モータドライブノート

本稿掲載の Web ページ

http://mybook-pub-site.sakura.ne.jp/Motor_Drive_note/index.html

古橋　武

2023 年 3 月 25 日

第２版のまえがき

PIC16F1825 を用いて DC モータの回転数制御回路を組み，その回路と制御プログラムの解説書を 2012 年から UP してきました．これまでに 2 万回を超えるダウンロードがありました．多くの方にお読み（ご覧）いただき，著者として嬉しい限りです．

初版で使用した統合開発環境とコンパイラがすっかり古くなったので，該当箇所を最新のものに書き換えました．初版では統合開発環境に MPLAB IDE を，コンパイラに HI-TECH C® Compiler を使用しました．Microchip 社が無償提供している最新の統合開発環境は MPLAB®X IDE です．また，同社は HI-TECH C Compiler の提供を終了し，MPLAB® XC8 C Compiler を後継コンパイラとして無償提供しています．2019 年 9 月時点のバージョンはそれぞれ v5.25, v2.10 です．そこで，本書の該当箇所を全て MPLAB®X IDE v5.25, MPLAB® XC8 C Compiler v2.10 を用いて書き換えました．

コンパイラの変更に伴い，プログラムを一部修正しなければなりませんでした．改訂プログラムを本書と同じ Web ページ

モータドライブノート

に掲載します．ご活用下さい．

2019.9
2021.2 （再改訂）
2022.5 （再々改訂）
著者記す

初版のまえがき

ブレッドボードでマイコンによるモータ制御回路が組めないかと考えてみた．ブレッドボードはハンダづけを必要としないので，初学者が製作体験をするにはうってつけである．そう考えていた筆者はPIC® マイコンと出会うことができた．PICマイコンにはブレッドボードに差し込めるDIPタイプが多種提供されている．さっそく，PIC12F, 16F, 18F, dsPIC30F, dsPIC33Fの各種タイプについて試してみたところ，いずれも容易に組めることが分かった．ハンダづけ作業を伴わないので回路の組み替えが簡単であり，浮かんだ着想を即座に試すことができる．

世にモータ制御に関する良書は多い．これらは二つのタイプに分類できる．

(1) 理論中心に書かれたもの．

(2) 製作事例集

いずれも重要であるが，両者の間にはギャップがある．理論中心の工学書で学んだ学生は，現実のモータ制御回路をどのように組めばよいのかほとんど分からない．一方，製作事例集では具体的な製作のノウハウは書かれてはいるが，理論との接点がほとんど見られない．

本稿の狙いは以下の通りである．

(1) モータ制御回路の製作事例を紹介する．

(2) 背景となるモータ制御の理論を紹介する．

製作事例で解説するマイコンのプログラムは全て本稿と同じWebページよりダウンロード可能である．本文中にはプログラムを載せた図の図説にフォルダ名をマゼンタ色で示してある．

本稿は大学の電気工学科等にてパワーエレクトロニクスを履修した学生を対象にしている．PWMインバータなどについて一通りのことを教科書と板書による講義を通して学んだ学生に対して，モータ制御という応用問題を提供することを狙っている．本稿にてモータ制御回路の題材を選ぶに当たって，心がけたことは以下の通りである．

(1) 確実に動くこと．

(2) 部品がネット通販により入手できること．

(3) ブレッドボードで（ハンダ付けを極力しないで）作れること．

選んだ題材と用いたマイコンは以下の通りである．

(1) 直流モータの回転数制御
PIC16F1825

(2) 直流モータの電流マイナーループ制御
PIC33FJ128MC802

(3) ブラシレスモータのベクトル制御
PIC33FJ128MC802

なお，PIC16F1825のコンパイラにはMicrochip社が無償で提供しているHI-TECH C® Compiler for PIC 10/12/16 MCUs LITE v9.83 を使用する．HI-TECH Cコンパイラは ANSI Cに準拠したコンパイラであるため，C言語の基本的文法がそのまま適用できる．

しかし，このコンパイラにはマイコン内蔵の各種モジュール（タイマ，A/D 変換，PWM など）を設定するための組み込み関数が（2012 年 7 月時点では）提供されていない．各種モジュール用のレジスタの設定は煩雑で，しかも，データシートと首っ引きでないとできない．本稿では，本稿で必要とするモジュール設定用の関数の作成とその引数に判りやすい表現を採用して，その（表現とレジスタの数値との対応表である）ヘッダファイルの作成についても解説する．

本稿によりインバータ回路等を学んだ学生が，さらにモータドライブに興味を持つことができれば，著者のこの上ない喜びである．

2012.7
著者記す

目 次

第 1 章 PIC16F1825 による DC モータの回転数制御 **5**
1.1 DC モータ回転数制御回路の組み立て 5
1.2 MPLAB® X IDE, XC8 コンパイラ，New Project，デバッガ 9
1.2.1 ICSP コネクタと MPLAB® Snap の接続 9
1.2.2 MPLAB® X IDE, XC8 コンパイラのインストール方法 12
1.2.3 New Project の作成方法 13
1.2.4 Build とマイコンへの書き込み 18
1.2.5 デバッガの使用方法 21
1.3 A/D 変換モジュールと PWM モジュール 23
1.3.1 組み立て 23
1.3.2 部品 26
1.3.3 XC8 コンパイラ 30
1.3.4 タイマ 1 による割り込み 30
1.3.5 関数とヘッダファイルの作成–タイマ割り込みプログラム– 36
1.3.6 A/D 変換モジュール 45
1.3.7 関数とヘッダファイルの作成–A/D 変換モジュールのプログラム– . 50
1.3.8 PWM モジュール 55
1.3.9 関数とヘッダファイルの作成–PWM モジュールのプログラム– . . . 63
1.4 DC モータの回転数制御 69
1.4.1 組み立て 69
1.4.2 インバータ回路 75
1.4.3 フィルタ回路 80
1.4.4 PI 制御と実験結果 82
1.4.5 関数とヘッダファイルの作成 88

第1章　PIC16F1825によるDCモータの回転数制御

1.1　DCモータ回転数制御回路の組み立て

本章では2010年から発売されたPIC16F1XXXシリーズの1つであるPIC16F1825をモータ制御用マイコンに使用する．本章の製作例ではPIC16F716, PIC12F615などのPWMモジュールを内蔵している旧来のPICマイコンでも十分なのであるが，これらの8ビットPICマイコンよりも大幅な性能向上が実現された新シリーズということで，使ってみることとした．

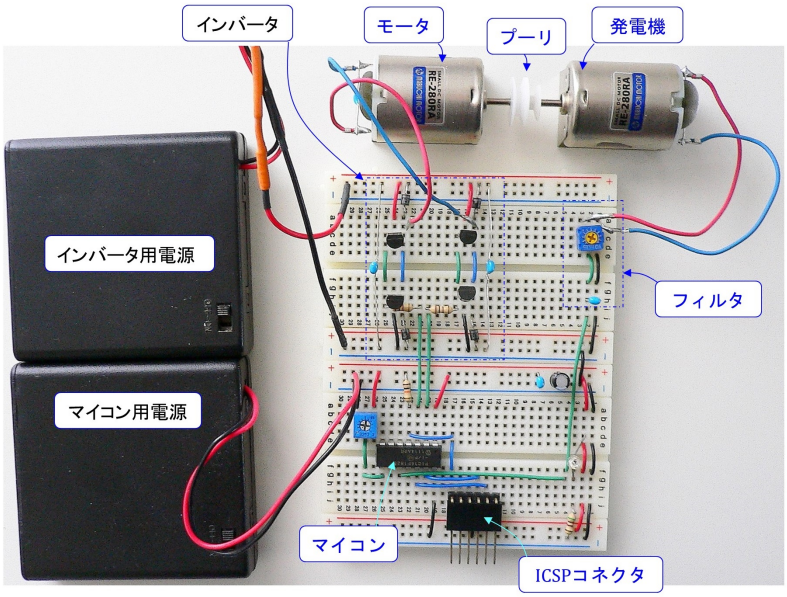


図 1.1: PIC16F1825を用いた直流 (DC) モータの回転数制御回路（全体写真）

PIC16F1825を用いた直流 (DC) モータの回転数制御回路の製作例を図1.1に示します．制御対象はマブチモータです．このDCモータをインバータにより駆動します．インバータはバイポーラトランジスタを用います．マブチモータをもう一個用意して，モータの軸同士をプーリでつなぎ，発電機として用います．この発電電圧をフィルタを通して回転数検出値としてマイコンに取り込みます．回転数制御にPICマイコン (PIC16F1825) を用います．マイコン用電源には単3の充電池4本を用いて5 [V] の電源とします．インバータ用電源は単3の電池もしくは充電池4本を用います．電源を2つに分けたのは，インバータ

回路の発生するノイズによって回転数制御回路の誤動作の可能性を減らすためです．ICSP コネクタにインサーキットデバッガ／プログラマを接続することで，マイコンへのプログラムの書き込みとデバッギングができます．

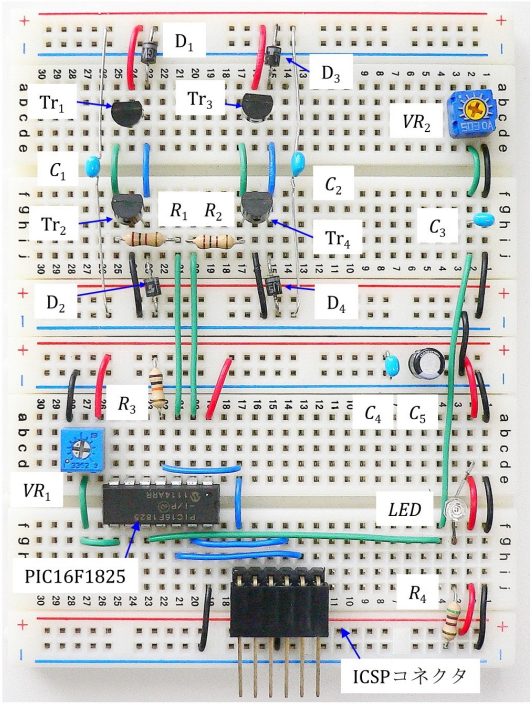


図 1.2: PIC16F1825 を用いた直流 (DC) モータの回転数制御回路（拡大写真）

図 1.2 はブレッドボード上の回路の写真です．写真では配線の様子が分かりにくいので，立体配線図を図 1.3 に示します．また，その回路図を図 1.4 に示します．インバータは Tr_1, Tr_3 に NPN 型トランジスタ 2SC2120 を用い，Tr_2, Tr_4 に PNP 型トランジスタ 2SA950 を用います．トランジスタと並列に接続してあるダイオード D_1〜D_4 にはショットキーダイオードを用います．トランジスタの駆動は PIC マイコンの PWM(Pulse Width Modulation) 制御出力によります．コンデンサ C_1, C_2 はインバータのスイッチング（オン・オフ動作）によるノイズの除去用です．モータ M_2 は発電機として用い，可変抵抗 VR_2 とコンデンサ C_3 はフィルタ回路です．この *RC* フィルタにより，M_2 の発電電圧に含まれるノイズを除去します．また，コンデンサ C_4, C_5 はノイズ対策用と制御電圧安定化用です．

回転数制御回路，インバータ，DC モータ，回転数検出回路などの詳細は以降の解説の中で詳述します．

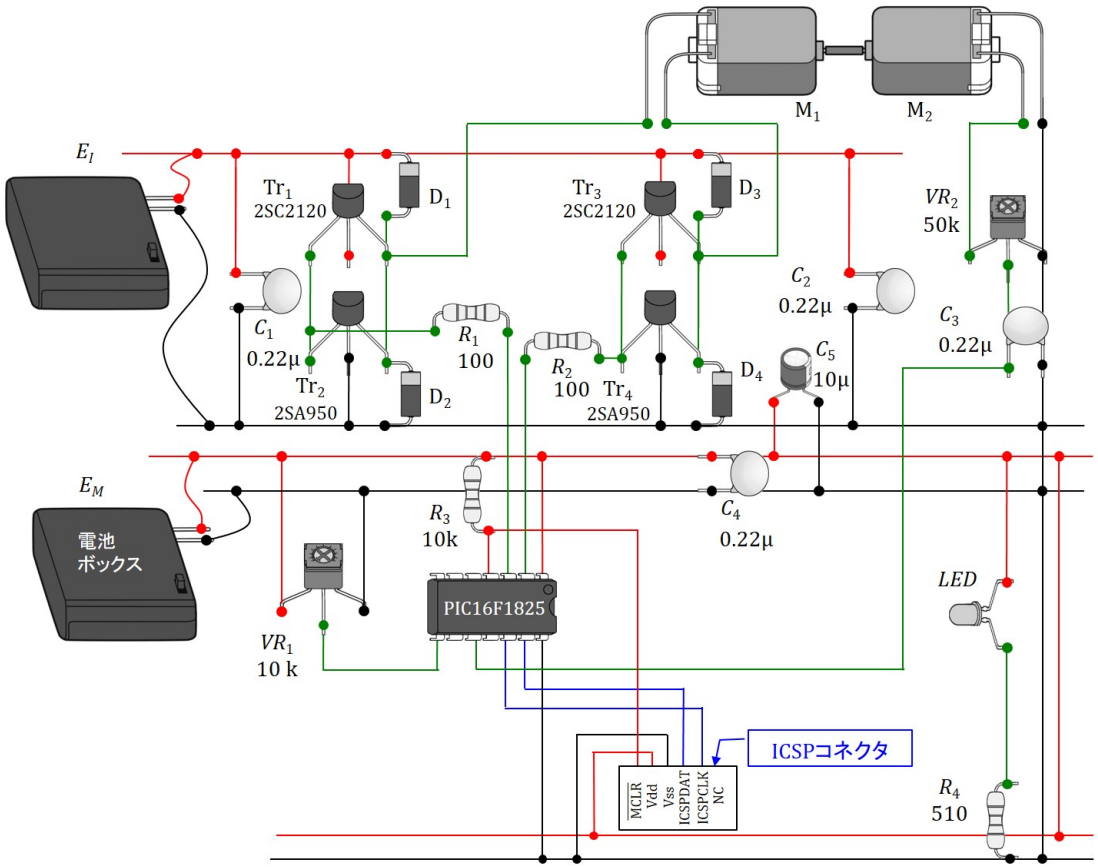


図 1.3: PIC16F1825 を用いた直流 (DC) モータの回転数制御回路 立体配線図

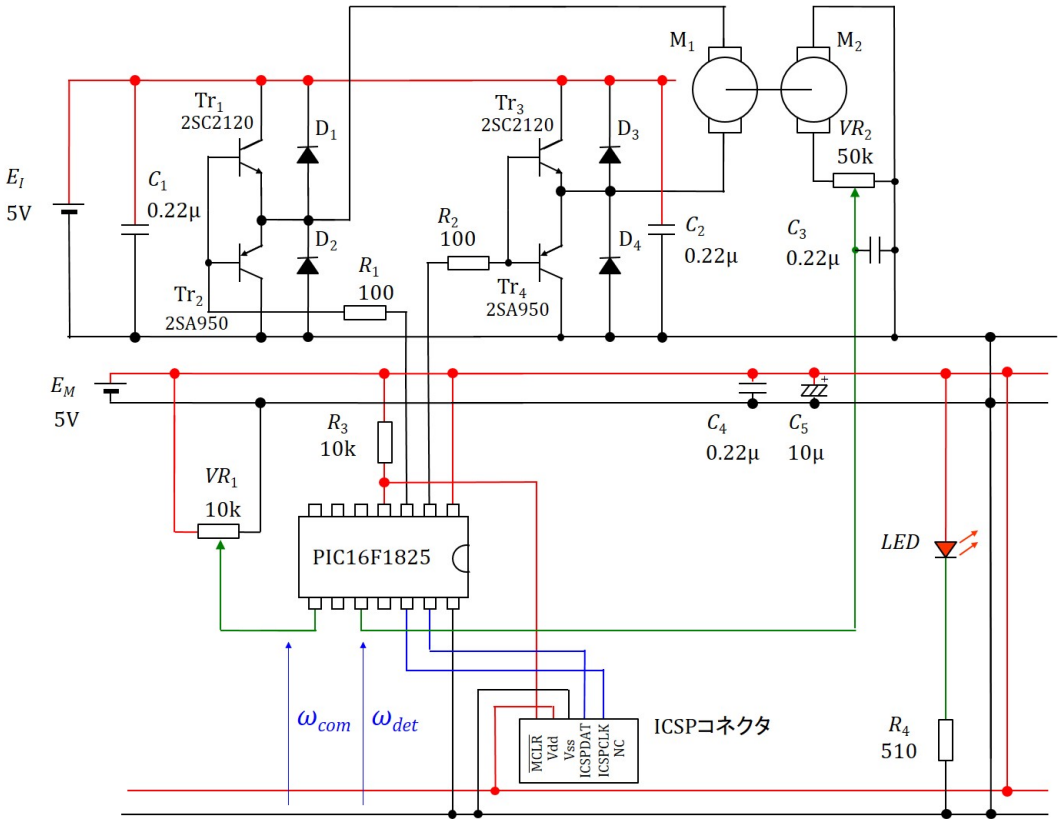


図 1.4: PIC16F1825 を用いた直流 (DC) モータの回転数制御回路図

1.2　MPLAB® X IDE, XC8 コンパイラ，New Project，デバッガ

1.2.1　ICSP コネクタと MPLAB® Snap の接続

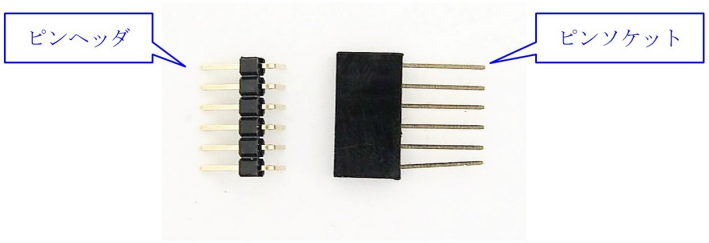


図 1.5: ピンヘッダとピンソケット

1.1 節の DC モータ制御回路のマイコンに DC_Motor_Cont のプログラムを書き込みます．マイコンへのプログラムの書き込みは図 1.2〜1.4 中の ICSP コネクタを介して行います．ICSP コネクタは，Fig. 1.5 の 6 ピンのピンヘッダとピンソケットを組み合わせて作ります．ピンヘッダは L 型を用います．

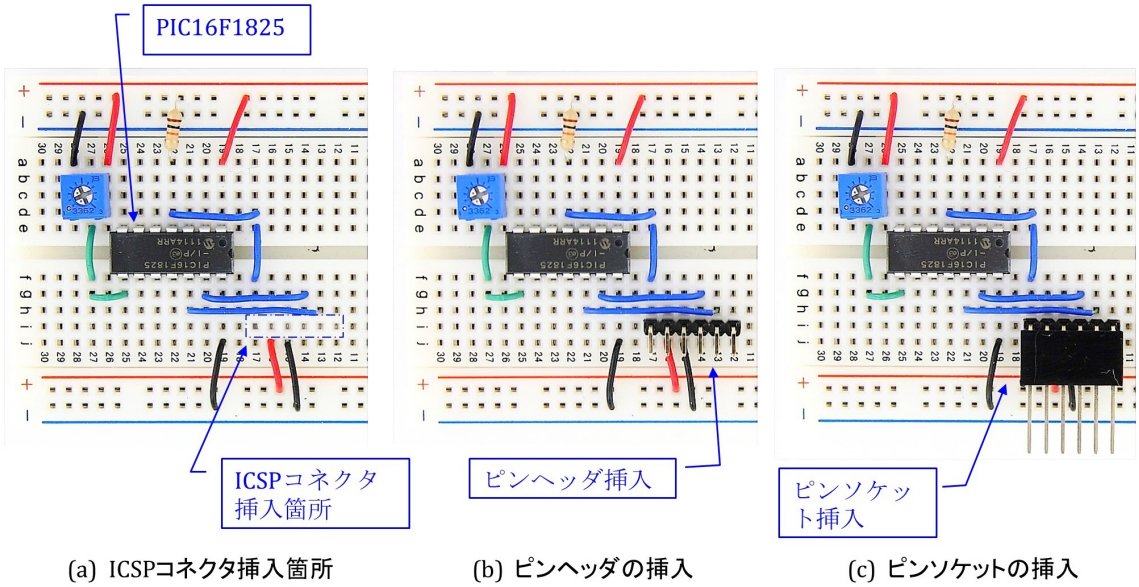


図 1.6: ICSP コネクタの製作

Fig.1.6 は ICSP コネクタの製作過程を示します．同図 (a) の破線で囲った 6 個の穴にピンヘッダを挿入します．同図 (b) がその写真です．このピンヘッダにピンソケットを挿入して ICSP コネクタができあがります．

