ar_state);
8  extern enum ar_state_e fsm_script_next(enum ar_state_e ar_state);
9
10 #endif /*__AR_FSM_H__*/
```

ソースファイル fsm.c (自動運転有限状態機械: 実装)

自動運転のための有限状態機械を実現する関数です. しかしここでは DIP スイッチの設定に従って別の関数を呼び出すだけです. 具体的には図 29.3 で示したトラックプラン 3 種に対する関数, および自動運転スクリプトを有限状態機械として実現する関数です.

ソースファイル番号 29.7: fsm.c

```
1  /*-----------------------------------------------------
2   * 自動運転状態機械
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "pp.h"
7  #include "ar.h"
8  #include "fsm.h"
9
10 enum ar_state_e fsm_next(enum ar_state_e ar_state)
```

```
{
  enum ar_state_e next = AR_STATE_INIT;
  switch (op_pattern) {
  case OPPAT_SHUTTLE:
    next = fsm_shuttle_next(ar_state);
    break;
  case OPPAT_ENDLESS:
    next = fsm_endless_next(ar_state);
    break;
  case OPPAT_REVERSE:
    next = fsm_reverse_next(ar_state);
    break;
  case OPPAT_EEPROMSCRIPT:
    next = fsm_script_next(ar_state);
    break;
  default:
    break;
  }
  return next;
}
```

ソースファイル fsm-shuttle.c (自動運転有限状態機械 (単線): 実装)

単線トラックプランに対する自動運転有限状態機械を実現します. トラックプランは図 29.3a で, 状態遷移図は図 29.21 で, それぞれ示したものです.

ソースファイル番号 29.8: fsm-shuttle.c

```
/*-------------------------------------------------------
 * トラックプラン: 単線往復
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "pp.h"
#include "pp-const.h"
#include "ar.h"
#include "ar-motor.h"
#include "fsm.h"

/*
 *    北
 *  西＋東
 *    南
 *
 *        西駅         Ｆ１             東駅
 *    ＝＃＝＝＃＝＝＝↑＝＝＝＝＝＝＝＃＝＝＃＝
 *      Ｓ４   Ｓ３                    Ｓ２   Ｓ１
 *  （Ｆ：フィーダ、Ｓ：センサ）
 *
 * 動作シーケンスの概要(以下の繰り返し):
 *   最初に列車は西駅に停車
 *   1. 西駅を出発し東駅に向かう
 *   2. 東駅で停車
 *   3. 東駅を出発し西駅に向かう
 *   4. 西駅で停車
 */
/* 状態 */
#define AR_STATE_GE1       10  /* 西駅→東駅, 走行開始 */
#define AR_STATE_GE2       11  /* 西駅→東駅, 走行継続 */
#define AR_STATE_SE1       12  /* 東駅, 停車開始 */
#define AR_STATE_SE2       13  /* 東駅, 停車継続 */
#define AR_STATE_GW1       14  /* 東駅→西駅, 走行開始 */
#define AR_STATE_GW2       15  /* 東駅→西駅, 走行継続 */
#define AR_STATE_SW1       16  /* 西駅, 停車開始 */
#define AR_STATE_SW2       17  /* 西駅, 停車継続 */
/* 減速パラメータ番号 */
#define AR_DECP_ID_EAST     1  /* 東駅用 */
```

```
40  #define AR_DECP_ID_WEST    2  /* 西駅用 */
41  /* センサ番号 */
42  #define AR_SN_EAST_STOP    1  /* S1: 東駅, 停止用 */
43  #define AR_SN_EAST_BRAKE   2  /* S2: 東駅, 制動用 */
44  #define AR_SN_WEST_BRAKE   3  /* S3: 西駅, 制動用 */
45  #define AR_SN_WEST_STOP    4  /* S4: 西駅, 停止用 */
46
47  /* 走行制御状態機械 */
48  enum ar_state_e fsm_shuttle_next(enum ar_state_e ar_state)
49  {
50    /* 初期状態: 西駅に停車 */
51    switch ((uint8_t)ar_state) {
52    default:
53      ar_state = AR_STATE_INIT;
54      break;
55    case AR_STATE_INIT:
56      ar_track1_dir_set(AR_TRACK_DIR_NORMAL); /* 東行 */
57      ar_motor_on(); /* モータ&常点灯On */
58      ar_state = AR_STATE_GE1;
59      break;
60    case AR_STATE_GE1: /* 西駅→東駅 */
61      ar_motor_run_until(AR_SPEEDSTEP_MAX, AR_DECP_ID_EAST,
62                         AR_SN_EAST_BRAKE, AR_SN_EAST_STOP);
63      ar_state = AR_STATE_GE2;
64      break;
65    case AR_STATE_GE2: /* 走行継続 */
66      if (!ar_motor_next()) {
67        ar_state = AR_STATE_SE1;
68      }
69      break;
70    case AR_STATE_SE1: /* 東駅, 停車 */
71      ar_track1_dir_set(AR_TRACK_DIR_REVERSE); /* 西行 */
72      ar_stay_start(AR_STAYTIME_DEFAULT);
73      ar_state = AR_STATE_SE2;
74      break;
75    case AR_STATE_SE2: /* 停車継続 */
76      if (!ar_stay_next()) {
77        ar_state = AR_STATE_GW1;
78      }
79      break;
80    case AR_STATE_GW1: /* 東駅→西駅 */
81      ar_motor_run_until(AR_SPEEDSTEP_MAX, AR_DECP_ID_WEST,
82                         AR_SN_WEST_BRAKE, AR_SN_WEST_STOP);
83      ar_state = AR_STATE_GW2;
84      break;
85    case AR_STATE_GW2: /* 走行継続 */
86      if (!ar_motor_next()) {
87        ar_state = AR_STATE_HOME;
88      }
89      break;
90    case AR_STATE_HOME:   /* 定位置(西駅) */
91      ar_state = AR_STATE_SW1;
92      break;
93    case AR_STATE_SW1: /* 西駅, 停車 */
94      ar_track1_dir_set(AR_TRACK_DIR_NORMAL); /* 東行 */
95      ar_stay_start(AR_STAYTIME_DEFAULT);
96      ar_state = AR_STATE_SW2;
97      break;
98    case AR_STATE_SW2: /* 停車継続 */
99      if (!ar_stay_next()) {
100       ar_state = AR_STATE_GE1;
101     }
102     break;
103   }
104   return ar_state;
105 }
```

このソースファイルにおけるプログラムコードは図 29.21 に示した状態遷移図がそのまま記述されているこ

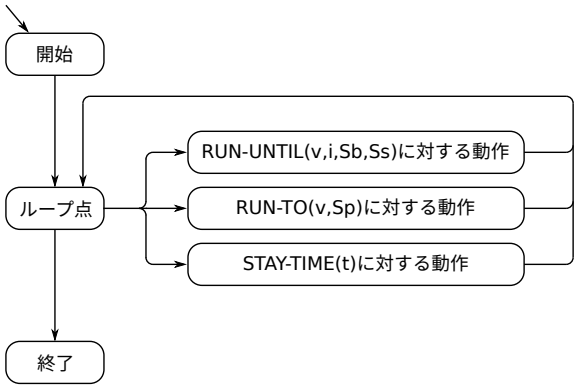


図 29.25: 自動運転パターン実行の抽象化状態遷移図

とが見て取れると思います. すなわち状態遷移図からプログラムコードへの変換を機械的に行っており, 自動運転パターンの設計・理解・実装は極めて簡単に行えることが分かります. 新たなトラックプランに対する自動運転の実現はこのソースファイルと同様なものを作成して, ソースファイル `AutoTrainOp.c` 中の DIP スイッチ設定の部分とソースファイル `fsm.c` 中の関数呼び出し部分に登録するだけです.

`fsm-shuttle.c` を読むと, C 言語で記述されている動作の列をデータ列として表現・記述してそのデータを逐次解釈・実行する機構が容易に作成できそうだと直感的に感じるのではないかと思います. 以下にこれについて説明します. 図 29.25 に `fsm-shuttle.c` の実行を抽象化した状態遷移図を示します. この抽象化状態遷移図は本章のファームウエアで共通な自動運転パターンの設計フレームワークで, この後に示す `fsm-endless.c` や `fsm-reverse.c` もこの形をしています. これはまさにコンピュータのプロセッサがコンピュータプログラムを実行する際に, メモリからの命令読み出しと実行の繰り返しと同一です. 自動運転の基本操作をデータ列としてメモリに格納しておき, メモリからの基本操作の読み出しと対応する動作の実行を繰り返すことで自動運転スクリプトの実行が実現できます. 第 30 章では独自のスクリプト言語により動作をデータとして記述し, 実行を行う機構を実現します.

ソースファイル `fsm-endless.c` (自動運転有限状態機械 (エンドレス): 実装)

エンドストラックプランに対する自動運転有限状態機械を実現します. トラックプランは図 29.3b で, 状態遷移図は図 29.22 で, それぞれ示したものです.

ソースファイル番号 29.9: `fsm-endless.c`

```
1  /*-------------------------------------------------------
2   * トラックプラン: 単線エンドレス
3   *-------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "pp.h"
7  #include "pp-const.h"
8  #include "ar.h"
9  #include "ar-motor.h"
10 #include "fsm.h"
11
12 /*
13  *     北
14  *   西＋東
15  *     南
16  *          ＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝
17  *     ／／                                  ＼＼
18  *    | |                                    | |
```

```
 *     ＼＼   Ｆ１       駅                        ／／
 *         ＝↑＝＝＝＃＝＝＝＝＃＝＝＝＝＝＝
 *                  Ｓ２         Ｓ１
 *   （Ｆ：フィーダ、Ｓ：センサ）
 *
 * 動作シーケンスの概要(以下の繰り返し):
 *   1. 東に向かい駅を出発
 *   2. 一周してきて駅で停車
 */
/* 状態 */
#define AR_STATE_RUN1     10  /* 東に向かい駅を出発, 走行開始 */
#define AR_STATE_RUN2     11  /* 走行継続 */
#define AR_STATE_STA1     12  /* 駅, 停車 */
#define AR_STATE_STA2     13  /* 駅, 停車継続 */
/* 減速パラメータ番号 */
#define AR_DECP_ID_STA     1  /* 駅用 */
/* センサ番号 */
#define AR_SN_STA_STOP     1  /* S1: 駅, 停止用 */
#define AR_SN_STA_BRAKE    2  /* S2: 駅, 制動用 */

/* 走行制御状態機械 */
enum ar_state_e fsm_endless_next(enum ar_state_e ar_state)
{
  /* 初期状態: 駅に停車 */
  switch ((uint8_t)ar_state) {
  default:
    ar_state = AR_STATE_INIT;
    break;
  case AR_STATE_INIT:
    ar_track1_dir_set(AR_TRACK_DIR_NORMAL); /* 東行 */
    ar_motor_on(); /* モータ＆常点灯On */
    ar_state = AR_STATE_RUN1;
    break;
  case AR_STATE_RUN1: /* 駅を出発 */
    ar_motor_run_until(AR_SPEEDSTEP_MAX, AR_DECP_ID_STA,
                       AR_SN_STA_BRAKE, AR_SN_STA_STOP);
    ar_state = AR_STATE_RUN2;
    break;
  case AR_STATE_RUN2: /* 走行継続 */
    if (!ar_motor_next()) {
      ar_state = AR_STATE_HOME;
    }
    break;
  case AR_STATE_HOME:   /* 定位置(駅) */
    ar_state = AR_STATE_STA1;
    break;
  case AR_STATE_STA1: /* 駅, 停車 */
    ar_stay_start(AR_STAYTIME_DEFAULT);
    ar_state = AR_STATE_STA2;
    break;
  case AR_STATE_STA2: /* 駅, 停車継続 */
    if (!ar_stay_next()) {
      ar_state = AR_STATE_RUN1;
    }
    break;
  }
  return ar_state;
}
```

ソースファイル fsm-reverse.c (自動運転有限状態機械 (リバース): 実装)

リバーストラックプランに対する自動運転有限状態機械を実現します. トラックプランは図 29.3c で, 状態遷移図は図 29.23 で, それぞれ示したものです.

ソースファイル番号 29.10: fsm-reverse.c

```
/*-------------------------------------------------------
```

```
 * トラックプラン: リバース
 *-----------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "pp.h"
#include "pp-const.h"
#include "ar.h"
#include "ar-motor.h"
#include "fsm.h"

/*
 *    北                              S３
 *  西＋東                          ＝＃＝
 *    南                          ／／  ＼＼
 *                              ｜｜      ｜｜
 *                             ＸＸ       ｜｜
 *      駅        Ｆ１        ／／        ／／
 *  ＝＃＝＝＃＝＝↑＝＝＊＝＝ＸＸ＝↑
 *    Ｓ２  Ｓ１        Ｔ１        Ｆ２
 *  （Ｆ：フィーダ、Ｓ：センサ、Ｔ：ポイント、ＸＸ：ギャップ）
 *
 * 動作シーケンスの概要(以下の繰り返し):
 *   1. 駅を出発
 *   2. エンドレス区間の途中(S3)でポイントT1を切替
 *   3. 駅に停車
 */
/* 状態 */
#define AR_STATE_RUN1     10 /* 駅を出発, 走行開始 */
#define AR_STATE_RUN2     11 /* 走行継続 */
#define AR_STATE_RUN3     12 /* リバース線処理 */
#define AR_STATE_RUN4     13 /* 走行継続 */
#define AR_STATE_STA1     14 /* 駅, 停車 */
#define AR_STATE_STA2     15 /* 駅, 停車継続 */
#define AR_STATE_STA3     16 /* 駅, 停車終了 */
/* 減速パラメータ番号 */
#define AR_DECP_ID_STA     1  /* 駅用 */
/* センサ番号 */
#define AR_SN_STA_BRAKE    1  /* S1: 駅, 制動用 */
#define AR_SN_STA_STOP     2  /* S2: 駅, 停止用 */
#define AR_SN_REVERSE      3  /* S3: リバース区間進入検知用 */
/* ポイント番号 */
#define AR_TO_REVERSE      1  /* T1: リバース用 */

/* 走行制御状態機械 */
enum ar_state_e fsm_reverse_next(enum ar_state_e ar_state)
{
  /* 初期状態: 駅に停車 */
  switch ((uint8_t)ar_state) {
  default:
    ar_state = AR_STATE_INIT;
    break;
  case AR_STATE_INIT:
    ar_track1_dir_set(AR_TRACK_DIR_NORMAL);
    ar_track2_dir_set(AR_TRACK_DIR_NORMAL);
    ar_motor_on(); /* モータ＆常点灯On */
    if_toq_set_normal(AR_TO_REVERSE);
    ar_state = AR_STATE_RUN1;
    break;
  case AR_STATE_RUN1: /* 駅を出発（リバース区間センサまで） */
    ar_motor_run_to(AR_SPEEDSTEP_MAX, AR_SN_REVERSE);
    ar_state = AR_STATE_RUN2;
    break;
  case AR_STATE_RUN2: /* 走行継続 */
    if (!ar_motor_next()) {
      ar_state = AR_STATE_RUN3;
    }
    break;
  case AR_STATE_RUN3: /* リバース線処理: 極性反転&ポイント切替 */
```

```
70      ar_track1_dir_set(AR_TRACK_DIR_REVERSE);
71      if_toq_set_reverse(AR_TO_REVERSE);
72      /* 駅に向けて走行 */
73      ar_motor_run_until(AR_SPEEDSTEP_MAX, AR_DECP_ID_STA,
74                         AR_SN_STA_BRAKE, AR_SN_STA_STOP);
75      ar_state = AR_STATE_RUN4;
76      break;
77    case AR_STATE_RUN4: /* 走行継続 */
78      if (!ar_motor_next()) {
79        ar_state = AR_STATE_HOME;
80      }
81      break;
82    case AR_STATE_HOME:   /* 定位置(駅) */
83      ar_state = AR_STATE_STA1;
84      break;
85    case AR_STATE_STA1: /* 駅, 停車 */
86      ar_track1_dir_set(AR_TRACK_DIR_NORMAL);
87      ar_stay_start(AR_STAYTIME_DEFAULT);
88      ar_state = AR_STATE_STA2;
89      break;
90    case AR_STATE_STA2: /* 停車継続 */
91      if (!ar_stay_next()) {
92        ar_state = AR_STATE_STA3;
93      }
94      break;
95    case AR_STATE_STA3: /* 停車終了 */
96      if_toq_set_normal(AR_TO_REVERSE); /* 定位 */
97      ar_state = AR_STATE_RUN1;
98      break;
99    }
100   return ar_state;
101 }
```

ソースファイル script-dummy.c (スクリプト言語: ダミー実装)

EEPROM に書き込まれた自動運転スクリプトを有限状態機械として実現する関数ですが, 本章ではこの機能は実現せずエラーとします. この機能を実現は第 30 章で行います.

ソースファイル番号 29.11: script-dummy.c

```
1  /*-----------------------------------------------------
2   * トラックプラン: スクリプト （＊未サポート＊）
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "ar.h"
7  #include "fsm.h"
8  #include "fatal.h"
9
10 /*-----------------------------------------------------
11  * 初期化
12  *-----------------------------------------------------*/
13 void script_init(void)
14 {
15   fatal(AR_ERROR_UNSUPPORTED);
16 }
17
18 /*-----------------------------------------------------
19  * スクリプト実行制御
20  *-----------------------------------------------------*/
21 enum ar_state_e fsm_script_next(enum ar_state_e ar_state)
22 {
23   fatal(AR_ERROR_UNSUPPORTED);
24   return AR_STATE_INIT;
25 }
```

ソースファイル if.h (インターフェースボード: ヘッダ)

インターフェースボードとはセンサ・ポイントインターフェースボードとアクセサリインターフェースボードの総称です. I/O エクスパンダ MCP23017 がこれらインターフェースボードに搭載されており, 車両センサ入力, ポイント制御出力, アクセサリ制御出力を行います. このソースファイルでは, インターフェースボードに関連する関数のプロトタイプ宣言や記号定数の定義を行います.

ソースファイル番号 29.12: if.h

```
#if !defined(__IF_H__)
#define __IF_H__

/* センサ番号 */
typedef uint8_t if_sn_id;
#define IF_SN_NULL   0

/* ポイント番号 */
typedef uint8_t if_to_id;
#define IF_TO_NULL   0

/* アクセサリ番号 */
typedef uint8_t if_ac_id;
#define IF_AC_NULL   0

/* ポイント開通方向 */
enum if_turnout_dir_e {
  TURNOUT_DIR_NORMAL,  /* 定位 */
  TURNOUT_DIR_REVERSE, /* 反位 */
  TURNOUT_DIR_NONE,    /* 未定 */
};

/* ポイント切り替え時間間隔 [mSec] */
#define IF_TURNOUT_TIME   100

/* 2通りのインターフェースモジュール:
 * - センサ・ポイントインターフェースボード
 *   センサ入力とポイント制御出力
 *     アドレス 0, 1, 2, ..., 6
 * - アクセサリインターフェースボード
 *   アクセサリ制御出力
 *     アドレス 7
 */

/* 最大インターフェースモジュール数 */
#define IF_MAX_MODULES          8
/* センサ・ポイントインターフェースボードの最大数 */
#define IF_MAX_MODULES_TYPE_A   7
/* アクセサリインターフェースボードの最大数 */
#define IF_MAX_MODULES_TYPE_B   1

/* ポート割付 (ソースコード中でハードコードされているので要注意)
 * センサ・ポイントインターフェースボード:
 *   GPA0:      センサ 1
 *   GPA1:      センサ 2
 *    ...
 *   GPA7:      センサ 8
 *   GPB0:      センサ 9
 *   GPB1:      センサ 10
 *   GPB2,GPB3: ポイント 1
 *   GPB4,GPB5: ポイント 2
 *   GPB6,GPB7: ポイント 3
 * アクセサリインターフェースボード:
 *   GPA0:      センサ 1
 *   GPA1:      センサ 2
 *     ...
 *   GPB7:      センサ 16
```

```
 */
/* センサ・ポイントインターフェースボード */
#define N_TURNOUTS_PER_MOD     3  /* ポイント制御出力数/1モジュール */
#define N_SENSORS_PER_MOD      10 /* 車両センサ数/1モジュール */
/* アクセサリインターフェースボード */
#define N_ACCESSARIES_PER_MOD  16 /* アクセサリ制御出力数/1モジュール */
/* メインボード内蔵のアクセサリ出力数 */
#define N_ACCESSARIES_BUILTIN   2
/* 最大のポイント制御出力数 */
#define IF_MAX_TURNOUTS   \
  (IF_MAX_MODULES_TYPE_A * N_TURNOUTS_PER_MOD)
/* 最大のセンサ数 */
#define IF_MAX_SENSORS    \
  (IF_MAX_MODULES_TYPE_A * N_SENSORS_PER_MOD)
/* 最大のアクセサリ制御出力数 */
#define IF_MAX_ACCESSARIES \
  (IF_MAX_MODULES_TYPE_B * N_ACCESSARIES_PER_MOD \
   + N_ACCESSARIES_BUILTIN)

/* インターフェースボードの接続の有無 */
extern uint8_t if_module[];
/* ポートA,ポートBの書き込み値 */
extern uint8_t if_wr_gpa[];
extern uint8_t if_wr_gpb[];
/* ポートA,ポートBの読み出し値 */
extern uint8_t if_rd_gpa[];
extern uint8_t if_rd_gpb[];

extern void   if_init(void);
extern void   if_sensor_init(void);
extern void   if_sensor_reset(void);
extern void   if_sensor_read(void);
extern int8_t if_sensor_get(if_sn_id sn);
extern int8_t if_sensor_detected(void);
extern void   if_turnout_init(void);
extern int8_t if_turnout_action(if_to_id p, enum if_turnout_dir_e d);
extern void   if_accessary_init(void);
extern int8_t if_accessary_on(if_ac_id ac);
extern int8_t if_accessary_off(if_ac_id ac);
extern void   if_accessary_off_all(void);

#endif /*__IF_H__*/
```

ソースファイル if.c (インターフェースボード: 実装)

インターフェースボードの初期化を行います. I^{2}C バスへに MCP23017 の接続を検出し, 初期化を行います.

ソースファイル番号 29.13: if.c

```
/*-----------------------------------------------------
 * インターフェースモジュール
 *-----------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "ar.h"
#include "fatal.h"
#include "if-toq.h"
#include "if.h"
#include "i2c.h"
#include "i2c-mcp23017.h"

/* インターフェースボードの接続の有無 */
uint8_t if_module[IF_MAX_MODULES];
/* ポートA,ポートBの書き込み値 */
uint8_t if_wr_gpa[IF_MAX_MODULES];
uint8_t if_wr_gpb[IF_MAX_MODULES];
/* ポートA,ポートBの読み出し値 */
```

```
uint8_t if_rd_gpa[IF_MAX_MODULES];
uint8_t if_rd_gpb[IF_MAX_MODULES];

/*------------------------------------------------------
 * インターフェースボード初期化
 *------------------------------------------------------*/
void if_init(void)
{
  uint8_t nif = 0;
  uint8_t caddr = 0;
  /* 書き込み値/読み出し値 */
  for (caddr = 0; caddr < IF_MAX_MODULES; caddr++) {
    if_wr_gpa[caddr] = 0x00;
    if_wr_gpb[caddr] = 0x00;
    if_rd_gpa[caddr] = 0x00;
    if_rd_gpb[caddr] = 0x00;
  }
  /* 初期化*/
  if_sensor_init();
  if_toq_init();
  if_turnout_init();
  if_accessary_init();
  /* 各I2Cアドレスに対してMCP23017の接続を確認し初期化 */
  for (caddr = 0; caddr < IF_MAX_MODULES; caddr++) {
    CLRWDT();
    if (!I2C_Exist(I2C_MCP23017_CTLBYTE, caddr)) {
      /* このアドレスに接続無し */
      if_module[caddr] = 0;
    } else {
      /* このアドレスに接続有り */
      if_module[caddr] = 1;
      nif++;
      /* 初期化 */
      i2c_mcp23017_init(caddr);
      if (caddr < IF_MAX_MODULES_TYPE_A) {
        /* センサ・ポイントインターフェースボード */
        /* IOCON: 設定 / INTCC=1: INTCAP読み出しでクリア */
        i2c_mcp23017_reg_write(caddr,
                               I2C_MCP23017_REG_IOCON, 0x01);
        /* DEFVAL: 比較値 / 通常はHで, Lへの変化を検出 */
        i2c_mcp23017_reg_write(caddr,
                               I2C_MCP23017_REG_DEFVALA, 0xFF);
        i2c_mcp23017_reg_write(caddr,
                               I2C_MCP23017_REG_DEFVALB, 0xFF);
        /* INTCON: 比較対象 / DEFVALとの比較 */
        i2c_mcp23017_reg_write(caddr,
                               I2C_MCP23017_REG_INTCONA, 0xFF);
        i2c_mcp23017_reg_write(caddr,
                               I2C_MCP23017_REG_INTCONB, 0xFF);
        /* IODIR: I/O方向 (ポート割付依存) */
        i2c_mcp23017_a_dir_set(caddr, 0xFF);
        i2c_mcp23017_b_dir_set(caddr, 0x03);
      } else {
        /* アクセサリインターフェースボード */
        /* IODIR: I/O方向 (ポート割付依存) */
        i2c_mcp23017_a_dir_set(caddr, 0x00);
        i2c_mcp23017_b_dir_set(caddr, 0x00);
      }
      /* ポート初期値書き込み */
      i2c_mcp23017_a_write(caddr, if_wr_gpa[caddr]);
      i2c_mcp23017_b_write(caddr, if_wr_gpb[caddr]);
      /* INTCAP 空読み*/
      i2c_mcp23017_reg_read(caddr, I2C_MCP23017_REG_INTCAPA);
      i2c_mcp23017_reg_read(caddr, I2C_MCP23017_REG_INTCAPB);
    }
  }
  if (nif == 0) {
    fatal(AR_ERROR_NO_IFBOARD);
  }
}
```

ソースファイル if-sensor.c (インターフェースボード・センサ部: 実装)

車両センサの値を読み取り記憶します. 車両センサの検出は一瞬の場合がありますから, MCP23017 のポートの値に加えて IOC 機能を持つ INTCAP レジスタの値も読み, クリアされるまでは記憶し続けます. プログラムは若干複雑に見えるのはセンサ番号と MCP23017 のポートとの対応関係の処理によるもので, それ以外は単純な動作です.

- `if_sensor_init()`
  センサ部の初期化をします.
- `if_sensor_reset()`
  センサ部の再初期化をします.
- `if_sensor_read()`
  すべてのセンサの検出値を読み取り, 記憶します.
- `if_sensor_get()`
  指定したセンサの検出値を返します. ただし検出値は既に記憶しているもので, 新たな読み取りは行いません. この関数の呼び出しにより記憶しているセンサの検出値をクリアします.

ソースファイル番号 29.14: if-sensor.c

```
1  /*------------------------------------------------------
2   * センサ入力 (センサ・ポイントインターフェースボード)
3   *------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "io.h"
8  #include "ar.h"
9  #include "if-toq.h"
10 #include "if.h"
11 #include "i2c.h"
12 #include "i2c-mcp23017.h"
13
14 /* 検出有無のまとめ: 1=あるセンサで検出有り, 0=検出無し */
15 static int8_t if_sn_detected = 0;
16
17 /*------------------------------------------------------
18  * 初期化
19  *------------------------------------------------------*/
20 void if_sensor_init(void)
21 {
22   /* ポート初期値 */
23   uint8_t caddr = 0;
24   for ( ; caddr < IF_MAX_MODULES_TYPE_A; caddr++) {
25     /*(ポート割付依存)*/
26     if_rd_gpa[caddr] = (unsigned)if_rd_gpa[caddr] & ~0xFF;
27     if_rd_gpb[caddr] = (unsigned)if_rd_gpb[caddr] & ~0x03;
28   }
29 }
30
31 /*------------------------------------------------------
32  * センサ読み取り値の全クリア
33  *------------------------------------------------------*/
34 void if_sensor_reset(void)
35 {
36   if_sn_detected = 0;
37   if_sensor_read();
38   if_sensor_init();
39 }
40
41 /*------------------------------------------------------
42  * センサ読み取り
```

```
 *-------------------------------------------------------*/
void if_sensor_read(void)
{
  if_sn_detected = 0;
  uint8_t caddr = 0;
  for ( ; caddr < IF_MAX_MODULES_TYPE_A; caddr++) {
    CLRWDT();
    if (if_module[caddr] != 0) {
      /* ポート読み出し */
      uint8_t vga = i2c_mcp23017_a_read(caddr);
      uint8_t vgb = i2c_mcp23017_b_read(caddr);
      /* IOC読み出し */
      uint8_t via
        = i2c_mcp23017_reg_read(caddr, I2C_MCP23017_REG_INTCAPA);
      uint8_t vib
        = i2c_mcp23017_reg_read(caddr, I2C_MCP23017_REG_INTCAPB);
      /* 検出->値1, 非検出->値0 */
      uint8_t a = (unsigned)(~vga | via) & 0xFF; /*(ポート割付依存)*/
      uint8_t b = (unsigned)(~vgb | vib) & 0x03; /*(ポート割付依存)*/
      /* 検出値の更新 */
      if_rd_gpa[caddr] |= a;
      if_rd_gpb[caddr] |= b;
      /* 検出有無のまとめ */
      if_sn_detected |= ((a | b) != 0);
    }
  }
}

/*-------------------------------------------------------
 * センサ読み取り値を返す（読出し後にクリア）
 *-------------------------------------------------------*/
int8_t if_sensor_get(if_sn_id sn)
{
  if ((sn == IF_SN_NULL) || (IF_MAX_SENSORS < sn)) {
    return 0;
  }
  /* 該当インターフェースボードのアドレスとポート番号
   * caddr := (sn-1) / N_SENSORS_PER_MOD
   *     g := (sn-1) % N_SENSORS_PER_MOD */
  uint8_t g = (uint8_t)(sn - 1);
  uint8_t caddr = 0;
  while (g >= N_SENSORS_PER_MOD) {
    g -= N_SENSORS_PER_MOD;
    caddr++;
  }
  /* センサ読み出し情報を取得 */
  int8_t val = 0;
  if (if_module[caddr] != 0) {
    /* 対応ビットの読み取りとクリア（ポート割付依存） */
    const uint8_t bit_a[8] = { /* ポートA: センサ 1..8 */
      0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };
    const uint8_t bit_b[8] = { /* ポートB: センサ 9,10 */
      0x01, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
    if (g < 8) {
      val = ((if_rd_gpa[caddr] & bit_a[(unsigned)g])   != 0);
      if_rd_gpa[caddr] &= (unsigned)~bit_a[(unsigned)g];
    } else {
      val = ((if_rd_gpb[caddr] & bit_b[(unsigned)g-8]) != 0);
      if_rd_gpb[caddr] &= (unsigned)~bit_b[(unsigned)g-8];
    }
  }
  return val;
}

/*-------------------------------------------------------
 * センサ検出有無のまとめ
 *-------------------------------------------------------*/
int8_t if_sensor_detected(void)
{
  return if_sn_detected;
}
```

ソースファイル if-turnout.c (インターフェースボード・ポイント部: 実装)

MCP23017 のポートを操作してポイント切替を行います. 前述のソースファイル if-toq.c からポイント切替の動作が呼び出されます.

- if_turnout_init()
  MCP23017 のポートのポイント部を初期化します.
- if_turnout_action()
  実際のポイント切替を実行する部分で, MCP23017 のポートの操作を行います.

ソースファイル番号 29.15: if-turnout.c