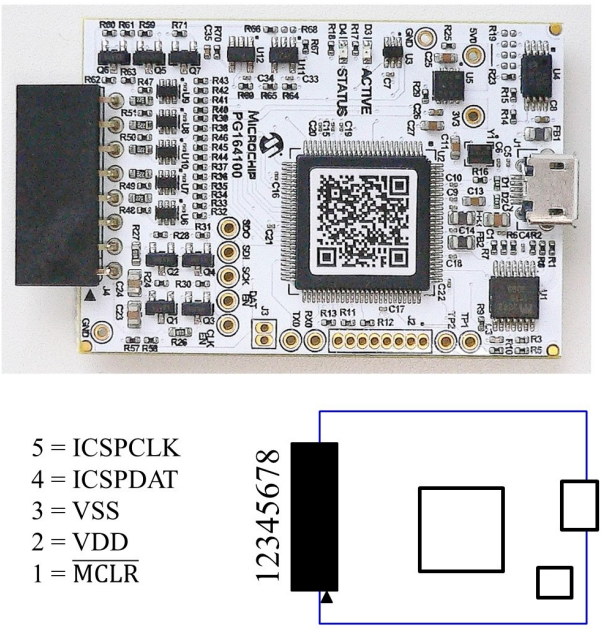


図 1.7: MPLAB® Snap

Fig. 1.7 は MPLAB® Snap の外観とピン配置を示します．MPLAB® Snap の外観とピン配置を示します．ピン穴は 8 個あり，▲印側から 1 ～ 5 番ピンまでが ICSP コネクタ用のメス・ピンです．

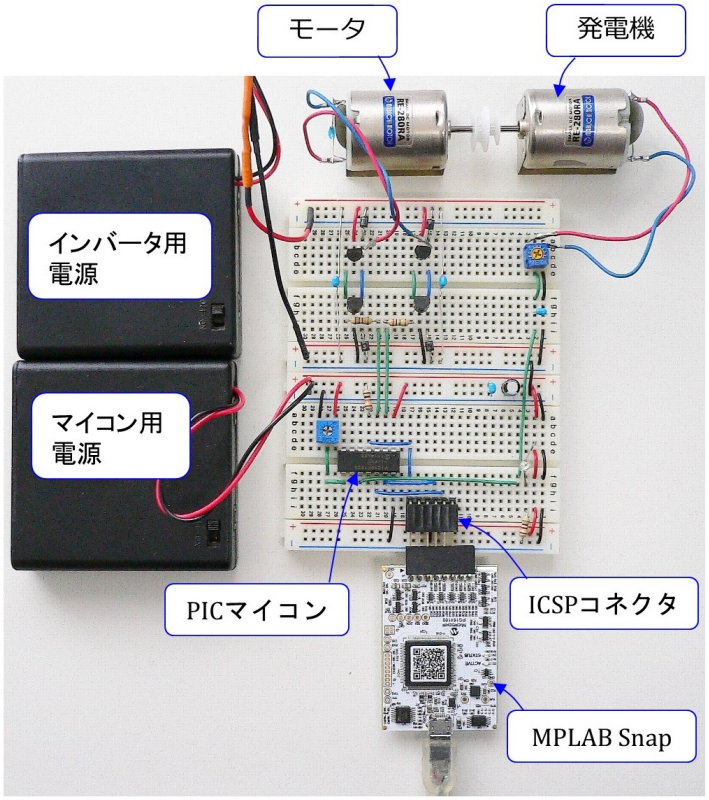


図 1.8: ICSP コネクタに MPLAB® Snap を接続

図 1.8 は ICSP コネクタに MPLAB® Snap を挿入した様子です．写真の向きに見て左端のピンを揃えます．右端の 2 穴は開放のままでよいです．MPLAB® Snap は USB ケーブルによりパソコンとつなぐことができ，パソコンよりプログラム書き込みおよびデバッギングができます．この ICSP コネクタには PICkit™4 も，配線を変更すること無く，挿入してプログラムの書き込み／デバッギングができます．

1.2.2　MPLAB® X IDE, XC8 コンパイラのインストール方法

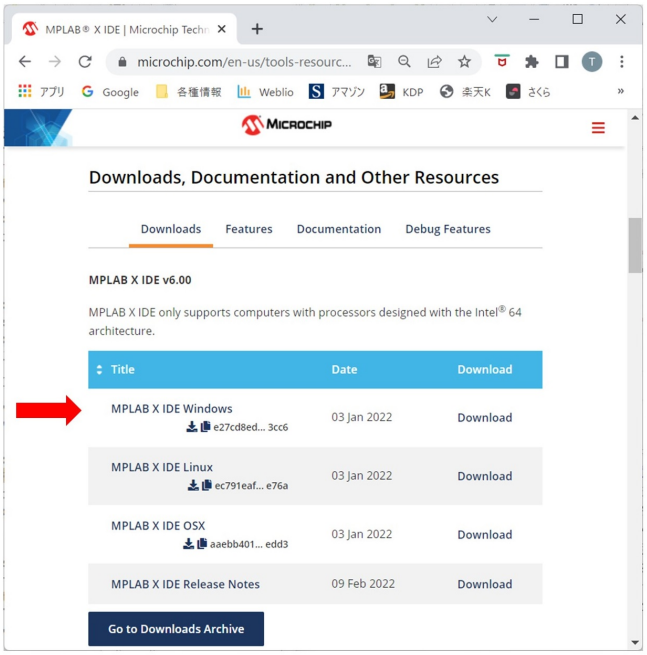


図 1.9: MPLAB®X IDE のダウンロード

本節では Microchip 社が無償提供している統合開発環境 MPLAB® X IDE (Integrated Development Environment：統合開発環境，および，MPLAB® XC8 コンパイラのダウンロード，インストール方法と使用方法の概要を紹介します．

MPLAB® X IDE は Microchip 社のホームページ → Tools and Resources → Develop → MPLAB® X IDE とたどることで図 1.9 の画面を開くことができます (2022 年 5 月時点)． MPLAB X IDE Windows を左クリック（マウスの左ボタンを 1 回クリック）すると，インストーラ (MPLABX-v6.00-windows-installer.exe) をダウンロードできます．このインストーラを立ち上げ，インストーラの推奨通りに Next ボタンを押していくことで，MPLAB® X IDE をインストールできます．無事インストールに成功すれば，C:¥Program Files (x86) および C:¥Program Files のフォルダ内に Microchip という名前のフォルダが，また，ユーザー・アカウント・フォルダ（デスクトップ・フォルダやドキュメント・フォルダ等が入っているフォルダ）内に MPLABXProjects という名前のフォルダが作られます．

同様に，MPLAB® XC8 コンパイラは Microchip 社のホームページ → Tools and Resources → Develop → MPLAB® XC Compilers → Compiler Downloads → MPLAB® XC8 Compiler v2.36 (2022 年 5 月時点) とたどることでインストーラ (Windows の場合は xc8-v2.36-full-install-windows-x64-installer.exe) をダウンロードできます．このインストーラを立ち上げ，推奨通りに Next ボタンを押していくことで，XC8 コンパイラをインストールできます．無事インストールに成功すると，C:¥Program Files¥Microchip フォルダ内に xc8¥v2.36 という名前のフォルダが作られます．v2.36¥pic¥include フォルダ内に本稿で用いるヘッダ・ファイル xc.h がダウンロードされています．

1.2.3　New Project の作成方法

本稿で解説するソースコードは，本稿と同じ
モータドライブノート
の「PIC16F1825 用 DC モータの回転数制御プログラム（圧縮フォルダ）」に入れてあります．同フォルダをダウンロード・解凍して，「￥16F1825_web_up_files」フォルダを例えばデスクトップに置いてください．「￥16F1825_web_up_files￥DC_Motor_Cont」フォルダ内のファイルを用いて，MPLAB® X IDE による編集，マイコンへの書き込み方法を記します．また，以下の設定開始前に，図 1.1 の実験回路を作成し，マイコンに 5[V] の電源電圧を印加しておいてください．ただし，インバータ用電源はオフにしておきます．そして，パソコンの USB ポートにインサーキット・デバッガ／プログラマ（例えば MPLAB Snap) をつなぎ，このデバッガ／プログラマをブレッドボード上の ICSP コネクタに挿入してください．

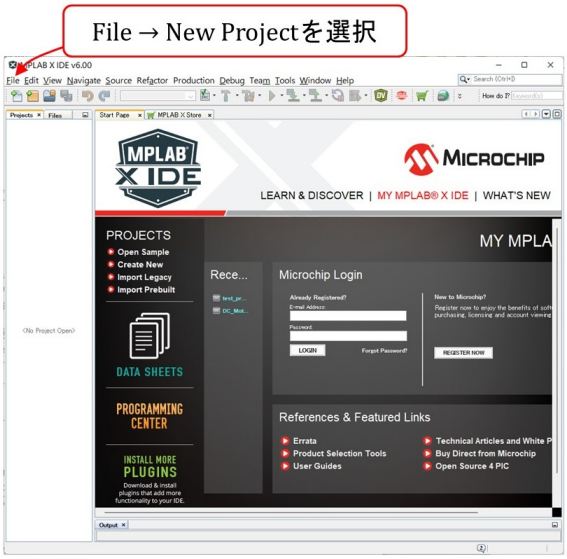


図 1.10: MPLAB® X IDE の立ち上げと New Project の設定

1.4.5 項で詳述するファイルを用いて，MPLAB® X IDE による編集，マイコンへの書き込み方法を記します．

MPLAB® X IDE のアイコンを左ダブルクリックすることで，この統合開発環境を立ち上げることができます．図 1.10 の画面が立ち上がったら，File → New Project を選択します．

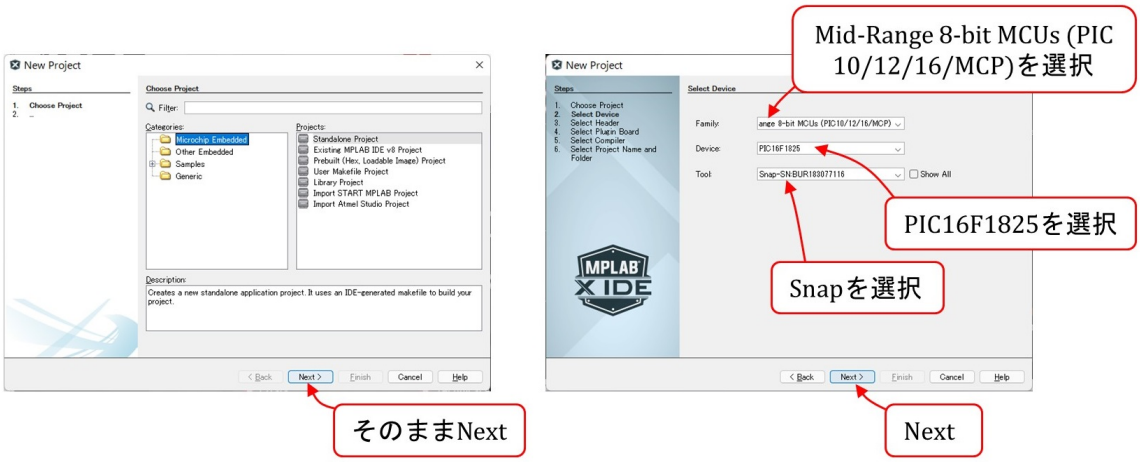


図 1.11: MPLAB® X IDE: Device, Tool 選択

次に図 1.11 のように進み，Device に PIC16F1825 を選択します．Tool には，パソコンの USB ポートにつないだインサーキット・デバッガ／プログラマがプルダウン・メニューに表示されるので，それを選択します．ここでは，MPLAB Snap を選択しています．

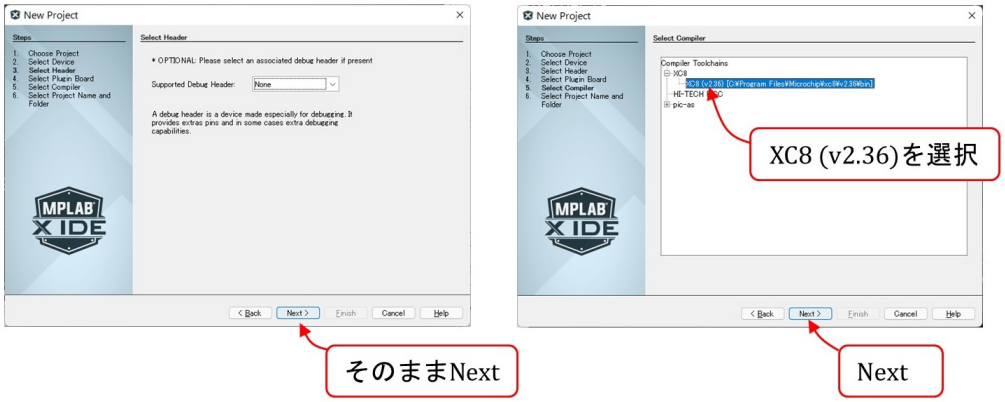


図 1.12: MPLAB® X IDE: Compiler 選択

その後は図 1.12 のようにコンパイラ (Compiler) に XC8(v2.36) を選択します．

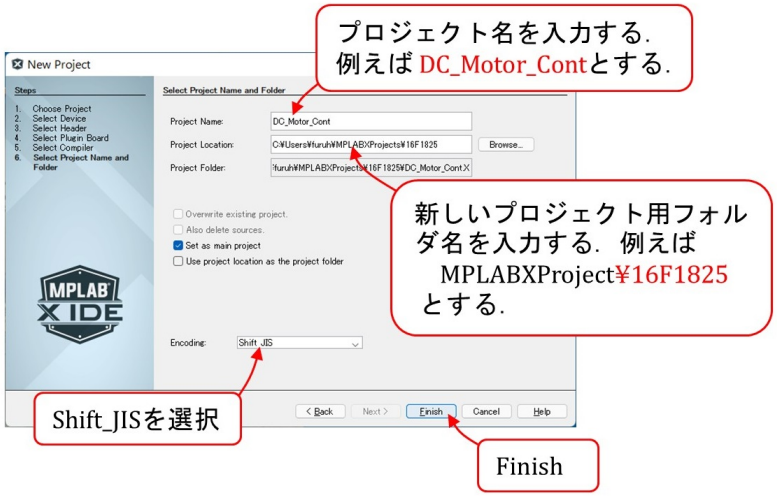


図 1.13: MPLAB® X IDE: Project Name, コード選択

図 1.13 は次に表示される画面です．Project Location に新しいプロジェクト用フォルダ名を入力します．例えば，¥16F1825 とします．そして，Project Name を，例えば，DC_Motor_Cont とします．文字コード (Encoding) に Shift_JIS を選択して，Finish ボタンをクリックします．

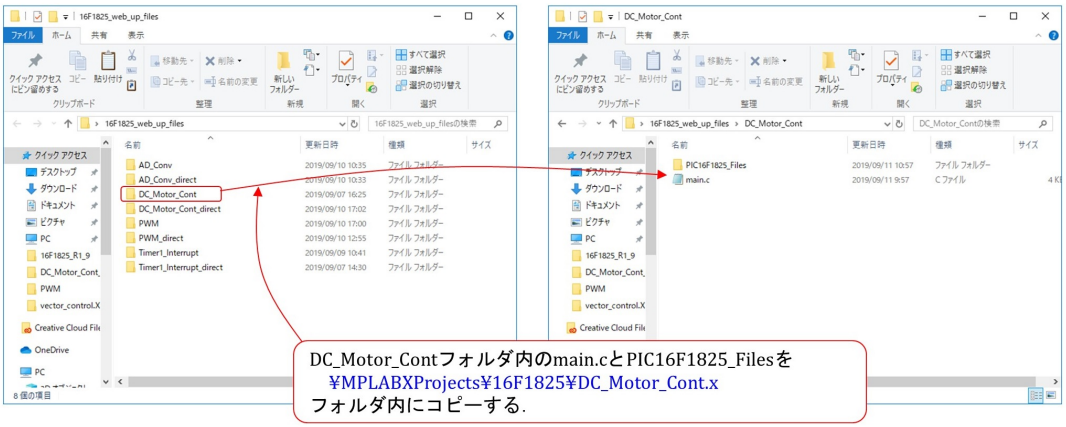


図 1.14: ソースファイル，関連フォルダの DC_Motor_Cont.X フォルダ内へのコピー

以上が完了した段階で，上の例では MPLABXProjects￥16F1825 のフォルダ内に DC_Motor_Cont.X フォルダが作られています．図 1.14 のように，このフォルダの中へ

モータドライブノート

からダウンロード・解凍しておいた「16F1825_web_up_files￥DC_Motor_Cont」フォルダ内のソースファイル (main.c) および関連ファイルフォルダ (PIC16F1825_Files) をコピーします．

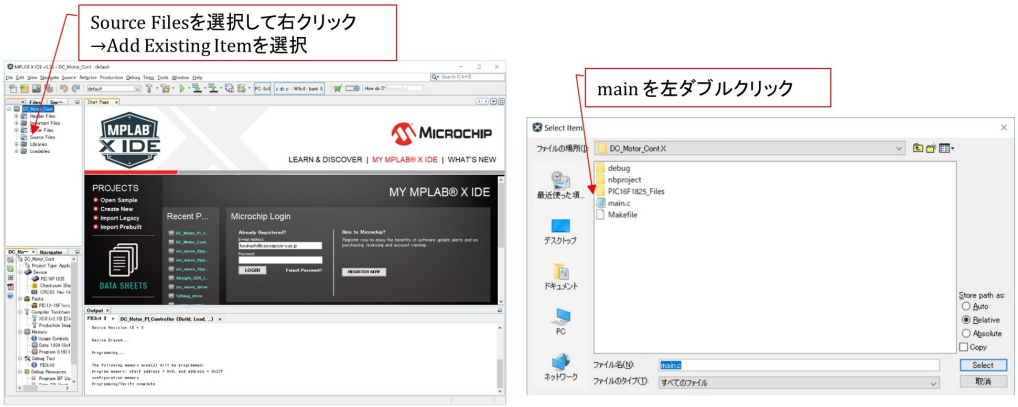


図 1.15: MPLAB® X IDE: Source file の追加

次に，これらのファイルをプロジェクトに追加します．図 1.15 のように，Source Files のフォルダを右クリックし，ADD Existing Item を選択します．そして，同図右のように main.c を選択します．

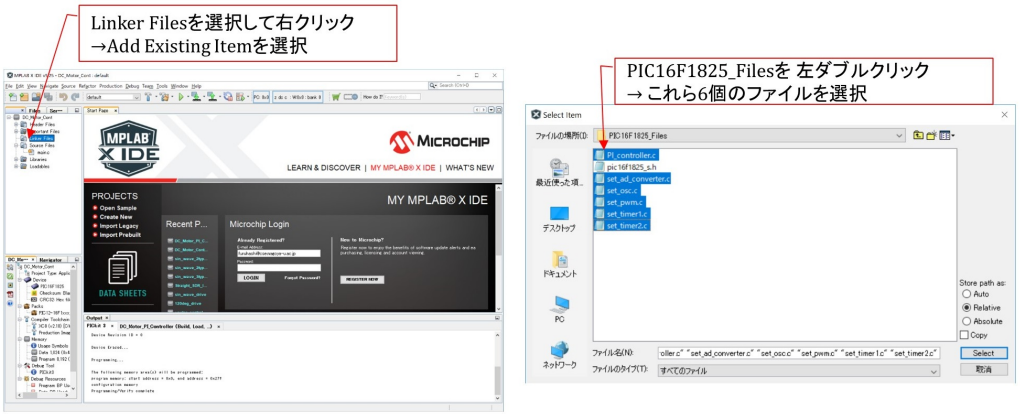


図 1.16: MPLAB® X IDE: Linker files の追加

次に，図 1.16 のように，Linker Files を右クリックし，ADD Existing Item を選択します．そして，同図右のように，PI_controller.c, set_ad_converter.c, set_osc.c, set_pwm.c, set_timer1.c, set_timer2.c を選択します．