```
1  /*-----------------------------------------------------------
2   * ポイント制御出力 (センサ・ポイントインターフェースボード)
3   *-----------------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "io.h"
8  #include "if.h"
9  #include "if-toq.h"
10 #include "i2c.h"
11 #include "i2c-mcp23017.h"
12
13 /*-----------------------------------------------------------
14  * 初期化
15  *-----------------------------------------------------------*/
16 void if_turnout_init(void)
17 {
18   /* ポート初期値 */
19   uint8_t caddr = 0;
20   for ( ; caddr < IF_MAX_MODULES_TYPE_A; caddr++) {
21     /* (ポート割付依存) */
22     if_wr_gpa[caddr] = (unsigned)if_wr_gpa[caddr] & ~0x00;
23     if_wr_gpb[caddr] = (unsigned)if_wr_gpb[caddr] & ~0xFC;
24   }
25 }
26
27 /*-----------------------------------------------------------
28  * 切替の実行
29  *-----------------------------------------------------------*/
30 int8_t if_turnout_action(if_to_id p,
31                          enum if_turnout_dir_e d)
32 {
33   if ((p == IF_TO_NULL) || (IF_MAX_TURNOUTS < p)) {
34     return 0;
35   }
36   /* 該当のインターフェースボード番号とポート番号
37    * caddr := (p-1) / N_TURNOUTS_PER_MOD
38    *     g := (p-1) % N_TURNOUTS_PER_MOD */
39   uint8_t caddr = 0;
40   uint8_t g = (uint8_t) (p - 1);
41   while (g >= N_TURNOUTS_PER_MOD) {
42     g -= N_TURNOUTS_PER_MOD;
43     caddr++;
44   }
45   /* ポイント制御出力の更新 (ポート割付依存) */
46   const uint8_t bit_mask[3] = { 0x0C, 0x30, 0xC0 };
47   const uint8_t bit_nor[3]  = { 0x04, 0x10, 0x40 };
48   const uint8_t bit_rev[3]  = { 0x08, 0x20, 0x80 };
49   if_wr_gpb[caddr] &= (unsigned)~bit_mask[g];
50   if (d == TURNOUT_DIR_NORMAL) {
51     if_wr_gpb[caddr] |= bit_nor[g];
```

```
52    } else {
53      if_wr_gpb[caddr] |= bit_rev[g];
54    }
55    /* 出力 */
56    if (if_module[caddr] != 0) {
57      i2c_mcp23017_b_write(caddr, if_wr_gpb[caddr]);
58    }
59    /* 返り値: 実際にインターフェースボード接続があるか否か */
60    return (if_module[caddr] != 0);
61  }
```

ソースファイル if-accessary.c (インターフェースボード・アクセサリ部: 実装)

アクセサリ制御出力のオン・オフを行います.

- if_accessary_init()
  アクセサリ部を初期化します.
- if_accessary_on()
  指定したアクセサリ制御出力をオンにします. これは遷移の動作部の記述で使用する記法 A(On) に対応する動作の実装です.
- if_accessary_off()
  指定したアクセサリ制御出力をオフにします. これは遷移の動作部の記述で使用する記法 A(Off) に対応する動作の実装です.
- if_accessary_action()
  実際のアクセサリ制御を実行する部分です. アクセサリ制御出力の一部はメインボードに実装してありますので, メインボード上のアクセサリ制御出力の場合にはマイコンの I/O ポートの操作を行います. アクセサリボード上のアクセサリ制御出力の場合には MCP23017 のポートの操作を行います.

ソースファイル番号 29.16: if-accessary.c

```
1   /*-------------------------------------------------------
2    * アクセサリ制御出力 (アクセサリインターフェースボード)
3    *-------------------------------------------------------*/
4   #include <xc.h>
5   #include <stdint.h>
6   #include "mcr.h"
7   #include "io.h"
8   #include "ar.h"
9   #include "if-toq.h"
10  #include "if.h"
11  #include "i2c.h"
12  #include "i2c-mcp23017.h"
13
14  static int8_t if_accessary_action(if_ac_id ac,
15                                    uint8_t action);
16
17  /*-------------------------------------------------------
18   * 初期化
19   *-------------------------------------------------------*/
20  void if_accessary_init(void)
21  {
22    ACC_1_OFF();
23    ACC_2_OFF();
24    uint8_t caddr = 7;
25    for ( ; caddr < IF_MAX_MODULES_TYPE_B; caddr--) {
26      /*(ポート割付依存)*/
27      if_wr_gpa[caddr] = (unsigned)if_wr_gpa[caddr] & ~0xFF;
28      if_wr_gpb[caddr] = (unsigned)if_wr_gpb[caddr] & ~0xFF;
29    }
30  }
```

```
31
32  /*------------------------------------------------------
33   * 出力On
34   *------------------------------------------------------*/
35  int8_t if_accessary_on(if_ac_id ac)
36  {
37    return if_accessary_action(ac, 1);
38  }
39
40  /*------------------------------------------------------
41   * 出力Off
42   *------------------------------------------------------*/
43  int8_t if_accessary_off(if_ac_id ac)
44  {
45    return if_accessary_action(ac, 0);
46  }
47
48  void if_accessary_off_all(void)
49  {
50    if_ac_id ac = 0;
51    for (ac = 1; ac <= IF_MAX_ACCESSARIES; ac++) {
52      if_accessary_action(ac, 0);
53    }
54  }
55
56  /*------------------------------------------------------
57   * アクセサリ出力制御 (action=1:On, action=0:Off)
58   *------------------------------------------------------*/
59  static int8_t if_accessary_action(if_ac_id ac,
60                                    uint8_t action)
61  {
62    if ((ac == IF_AC_NULL) || (IF_MAX_ACCESSARIES < ac)) {
63      return 0;
64    }
65    /* 出力*/
66    int8_t physically_done = 1;
67    if (ac <= N_ACCESSARIES_BUILTIN) {
68      /* 出力はメインボード上で */
69      switch (ac) {
70      case 1:
71        if (action == 1) { /* On  */
72          ACC_1_ON();
73        } else {           /* Off */
74          ACC_1_OFF();
75        }
76        break;
77      case 2:
78        if (action == 1) { /* On  */
79          ACC_2_ON();
80        } else {           /* Off */
81          ACC_2_OFF();
82        }
83        break;
84      default:
85        physically_done = 0;
86        break;
87      }
88    } else {
89      /* 出力はインターフェースボード上で */
90      uint8_t xac = (uint8_t)(ac - N_ACCESSARIES_BUILTIN);
91      /* 該当のインターフェースボード番号とポート番号
92       * caddr := (xac-1) / N_ACCESSARIES_PER_MOD
93       *     g := (xac-1) % N_ACCESSARIES_PER_MOD */
94      uint8_t caddr = 7;
95      uint8_t g = (unsigned)xac - 1;
96      while (g >= N_ACCESSARIES_PER_MOD) {
97        g -= N_ACCESSARIES_PER_MOD;
98        caddr--;
99      }
100     /* アクセサリ制御出力の更新（ポート割付依存）*/
101     const uint8_t bit_table[16] = {
```

```
102         0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
103         0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80 };
104       if (action == 1) { /* On  */
105         if (g < 8) {
106           if_wr_gpa[caddr] |= (unsigned) bit_table[g];
107         } else {
108           if_wr_gpb[caddr] |= (unsigned) bit_table[g];
109         }
110       } else {            /* Off */
111         if (g < 8) {
112           if_wr_gpa[caddr] &= (unsigned)~bit_table[g];
113         } else {
114           if_wr_gpb[caddr] &= (unsigned)~bit_table[g];
115         }
116
117       }
118       /* 出力 */
119       if (if_module[caddr] != 0) {
120         i2c_mcp23017_a_write(caddr, if_wr_gpa[caddr]);
121         i2c_mcp23017_b_write(caddr, if_wr_gpb[caddr]);
122       } else {
123         physically_done = 0;
124       }
125     }
126     /* 返り値: 実際にインターフェースボード接続があるか否か */
127     return physically_done;
128   }
```

ソースファイル ar.h (自動運転: ヘッダ)

自動運転に関する関数のプロトタイプ宣言や記号定数の定義を行います.

ソースファイル番号 29.17: ar.h

```
1   #if !defined(__AR_H__)
2   #define __AR_H__
3
4   #include "if.h"
5   #include "if-toq.h"
6   #include "ar-decp.h"
7   #include "ar-motor.h"
8   #include "ar-stay.h"
9
10  /* 走行パターン */
11  enum op_pattern_e {
12    OPPAT_UNDEF = 0,
13    OPPAT_SHUTTLE,
14    OPPAT_ENDLESS,
15    OPPAT_REVERSE,
16    OPPAT_TEST_IF_BOARD,
17    OPPAT_EEPROMSCRIPT,
18  };
19
20  /* 走行状態 */
21  enum ar_state_e {
22    AR_STATE_INIT =  0, /* 初期状態 */
23    AR_STATE_HOME,      /* 定位置 */
24  };
25
26  /* 線路極性 */
27  enum ar_track_dir_e {
28    AR_TRACK_DIR_NORMAL = 0, /* 順方向 */
29    AR_TRACK_DIR_REVERSE,    /* 逆方向 */
30  };
31
32  /* オートオフ時間 [mSec] */
33  #define T_AUTO_OFF        (3600000)  /* 60分 */
34  #define T_AUTO_OFF_BRUTE  ( 300000)  /*  5分 */
35
```

```
36 extern enum op_pattern_e op_pattern;
37 extern struct ar_motor_s ar_motor;
38 extern uint16_t  p_vslow;
39 extern uint16_t  p_staytim;
40
41 extern void auto_run_init(void);
42 extern void auto_run_start(void);
43 extern void ar_track1_dir_set(enum ar_track_dir_e tdir);
44 extern void ar_track2_dir_set(enum ar_track_dir_e tdir);
45 extern void ar_param_read(void);
46
47 #endif /*__AR_H__*/
```

ソースファイル ar.c (自動運転: 実装)

自動運転のための関数の実装を行います. 複雑そうに見えますが, パワーパックのファームウエアと類似していますから容易に理解できると思います. 本ソースファイルでは以下の関数を記述しています.

- ar_param_read()
  可変抵抗器による各種パラメータの設定を読み込みます.
- ar_track1_dir_set()
  フィーダ出力 1 の線路極性を設定します. パワーパックの場合ではモータドライバ IC に与える信号を変えることで線路極性を切り替えていましたが, 本装置ではリレーの切り替えにより線路極性を切り替えます. これは遷移の動作部の記述で使用する記法 F(1,d) に対応する動作の実装です.
- ar_track2_dir_set()
  フィーダ出力 2 の線路極性を設定します. これは遷移の動作部の記述で使用する記法 F(2,d) に対応する動作の実装です.
- auto_run_init()
  自動運転の初期化を行います. まず車両センサの検出値をリセットします. 次に可変抵抗器によるパラメータ設定を読み込んで, その値に従って減速度パラメータの初期値を設定します. 関数 main() の初期化の部分でこの関数が呼び出された後は, この関数が呼び出されることはありません. つまり減速度パラメータの初期化は一度に限っている点に注意して下さい. これは自動運転で学習した減速度パラメータを次回の自動運転でも使用できるようにして, 自動運転の度に学習をやり直さないようにするためです.
- auto_run_start()
  自動運転を行う際の中心的な実行のループとなる部分です. パワーパックのファームウエアにおける関数 pp_control() と構造が酷似していますから, 比較しながら読むと理解が進むと思います. 自動運転には一定時間経過をすれば自動運転を終了するオートオフ機能を実現します. 終了ボタン END またはオートオフによる自動運転の実際の終了は, 有限状態機械が HOME 状態になってから, すなわち模型列車が定位置に戻ってからとしています.

関数 auto_run_start() の少し詳しい骨子を図 29.26 に示します.

これまでに紹介したパワーパックでは人が操作する可変抵抗器やスイッチに基づいて走行速度と走行方向を決定していました. 自動運転では有限状態機械の決定に基づいて走行速度と走行方向を決定します. 違いは僅かにこれだけですから, パワーパックの動作の理解を元に自動運転の動作の理解に努めて下さい.

ソースファイル番号 29.18: ar.c

```
1 /*-----------------------------------------------------
2  * 自 動 運 転 モ ー ド
3  *-----------------------------------------------------*/
```

```
【ソースファイル AutoTrainOp.c】

void main(void)
{
  初期化;
  auto_run_start();
}

【ソースファイル ar.c】

ar_state = AR_STATE_INIT;  /* 有限状態機械の状態 */

void auto_run_start(void)
{
  自動運転開始の押しボタンスイッチの押下を待機;
  全ポイントを定位に切替;
  for(;;) {
    if_sensor_read();                /* センサの読み込み */
    ar_state = fsm_next(ar_state);   /* 有限状態機械の動作 */
    if_toq_next();                   /* ポイント切替 */
    ar_param_read();                 /* 可変抵抗器の設定の読み込み */
    md = モータPWMデューティ比;
    pp_sb_duty_set(md);
  }
}
```

図 29.26: 関数 auto_run_start() の構造 (概要)

```
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "io.h"
#include "pp.h"
#include "pp-const.h"
#include "pp-sb.h"
#include "ar.h"
#include "fsm.h"
#include "if-toq.h"
#include "if.h"

/* 走行制御状態機械 */
struct ar_motor_s ar_motor;
/* 自動運転状態 */
static enum ar_state_e ar_state = AR_STATE_INIT;
/* 自動運転パラメータ値 */
uint16_t  p_vslow   = 0;
uint16_t  p_staytim = 0;

static void wait_start_button_release(void);

/* 各種パラメータ読み込み */
void ar_param_read(void)
{
  /* 標準パラメータを読み込む */
  pp_param_read();
  /* ボリューム値の読み込み */
  uint16_t vr_vslow   = pp_adc_read(ADC_CH_VSLOW);
  uint16_t vr_staytim = pp_adc_read(ADC_CH_STAYTIM);
  /* 割り込み禁止 */
```

```
  int8_t ogie  = GIE;
  int8_t opeie = PEIE;
  GIE  = 0;
  PEIE = 0;
  /* パラメータ値の決定 */
  p_vslow   = vr_vslow >> 1; /* 0 ..  511 */
  p_staytim = vr_staytim;    /* 0 .. 1023 */
  /* 割り込み禁止/許可状態を復元 */
  GIE  = ogie;
  PEIE = opeie;
}

/* 線路極性設定(トラック1) */
void ar_track1_dir_set(enum ar_track_dir_e tdir)
{
  if (tdir == AR_TRACK_DIR_NORMAL) {
    TDIR1_NORM();
  } else {
    TDIR1_REV();
  }
}

/* 線路極性設定(トラック2) */
void ar_track2_dir_set(enum ar_track_dir_e tdir)
{
  if (tdir == AR_TRACK_DIR_NORMAL) {
    TDIR2_NORM();
  } else {
    TDIR2_REV();
  }
}

/* 走行開始ボタンの開放を待機 */
static void wait_start_button_release(void)
{
  uint8_t c = 0;
  /* 一定回数連続したSTARTボタンの解放を検出(チャタリング対策) */
  while (c != 100) {
    pp_delay_ms(1);
    CLRWDT();
    if (SW_START()) {
      c = 0; /* 押下中 */
    } else {
      c++;   /* 開放中 */
    }
  }
}

/* 自動運転初期化 */
void auto_run_init(void)
{
  /* センサ状態の初期化 */
  if_sensor_reset();
  /* 減速度パラメータの初期化 */
  ar_param_read();
  ar_decp_init();
}

/* 自動運転開始 */
void auto_run_start(void)
{
  /* LED */
  LED_RUN_OFF();
  LED_IDLE_ON();
  LED_SENS_OFF();
  /* 自動運転開始ボタンの押下を待機 */
  while (!SW_START()) {
    CLRWDT();
    /* 現在モード終了 */
    if (!SW_MODE_AUTO()) {
      return;
```

```
106       }
107       /* センサ検出 LED 表示 */
108       if_sensor_read();
109       if (if_sensor_detected()) {
110         LED_SENS_ON();
111       } else {
112         LED_SENS_OFF();
113       }
114       /* モータードライバ LSI TB6643KQ 制御 (過電流検出の解除) */
115       if (SW_EMG_STOP()) { /* 解除要求の発生 */
116         extern void MotorDriver_TB6643KQ_Reset(void);
117         MotorDriver_TB6643KQ_Reset();
118       }
119     }
120     /* 自動運転開始ボタンの開放を待機 */
121     wait_start_button_release();
122     /* LED表示(全On) */
123     LED_RUN_ON();
124     LED_IDLE_ON();
125     LED_SENS_ON();
126     /* 自動運転モーター制御の初期化 */
127     ar_motor_init();
128     /* 全ポイント定位 */
129     if_toq_set_all_reverse(); /* 全反位 */
130     while (if_toq_next(RUN_INTERVAL)) {
131       pp_delay_ms(RUN_INTERVAL);
132       CLRWDT();
133     }
134     if_toq_set_all_normal();  /* 全定位 */
135     while (if_toq_next(RUN_INTERVAL)) {
136       pp_delay_ms(RUN_INTERVAL);
137       CLRWDT();
138     }
139     /* LED表示(全Off) */
140     LED_RUN_OFF();
141     LED_IDLE_OFF();
142     LED_SENS_OFF();
143     /* オートオフ用タイマ */
144     uint32_t  t_auto_off_grace = 0;
145     uint32_t  t_auto_off_brute = 0;
146     /* 終了予告 LED タイマ */
147     uint16_t  t_led_stop = 0;
148     /* 終了フラグ(定位置にて) */
149     int8_t stop_gracefully = 0;
150     /* 主ループ(自動運転) */
151     ar_state = AR_STATE_INIT;
152     for (;;) {
153       pp_delay_ms(RUN_INTERVAL);
154       CLRWDT();
155       /* RUN LED 点滅 */
156       static uint8_t t_led_tick = 0;
157       t_led_tick++;
158       if (t_led_tick == NRUNS_PER_SEC) {
159         LED_RUN_ON();
160         t_led_tick = 0;
161       } else {
162         LED_RUN_OFF();
163       }
164       /* モータードライバ LSI TB6643KQ 制御 (過電流検出の解除) */
165       if (SW_EMG_STOP()) { /* 解除要求の発生 */
166         extern void MotorDriver_TB6643KQ_Reset(void);
167         MotorDriver_TB6643KQ_Reset();
168       }
169       /* 非常停止 */
170       if (SW_EMG_STOP()) {
171         break;
172       }
173       /* 現在モード終了 */
174       if (!SW_MODE_AUTO()) {
175         break;
```

```
176       }
177       /* 終了ボタン押下チェック */
178       if (SW_END()) {
179         stop_gracefully = 1;
180       }
181       /* オートオフ処理 */
182       if (t_auto_off_grace < T_AUTO_OFF) {
183         t_auto_off_grace += RUN_INTERVAL;
184         t_auto_off_brute = 0;
185       } else {
186         stop_gracefully = 1; /* 列車が定位置に来れば終了 */
187         if (t_auto_off_brute < T_AUTO_OFF_BRUTE) {
188           t_auto_off_brute += RUN_INTERVAL;
189         } else {
190           break; /* 強制終了(定位置にあるか否かに関わらず) */
191         }
192       }
193       /* 終了予告: IDLE LED 点滅 */
194       if (stop_gracefully) {
195         t_led_stop += RUN_INTERVAL;
196         if (t_led_stop > T_LED_WARNING) {
197           t_led_stop = 0;
198           LED_IDLE_TOGGLE();
199         }
200       }
201       /* センサ読み込み */
202       if_sensor_read();
203       /* センサ検出 LED 点灯 */
204       if (if_sensor_detected()) {
205         LED_SENS_ON();
206       } else {
207         LED_SENS_OFF();
208       }
209       /* 自動運転の状態遷移 */
210       ar_state = fsm_next(ar_state);
211       /* 終了(列車は定位置) */
212       if ((ar_state == AR_STATE_HOME) && stop_gracefully) {
213         break;
214       }
215       /* ポイント切替 */
216       if_toq_next(RUN_INTERVAL);
217       /* ボリューム読み込み */
218       ar_param_read();
219       /* 現在走行速度 */
220       uint16_t speed = ar_motor.speed;
221       /* 走行速度を変換して, モーターPWMデューティー比を得る */
222       uint16_t md = speed;
223       /* モーターPWMデューティー比を出力 */
224       pp_sb_duty_set(md);
225     }
226   }
```

ソースファイル ar-decp.h (自動運転減速度パラメータ: ヘッダ)

減速度パラメータに関する記号定数の定義, 管理するデータ構造の定義, および関連する関数のプロトタイプ宣言を行います. 記号定数の定義により, 減速度パラメータは 128 個の使用が可能です. 自動学習に必要なデータは構造体 ar_decp_s を用いて保持します. この構造体のメンバは以下の通りです.

- decel
  現在の減速度パラメータの値です.
- decel_unreached
  過去の減速度パラメータの値で, 減速が強過ぎたために停止センサに到達する前に速度が低下して最低維持速度での運転になった事例での値です. そのような過去の事例の内で減速度パラメータの値が最も

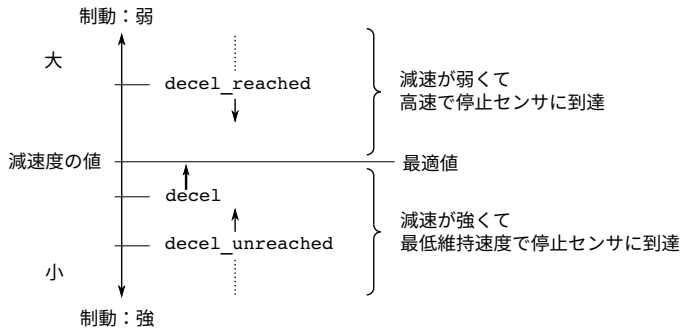


図 29.27: 減速度パラメータの改善プロセス

大きいものを保持します. なお減速度パラメータは数値として値が小さいほど減速が急で, 値が大きいほど減速が緩やかな点に注意して下さい.

- decel_reached
  過去の減速度パラメータの値で, 減速が弱過ぎたために走行速度が高速のまま停止センサに到達した事例での値です. そのような過去の事例の内で減速度パラメータの値が最も小さいものを保持します.

図 29.27 にこれらの値の関係を示します. 自動学習では模型列車が走行する度に上記の項目の数値を更新し, それらの数値と式 29.5 に基づいて減速度パラメータを最適値へと次第に近づけて行きます.

ソースファイル番号 29.19: ar-decp.h

```
1  #if !defined(__AR_DECP_H__)
2  #define __AR_DECP_H__
3
4  /* 減速度パラメータ番号 */
5  typedef uint8_t  ar_decp_id;
6  #define AR_DECP_ID_NULL   0
7
8  /* 減速度パラメータ総数 */
9  #define N_AR_DECP   128
10
11 /* 減速度パラメータ */
12 struct ar_decp_s {
13   uint16_t decel;
14   uint16_t decel_unreached;
15   uint16_t decel_reached;
16 };
17
18 extern struct ar_decp_s ar_decp[];
19 extern void ar_decp_init(void);
20
21 #endif /*__AR_DECP_H__*/
```

ソースファイル ar-decp.c (自動運転減速度パラメータ: 実装)

減速度パラメータに関する関数の実装を行います. ここでは減速度パラメータの過去の事例をクリアするだけです.

ソースファイル番号 29.20: ar-decp.c

```
1  /*-----------------------------------------------------
2   * 減速度パラメータ(自動運転用)
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "ar-decp.h"
```

```
7
8  /* 減速度パラメータ値（添字0は不使用; 要素数+1に注意） */
9  struct ar_decp_s ar_decp[N_AR_DECP+1];
10
11 /* 初期化 */
12 void ar_decp_init(void)
13 {
14   uint8_t decpi = 0;
15   for (decpi = 0; decpi <= N_AR_DECP; decpi++) {
16     ar_decp[decpi].decel = 0;
17   }
18 }
```

ソースファイル ar-motor.h (自動運転モーター制御: ヘッダ)

先述した 2 つの基本操作のセンサ停止走行 RUN-UNTIL(v,i,Sb,Ss) およびセンサ通過走行 RUN-TO(v,Sp) に関連する関数のプロトタイプ宣言を行います.

ソースファイル番号 29.21: ar-motor.h

```
1  #if !defined(__AR_MOTOR_H__)
2  #define __AR_MOTOR_H__
3
4  /* 減速度の最大値(オーバフロー対策) */
5  #define AR_DECEL_MAX    0x2000
6
7  /* 走行の状態(力行, 惰行, 制動) */
8  enum ar_motor_state_e {
9    AR_MOTOR_ACC_COA, /* 力行・惰行 */
10   AR_MOTOR_BRAKE,   /* 制動（制動センサから停止センサまで） */
11   AR_MOTOR_SLOW,    /* 最低維持速度で走行継続 */
12   AR_MOTOR_STOP1,   /* 減速1(停止センサ到達後) */
13   AR_MOTOR_STOP2,   /* 減速2(停止センサ到達後) */
14   AR_MOTOR_DONE,    /* 完了 */
15 };
16
17 /* 最低維持速度から停車までの時間 [Sec] */
18 #define AR_TIME_TO_STOP   1
19
20 /* 停止センサへの到達 */
21 enum ar_snstop_reach_e {
22   AR_SNSTOP_UNDEF = 0, /* 未定義 */
23   AR_SNSTOP_REACH,     /* 停止センサ到達時は維持速度より高速 */
24   AR_SNSTOP_UNREACH    /* 停止センサ到達前に維持速度にある */
25 };
26
27 /* モータ速度パラメータ */
28 struct ar_motor_s {
29   enum ar_motor_state_e state;  /* 現在状態 */
30   uint16_t   speed;         /* 現在速度 */
31   uint16_t   speed_target; /* 最高速度 */
32   uint16_t   accel;         /* 加速度 */
33   uint16_t   decel_stop;    /* 減速度(最低維持速度から停車時) */
34   ar_decp_id decpi;         /* 減速パラメータ番号 */
35   if_sn_id   sn_pass;       /* 通過センサ番号 */
36   if_sn_id   sn_brake;      /* 制動センサ番号 */
37   if_sn_id   sn_stop;       /* 停止センサ番号 */
38   uint16_t   spd_brake; /* 制動センサ通過時の速度 */
39   uint16_t   spd_stop;  /* 停止センサ通過時の速度 */
40   uint16_t   spd_slow;  /* 最低維持速度 */
41   uint32_t   run_dist;  /* 制動/停止センサ間での累積走行距離 */
42   enum ar_snstop_reach_e snstop_reach; /* 停止センサへの到達 */
43 };
44
45 /* 走行速度係数の最大値 */
46 typedef uint8_t  ar_speedstep_t;
```

```
47 #define AR_SPEEDSTEP_MAX          32
48 #define AR_SPEEDSTEP_SCALE(SS)                  \
49   (((unsigned)(SS) * SPEED_MAX) >> 5)
50
51 extern void   ar_motor_init(void);
52 extern void   ar_motor_on(void);
53 extern void   ar_motor_off(void);
54 extern void   ar_motor_run_until(ar_speedstep_t spdstep_target,
55                                  ar_decp_id decpi,
56                                  if_sn_id sn_brake,
57                                  if_sn_id sn_stop);
58 extern void   ar_motor_run_to(ar_speedstep_t spdstep_target,
59                               if_sn_id sn_pass);
60 extern int8_t ar_motor_next(void);
61
62 #endif /*__AR_MOTOR_H__*/
```

ソースファイル ar-motor.c (自動運転モーター制御: 実装)

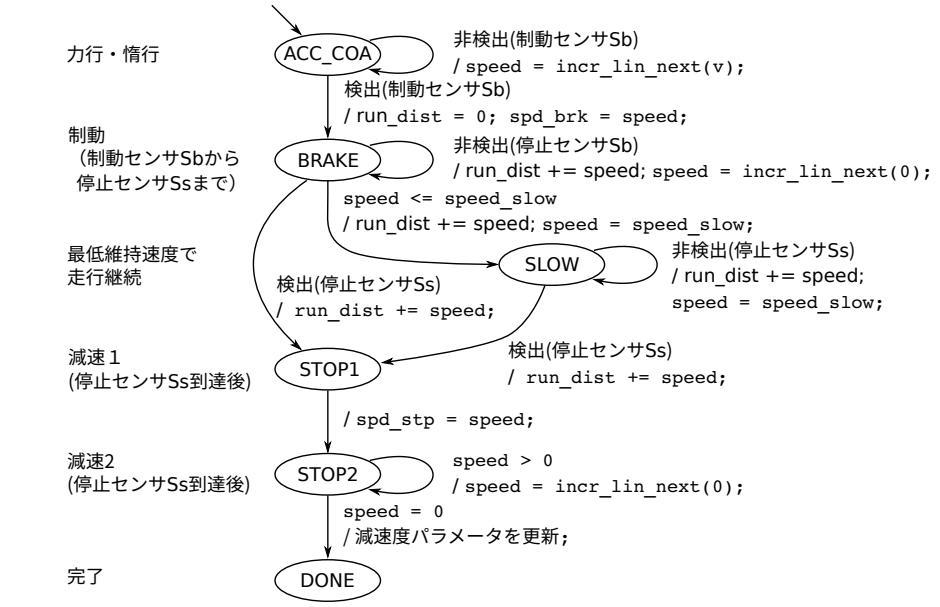
先述した 2 つの基本操作のセンサ停止走行 RUN-UNTIL(v,i,Sb,Ss) およびセンサ通過走行 RUN-TO(v,Sp) に関連する関数の実装を行います.

- ar_motor_init()
  センサ停止走行およびセンサ通過走行の初期化を行います.
- ar_motor_on()
  フィーダ出力をオンにします. これは遷移の動作部の記述で使用する記法 M(On) に対応する動作の実装です.
- ar_motor_off()
  フィーダ出力をオフにします. これは遷移の動作部の記述で使用する記法 M(Off) に対応する動作の実装です.
- ar_motor_run_until()
  これは遷移の動作部の記述で使用する記法 RunUntil(v,i,Sb,Ss) に対応する動作の実装です. センサ停止走行基本操作の引数を設定して, 有限状態遷移機械を初期化します.
- ar_motor_run_to()
  これは遷移の動作部の記述で使用する記法 RunTo(v,Sp) に対応する動作の実装です. センサ通過走行基本操作の引数を設定して, 有限状態遷移機械を初期化します.
- ar_motor_next()
  これは遷移の動作部の記述で使用する記法 MotorNext() に対応する動作の実装です. この関数内でセンサ停止走行およびセンサ通過走行の動作の実現を行います.

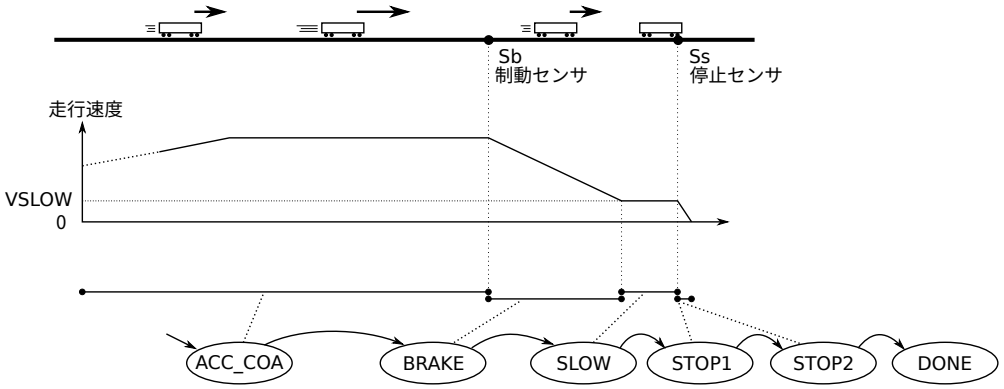
センサ停止走行およびセンサ通過走行の動作では, 力行・惰行・制動やセンサ検出による減速・停車など動作は少々複雑です. この複雑さへの対処法として図 29.28 および図 29.29 に示す通り, 有限状態遷移機械による動作のモデル化を行うことで設計を単純明快にします. なお力行・制動の際には第 6 章で紹介したブレゼンハムのアルゴリズムを使用します.

停止センサに到達したときには模型車両を停車しなくてはなりませんが, もし瞬時に走行速度をゼロにしたのでは滑らかな停車とはなりません. そこで減速度パラメータの適切さを以下の 3 通りに分類しています.

- 減速度パラメータの精度が十分な範囲に収まっている場合
  減速は十分にできており, 滑らかな停車ができている場合です. これは図 29.15 で示している理想制御の場合と最低維持速度・停車の場合に当たります. この場合には完全な停車が 1 秒後になるように減速度



(a) 有限状態機械

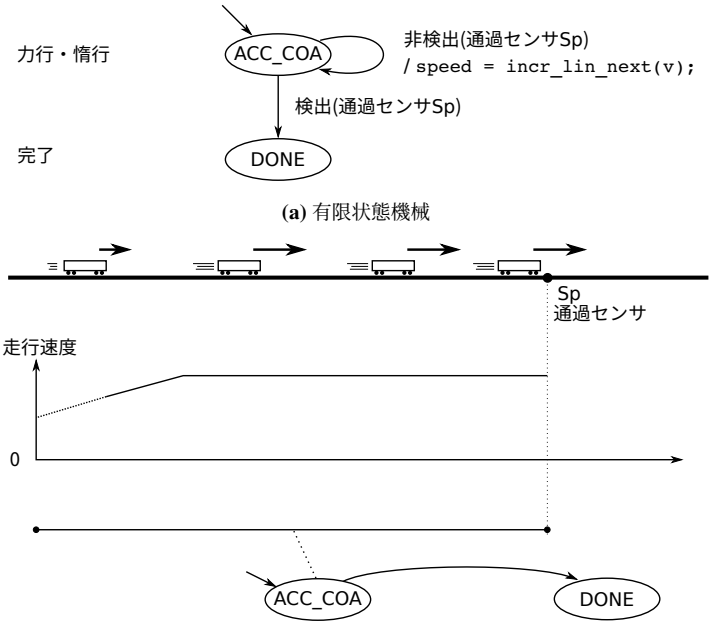


(b) 走行速度の変化と状態の遷移

図 29.28: センサ停止走行 RUN-UNTIL(v,i,Sb,Ss)

を設定して, 模型車両の走行を続けます. 十分に減速できていますから過走を 1 秒間行っても走行距離は僅かですし, 停車はすいぶん滑らかです. この 1 秒という値は記号定数 AR_TIME_TO_STOP で定義しています.

- 減速度パラメータの精度が許容可能な範囲に収まっている場合
  減速はあまり十分ではなく, 少々急な制動が必要な場合です. これは図 29.15 で示している急制動の場合に当たります. この場合には 0.25 秒後に完全な停車となるように減速度を設定して, 模型車両の走行を続けます. 減速はあまり十分できてはいませんが, 0.25 秒間の過走を行って走行距離は少しですし, 停車はまずまず滑らかです.
- 減速度パラメータの精度が許容可能を越えている場合
  減速はまったく不十分で, 緊急停止が必要な場合です. これは図 29.15 で示している緊急停車の場合に当たります. この場合には過走しないよう瞬時に走行速度をゼロにします. 停車はまったく滑らかではありませんが事故防止のためにはやむを得ません.



(a) 有限状態機械

(b) 走行速度の変化と状態の遷移

図 29.29: センサ通過走行 RUN-TO(v,Sp)

ソースファイル番号 29.22: ar-motor.c