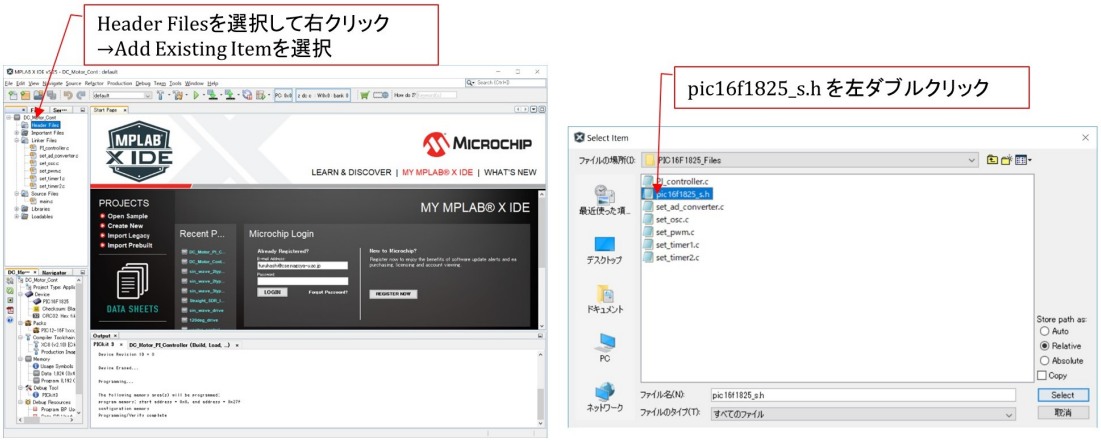


図 1.17: MPLAB® X IDE: Header の追加

最後に，図 1.17 の画面のように，Header Files を右クリックして，ヘッダファイル pic16f1825_s.h を追加します．

1.2.4　Build とマイコンへの書き込み

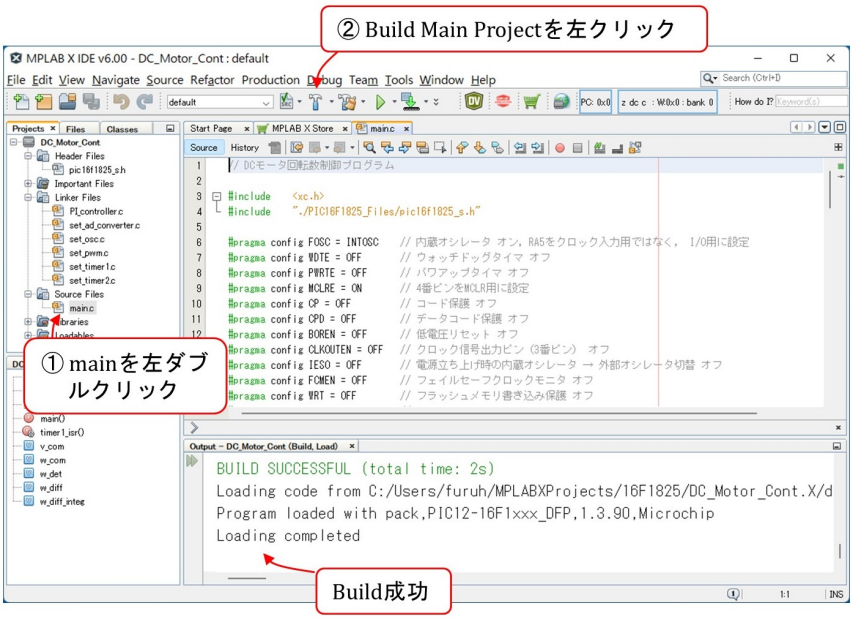


図 1.18: MPLAB® X IDE: Build

以上で MPLAB® X IDE によるプログラムの編集と，PIC マイコンへの書き込み準備ができました．図 1.18 のように，画面内の main.c を左ダブルクリックすることで，このファイルのプログラムを開くことができます．そして，Build Main Project ボタンをクリックして，同図下のように

BUILD SUCCESSFUL (total time: xxs)

のメッセージが出れば，全ての準備が完了です．

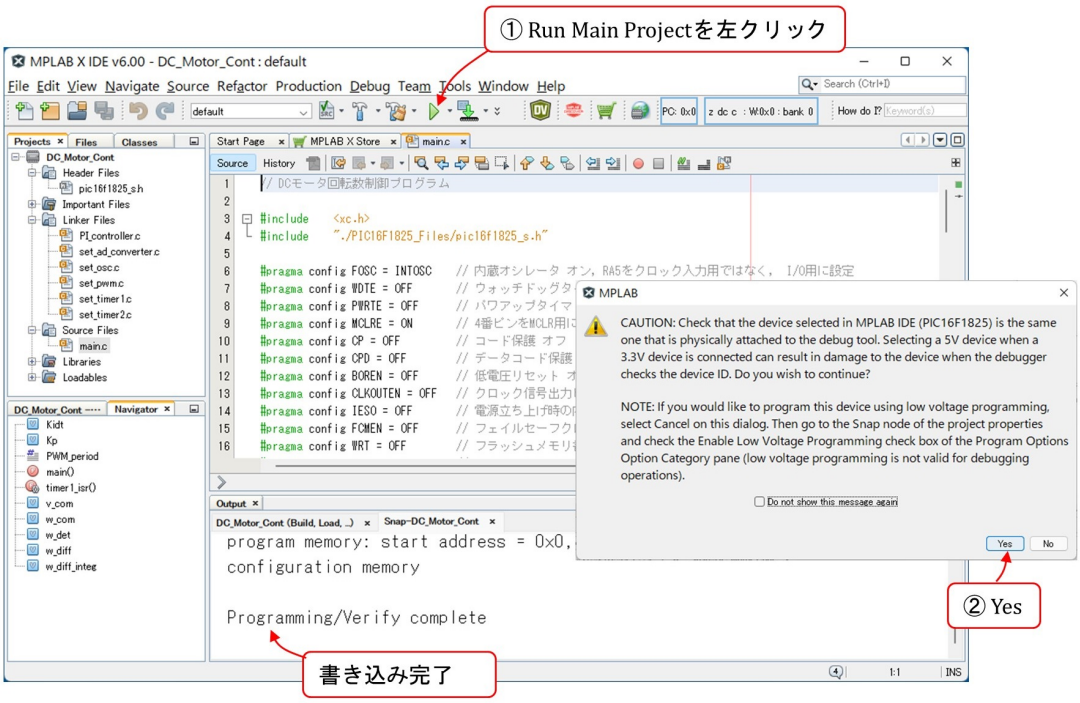


図 1.19: MPLAB® X IDE: Build とマイコンへの書き込み

それぞれの電池ボックスに 4 本の乾電池 (6[V]) もしくは充電池 (5[V]) を入れて，マイコン用電源のスイッチを入れます．Fig. 1.19 のように ▷ ボタン（Run Main Project ボタン）を押すと，しばらくして CAUTION 画面が表示されます．ボード上に PIC16F1825 が正しく挿入されていることを確認して OK ボタンをクリックします．そして，同図下の画面のように

Programming/Verify complete

が表示されれば，書き込み完了です．インバータ用電源のスイッチを入れるとモータが回り出します．可変抵抗器 VR_1 上面のつまみをネジ回しで回すと，モータの回転数が変わります．

もし，つぎのエラーメッセージが出た場合には

You have set the program speed to Normal. The circuit on your board may require you to slow the speed down. Please change the setting in the tool properties to low and try the operation again.

図 1.20 の手順に従って，Program Speed を Low に設定してください．

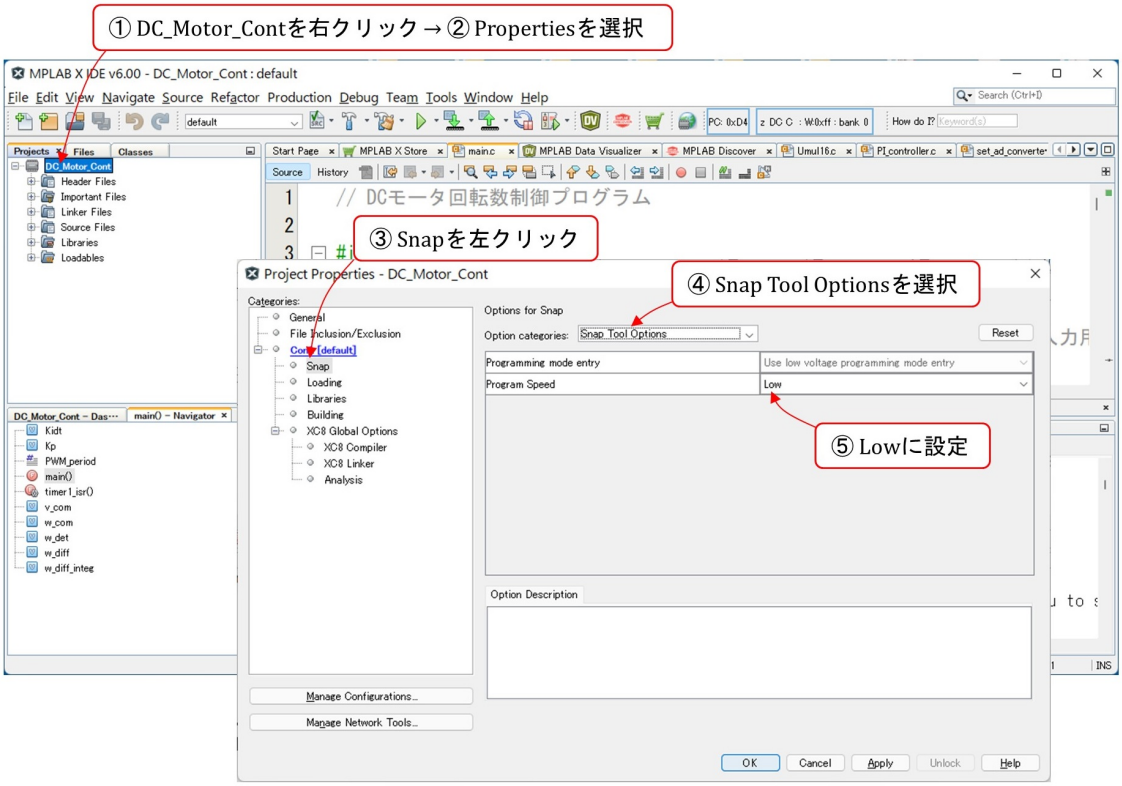


図 1.20: Program Speed を Low に設定する手順

1.2.5　デバッガの使用方法

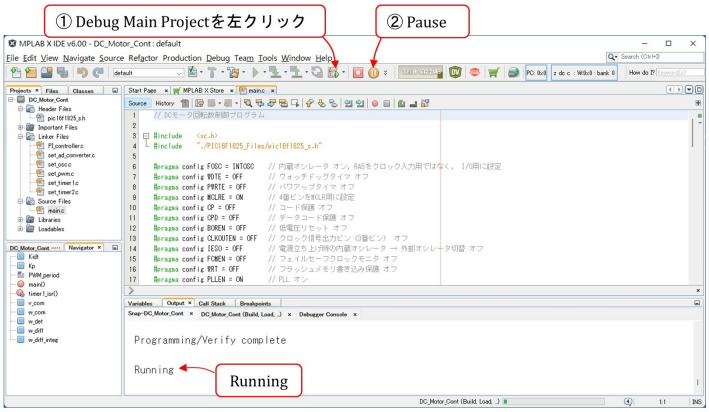


図 1.21: デバッガの起動と一時停止

プログラムの作成／改変時にはデバッガが便利です．Fig. 1.21 の Debug Main Project ボタンを左クリックすることで，プログラムのビルドとマイコンへの書き込み，デバッガの起動を実行します．マイコンへの書き込みが終了すると，画面下に Running のメッセージが現れます．Pause ボタンを左クリックすることで，プログラムを一時停止できます．このとき，モータが最高速で回り出すので，Pause ボタンを押したらただちにインバータ用の電源を切りましょう．

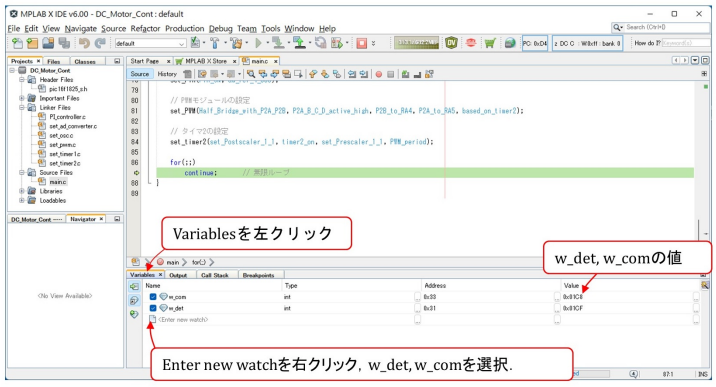


図 1.22: 表示データの指定

Variables のタブを左クリックすると，停止ボタンを押した時点のグローバル変数の値，レジスタの値などを表示させることができます．例えば，1.4.5 節のプログラムは，グローバル変数として回転数指令値 w_com, 回転数検出値 w_det などを持っています．Enter new watch を右クリックすると，プルダウンメニューが表示されます．New Watch を選択すると，表示可能なレジスタおよび変数の一覧が表示されます．その中から w_com, w_det を選択して OK ボタンを押すと，図 1.22 の様にこれら変数の値が表示されます．表示形式は 10 進，16 進，2 進を選べます．

なお，MPLAB Snap および PICkit4 のデバッガはしばしば勝手に一時停止してしまいます．モータのブラシ・整流子機構から発生する火花ノイズが原因と推定されます．モータを回さなければ誤停止は起きません．ノイズ対策を試行錯誤しましたが，筆者の力不足で，有効な対策を見つけられません．

モータを回す前段階のタイマモジュール，A/D 変換モジュール，PWM モジュールのプログラム開発には，MPLAB Snap および PICkit4 のデバッガが誤停止無く使え，重宝します．

PICkit3 のデバッガはモータを回しても誤停止無く使用できます．

1.3　A/D 変換モジュールと PWM モジュール

1.3.1　組み立て

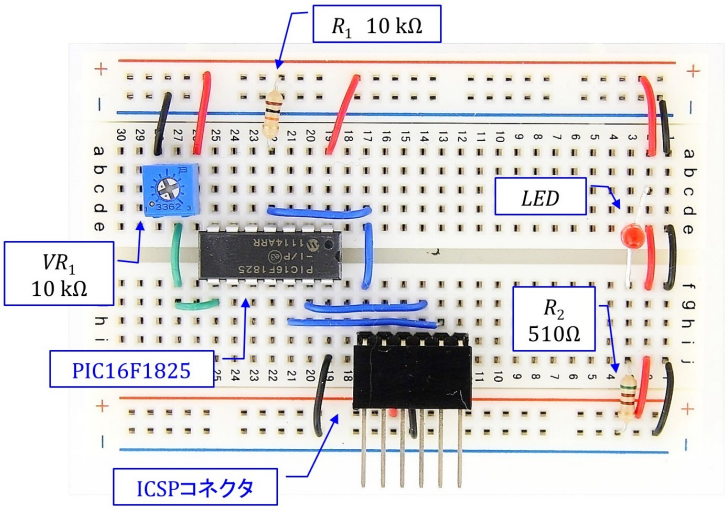


図 1.23: PIC16F1825 を用いた A/D 変換モジュールと PWM モジュールの実験回路

回転数制御に必要な PIC マイコン内のモジュールは A/D(Analog/Digital) 変換モジュールと PWM モジュールである．本節ではこれらモジュールとその利用方法およびマイコンの周辺部品について解説する．

図 1.23 はブレッドボード上に製作した実験回路である．PIC16F1825 を用いて，内蔵の A/D 変換モジュールと PWM モジュールの実験を行うための回路である．主な部品は PIC マイコン (PIC16F1825)，可変抵抗器 VR_1，ICSP コネクタ，LED とブレッドボードである．ブレッドボードは回路製作に際してハンダ付けを要しないため，回路の製作，改変が容易であり，回路の試作に便利である．可変抵抗器はマイコンの A/D 変換モジュールへの入力電圧の調整用である．ICSP コネクタはマイコンへのプログラムの書き込みとデバッグ用であり，本書ではこのコネクタに PICkit3 を接続して，プログラムの書き込みとデバッグを行う．LED は電源ランプとして用いている．

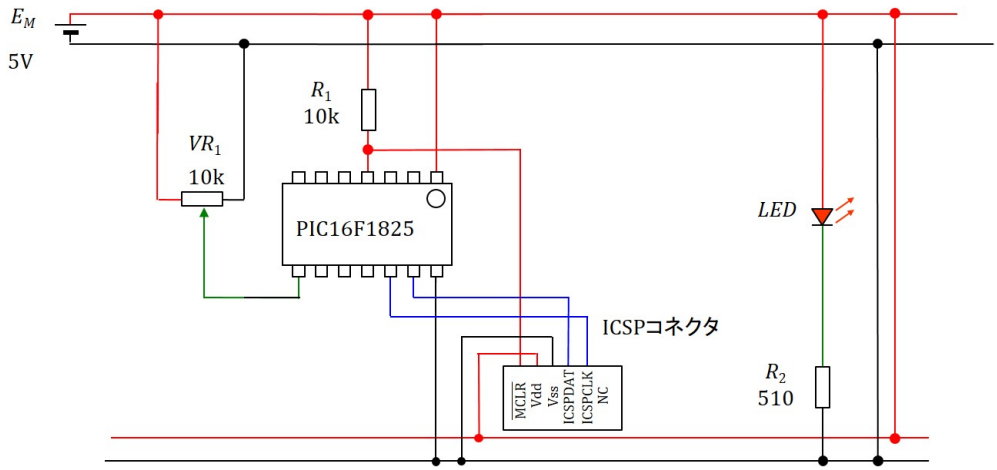


図 1.24: PIC16F1825 を用いた A/D 変換モジュールと PWM モジュールの実験回路図

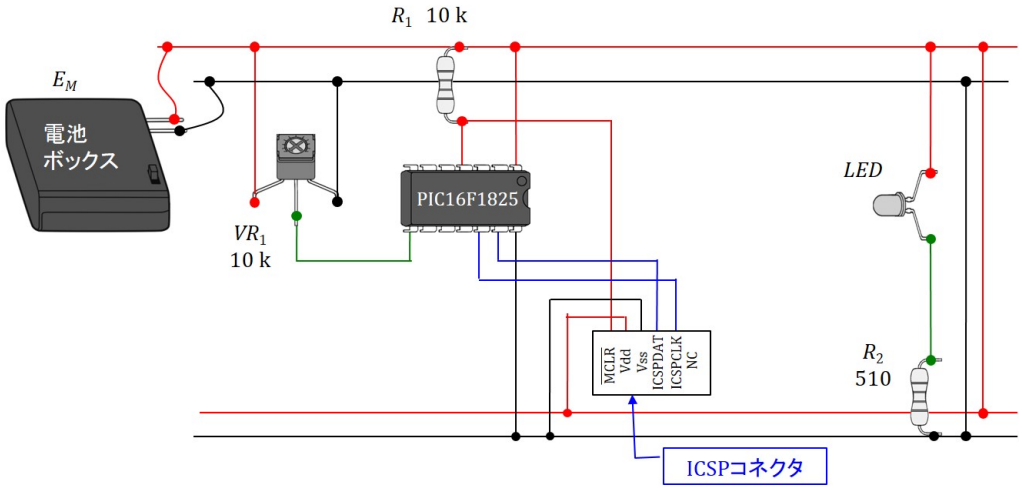


図 1.25: PIC16F1825 を用いた A/D 変換モジュールと PWM モジュールの立体配線図

図 1.24 は実験回路の回路図，図 1.25 は立体配線図である．表 1.1 にこの実験回路に使用する部品およびデバッガ／プログラマをまとめて示す．表には令和元年 9 月時点での部品入手先の例を記してある．いずれもネット通販である．ただし，値段に送料は含まれていない．電源には充電池 4 本を使用して 5 [V] の電圧，もしくは，乾電池 4 本を使用して 6 [V] の電圧を得る．乾電池 3 本で 4.5[V] としても問題なく動く．以降，回路内の各部品について順次解説し，その後に A/D 変換モジュールと PWM モジュールの実験用プログラムについて説明する．