```
1  /*-----------------------------------------------------
2   * モータ制御(自動運転用)
3   *-----------------------------------------------------*/
4  #include <xc.h>
5  #include <stdint.h>
6  #include "mcr.h"
7  #include "mcr-motor.h"
8  #include "io.h"
9  #include "pp.h"
10 #include "pp-const.h"
11 #include "ar.h"
12 #include "ar-est.h"
13 #include "incr-lin.h"
14
15 static void set_run_params(ar_speedstep_t spdstep_target,
16                            ar_decp_id decpi,
17                            uint16_t accel,
18                            if_sn_id sn_pass,
19                            if_sn_id sn_brake,
20                            if_sn_id sn_stop);
21
22 /* 初期化 */
23 void ar_motor_init(void)
24 {
25   ar_motor.speed = 0;
26   pp_motor_dir_set(DIR_NONE);
27   set_run_params(0, AR_DECP_ID_NULL, 0,
28                  IF_SN_NULL, IF_SN_NULL, IF_SN_NULL);
29 }
30
31 /* モータOn (常点灯On) */
32 void ar_motor_on(void)
33 {
34   pp_motor_dir_set(DIR_FWD);
35 }
36
37 /* モータOff (常点灯Off) */
```

```
38  void ar_motor_off(void)
39  {
40    pp_motor_dir_set(DIR_NONE);
41  }
42
43  /* 走行制御状態機械の走行パラメータ設定:
44   * 現在の走行速度から力行・惰行し、停止センサで停車 */
45  void ar_motor_run_until(ar_speedstep_t spdstep_target,
46                          ar_decp_id decpi,
47                          if_sn_id sn_brake,
48                          if_sn_id sn_stop)
49  {
50    set_run_params(spdstep_target, decpi, p_accel,
51                   IF_SN_NULL, sn_brake, sn_stop);
52  }
53
54  /* 走行制御状態機械の走行パラメータ設定:
55   * 現在の走行速度から力行・惰行し、指定センサまで走行(停車しない) */
56  void ar_motor_run_to(ar_speedstep_t spdstep_target,
57                       if_sn_id sn_pass)
58  {
59    set_run_params(spdstep_target, AR_DECP_ID_NULL, p_accel,
60                   sn_pass, IF_SN_NULL, IF_SN_NULL);
61  }
62
63  /* 走行制御状態機械の走行パラメータ設定 */
64  static void set_run_params(ar_speedstep_t spdstep_target,
65                             ar_decp_id decpi,
66                             uint16_t accel,
67                             if_sn_id sn_pass,
68                             if_sn_id sn_brake,
69                             if_sn_id sn_stop)
70  {
71    /* 走行パラメータを設定 */
72    incr_lin_init(ar_motor.speed);
73    ar_motor.state = AR_MOTOR_ACC_COA;
74    /* 目標速度 (p_vmax によるスケール前) */
75    ar_motor.speed_target = AR_SPEEDSTEP_SCALE(spdstep_target);
76    /* 減速度 */
77    if ((decpi != AR_DECP_ID_NULL) &&
78        (ar_decp[decpi].decel == 0)) {
79      ar_decp[decpi].decel = p_decel << 2;
80    }
81    ar_motor.decpi = decpi;
82    /* 加速度 */
83    ar_motor.accel = accel << 2;
84    /* その他各種パラメータ */
85    ar_motor.sn_pass  = sn_pass;
86    ar_motor.sn_brake = sn_brake;
87    ar_motor.sn_stop  = sn_stop;
88    ar_motor.spd_brake = 0;
89    ar_motor.spd_stop  = 0;
90    ar_motor.run_dist = 0;
91    ar_motor.snstop_reach = AR_SNSTOP_UNDEF;
92    /* センサ値をクリア */
93    if_sensor_read();
94    if_sensor_get(ar_motor.sn_pass);
95    if_sensor_get(ar_motor.sn_brake);
96    if_sensor_get(ar_motor.sn_stop);
97  }
98
99  /* 走行制御状態機械 */
100 int8_t ar_motor_next(void)
101 {
102   /* センサ値 */
103   int8_t det_pass  = if_sensor_get(ar_motor.sn_pass);
104   int8_t det_brake = if_sensor_get(ar_motor.sn_brake);
105   int8_t det_stop  = if_sensor_get(ar_motor.sn_stop);
106   /* 減速度パラメータ */
107   uint16_t decel = 0;
108   if (ar_motor.decpi != AR_DECP_ID_NULL) {
```

```
109       decel = ar_decp[(uint8_t)ar_motor.decpi].decel;
110     }
111     if (decel > AR_DECEL_MAX) {
112       decel = AR_DECEL_MAX;
113     }
114     /* 最低維持速度 */
115     uint16_t speed_slow = p_vslow;
116     /* 目標速度(p_vmaxによる補正後) */
117     uint16_t speed_target = 0;
118     /* 走行距離の更新値 */
119     uint32_t run_dist_next = 0;
120     /* 状態遷移 */
121     switch (ar_motor.state) {
122     case AR_MOTOR_ACC_COA:  /* 力行・惰行 */
123       /* 目標速度(p_vmaxによる補正後)を最低維持速度以上に補正 */
124       speed_target
125         = PP_CONV_SPEED_TO_MDUTY(ar_motor.speed_target, p_vmax);
126       if (speed_target < speed_slow) {
127         speed_target = speed_slow;
128       }
129       /* 走行速度 */
130       ar_motor.speed
131         = incr_lin_next(ar_motor.accel, SPEED_INC_DY, speed_target);
132       /* 事象検査 */
133       if ((ar_motor.sn_pass != IF_SN_NULL) &&
134           det_pass) { /* 通過センサへ到達 -> 完了(通過) */
135         ar_motor.state = AR_MOTOR_DONE;
136       } else if ((ar_motor.sn_brake != IF_SN_NULL) &&
137                  det_brake) { /* 制動センサへ到達 -> 制動開始 */
138         ar_motor.state = AR_MOTOR_BRAKE;
139         ar_motor.run_dist = 0;
140         ar_motor.spd_brake = ar_motor.speed;
141         incr_lin_init(ar_motor.speed);
142       }
143       break;
144     case AR_MOTOR_BRAKE: /* 制動 (制動センサから停止センサへ) */
145       /* 走行距離積算 */
146       run_dist_next = ar_motor.run_dist + ar_motor.speed;
147       if (ar_motor.run_dist < run_dist_next) { /* オーバーフロー対策 */
148         ar_motor.run_dist = run_dist_next;
149       }
150       /* 制動 */
151       ar_motor.speed = incr_lin_next(decel, SPEED_INC_DY, 0);
152       if (ar_motor.speed < speed_slow) { /* 速度低下 */
153         /* 最低維持速度で走行継続 */
154         ar_motor.speed = speed_slow;
155       }
156       /* 事象検査 */
157       if ((ar_motor.sn_stop != IF_SN_NULL) &&
158           det_stop) { /* 停止センサへ到達 -> 停車 */
159         ar_motor.state = AR_MOTOR_STOP1;
160         ar_motor.snstop_reach = AR_SNSTOP_REACH;
161       } else if (ar_motor.speed <= speed_slow) {
162         /* 最低維持速度で走行継続 */
163         ar_motor.state = AR_MOTOR_SLOW;
164         ar_motor.speed = speed_slow;
165       }
166       break;
167     case AR_MOTOR_SLOW: /* 最低維持速度で走行継続 */
168       /* 走行距離積算 */
169       run_dist_next = ar_motor.run_dist + ar_motor.speed;
170       if (ar_motor.run_dist < run_dist_next) { /* オーバーフロー対策 */
171         ar_motor.run_dist = run_dist_next;
172       }
173       /* 事象検査 */
174       if ((ar_motor.sn_stop != IF_SN_NULL) &&
175           det_stop) { /* 停止センサへ到達 -> 停車 */
176         ar_motor.state = AR_MOTOR_STOP1;
177         ar_motor.snstop_reach = AR_SNSTOP_UNREACH;
```

```
178       } else { /* 最低維持速度で走行継続 */
179         ar_motor.speed = speed_slow;
180       }
181       break;
182     case AR_MOTOR_STOP1: /* 減速1(停止センサ到達後) */
183       ar_motor.state = AR_MOTOR_STOP2;
184       ar_motor.spd_stop = speed_slow;
185       incr_lin_init(ar_motor.speed);
186       /* 減速度(最低維持速度から停車時の間) */
187       uint32_t z = 0;
188       if (ar_motor.speed < (speed_slow << 1)) {
189         /* 指定時間(AR_TIME_TO_STOP)で停止 */
190         z = (uint32_t)AR_TIME_TO_STOP * NRUNS_PER_SEC * SPEED_INC_DY;
191       } else if (ar_motor.speed < (speed_slow << 2)) {
192         /* 0.25 [Sec] で停止 */
193         z = (uint32_t)NRUNS_PER_SEC * SPEED_INC_DY >> 2;
194       } else {
195         /* 即座に停止 */
196         z = 0;
197       }
198       if (ar_motor.speed != 0) {
199         ar_motor.decel_stop = z / ar_motor.speed;
200       } else {
201         ar_motor.decel_stop = z;
202       }
203       break;
204     case AR_MOTOR_STOP2: /* 減速2(停止センサ到達後) */
205       /* 事象検査 */
206       if (ar_motor.speed > 0) { /* 減速 */
207         ar_motor.speed
208           = incr_lin_next(ar_motor.decel_stop, SPEED_INC_DY, 0);
209       } else { /* 完了(停車) */
210         if (ar_motor.snstop_reach == AR_SNSTOP_UNREACH) {
211           /* 停車センサ到達前に速度は最低維持速度へ低下: 減速度が大 */
212           ar_estimate_unreached();
213         } else if (ar_motor.snstop_reach == AR_SNSTOP_REACH) {
214           /* 速度が最低維持速度に低下する前に停車センサへ到達: 減速度が小 */
215           ar_estimate_reached();
216         }
217         ar_motor.state = AR_MOTOR_DONE;
218       }
219       break;
220     case AR_MOTOR_DONE: /* 完了(停車/通過) */
221       break;
222     }
223     return (ar_motor.state != AR_MOTOR_DONE);
224   }
```

上記のソースファイルでは式 29.5 に基づく減速度パラメータを計算するために $w_1, w_2, \sum_s w(s)$ を測定しています.

- ar_motor.spd_brake

  w_1 の値を保持します. 制動センサが模型車両を検出したときの PWM デューティ比です. 具体的には, 状態 AR_MOTOR_ACC_COA において ar_motor.speed の値を代入しています.

- ar_motor.spd_stop

  w_2 の値を保持します. 停止センサが模型車両を検出するときに取るべき値で, 最低維持速度 VSLOW の PWM デューティ比の値です. この値は可変抵抗器から読み取ることで得られます. 具体的には, 状態 AR_MOTOR_STOP1 において speed_slow (=p_vslow) の値を代入しています.

- ar_motor.run_dist

  $\sum_s w(s)$ の値を保持します. 制動センサが模型車両を検出してから停止センサが模型車両を検出するまでの間の PWM デューティ比の総和です. 具体的には, 状態 AR_MOTOR_と状態 AR_MOTOR_SLOW において

PWM デューティ比を累積しています.

ソースファイル ar-est.h (自動運転減速度予測: ヘッダ)

最適な減速度パラメータを式 29.5 に基づいて計算を行う関数のプロトタイプ宣言を行います.

ソースファイル番号 29.23: ar-est.h

```
1 #if !defined(__AR_EST_H__)
2 #define __AR_EST_H__
3
4 extern void ar_estimate_unreached(void);
5 extern void ar_estimate_reached(void);
6
7 #endif /*__AR_EST_H__*/
```

ソースファイル ar-est.c (自動運転減速度推測: 実装)

最適な減速度パラメータを式 29.5 に基づいて計算を行う関数の実装を行います. 以下のふたつの関数により理想的な減速度パラメータを推測します. 先述の図 29.27 も参照して下さい.

- ar_estimate_unreached()
  減速が強過ぎたために停止センサに到達する前に速度が低下して最低維持速度での走行となった場合にこの関数を使用します.
- ar_estimate_reached()
  減速が弱過ぎたために走行速度が高速のまま停止センサに到達した場合にこの関数を使用します.

ふたつの関数はいずれも過去の事例の値を更新しますが, もし今回の測定結果が過去の事例に矛盾すれば今回の方を優先することで過去の事例の値が矛盾しないようにします. なお今回の事例と過去の事例とが矛盾する原因としては以下のものが挙げられます.

- 物理的な動作のばらつき
  同一の走行パラメータで走行しても, 走行の度に走行時間等が厳密に同じであることはまずありません.
- 走行パラメータの変更
  装置の可変抵抗器が操作されて, 最高走行速度 VMAX や最低維持速度 VSLOW の値が変化する場合があります.

自動運転中に走行パラメータが変更されても, 本ファームウエアは常に減速度パラメータを更新して理想的な減速度パラメータへと自動的に追従する機能を持ちます.

理想的な減速度パラメータの計算を説明します. 式 29.5 が理想的な減速度パラメータ D^* の値を推測する式でした. 減速度は速度時間変化の傾きですから $D^* = \Delta_Y/\Delta_X$ とおくことにします. ソースファイル ar-motor.c では Δ_Y の値は記号定数 SPEED_INC_DY で固定しており, Δ_X の値のみを変化させています. そしてこのふたつの値 Δ_Y と Δ_X をブレゼンハムのアルゴリズムに与えて速度制御を行います. Δ_X の値の計算は式 29.5 に基づき以下の通り計算できます.

$$
\begin{aligned}
D^* = \Delta_Y/\Delta_X &\approx \frac{w_1^2 - w_2^2}{2\sum_{s=0}^{S-1} w(s)} \\
\therefore \Delta_X &\approx \frac{2\Delta_Y \sum_{s=0}^{S-1} w(s)}{w_1^2 - w_2^2}
\end{aligned}
\tag{29.6}
$$

この式で使用されている $w_1, w_2, \sum_s w(s)$ の値はそれぞれ ar_motor.spd_brake, ar_motor.spd_stop, ar_motor.run_dist に値が保持されており, ソースファイル ar-motor.c の中で測定しています. Δ_X の値はこれらの値を参照して計算します.

ソースファイル番号 29.24: ar-est.c

```
/*------------------------------------------------------
 * 減速度の更新値見積と自動学習(自動運転用)
 *------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "mcr-motor.h"
#include "io.h"
#include "pp.h"
#include "pp-const.h"
#include "ar.h"
#include "ar-est.h"
#include "incr-lin.h"

/* 3つのパラメータ:
 *   dc = .decel             減速度(今回使用した値)
 *   du = .decel_unreached   減速度(停止センサに到達しなかった事例)
 *   dr = .decel_reached     減速度(停止センサに到達した事例)
 *   (値が小さい: すぐに停止, 値が大きい: ゆっくり停止)
 * 3つの値の間で以下の関係を保つ:
 *   decel_unreached <= decel_current <= decel_reached
 */

/* 最低維持速度で走行しながら停止センサに達した場合 */
void ar_estimate_unreached(void)
{
  ar_decp_id decpi = ar_motor.decpi;
  if (decpi == AR_DECP_ID_NULL) {
    return;
  }
  /* 現在値と過去の事例 */
  uint16_t dc = ar_decp[decpi].decel;
  uint16_t du = ar_decp[decpi].decel_unreached;
  uint16_t dr = ar_decp[decpi].decel_reached;
  /* 過去の事例を更新 */
  du = dc;
  /* 過去と今回の事例が互いに矛盾していれば補正 */
  if ((dr != 0) && (du >= dr)) {
    dr = du + (du >> 3);
  }
  /* 見積 */
  uint16_t dc_next = 0; /* 次回の減速度 */
  uint32_t sb2
    = (uint32_t)ar_motor.spd_brake * ar_motor.spd_brake;
  uint32_t ss2
    = (uint32_t)ar_motor.spd_stop  * ar_motor.spd_stop;
  if (sb2 > ss2) { /* 推測式に基づく推定 */
    dc_next
      = ((uint32_t)2*SPEED_INC_DY*ar_motor.run_dist) / (sb2 - ss2);
  } else { /* 値を1.25倍にして再試行 */
    dc_next = dc + (dc >> 2) + 1;
  }
  /* 過去の事例を基に補正 */
  if (dc_next <= dc) {
    if (dr != 0) {
      dc_next = dc + ((dr - dc) >> 1);
    } else {
      dc_next = dc + (dc >> 2);
    }
  }
  if (dc_next == 0) {
    dc_next = 1;
  }
```

```
64    /* 更新 */
65    ar_decp[decpi].decel           = dc_next;
66    ar_decp[decpi].decel_unreached = du;
67    ar_decp[decpi].decel_reached   = dr;
68  }
69
70  /* 最低維持速度を越えたまま停止センサに達した場合 */
71  void ar_estimate_reached(void)
72  {
73    ar_decp_id decpi = ar_motor.decpi;
74    if (decpi == AR_DECP_ID_NULL) {
75      return;
76    }
77    /* 現在値と過去の事例 */
78    uint16_t dc = ar_decp[decpi].decel;
79    uint16_t du = ar_decp[decpi].decel_unreached;
80    uint16_t dr = ar_decp[decpi].decel_reached;
81    /* 過去の事例を更新 */
82    dr = dc;
83    /* 過去と今回の事例が互いに矛盾していれば補正 */
84    if ((du != 0) && (du > dr)) {
85      du = dr - (dr >> 3);
86    }
87    /* 見積 */
88    uint16_t dc_next = 0; /* 次回の減速度 */
89    uint32_t sb2
90      = (uint32_t)ar_motor.spd_brake * ar_motor.spd_brake;
91    uint32_t ss2
92      = (uint32_t)ar_motor.spd_stop  * ar_motor.spd_stop;
93    if (sb2 > ss2) { /* 推測式に基づく推定 */
94      dc_next
95        = ((uint32_t)2*SPEED_INC_DY*ar_motor.run_dist) / (sb2 - ss2);
96    } else { /* 値を0.75倍にして再試行 */
97      dc_next = dc - (dc >> 2) + 1;
98    }
99    /* 過去の事例を基に補正 */
100   if (dc <= dc_next) {
101     if (du != 0) {
102       dc_next = dc - ((dc - du) >> 2);
103     } else {
104       dc_next = dc - (dc >> 3);
105     }
106   }
107   if (dc_next == 0) {
108     dc_next = 1;
109   }
110   /* 更新 */
111   ar_decp[decpi].decel           = dc_next;
112   ar_decp[decpi].decel_unreached = du;
113   ar_decp[decpi].decel_reached   = dr;
114 }
```

ソースファイル ar-stay.h (自動運転停車関数: ヘッダ)

指定された時間の待機を行う関数のプロトタイプ宣言と記号定数の定義を行います.

ソースファイル番号 29.25: ar-stay.h

```
1  #if !defined(__AR_STAY_H__)
2  #define __AR_STAY_H__
3
4  /* 待機時間係数の最大値 */
5  #define AR_STAYTIME_MAX       32
6  #define AR_STAYTIME_DEFAULT   32
7  typedef uint8_t  ar_staytime_t;
8
9  extern void    ar_stay_start(ar_staytime_t st);
10 extern void    ar_stay_abs_start(uint8_t t);
11 extern int8_t  ar_stay_next(void);
```

```
12
13  #endif /*__AR_STAY_H__*/
```

ソースファイル ar-stay.c (自動運転停車関数: 実装)

指定された時間の待機を行う関数の実装を行います. 待機時間は回路上の可変抵抗器 STAYTIME により設定は可変とし, 待機時間の指定はこの設定に対する相対的な値で行います. 待機時間の 100% に相当する相対値は 32 とします. 例えば相対値 16 では待機時間の 50% に相当する待機時間です. これとは別に, 相対値ではなく絶対値で待機時間を指定する関数も用意します.

- ar_stay_start()
  指定した相対時間の待機を開始します. これは遷移の動作部の記述で使用する記法 StayStart() に対応する動作の実装です.
- ar_stay_abs_start()
  指定した絶対時間の待機を開始します. 引数の値は 100 ミリ秒単位とします.
- ar_stay_next()
  待機時間の経過の完了を検査します. まだ完了していなければ真を, 完了していれば偽を返します. これは遷移の動作部の記述で使用する記法 StayNext() に対応する動作の実装です.

ソースファイル番号 29.26: ar-stay.c

```
1   /*------------------------------------------------------
2    * 停止(相対時間で指定) / 待機(絶定時間で指定)
3    *------------------------------------------------------*/
4   #include <xc.h>
5   #include <stdint.h>
6   #include "pp.h"
7   #include "pp-const.h"
8   #include "ar.h"
9   #include "ar-stay.h"
10
11  /* 停止時間 [mSec] */
12  static uint16_t t_stay_msec = 0;
13
14  /* 停止開始(相対時間) */
15  void ar_stay_start(ar_staytime_t st)
16  {
17    if (st > AR_STAYTIME_MAX) {
18      st = AR_STAYTIME_MAX;
19    }
20    /* 停止時間: 0 [mSec] ... 51150 [mSec] */
21    /*   (p_staytim = 0 ... 1023) */
22    t_stay_msec =
23      ((uint32_t)p_staytim * st * 50) / (uint32_t)AR_STAYTIME_MAX;
24  }
25
26  /* 停止開始(絶対時間) */
27  void ar_stay_abs_start(uint8_t t)
28  {
29    t_stay_msec = (uint32_t)t * 100;
30  }
31
32  /* 停止継続 */
33  int8_t ar_stay_next(void)
34  {
35    if (t_stay_msec < RUN_INTERVAL) {
36      return 0; /* 完了 */
37    }
38    t_stay_msec -= RUN_INTERVAL;
39    return 1; /* 継続中 */
```

```
}
```

ソースファイル mcr.h (MCR・周辺モジュール: ヘッダ)

マイコンの初期化等の関数のプロトタイプ宣言やマクロ定義を行います.

ソースファイル番号 29.27: mcr.h

```
#if !defined(__MCR_H__)
#define __MCR_H__

#define _XTAL_FREQ 32000000 /* 32MHz */

extern void System_Init(void);

#endif /*__MCR_H__*/
```

ソースファイル mcr.c (MCR・周辺モジュール: 実装)

マイコンの初期化等を行います.

ソースファイル番号 29.28: mcr.c

```
/*-------------------------------------------------------
 * MCR設定&周辺モジュール (PIC16F18877)
 *-------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "mcr.h"
#include "mcr-motor.h"
#include "adc.h"
#include "i2c.h"

/*-----------------------------------------------*/
/* MCR初期化 (FOSC=32MHz) */
void System_Init(void)
{
  /* 発振器 */
  OSCCON1 = 0x10; /* HFINTOSC with 2x PLL */
  OSCCON3 = 0x00;
  OSCEN   = 0x00;
  OSCFRQ  = 0x05; /* 16MHz */
  OSCTUNE = 0x00;
  while (PLLR == 0) {
    ;
  }
  /* I/Oピン */
  LATA   = 0x00;
  LATB   = 0x00;
  LATC   = 0x00;
  LATD   = 0x00;
  LATE   = 0x00;
  ANSELA = 0xFF; /* ANA7,ANA6,ANA5,ANA4,ANA3,ANA2,ANA1,ANA0 */
  ANSELB = 0x00; /* - */
  ANSELC = 0x01; /* ANC0 */
  ANSELD = 0x00; /* - */
  ANSELE = 0x00; /* - */
  TRISA  = 0xFF; /* RA7,RA6,RA5,RA4,RA3,RA2,RA1,RA0 */
  TRISB  = 0xDF; /* (RB7,RB6),RB4,RB3,RB2,RB1,RB0 */
  TRISC  = 0x3D; /* RC5,RC4,RC3,RC2,RC0 */
  TRISD  = 0xF3; /* RD7,RD6,RD5,RD4,RD1,RD0 */
  TRISE  = 0x08; /* (RE3) */
  WPUA   = 0x00;
  WPUB   = 0xC0; /* (RB7,RB6) */
  WPUC   = 0x00; /* - */
  WPUD   = 0x00; /* - */
  WPUE   = 0x08; /* (RE3) */
  ODCONA = 0x00;
```

```
46    ODCONB = 0x00;
47    ODCONC = 0x00;
48    ODCOND = 0x00;
49    ODCONE = 0x00;
50    int8_t ogie = GIE;
51    GIE = 0;
52    PPSLOCK = 0x55;
53    PPSLOCK = 0xAA;
54    PPSLOCKbits.PPSLOCKED = 0;
55    RC1PPS = 0x05;      /* RC1 <- CWG1A */
56    RB5PPS = 0x06;      /* RB5 <- CWG1B */
57    RC4PPS = 0x15;      /* RC4 <- SDA1  */
58    RC5PPS = 0x14;      /* RC5 <- SCL1  */
59    SSP1DATPPS = 0x14; /* RC4 -> SDA1  */
60    SSP1CLKPPS = 0x15; /* RC5 -> SCL1  */
61    PPSLOCK = 0x55;
62    PPSLOCK = 0xAA;
63    PPSLOCKbits.PPSLOCKED = 1;
64    GIE = ogie;
65    /* 周辺モジュール */
66    ADC_Init();
67    I2C_Init(I2C_BAUD_RATE_SSPxADD_1MHZ(_XTAL_FREQ));
68    MCR_Motor_Init();
69  }
70
71  /*-------------------------------------------*/
72  /* 割込ハンドラ */
73  __interrupt() void isr(void)
74  {
75    if ((TMR1IE != 0) && (TMR1IF != 0)) {
76      extern void TMR1_isr(void);
77      TMR1IF = 0;
78      TMR1_isr();
79    }
80  }
81
82  /*-------------------------------------------*/
83  /* CONFIG1 */
84  #pragma config FEXTOSC  = OFF
85  #pragma config RSTOSC   = HFINTPLL
86  #pragma config CLKOUTEN = OFF
87  #pragma config CSWEN    = ON
88  #pragma config FCMEN    = ON
89  /* CONFIG2 */
90  #pragma config MCLRE    = OFF /* MCLRをI/Oピン */
91  #pragma config PWRTE    = OFF
92  #pragma config LPBOREN  = OFF
93  #pragma config BOREN    = ON
94  #pragma config BORV     = LO
95  #pragma config ZCD      = OFF
96  #pragma config PPS1WAY  = ON
97  #pragma config STVREN   = ON
98  #pragma config DEBUG    = OFF
99  /* CONFIG3 */
100 #pragma config WDTCPS   = WDTCPS_31
101 #pragma config WDTE     = ON  /* WDT有効, SWDTEN無視 */
102 #pragma config WDTCWS   = WDTCWS_7
103 #pragma config WDTCCS   = SC
104 /* CONFIG4 */
105 #pragma config WRT      = OFF
106 #pragma config SCANE    = available
107 #pragma config LVP      = OFF /* 低電圧書込不可 */
108 /* CONFIG5 */
109 #pragma config CP       = OFF
110 #pragma config CPD      = OFF
```

課題

1. 図 29.3b で示したエンドレストラックプランにおける基本操作の列と模型列車の位置, 走行速度, 通過する車両センサの関係を, 図 29.18 に倣って図示して下さい.

2. 図 29.3c で示したリバーストラックプランにおける基本操作の列と模型列車の位置, 走行速度, 通過する車両センサの関係を, 図 29.18 に倣って図示して下さい.
3. 自動運転を実現しているソースファイル `fsm-shuttle.c` (585 ページ), `fsm-endless.c` (587 ページ), `fsm-reverse.c` (588 ページ) それぞれを改造して, 模型列車が駅を出発する時に発車ベルを鳴らせるようにアクセサリ制御出力のオン・オフの追加を行って下さい.
4. エンドレストラックプランの自動運転を実現しているソースファイル `fsm-endless.c` (587) を改造して, 駅の通過と停車を交互に行うようにして下さい.

第 30 章

スクリプトによる
自動運転システム

本章ではユーザ作成の自動運転スクリプトを EEPROM に書き込んで自動運転を実現するシステムを紹介します. ハードウエアは第 29 章で紹介したものをそのまま使用して, ファームウエアの入れ替えにより実現します. 一連の作業の全体像を図 30.1 に示します.

本章で紹介するファームウエアの特徴は以下の通りです.

- 独自の自動運転パターンをプログラム可能
  自動運転パターンを EEPROM に書き込む方式を採用して, 独自の自動運転パターンが実現可能です.
- 自動運転パターンの記述に特化した独自の自動運転スクリプト言語
  汎用性の高いスクリプト言語を使用するのではなく, 本装置の自動運転パターンの記述に特化した独自のスクリプト言語を新たに設計して, 自動運転スクリプトの記述の容易さとファームウエアにおける実装の容易さの両立を行います.

本章で学ぶ内容は以下の通りです.

- 自動運転スクリプトのプログラミング法

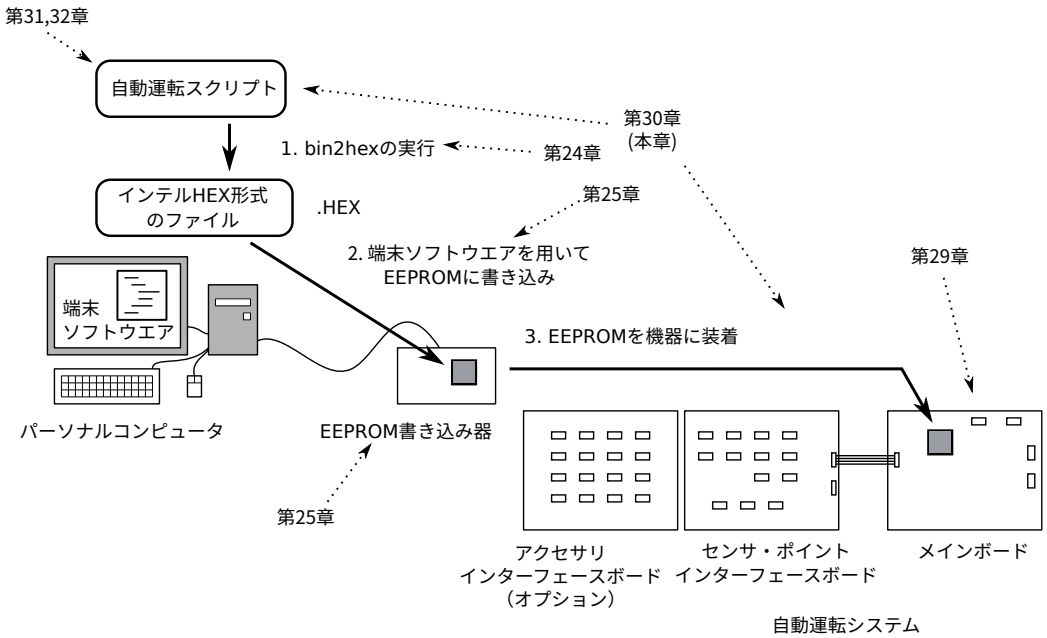


図 30.1: スクリプトによる自動運転システムの利用のための作業手順

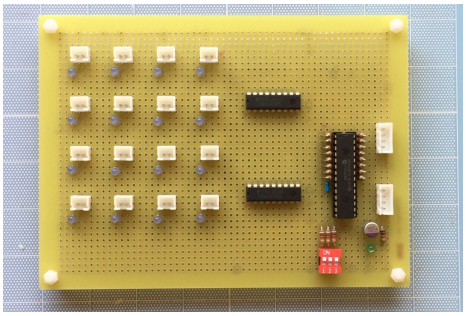


図 30.2: アクセサリインターフェースボードの試作例

- 自動運転スクリプト言語の構文解析法
- 自動運転スクリプト言語の実行機構

本章での自動運転に必要な構成要素は以下の通りです.

- メインボード: 1 枚 (第 29 章)
- センサ・ポイントインターフェースボード: 1 枚以上の必要数 (第 29 章)
- アクセサリボード: 必要数 (本章)
- 車両センサ: 必要数 (第 16 章)
- ポイント切替器: 必要数 (第 12 章および第 14 章)

なお, アクセサリ機能を使用した自動運転スクリプトの場合でも必ずしもアクセサリボードを接続する必要はありません. 無接続の場合はアクセサリ出力制御は無視して自動運転を行います.

EEPROM に自動運転スクリプトを書き込む際には以下のものが必要です.

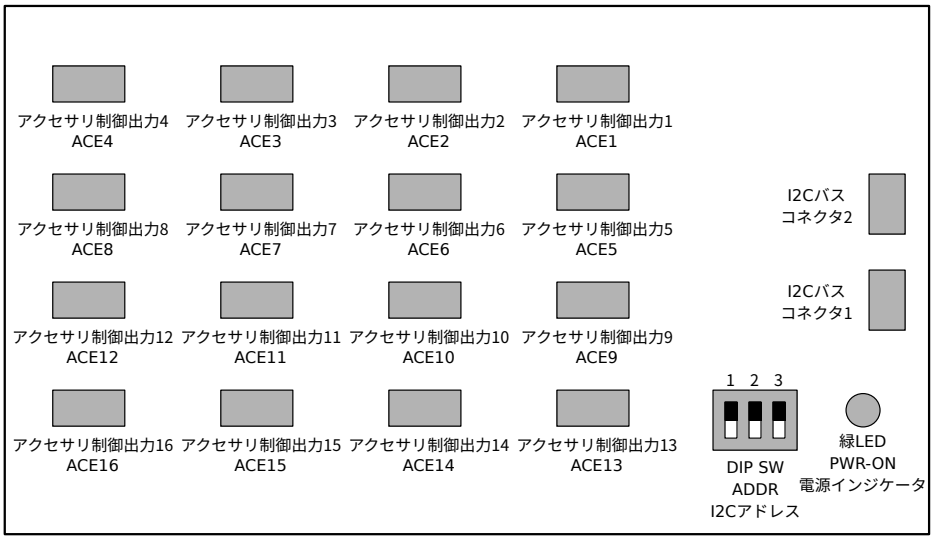
- インテル HEX 形式への変換ソフトウエア bin2hex (第 24 章)
- EEPROM 書き込み器 (第 25 章)

マイコンのファームウエアに独自の運転パターンを組み込むにはファームウエアの書き換えが必要であり, 特定の開発環境である MPLAB X IDE や PICkit4 等が必要なため 30 年後にも利用可能とは思えません. EEPROM 方式では第 25 章で紹介した自作の書き込み器を使用することで永きに渡って自動運転パターンの変更が可能です.

30.1　使用方法

本ファームウエアの使用方法を説明します. 最初に行う準備は以下の通りです. 自動運転パターンを変更する場合には, 新たな自動運転パターンに対して再度以下の手順を実行して下さい.

1. EEPROM の準備
   EEPROM には容量が 8K バイトの 24FC64 を使用します. 自動運転スクリプトが大きくて 8K バイトを越える場合を想定して, 24FC256 (32K バイト) や 24FC512 (64K バイト) も使用可能なファームウエアにしてあります. なお本書で紹介する自動運転スクリプトはいずれも 24FC64 に収まる大きさです.
2. 自動運転スクリプトの用意
   パーソナルコンピュータ上に自動運転スクリプトのファイルを用意して下さい. 自動運転スクリプトは本章の第 30.3 節を参考に自作したり, 本章の第 30.4 節および第 31 章で紹介するいくつかの例を使用す



(a) アクセサリインターフェースボード

図 30.3: アクセサリインターフェースボード部品の配置例と名称

表 30.1: 致命エラー時の LED の点滅パターン

点滅回数	エラー理由	備考
1	不正値 ※	文のパラメータが不正あるいは範囲外
2	未定義指示子 ※	存在しない指示子の使用
3	未定義文 ※	存在しない文の使用
4	不正構文 ※	タイプミスなどの構文エラー
5	未定義ラベル ※	ジャンプ先ラベルが未定義
6	スクリプト終端到達 ※	スクリプト終端指示子へ実行が到達
7	未実装機能	DIP スイッチ設定誤り
8	EEPROM 装着	EEPROM が装着されていない
9	インターフェースボード無装着	センサ・ポイントボードもアクセサリボードも接続がない
10	内部エラー	ファームウエアバグの検出

※自動運転スクリプトでのエラー

ると良いでしょう.

3. EEPROM への自動運転スクリプトの書き込み
   第 24 章で紹介したソフトウエア bin2hex を使用して自動運転スクリプトを HEX ファイルにします. 次に第 25 章で紹介した EEPROM 書き込み器を使用して自動運転スクリプトを EEPROM に書き込みます.
4. EEPROM の装着
   EEPROM を本装置の EEPROM 用 IC ソケットに装着します.
5. DIP スイッチの設定
   EEPROM 内の自動運転スクリプトを実行するように, 表 29.1 (552 ページ) に従って DIP スイッチを設定します.

6. 配線の接続
   センサ・ポイントボード, アクセサリボード, フィーダ, センサ, ポイントそれぞれを適切に配線して下さい. これは第 29 章での場合と同一です.

設定パラメータの調整, 自動運転モード・手動運転モードの切り替え, 手動運転モードでの操作, 自動運転の実行方法などは, 第 29 章で説明した通りです. なおエラー時の LED 点滅は表 30.1 で示す通りで, 自動運転スクリプトに誤りがあった場合には LED にエラーが表示されます.

30.2　回路図

本章ではアクセサリインターフェースボードを紹介します. これは必ずしも必要ではなく, アクセサリ制御出力を充実したい方が作製して下さい. 回路図を図 30.4 に, 部品表を表 30.2 に, それぞれ示します.

表 30.2: 【部品表】 自動往復運転システム / アクセサリインターフェースボード

記号	数量	部品名
U2	1	MCP23017-E/SP　I/O エキスパンダ [I-09486]
U1, U3	2	TBD62083APG　トランジスタアレイ [I-10669]
D1, D2, D3, D4, D5, D6, D7, D8, D10, D11, D12, D13, D14, D15, D16, D17	16	LED 青　[I-06413]
D9	1	LED 緑　[I-06413]
C1	1	電解コンデンサ 100μF　耐圧 16V 以上 [P-08290]
C2	1	セラミックコンデンサ 0.1μF　[P-00090]
R1, R2, R3, R4, R5, R6, R7, R8, R9, R13, R14, R15, R16, R17, R18, R19, R20	17	抵抗 2.7kΩ　1/4W [R-25272]
R10, R11, R12	3	抵抗 10kΩ　1/4W [R-25103]
	1	IC ソケット 28 ピン　[P-01339]
SW1	1	DIP SW 3P　[P-09992]
J3, J4, J5, J6, J7, J8, J9, J10, J11, J12, J13, J14, J15, J16, J17, J18	16	基板用コネクタ 2 ピン　JST XH コネクタ B2B-XH-A(LF) [C-12247]
J1, J2	2	基板用コネクタ 4 ピン　JST XH コネクタ B4B-XH-A(LF) [C-12249]
	2	コネクタハウジング 4 ピン　JST XH コネクタ XHP-4 [C-12257]
	8	コネクタハウジング用コンタクト　JST XH コネクタ SXH-001T-P0.6 [C-12264]
		ほか接続用ケーブル・基板・配線材等

アクセサリインターフェースボードは I/O エクスパンダ MCP23017 のポートをトランジスタアレイを介してアクセサリ制御出力とすることで十分な電流の駆動を可能としています. またアクセサリ出力の状態が分かるように動作モニタ用の LED を備えています. アクセサリインターフェースボードにも I^2C アドレス設定用の DIP スイッチを取り付けています.

30.3　自動運転スクリプト言語の構文

図 30.5 に自動運転スクリプト言語の構文を示します. このスクリプト言語は本書の自動運転装置での自動走行に特化したものです. このような特定の応用タスク向けに設計されたコンピュータ言語はドメイン固有言語

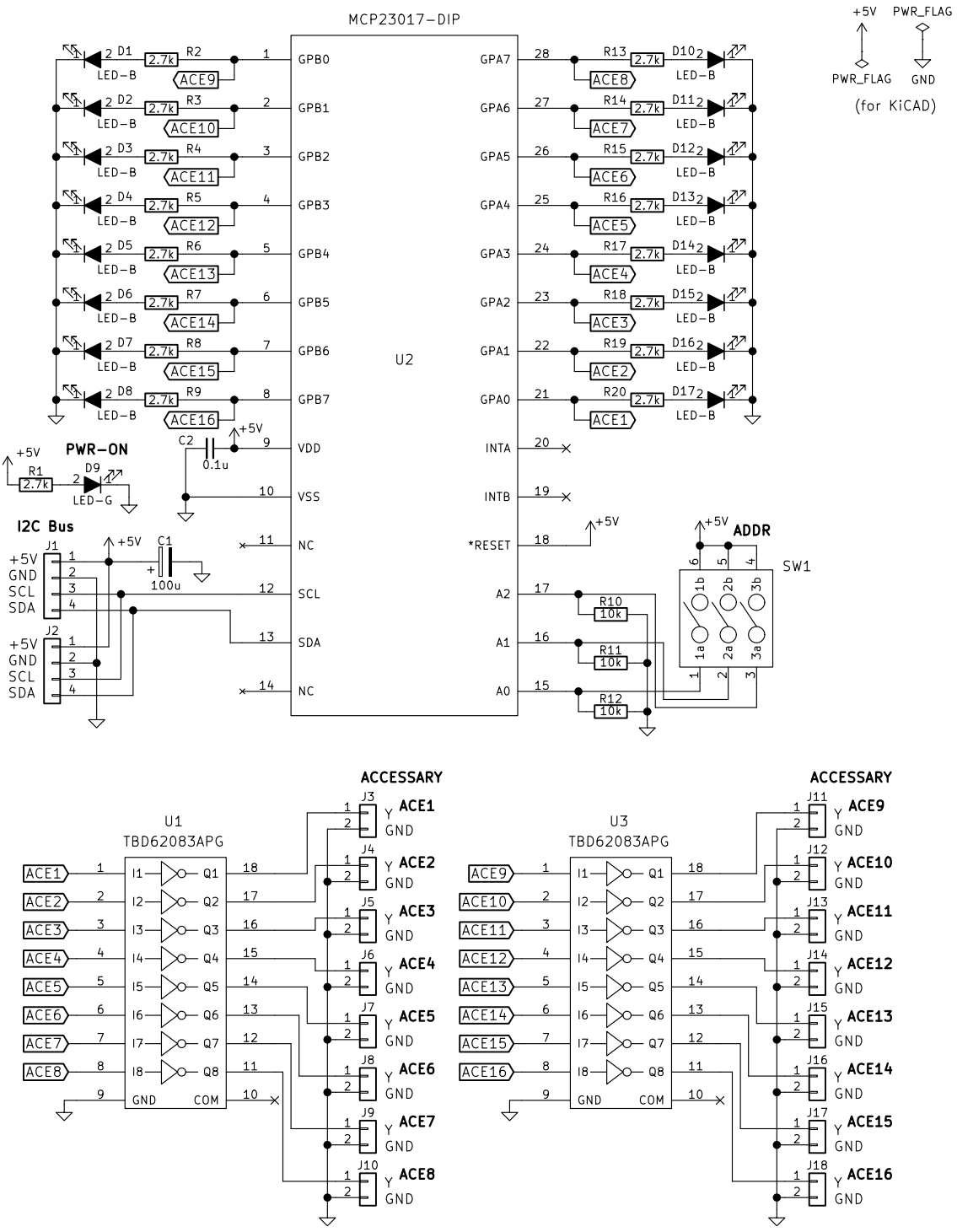


図 30.4: 【回路図】 自動往復運転システム / アクセサリインターフェースボード

自動運転スクリプト言語 (1/3)

- テキストファイルで記述します. 改行は CR と CRLF のどちらでも構いません. 使用する文字は ASCII 文字 (いわゆる半角文字) ですが, コメント部分にのみ例外として日本語 (いわゆる全角文字) も使用できます.
- 行の 1 文字目は指示子を表し, #, :, /, ␣ の 4 種の文字のいずれかでなければなりません. もしこれに違反している場合は未定義指示子エラーとなります. 指示子の出現順序は問いません.
  ※なお␣は 1 文字以上の空白文字またはタブ文字を表します.
- 指示子
  - コメント指示子 #
    構文: #⟨ コメント ⟩
    行末までコメントを書き入れることができます.
  - ラベル定義指示子 :
    構文: :⟨ ラベル ⟩:
    ⟨ ラベル ⟩ は 0 から 63 までの整数値です. ラベル 0 は定位置状態 HOME を表す特別なラベルです.
  - スクリプト終端指示子 /
    構文: /
    自動運転スクリプトの終端を表します.
  - 文実行指示子 ␣
    構文: ␣⟨ 文 ⟩;
- 文
  文実行指示子のあとに文を記述します. 文の種類によりいくつかのパラメータを持ち, いずれの文もセミコロン; で終わります. 最後のパラメータからセミコロンの間には任意の数の空白を置くことができます. セミコロンから改行までの間の文字は無視しますので, セミコロンより後ろにコメントを書くことが可能です.
  - 無条件ジャンプ文 J
    構文: J␣⟨ ラベル ⟩;
    指定ラベルにジャンプします.
  - 確率的ジャンプ文 ?
    構文: ?␣⟨ ラベル ⟩␣⟨ 確率 ⟩;
    指定した確率で指定ラベルにジャンプします. ⟨ 確率 ⟩ は 0 から 10 までの整数値で, 0.1 を乗じた値を確率値とします.
  - センサ条件ジャンプ文 $
    構文: $␣⟨ ラベル ⟩␣⟨ センサ番号 ⟩␣⟨ オン・オフ ⟩;
    指定した番号の車両センサの検出値が指定の条件に一致すれば, 指定ラベルにジャンプします. ⟨ センサ番号 ⟩ は車両センサの番号で, 1 以上の整数値です. ⟨ オン・オフ ⟩ は検出の有無で, それぞれ 1 と 0 で表します.

図 30.5: 自動運転スクリプト言語の構文 (1/3)

自動運転スクリプト言語 (2/3)

- 文 (つづき)
  - モータ・常点灯文 M

    構文: M ␣ ⟨ 出力 ⟩ ;

    モータ出力および常点灯の出力のオン・オフを設定します. ⟨ オン・オフ ⟩ は出力の設定で, オンは 1, オフは 0 の値です.
  - 線路極性文 D

    構文: D ␣ ⟨ フィーダ番号 ⟩ ␣ ⟨ 極性 ⟩ ;

    指定したフィーダの線路電圧の出力を指定の極性にします. ⟨ フィーダ ⟩ はフィーダ出力の系統番号を指定する数値で, 1 または 2 の値です. ⟨ 極性 ⟩ はフィーダ出力の極性を指定する数値で, 前進は 1 で後進は 2 の値です.
  - センサ停止走行文 R

    構文: R ␣ ⟨v⟩ ␣ ⟨i⟩ ␣ ⟨ センサ番号 Sb⟩ ␣ ⟨ センサ番号 Ss⟩ ;

    センサ停止走行 RUN-UNTIL(v,i,Sb,Ss) を実行します. ⟨v⟩ は惰行速度で, 0 から 32 までの整数値です. 実際の速度は可変抵抗器 VMAX で指定した値との相対値で, 値が 32 のときを 100% とします. ⟨i⟩ は減速度番号で, 1 から 128 までの整数値です. ⟨ センサ番号 Sb⟩ は制動センサとして使用する車両センサ番号です. ⟨ センサ番号 Ss⟩ は停止センサとして使用する車両センサ番号です.
  - センサ通過走行文 X

    構文: X ␣ ⟨v⟩ ␣ ⟨ センサ番号 Sp⟩ ;

    センサ通過走行 RUN-TO(v,Sp) を実行します. ⟨v⟩ は惰行速度で, 0 から 32 までの整数値です. 実際の速度は可変抵抗器 VMAX で指定した値との相対値で, 値が 32 のときを 100% とします. ⟨ センサ番号 Sp⟩ は通過センサとして使用する車両センサ番号です.
  - 相対時間待機文 S

    構文: S ␣ ⟨t⟩ ;

    時間待機 STAY-TIME(t) を実行します. ⟨t⟩ は待機時間で 0 から 32 までの整数値です. 実際の待機時間は可変抵抗器 STAYTIME で指定した値との相対値で, 値が 32 のときを 100% とします.
  - 絶対時間待機文 Z

    構文: Z ␣ ⟨t⟩ ;

    時間待機 STAY-TIME(t) を実行します. ⟨t⟩ は待機時間で 0 から 200 までの整数値です. 実際の待機時間は ⟨t⟩ で与えられる整数値に 0.1 を乗じた秒数です. 待機時間の最大値は 20 秒 (= 200×0.1) です.

図 30.5: 自動運転スクリプト言語の構文 (2/3)

自動運転スクリプト言語 (3/3)

- 文 (つづき)
  - ポイント文 T

    構文: T ␣ ⟨ ポイント番号 ⟩ ␣ ⟨ 開通方向 ⟩ ;

    指定したポイントを指定した開通方向に切り替えます. ⟨ ポイント番号 ⟩ はポイント番号で, 1 以上の整数値です. ⟨ 開通方向 ⟩ はポイントの開通方向で, 定位は 1, 反位は 2 の値です.
  - アクセサリ文 A

    構文: A ␣ ⟨ アクセサリ番号 ⟩ ␣ ⟨ オン・オフ ⟩ ;

    指定した番号のアクセサリ出力のオン・オフを設定します. ⟨ アクセサリ番号 ⟩ はアクセサリ制御出力の番号で, 1 以上の整数値です. ⟨ オン・オフ ⟩ は出力の設定で, オンは 1, オフは 0 の値です.
- 自動運転スクリプトの実行

  スクリプト中の実行する行の番号を現在行と呼ぶことにします. 現在行を 1 に設定して自動運転スクリプトの実行を開始します. ただし文実行指示子以外の指示子の場合には現在行を 1 増加して, 文実行指示子以外の指示子を無視します. 現在行が文実行指示子であればその文を実行し, 現在行を 1 増加します. ジャンプ文により実行する行を変更する場合には, ジャンプ先のラベルが定義されている行を現在行に設定します.
- エラー
  - 不正値: 文のパラメータが不正あるいは範囲外の場合のエラーです. 具体的にはパラメータ ⟨v⟩, ⟨i⟩, ⟨t⟩, ⟨ 確率 ⟩, ⟨ センサ番号 ⟩, ⟨ オン・オフ ⟩, ⟨ フィーダ ⟩, ⟨ 極性 ⟩, ⟨ ポイント番号 ⟩, ⟨ 開通方向 ⟩ が該当します.
  - 未定義指示子: 存在しない指示子が使用された場合のエラーです.
  - 未定義文: 存在しない文が使用された場合のエラーです.
  - 不正構文: 構文に合致しない場合のエラーです. ラベル定義指示子の構文に合致しない, 文のパラメータが足りないあるいは多い, 文の終了を表すセミコロンがない, などの場合が該当します.
  - 未定義ラベル: ジャンプ先に指定しているラベルが定義されていない場合のエラーです.
  - スクリプト終端: 自動運転スクリプトの実行中にスクリプト終端指示子へ実行が到達した場合のエラーです. スクリプト終端指示子には決して実行が到達しない無限ループでなければなりません.

図 30.5: 自動運転スクリプト言語の構文 (3/3)

(Domain-Specific Language; DSL) とも総称されます. 一度に走行できる模型車両を 1 編成に限定していることを前提に設計した言語ですので無駄な汎用性はまったくなく実に単純明解です.

30.4　自動運転スクリプトの例

図 30.5 で紹介した自動運転スクリプト言語の構文に基づいて記述した自動運転スクリプトの例を紹介します. ここで紹介するものは, 第 29 章で紹介した C 言語による実装の自動運転プログラムを自動運転スクリプト言語で記述したものです. 同一の自動運転パターンに対する C 言語および自動運転スクリプト言語による実装を比較しながら, 自動運転スクリプトの書き方を理解して下さい.

30.4.1　単線トラックプラン

トラックプランは図 29.3a (547 ページ) で, C 言語による実装はソースファイル `fsm-shuttle.c` (585 ページ) で示したものに対応しています. ラベル 0 は定位置状態 HOME を表す特別なラベルで, 模型列車が定位置に戻ってきたときの状態です. 自動運転終了時には定位置に停車させることで次回の自動運転をその位置のままで開始できて便利です. 自動運転スクリプトは以下の通りです.

ソースファイル番号 30.1: `ato-shuttle.txt`

```
#----------------------------
# 単線往復
#----------------------------
#   減速度番号割当
#     1: 東駅
#     2: 西駅
#   センサ番号割当
#     S1: 東駅, 停止
#     S2: 東駅, 制動
#     S3: 西駅, 制動
#     S4: 西駅, 停止
#----------------------------
# 西駅, 停車(東向)
  # 初期化
  D 1 1;        # 東行 (トラック1: 順方向)
  M 1;          # モータ&常点灯On
# ■ 繰返点
:1:
  # 西駅→東駅, 走行
  R 32 1 2 1;  # 速度 32(=100%), 減速度番号 1, 制動:S2, 停止:S1
  # 東駅, 停車
  D 1 2;        # 西行 (トラック1: 逆方向)
  S 32;         # 待機時間32(=100%)
  # 東駅→西駅, 走行
  R 32 2 3 4;  # 速度 32(=100%), 減速度番号 2, 制動:S3, 停止:S4
# ■ 定位置 (西駅)
:0:
  # 西駅, 停車
  D 1 1;        # 東行  (トラック1: 順方向)
  S 32;         # 待機時間 32(=100%)
  # 繰返し
  J 1;
# データ終了
/
#----------------------------
```

30.4.2　エンドレストラックプラン

トラックプランは図 29.3b (547 ページ) で示し, C 言語による実装はソースファイル `fsm-endless.c` (587 ページ) で示したものに対応しています. 自動運転スクリプトは以下の通りです.

ソースファイル番号 30.2: ato-endless.txt

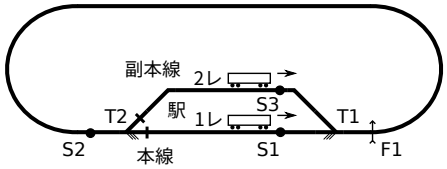
```
#----------------------------
# 単線エンドレス
#----------------------------
#   減速度番号割当
#     1: 駅
#   センサ番号割当
#     S1: 駅, 停止
#     S2: 駅, 制動
#----------------------------
# 駅, 停車(東向)
# 初期化
  D 1 1;        # 東行 (トラック1: 順方向)
  M 1;          # モーター&常点灯On
# ■ 繰返点
:1:
  # 駅を出発, 走行
  R 32 1 2 1;  # 速度32(100%), 減速度番号 1, 制動:S2, 停止:S1
# ■ 定位置 (駅)
:0:
  # 駅, 停車
  S 32;         # 待機時間 32(=100%)
  # 繰返し
  J 1;
# データ終了
/
#----------------------------
```

30.4.3　リバーストラックプラン

トラックプランは図 29.3c (547 ページ) で示し, C 言語による実装はソースファイル fsm-reverse.c (588 ページ) で示したものに対応しています. 自動運転スクリプトは以下の通りです.

ソースファイル番号 30.3: ato-reverse.txt

```
#----------------------------
# リバース線
#----------------------------
#   減速度番号割当
#     1: 駅
#   センサ番号割当
#     S1: 駅, 制動
#     S2: 駅, 停止
#     S3: リバース区間進入検知
#   分岐器番号割当
#     T1: リバース
#----------------------------
# 駅, 停車(東向)
# 初期化
  D 1 1;        # 東行 (トラック1: 順方向)
  D 2 1;        # 東行 (トラック2: 順方向)
  M 1;          # モータ&常点灯On
  T 1 1;        # 分岐器1: 定位
# ■ 繰返点
:1:
  # 駅を出発, 走行
  X 32 3;       # 速度32(=100%), 通過: S3
  # リバース線処理: 給電極性反転&ポイント切替
  D 1 2;        # 西行 (トラック1: 逆方向)
  T 1 2;        # 分岐器1: 反位
  R 32 1 1 2;  # 速度32(=100%), 減速度番号 1, 制動:S1, 停止:S2
# ■定位置 (駅)
:0:
  # 駅, 停車
```



列車退避を行う単線の駅で, 本線と副本線があります. 模型列車は 2 編成あり, 列車番号をそれぞれ 1 レ (第 1 列車で優等列車), 2 レ (第 2 列車) として区別します. いずれの模型列車も走行を開始する位置は駅です.

1. 1 レは駅を出発します. レイアウトを 1 周して駅に停車しますが, ときどきランダムに駅を通過して 2 レを追い越します.
2. 1 レは駅に停車すれば, 次は 2 レが駅を出発します. 2 レがレイアウトを 1 周すれば駅に停車します. 以上を繰り返します.

なお 1 レ, 2 レ共に駅を発車する際にはアクセサリ制御出力 1 番を 3 秒間オンにしますので, 発車ベルなどに使用可能です.

図 30.6: エンドレス退避駅トラックプラン

```
30    D 1 1;          # 東行 (トラック1: 正方向)
31    S 32;           # 待機時間 32(=100%)
32    # 駅, 停車終了
33    T 1 1;          # 分岐器: 定位
34    # 繰返し
35    J 1;
36  # データ終了
37  /
38  #----------------------------
```

30.4.4　エンドレス退避駅トラックプラン

エンドレス退避駅トラックプランを図 30.6 に示します. 自動運転スクリプトは以下の通りです. この自動運転スクリプトでは, ランダムな駅の通過, アクセサリ制御出力, 2 編成の選択的な走行を実現しています. トラックプランでは必ず選択式のポイントを使用してください. ポイントの切り替えにより走行する模型列車への電源供給を切り替えているためです. 停止センサは S1 と S3 で, 制動センサは S2 です. ここで制動センサと停止センサの組として S2 と S1 および S2 と S3 の 2 組を使用しており, 両方の組で S2 が共用されている点に注意して下さい. このようにひとつの車両センサを複数の目的に使用することも可能です.

この自動運転スクリプトの特長はランダムな動作が含まれているところです. 駅を通過をランダムに決定するスクリプトの考え方は, 以下の一連のステップのループで構成します.

1. 模型列車 1 レは駅に停車しています.
2. ランダムに以下のステップ 3 と 4 のいずれかへ実行を移します.
3. 駅を通過する場合

   センサ通過走行 RUN-TO(32, S2), RUN-TO(32, S1) を順に実行して駅を通過します. そしてステップ 2 にジャンプして, 駅通過のループ動作を繰り返します.
4. 駅を通過しない場合

   センサ停止走行 RUN-UNTIL(0, 1, S2, S1) を実行して駅のホームで停車します. 駅通過のループ動作は

これで終了します.

ソースファイル番号 30.4: ato-sta1-endless1.txt