表 1.1: 部品表

					（2019年9月時点）
品名	型式	個数	単価	値段	入手先の例
PICマイコン	PIC16F1825-I/P	1	150	150	秋月電子通商
抵抗	510Ω, 1/4W　100個入り	1	100	100	〃
〃	10kΩ, 1/4W　100個入り	1	100	100	〃
半固定ボリューム	10kΩ	1	40	40	〃
LED	3mm 赤	1	10	10	〃
ピンヘッダ	L型オス, 1×6 (6P)	1	10	10	〃
ピンソケット	1×6 (6P)	1	40	40	〃
ブレッドボード	EIC-801	1	270	270	〃
電池ボックス	単3×4本 フタ付きプラスチック・スイッチ付き	1	110	110	〃
耐熱通信機器用ビニル電線	2m×10色 導体径0.65mm 単芯	1	620	620	〃
インサーキット　デバッガ／プログラマ	PICkit 3	1	5000	5000	〃
充電器（充電池4本つき）	単3×4本用	1	3500	3500	アマゾン
充電池	単3×4本	1	1071	1071	〃
			総計	11021	円

1.3.2　部品

PIC マイコン　- PIC16F1825-

表 1.2: PIC16F1825 の主な仕様

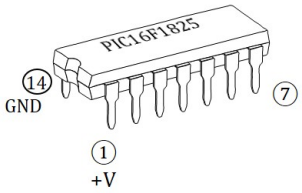
(a) PIC16F1825の主な仕様

クロック周波数	32MHz
プログラムメモリ	8 kWords
データメモリ	1 kbytes
10ビット AD変換器	8 ch
PWM出力	ハーフ/フル ブリッジ
8/16ビット タイマー	4/1個
電源電圧	1.8～5.5 V

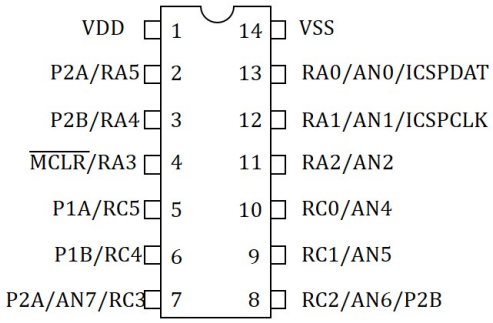
(b) PIC16F716の主な仕様

クロック周波数	20MHz
プログラムメモリ	2 kWords
データメモリ	128 bytes
8ビット AD変換器	4 ch
PWM出力	ハーフ/フル ブリッジ
8/16ビット タイマー	2/1個
電源電圧	2.0～5.5 V

(a) 外観



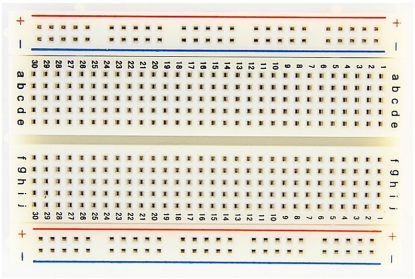
(b) ピン番号



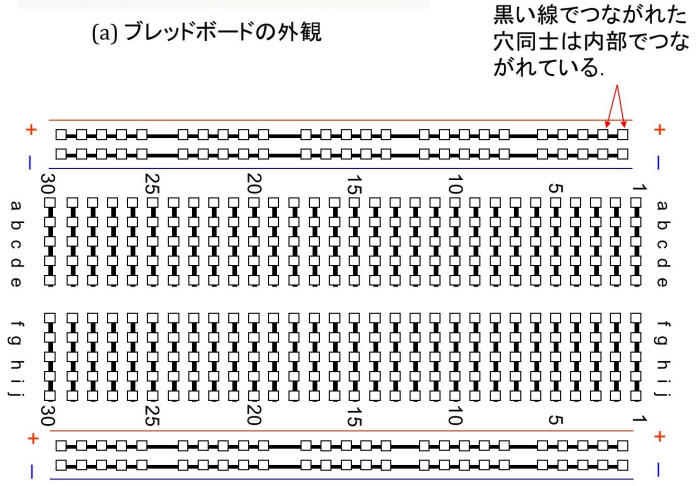
(c) ピン配置

図 1.26: PIC16F1825 のピン配置

PIC マイコン (PIC16F1825) の主な仕様を表 1.2(a) に示す．同表 (b) には旧来シリーズの代表例の 1 つである PIC16F716 の仕様を比較として示す．PIC16F1xxx シリーズのマイコンは 2010 年から発売された新シリーズであり，旧シリーズの PIC16Fxxx のクロックが最高 20MHz であったのに対して，最高 32MHz の高速動作を特徴とする．また，プログラムを格納するプログラムメモリ，データを格納するデータメモリともに容量が拡充されている．図 1.26 には PIC16F1825 の外観とピン番号の見方，およびピン配置を示す．このマイコンは 14 個の電極を持ち，電極はピンと呼ばれる．ピン番号はマイコンを同図 (b) のように見たとき，凹みの左下のピンを 1 番として反時計回りに付けられている．ピン配置図には PIC16F1825 のピンの機能のうち，本節に関係するものを記してある．VDD には電池の＋側を接続し，VSS には電池の－側を接続する．また，P2A/RA5 のように/記号で併記された機能は，プログラムの設定により使い分けできることを意味する．各機能



(a) ブレッドボードの外観



(b) ブレッドボードの穴のつながりの様子

図 1.27: ブレッドボード

についてはプログラムの解説の際に記す．詳細は Microchip 社の Web ページから無料でダウンロードできるデータシート”PIC16F1825 Data Sheet” を参照されたい．

ブレッドボード

図 1.27 はブレッドボードの外観とボード内部での穴同士のつながりの様子を示す．このボードでは縦に 10 個並んだ穴が 30 列あり，5 個ずつがそれぞれ内部で接続されている．また，横に 25 個並んだ穴が 4 行あり，各行の穴がそれぞれ接続されている．接続されている穴同士が黒い線でつながれている．図 1.23 の製作回路は一番上の行と下から 2 番目の行を＋電源（電池の＋側）とし，上から 2 番目の行と一番下の行をグラウンド（電池の－側）として使用している．

可変抵抗器

図 1.28 は可変抵抗器の (a) 外観と (b) 内部構造および (c) 記号を示す．図は 10 [kΩ] の可変抵抗器の例である．抵抗器正面に印字された 103 の数字は $10 \times 10^3[\Omega] = 10[\mathrm{k}\Omega]$ を意味する．この数字が 511 であれば $51 \times 10^1[\Omega] = 510[\Omega]$ である．この可変抵抗器には電極が 3 つあり，それぞれ a, b, c の記号を付けてある．a, c 電極は固定部に接続され，b 電極が

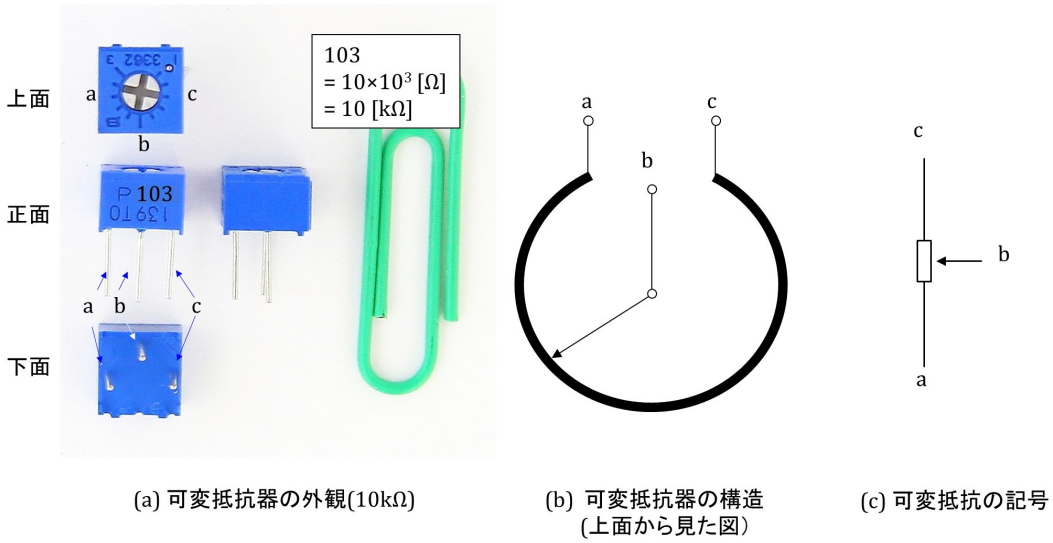


図 1.28: 可変抵抗器

可動部に接続されている．電極 a-c 間に 10[kΩ] の抵抗線が接続され，電極 b は，抵抗器上面の＋の溝のついた可動部分を回転させることで，抵抗線上を摺動する．これにより a-b 間，b-c 間の抵抗値をそれぞれ 0〜10[kΩ], 10〜0[kΩ] の範囲で可変とする．

図 1.24 ではマイコンの 8 番ピンに可変抵抗 VR_1 の b 電極が接続されている．したがって，マイコンの 8 番ピンには電源電圧の 5[V] から 0[V] の範囲の電圧が印加される．後述するように，マイコンの内部では 8 番ピンには A/D 変換モジュールの入力端子が割り当てられる．

抵抗器

図 1.29 は抵抗器の (a) 外観と (b) 記号および (c) カラーコード表を示す．図は 510 [Ω], 1/4 [W] の抵抗器の例である．抵抗器の抵抗値は色の帯で表される．10 色の色が 0〜9 の数字に対応付けられている．この対応の規則がカラーコードである．憶え方のひとつに「く（黒）ち（茶）あ（赤）だ（橙）き（黄），み（緑）あ（青）む（紫）は（灰）し（白）」がある．すなわち，「くちあ」という滝と「みあむ」という橋があると憶えておくと，おかしな名前の滝と橋ではあるが，以外と記憶にとどめておくことができる．同図 (a) の抵抗器には緑，茶，茶，金の色の帯が付されている．抵抗値は $511 = 51 \times 10^1 = 510\Omega$ である．最後の金色はこの抵抗値の精度が ±5% であることを意味する．

LED（発光ダイオード）

図 1.30 は LED（Light Emitting Diode: 発光ダイオード）の (a) 外観と (b) 記号を示す．写真は円筒部分の直径が 3mm の LED の例である．足の長い電極がアノードと呼ばれ，短い電極がカソードと呼ばれる．

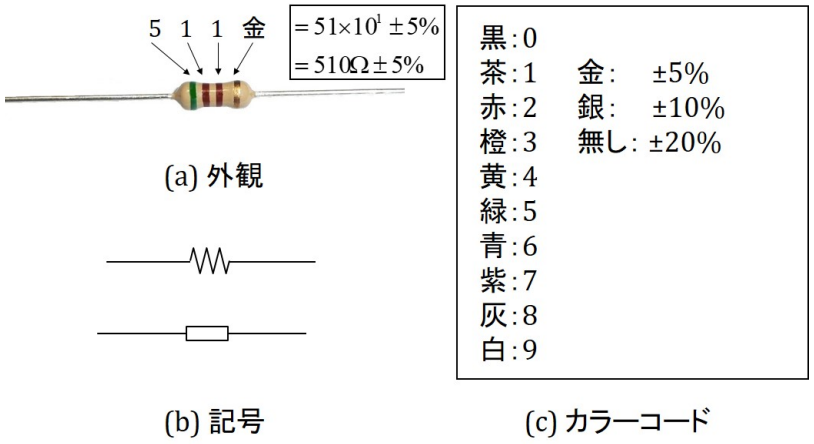


図 1.29: 抵抗器

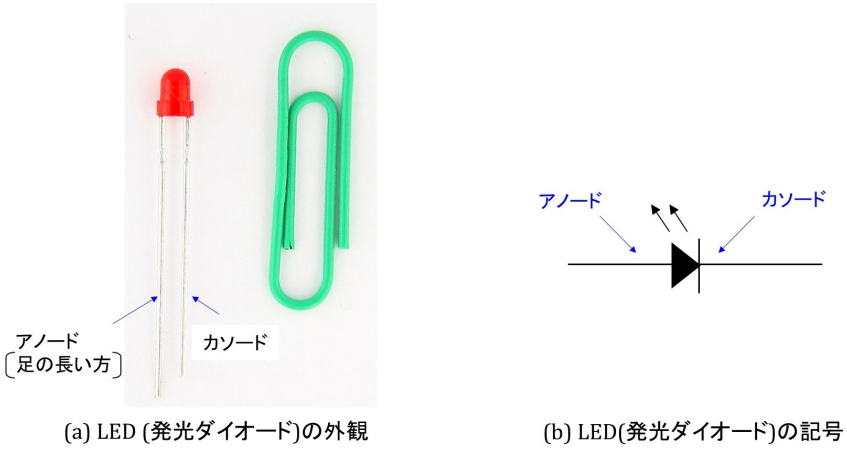


図 1.30: LED（発光ダイオード）

図 1.24 の回路では，5 [V] 電源をつないだときに LED が点灯する．この回路は LED を電源ランプとして用いている．赤色 LED の通電時のアノード・カソード間電圧（オン電圧と呼ばれる）は約 1.9 [V] であり，LED を明るく光らすには数 [mA] の電流を流す必要があるので，抵抗 R_2 の抵抗値は

$$R_2 = \frac{5 - 1.9[\mathrm{V}]}{6[\mathrm{mA}]} \approx 517 \quad [\Omega] \tag{1.1}$$

と求められる．

1.3.3　XC8 コンパイラ

本章では Microchip 社が無償で提供している XC8 C Compiler v2.10 を使用する．このコンパイラは同社提供の開発環境 MPLAB®X IDE (Integrated Development Environment : 統合開発環境）上にて利用できる．

XC8 C コンパイラは ANSI C に準拠したコンパイラであるため，C 言語の基本的文法がそのまま適用できる．しかし，このコンパイラにはマイコン内蔵の各種モジュール（タイマ，A/D 変換，PWM など）を設定するための組み込み関数が，2012 年 7 月時点で，提供されていなかった．(2019 年 9 月時点でも見当たらない.）各種モジュール用のレジスタの設定は煩雑で，しかも，データシートと首っ引きでないとできない．そこで，本稿で必要とするモジュール設定用の関数とその引数に判りやすい表現を採用して，その（表現とレジスタの用語との対応表である）ヘッダファイルを作成・解説した．本改訂版でも，これらモジュール設定用関数およびヘッダファイルを用いる．

1.3.4　タイマ 1 による割り込み

データシート (PIC16F1825_Data_Sheet) の Timer0, 1, 2/4/6 Modules の説明によると，PIC16F1825 には 8 ビットのタイマモジュール (Timer0, 2, 4, 6) と 16 ビットのタイマモジュール (Timer1) がある．8 ビットタイマの番号が跳んでいるのは，マイクロチップ社の番号付けの慣習によるとのことである．本節では Timer1 を用いて，1 [msec] の周期で割り込みを行うプログラムを示す．図 1.31〜図 1.33 にデバイスコンフィギュレーション，タイマ 1 による割り込み処理ルーチン，メインプログラムを示す．図説の赤字はこのプログラムを収納してあるフォルダ名である．本稿と同じ Web ページに圧縮ファイルをアップ[1]してあるので，ダウンロードして試してみられたい．圧縮ファイルを解凍すると「16F1825_web_up_files」という名前のフォルダが得られる．1.2.3 項の手順と同様にして，新しいプロジェクト「Timer1_interrupt_direct」を作成する．そして，新たに作られた￥16F1825_chap1￥Timer1_interrupt_direct.X フォルダへ，￥16F1825_web_up_files￥Timer1_interrupt_direct フォルダから main.c ファイルをコピーする．次に，Source Files に main.c を “add” する．

PIC マイコンのプログラムにおいて最初に行うのが，図 1.31 のインクルードファイルの設定である．本章で使用しているデバイスは PIC16F1825 であるので，

$$\#\text{include} < \text{pic16f1825.h} > \tag{1.2}$$

によりこのデバイス用のヘッダファイルをインクルードする．このヘッダファィルは XC8 C コンパイラをダウンロード，インストールすると，例えば C:￥Program Files (x86)￥microchip￥xc8￥v2.10￥pic￥include フォルダの中に自動的に保存されている．このヘッダファイルにより，データシートの中のレジスタに関する用語を用いて，レジスタへの書き込み，レジスタからの読み出しができるようになる．ただし，XC8 C Compiler User's

[1] これらのファイルが本書の記述の範囲内では正常に動くことを確認してあります．しかし，利用するに当たっては，全て読者の責任で行ってください．

```
#include <xc.h>

#pragma config FOSC = INTOSC        // 内蔵オシレータ オン，RA5をクロック入力用ではなく，I/O用に設定
#pragma config WDTE = OFF           // ウォッチドッグタイマ オフ
#pragma config PWRTE = OFF          // パワアップタイマ オフ
#pragma config MCLRE = ON           // 4番ピンをMCLR用に設定
#pragma config CP = OFF             // コード保護 オフ
#pragma config CPD = OFF            // データコード保護 オフ
#pragma config BOREN = OFF          // 低電圧リセット オフ
#pragma config CLKOUTEN = OFF       // クロック信号出力ピン（3番ピン）オフ
#pragma config IESO = OFF           // 電源立ち上げ時の内蔵オシレータ → 外部オシレータ切替 オフ
#pragma config FCMEN = OFF          // フェイルセーフクロックモニタ オフ
#pragma config WRT = OFF            // フラッシュメモリ書き込み保護 オフ
#pragma config PLLEN = ON           // PLL オン
#pragma config STVREN = OFF         // スタックオーバー／アンダーフローリセット オフ
#pragma config BORV = HI            // 低電圧リセットの設定電圧を2.7[V]に設定 (LOW=1.9 [V])
#pragma config LVP = OFF            // 低電圧（例えば3.3 [V]）でのプログラミング オフ
```

図　1.31:　タイマ 1 による割り込みプログラム（デバイスコンフィギュレーション）(Timer1_Interrupt_direct)

```
static void __interrupt()       // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    RC1 = 1;                    // ポートCのRC1(9番ピン)に1を出力する．割り込み発生のモニタリング用

    TMR1 = 0x8330;              // タイマ1のカウントアップ値の設定
                                // 入力値を初期値としてカウントアップを行い，オーバーフローで割り込みを発生
    PIR1bits.TMR1IF = 0;        // タイマ1の割り込みフラグを0にして，次のタイマ1による割り込みを受付け可とする．

    RC1 = 0;                    // ポートCのRC1(9番ピン)に0を出力する．割り込み発生のモニタリング用
}
```

図　1.32:　タイマ 1 による割り込みプログラム（タイマ 1 割り込み処理ルーチン）(Timer1_Interrupt_direct)