```
#----------------------------
# エンドレス退避駅 / 2列車
#----------------------------
#   減速度番号割当
#     1: 駅停車(本線/1レ)
#     2: 駅停車(副本線/2レ)
#   センサ番号割当
#     S1: 駅(本線/1レ),    停止
#     S2: 駅(共通),        制動
#     S3: 駅(副本線/2レ), 停止
#   分岐器番号割当
#     T1: 駅東  (本線側が定位)
#     T2: 駅西  (本線側が定位)
#----------------------------
# 駅, 停車 1レ
# 駅, 停車 2レ
# 初期化
  D 1 1;        # 東行 (トラック1: 順方向)
  M 1;          # モータ&常点灯On
  # 進路の構成
  T 1 1;        # 分岐器1: 定位
  T 2 1;        # 分岐器2: 定位
  Z 10;         # 1秒待機
# ■ 繰返点
:1:
# ■ 列車 1レ
  # 出発ベル(3秒)
  A 1 1;        # アクセサリ1 On
  Z 30;         # 3秒待機
  A 1 0;        # アクセサリ1 Off
  # ランダムに駅を通過
:2:
  ? 3 5;        # 確率0.5
    # 1レ 駅→  駅通過
    X 32 2;     # 速度 32(=100%), 通過:S2
    X 32 1;     # 速度 32(=100%), 通過:S1
    J 2;
:3:
  # 1レ 駅→駅, 走行
  R 32 1 2 1;  # 速度 32(=100%), 減速度番号 1, 制動:S2, 停止:S1
  # 停車
  S 32;         # 待機時間32(=100%)
# ■ 列車 2レ
  # 進路の構成
  T 1 2;        # 分岐器1: 反位
  T 2 2;        # 分岐器2: 反位
  Z 10;         # 1秒待機
  # 出発ベル(3秒)
  A 1 1;        # アクセサリ1 On
  Z 30;         # 3秒待機
  A 1 0;        # アクセサリ1 Off
  # 2レ  駅→駅, 走行
  R 32 2 2 3;  # 速度 32(=100%), 減速度番号 2, 制動:S2, 停止:S3
# ■ 定位置 (駅)
:0:
  # 進路の構成
  T 1 1;        # 分岐器1: 定位
  T 2 1;        # 分岐器2: 定位
  # 停車
  S 32;         # 待機時間32(=100%)
# 繰返し
```

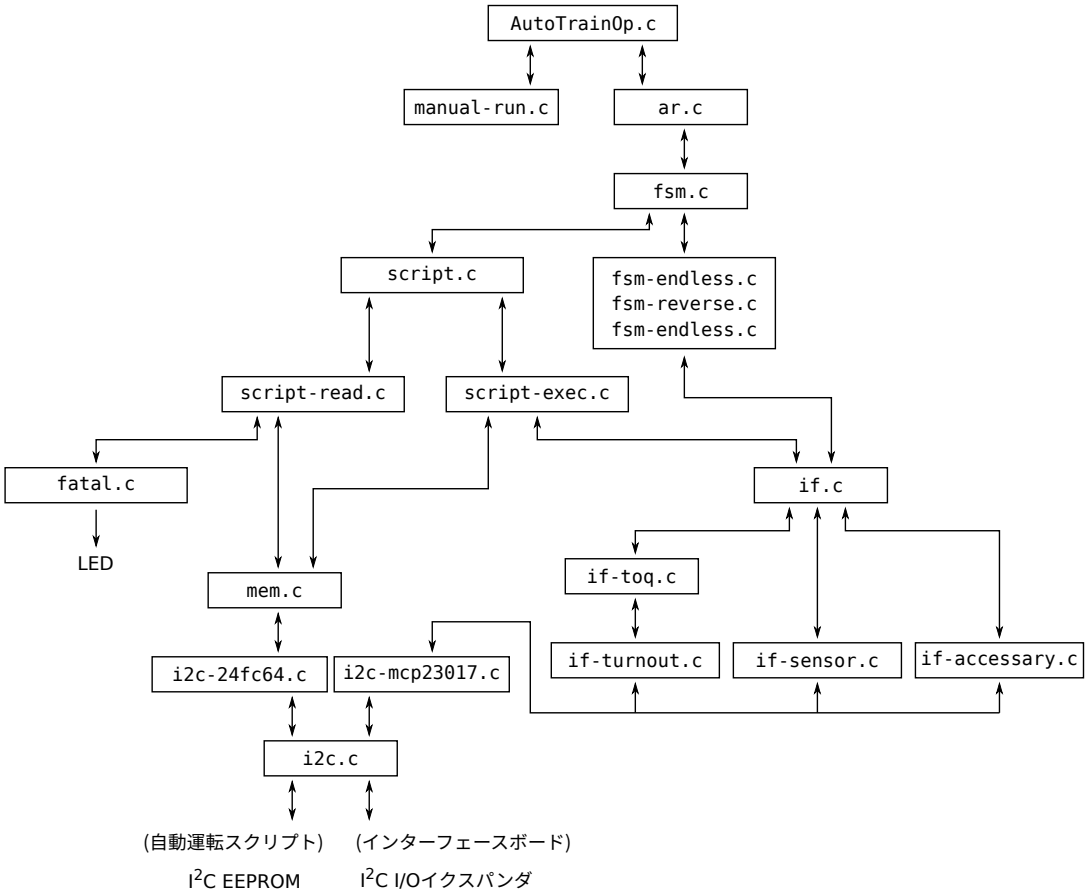


図 30.7: ファームウエアにおける主要ソースファイル間の関係

```
62      J 1;
63  # データ終了
64  /
65  #-----------------------------
```

30.5　ファームウエア

本ファームウエアを構成するソースファイル一覧を以下に示します. 図 30.7 に主要なソースファイル間の関係を示します.

表 30.3: 【ソースファイル一覧】 EEPROM プログラム型自動運転システム

初	頁	番号	ファイル名	内容	行数
	577	29.2	AutoTrainOp.c	主ソースコード	175
	580	29.3	fatal.h	エラー表示: ヘッダ	24
	581	29.4	fatal.c	エラー表示: 実装	48
	582	29.5	manual-run.c	手動運転: 実装	120
	584	29.6	fsm.h	自動運転有限状態機械: ヘッダ	10
	584	29.7	fsm.c	自動運転有限状態機械: 実装	30
	585	29.8	fsm-shuttle.c	自動運転有限状態機械 (単線): 実装	105

(つづく)

(つづき)

初	頁	番号	ファイル名	内容	行数
	587	29.9	fsm-endless.c	自動運転有限状態機械 (エンドレス): 実装	76
	588	29.10	fsm-reverse.c	自動運転有限状態機械 (リバース): 実装	101
	591	29.12	if.h	インターフェースボード: ヘッダ	99
	592	29.13	if.c	インターフェースボード: 実装	90
	374	22.16	if-toq.h	ポイント切替動作キュー: ヘッダ	12
	392	23.2	if-toq.c	インターフェースボード・ポイント切替動作キュー: 実装	122
	594	29.14	if-sensor.c	インターフェースボード・センサ部: 実装	113
	596	29.15	if-turnout.c	インターフェースボード・ポイント部: 実装	61
	597	29.16	if-accessary.c	インターフェースボード・アクセサリ部: 実装	128
	599	29.17	ar.h	自動運転: ヘッダ	47
	600	29.18	ar.c	自動運転: 実装	226
	605	29.19	ar-decp.h	自動運転減速度パラメータ: ヘッダ	21
	605	29.20	ar-decp.c	自動運転減速度パラメータ: 実装	18
	606	29.21	ar-motor.h	自動運転モーター制御: ヘッダ	62
	609	29.22	ar-motor.c	自動運転モーター制御: 実装	224
	613	29.23	ar-est.h	自動運転減速度予測: ヘッダ	7
	614	29.24	ar-est.c	自動運転減速度予測: 実装	114
	615	29.25	ar-stay.h	自動運転停車関数: ヘッダ	13
	616	29.26	ar-stay.c	自動運転停車関数: 実装	40
✓	636	30.5	script.h	スクリプト言語: ヘッダ	79
✓	638	30.6	script.c	スクリプト言語: 実装	109
✓	640	30.7	script-read.c	スクリプト構文解析部: 実装	272
✓	645	30.8	script-exec.c	スクリプト実行エンジン: 実装	149
	189	10.1	pp.h	パワーパック関数宣言: ヘッダ	46
	172	9.2	pp-const.h	パワーパック定数: ヘッダ	17
	191	10.3	pp-misc.c	自動加減速制御: 実装	111
	196	10.6	pp-sb.h	低速ブースト機能: ヘッダ	8
	198	10.7	pp-sb.c	低速ブースト機能: 実装	171
	394	23.3	i2c.h	I^2C インターフェース関数: ヘッダ	25
	394	23.4	i2c.c	I^2C インターフェース関数: 実装	111
	396	23.5	i2c-mcp23017.h	MCP23017 アクセス関数: ヘッダ	53
	397	23.6	i2c-mcp23017.c	MCP23017 アクセス関数: 実装	109
	456	25.17	i2c-24fc64.h	24FC64 アクセス関数: ヘッダ	20
	457	25.18	i2c-24fc64.c	24FC64 アクセス関数: 実装	133
	525	28.9	mem.h	EEPROM メモリアクセス: ヘッダ	24
	526	28.10	mem.c	EEPROM メモリアクセス: 実装	133
	129	6.1	incr-lin.h	線形増加アルゴリズム: ヘッダ	9

(つづく)

(つづき)

初	頁	番号	ファイル名	内容	行数
	130	6.2	incr-lin.c	線形増加アルゴリズム: 実装	78
	148	8.2	random.h	疑似乱数: ヘッダ	7
	149	8.3	random.c	疑似乱数: 実装	26
	574	29.1	io.h	I/O ピン割付: ヘッダ	81
	617	29.27	mcr.h	MCR・周辺モジュール: ヘッダ	8
	617	29.28	mcr.c	MCR・周辺モジュール: 実装	110
	175	9.6	mcr-motor.h	モーター制御周辺モジュール: ヘッダ	25
	177	9.7	mcr-motor.c	モーター制御周辺モジュール: 実装	98
	117	5.5	adc.h	ADC モジュール: ヘッダ	9
	118	5.6	adc.c	ADC モジュール: 実装	76

計 4183 行 (うち初出 609 行, 既出 3574 行)

第 29 章の自動往復運転システムで紹介したファームウエアでは自動運転パターンを C 言語の形で書いていました. 自動運転パターンの実行はセンサ停止走行 RUN-UNTIL(v,i,Sb,Ss), センサ通過走行 RUN-TO(v,Sp), 時間待機 STAY-TIME(t) の 3 つの基本操作を順次呼び出すだけの簡潔な基本構造をしており, この構造は図 29.25 で示した抽象化状態遷移図として抽象化できることを説明しました. 本章のファームウエアでは動作の列をデータの列として表現・記述して, データを逐次解釈・実行する機構を作成します. これを実現するプログラムの骨子は以下の通りで, 関数 script_exec() の中で実現しています.

```
1  【ソースファイル AutoTrainOp.c】
2
3  void main(void)
4  {
5    初期化;
6    auto_run_start();
7  }
8
9  【ソースファイル ar.c】
10   --- パワーパックのソースコードの構造に類似している点に注意.
11
12 ar_state = AR_STATE_INIT;  /* 有限状態機械の状態 */
13
14 void auto_run_start(void)
15 {
16   自動運転開始の押しボタンスイッチの押下を待機;
17   全ポイントを定位に切り替え;
18   for(;;) {
19     if_sensor_read();               /* センサの読み込み */
20     ar_state = fsm_next(ar_state);  /* 有限状態機械の動作 */
21     if_toq_next();                  /* ポイント切り替え */
22     ar_param_read();                /* 可変抵抗器の設定の読み込み */
23     md = モータPWMデューティ比;
24     pp_sb_duty_set(md);
25   }
26 }
27
```

```
【ソースファイル script.c】
  --- 手動による走行速度やポイントの切り替えをスクリプトが代わりに行う.

fsm_next(ar_state)
{
  switch (ar_state) {
  case AR_STATE_INIT: /* 初期状態 */
    ar_state = AR_STATE_RUNNING;
    スクリプト実行準備; /* 構文検査, ラベル定義等 */
    break;
  case AR_STATE_HOME: /* 模型車両が定位置にある */
    ar_state = AR_STATE_RUNNING;
    break;
  case AR_STATE_RUNNING: /* 自動運転スクリプト実行状態 */
    ar_state = script_exec();
    break;
  }
  return ar_state;
}

【ソースファイル script-exec.c】
  --- スクリプト実行有限状態機械の実現.
  --- 走行速度の変化などはここを起点にして行う.

script_st.state = SCRIPT_STATE_IDLE;  /* 実際にはscript.cで変数宣言 */

script_exec()
{
  switch (script_st.state) {
  case SCRIPT_STATE_IDLE: /* 実行途中の文がない */
    EEPROMから次に実行すべき文を読み出す;
    /* 読み出した文の種別によりscript_st.state が適宜設定される */
    break;
  case SCRIPT_STATE_xxx:    文xxxの実行; break;
  case SCRIPT_STATE_yyy:    文yyyの実行; break;
  case SCRIPT_STATE_zzz:    文zzzの実行; break;
    ... /* 文の実行に際しては基本操作の関数等を適宜呼び出す */
        /* script_st.state の更新により状態遷移を行う */
  }
}
```

自動運転パターンは基本操作の関数の順次呼び出しであると見做すことができます.

- 第 29 章での C 言語のプログラムでは, C 言語のプログラムコードでの記述により基本操作の関数の呼び出しの順序を決めて自動運転パターンを実現しています.
- 上記の関数 script_exec() では, EEPROM 内のデータでの記述により基本操作の関数の呼び出しの順序を決めて自動運転パターンを実現しています.

自動運転スクリプトの実行機構の基本的な考え方を説明します.

- 第 29 章において C 言語で記述した自動運転プログラムで使用している基本操作を最小実行単位に分解します. 実はこれは既に ar_motor_run_until() や ar_motor_next() などの C 言語の関数で実現して

おり, これらが最小実行単位の例です.

- 自動運転パターンにおいて逐次実行すべき基本操作の列をデータの列の形で表現して EEPROM に記憶します.
- 自動運転の際には EEPROM から実行すべき基本操作をひとつ読み出して実行します. 実行の細かな制御のために有限状態機械を使用し, その状態は script_st.state に保持します. この有限状態機械のことを「スクリプト実行有限状態機械」と呼びます.

ソースファイル script.h (スクリプト言語: ヘッダ)

スクリプト実行有限状態機械の状態, スクリプト言語での指示子と文に対する記号定数の定義のほか, EEPROM から読み出したスクリプト 1 行に対する構文解析結果を保持する構造体 script_st_s を定義します.

ソースファイル番号 30.5: script.h

```
1  #if !defined(__SCRIPT_H__)
2  #define __SCRIPT_H__
3
4  /* 実行過程 */
5  #define SCRIPT_PASS_1  1  /* ラベル定義と文法検査 */
6  #define SCRIPT_PASS_2  2  /* ラベル参照検査 */
7  #define SCRIPT_PASS_3  3  /* スクリプト実行 */
8
9  /* ラベル表 (ラベルは整数値で表現) */
10 #define N_LABELS          64   /* 1..63: ユーザー用 */
11 #define LABEL_ID_HOME      0   /* 0:     HOME状態専用 */
12 #define LABEL_UNDEFINED    0xFFFF  /* ラベル未定義 */
13 extern uint16_t script_label_memaddr[];  /* ラベル表 */
14 extern uint16_t script_label_lineno[];   /* 行番号 */
15
16 /* 命令実行状態 */
17 enum script_state_e {
18   SCRIPT_STATE_IDLE = 0,
19   SCRIPT_STATE_NOP,
20   SCRIPT_STATE_END,              /* 指示子: スクリプト終端 */
21   SCRIPT_STATE_COMMENT_FETCHED,  /* 指示子&文: コメント */
22   SCRIPT_STATE_LABEL_FETCHED,    /* 指示子: ラベル定義 */
23   SCRIPT_STATE_JUMP_FETCHED,         /* 文: 無条件ジャンプ */
24   SCRIPT_STATE_JUMP_RAND_FETCHED,    /* 文: 確率的ジャンプ */
25   SCRIPT_STATE_JUMP_SENS_FETCHED,    /* 文: センサ条件ジャンプ */
26   SCRIPT_STATE_MOTORON_FETCHED,      /* 文: モーター&常点灯 */
27   SCRIPT_STATE_TRACKDIR_FETCHED,     /* 文: 線路極性 */
28   SCRIPT_STATE_TURNOUT_FETCHED,      /* 文: ポイント転換 */
29   SCRIPT_STATE_TURNOUT_W_FETCHED,    /* 文: ポイント待機 */
30   SCRIPT_STATE_RUNUNTIL_FETCHED,     /* 文: センサまで走行&停止 */
31   SCRIPT_STATE_RUNTO_FETCHED,        /* 文: センサまで走行 */
32   SCRIPT_STATE_RUN_EXEC,
33   SCRIPT_STATE_ACCESARY_FETCHED,     /* 文: アクセサリ */
34   SCRIPT_STATE_STAY_FETCHED,         /* 文: 停止時間(相対時間) */
35   SCRIPT_STATE_STAY_ABS_FETCHED,     /* 文: 停止時間(絶対時間) */
36   SCRIPT_STATE_STAY_EXEC,
37 };
38
39 /* スクリプト 文構文解析結果 */
40 #define SCRIPT_MAX_N_PARAMS   4
41 struct script_st_s {
42   enum script_state_e state;
43   uint8_t p[SCRIPT_MAX_N_PARAMS];
44   uint16_t lineno;
45 };
46 extern struct script_st_s script_st;
47
48 /* スクリプト 指示子/文 */
```

```
49 #define SCRIPT_CH_END           '/' /* 指示子:スクリプト終端 */
50 #define SCRIPT_CH_LABEL         ':' /* 指示子:ラベル定義 */
51 #define SCRIPT_CH_COMMENT       '#' /* 指示子/文:コメント */
52 #define SCRIPT_CH_JUMP          'J' /* 文:無条件ジャンプ */
53 #define SCRIPT_CH_JUMP_RAND     '?' /* 文:確率的ジャンプ */
54 #define SCRIPT_CH_JUMP_SENS     '$' /* 文:センサ条件ジャンプ */
55 #define SCRIPT_CH_MOTORON       'M' /* 文:モーター・常点灯 */
56 #define SCRIPT_CH_TRACKDIR      'D' /* 文:線路極性 */
57 #define SCRIPT_CH_TURUNOUT      'T' /* 文:ポイント転換 */
58 #define SCRIPT_CH_TURUNOUT_W    'U' /* 文:ポイント待機 */
59 #define SCRIPT_CH_RUNUNTIL      'R' /* 文:センサ停止走行 */
60 #define SCRIPT_CH_RUNTO         'X' /* 文:センサ通過走行 */
61 #define SCRIPT_CH_ACCESARY      'A' /* 文:アクセサリ */
62 #define SCRIPT_CH_STAY          'S' /* 文:相対時間待機 */
63 #define SCRIPT_CH_STAY_ABS      'Z' /* 文:絶対時間待機 */
64 #define SCRIPT_CH_TERMINATOR    ';' /* 文終了 */
65
66 /* 自動運転状態機械での状態 */
67 #define AR_STATE_RUNNING   10
68
69 /* 待機時間の最大値 */
70 #define AR_SLEEP_MAX      200 /* 0.1 秒単位 (-> 最大20秒) */
71
72 extern void             script_read(uint8_t pass);
73 extern void             script_intern_labels(void);
74 extern void             script_verify(void);
75 extern enum ar_state_e  script_exec(void);
76 extern void             script_rewind(void);
77 extern int8_t           script_eof(void);
78
79 #endif /*__SCRIPT_H__*/
```

ソースファイル script.c (スクリプト言語: 実装)

自動運転スクリプト実行機構の全体的な制御を行う関数群より成るソースファイルです.

- script_init()
  EEPROM アクセスの初期化, ラベル定義, スクリプト検証を行います. 本ファームウエアでは EEPROM アクセスの実装には f-24fc64.c (453 ページ) を使用しており, このソースコードでは 24FC64 (8K バイト), 24FC256 (32K バイト), 24FC512 (64K バイト) が使用可能でした. そのため本装置のファームウエアでも EEPROM の容量は 24FC64, 24FC256, 24FC512 のいずれでも使用可能にします. そこでこれらの中で最大容量の 24FC512 を指定して mem_init() を呼び出します.
- script_intern_label()
  ラベル定義を行う関数で, script_init() から呼び出されます. EEPROM 内のスクリプトを 1 行づつ読み込んで, 関数 script_read() の中でラベル表に登録します. ラベル表は, ラベル定義が行われている EEPROM 内のメモリアドレスにより構成されています. またスクリプトのデバッグの目的でスクリプト内での行番号も記憶します.
- script_verify()
  スクリプト終端指示子への到達可能性の検証を行います. もしスクリプト終端指示子へ到達すると実行の継続が不可能となりますので, スクリプトの実行がスクリプト終端指示子へ到達する可能性の有無を検証します.
- fsm_script_next()
  自動運転スクリプトの実行を行います. この関数では初期化と定位置状態 HOME を取扱い, 実際の実行は関数 script_exec() の中で行います.

自動運転スクリプトの検証を自動運転の開始時に模型列車が走行する前に行っていることに注意して下さい. 模型列車が走行してから自動運転スクリプトに誤りが見つかり自動運転が強制終了してしまうというのは実に興醒めです. 軽微なタイプミスですら実際に自動運転を行わなければ誤りが発見できないというのも修正作業に手間がかかり非効率です. そこで本ファームウエアでは可能な限り事前に検証を行うことで自動運転スクリプトのデバッグ効率を高めます.

ソースファイル番号 30.6: script.c

```
/*------------------------------------------------------
 * トラックプラン: スクリプト (EEPROM内データ)
 *------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "ar.h"
#include "fsm.h"
#include "mem.h"
#include "script.h"
#include "fatal.h"

/* ラベル表 (ラベルのEEPROMアドレス) */
uint16_t script_label_memaddr[N_LABELS];
uint16_t script_label_lineno[N_LABELS];
/* スクリプト文の解析結果 */
struct script_st_s script_st;

/*------------------------------------------------------
 * 初期化
 *------------------------------------------------------*/
void script_init(void)
{
  /* EEPROM接続検査 */
  if (!mem_probe_device()) {
    fatal(AR_ERROR_NO_EEPROM);
  }
  mem_init(MEM_TYPE_24FC512);
  /* ラベル登録 */
  script_intern_labels();
  /* スクリプト検証(静的検査) */
  script_verify();
}

/*------------------------------------------------------
 * ラベル登録
 *------------------------------------------------------*/
void script_intern_labels(void)
{
  /* ラベル表をクリア */
  uint8_t lid;
  for (lid = 0; lid < N_LABELS; lid++) {
    script_label_memaddr[lid] = LABEL_UNDEFINED;
    script_label_lineno[lid]  = 0;
  }
  /* PASS1: ラベル定義と構文検査 */
  script_rewind();
  script_st.state = SCRIPT_STATE_IDLE;
  while (!script_eof() &&
         (script_st.state != SCRIPT_STATE_END)) {
    script_read(SCRIPT_PASS_1);
  }
}

/*------------------------------------------------------
 * スクリプト検証(静的検査)
 *------------------------------------------------------*/
void script_verify(void)
{
  /* PASS2: スクリプト検証 */
```

```
60    /*   参照ラベル定義の検査 */
61    /*   実行時でのスクリプト終端への到達可能性の検査 */
62    script_rewind();
63    script_st.state = SCRIPT_STATE_IDLE;
64    int8_t reachable = 1; /* 1文目へは到達可能 */
65    while (!script_eof() &&
66           (script_st.state != SCRIPT_STATE_END)) {
67      script_read(SCRIPT_PASS_2); /* ジャンプ先ラベル定義検査 */
68      switch (script_st.state) {  /* スクリプト終端への到達可能性の追跡 */
69      case SCRIPT_STATE_LABEL_FETCHED:
70        reachable = 1; /* 次の文は到達可能 */
71        break;
72      case SCRIPT_STATE_JUMP_FETCHED:
73        reachable = 0; /* 次の文は到達不可能 */
74        break;
75      default:          /* 次の文の到達可能性は今の文と同じ */
76        break;
77      }
78    }
79    if ((script_st.state == SCRIPT_STATE_END) &&
80        reachable) { /* 実行時にスクリプト終端へ到達する可能性有り */
81      fatal_s(AR_ERROR_OUT_OF_RANGE, script_st.lineno,
82              "REACHABLITY TO END-OF-SCRIPT");
83    }
84  }
85
86  /*-----------------------------------------------------
87   * スクリプト実行制御
88   *-----------------------------------------------------*/
89  enum ar_state_e fsm_script_next(enum ar_state_e ar_state)
90  {
91    switch (ar_state) {
92    default:
93      fatal(AR_ERROR_INTERNAL);
94      break;
95    case AR_STATE_INIT:  /* 初期状態 */
96      ar_state = AR_STATE_RUNNING;
97      /* スクリプト実行準備 */
98      script_rewind();
99      script_st.state = SCRIPT_STATE_IDLE;
100     break;
101   case AR_STATE_HOME: /* 定位置状態 */
102     ar_state = AR_STATE_RUNNING;
103     break;
104   case AR_STATE_RUNNING: /* スクリプト実行状態 */
105     ar_state = script_exec();  /* PASS3: スクリプト実行 */
106     break;
107   }
108   return ar_state;
109 }
```

スクリプト終端指示子への到達可能性の検証の仕組みを説明します. 関数 script_verify() では以下の方法で自動運転スクリプトの各行に対する到達可能性の有無を求めます.

- 自動運転スクリプトの 1 行目は到達可能です. なぜなら実行の開始を 1 行目と決めているからです.

以下の通り場合分けをして判定を行います. $i = 1$ に対して i 行目は到達可能なのは上述の通りですので, $i = 2$ と設定して以下の手順を繰り返して各行の到達可能性の判定を順次行います. そしてスクリプト終端指示子の行に来ると判定は終了します.

- i 行目がラベル定義指示子の場合:
  $i + 1$ 行目は到達可能と見なします. そのラベルへのジャンプが実際に起きるか否かは厳密には確定していませんが, 起き得ると仮定して到達可能とします.

- i 行目が指示子でラベル定義以外の場合:
  もし i 行目が到達可能であれば $i+1$ 行目も到達可能です. そうでなければ $i+1$ 行目は到達不能です.
- i 行目が無条件ジャンプ文の場合:
  $i+1$ 行目は到達不能です.
- i 行目が無条件ジャンプ文以外の場合:
  もし i 行目が到達可能であれば $i+1$ 行目も到達可能です. そうでなければ $i+1$ 行目は到達不能です.

この検証方法は非常に単純なことしか行っていませんが, スクリプト終端指示子 / に到達しないことを確認するには十分です. 自動運転スクリプトの 1 行目から最後の行まで, 1 行づつ順次スクリプトを読み込むだけで判定できるのが利点です. また現在注目している 1 つの行に対する到達可能性だけを記憶すれば良いので, 自動運転スクリプトの規模に関わらず定数のメモリ量で済むことも利点です. つまり極めて小規模なマイコンでも効率よく実行できる工夫がしてあります.

ソースファイル script-read.c (スクリプト構文解析部: 実装)

自動運転スクリプトの 1 行を読み込んで構文解析を行います. 構文は非常に原始的なものに設計していますから, 構文解析は非常に簡潔に行えます. 自動運転スクリプトの構文の検証をここで行います. このソースファイルに実装されている主要な関数は以下の通りです.

- `script_read()`
  自動運転スクリプトの 1 行を読み込んで構文解析を行います. 行の 1 文字目で指示子の種類を区別し, さらに指示子や文の種別で細かく場合分けを行い, 構文解析した結果を構造体 `script_st` に記憶します. EEPROM から自動運転スクリプトの読み込みは 3 回行います.
  - パス 1 (記号定数 SCRIPT_PASS_1)
    読み込み第 1 回目の動作です. EEPROM の先頭から順に 1 文づつ読み込み, 構文とパラメータ値の検証をした上でラベル登録を行います. これは先述の関数 `script_intern_label()` からの呼び出しで行われます. 文のパラメータの構文解析と検証は後述の関数 `read_params()` を使用します.
  - パス 2 (記号定数 SCRIPT_PASS_2)
    読み込み第 2 回目の動作です. EEPROM の先頭から順に 1 文づつ読み込んでジャンプ文で使用しているラベルが実際に登録されているか否かを検査するとともにスクリプト終端指示子に実行が到達するか否かを検査します. これは先述の関数 `script_verify()` を呼び出して行います.
  - パス 3 (記号定数 SCRIPT_PASS_3)
    読み込み第 3 回目の動作です. 自動運転スクリプトの実行はこのときに行います. この時点では自動運転スクリプトには構文誤りと未定義ラベルは存在せず, しかも実行がスクリプト終端指示子に決して到達しないことはすでに検証済みとなっています.
- `read_params()`
  文のパラメータの構文解析を行う機構で, 文の i 番目のパラメータの数値を配列 `script_st.p[]` の i 番目の要素に格納します. 例えば第 1 パラメータの数値を `script_st.p[0]` に格納します. 同時に構文やパラメータの数値の範囲の検査も行います. この関数は与えられたパラメータ仕様の文字列に従って構文検査を行うところに特長があります. パラメータ仕様の文字列では, 文字列の各 1 文字がパラメータの種別を表しており, 文字の並びでパラメータ種別の並びを表します. この関数は EEPROM の 1 行を順次読み進めて数字列を数値に変換し, パラメータの種別に応じた値の範囲の検査を実施します.

ソースファイル番号 30.7: script-read.c

```
/*--------------------------------------------------------
 * スクリプト構文解析器
 *--------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include <ctype.h>
#include <string.h>
#include "ar.h"
#include "mem.h"
#include "script.h"
#include "fatal.h"

/* スクリプト文の現在の行番号 */
static uint16_t lineno = 1;

static void script_next_line(void);
static void read_params(const char *spec,
                        uint8_t pass, uint16_t ln);

/* スクリプト1文の解析 */
void script_read(uint8_t pass)
{
  CLRWDT();
  uint8_t lid = 0;
  uint16_t ma = mem_addr_get();
  uint16_t ln = lineno;
  script_st.lineno = ln;
  script_st.state  = SCRIPT_STATE_NOP;
  uint8_t ch = mem_read(); /* 行先頭の文字 */
  if (!mem_is_space(ch)) {
    switch (ch) { /* コマンド */
    case SCRIPT_CH_END:     /* 指示子: データ終了 */
      script_st.state = SCRIPT_STATE_END;
      break;
    case SCRIPT_CH_COMMENT: /* 指示子: コメント */
      script_st.state = SCRIPT_STATE_COMMENT_FETCHED;
      script_next_line();
      break;
    case SCRIPT_CH_LABEL:   /* 指示子: ラベル定義 */
      script_st.state = SCRIPT_STATE_LABEL_FETCHED;
      /* ラベル番号 */
      lid = mem_read_number();
      if ((pass == SCRIPT_PASS_1) &&
          (lid >= N_LABELS)) {
        fatal_si(AR_ERROR_OUT_OF_RANGE, ln,
                 "ILLEGAL LABEL", lid);
      }
      ch = mem_read();
      mem_unread();
      if ((pass == SCRIPT_PASS_1) &&
          (ch != SCRIPT_CH_LABEL)) {
        fatal_s(AR_ERROR_SYNTAX, ln,
                "SYNTAX ERROR");
      }
      script_st.p[0] = lid;
      script_next_line();
      /* ラベル登録 */
      if (pass == SCRIPT_PASS_1) {
        script_label_memaddr[lid] = ma;
        script_label_lineno[lid]  = ln;
      }
      break;
    default: /* 未定義指示子 */
      if (pass == SCRIPT_PASS_1) {
        if (isascii(ch)) {
          fatal_sc(AR_ERROR_UNKNOW_DIRECTIVE, ln,
                   "UNKNOWN DIRECTIVE", ch);
        } else {
          fatal_s(AR_ERROR_UNKNOW_DIRECTIVE, ln,
```

```
                    "NON-ASCII CHARACTER FOR DIRECTIVE");
        }
      }
      break;
    }
  } else {
    mem_skip_space();
    ch = mem_read();
    switch (ch) { /* 文 */
    case SCRIPT_CH_COMMENT: /* 文: コメント */
      script_st.state = SCRIPT_STATE_COMMENT_FETCHED;
      script_next_line();
      break;
    case SCRIPT_CH_JUMP:    /* 文: 無条件ジャンプ */
      script_st.state = SCRIPT_STATE_JUMP_FETCHED;
      read_params("L", pass, ln);
      break;
    case SCRIPT_CH_JUMP_RAND: /* 文: 確率的ジャンプ */
      script_st.state = SCRIPT_STATE_JUMP_RAND_FETCHED;
      read_params("LX", pass, ln);
      break;
    case SCRIPT_CH_JUMP_SENS: /* 文: センサ条件ジャンプ */
      script_st.state = SCRIPT_STATE_JUMP_SENS_FETCHED;
      read_params("LNB", pass, ln);
      break;
    case SCRIPT_CH_MOTORON: /* 文: モーター・常点灯 */
      script_st.state = SCRIPT_STATE_MOTORON_FETCHED;
      read_params("B", pass, ln);
      break;
    case SCRIPT_CH_TRACKDIR: /* 文: 線路極性 */
      script_st.state = SCRIPT_STATE_TRACKDIR_FETCHED;
      read_params("TR", pass, ln);
      break;
    case SCRIPT_CH_TURUNOUT: /* 文: ポイント切替 */
      script_st.state = SCRIPT_STATE_TURNOUT_FETCHED;
      read_params("PR", pass, ln);
      break;
    case SCRIPT_CH_TURUNOUT_W: /* 文: ポイント待機 */
      script_st.state = SCRIPT_STATE_TURNOUT_W_FETCHED;
      read_params("", pass, ln);
      break;
    case SCRIPT_CH_RUNUNTIL: /* 文: センサ停止走行 */
      script_st.state = SCRIPT_STATE_RUNUNTIL_FETCHED;
      read_params("SDNN", pass, ln);
      break;
    case SCRIPT_CH_RUNTO: /* 文: センサ通過走行 */
      script_st.state = SCRIPT_STATE_RUNTO_FETCHED;
      read_params("SN", pass, ln);
      break;
    case SCRIPT_CH_ACCESARY: /* 文: アクセサリ */
      script_st.state = SCRIPT_STATE_ACCESARY_FETCHED;
      read_params("AB", pass, ln);
      break;
    case SCRIPT_CH_STAY: /* 文: 相対時間待機 */
      script_st.state = SCRIPT_STATE_STAY_FETCHED;
      read_params("Z", pass, ln);
      break;
    case SCRIPT_CH_STAY_ABS: /* 文: 絶対時間待機 */
      script_st.state = SCRIPT_STATE_STAY_ABS_FETCHED;
      read_params("W", pass, ln);
      break;
    default: /* 未定義文 */
      if (pass == SCRIPT_PASS_1) {
        if (isascii(ch)) {
          fatal_sc(AR_ERROR_UNKNOW_STATEMENT, ln,
                   "UNKNOWN STATEMENT", ch);
        } else {
          fatal_s(AR_ERROR_UNKNOW_STATEMENT, ln,
                  "NON-ASCII CHARACTER FOR STATEMENT");
        }
```

```
      }
      break;
    }
  }
}

/* パラメータの読み取りと検査 */
static void read_params(const char *spec,
                        uint8_t pass,
                        uint16_t ln)
{
  uint8_t n = strlen(spec);
  uint8_t i = 0;
  /* パラメータの読み取り */
  for (i = 0; i < n; i++) {
    mem_skip_space();
    uint8_t ch = mem_read();
    mem_unread();
    if ((pass == SCRIPT_PASS_1) &&
        (ch == SCRIPT_CH_TERMINATOR)) {
      fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
              "MISSING PARAMETERS");
    }
    script_st.p[i] = mem_read_number();
  }
  if (i != n) { /* パラメータ数が不一致 */
    fatal(AR_ERROR_SYNTAX);
  }
  mem_skip_space();
  if (mem_read() != SCRIPT_CH_TERMINATOR) {
    fatal(AR_ERROR_SYNTAX);
  }
  mem_unread();
  script_next_line();
  /* 各パラメータの値が所定の範囲にあるか否かを検査 */
  for (i = 0; spec[i] != '\0'; i++) {
    uint8_t lid = 0;
    switch (spec[i]) {
    case 'L': /* Label */
      lid = script_st.p[i];
      if (lid >= N_LABELS) {
        fatal_si(AR_ERROR_OUT_OF_RANGE, ln,
                 "LABEL OUT OF RANGE", lid);
      }
      if ((pass == SCRIPT_PASS_2) &&
          (script_label_memaddr[lid] == LABEL_UNDEFINED)) {
        fatal_si(AR_ERROR_UNDEFINED_LABEL, ln,
                 "LABEL UNDEFINED", lid);
      }
      break;
    case 'X': /* Probability: [0,10] */
      if (script_st.p[i] >= 11) {
        fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
                "PROBABILITY OUT OF RANGE");
      }
      break;
    case 'M': /* Speedstep: < AR_SPEEDSTEP_MAX */
      if (script_st.p[i] > AR_SPEEDSTEP_MAX) {
        fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
                "ILLEGAL SPEEDSTEP VALUE");
      }
      break;
    case 'D': /* Decp ID: 1,2,3... */
      if (script_st.p[i] == 0) {
        fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
                "ILLEGAL DECP ID");
      }
      break;
    case 'N': /* Sensor ID: 1,2,3,... */
      if (script_st.p[i] == 0) {
        fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
                "SENSOR ID OUT OF RANGE");
```

```
212         }
213         break;
214       case 'T': /* Track ID: [1,2] */
215         if ((script_st.p[i] == 0) || (script_st.p[i] >= 3)) {
216           fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
217                   "ILLEGAL TRACK ID");
218         }
219         break;
220       case 'P': /* Turnout ID: 1,2,3,... */
221         if (script_st.p[i] == 0) {
222           fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
223                   "ILLEGAL TURNOUT ID");
224         }
225         break;
226       case 'R': /* Direction: [1,2] */
227         if ((script_st.p[i] == 0) || (script_st.p[i] >= 3)) {
228           fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
229                   "ILLEGAL DIRECTION");
230         }
231         break;
232       case 'B': /* Boolean (Off/On): [0,1] */
233         if (script_st.p[i] >= 2) {
234           fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
235                   "BOOLEAN VALUE OUT OF RANGE");
236         }
237         break;
238       case 'Z': /* Staytime (Relative) */
239         if (script_st.p[i] > AR_STAYTIME_MAX) {
240           fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
241                   "STAYTIME (RELATIVE) VALUE OUT OF RANGE");
242         }
243         break;
244       case 'W': /* Staytime (Absolute) */
245         if (script_st.p[0] > AR_SLEEP_MAX) { /* time (abs) */
246           fatal_s(AR_ERROR_OUT_OF_RANGE, ln,
247                   "STAYTIME (ABSOLUTE) VALUE OUT OF RANGE");
248         }
249         break;
250       }
251     }
252   }
253
254   /* スクリプトの先頭へ移動 */
255   void script_rewind(void)
256   {
257     lineno = 1;
258     mem_addr_set(0);
259   }
260
261   /* スクリプトメモリ終了か */
262   int8_t script_eof(void)
263   {
264     return mem_is_end();
265   }
266
267   /* 次の行へ移動 */
268   static void script_next_line(void)
269   {
270     lineno++;
271     mem_skip_to_eol();
272   }
```

ソースファイル script-exec.c (スクリプト実行エンジン: 実装)

自動運転スクリプトの実行機構の中心部分です. 上述の関数 script_read() を呼び出すと文の構文解析結果は構造体 script_st に保持されます. 文の種別は script_st.state に記されますので, この構造体の値に従って読み出された文に対応する動作を行います. 自動運転スクリプト実行機構では script_st.state をスクリプト実行有限状態機械の状態を保持する目的にも使用します. スクリプト実行有限状態機械の動作には, そ

の状態および実行中の文の EEPROM 内のメモリアドレスの 2 つの情報を使用します. この自動運転スクリプト実行機構は自動運転スクリプトの全体をマイコンのメモリに読み込むことはせず, 1 文を実行する度に 1 文を EEPROM から読み出す方針としています. スクリプト実行有限状態機械の状態の一部を以下で説明します.

- 状態 SCRIPT_STATE_IDLE
  実行中の文がない状態です. この場合には関数 script_read() を呼び出して次に実行すべき 1 文を EEPROM から読み出します. 関数 script_read() は読み出した文の種別により, 状態を以下のいずれかに設定します.
- 状態 SCRIPT_STATE_NOP
  読み出された文が存在しない場合です. 本来はこの場合は起き得ないようにしてありますがバグに備えて緊急停止します.
- 状態 SCRIPT_STATE_END
  スクリプト終端指示子を読み込んだ場合です. 検証機構により本来はこの場合は起き得ないようにしてありますが検証機構のバグに備えて緊急停止します.
- 状態 SCRIPT_STATE_JUMP_FETCHED
  無条件ジャンプ文を読み込んだ直後の状態です. 無条件ジャンプ文の構文の仕様では第 1 パラメータがジャンプ先のラベルですから, script_st.p[0] にラベルが格納されています. ラベル表を参照し, そのラベルを定義している EEPROM 内のメモリアドレスを次の実行すべきスクリプトの文として設定します. 以上が済むと状態 SCRIPT_STATE_IDLE に遷移して次の文が読み込まれるようにします. なお状態名の末尾の FETCHED は, 文が読み込まれた直後の状態を表す慣習としています.
- 状態 SCRIPT_STATE_MOTORON_FETCHED
  モータ・常点灯文を読み込んだ直後の状態です. 第 1 パラメータがオン・オフを表しますから, script_st.p[0] の値に従って対応する関数を呼び出して状態 SCRIPT_STATE_IDLE へ遷移します.
- 状態 SCRIPT_STATE_TURNOUT_FETCHED
  ポイント文を読み込んだ直後の状態です. 待ち行列にポイント切替要求を入れます.
- 状態 SCRIPT_STATE_TURNOUT_W_FETCHED
  要求中のポイント切替すべてが完了するまでこの状態に留まり続けます. 完了すれば状態 SCRIPT_STATE_IDLE へ遷移します.
- 状態 SCRIPT_STATE_RUNUNTIL_FETCHED
  センサ停止走行文 RUN-UNTIL(v,i,Sb,Ss) を読み込んだ直後の状態です. この文の実行は以下の 2 段階による状態遷移で実現します.
  - 状態 SCRIPT_STATE_RUNUNTIL_FETCHED
    文を読み込んだ直後の状態です. 文のパラメータを引数に関数 ar_motor_run_until() を呼び出して, 次の状態に遷移します.
  - 状態 SCRIPT_STATE_RUNUNTIL_EXEC
    関数 ar_motor_next() の呼び出し結果が真である限りこの状態に留まり続けます. 偽になれば状態 SCRIPT_STATE_IDLE へ遷移します.

  同様な動作により, センサ通過走行文 RUN-TO(v,Sp) や時間待機文 STAY-TIME(t) に対する実現を行います.

ソースファイル番号 30.8: script-exec.c

```
/*------------------------------------------------------
 * スクリプト実行エンジン
 *------------------------------------------------------*/
#include <xc.h>
#include <stdint.h>
#include "pp-const.h"
#include "ar.h"
#include "mem.h"
#include "script.h"
#include "random.h"
#include "fatal.h"

/* 確率表 */
static uint16_t prob10_table[11] = {
  0,       /* =  0 * 65535 / 10 */
  6553,    /* =  1 * 65535 / 10 */
  13107,   /* =  2 * 65535 / 10 */
  19660,   /* =  3 * 65535 / 10 */
  26214,   /* =  4 * 65535 / 10 */
  32767,   /* =  5 * 65535 / 10 */
  39321,   /* =  6 * 65535 / 10 */
  45874,   /* =  7 * 65535 / 10 */
  52428,   /* =  8 * 65535 / 10 */
  58981,   /* =  9 * 65535 / 10 */
  65535    /* = 10 * 65535 / 10 */
};

enum ar_state_e script_exec(void)
{
  enum ar_state_e  ar_state = AR_STATE_RUNNING;
  uint8_t lid = 0;
  uint8_t p = 0;
  /* 文種に応じて実行 */
  switch (script_st.state) {
  default:
  case SCRIPT_STATE_NOP:
  case SCRIPT_STATE_END:
    /* モーター停止 */
    ar_motor_off();
    fatal_s(AR_ERROR_INTERNAL, script_st.lineno,
            "INTERNAL ERROR"); /* 内部エラー */
    break;
  case SCRIPT_STATE_IDLE: /* アイドル状態 */
    script_read(SCRIPT_PASS_3); /* 次の1文を読み取る */
    break;
  case SCRIPT_STATE_COMMENT_FETCHED: /* コメント */
    script_st.state = SCRIPT_STATE_IDLE;
    break;
  case SCRIPT_STATE_LABEL_FETCHED: /* ラベル定義 */
    script_st.state = SCRIPT_STATE_IDLE;
    lid = script_st.p[0];
    if (lid == LABEL_ID_HOME) { /* 定位置 */
      ar_state = AR_STATE_HOME;
    }
    break;
  case SCRIPT_STATE_JUMP_FETCHED: /* 無条件ジャンプ */
    script_st.state = SCRIPT_STATE_IDLE;
    lid = script_st.p[0]; /* label */
    mem_addr_set(script_label_memaddr[lid]);
    break;
  case SCRIPT_STATE_JUMP_RAND_FETCHED: /* 確率的ジャンプ */
    script_st.state = SCRIPT_STATE_IDLE;
    lid = script_st.p[0]; /* label */
    p   = script_st.p[1]; /* 確率 p/10 で */
    if ((p == 10) ||
        ((p <= 10) && (random_get16() < prob10_table[p]))) {
      mem_addr_set(script_label_memaddr[lid]);
    }
    break;
  case SCRIPT_STATE_JUMP_SENS_FETCHED: /* センサ条件ジャンプ */
```

```
71      script_st.state = SCRIPT_STATE_IDLE;
72      lid = script_st.p[0]; /* label */
73      if (if_sensor_get(script_st.p[1]) /* sn */
74          == script_st.p[2]) {          /* on(=1), off(=0) */
75        mem_addr_set(script_label_memaddr[lid]);
76      }
77      break;
78    case SCRIPT_STATE_MOTORON_FETCHED: /* モータ&常点灯On/Off */
79      script_st.state = SCRIPT_STATE_IDLE;
80      switch (script_st.p[0]) { /* on/off */
81      case 0: ar_motor_off(); break; /* オフ */
82      case 1: ar_motor_on();  break; /* オン */
83      }
84      break;
85    case SCRIPT_STATE_TRACKDIR_FETCHED: /* 線路極性 */
86      script_st.state = SCRIPT_STATE_IDLE;
87      enum ar_track_dir_e tdir = AR_TRACK_DIR_NORMAL;
88      switch (script_st.p[1]) { /* dir */
89      case 1: tdir = AR_TRACK_DIR_NORMAL;  break; /* 正方向 */
90      case 2: tdir = AR_TRACK_DIR_REVERSE; break; /* 逆方向 */
91      }
92      switch (script_st.p[0]) { /* track */
93      case 1: ar_track1_dir_set(tdir); break; /* トラック1 */
94      case 2: ar_track2_dir_set(tdir); break; /* トラック2 */
95      }
96      break;
97    case SCRIPT_STATE_TURNOUT_FETCHED: /* ポイント切替 */
98      script_st.state = SCRIPT_STATE_IDLE;
99      switch (script_st.p[1]) { /* dir */
100     case 1: if_toq_set_normal(script_st.p[0]);  break;
101     case 2: if_toq_set_reverse(script_st.p[0]); break;
102     }
103     break;
104   case SCRIPT_STATE_TURNOUT_W_FETCHED: /* ポイント待機 */
105     if (if_toq_is_done()) {
106       script_st.state = SCRIPT_STATE_IDLE;
107     }
108     break;
109   case SCRIPT_STATE_RUNUNTIL_FETCHED: /* 走行開始 */
110     script_st.state = SCRIPT_STATE_RUN_EXEC;
111     ar_motor_run_until(script_st.p[0],  /* speed */
112                        script_st.p[1],  /* decpi */
113                        script_st.p[2],  /* sn-brake */
114                        script_st.p[3]); /* sn-stop */
115     break;
116   case SCRIPT_STATE_RUNTO_FETCHED: /* 走行開始 */
117     script_st.state = SCRIPT_STATE_RUN_EXEC;
118     ar_motor_run_to(script_st.p[0],  /* speed */
119                     script_st.p[1]); /* sn-pass */
120     break;
121   case SCRIPT_STATE_RUN_EXEC: /* 走行継続 */
122     if (!ar_motor_next()) {
123       script_st.state = SCRIPT_STATE_IDLE;
124     }
125     random_get16();
126     break;
127   case SCRIPT_STATE_ACCESARY_FETCHED: /* アクセサリ制御 */
128     script_st.state = SCRIPT_STATE_IDLE;
129     switch (script_st.p[1]) { /* on/off */
130     case 0: if_accessary_off(script_st.p[0]); break; /* オフ */
131     case 1: if_accessary_on(script_st.p[0]);  break; /* オン */
132     }
133     break;
134   case SCRIPT_STATE_STAY_FETCHED: /* 停止開始(相対時間) */
135     script_st.state = SCRIPT_STATE_STAY_EXEC;
136     ar_stay_start(script_st.p[0]); /* time (rel) */
137     break;
138   case SCRIPT_STATE_STAY_ABS_FETCHED: /* 停止開始(絶対時間) */
139     script_st.state = SCRIPT_STATE_STAY_EXEC;
140     ar_stay_abs_start(script_st.p[0]); /* time (abs) */
```

```
141       break;
142     case SCRIPT_STATE_STAY_EXEC: /* 停止継続 */
143       if (!ar_stay_next()) {
144         script_st.state = SCRIPT_STATE_IDLE;
145       }
146       break;
147     }
148     return ar_state;
149   }
```

課題

1. ソースファイル script.c の中の関数 script_verify() におけるスクリプト終端指示子への到達可能性の検証の判定方法について考えます. もし判定結果が到達不能であった場合には, どのような実行に対してもスクリプト終端指示子には到達することは決してありません. なぜでしょうか. その理由を説明して下さい
2. スクリプト終端指示子への到達可能性の検証の判定方法について考えます. どのような実行に対してもスクリプト終端指示子には到達することは決してないにも関わらず, 到達可能と判定される場合があります. どのような自動運転スクリプトでそれが起きるか, 簡潔な具体例を示し, 到達可能と判定される理由を説明して下さい.
3. スクリプト終端指示子への到達可能性の検証の判定方法の改良を考えます. どのような実行に対しても, スクリプト終端指示子には決して到達しない場合のみ及びその場合に限り到達不能と判定するように, 判定方法を改良してください. 考案した判定方法を示し, なぜその判定方法で目的を達成できるのかを説明して下さい. そしてその判定方法を実装して下さい.

第 31 章

自動運転スクリプトの作成技法

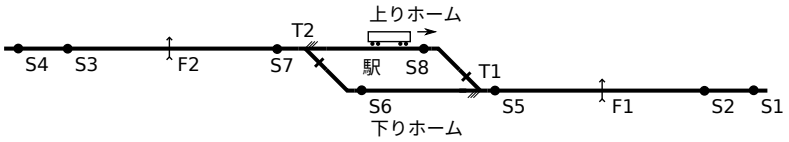
本章ではいくつかの自動運転スクリプトを紹介します. 多数の具体例を読解することで, 独自の自動運転スクリプトが作成できるようになって下さい.

本章で学ぶ内容は以下の通りです.

- 自動運転スクリプトの作成方法

31.1 単線退避駅トラックプラン

単線退避駅トラックプランを図 31.1 に示します.



列車退避を行う単線の駅で, 上りと下りの 2 つのホームがあります. 模型列車は 1 編成で, 走行を開始する位置は駅の上りホームです.

1. 模型列車は上りホームから東に向けて駅を出発します.
2. 東端で停車した後に走行方向を変えて西に向けて走行し, 駅の下りホームに停車します.
3. 模型列車は下りホームから西に向けて駅を出発します.
4. 西端で停車した後に走行方向を変えて東に向けて走行し, 駅の上りホームに停車します. 以上を繰り返します.

なお模型列車が駅を発車する際にはアクセサリ制御出力 1 番を 3 秒間オンにしますので, 発車ベルなどに使用可能です.

図 31.1: 単線退避駅トラックプラン

自動運転スクリプトは以下の通りです.

ソースファイル番号 31.1: ato-sta1-shuttle.txt

```
1 #-----------------------------
2 # 単線退避駅 / 1列車
3 #-----------------------------
4 #   減速度番号割当
5 #     1: 東端
6 #     2: 駅停車(西向)
7 #     3: 西端用
8 #     4: 駅停車(東向)
9 #   センサ番号割当
```

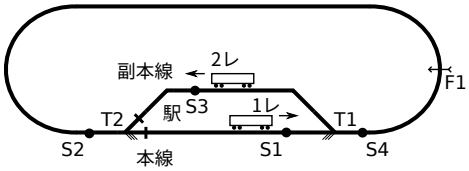
```
#      S1: 東端, 停止
#      S2: 東端, 制動
#      S3: 西端, 制動
#      S4: 西端, 停止
#      S5: 駅(西向), 制動
#      S6: 駅(西向), 停止
#      S7: 駅(東向), 制動
#      S8: 駅(東向), 停止
#    分岐器番号割当
#      T1: 駅東  (東端->駅進入が定位)
#      T2: 駅西  (西端->駅進入が定位)
#----------------------------
# 駅, 停車(東向)
# 初期化
  # 進路の構成
  T 1 1;          # 分岐器1: 定位
  T 2 1;          # 分岐器2: 定位
  D 1 1;          # 東行 (トラック1: 順方向)
  D 2 1;          # 東行 (トラック2: 順方向)
  M 1;            # モータ&常点灯On
  Z 30;           # 3秒待機
# ■ 繰返点
:1:
# ■ 駅(東向)
  # 進路の構成
  T 1 2;          # 分岐器1: 反位
  Z 30;           # 3秒待機
  # 出発ベル(3秒)
  A 1 1;          # アクセサリ1 On
  Z 30;           # 3秒待機
  A 1 0;          # アクセサリ1 Off
  # 駅→東端, 走行
  R 32 1 2 1;  # 速度 32(=100%), 減速度番号 1, 制動:S2, 停止:S1
  T 1 1;          # 分岐器1: 定位
# ■ 東端
  # 東端, 停車
  D 1 2;          # 西行 (トラック1: 逆方向)
  D 2 2;          # 西行 (トラック2: 逆方向)
  S 32;           # 待機時間32(=100%)
  # 東端→駅, 走行
  R 32 2 5 6;  # 速度 32(=100%), 減速度番号 2, 制動:S5, 停止:S6
# ■ 駅(西向)
  # 駅, 停車(西向)
  S 32;           # 待機時間32(=100%)
  # 進路の構成
  T 2 2;          # 分岐器2: 反位
  Z 30;           # 3秒待機
  # 出発ベル(3秒)
  A 1 1;          # アクセサリ1 On
  Z 30;           # 3秒待機
  A 1 0;          # アクセサリ1 Off
  # 駅→西端, 走行
  R 32 3 3 4;  # 速度 32(=100%), 減速度番号 3, 制動:S3, 停止:S4
  T 2 1;          # 分岐器2: 定位
# ■ 西端
  # 西端, 停車
  D 1 1;          # 東行 (トラック1: 正方向)
  D 2 1;          # 東行 (トラック2: 正方向)
  S 32;           # 待機時間32(=100%)
  # 西端→駅, 走行
  R 32 4 7 8;  # 速度 32(=100%), 減速度番号 4, 制動:S7, 停止:S8
# ■ 定位置 (駅)
:0:
  # 駅, 停車(東向)
  S 32;           # 待機時間 32(=100%)
```

```
75  # 繰返し
76    J 1;
77  # データ終了
78  /
79  #----------------------------
```

このトラックプランでは東端と西端はトンネルに隠しておき, 中央には山中の谷間の駅がある場合を想定しています. その他, 東端と西端に駅を設置して, 3 つの駅があるレイアウトにすることもできます.

31.2　エンドレス交換駅トラックプラン

エンドレス交換駅トラックプランを図 31.2 に示します.



列車交換を行う単線の駅で, 本線と副本線があります. 模型列車は 2 編成あり, 列車番号をそれぞれ 1 レ (第 1 列車; 優等列車), 2 レ (第 2 列車; 普通列車) とします. いずれの模型列車も走行を開始する位置は駅です.

1. 1 レが東に向けて駅を出発します. レイアウトを 1 周しますが, ときどきランダムに駅を通過します. やがて 1 レは駅に停車します.
2. 2 レが西に向けて駅を出発します. 2 レはレイアウトを 1 周すれば駅に停車します. 以上を繰り返します.

なお 1 レ, 2 レ共に駅を発車する際にはアクセサリ制御出力 1 番を 3 秒間オンにしますので, 発車ベルなどに使用可能です.

図 31.2: エンドレス交換駅トラックプラン

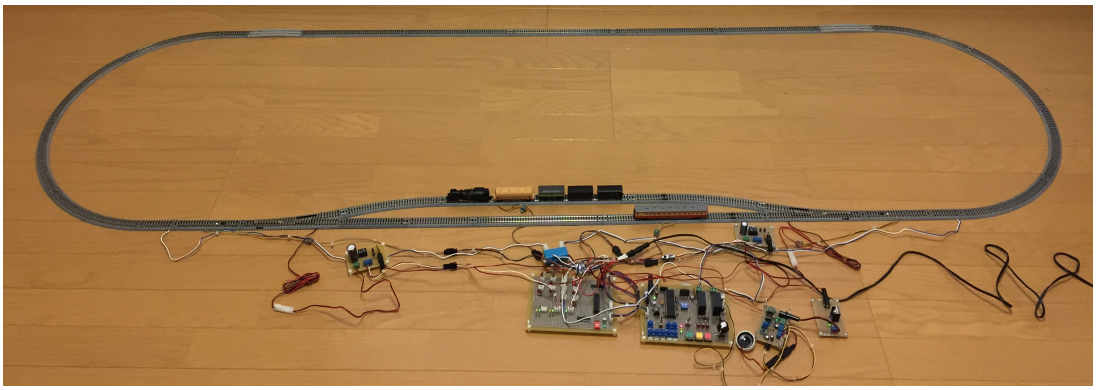


図 31.3: エンドレス交換駅トラックプランの実施例 (KATO ユニトラック)

自動運転スクリプトは以下の通りです.

ソースファイル番号 31.2: `ato-sta1-endless2.txt`

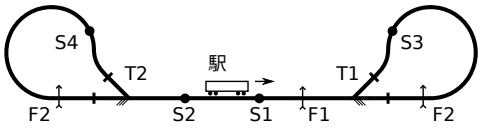
```
1  #----------------------------
```

```
# エンドレス交換駅 / 2列車
#----------------------------
#   減速度番号割当
#     1: 駅停車(本線/1レ)     東へ
#     2: 駅停車(副本線/2レ)   西へ
#   センサ番号割当
#     S1: 駅(本線),    停止
#     S2: 駅(本線),    制動
#     S3: 駅(副本線), 停止
#     S4: 駅(副本線), 制動
#   分岐器番号割当
#     T1: 駅東  (本線側が定位)
#     T2: 駅西  (本線側が定位)
#----------------------------
# 駅, 停車 1レ
# 駅, 停車 2レ
# 初期化
  # 進路の構成
  D 1 1;        # 東行 (トラック1: 順方向)
  T 1 1;        # 分岐器1: 定位
  T 2 1;        # 分岐器2: 定位
  M 1;          # モータ&常点灯On
  Z 10;         # 1秒待機
# ■ 繰返点
:1:
# ■ 列車 1レ
  # 出発ベル(3秒)
  A 1 1;        # アクセサリ1 On
  Z 30;         # 3秒待機
  A 1 0;        # アクセサリ1 Off
  # ランダムに駅を通過
:2:
  ? 3 5;        # 確率0.5
    # 1レ  駅→  駅通過
    X 32 2;     # 速度 32(=100%), 通過:S2
    X 32 1;     # 速度 32(=100%), 通過:S1
    J 2;
:3:
  # 1レ  駅→駅, 走行
  R 32 1 2 1;  # 速度 32(=100%), 減速度番号 1, 制動:S2, 停止:S1
  # 停車
  S 32;         # 待機時間32(=100%)
# ■ 列車 2レ
  # 進路の構成
  M 0;          # モータ&常点灯Off
  D 1 2;        # 東行 (トラック1: 逆方向)
  T 1 2;        # 分岐器1: 反位
  T 2 2;        # 分岐器2: 反位
  M 1;          # モータ&常点灯On
  Z 10;         # 1秒待機
  # 出発ベル(3秒)
  A 1 1;        # アクセサリ1 On
  Z 30;         # 3秒待機
  A 1 0;        # アクセサリ1 Off
  # 2レ  駅→駅 走行
  R 32 2 4 3;  # 速度 32(=100%), 減速度番号 2, 制動:S4, 停止:S3
# ■ 定位置 (駅)
:0:
  # 進路の構成
  M 0;          # モータ&常点灯Off
  D 1 1;        # 東行 (トラック1: 順方向)
  T 1 1;        # 分岐器1: 定位
  T 2 1;        # 分岐器2: 定位
  M 1;          # モータ&常点灯On
  # 停車
```

```
  S 32;          # 待機時間32(=100%)
# 繰返し
  J 1;
# データ終了
/
#-----------------------------
```

31.3　二重リバーストラックプラン

二重リバーストラックプランを図 31.4 に示します.



単線の駅で, 2 つのリバース線があります. 模型列車は 1 編成で, 走行を開始する位置は駅です.

1. 模型列車は駅を東に向けて出発します. リバース線を走行した後は西向きで駅に進入し, 停車します.
2. 模型列車は駅を西に向けて出発します. リバース線を走行した後は東向きで駅に進入し, 停車します.
   以上を繰り返します.

※ リバース区間は模型列車の編成長よりも長くして下さい.

図 31.4: 二重リバーストラックプラン

自動運転スクリプトは以下の通りです. リバース区間が左右に 2 つあるのが特長のトラックプランです. リバース区間から模型列車が出る前には, 車両センサ S3 または S4 が模型列車を検出します. リバース区間と駅区間の給電の極性は必要に応じて反転して, ショートが起きないようにします.

ソースファイル番号 31.3: ato-wreverse.txt

```
#-----------------------------
# 両方向リバース線
#-----------------------------
#    減速度番号割当
```

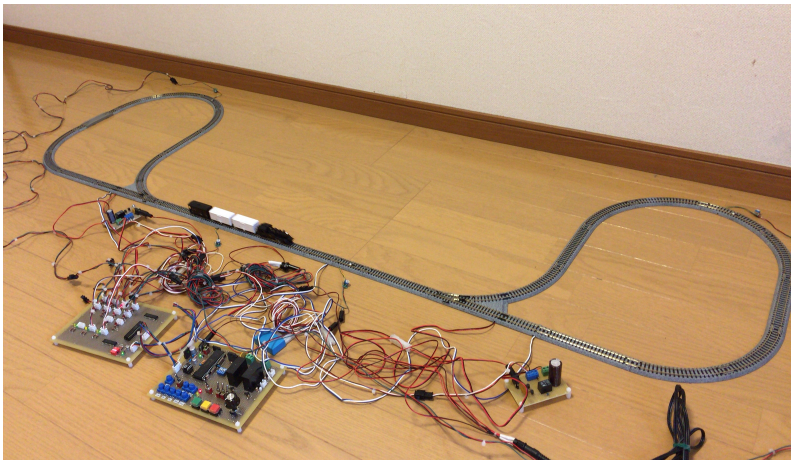


図 31.5: 二重リバーストラックプランの実施例 (KATO ユニトラック)