Guide(同じく Web ページからダウンロードできる）によると，XC8 C コンパイラでは

$$\text{xc.h} \tag{1.3}$$

のヘッダファイルを指定することで，個別のデバイスのヘッダファイルを指定しなくても，デバイスごとのヘッダファイルが自動的にインクルードされる．

ヘッダファイルのインクルードの後に必要な設定がデバイスコンフィギュレーションである．データシートの DEVICE CONFIGURATION によると，プログラムの先頭で CONFIGURATION WORD 1, 2 レジスタの各ビットを設定する必要がある．このレジスタの設定は，Fig.1.31 のように，例えば

$$\text{\#pragma config FOSC} = \text{INTOSC} \tag{1.4}$$

```
void main(void)
{
    TRISA = 0x00;                   // ポートAを出力ポートに設定する.
    TRISC = 0x00;                   // ポートCを出力ポートに設定する.

    //オシレータの設定
    OSCCONbits.IRCF = 0b1110;   // 内蔵オシレータからの各種分周クロックの中から8MHzを選択
    OSCCONbits.SCS =  0b00;     // クロックソースはコンフィギュレーション設定に従う.
                                // コンフィギュレーションでは8MHz+4逓倍PLL = 32MHzの設定

    // タイマ1の設定
    T1CONbits.TMR1CS = 0b01;    // タイマ1のクロックソースをシステムクロック(FOSC)に設定
    T1CONbits.T1CKPS = 0b00;    // タイマ1のクロック分周率を1:1に設定（分周しない）
    T1CONbits.TMR1ON = 0b1;     // タイマ1をオンとする.

    // タイマ1による割り込み設定
    PIE1bits.TMR1IE = 1;            // タイマ1による割り込み可とする.
    INTCONbits.PEIE = 1;            // タイマ1などの周辺モジュールによる割り込みを可とする.
    INTCONbits.GIE = 1;             // 全ての割り込みを可とする.

    for(;;)
        continue;           // 無限ループ
}
```

図 1.33: タイマ 1 による割り込みプログラム（メイン）(Timer1_Interrupt_direct)

とする．# pragma はプリプロセッサと呼ばれる．多くの設定項目がある中で，本稿では

FOSC = INTOSC : 内蔵オシレータオン, RA5 をクロック入力用ではなく, I/O 用に設定
MCLRE = ON :$\overline{\text{MCLR}}$ピン（4 番ピン）による強制リセットを可能とする
PLLEN = ON : 内蔵オシレータにより得られたクロックを PLL により 4 倍にする

のみ設定している．実験室レベルでは，これで十分である．

図 1.32 はタイマ 1 による割り込み処理ルーチンである．timer1_isr(void) の timer1_isr はどのような名前にしてもよいが，ここでは分かりやすく timer1_isr と記している．なお，isr は Interrupt Service Routine の略である．

本稿の最初に述べたように，ヘッダファイル pic16f1825.h をインクルードすることにより，データシート (PIC16F1825_Data_Sheet) の中のレジスタに関する用語を用いて，レジスタへの書き込み，レジスタからの読み出しができるようになる．

$$\text{RC1} = 1 \tag{1.5}$$

は，データシートの PORTC REGISTER より，PORTC レジスタの RC1 ビットに 1 を書き込んでいる．これは RC1（図 1.26 より，PIC16F1825 の 9 番ピン）に 1 を出力する命令である．このルーチンの最後では

$$\text{RC1} = 0 \tag{1.6}$$

により，RC1 に 0 を出力している．これにより，タイマ 1 による割り込み周期と，このルーチンの処理に要する時間を計測することができる．

$$\mathrm{TMR1} = 0\mathrm{x}8330 \tag{1.7}$$

は，TMR1(Timer1 Register) に 0x8330 を書き込んでいる．これはタイマ 1 のカウントアップの初期値を再設定している．データシートの SUMMARY OF REGISTERS ASSOCIATED WITH TIMER1 より，タイマ 1 レジスタは TMR1H，TMR1L の 2 つの 8 ビットレジスタのペアからなる．データシートの Timer1 Interrupt より，タイマ 1 レジスタはカウントアップされること，その値は 0xFFFF に達すると次は 0x0000 となること，そして，0x0000 となったときにタイマ 1 は割り込み処理ルーチンを起動することが分かる．割り込み処理ルーチンの中で TMR1 を再設定しないと，タイマ 1 は 0 を初期値としてカウントアップを再開してしまう．タイマ 1 のクロックが 32MHz であるので，

$$\begin{aligned} 0\mathrm{x}10000 - 0\mathrm{x}8300 &= 0\mathrm{x}7\mathrm{D}00 \\ 0\mathrm{x}7\mathrm{D}00_{(16)} &= 32000_{(10)} \end{aligned} \tag{1.8}$$

により，カウントアップの初期値を 0x8300 とすればよいことが分かる．実際には 0x8300 では、割り込み周期 T_{int} は 1 [ms] より少し長くなるため，初期値を 0x8330 とし，0x0000 までのカウントアップの所要時間を短くして，$T_{int} \approx 1$ [ms] となるようにしている．

$$\mathrm{PIR1bits.TMR1IF} = 0; \tag{1.9}$$

は，データシートの PIR1 REGISTER よりタイマ 1 の割り込みフラグを 0 にしている．TMR1IF(Timer1 Interrupt Flag) は，これが 1 であるとき，タイマ 1 による新たな割り込みを受け付けない．そこで，このフラグを 0 とすることにより，タイマ 1 による次の割り込みを受付け可としている．

図 1.33 はメインプログラムである．レジスタ，オシレータ，タイマの初期設定を行っている．

$$\begin{aligned} \mathrm{TRISA} &= 0\mathrm{x}00; \\ \mathrm{TRISC} &= 0\mathrm{x}00; \end{aligned} \tag{1.10}$$

はデータシートの PORTA TRI-STATE REGISTER, PORTC TRI-STATE REGISTER よりそれぞれ TRISA, TRISC レジスタに 0x00 を書き込んでいる．これにより PORTA と PORTC を出力ポートに設定している．

$$\mathrm{OSCCONbits.IRCF} = 0\mathrm{b}1110; \tag{1.11}$$

$$\mathrm{OSCCONbits.SCS} = 0\mathrm{b}00; \tag{1.12}$$

は OSCCON(Oscillator Control Register) レジスタへの書き込みを行い，オシレータの設定を行っている．データシートの OSCILLATOR CONTROL REGISTER より式 (1.11)

は，IRCF(Interrupt Oscillator Frequency Select bits) に 0b1110 を書き込んで，内蔵オシレータのクロックを分周して得られる各種クロックの中から 8MHz を選択している．式 (1.12) は，SCS(System Clock Select bits) を 0 とすることで，システムクロックがデバイスコンフィギュレーションの FOSC_INTOSC と PLLEN_ON により決定される設定としてる．ここで，システムクロック (FOSC) とは，CPU や周辺モジュール（タイマ，A/D 変換，PWM など) の動作クロックの意味で用いられている．なお，OSCCON レジスタの SPLLEN(Software PLL Enable bit) は，デバイスコンフィギュレーションにて PLLEN_ON である場合には無視されるため，メインプログラムでは記述を省略している．

$$\text{T1CONbits.TMR1CS} = 0\text{b01}; \tag{1.13}$$

$$\text{T1CONbits.T1CKPS} = 0\text{b00}; \tag{1.14}$$

$$\text{T1CONbits.TMR1ON} = 0\text{b1}; \tag{1.15}$$

は T1CON(Timer1 Control Register) レジスタへの書き込みを行い，タイマ 1 の設定を行っている．データシートの TIMER1 CONTROL REGISTER より TMR1CS(Timer1 Clock Source Select bits) に 0b01 を書き込むことにより，タイマ 1 のクロックソースをシステムクロック (FOSC = 32MHz) に設定している．なお，データシートでは FOSC の用語は CONFIGURATION WORD 1 の中の FOSC とシステムクロックの FOSC の両方の意味で用いられているので注意されたい．前者はシステムクロックにどのクロック源を利用するかを選定するためのものである．後者はシステムクロックそのものを指す．T1CKPS(Timer1 Input Clock Prescale Select bits) に 0b00 を書き込むことで，タイマ 1 のクロック分周率を 1:1 に設定している．これにより，タイマ 1 はシステムクロックを分周しないで 32MHz のまま用いる．TMR1ON はタイマ 1 を起動する．

$$\text{PIE1bits.TMR1IE} = 1; \tag{1.16}$$

$$\text{INTCONbits.PEIE} = 1; \tag{1.17}$$

$$\text{INTCONbits.GIE} = 1; \tag{1.18}$$

はタイマ 1 による割り込み設定を行っている．データシートの PERIPHERAL INTERRUPT ENABLE REGISTER 1 より TMR1IE (Timer1 Overflow Interrupt Enable bit) は TMR1 レジスタのカウントアップによってオーバーフロー (0xFFFF から 0x0000) となったときに割り込みをかけられるようにする．INTERRUPT CONTROL REGISTER より PEIE (Peripheral Interrupt Enable bit) は，タイマ，AD 変換，PWM などの周辺モジュールからの割り込みを受け付けられるようにする．また，GIE (Global Interrupt Enable bit) は，同じく INTERRUPT CONTROL REGISTER より，全ての割り込みを受け付けられるようにする．これら 3 つのいずれの設定が欠けても，タイマ 1 による割り込みはできない．

```
for(;;)
    continue;
```
(1.19)

は無限ループである．メインプログラムは何もしない．この間，タイマ 1 による割り込みで，タイマ 1 割り込みルーチンが 1 [ms] の周期で起動される．

図 1.34 は，図 1.31〜図 1.33 のプログラムを PIC16F1825 に書き込み・実行させたときの，RC1(9 番ピン）の電圧波形の計測結果である．横軸は 500 [ns/div] なので，一目盛りが 500 [ns] である．縦軸は 2 [V/div] である．このオシロスコープは周波数カウンタの機能も持っていて，計測結果が画面右下に表示されている．繰り返し周波数は 1.00037 [kHz] であった．また，タイマ 1 割り込み処理ルーチンの処理時間は約 750 [ns] であった．

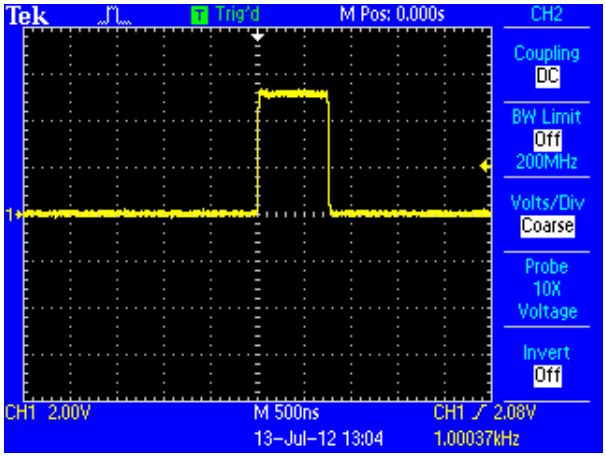


図 1.34: タイマ 1 による割り込みの実験波形

1.3.5　関数とヘッダファイルの作成–タイマ割り込みプログラム–

図 1.32, 1.33 のプログラムでは，レジスタのビットレベルでの入力を行っている．時間が経ってしまうとコメントに頼らなければこれらの処理内容を思い出すことは困難である．また，数値を含む命令の羅列は，プログラムの判読性を著しく損なってしまう．そこで，本節では判読性の高いプログラムへの改変を試みる．改変の方針は以下の通りである．

(1) 命令の羅列を関数にまとめる．

(2) 関数の名前は処理内容が判りやすいものとする．

(3) レジスタ設定の数値を処理内容の判る表現と対応付ける．ヘッダファイルに対応関係を定義する．

```
static void __interrupt()      // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    RC1 = 1;                   // ポートCのRC1(9番ピン)に1を出力する. 割り込み発生のモニタリング用

    set_timer1_count_down_ini_num(0x7BF0);
                               // タイマ1のカウントダウン値の設定.
                               // 入力値を初期値としてカウントダウンを行い，0で割り込みを発生することとなる.
    clear_interrupt_flag_of_timer1();
                               // タイマ1の割り込みフラグを0にして，次のタイマ1による割り込みを受付け可とする.
    RC1 = 0;                   // ポートCのRC1(9番ピン)に0を出力する. 割り込み発生のモニタリング用
}
```

図 1.35: タイマ 1 による割り込みプログラム_タイマ 1 割り込み処理ルーチン（判読性を高めたプログラム）Timer1_Interrupt

```
void main(void)
{
    TRISA = 0x00;     // ポートAを出力ポートに設定する.
    TRISC = 0x00;     // ポートCを出力ポートに設定する.

    //オシレータの設定
    set_osc(Int_OSC_Freq_8MHz, SysClockSource_detmd_by_Config);
                      // 内蔵オシレータ8MHzでFOSC=32MHzと設定する.

    //  タイマー1の設定
    set_timer1(TMR1_clock_source_FOSC, T1Clock_PreScale_1_1, TMR1_ON);
                      // タイマ1のクロックソース，分周率を設定して，タイマ1をオンとする.
    set_interrupt_by_timer1();
                      // タイマ1による割り込みを可とする.

    for(;;)
        continue;     // 無限ループ
}
```

図 1.36: タイマ 1 による割り込みプログラム_メイン（判読性を高めたプログラム）(Timer1_Interrupt)

図 1.35, 1.36 はその試みのプログラムである．図 1.32, 1.33 と対比してみられたい．式

(1.7) を

$$\text{TMR1} = 0\text{x}8330 \rightarrow \quad \text{set\_timer1\_count\_down\_ini\_num}(0\text{x}7\text{BF}0); \tag{1.20}$$

としている．この新しい関数名を見ただけで，この関数がタイマ 1 のカウントダウンの初期値設定をするものであることが分かる．変更のついでに，カウントアップではなくカウントダウン値として，0 になったら次の割り込みが入る設定としている．0x7BF0 はタイマ 1 の割り込み周期がほぼ 1 [ms] となるように試行錯誤で微調整した．

次に式 (1.9) を

$$\text{PIR1bits.TMR1IF} = 0; \rightarrow \quad \text{clear\_interrupt\_flag\_of\_timer1}(); \tag{1.21}$$

としている．これもコメント文が無くても分かる表現としている．

同様にして以降を

$$\begin{aligned} &\text{OSCCONbits.IRCF} = 0\text{b}1110; \\ &\text{OSCCONbits.SCS} = 0\text{b}00; \\ \rightarrow\ &\text{set\_osc}(\text{Int\_OSC\_Freq\_8MHz}, \text{SysClockSource\_detmd\_by\_Config}); \end{aligned} \tag{1.22}$$

$$\begin{aligned} &\text{T1CONbits.TMR1CS} = 0\text{b}01; \\ &\text{T1CONbits.T1CKPS} = 0\text{b}00; \\ &\text{T1CONbits.TMR1ON} = 0\text{b}1; \\ \rightarrow\ &\text{set\_timer1}(\text{TMR1\_clock\_source\_FOSC}, \text{T1Clock\_PreScale\_1\_1}, \text{TMR1\_ON}); \end{aligned} \tag{1.23}$$

$$\begin{aligned} &\text{PIE1bits.TMR1IE} = 1; \\ &\text{INTCONbits.PEIE} = 1; \\ &\text{INTCONbits.GIE} = 1; \\ \rightarrow\ &\text{set\_interrupt\_by\_timer1}(); \end{aligned} \tag{1.24}$$

としている．いずれも関数名とその引数から，これらが何をどう設定するものであるかを分かりやすくしている．

以上の改変には，新しく導入した関数の定義と引き数の定義を必要とする．引数の定義はヘッダファイルに記す．ヘッダファイルは図 1.37 のようにプログラムの先頭で次式によりインクルードする．

$$\#\text{include ".¥PIC16F1825\_Files¥pic16f1825\_s.h"} \tag{1.25}$$

上式中の pic16f1825_s.h が作成したヘッダファイルである．新しいヘッダファイルと新しい関数は全て ¥PIC16F1825_Files に入れてある．

```
#include <xc.h>
#include ".¥PIC16F1825_Files¥pic16f1825_s.h“

#pragma config FOSC = INTOSC    // 内蔵オシレータ オン，RA5をクロック入力用ではなく，I/O用に設定
#pragma config WDTE = OFF       // ウォッチドッグタイマ オフ
#pragma config PWRTE = OFF      // パワアップタイマ オフ
#pragma config MCLRE = ON       // 4番ピンをMCLR用に設定
#pragma config CP = OFF         // コード保護 オフ
#pragma config CPD = OFF        // データコード保護 オフ
#pragma config BOREN = OFF      // 低電圧リセット オフ
#pragma config CLKOUTEN = OFF // クロック信号出力ピン（3番ピン） オフ
#pragma config IESO = OFF       // 電源立ち上げ時の内蔵オシレータ → 外部オシレータ切替 オフ
#pragma config FCMEN = OFF      // フェイルセーフクロックモニタ オフ
#pragma config WRT = OFF        // フラッシュメモリ書き込み保護 オフ
#pragma config PLLEN = ON       // PLL オン
#pragma config STVREN = OFF     // スタックオーバー／アンダーフローリセット オフ
#pragma config BORV = HI        // 低電圧リセットの設定電圧を2.7[V]に設定．(LOW=1.9 [V])
#pragma config LVP = OFF        // 低電圧（例えば3.3 [V]）でのプログラミング オフ
```

図 1.37: タイマ 1 による割り込みプログラム_ヘッダファイルのインクルードとデバイスコンフィギュレーション（判読性を高めたプログラム）(Timer1_Interrupt)

pic16f1825_s.h

```
// OSCCON(Oscillator Control Register)の用語の設定

    // IRCF(内蔵オシレータの発信周波数設定)
    #define   Int_OSC_Freq_16MHz      0b1111
    #define   Int_OSC_Freq_8MHz       0b1110
    #define   Int_OSC_Freq_4MHz       0b1101
    #define   Int_OSC_Freq_2MHz       0b1100
    #define   Int_OSC_Freq_1MHz       0b1011
    #define   Int_OSC_Freq_500kHz     0b1010

    // SCS(システムクロック(FOSC)にどのクロックを利用するかを設定する. )
    #define SysClockSource_int_OSC_block      0b11
                                        // 内蔵オシレータブロックの出力を用いる.
                                        // IRCFレジスタで設定した値がFOSCとなる.
    #define SysClockSource_32_768kHz          0b01
                                        // T1OSI(2番ピン), T1OSOピン(3番ピン)に32.768kHzの
                                        // 水晶発振器を接続して, その出力を用いる.
    #define SysClockSource_detmd_by_Config   0b00
                                        //デバイスコンフィグ設定のFOSC_XXとPLLEN_xxの設定に従う.


//関数の宣言
void set_osc(unsigned int x,unsigned int y);
```

図 1.38: ヘッダファイル _OSCCON(Timer1_Interrupt)

図 1.38 はオシレータの設定を行う OSCILLATOR CONTROL REGISTER（OSCCON）のためのヘッダファイルである．OSCCON レジスタの用語に SPLLEN, IRCF, SCS がある．それぞれ，PLL の利用の有無の設定，内蔵オシレータから得られる各種周波数の中からどの周波数を利用するかの設定，システムクロック (FOSC) にどのクロックを利用するかの設定を行う．SPLLEN はデバイスコンフィギュレーションの PLLEN_ON（図 1.37）によりマスクされる（無視される）ため，ここでは省略している．