```
#      1: 駅, 東->西進入
#      2: 駅, 西->東進入
#   センサ番号割当
#      S1: 駅 (東->西進入時制動/西->東進入時停止)
#      S2: 駅 (西->東進入時制動/東->西進入時停止)
#      S3: 東リバース区間進入検知
#      S4: 西リバース区間進入検知
#   分岐器番号割当
#      T1: 東リバース
#      T2: 西リバース
#-----------------------------
# 駅, 停車(東向)
# 初期化
  # 進路の構成
  D 1 1;         # 東行 (トラック1: 順方向)
  D 2 1;         # 東行 (トラック2: 順方向)
  T 1 1;         # 分岐器1: 定位
  T 2 1;         # 分岐器2: 定位
  M 1;           # モータ&常点灯On
# ■ 繰返点
:1:
  # 駅から東に向けて出発, 走行
  X 32 3;        # 速度32(=100%), 通過: S3
  # 東リバース線処理: 給電極性反転&ポイント切替
  D 1 2;         # 西行 (トラック1: 逆方向)
  T 1 2;         # 分岐器1: 反位
  R 32 1 1 2;  # 速度32(=100%), 減速度番号 1, 制動:S1, 停止:S2
# ■ 駅
  # 駅, 停車
  S 32;          # 待機時間 32(=100%)
  # 進路の構成
  D 2 2;         # 西行 (トラック2: 逆方向)
  T 1 1;         # 分岐器1: 定位
  # 駅から西に向けて出発, 走行
  X 32 4;        # 速度32(=100%), 通過: S4
  # 西リバース線処理: 給電極性反転&ポイント切替
  D 1 1;         # 東行 (トラック1: 正方向)
  T 2 2;         # 分岐器2: 反位
  R 32 2 2 1;  # 速度32(=100%), 減速度番号 2, 制動:S2, 停止:S1
# ■定位置 (駅)
:0:
  # 駅, 停車
  S 32;          # 待機時間 32(=100%)
  # 進路の構成
  D 2 1;         # 西行 (トラック2: 正方向)
  T 2 1;         # 分岐器1: 定位
  # 繰返し
  J 1;
# データ終了
/
#-----------------------------
```

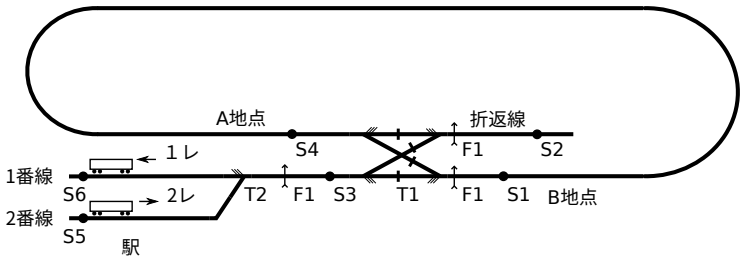
31.4　エンドレススイッチバック駅トラックプラン

エンドレススイッチバック駅トラックプランを図 31.6 に示します.

自動運転スクリプトは以下の通りです. 少々複雑な自動運転スクリプトですが, 列車の位置を想定しながら読み進めてみて下さい. 両渡り線 T1 は, 例えば KATO ユニトラックの両渡り線のように, ひとつのポイント制御出力で制御ができて, 定位では直通し, 反位では渡り線を通過するものを想定しています.

ソースファイル番号 31.4: ato-switchback.txt

```
#-----------------------------
# スイッチバック+エンドレス(交換) / 2列車
```



エンドレス中にスイッチバック駅があるレイアウトです. 駅には 2 線のホームがあり, それぞれ 1 番線, 2 番線と呼ぶことにします. 模型列車は 2 編成あり, それぞれ 1 レ (第 1 列車), 2 レ (第 2 列車) として区別します. 走行を開始する位置はそれぞれ駅の 1 番線, 2 番線です.

1. 1 レは 1 番線から東に向けて出発して, 両渡り線 T1 を通過してスイッチバックの折返線に入ります. 走行方向を反転して西に向けて走行し, A 地点, B 地点の順にエンドレスを 1 周して駅の 1 番線に入線して停車します.
2. 2 レは 2 番線から東に向けて出発し B 地点, A 地点の順に通過して, エンドレスを 1 周してスイッチバックの折返線に入ります. 走行方向を反転して両渡り線 T1 を通過し, 駅の 2 番線に入線して停車します. 以上を繰り返します.

なお模型列車が駅の発車する際にはアクセサリ制御出力を 3 秒間オンにしますので, 発車ベルなどに使用可能です. なお 1 番線の時はアクセサリ制御出力 1 番を, 2 番線の時はアクセサリ制御出力 2 番をオンにしますので, 異なる発車ベルを鳴らすことが可能です.

図 31.6: エンドレススイッチバック駅トラックプラン

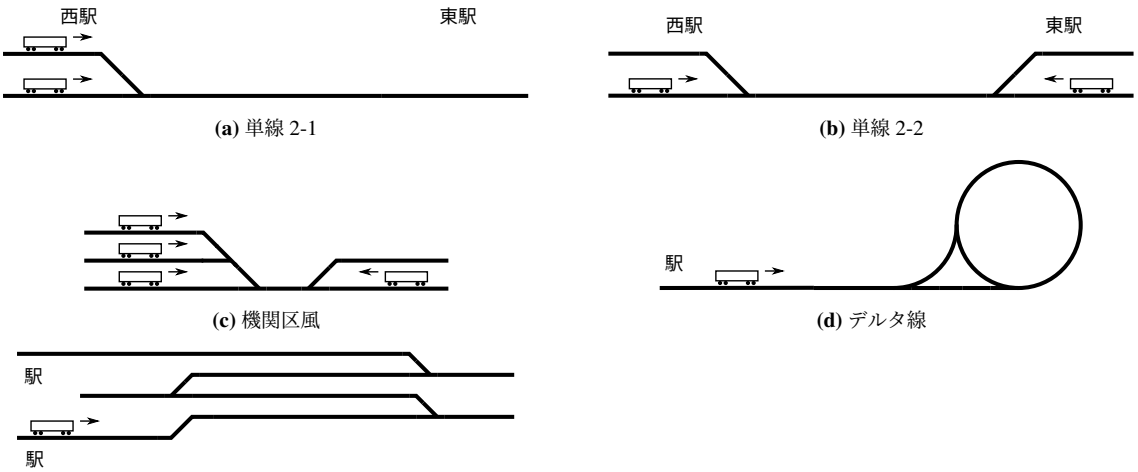
```
3  #-----------------------------
4  #   減速度番号割当
5  #     1: 1レ/本線→折返線     S4→(T1)→S2
6  #     2: 1レ/折返線→1番線    S3→(T2)→S6
7  #     3: 2レ/2番線→折返線    S3→(T1)→S2
8  #     4: 2レ/本線→2番線      S1→(T1,T2)→S5
9  #   センサ番号割当
10 #     S1: 駅進入, 制動 (2レ)
11 #     S2: 折返線, 停止 (1レ/2レ)
12 #     S3: 駅進入, 制動 (1レ); 折返進入, 制動 (2レ)
13 #     S4: 折返線進入, 制動 (1レ)
14 #     S5: 駅2番線, 停止 (2レ)
15 #     S6: 駅1番線, 停止 (1レ)
16 #   ポイント番号割当
17 #     T1: ダブルクロス; 駅←→折返線/本線 (駅←→本線が定位)
18 #     T2: 本線 1番線/2番線 (1番線側が定位)
19 #-----------------------------
20 # 駅1番線: 1レ
21 # 駅2番線: 2レ
22 # 初期化
23   # 進路の構成
24   D 1 1;        # 東行 (トラック1: 順方向)
25   T 1 1;        # ポイント1: 定位
26   T 2 1;        # ポイント2: 定位
27   M 1;          # モータ&常点灯On
28   Z 10;         # 待機時間 1 [秒]
29 # ■ 繰返点
30 :1:
```

```
31 |   # 出発ベル(3秒) / アクセサリ1
32 |   A 1 1;        # アクセサリ1 On
33 |   Z 30;         # 3秒待機
34 |   A 1 0;        # アクセサリ1 Off
35 | # ■ 列車 1レ
36 |   # 駅1番線から本線を走り, 折返線まで走行
37 |   R 32 1 4 2;  # 速度32(=100%), 減速度番号 1, 制動:S4, 停止:S2
38 |   # 進路の構成
39 |   T 1 2;        # ポイント1: 反位
40 |   # 折返線で停車
41 |   D 1 2;        # 西行 (トラック1: 逆方向)
42 |   Z 10;         # 待機時間 1 [秒]
43 |   # 折返線から駅1番線まで走行
44 |   R 20 2 3 6;  # 速度20(=63%), 減速度番号 2, 制動:S3, 停止:S6
45 |   D 1 1;        # 東行 (トラック1: 順方向)
46 |   # 駅で停車
47 |   S 32;         # 待機時間 32(=100%)
48 | # ■ 列車 2レ
49 |   # 進路の構成
50 |   T 2 2;        # ポイント2: 反位
51 |   Z 10;         # 待機時間 1 [秒]
52 |   # 出発ベル(3秒) / アクセサリ2
53 |   A 2 1;        # アクセサリ2 On
54 |   Z 30;         # 3秒待機
55 |   A 2 0;        # アクセサリ2 Off
56 |   # 駅2番線から折返線まで走行
57 |   R 20 3 3 2;  # 速度20(=63%), 減速度番号 3, 制動:S3, 停止:S2
58 |   # 進路の構成
59 |   T 1 1;        # ポイント1: 定位
60 |   # 折返線で停車
61 |   D 1 2;        # 西行 (トラック1: 逆方向)
62 |   Z 10;         # 待機時間 1 [秒]
63 |   # 折返線から駅2番線まで走行
64 |   R 32 4 1 5;  # 速度32(=100%), 減速度番号 4, 制動:S1, 停止:S5
65 | # ■定位置 (駅)
66 | :0:
67 |   # 進路の構成
68 |   T 2 1;        # ポイント2: 定位
69 |   D 1 1;        # 東行 (トラック1: 順方向)
70 |   # 駅, 停車
71 |   S 32;         # 待機時間 32(=100%)
72 |   # 繰返し
73 |   J 1;
74 | # データ終了
75 | /
76 | #----------------------------
```

課題

1. 本章で紹介した自動運転スクリプト言語は多分にアセンブリ言語的であり, 言語処理機構の開発は楽に行えました. しかし自動運転スクリプトの作成者は, そのしわ寄せで不便なデバッグ作業を強いられます. そこでより高水準な記述が行える言語を設計し, 言語処理機構の実装を行って下さい.
2. 図 31.4 に示す二重リバーストラックプランに対する自動運転スクリプトをソースファイル ato-wreverse.txt (653 ページ) で紹介しました. この自動運転パターンでは模型列車は必ず駅に停車をします. この自動運転スクリプトを改造して, ランダムに駅を通過するようにして下さい.
3. 図 31.7a に示すレイアウトに対するトラックプランを考え, 自動運転スクリプトを作成して下さい. このレイアウトでは, 2 編成の模型列車が交互に西駅と東駅の間を往復します.
4. 図 31.7b に示すレイアウトに対するトラックプランを考え, 自動運転スクリプトを作成して下さい. このレイアウトでは, 2 編成の模型列車が交互に西駅と東駅の間を走行します.

5. 図 31.7c に示すレイアウトに対するトラックプランを考え, 自動運転スクリプトを作成して下さい. このレイアウトでは, 機関車が頻繁に行き来する賑やかな機関区の様子を再現します. アクセサリ出力と PCM サウンド再生器 (第 27 章) を利用して, ときどき機関車のホイッスルを鳴らすようにすると良いでしょう.
6. 図 31.7d に示すレイアウトに対するトラックプランを考え, 自動運転スクリプトを作成して下さい. このレイアウトでは, 駅を発車した模型列車は何度かエンドレスを周回して再び駅に戻ってきます.
7. 図 31.7e に示すレイアウトに対するトラックプランを考え, 自動運転スクリプトを作成して下さい. このレイアウトでは, 急斜面を登って行く連続スイッチバックの様子を再現します. このレイアウトに対する自動運転スクリプトが完成した後はセンサ・ポイントインターフェースボードを増設して, 立山砂防工事専用軌道の連続 18 段スイッチバックの再現に挑戦してください.



(a) 単線 2-1

(b) 単線 2-2

(c) 機関区風

(d) デルタ線

(e) 3 段連続スイッチバック

図 31.7: 課題用レイアウト 5 題

第 32 章

自動運転スクリプトの
エミュレーション技法

本章では自動運転スクリプトの開発に使用するパーソナルコンピュータ上で検証を行う簡単なエミュレータソフトウエアを紹介します. このエミュレータの利用により自動運転スクリプトの開発効率は大幅に向上します. なお本章で紹介するエミュレータは図 32.1 に示すようにパーソナルコンピュータ上で動作するものであり, PIC マイコン用のファームウエアではありません. 自動運転スクリプトの開発に使用するものですので, 開発環境の一部をなすものと考えて下さい. 一連の作業の全体像を図 32.2 に示します.

第 30 章で紹介したファームウエア内部の自動運転スクリプト検証機構では, 間違い箇所を見つけるのにずいぶんと苦労をします. なぜなら自動運転スクリプトを修正・変更する度に, EEPROM の書き換え, 自動運転装置への装着, そして試走が必要だからです. エラーは LED の点滅で表示されるだけで自動運転スクリプトの何行目が間違っているかは表示されず, 実機上でのデバッグ作業は極めて困難です. 実機デバッグが困難な装置が相手ならエミュレータを使わざるを得ません.

本章で紹介する自動運転エミュレータプログラム emuscript の特徴は以下の通りです.

- パーソナルコンピュータ上での自動運転スクリプトの検証
  構文誤り等の初歩的なミスはパーソナルコンピュータ上で発見できますから, 実機テストを行う回数が激減します.
- 自動運転スクリプトのエミュレーション実行
  自動運転スクリプトを眺めるだけではどのような順序で制御が進行するかまでは捉えにくいものです. そこでパーソナルコンピュータ上で自動運転スクリプトをエミュレーション実行することで, 制御の進



図 32.1: emuscript 使用中の様子 (Ubuntu 上)

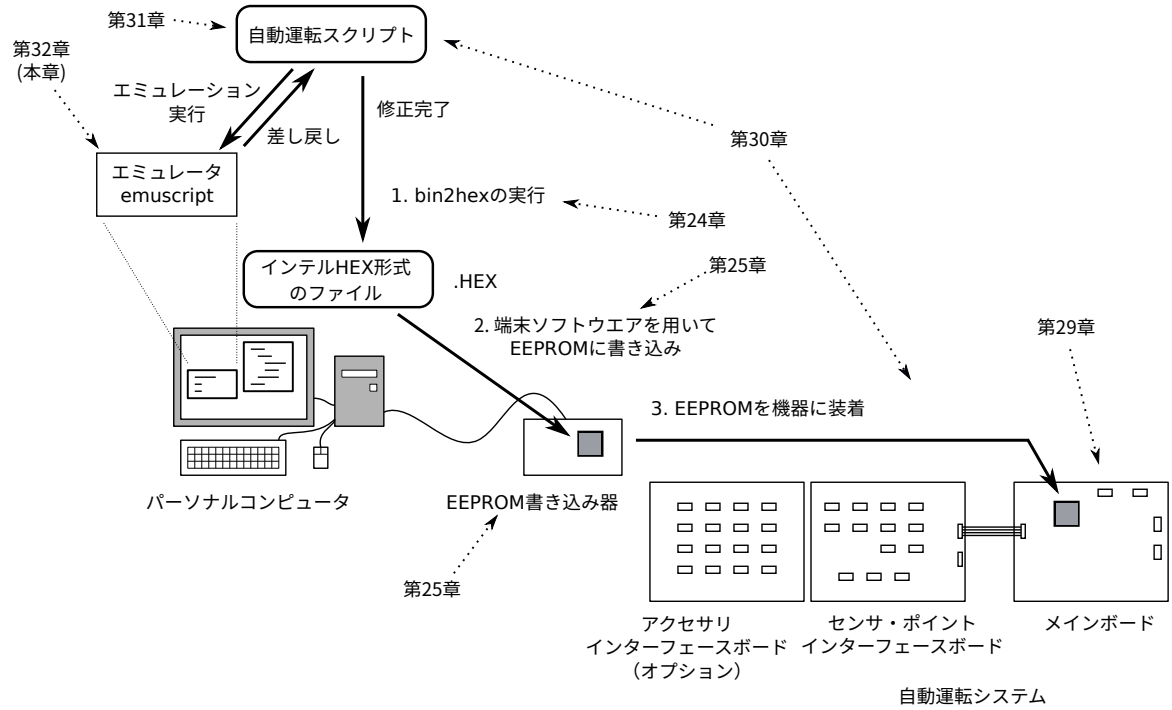


図 32.2: エミュレータを使用した自動運転スクリプト開発のための作業手順

行の追跡・確認を容易にします.

- 標準 C 言語で記述したコンソールアプリケーション
  標準 C 言語で記述しているために可搬性が高く, 特定のオペレーティングシステムや GUI に依存することなく様々なコンピュータ環境で動作可能です.

本章で学ぶ内容は以下の通りです.

- 自動運転スクリプトのエミュレーション実行方法

32.1　使用方法

自動運転エミュレータプログラム emuscript の使用方法を説明します. このプログラムはコマンドライン上で動作します. なお実行の際にはプログラムへの実行パスが通っているようにして下さい.

実行例

基本的な使用方法は単に自動運転スクリプトのファイルをコマンドライン引数に指定するだけです. エラーがあるときには原因が詳しく表示されます. 表 32.1 がエラーメッセージの一覧ですので, エラーがあるときにはこれを参考にして修正作業に取り組んで下さい.

例として単線レイアウト用の自動運転スクリプト `ato-shuttle.txt` (628 ページ) に対して実行した場合の出力を図 32.3 に示します.

パス 1 では構文の検査とラベルの定義を行い, パス 2 では自動運転スクリプトの検証を行い, パス 3 では自動運転スクリプトのエミュレーション実行を行います. エミュレーション実行の際には, 最初にラベル情報を表示し, 次に実行トレースを表示します.

*** LABEL の行からは, 定義されたラベルの情報が 1 行ごとに表示されます. 上の例での

```
> emuscript ato-shuttle.txt
PASS 1: SYNTAX CHECK & LABEL DEFINITION...
PASS 2: VERIFICATION...
PASS 3: EXECITION...
*** LABEL
   0: 0x030C (LINE=27)
   1: 0x01B4 (LINE=18)
*** EXECUTION TRACE
1 (LINE 15) TRACK #1 DIR=NORMAL
2 (LINE 16) MOTOR ON
3 (LINE 20) RUN-UNTIL SPEEDSTEP=32 DECPI=1 SNBRK=2 SNSTOP=1
4 (LINE 22) TRACK #1 DIR=REVERSE
5 (LINE 23) STAY T=32
6 (LINE 25) RUN-UNTIL SPEEDSTEP=32 DECPI=2 SNBRK=3 SNSTOP=4
HOME POSITION
7 (LINE 29) TRACK #1 DIR=NORMAL
8 (LINE 30) STAY T=32
9 (LINE 35) RUN-UNTIL SPEEDSTEP=32 DECPI=1 SNBRK=2 SNSTOP=1
10 (LINE 37) TRACK #1 DIR=REVERSE
11 (LINE 38) STAY T=32
12 (LINE 40) RUN-UNTIL SPEEDSTEP=32 DECPI=2 SNBRK=3 SNSTOP=4
HOME POSITION
```

図 32.3: 自動運転スクリプト ato-shuttle.txt のエミュレーション実行時の出力

```
1: 0x01B4 (LINE=18)
```

には以下の情報が含まれています.

- ラベル名: 1
- EEPROM 内でラベルを定義されている部分のメモリアドレス: 0x01B4
- 自動運転スクリプト内でラベルを定義している行番号: LINE=18

*** EXECUTION TRACE の行からは, エミュレーション実行した内容を 1 文ごとに順次表示します. なお模型列車が定位置にあるときは HOME POSITION と表示されます. 上の例での

```
6 (LINE 25) RUN-UNTIL SPEEDSTEP=32 DECPI=2 SNBRK=3 SNSTOP=4
```

には以下の情報が含まれています.

- 動作ステップ番号: 6
- 実行した文の自動運転スクリプトでの行番号: LINE 25
- 実行した文の内容: RUN-UNTIL SPEEDSTEP=32 DECPI=2 SNBRK=3 SNSTOP=4
  これはセンサ停止走行 RUN-UNTIL(32,2,3,4) を示しています. SPEEDSTEP はスピードステップ, DECPI は減速度番号, SNBRK は制動センサ番号, SNSTOP は停止センサ番号です.

標準的なエミュレーション実行では模型列車が定位置に 2 回到達すると終了します. 上の例では 2 回目の

表 32.1: エラーメッセージ表

エラーメッセージ	説明
UNKNOWN DIRECTIVE	未定義の指示子を使用
NON-ASCII CHARACTER FOR DIRECTIVE	指示子に全角文字を使用
LABEL UNDEFINED	未定義のラベルを使用
SYNTAX ERROR	構文に誤り
UNKNOWN STATEMENT	未定義の文を使用
NON-ASCII CHARACTER FOR STATEMENT	全角文字を文に使用
ILLEGAL LABEL	値が範囲外
MISSING PARAMETERS	パラメータの数が不足
LABEL OUT OF RANGE	ラベルが範囲外
PROBABILITY OUT OF RANGE	確率値が範囲外
ILLEGAL SPEEDSTEP VALUE	スピードステップ値が範囲外
ILLEGAL DECP ID	減速度パラメータ番号が範囲外
SENSOR ID OUT OF RANGE	車両センサ番号が範囲外
ILLEGAL TRACK ID	フィーダ番号が範囲外
ILLEGAL TURNOUT ID	ポイント番号が範囲外
ILLEGAL DIRECTION	走行方向の値が不正
BOOLEAN VALUE OUT OF RANGE	ブール値でない
STAYTIME (RELATIVE) VALUE OUT OF RANGE	待機時間 (相対値) が範囲外
STAYTIME (ABSOLUTE) VALUE OUT OF RANGE	待機時間 (絶対値) が範囲外
INTERNAL ERROR	内部エラー

HOME POSITION の表示で終了しているのが分かると思います. 詳しくは後述しますが, この回数は変更が可能です.

上の例で使用した自動運転スクリプト ato-shuttle.txt にはバグはありませんでしたが, バグのある自動運転スクリプトの場合の例を以下に示します.

```
> emuscript ato-shuttle-buggy.txt
PASS 1: SYNTAX CHECK & LABEL DEFINITION...
FATAL ERROR (2) AT LINE 24. UNKNOWN DIRECTIVE: K
```

この場合ではパス 1 でエラーが発生しています. 自動運転スクリプトの 24 行目にエラーがあり, その原因は UNKNOWN DIRECTIVE: K すなわち未定義の指示子 K が使用されていることを示しています. 自動運転スクリプトの修正はこれら行番号と原因の情報をもとにして行います.

起動時のコマンドラインオプション

自動運転エミュレータプログラム emuscript の起動方法は以下の通りです.

```
emuscript [OPTION] SCRIPT
```

SCRIPT 部は自動運転スクリプトの指定で以下のものが用意されており, この部分の指定は必須です.

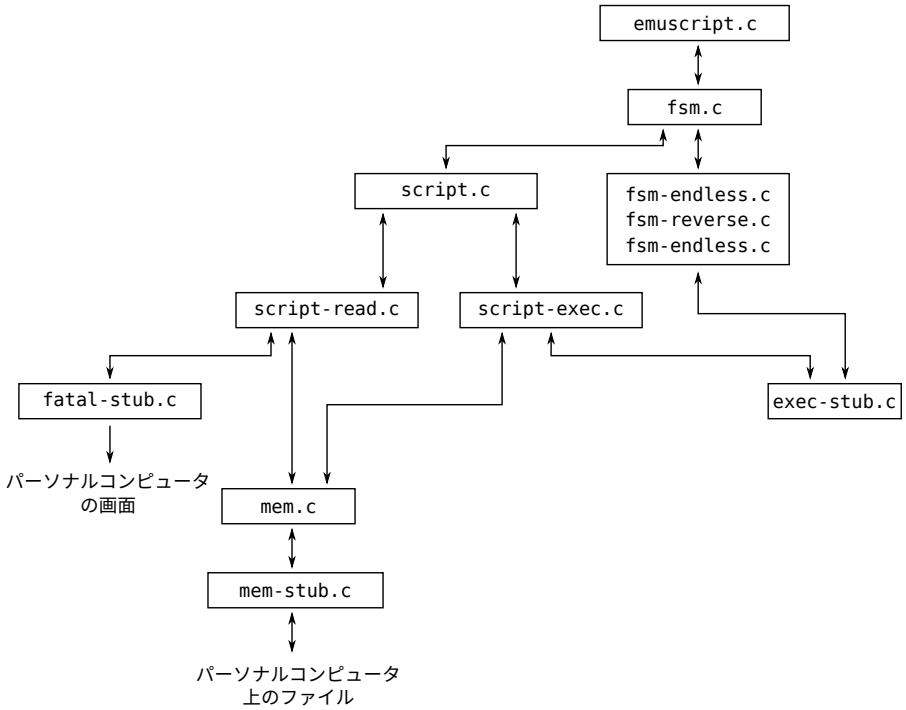


図 32.4: 自動運転エミュレータ emuscript における主要ソースファイル間の関係

- `-shuttle`, `-endless`, `-reverse`
  第 29 章で紹介した C 言語による事前組み込み済み自動運転制御プログラムの動作をエミュレートします. それぞれ, 往復, エンドレス, リバースとなっています.
- `FILE`
  自動運転スクリプトのファイル名を指定します. インテル HEX ファイルではなく, テキストファイルです.

OPTION 部はコマンドラインオプションの指定で以下のものが用意されています.

- `-r` *N*
  擬似乱数の初期値を *N* に設定します.
- `-h` *H*
  エミュレーション実行の終了条件を, 模型車両が定位置を *H* 回目に訪れたときとします. デフォルトでは *H* の値は 2 です.
- `-md`
  フィーダ出力 1 と 2 の状態を常に表示します. `-md1` ではフィーダ出力 1 の状態を常に表示し, `-md2` ではフィーダ出力 2 の状態を常に表示します.
- `-t` *T*
  ポイント番号 *T* の開通方向を常に表示します.

32.2　自動運転エミュレータプログラム emuscript

自動運転エミュレータ emuscript を構成するソースファイル一覧を以下に示します. 図 32.4 に主要なソースファイル間の関係を示します. これまでに作成したソースファイルのほとんどを再利用し, ハードウエアの違いに関する部分だけを置き換えます. この関係を図 32.5 に示します. また主要ソースファイル間の関係を示した図 30.7 と図 32.4 とを比較してみて下さい. 例えば EEPROM へのアクセスは, パーソナルコンピュータ上でファイルへのアクセスに置き換えます. ソースファイル再利用は開発と保守の労力を削減するだけでなく, マイコンでのファームウエアとパーソナルコンピュータでのソフトウエアの動作の同一性を実現します.

表 32.2: 【ソースファイル一覧】 自動運転スクリプトエミュレータ emuscript

初	頁	番号	ファイル名	内容	行数
✓	665	32.1	`emuscript.c`	主ソースコード	180
✓	667	32.2	`usage.c`	使用法表示	28
✓	668	32.3	`fatal-stub.c`	エラー表示: 実装	33
	542	28.26	`mem-stub.c`	EEPROM エミュレーション: 実装	41
✓	669	32.4	`exec-stub.c`	ハードウエアエミュレーション: 実装	131
	543	28.27	`xc.h`	XC8 代替ヘッダ	2
	584	29.6	`fsm.h`	自動運転実現: ヘッダ	10
	584	29.7	`fsm.c`	自動運転実現: 実装	30
	585	29.8	`fsm-shuttle.c`	自動運転実現 (往復): 実装	105
	587	29.9	`fsm-endless.c`	自動運転実現 (エンドレス): 実装	76
	588	29.10	`fsm-reverse.c`	自動運転実現 (リバース): 実装	101
	636	30.5	`script.h`	スクリプト言語: ヘッダ	79
	638	30.6	`script.c`	スクリプト言語: 実装	109
	640	30.7	`script-read.c`	スクリプト構文解析部: 実装	272
	645	30.8	`script-exec.c`	スクリプト実行エンジン: 実装	149
	525	28.9	`mem.h`	EEPROM メモリアクセス: ヘッダ	24
	148	8.2	`random.h`	疑似乱数: ヘッダ	7
	149	8.3	`random.c`	疑似乱数: 実装	26
				計 1403 行 (うち初出 372 行, 既出 1031 行)	

ソースファイル `emuscript.c` (主ソースコード)

動作の主たる部分です. コマンドライン引数を解析して, エミュレーション実行に必要なパラメータを設定し, 検証を行った後に自動運転の実行を呼び出します. 動作の状況を表示する関数も用意しておき, 実行状況が随時分かるようにします.

エミュレーション実行の中心部分が関数 `emu_execution()` で, 図 32.6 にその動作の骨子を示します.

関数 `fsm_next()` は自動運転の実行を司る有限状態機械であり, ソースファイル `fsm.c` (584 ページ) で示した通りで, さらにこの関数から以下の関数が呼び出されています.

- 関数 `fsm_shuttle_next()` — ソースファイル `fsm-shuttle.c` (585 ページ)
- 関数 `fsm_endless_next()` — ソースファイル `fsm-endless.c` (587 ページ)

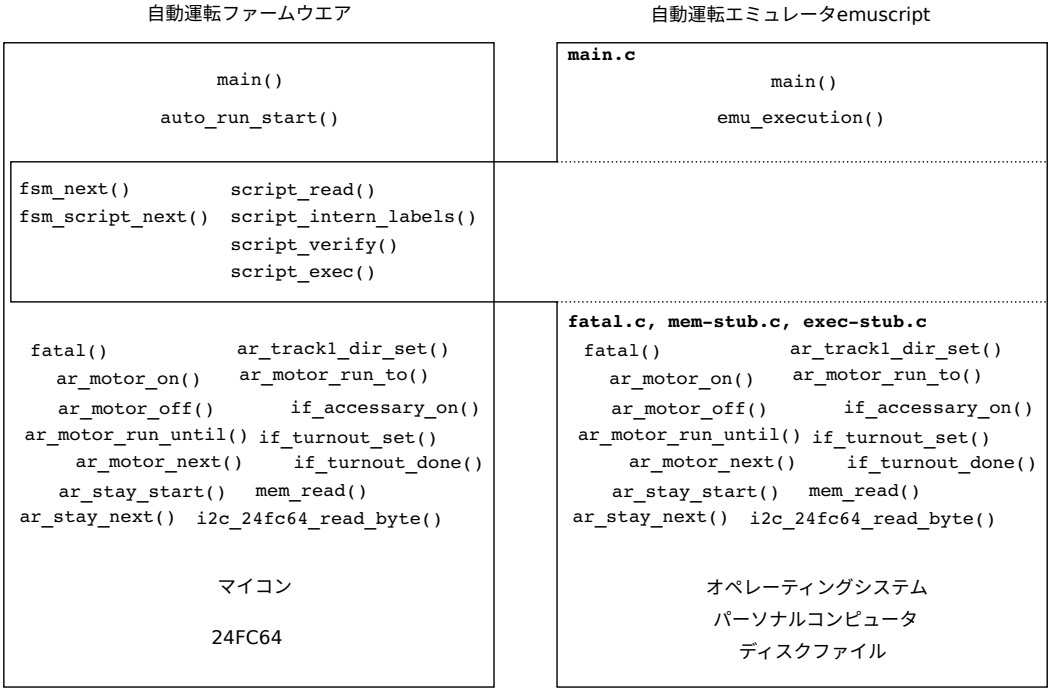


図 32.5: ソースコード関係図