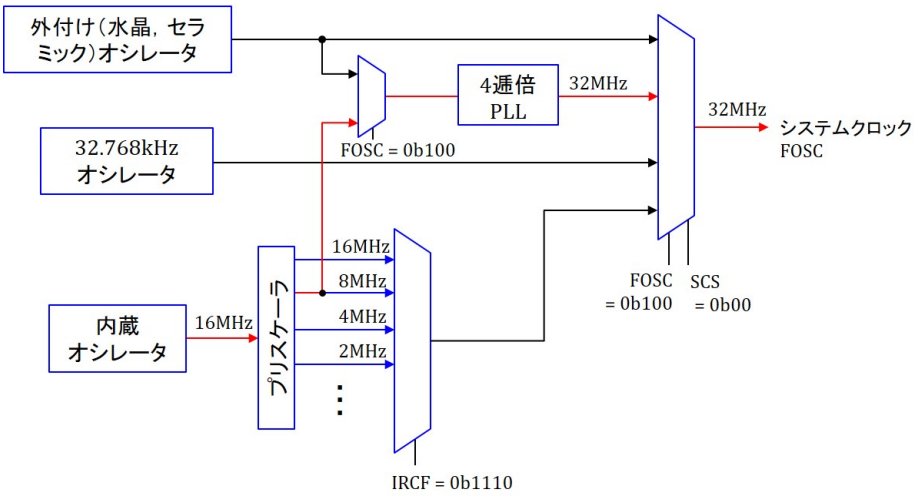


図 1.39: クロック源のブロック図

図 1.39 はデータシートのクロック源（クロックソース）のブロック図 (SIMPLIFIED PIC MCU CLOCK SOURCE BLOCK DIAGRAM) の抜粋である．IRCF は内蔵オシレータの周波数を選定する．IRCF = 0b1110 (Int_OSC_Freq_8MHz) とすることで 8MHz のクロックを選定する．そして，SCS = 0b11 もしくは 0b10 (SysClockSource_int_OSC_block) とすると，IRCF で選んだクロックをシステムクロック FOSC とする．SCS = 0b00 (SysClockSource_detmd_by_Config) とするとデバイスコンフィギュレーションで選定したクロックを FOSC とする．図 1.37 のデバイスコンフィギュレーションにて FOSC_INTOSC, PLLEN_ON とし，図 1.36 にて Int_OSC_Freq_8MHz, SysClockSource_detmd_by_Config としているので，図 1.39 に示す経路を通って，システムクロック FOSC = 32 [MHz] と設定できる．なお，4 逓倍 PLL は内蔵オシレータを利用する場合には，上記の設定の場合にのみ利用できる．この設定を，例えば Int_OSC_Freq_4MHz に変えると，4 逓倍 PLL の出力は使われず，内蔵オシレータから得られた 4 [MHz] のクロックがそのまま使われて，FOSC = 4 [MHz] となる．

pic16f1825_s.h（つづき）

```
// T1CON(Timer1 Control Register)の用語の設定

    // TMR1CS(タイマ1のクロックソースを設定する)
    #define TMR1_clock_source_CPSOSC        0b11    // Capacitive sensing oscillatorのクロックを利用する.
    #define TMR1_clock_source_T1Ck_T1OS     0b10    // 外部クロック入力(T1CK0より)or T1OSI/T1OSO入力を利用する.
    #define TMR1_clock_source_FOSC          0b01    // システムクロック(FOSC)を利用する.
    #define TMR1_clock_source_FOSC_1_4      0b00    // システムクロック(FOSC/4)を利用する.

    // T1CKPS(タイマ1に入れるクロックの分周率を設定する. )
    #define T1Clock_PreScale_1_8    0b11            // 1/8にする.
    #define T1Clock_PreScale_1_4    0b10            // 1/4にする.
    #define T1Clock_PreScale_1_2    0b01            // 1/2にする.
    #define T1Clock_PreScale_1_1    0b00            // 1/1にする.

    // TMR1ON(タイマ1のオン／オフを設定. )
    #define TMR1_ON     0b1
    #define TMR1_OFF    0b0

    //その他の用語はデフォルト設定で本書の使い方に無関係なので, 省略する. )

//関数の宣言
void set_timer1_count_down_ini_num(unsigned int i);
void clear_interrupt_flag_of_timer1(void);
void set_timer1(unsigned int a,unsigned int b, unsigned int c);
void set_interrupt_by_timer1(void);
```

図 1.40: ヘッダファイル _T1CON(Timer1_Interrupt)

図 1.40 はタイマ 1 の設定を行うレジスタ TIMER1 CONTROL REGISTER(T1CON) のためのヘッダファイルである． それぞれ，タイマ 1 のクロック源の選定 (TMR1CS)，分周率の決定 (T1CKPS)，そして，タイマ 1 の起動の設定 (TMR1ON) である．図 1.41 はデータシートの TIMER1 BLOCK DIAGRAM の抜粋である．TMR1CS によりタイマ 1 のクロック源の選定を行っている．タイマ 1 のクロックとしてシステムクロック (FOSC) を用いる場合には，TMR1CS = 0b01 (TMR1_clock_source_FOSC) により FOSC を選べる．TRM1CS = 0b00 (TMR1_clock_source_FOSC_1_4) とすれば FOSC の 1/4 分周を選べる．選定したクロックに対して，T1CKPS = 0b00 (T1Clock_PreScale_1_1) とすることで 1/1 分周を選定できる．デフォルト設定（何も設定をしないこと）で $\overline{\text{T1SYNC}} = 1$, TMR1GE = 0(TIMER1 BLOCK DIAGRAM 参照) であるので，TMR1ON = 1 により TMR1 レジスタにクロック信号 (T1CLK) を入力する．TMR1 レジスタ入力の ▷ の記号は，このレジスタがクロックの立ち上がりエッジ（クロックの立ち上がりのタイミング）を捉えることを意味する．TMR1 レジスタはカウントアップを行う．

図 1.38，1.40 の最後で式 (1.20)〜(1.24) の関数の宣言をしている．これらは OSCCON レジスタと T1CON レジスタを設定する関数である． それぞれ set_osc.c, set_timer1.c のファイルに記述してある．いずれも PIC16F1825_Files のフォルダの中に入れてある．図 1.42，1.43 に set_osc.c, set_timer1.c を示す．式 (1.20)〜(1.24) のビットレベルの設定をまとめたものである．

set_osc.c, set_timer1.c のファイルをプロジェクトに追加するには，図 1.44 のように，

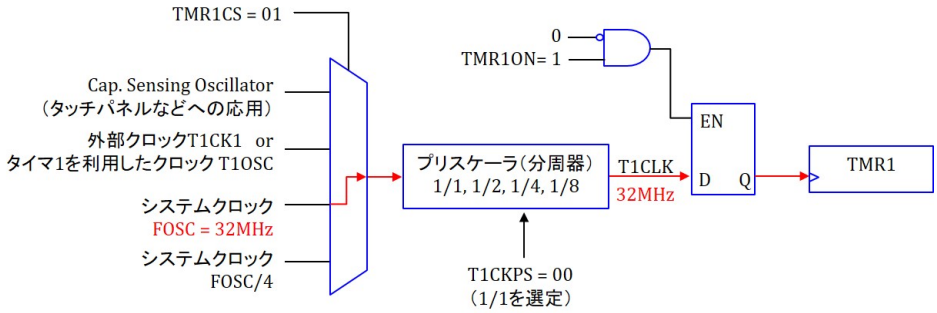


図 1.41: タイマ 1 のブロック図

set_osc.c

```
#include    <xc.h>
#include    "../PIC16F1825_Files/pic16f1825_s.h"

// OSCCON(Oscillator Control Register)の設定（ヘッダファイルpic16f1825_s.h参照）
void set_osc(unsigned int x,unsigned int y)
{
            OSCCONbits.IRCF = x;
            OSCCONbits.SCS = y;
}
```

図 1.42: OSCCON レジスタ設定関数 (Timer1_Interrupt)

Linker Files を右クリック → Add Existing Item を選択 → PIC16F1825_Files を左ダブルクリック → set_osc.c と set_time1.c を選択する．同図右がその結果の画面である．

以上のように意味の分かりやすい用語をヘッダファイルに定義し，また，レジスタ設定の命令群を関数としてまとめておけば，以降のプログラム開発におけるレジスタ設定は容易となる．関数は一度定義しておけば再利用ができる．できあがったプログラムは判読性の高いものとなる．

set_timer1.c

```
#include <xc.h>
#include "../PIC16F1825_Files/pic16f1825_s.h"

// T1CON(Timer1 Control Register)の設定（ヘッダファイルpic16f1825_s.h参照）
void set_timer1(unsigned int a,unsigned int b, unsigned int c)
{
    T1CONbits.TMR1CS = a;
    T1CONbits.T1CKPS = b;
    T1CONbits.TMR1ON = c;
}

// タイマ1はカウントアップを行う．入力値はカウントダウン値なので，この関数内で換算している．
void set_timer1_count_down_ini_num(unsigned int i)
{
    TMR1 = 0xFFFF - i + 1;
}

// タイマ1の割り込みフラグを0にして，次のタイマ1による割り込みを受付け可とする．
void clear_interrupt_flag_of_timer1()
{
    PIR1bits.TMR1IF = 0;
}

// タイマ1による割り込み設定
void set_interrupt_by_timer1(void)
{
    PIE1bits.TMR1IE =    1;        // タイマ1による割り込み可とする．
    INTCONbits.PEIE =    1;        // タイマ1などの周辺モジュールによる割り込みを可とする．
    INTCONbits.GIE =     1;        // 全ての割り込みを可とする．
}
```

図 1.43: T1CON レジスタ設定関数 (Timer1_Interrupt)

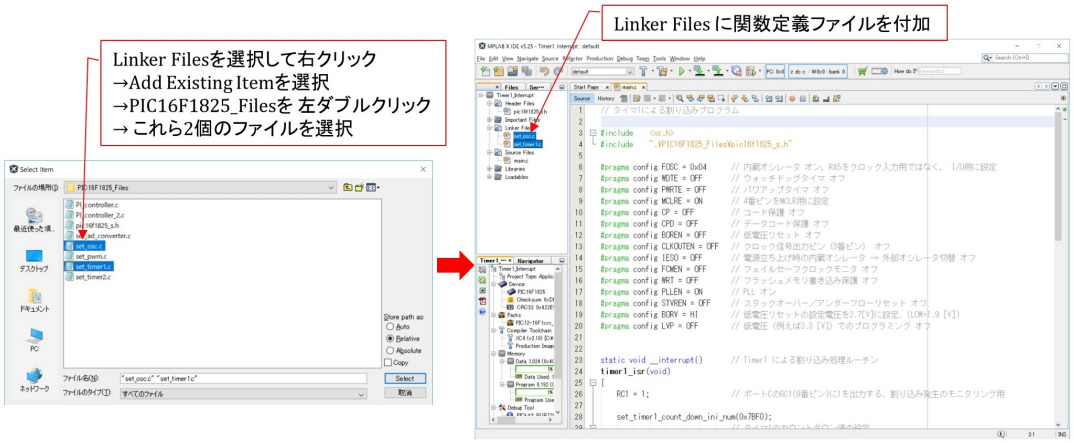


図 1.44: MPLAB 画面 _ 関数定義ファイルの読み込み

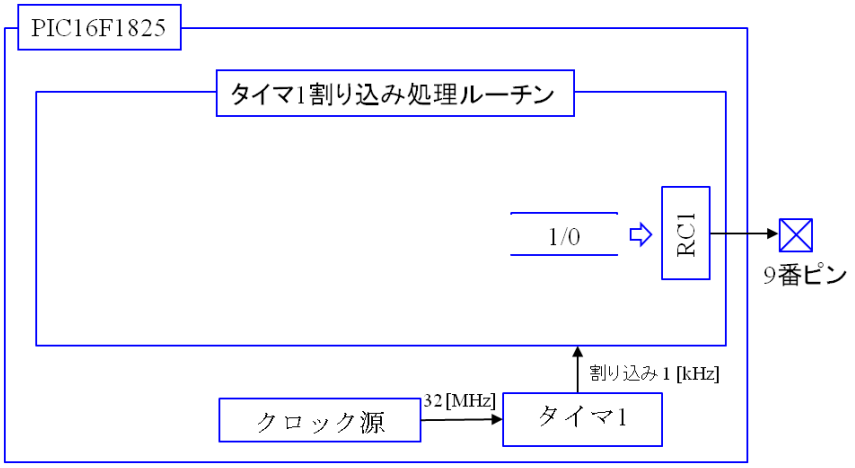


図 1.45: タイマ 1 による割り込みプログラムのブロック図

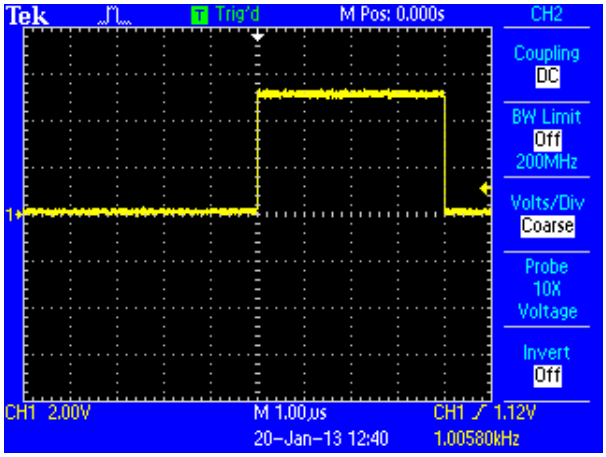


図 1.46: タイマ 1 による割り込み（判読性を高めたプログラム）の実験波形

図 1.45 は図 1.37，1.35，1.36 のタイマ 1 による割り込みを行うプログラムのブロック図を示す．タイマ 1 はクロック源から 32 [MHz] のシステムクロックを受け取り，1 [kHz] の繰り返し周波数でタイマ 1 割り込み処理ルーチンを起動する．このルーチンは処理開始時に 9 番ピンに 1 を出力し，処理終了時に 0 を出力する．図 1.46 はこのプログラムによる 9 番ピンの電圧波形の計測結果である．横軸は 1 [μs/div] である．割り込みの繰り返し周波数は 1.00580 [kHz] であった．タイマ 1 割り込みルーチンの処理時間は約 4 [μs] であった．関数を利用したことで，処理時間が延びてしまうというデメリットがあることが分かる．各ルーチン内の処理が多くなるにつれてこのデメリットは相対的に小さくなる．必要な処理を行うのに時間的余裕がある場合には，プログラムの判読性を高めることはプログラム開発の効率を高めることにつながり望ましい．

1.3.6　A/D 変換モジュール

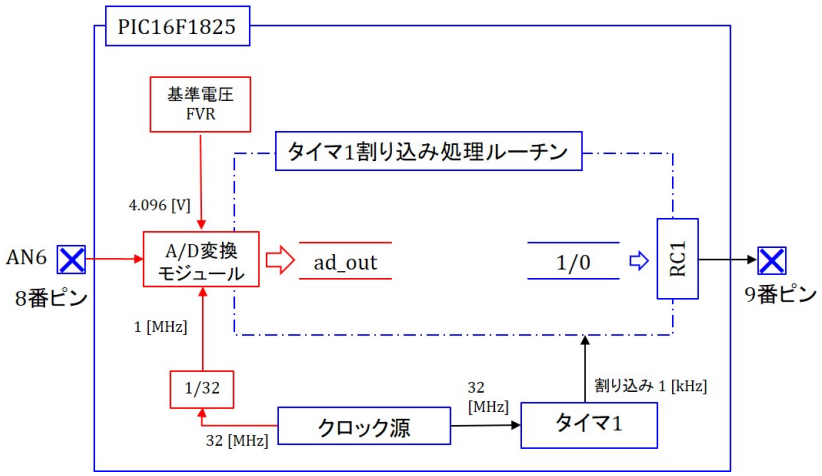


図 1.47: A/D 変換モジュールプログラムのブロック図

本項ではタイマ 1 による割り込みの度に A/D 変換を実行するプログラムを示す．図 1.24 の実験回路を用いる．この回路では可変抵抗 VR_1 より PIC16F1825 の 8 番ピンに可変のアナログ電圧を印加している．図 1.47 は作成したプログラムのブロック図である．図 1.45 のプログラムに対して，タイマ 1 による割り込み処理ルーチン内に新たに A/D 変換モジュールの起動とデータ読み出しに関する命令を追加している．タイマ 1 により 1 [ms] ごとに割り込みがかかり，割り込み処理ルーチンはその度に A/D 変換モジュールを起動する．A/D 変換モジュールは 8 番ピンからアナログ電圧信号を読み込み，ADRES レジスタに変換結果を格納する．割り込み処理ルーチンは A/D 変換終了を受けて，変数 ad_out に ADRES レジスタの内容を読み出す．A/D コンバータの動作クロックはシステムクロック (32 [MHz]) を 32 分周して 1 [MHz] を用いている．これは，データシートの ADC CLOCK PERIOD (TAD) VS. DEVICE OPERATING FREQUENCIES より推奨クロックが 1 [MHz] 以下であることによる．また，PIC16F1825 は基準電圧 (Fixed Volatage Reference: FVR) 源を内部に持っている．基準電圧は 1.024, 2.048, 4.096 [V] の 3 段階から選ぶことができる．このプログラムでは 4.096 [V] を用いる設定としている．

```
unsigned int ad_out;

static void __interrupt()          // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    RC1 = 1;                       // ポートCのRC1(9番ピン)に1を出力する. 割り込み発生のモニタリング用

    TMR1 = 0x8330;                 // タイマ1のカウントアップ値の設定.
                                   // 入力値を初期値としてカウントアップを行い, オーバーフローで割り込みを発生
    PIR1bits.TMR1IF = 0;           // タイマ1の割り込みフラグを0にして, 次のタイマ1による割り込みを受付け可とする.

    ADCON0bits.GO = 0b1;           // A/Dコンバータの変換開始(A/D CONTROL REGISTER 0)
    while(ADCON0bits.GO);          // 変換終了待ち
    ad_out = ADRES;                // 変換結果の読み出し

    RC1 = 0;                       // ポートCのRC1(9番ピン)に0を出力する. 割り込み発生のモニタリング用
}
```

図 1.48: A/D 変換モジュールのプログラム (AD_Conv_direct)

作成したプログラムを図 1.48, 図 1.49 に示す．図には図 1.31〜1.33 のプログラムに対して新たに追加した命令を青色で示してある．各命令の意味はそれぞれのコメント文を参照されたい．A/D コンバータの変換結果を格納しておく変数 ad_out をグローバル変数として

$$\text{unsigned int} \qquad \text{ad\_out} \tag{1.26}$$

と宣言している．グローバル変数とする理由は，MPLAB X のデバッガを用いた場合，グローバル変数でないとその数値を読み出せないためである．

また，

$$\text{TRISC} = 0\text{x}04 = 0\text{b}0000\ \ 0100 \tag{1.27}$$

により，8 番ピン (RC2/AN6) を入力ポートに設定している．

```
void main(void)
{
    TRISA = 0x00;                   // ポートAを出力ポートに設定する.
    TRISC = 0x04;                   // ポートC2を入力ポートに設定する.

    //オシレータの設定
    OSCCONbits.IRCF = 0b1110;   // 内蔵オシレータからの各種分周クロックの中から8MHzを選択
    OSCCONbits.SCS =  0b00;     // クロックソースはコンフィギュレーション設定に従う.
                                // コンフィギュレーションでは8MHz+4逓倍PLL = 32MHzの設定

    // タイマ1の設定
    T1CONbits.TMR1CS = 0b01;    // タイマ1のクロックソースをシステムクロック(FOSC)に設定
    T1CONbits.T1CKPS = 0b00;    // タイマ1のクロック分周率を1:1に設定（分周しない）
    T1CONbits.TMR1ON = 0b1;     // タイマ1をオンとする.

    // タイマ1による割り込み設定
    PIE1bits.TMR1IE = 1;            // タイマ1による割り込み可とする.
    INTCONbits.PEIE = 1;            // タイマ1などの周辺モジュールによる割り込みを可とする.
    INTCONbits.GIE = 1;             // 全ての割り込みを可とする.

    // A/D変換モジュールの設定
    ADCON0bits.CHS = 0b00110;  // アナログチャネルをAN6(8番ピン)に設定
    ADCON0bits.ADON = 0b1;     // A/Dコンバータをオンとする.
    ADCON1bits.ADFM = 0b1;     // 変換結果を16ビットレジスタの下位10ビットに入れる.
    ADCON1bits.ADCS = 0b010;   // A/Dコンバータのクロックを FOSC/32に設定
    ADCON1bits.ADNREF = 0b0;   // 基準電圧の−側をVSSとする.
    ADCON1bits.ADPREF = 0b11; // 基準電圧の＋側をFVR(Fixed Voltage Ref.)とする.

    // A/Dコンバータの基準電圧(FVR)の設定(FIXED VOLTAGE REFERENCE CONTROL REGISTER)
    FVRCONbits.FVREN = 0b1;    // FVRをオンとする.
    FVRCONbits.ADFVR = 0b11;   // A/Dコンバータの基準電圧を4.096 [V]に設定

    for(;;)
        continue;                   // 無限ループ
}
```