```
1  void emu_execution(int nh)
2  {
3    int nhome = 0;                  — 定位置への到達回数の計数用
4    printf("*** EXECUTION TRACE\n");
5    ar_state = AR_STATE_INIT;           — 有限状態機械を初期化
6    while (nhome < nh) {            — 指定した回数まで実行継続
7      if (ar_state == AR_STATE_HOME) {
8        printf("HOME POSITION\n");
9        nhome++;                    — 定位置へ到達
10     }
11     ar_state = fsm_next(ar_state);    — 有限状態機械の実行
12     while (!if_toq_is_done()) {       — ポイント切替
13       if_toq_next(1);
14     }
15   }
16 }
```

図 32.6: エミュレーション実行機構の概要

- 関数 fsm_reverse_next() — ソースファイル fsm-reverse.c (588 ページ)
- 関数 fsm_script_next() — ソースファイル script.c (638 ページ)

いずれの関数の動作も, 車両センサの検出および一定時間の経過の検出を契機として状態遷移が生じる作りになっています. これらは IOC や TMR などのマイコン内蔵の周辺モジュールによるハードウエア的な信号として検出されるものですが, 本エミュレータではそれら信号を擬似的に検出するソフトウエア上の仕組みを導入しますが, 詳細は後述します. この仕組みにより自動運転スクリプト実行の有限状態機械が動作して, エミュ

レーションが実現されます.

ソースファイル番号 32.1: emuscript.c

```
/*
 * SCRIPT SIMULATOR
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <time.h>
#include "ar.h"
#include "mem.h"
#include "script.h"
#include "fsm.h"
#include "random.h"

#define NH_DEFAULT  2

enum op_pattern_e op_pattern = OPPAT_UNDEF;
enum ar_state_e ar_state = AR_STATE_INIT;

#define TRACKDIR_TRACK1  0
#define TRACKDIR_TRACK2  1
int monitor_trackdir[2];
int monitor_turnout[IF_MAX_TURNOUTS+1];
enum ar_track_dir_e   state_trackdir[2];
enum if_turnout_dir_e state_turnout[IF_MAX_TURNOUTS+1];

void emu_script_builtin(int nh);
void emu_script_eeprom(int nh, char *script);
void emu_execution(int nh);
void monitor_init(void);
void monitor_dump(void);
extern void mem_emu_init(char *f);
extern void usage(void);

int main(int argc, char **argv)
{
  char *script = NULL;
  int r = time(NULL) % 256;
  int nh = NH_DEFAULT;
  int i;
  monitor_init();
  op_pattern = OPPAT_UNDEF;
  /* Command line args */
  for (argc--, argv++; argc > 0; argc--, argv++) {
    if ((strcmp(argv[0], "-r") == 0) && (argc > 2)) {
      r = atoi(argv[1]);
    } else if ((strcmp(argv[0], "-h") == 0) && (argc > 2)) {
      nh = atoi(argv[1]);
    } else if (strcmp(argv[0], "-md") == 0) {
      monitor_trackdir[TRACKDIR_TRACK1] = 1;
      monitor_trackdir[TRACKDIR_TRACK2] = 1;
    } else if (strcmp(argv[0], "-md1") == 0) {
      monitor_trackdir[TRACKDIR_TRACK1] = 1;
    } else if (strcmp(argv[0], "-md2") == 0) {
      monitor_trackdir[TRACKDIR_TRACK2] = 1;
    } else if ((strcmp(argv[0], "-mt") == 0) && (argc > 2)) {
      int t = atoi(argv[1]);
      if ((t < 1) || (t > IF_MAX_TURNOUTS)) {
        printf("Value out of range for %s option\n", argv[0]);
        usage();
      }
      monitor_turnout[t] = 1;
    } else if (strcmp(argv[0], "-shuttle") == 0) {
      op_pattern = OPPAT_SHUTTLE;
    } else if (strcmp(argv[0], "-endless") == 0) {
      op_pattern = OPPAT_ENDLESS;
    } else if (strcmp(argv[0], "-reverse") == 0) {
      op_pattern = OPPAT_REVERSE;
```

```
    } else if (argv[0][0] != '-') {
      op_pattern = OPPAT_EEPROMSCRIPT;
      script = argv[0];
    }
  }
  if ((op_pattern == OPPAT_UNDEF) || (r < 0) || (nh <= 0)){
    usage();
  }
  /* Randomeness */
  for ( ; r > 0; r--) {
    (void)random_get16();
  }
  /* Run emulator */
  if (op_pattern == OPPAT_EEPROMSCRIPT) {
    emu_script_eeprom(nh, script);
  } else {
    emu_script_builtin(nh);
  }
  return 0;
}

void emu_script_eeprom(int nh, char *script)
{
  /* EEPROM EMULATION */
  mem_emu_init(script);
  /* Pass 1: SYNTAX CHECK & LABEL DEFINITION */
  printf("PASS 1: SYNTAX CHECK & LABEL DEFINITION...\n");
  script_intern_labels();
  /* Pass 2: VERIFICATION */
  printf("PASS 2: VERIFICATION...\n");
  script_verify();
  /* Pass 3: EXECUTION */
  printf("PASS 3: EXECITION...\n");
  printf("*** LABEL\n");
  uint8_t lid;
  for (lid = 0; lid < N_LABELS; lid++) {
    if (script_label_memaddr[lid] != LABEL_UNDEFINED) {
      printf("  % 2d: 0x%04X (LINE=%d)\n", lid,
             script_label_memaddr[lid],
             script_label_lineno[lid]);
    }
  }
  script_rewind();
  script_st.state = SCRIPT_STATE_IDLE;
  emu_execution(nh);
}

void emu_script_builtin(int nh)
{
  emu_execution(nh);
}

void emu_execution(int nh)
{
  int nhome = 0;
  printf("*** EXECUTION TRACE\n");
  ar_state = AR_STATE_INIT;
  while (nhome < nh) {
    if (ar_state == AR_STATE_HOME) {
      printf("HOME POSITION\n");
      nhome++;
    }
    ar_state = fsm_next(ar_state);
    while (!if_toq_is_done()) {
      if_toq_next(1);
    }
  }
}

#define MONITOR_LINE_PREFIX   "     * "

void monitor_init(void)
```

```
{
  int i;
  for (i = 0; i < 2; i++) {
    monitor_trackdir[i] = 0;
  }
  for (i = 1; i <= IF_MAX_TURNOUTS; i++) {
    monitor_turnout[i] = 0;
  }
  for (i = 0; i < 2; i++) {
    state_trackdir[i] = AR_TRACK_DIR_NORMAL;
  }
  for (i = 1; i <= IF_MAX_TURNOUTS; i++) {
    state_turnout[i] = TURNOUT_DIR_NORMAL;
  }
}

void monitor_dump(void)
{
  int i;
  if (monitor_trackdir[TRACKDIR_TRACK1]) {
    printf("%s", MONITOR_LINE_PREFIX);
    printf("TRACK #1 DIR=%s\n",
           (state_trackdir[0] == AR_TRACK_DIR_NORMAL)
           ? "NORMAL" : "REVERSE");
  }
  if (monitor_trackdir[TRACKDIR_TRACK2]) {
    printf("%s", MONITOR_LINE_PREFIX);
    printf("TRACK #2 DIR=%s\n",
           (state_trackdir[1] == AR_TRACK_DIR_NORMAL)
           ? "NORMAL" : "REVERSE");
  }
  for (i = 0; i < IF_MAX_TURNOUTS; i++) {
    if (monitor_turnout[i]) {
      printf("%s", MONITOR_LINE_PREFIX);
      printf("TURNOUT #%d DIR=%s\n", i,
             (state_turnout[i] == TURNOUT_DIR_NORMAL)
             ? "NORMAL" : "REVERSE");
    }
  }
}
```

ソースファイル usage.c (使用法表示)

emuscript のコマンドラインオプションを表示します.

ソースファイル番号 32.2: usage.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "ar.h"

void usage(void)
{
  printf("emusciptsim - script emulator\n");
  printf("Usage: emuscript [OPTION] SCRIPT\n");
  printf("OPTION: \n");
  printf("  -r N   Set random seed\n");
  printf("  -h H   Finish when home is visited H times\n");
  printf("  -md    Monitor directions of tracks 1 and 2\n");
  printf("  -md1   Monitor direction of track 1\n");
  printf("  -md2   Monitor direction of track 2\n");
  printf("  -mt T  Monitor direction of turnout T (T: 1..%d)\n",
         IF_MAX_TURNOUTS);
  printf("SCRIPT: \n");
  printf("  -shuttle  Built-in script\n");
  printf("  -endless  Built-in script\n");
  printf("  -reverse  Built-in script\n");
  printf("  FILE      EEPROM script file (plain text)\n");
  printf("Examples:\n");
  printf(" emuscript -shuttle\n");
```

```
25    printf(" emuscript -r 0 -h 3 ato-reverse.txt\n");
26    printf(" emuscript -md -mt 1 -mt 2 ato-reverse.txt\n");
27    exit(0);
28  }
```

ソースファイル fatal-stub.c (エラー表示: 実装)

自動運転スクリプトのエラー表示をパーソナルコンピュータ上の画面に行うように, ファームウエアでの同名の関数に対する代替をします. ファームウエアでは表示デバイスが LED の点滅に限定されていたためにエラーに関する詳しい情報の表示ができませんでした. ここではパーソナルコンピュータの画面にエラーのある行番号やエラー原因など十分な情報を表示してデバッグ効率を高めます.

ソースファイル番号 32.3: fatal-stub.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <stdint.h>
4
5   void fatal(uint8_t err)
6   {
7     printf("FATAL ERROR (%d)\n", err);
8     exit(0);
9   }
10
11  void fatal_s(uint8_t err, uint16_t lineno,
12               const char *s)
13  {
14    printf("FATAL ERROR (%d) AT LINE %d. %s\n",
15           err, lineno, s);
16    exit(0);
17  }
18
19  void fatal_sc(uint8_t err, uint16_t lineno,
20                const char *s, char ch)
21  {
22    printf("FATAL ERROR (%d) AT LINE %d. %s: %c\n",
23           err, lineno, s, ch);
24    exit(0);
25  }
26
27  void fatal_si(uint8_t err, uint16_t lineno,
28                const char *s, uint16_t i)
29  {
30    printf("FATAL ERROR (%d) AT LINE %d. %s: %d\n",
31           err, lineno, s, i);
32    exit(0);
33  }
```

ソースファイル exec-stub.c (ハードウエアエミュレーション: 実装)

自動運転装置ハードウエアのエミュレーションを行います. ハードウエアのアクセス関数や制御関数に対する代替ですが実際には何もせず, 動作内容を画面に表示をするだけです. 返り値が必要な関数について以下に説明します.

- ar_motor_next()
  エミュレーション実行の際には関数値として偽を返すことで, 直ちに目的の車両センサが模型車両を検知したことにします. つまりセンサ停止走行 RUN-UNTIL(v,i,Sb,Ss) およびセンサ通過走行 RUN-TO(v,Sp) に対するエミュレーションは, 直ちに目的のセンサに模型車両が到達したことにします.
- ar_stay_next()
  エミュレーション実行の際には関数値として偽を返すことで, 直ちに待機時間が経過したことにします.

- if_sensor_get()
  真または偽をランダムに返します. センサ条件ジャンプ文においてジャンプ発生の有無の両方の場合をランダムに発生させて, すべての場合に対する動作確認ができるようにします. コマンドライン引数による疑似乱数の初期値を設定は, 同じ初期値を与えれば同じ乱数系列を生成することを利用して, バグの再現を可能にします.

ソースファイル番号 32.4: exec-stub.c

```
#include <stdio.h>
#include <stdint.h>
#include "ar.h"
#include "script.h"
#include "fsm.h"
#include "random.h"

extern enum ar_track_dir_e   state_trackdir[];
extern enum if_turnout_dir_e state_turnout[];
extern void monitor_dump(void);

void pr_t(void)
{
  static int t = 0;
  monitor_dump();
  printf("%d (LINE %d) ", ++t, script_st.lineno);
}

void ar_motor_off(void)
{
  pr_t();
  printf("MOTOR OFF\n");
}

void ar_motor_on(void)
{
  pr_t();
  printf("MOTOR ON\n");
}

void ar_track1_dir_set(enum ar_track_dir_e tdir)
{
  pr_t();
  if (tdir == AR_TRACK_DIR_NORMAL) {
    state_trackdir[0] = AR_TRACK_DIR_NORMAL;
    printf("TRACK #1 DIR=NORMAL\n");
  } else {
    state_trackdir[0] = AR_TRACK_DIR_REVERSE;
    printf("TRACK #1 DIR=REVERSE\n");
  }
}

void ar_track2_dir_set(enum ar_track_dir_e tdir)
{
  pr_t();
  if (tdir == AR_TRACK_DIR_NORMAL) {
    state_trackdir[1] = AR_TRACK_DIR_NORMAL;
    printf("TRACK #2 DIR=NORMAL\n");
  } else {
    state_trackdir[1] = AR_TRACK_DIR_REVERSE;
    printf("TRACK #2 DIR=REVERSE\n");
  }
}

void ar_motor_run_until(ar_speedstep_t spdstep_target,
                        ar_decp_id decpi,
                        if_sn_id sn_brake,
                        if_sn_id sn_stop)
{
  pr_t();
```

```
  printf("RUN-UNTIL SPEEDSTEP=%d DECPI=%d SNBRK=%d SNSTOP=%d\n",
         spdstep_target, decpi, sn_brake, sn_stop);
}

void ar_motor_run_to(ar_speedstep_t spdstep_target,
                     if_sn_id sn_pass)
{
  pr_t();
  printf("RUN-TO SPEEDSTEP=%d SNPASS=%d\n",
         spdstep_target, sn_pass);
}

int8_t ar_motor_next(void)
{
  return 0;
}

void ar_stay_start(ar_staytime_t st)
{
  pr_t();
  printf("STAY T=%d\n", st);
}

void ar_stay_abs_start(uint8_t t)
{
  pr_t();
  printf("STAY-ABS T=%d(=%.1f [SEC])\n", t, 0.1*t);
}

int8_t ar_stay_next(void)
{
  return 0;
}

void if_turnout_init(void)
{
  ;
}

int8_t if_turnout_action(if_to_id p,
                         enum if_turnout_dir_e d)
{
  pr_t();
  if (d == TURNOUT_DIR_NORMAL) {
    printf("TURNOUT #%d NORMAL\n", p);
    state_turnout[p] = TURNOUT_DIR_NORMAL;
  } else {
    printf("TURNOUT #%d REVERSE\n", p);
    state_turnout[p] = TURNOUT_DIR_REVERSE;
  }
}

int8_t if_sensor_get(if_sn_id sn)
{
  uint16_t r = random_get16();
  return (r & 0x01) == 0;
}

int8_t if_accessary_off(if_ac_id ac)
{
  pr_t();
  printf("ACCESSARY-OFF AC=%d\n", ac);
  return 1;
}

int8_t if_accessary_on(if_ac_id ac)
{
  pr_t();
  printf("ACCESSARY-ON AC=%d\n", ac);
  return 1;
}
```

32.3　各種オペレーティングシステムでのコンパイル方法

Linux や FreeBSD などの Unix 系オペレーティングシステムおよび Windows オペレーティングシステムにおいて emuscript をビルトする方法を説明します. コンパイラ等の開発環境は第 24.4 節 (411 ページ) を参照して下さい.

ビルドのための Makefile の内容を以下に示します. この Makefile は Unix 系オペレーティングシステムと Windows オペレーティングシステムのどちらでも make によるビルドが可能なように書き方を工夫しています. なおソースファイルのディレクトリの設定が Makefile の先頭にありますから, 各自の状況に合わせて適宜修正を行って下さい.

ソースファイル番号 32.5: Makefile

```
ifeq ($(OS), Windows_NT)
  EXEEXT = .exe
  CC = gcc
  RM = cmd.exe /C del
else
  EXEEXT =
  CC = gcc
  RM = rm -f
endif

SRC1DIR=../stub
SRC2DIR=../../firmware/AutoTrainOp-D.X
SRC3DIR=../../firmware/lib
SRC4DIR=../../firmware/PowerPack-Common

PROGRAM=emuscript$(EXEEXT)
OBJS_EMU=emuscript.o usage.o exec-stub.o mem-stub.o fatal-stub.o
OBJS_FAR=fsm.o fsm-shuttle.o fsm-endless.o fsm-reverse.o script.o \
         script-read.o script-exec.o if-toq.o \
         mem.o random.o
OBJS=$(OBJS_EMU) $(OBJS_FAR)
VPATH=$(SRC1DIR) $(SRC2DIR) $(SRC3DIR)
CFLAGS=-I. -I$(SRC1DIR) -I$(SRC2DIR) -I$(SRC3DIR) -I$(SRC4DIR)

.PHONY: all clean distclean
all: $(PROGRAM)
clean:
  -$(RM) $(PROGRAM) *.o *.OBJ
distclean:
  -$(RM) *.o *.OBJ core *.core *~

$(PROGRAM): $(OBJS)
  $(CC) -o $(PROGRAM) $(OBJS)
%.o: %.c xc.h
  $(CC) -c $(CFLAGS) -o $@ $<
```

make コマンドを実行すると自動的にビルドが行われます. なお Windows オペレーティングシステムの場合には, コマンド名は make ではなく mingw32-make となりますので注意して下さい.

課題

1. 自動運転スクリプトを作成する際は, 複数箇所に誤りがある場合が多々あります. しかし本章で紹介した自動運転エミュレータプログラム emuscript は, 1 つの誤りを表示したら即座にエラーメッセージを表示して終了します. これでは一度の実行でひとつしか誤りを発見できません. そこでプログラムを改造して, 一度の実行で可能な限り多くの誤りを発見して表示するようにして下さい.
2. 自動運転エミュレータプログラム emuscript では, 線路の接続関係を一切考慮していません. 例えばポイントの開通方向や車両センサの実際の設置に関わらず, 目的の車両センサに直ちに到達することにしてエミュレーションを進めています. しかしこれでは完全なデバッグは行えません. 線路, ポイント, 車両

センサ, フィーダ等の接続情報もエミュレータに与えて, より正確な自動運転スクリプトのデバッグが行えるソフトウエアを作成して下さい.

おわりに

鉄道模型を楽しむための様々なマイコン電子工作とファームウエアを紹介しました. 特に電子回路部分はできるだけ簡潔にして, ファームウエアの工夫により多彩な動作を目指しました. ソフトウエアの機能をうまく分割してソースコードの再利用を可能にした結果, 開発の労力を大きく削減できました. 最終的に製作した自動運転システムの全体は少し複雑ですが, 各要素を段階的に開発したので意外と簡単です. 再利用可能なソースコードの設計は回りくどく面倒に見えますが, 設計・理解・デバッグ・後々の機能拡張の点で, 結局は近道です. 抽象化が, 設計と理解を容易にする鍵です.

著者が回路やファームウエアの設計ミス, あるいは書籍化の際に記載誤りをしてるかもしれません. また鉄道に詳しくないこともあり, 趣味者の方には不満足な点も多々あろうかと思います. その場合はどうか容赦下さい.

模型列車が自動走行する様子を眺めていると, 模型列車の中に運転士が居るような神秘的な錯覚をするかも知れません. しかし本書で詳細に見てきたように, システムは単純動作をする電子部品とコンピュータプログラムの集合体に過ぎず, それらが相互に作用しながら原因と結果の因果関係に基づく現象が絶えることなく起きているだけです. ときには自分の思う通りに模型列車が動かない事もあるでしょう. そのときは模型列車の中の運転士にお祈りしたり腹を立てたりするのではなく, 起きていることを冷静に観察することが問題の原因の解明, そして最終的な解決へと向かう正しい方法です.

本書は自習用の入門書として基本的な内容に限定しています. これを出発点として, 皆様がさらに優れたものを作り出されることを期待いたします.

参考文献

[1] Roger Amos. *Complete Book of Model Railway Electronics*. 1998, 2nd edition, Haynes Publishing.

[2] Jack E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, Vol. 4, No. 1, pp. 25–30, 1965.

[3] 智田聡丞. 鉄道模型と電子工作 – 華麗な列車走行と光・サウンドで飾る. CQ 出版社, 2009 年.

[4] 長真弓. 鉄道模型のエレクトロニクス工作. ホビーテクニック, No. 49. 日本放送出版協会, 昭和 59 年.

[5] MOROP. NEM 010 Maßstäbe, Nenngrößen, Spurweiten. http://www.miba.de/morop/nem010_d.pdf, 2011. 2019 年 6 月 13 日閲覧.

[6] 内藤春雄. Arduino で楽しむ鉄道模型. 技術評論社, 2018 年.

[7] 根津達也. DCC マニュアル 2009. 機芸出版社, 2009 年.

[8] NMRA. Electrical standards for digital command control S-9.1. https://www.nmra.org/sites/default/files/standards/sandrp/pdf/s-9.1_electrical_standards_2006.pdf, 2006. 2019 年 8 月 16 日閲覧.

[9] NMRA. General standard scales S-1.2. https://www.nmra.org/sites/default/files/standards/sandrp/pdf/S-1.2%202009.07.pdf, 2009. 2019 年 6 月 13 日閲覧.

[10] Larry Puckett. *Wiring Your Model Railroad*. Kalmbach Publishing Company, 2015.

[11] NXP Semiconductors. I^2C バス仕様およびユーザーマニュアル. https://www.nxp.com/docs/ja/user-guide/UM10204.pdf, 2012. 2019 年 6 月 13 日閲覧.

[12] 曽根悟. 信号と運転保安の考え方. 鉄道ピクトリアル (特集 信号と運転の基礎), No. 655, pp. 10 – 18, 1998 年 7 月.

[13] Peter J. Thorne. *Easy-To-Build Electronic Projects for Model Railroaders*. Kalmbach Pub. Co., 1988.

[14] Peter J. Thorne. *34 New Electronic Projects for Model Railroaders*. Kalmbach Pub. Co., 1993.

[15] 山崎喜陽. あなたもギャップが決められる. レイアウト全書, pp. 161 – 165. 機芸出版社, 昭和 42 年.

[16] 山崎喜陽. 鉄道模型. カラーブックス, No. 380. 保育社, 昭和 51 年.

[17] Alois Zingl. A rasterizing algorithm for drawing curves. http://members.chello.at/~easyfilter/Bresenham.pdf, 2012. 2019 年 6 月 13 日閲覧.

[18] 角川裕次. 鉄道模型列車の自動運転装置及びその加減速度の自動調整方法. 特許第 6485789 号, 2019 年 3 月.

索引

__delay_ms(), 120
1/f ゆらぎ, 150
16 進ダンプ, 409
24FC1025, 413
24FC64, 413, 514
2N7000, 74
386, 80, 490
7805, 75
7 セグメント LED, 72, 271
　ダイナミック点灯, 271, 535

action-stub.c, 541
action.c, 532
action.h, 532
adc.c, 117
adc.h, 117
ar-decp.c, 605
ar-decp.h, 604
ar-est.c, 613
ar-est.h, 613
ar-motor.c, 607
ar-motor.h, 606
ar-stay.c, 616
ar-stay.h, 615
ar.c, 600
ar.h, 599
aspect.c, 307
aspect.h, 306
Audacity, 468
auto-sel.c, 372
AutoTrainOp.c, 576

bin2hex, 406, 466, 488, 511, 621
bindump, 409
buff.c, 501
buff.h, 501

chkroute, 536, 538, 543
chkroute.c, 540
cmd.c, 361, 436
cmd.h, 360, 436
com.c, 448
com.h, 448
config.c, 279
config.h, 279
cons.c, 446
cons.h, 446
consumer.h, 504

DCC, 49
　—デコーダ, 51, 334
　ファンクション出力, 334
defs.h, 275, 478, 519
devinfo.c, 434
devinfo.h, 434
DirDetect.c, 328
Domain-Specific Language (DSL), 36, 299, 628
D サブコネクタ, 70, 421

EEPROM, 77
emuscript, 659, 663, 671
emuscript.c, 663
exec-stub.c, 668

f-24fc1025.c, 455
f-24fc1025.h, 454
f-24fc64.c, 453
f-24fc64.h, 453
fatal-stub.c, 541, 668
fatal.c, 523, 581
fatal.h, 523, 580
fsm-endless.c, 587
fsm-reverse.c, 588
fsm-shuttle.c, 585
fsm.c, 584
fsm.h, 584

GNU Compiler Collection (GCC), 411, 483

HELLO WORLD, 93
H ブリッジ, 165

I/O エクスパンダ, 77, 386
I^2C, 382, 393
i2c-24fc1025.c, 459
i2c-24fc1025.h, 459
i2c-24fc64.c, 456
i2c-24fc64.h, 456
i2c-mcp23017.c, 397
i2c-mcp23017.h, 396
i2c.c, 394
i2c.h, 393
if-accessary.c, 597
if-sensor.c, 594
if-toq-dummy.c, 374
if-toq.c, 391
if-toq.h, 374
if-turnout.c, 399, 596
if.c, 592
if.h, 374, 390, 591
ihex.c, 443
ihex.h, 442
instate.c, 223
io.h, 114, 170, 221, 248, 274, 303, 328, 354, 429, 494, 518, 574
ir-frame.c, 363
ir-frame.h, 363

keysym.h, 360
KiCAD, 87
　(for KiCAD), 87

LED, 71
led-ir-mon.c, 359
led-ir-mon.h, 359
LED-Light-Bulb.c, 140
LED-Light-Candle.c, 151
LED-Light-FluoLamp.c, 154
LED-Light-LED.c, 119
LED-Light-Neon.c, 147
LED-Light-Winker-Bulb.c, 143

LED-Light-Winker-LED.c, 137
led.c, 304
led.h, 304
led7seg.c, 282
led7seg.h, 282
light-fsm.c, 313
light-fsm.h, 312
LSB, 149
L チカ, 93

main.c, 114
make, 411, 483
Makefile, 411, 412, 484, 543, 671
Makefile, 543
manual-run.c, 582
MCP23017, 386, 397, 514, 557, 594, 596
mcr-motor.c, 176
mcr-motor.h, 175
mcr.c, 115, 174, 206, 228, 255, 288, 316, 331, 375, 461, 505, 535, 617
mcr.h, 114, 173, 206, 228, 254, 288, 316, 331, 375, 461, 504, 534, 617
mem-stub.c, 542
mem.c, 526
mem.h, 525
MinGW, 411
misc.c, 479
MPLAB X IDE, 92
MSB, 149

NJM386BD, 80, 490

PIC
  16F18313, 83, 86, 104, 136, 146, 210, 230, 486
  16F18326, 83, 242, 320, 413
  16F18346, 83, 291
  16F18857, 83, 160, 180, 201, 267, 346, 379, 509
  16F18877, 83, 546, 620
  PICkit3, 93
  PICkit4, 93
  書き込みアダプタ, 93
play, 473
PowerPack-Basic.c, 172
PowerPack-IR.c, 354
PowerPack-SlowBoost.c, 189
pp-const.h, 171
pp-control-ir.c, 356
pp-control-mc.c, 204
pp-control-vr.c, 193
pp-control.c, 194
pp-misc.c, 190
pp-sb.c, 196
pp-sb.h, 196
pp.h, 187
producer.c, 503
producer.h, 502
PTC, 68
PWM, 35, 163
  —周波数, 163
  デューティ・オフ期間, 169

Q0.16 形式, 149

random.c, 148
random.h, 148
rc-data.c, 368
rc.c, 367
rc.h, 367
rotenc.c, 524
rotenc.h, 524
route.c, 529
route.h, 528
RouteCtr.c, 520
RS-232C ケーブル, 414

S6809, 77, 260, 262
S7136, 77, 260, 262
script-dummy.c, 590
script-exec.c, 644
script-read.c, 640
script.c, 637
script.h, 636
servo-pulse.c, 253
servo.h, 249
ServoMotorCtr.c, 250
Signal-TimerBased.c, 309
snd-eeprom.c, 499
snd-eeprom.h, 478
SoundPlayer-EEPROM.c, 496
SoX, 473
speed-timer.c, 281
speed-timer.h, 281
Speedometer.c, 275
sw.c, 498
sw.h, 498

TB6643KQ, 76, 165, 216, 230
TBD62083APG, 74, 380
turnout.h, 221
TurnoutCtr-Motor.c, 232
TurnoutCtr-Solenoid.c, 225

usage.c, 667

wav2img, 467, 478, 483, 488
wav2img.c, 480
WAV ファイル形式, 475
WDT, 219
Writer-EEPROM.c, 431

xc.h, 543
XC8, 92
Xon/Xoff, 420
xorshift 法, 148

秋月電子通商, 87
アクセサリ制御出力, 43, 338
アクセサリ制御入力, 43, 338
アクティブ・ハイ, 167
アクティブ・ロウ, 167
アクノリッジポーリング, 426, 427
アノード, 71
インテル HEX 形式, 97, 403
ウオッチドッグタイマ, 219
オーディオエディタ, 468
オーム, 61

カソード, 71
可変抵抗器, 61
間欠カオス法, 150
擬似乱数生成, 148
逆転スイッチ, 56
ギャップ, 56, 57
キュー, 391
クロスケーブル, 422
固定小数点数, 149

固定抵抗器, 61
コンタクト, 70
コンデンサ, 62
　セラミック—, 62
　電解—, 62

サーボモータ, 246
最下位ビット, 149
最上位ビット, 149
三端子レギュレータ, 75
車両センサインターフェイス仕様, 259
出力ステアリングモード, 163, 303
巡回バッファ, 424
状態遷移図, 121
常点灯機能, 49, 183
スイッチ, 64
　オルタネイト型, 65, 66
　DIP—, 66
　モーメンタリ型, 65, 66
　ロータリ—, 66
ストレートケーブル, 422
絶対最大定格, 61
線形合同法, 148

ダイオード, 70
　ショットキーバリア—, 70
　整流—, 70
　発光—, 71
ダイナミック点灯, 271
端末ソフトウエア, 415
チャタリング, 493, 515, 524
抵抗器, 61
　可変—, 61
　固定—, 61
低速ブースト, 35, 186
　—周波数, 187
　—パルス幅, 187
定電圧レギュレータ, 75
デューティ比, 48
ドメイン固有言語, 36, 299, 623
トラックプラン, 548
トランザクション, 426
トランジスタアレイ, 380

内蔵周辺モジュール
　ADC, 106, 117
　CCP, 109, 176, 302
　CM, 325, 331
　CWG, 163, 176, 302, 303
　DAC, 107
　EUSART, 422, 448
　FVR, 107, 324, 331
　IOC, 358, 363, 364
　MSSP, 384, 394
　PORT, 105
　PPS, 110
　TMR0, 492, 493, 505
　TMR1/3/5, 185, 246, 283, 505, 535
　TMR2/4/6, 108, 176, 281, 505
　WDT, 219

ハウジング, 70
バッファリング, 424
パワーパック, 47
光変調型フォト IC, 260, 262
フィーダ, 56
フォトカプラ, 80, 322, 335
部品表, 87

ブレゼンハムアルゴリズム, 124, 316
フロー制御, 420
ポイント, 46, 52
　選択式—, 53
　非選択式—, 53
　—マシン, 53, 210, 230, 237, 239, 240
ポイント制御入力, 43, 213
ホール IC, 79, 264
ポリスイッチ, 68

無限ループ, 219
モータドライバ, 76, 165

有限状態機械, 121

リードスイッチ, 66, 263
リセッタブルヒューズ, 68
リバース区間, 59
リレー, 66
ロータリエンコーダ, 68, 515
ロータリスイッチ, 202
ローパスフィルタ, 492

ワイヤード OR, 339
割り込み, 184
　IOC—, 364
　RX—, 424
　TMR0—, 493
　TMR1—, 185, 197, 283, 535
　TMR2—, 109
　TX—, 424
　—ハンドラ, 185, 364

著者紹介

角川裕次 (かくがわひろつぐ)

山口大学工学部電子工学科卒業, 広島大学大学院工学研究科情報工学専攻博士課程前期修了. 広島大学助手, 講師, 助教授, 大阪大学准教授を経て, 現在, 龍谷大学理工学部教授. 博士 (工学). 専門は理論計算機科学. 著書『C とアセンブリ言語で学ぶ計算機プログラミングの基礎概念』(森北出版).