図 1.49: A/D 変換モジュールのプログラム（つづき）(AD_Conv_direct)

図 1.50 は A/D 変換モジュールのブロック図 (データシート ADC BLOCK DIAGRAM) である．このモジュールの設定は主に ADCON0 (A/D CONTROL REGISTER 0), ADCON1(A/D CONTROL REGISTER 1) レジスタにより行う．図中の経路はメインルーチン内の設定を表している．なお，A/D コンバータは 10 ピットであり，変換結果を格納する ADRES レジスタは 16 ビットである．そこで，ADFM(A/D Result Format Select bit) により，ADRES レジスタの上位 10 ビットに格納するか，下位 10 ビットに格納するかを選択できる．図では ADFM = 1 として，A/D 変換結果を下位 10 ビットに格納している．また，ADPREF(A/D Positive Voltage Reference Configuraton bits) = 0b11 とすることで，A/D コンバータの基準電圧源として FVR を選定している．メインルーチン内では FVRCON(FIXED VOLTAGE REFERENCE CONTROL REGISTER) レジスタの設定も行っている．

タイマ 1 による割り込み処理ルーチン内では，

$$\text{ADCON0bits.GO} = \text{0b1} \tag{1.28}$$

により，A/D 変換を開始させている．データシートの ANALOG-TO-DIGITAL CONVER-

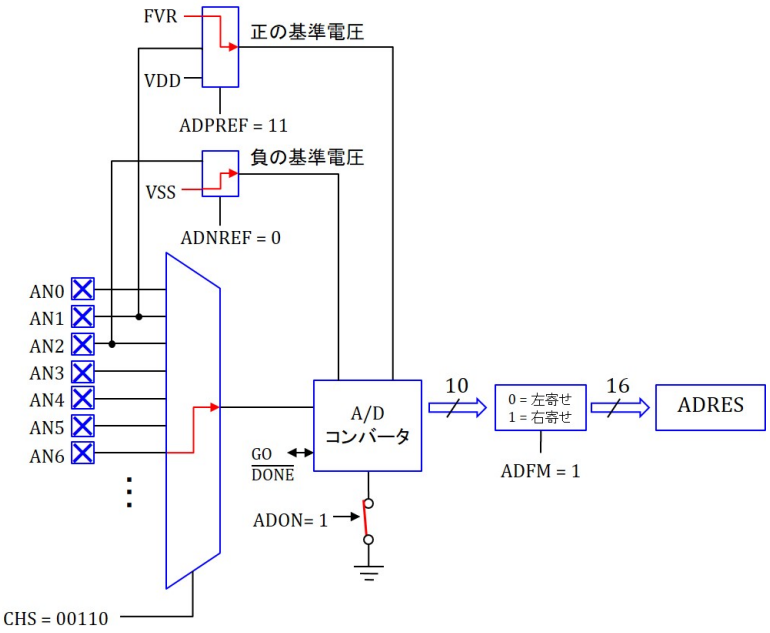


図 1.50: A/D 変換モジュールのブロック図

SION TAD CYCLES よりこの A/D 変換にはほぼ 12×TAD サイクルを要するとある．TAD は A/D コンバータのクロック周期であり，

$$\text{ADCON0bits.ADCS} = 0\text{b}010 \tag{1.29}$$

により

$$\text{TAD} = \frac{1}{\frac{\text{FOSC}}{32}} = \frac{1}{\frac{32[\text{MHz}]}{32}} = 1[\mu\text{s}] \tag{1.30}$$

である．よって，ADRES レジスタの値を読み出すのは A/D 変換を開始させてから 12 [μs] は待たなければならない．ADCON0bits.GO のビットは A/D 変換終了時に 0 にリセットされるようになっている．そこで，

$$\text{while(ADCON0bits.GO)} \tag{1.31}$$

により，A/D 変換終了まで何の処理もしないで待つ設定としている．A/D 変換終了後に ADRES レジスタの値を ad_out に読み出している．

図 1.51 は，図 1.8 に示すように実験回路に PICkit3 を接続し，MPLAB X のデバッガ (Debugger) で以上のプログラムを PIC16F1825 に書き込み，8 番ピンの入力電圧と ad_out の値を計測した結果を示す．1.2.5 項と同様にして，Variables → Enter new watch（右クリック） → New Watch → ad_out と進むと，ad_out の値を見ることができる．図 1.51 の横軸は 8 番ピンの入力電圧であり，縦軸は ad_out の値を 10 進数で表示した結果である．入力電圧がほぼ 4 [mV] 増えるごとに変換値は 1 ずつ増えていく様子が見て取れる．基準電圧 FVR = 4.096 [V] を 10 ビットで A/D 変換した場合

$$\frac{4.096[\text{V}]}{2^{10}} = 4[\text{mV}] \tag{1.32}$$

であり，実験結果はほぼ一致している．

図 1.52 は，図 1.48 のプログラムを用いた場合の，タイマ 1 による割り込み処理ルーチンの処理時間の計測結果である．約 14 [μs] を要している．

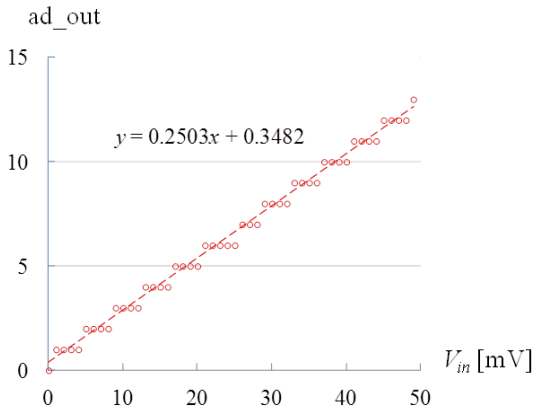


図 1.51: A/D 変換モジュールの変換実験結果

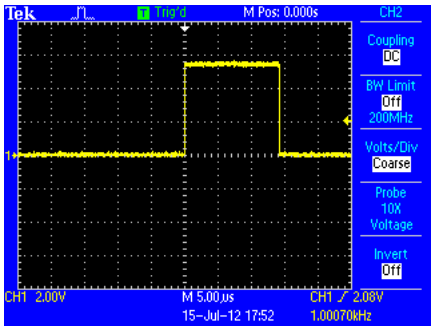


図 1.52: A/D コンバータ実行時の処理時間

1.3.7　関数とヘッダファイルの作成–A/D 変換モジュールのプログラム–

```
unsigned int  ad_out;

static void __interrupt()      // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    RC1 = 1;                   // ポートCのRC1に1を出力する. 割り込み発生のモニタリング用

    set_timer1_count_down_ini_num(0x7BF0);
                               // タイマ1のカウントダウン値の設定.
                               // 入力値を初期値としてカウントダウンを行い, 0で割り込みを発生することとなる.
    clear_interrupt_flag_of_timer1();
                               // タイマ1の割り込みフラグを0にして, 次のタイマ1による割り込みを受付け可とする.

    start_ad_conversion();
    ad_out = read_result_of_ad_conversion();

    RC1 = 0;                   // ポートCのRC1に0を出力する. 割り込み発生のモニタリング用
}
```

図 1.53: A/D 変換モジュールのプログラム（判読性を高めたプログラム）(AD_Conv)

図 1.53，図 1.54 は図 1.48，1.49 のプログラムの判読性を高めたプログラムである．図 1.35〜1.37 のプログラムに対して，新たに追加した命令を青色で示してある．

$$\text{start\_ad\_conversion();} \tag{1.33}$$

により，A/D 変換を開始する．

$$\text{ad\_out} = \text{read\_result\_of\_ad\_conversion();} \tag{1.34}$$

により変換結果を ad_out に読み出す．変換終了までの待ちの操作は関数の中で記述することとして，ここでは変換開始と，変換結果の読み出しが分かる記述としている．

A/D コンバータの設定は

$$\begin{aligned}\text{set\_AD\_Converter}(&\text{select\_AN6}, \text{AD\_ON}, \text{right\_justified}, \text{clock\_1\_32}\\ &, \text{neg\_ref\_VSS}, \text{pos\_ref\_FVR});\end{aligned} \tag{1.35}$$

による．また，FVR (Fixed Volatage Reference) の設定は

$$\text{set\_FVR}(\text{FVR\_ON}, \text{ad\_ref\_4\_096}); \tag{1.36}$$

により，基準電圧を 4.096 [V] に設定する．

式 (1.33)〜(1.36) の関数は set_ad_ converter.c のファイルに記述し，￥PIC16F1825_Files のフォルダに set_osc.c, set_timer1.c と一緒に入れてある．

```
void main(void)
{
    TRISA = 0x00;     // ポートAを出力ポートに設定する.
    TRISC = 0x04;     // ポートCを入力ポートに設定する.

    //オシレータの設定
    set_osc(Int_OSC_Freq_8MHz, SysClockSource_detmd_by_Config);
                      // 内蔵オシレータ8MHzでFOSC=32MHzと設定する.

    //  タイマー1の設定
    set_timer1(TMR1_clock_source_FOSC, T1Clock_PreScale_1_1, TMR1_ON);
                      // タイマ1のクロックソース，分周率を設定して，タイマ1をオンとする.
    set_interrupt_by_timer1();
                      // タイマ1による割り込みを可とする.

    // A/D変換モジュールの設定
    set_AD_Converter(select_AN6, AD_ON, right_justified, clock_1_32, neg_ref_VSS, pos_ref_FVR);

    // FVRの設定
    set_FVR(FVR_ON, ad_ref_4_096);

    for(;;)
        continue;          // 無限ループ
}
```

図 1.54:　A/D 変換モジュールのプログラム（判読性を高めたプログラム，つづき）(AD_Conv)

```
#include <xc.h>
#include "../PIC16F1825_Files/pic16f1825_s.h"

// A/D変換モジュールの設定
void set_AD_Converter(unsigned int a, unsigned int b, unsigned int c, unsigned int d, unsigned int e, unsigned int f)
{
    ADCON0bits.CHS = a;         // アナログチャネル設定
    ADCON0bits.ADON = b;        // A/Dコンバータをオン/オフ設定
    ADCON1bits.ADFM = c;        // 変換結果を16ビットレジスタの上位／下位10ビットに入れる.
    ADCON1bits.ADCS = d;        // A/Dコンバータのクロック設定
    ADCON1bits.ADNREF = e;      // 基準電圧の−側の設定
    ADCON1bits.ADPREF = f;      // 基準電圧の＋側の設定
}

void set_FVR(unsigned int a, unsigned int b)
{
    // A/Dコンバータの基準電圧(FVR: Fixed Voltage Reference)の設定
    FVRCONbits.FVREN = a;       // FVRをオン／オフ設定
    FVRCONbits.ADFVR = b;       // A/Dコンバータの基準電圧を設定
}

void start_ad_conversion()
{
    ADCON0bits.GO = 0b1;        // A/Dコンバータの変換開始
    while(ADCON0bits.GO);       // 変換終了待ち
}

unsigned int read_result_of_ad_conversion()
{
    unsigned int a;

    a = ADRES;                  // 変換結果の読み出し
    return(a);
}
```

図 1.55: A/D 変換モジュールの設定関数 (set_ad_conver.c)(AD_Conv)

図 1.55 は set_ad_converter.c のファイル内の関数を示す．図 1.48，1.49 の青字の箇所をそれぞれ移植してある．

Pic16f1825_s.h

```
// ADCON0(A/D Control Register 0)の用語の設定
    // CHS(Analog Channel Select bits)
    #define select_AN0    0b00000     //AN0を選定
    #define select_AN1    0b00001     //AN1
    #define select_AN2    0b00010     //AN2
    #define select_AN3    0b00011     //AN3
    #define select_AN4    0b00100     //AN4
    #define select_AN5    0b00101     //AN5
    #define select_AN6    0b00110     //AN6
    #define select_AN7    0b00111     //AN7
    // ADON(A/D Conversion Enable bit)
    #define AD_ON         0b1         //A/Dコンバータをオンとする
    #define AD_OFF        0b0         //                  オフ

// ADCON1(A/D Control Register 1)の用語の設定
    // ADFM(A/D Result Format Select bit)
    #define right_justified  0b1      //変換結果を16ビットレジスタの右よせで格納する.
    #define left_justified   0b0      //                              左よせ
    // ADCS(A/C Conversion Clock Select bits)
    #define clock_1_2     0b000       // A/Dコンバータのクロックを FOSC/2とする.
    #define clock_1_8     0b001       //                          FOSC/8
    #define clock_1_32    0b010       //                          FOSC/32
    #define clock_FRC     0b011       // RCオシレータのクロックを使う.
    #define clock_1_4     0b100       //                          FOSC/4
    #define clock_1_16    0b101       //                          FOSC/16
    #define clock_1_64    0b110       //                          FOSC/64
    #define clock_FRC1    0b111       // RCオシレータのクロックを使う.
    // ADNREF(A/D Negative Voltage Reference Configuration bit)
    #define neg_ref_VSS           0b0 // A/Dコンバータの基準電圧の−側をVSSとする.
    #define neg_ref_exVREF_Neg  0b1 //                              を外部VREF-ピンとする
    // ADPREF(A/D Positive Voltage Reference Configuration bit)
    #define pos_ref_VDD           0b00// A/Dコンバータの基準電圧の+側をVDDとする.
    #define pos_ref_exVREF_Pos 0b10 //                              を外部VREF+ピンとする
    #define pos_ref_FVR  0b11         // 基準電圧にFVR(Fixed Voltage Reference)の電圧を使用

// FVRCON(Fixed Voltage Reference Control Register)の設定
    // FVREN(FVR Enable bit)
    #define FVR_OFF       0b0         // FVR オフ
    #define FVR_ON        0b1         // FVR オン
    // ADFVR(FVR setting for AD Converter)
    #define ad_ref_OFF     0b00       // FVRによる基準電圧をオフとする.
    #define ad_ref_1_024 0b01         // FVRによる基準電圧を1.024 [V]に設定
    #define ad_ref_2_048 0b10         //                          2.048 [V]
    #define ad_ref_4_096 0b11         //                          4.096 [V]

//関数の宣言
void set_AD_Converter(unsigned int a, unsigned int b, unsigned int c, unsigned int d, unsigned int e,
unsigned int f);
void set_FVR(unsigned int a, unsigned int b);
void start_ad_conversion(void);
unsigned int read_result_of_ad_conversion(void);
void start_ad_conversion_ch_select(unsigned int a);
```

図 1.56: A/D 変換モジュールのヘッダファイル (AD_Conv)

また，式 (1.35)，(1.36) 内の引数と数値の対応関係は pic16f1825_s.h のヘッダファイルに追記してある．図 1.56 はヘッダファイルの追加部分を示す．レジスタ設定の数値に意味の分かりやすい表現を対応付けてある．

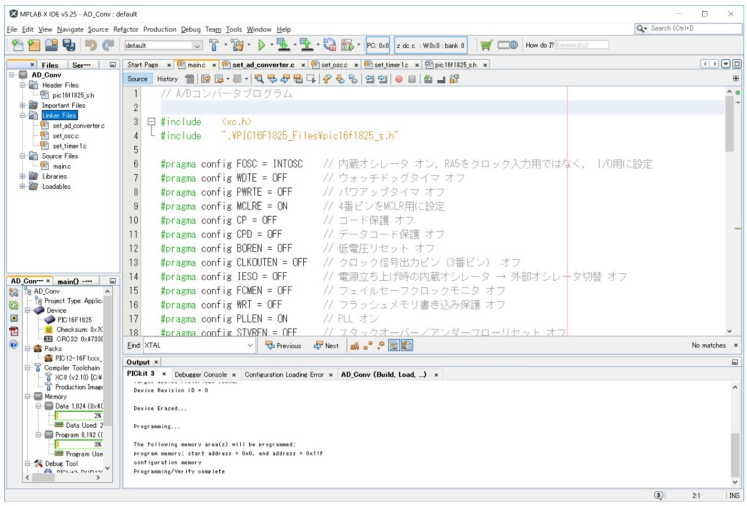


図 1.57: MPLAB 画面 _ 関数定義ファイルの読み込み（A/D 変換モジュールの設定関数）

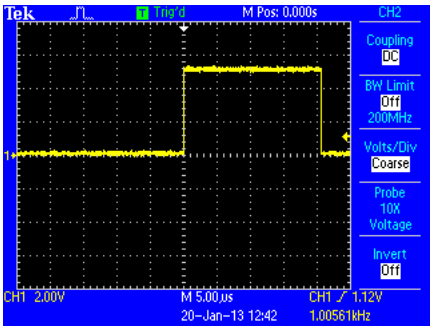


図 1.58: A/D コンバータ実行時の処理時間（判読性を高めたプログラムの場合）

図 1.57 は set_ad_converter.c を set_osc.c, set_timer1.c と一緒に Linker Files に追加した様子を示す..

図 1.58 は，図 1.53，1.54 のプログラムを用いた場合の，タイマ 1 による割り込み処理ルーチンの処理時間の計測結果である．約 20.5 [μs] を要している．ルーチン内の処理時間が延びるにつれて，判読性を高めることによる処理時間の増大率は相対的に小さくなっている．

1.3.8　PWM モジュール

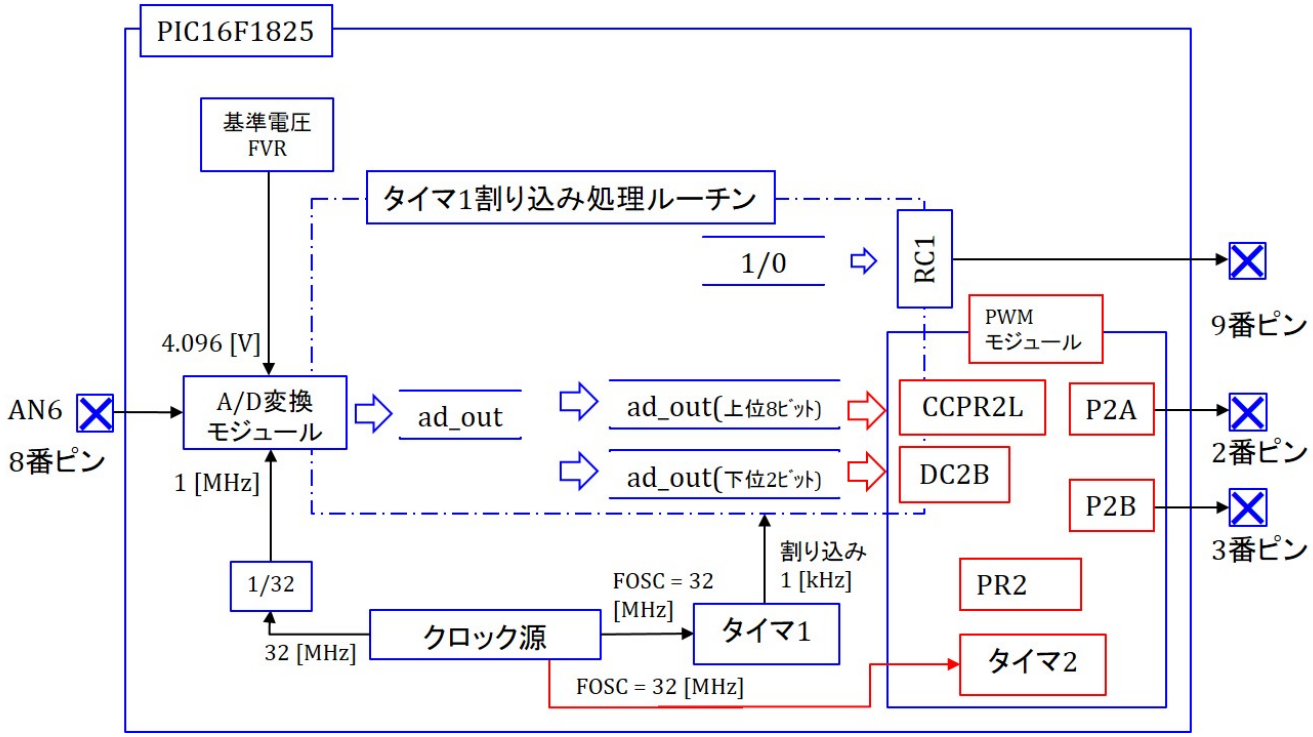


図 1.59: PWM モジュールプログラムのブロック図

図 1.59 は PWM モジュールプログラムのブロック図を示す．8 番ピンに入力する電圧で 2, 3 番ピンに出力される PWM 波形を制御する．PWM モジュールはカウンタを必要とする．このカウンタとして，タイマ 2, 4, 6 のいずれをも用いることができる．本項ではタイマ 2 を用いる．データシート (TIMER2/4/6 MODULES) によると，タイマ 2 は 8 ビットカウンタであり，システムクロックを 1/4 に分周した FOSC/4 = 8 [MHz] により駆動されると記されている．しかし，PWM モジュールではタイマ 2 は 10 ビットカウンタとして機能し，システムクロック FOSC = 32 [MHz] により駆動されると解釈した方が，PWM モジュールの動作を理解しやすい．

図 1.60 は PWM モジュールのブロック図 (SIMPLIFIED PWM BLOCK DIAGRAM) である．タイマ 2(TMR2) にシステムクロック FOSC が入力され，タイマ 2 は常に入力クロックをカウントアップする．このタイマ 2 の上位 8 ビットと PR2 (Timer2 Module Period Register) レジスタ (8 ビットレジスタ) の値が図中下側の Comparator（比較器）により比較され，両者が一致すると比較器は，タイマ 2 の値を 0 に初期化し，R-S フリップフロップをセット ($Q = 1, \overline{Q} = 0$) する．また，CCPR2H (Capture/Compare/PWM Register 2 High Byte) レジスタ (10 ビットレジスタ）の上位 8 ビットに，CCPR2L レジスタ (8 ビットレジスタ）の値を読み込み，下位 2 ビットに CCP2CON (CCP2 Control Register) レジスタの DC2B (2 ビット) の値を読み込む．そして，タイマ 2 は再び 0 からカウントアップを始める．図中上側の比較器により TMR2 レジスタの値と CCPR2H レジスタの値が比較

され，両者が一致したときに比較器は R-S フリップフロップをリセット ($Q = 0, \overline{Q} = 1$) する．

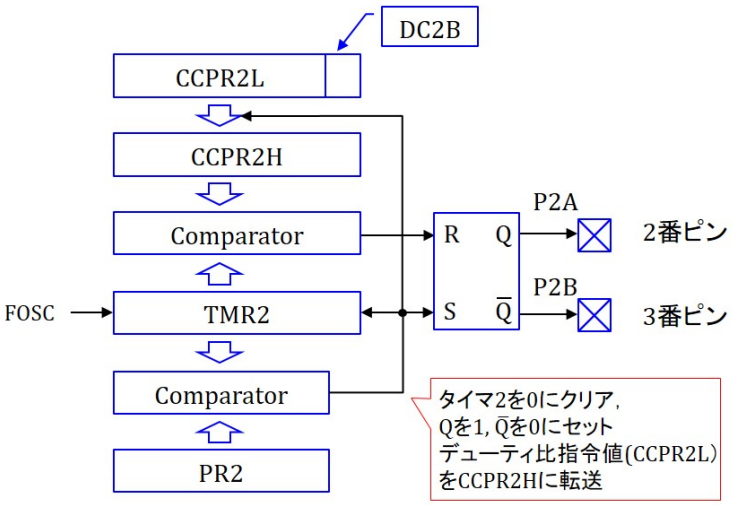


図 1.60: PWM モジュールのブロック図

以上の動作のイメージを図 1.61 に示す．TMR2 レジスタの値が 0 に初期化されたとき，PWM モジュール出力である P2A $= Q = 1$, P2B $= \overline{Q} = 0$ とセットされる．TMR2 の値は FOSC によりカウントアップされるので，時間に比例して増加する．

$$\text{TMR2} == \text{CCPR2L} \times 4 + \text{DC2B} \tag{1.37}$$

となったとき，P2A $= Q = 0$, P2B $= \overline{Q} = 1$ にリセットされる．

$$\text{TMR2} == \text{PR2} \times 4 \tag{1.38}$$

となったとき，TMR2 $= 0$，P2A $= 1$, P2B $= 0$ とセットされる．そして，TMR2 のカウントアップが再開される．以上により，P2A, P2B の値の平均値は CCPR2L $\times 4 +$ DC2B の値に比例する．このように P2A, P2B の幅（パルス幅と呼ぶ）を制御する手法を PWM(Pulse Width Modulation : パルス幅変調) 制御法と呼ぶ．TMR2 の三角波の繰り返し周期 T_{PWM} を PWM 周期と呼ぶ．この逆数を PWM 周波数 $f_{PWM} = 1/T_{PWM}$ と呼ぶ．また，P2A $=$ 1 の期間を T_{ON} とすると $\delta = T_{ON}/T_{PWM}$ をデューティ比と呼ぶ

FOSC $= 32$ [MHz] のとき，PR2 $=$ 0xFF とすると，PWM 周期 T_{PWM} は

$$\begin{aligned} T_{PWM} &= \frac{1}{\frac{\text{FOSC}}{256 \times 4}} \\ &= 32[\mu\text{s}] \end{aligned} \tag{1.39}$$

であり，PWM 周波数は 31.25 [kHz] である．また，デューティ比 δ の分解能は $1/2^{10} = 1/1024$ である．

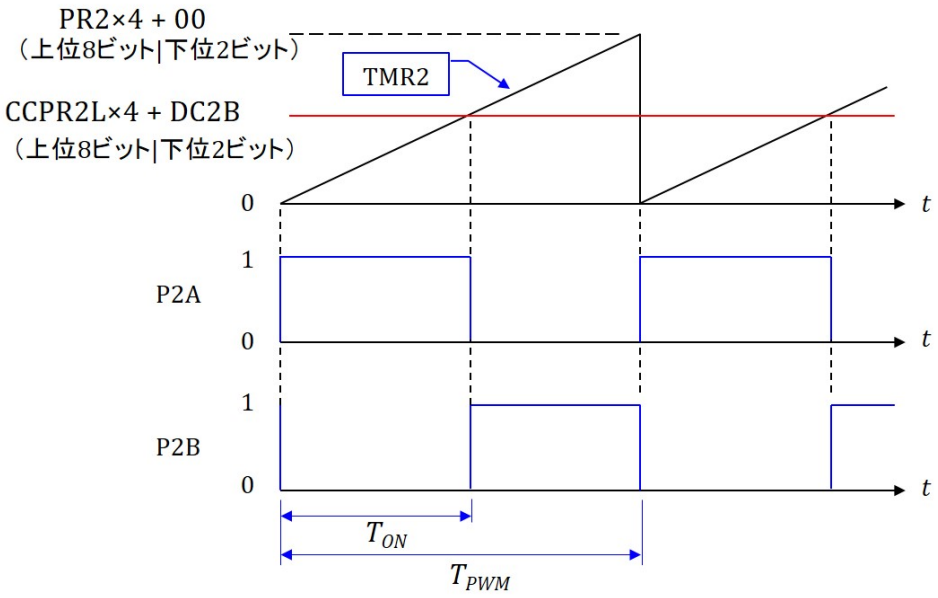


図 1.61: PWM モジュールの動作

```
#define PWM_period  0xFF
unsigned int ad_out;

static void __interrupt()      // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    RC1 = 1;                   // ポートCのRC1に1を出力する. 割り込み発生のモニタリング用

    TMR1 = 0x8330;             // タイマ1のカウントアップ値の設定
                               // 入力値を初期値としてカウントアップを行い, オーバーフローで割り込みを発生
    PIR1bits.TMR1IF = 0; // タイマ1の割り込みフラグを0にして, 次のタイマ1による割り込みを受付け可とする.


    ADCON0bits.GO = 0b1;       // A/Dコンバータの変換開始
    while(ADCON0bits.GO);      // 変換終了待ち
    ad_out = ADRES;            // 変換結果の読み出し

    CCP2CONbits.DC2B = ad_out; // DC2B にA/D変換結果の下位2ビットを格納する.
    CCPR2L = ad_out >>2;       // CCPR2L にA/D変換結果の上位8ビットを格納する.

    RC1 = 0;                   // ポートCのRC1に0を出力する. 割り込み発生のモニタリング用
}
```

図 1.62: PWM モジュールのプログラム (PWM_direct)

図 1.62，1.63 は PWM モジュールのプログラムを示す．図 1.48，1.49 の A/D 変換モジュールのプログラムに対して，新たに追加した命令を青色で示してある．タイマ 1 による割り込み処理ルーチン内では，PWM 制御のデューティ比 δ を A/D 変換結果 ad_out により設定している．ad_out には図 1.50 で述べたように下位 10 ビットに A/D 変換結果が格納されている．また，図 1.60 で述べたように，CCPR2L レジスタは 8 ビット，DC2B は 2 ビットである．

$$\text{CCP2CONbits.DC2B} = \text{ad\_out}; \tag{1.40}$$

により，ad_out の下位 2 ビットを DC2B に格納している．また，

$$\text{CCPR2L} = \text{ad\_out} >> 2; \tag{1.41}$$

により，ad_out を右に 2 ビットシフトして，その結果を CCPR2L レジスタに格納している．右へ 2 ビットシフトさせることで，2〜9 ビット目の値を CCP2RL に格納できる．

```
void main(void)
{
    TRISA = 0x00;                   // ポートAを出力ポートに設定する.
    TRISC = 0x04;                   // ポートC2を入力ポートに設定する.
    //オシレータの設定
    OSCCONbits.IRCF = 0b1110;   // 内蔵オシレータからの各種分周クロックの中から8MHzを選択
    OSCCONbits.SCS =  0b00;     // クロックソースはコンフィギュレーション設定に従う.
                                // コンフィギュレーションでは8MHz+4逓倍PLL = 32MHzの設定
    // タイマ1の設定
    T1CONbits.TMR1CS = 0b01;    // タイマ1のクロックソースをシステムクロック(FOSC)に設定
    T1CONbits.T1CKPS = 0b00;    // タイマ1のクロック分周率を1:1に設定（分周しない）
    T1CONbits.TMR1ON = 0b1;     // タイマ1をオンとする.

    // タイマ1による割り込み設定
    PIE1bits.TMR1IE = 1;            // タイマ1による割り込み可とする.
    INTCONbits.PEIE = 1;            // タイマ1などの周辺モジュールによる割り込みを可とする.
    INTCONbits.GIE = 1;             // 全ての割り込みを可とする. (p.92).

    // A/D変換モジュールの設定
    ADCON0bits.CHS = 0b00110;   // アナログチャネルをAN6(8番ピン)に設定
    ADCON0bits.ADON = 0b1;      // A/Dコンバータをオンとする.
    ADCON1bits.ADFM = 0b1;      // 変換結果を16ビットレジスタの下位10ビットに入れる.
    ADCON1bits.ADCS = 0b010;    // A/Dコンバータのクロックを FOSC/32に設定（推奨値)
    ADCON1bits.ADNREF = 0b0;    // 基準電圧の−側をVSSとする.
    ADCON1bits.ADPREF = 0b11;   // 基準電圧の＋側をFVR(Fixed Voltage Ref.)とする.

    // A/Dコンバータの基準電圧(FVR: Fixed Voltage Reference)の設定
    FVRCONbits.FVREN = 0b1;     // FVRをオンとする.
    FVRCONbits.ADFVR = 0b11;    // A/Dコンバータの基準電圧を4.096 [V]に設定

    //PWMモジュールの設定
    CCP2CONbits.P2M = 0b10;         // Half Bridge, P2A, P2Bを利用.　P2C, P2Dはポートピン
    CCP2CONbits.CCP2M = 0b1100;     // PWMモード　P2A, P2B を正論理とする.

    // PWM出力ピンの割り当て
    APFCON1bits.P2BSEL = 0b1;   // P2BをRA4(3番ピン)に割り当てる.
    APFCON1bits.CCP2SEL = 0b1; // P2AをRA5(2番ピン)に割り当てる.

    // PWM用のタイマの割り当て
    CCPTMRSbits.C2TSEL = 0b00; // PWM用のタイマとしてタイマ2を割り当てる.

    // タイマ2の設定
    T2CONbits.T2OUTPS = 0b000; // タイマ2出力の分周率を1:1に設定（分周しない）
    T2CONbits.TMR2ON = 0b1;     // タイマ2をオンとする.
    T2CONbits.T2CKPS = 0b00;    // タイマ2の入力クロック(FOSC/4)を1:1に分周する.
    PR2 = PWM_period;           // PWM周波数設定　FOSC/4/256 = 32[MHz]/4/256 = 31.25[kHz]

    for(;;)
        continue;       // 無限ループ
}
```

図 1.63: PWM モジュールのプログラム（つづき）(PWM_direct)

図 1.63 では新たに PWM モジュールの設定，PWM モジュール出力ピンの割り当て，PWM モジュール用のタイマの割り当て，そして，タイマ 2 の設定を行っている．PWM モジュールの設定は，CCPxCON レジスタ (CCPx CONTROL REGISTER) により行う．CCP は Capture, Compare, PWM のイニシャルである．P2A と P2B を利用するためには CCP2CON レジスタの設定を必要とする．

$$\text{CCP2CONbits.P2M} = 0\text{b10}; \tag{1.42}$$

により，P2M(Enhanced PWM Output Configuration bits) に 0b10 を書き込むことで，P2A, P2B を利用する設定としている．また，

$$\text{CCP2CONbits.CCP2M} = 0\text{b1100}; \tag{1.43}$$

により，P2A, P2B を正論理 (active high) に設定している．これを

$$\text{CCP2CONbits.CCP2M} = \text{0b1111;} \tag{1.44}$$

として，P2A, P2B ともに負論理 (active low) に設定すると，図 1.61 において P2A, P2B の値は 1, 0 が反転する．

P2A, P2B を出力するピンはそれぞれ 2 つのピンの中から選択できる．P2A は 2 番ピンか 7 番ピン, P2B は 3 番ピンか 8 番ピンのいずれかを選択できる（図 1.26 参照）．この選択は APFCON1(ALTERNATE PIN FUNCTION CONTROL REGISTER 1) レジスタにより行う．

$$\text{APFCON1bits.CCP2SEL} = \text{0b1;} \tag{1.45}$$

により，P2A を RA5(2 番ピン) に割り当て，

$$\text{APFCON1bits.P2BSEL} = \text{0b1;} \tag{1.46}$$

により，P2B を RA4(3 番ピン) に割り当てている．

PWM モジュール用のタイマにはタイマ 2, 4, 6 のいずれかを使うことができる．この選択は CCPTMRS (PWM TIMER SELECTION CONTROL REGISTER 0) レジスタにより行う．

$$\text{CCPTMRSbits.C2TSEL} = \text{0b00;} \tag{1.47}$$

により，タイマ 2 を選択している．

追加した命令の最後はタイマ 2 の設定である．タイマ 1 の設定との大きな違いは，タイマ 1 ではクロック源を各種選択できるのに対して，タイマ 2 ではクロック源はシステムクロック FOSC に限定されている．選択できるのは分周比だけである．PR2(Timer2 Module Period Register) を

$$\text{PR2} = \text{0xFF} \tag{1.48}$$

とすることで，図 1.60 で説明したように，TMR2 レジスタ (10 ビット）の上位 8 ビットが PR2 レジスタの値と比較される．データシートの PWM PERIOD より，PWM 周波数 f_{PWM} は

$$f_{PWM} = \text{FOSC}/4/(255+1) = 32[\text{MHz}]/4/256 = 31.25[\text{kHz}] \tag{1.49}$$

と求められる．これは話の順序が逆で，32.15 [kHz] の f_{PWM} を得るためには，

$$\text{PR2} = 31.25[\text{kHz}]/(\text{FOSC}/4) - 1 = 31.25[\text{kHz}]/32[\text{MHz}]/4 - 1 = 255 \tag{1.50}$$

と設定すればよい．

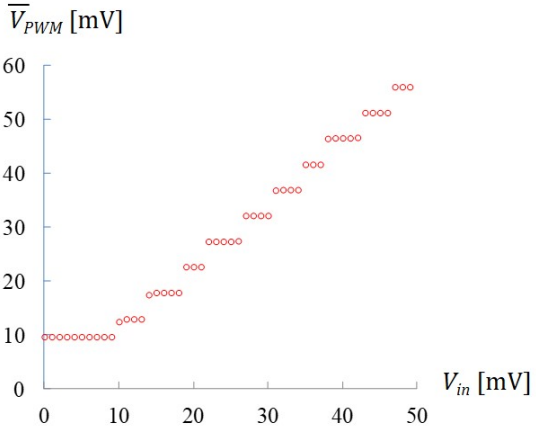
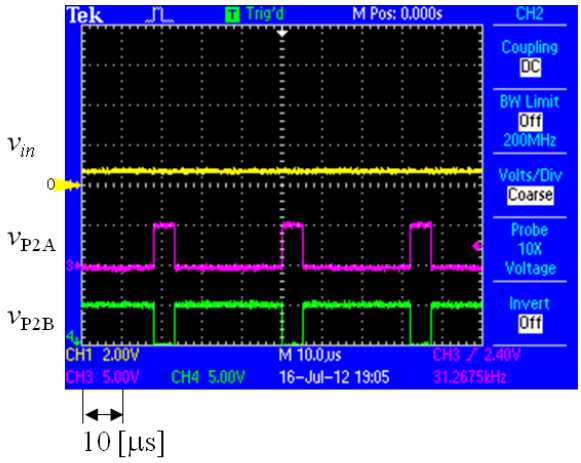


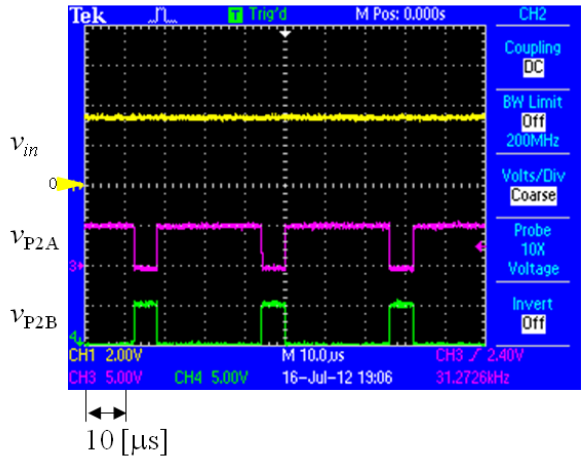
図 1.64: PWM モジュールの入出力電圧の計測結果

図 1.64 は，図 1.8 に示すように実験回路に PICkit3 を接続し，MPLAB X により以上のプログラムを PIC16F1825 に書き込み，8 番ピンの入力電圧と 2 番ピンの出力電圧を計測した結果を示す．図の横軸は 8 番ピンの入力電圧 V_{in} であり，縦軸は 2 番ピンの出力電圧の平均値 $\overline{V}_{PWM}$ である．V_{in} がほぼ 4 [mV] 増えると $\overline{V}_{PWM}$ は階段状に増加していく様子が見て取れる．$V_{in} \leq$ 数 [mV] では $\overline{V}_{PWM}$ が変化しない様子も見て取れる．

図 1.65 は，8 番ピンの入力電圧 v_{in} と 2, 3 番ピンの PWM モジュール出力電圧 $v_{\mathrm{P2A}}, v_{\mathrm{P2B}}$ の実験波形例を示す．同図 (a) は入力電圧 v_{in} が低い場合であり，(b) は v_{in} が高い場合である．入力電圧の高低に応じて，$v_{\mathrm{P2A}}, v_{\mathrm{P2B}}$ のパルス幅が変化している様子が分かる．また，PWM 周波数 $f_{PWM} \approx 31.27$ [kHz] であることが分かる．また，図 1.66 は A/D 変換モジュールと PWM モジュール実行時のタイマ 1 による割り込み処理ルーチンの処理時間を示す．図の横軸は 10 [μs/div] であるので，処理時間は約 18 [μs] であった．



(a) v_{in} 低



(b) v_{in} 高

図 1.65: PWM モジュール実行時の入出力波形

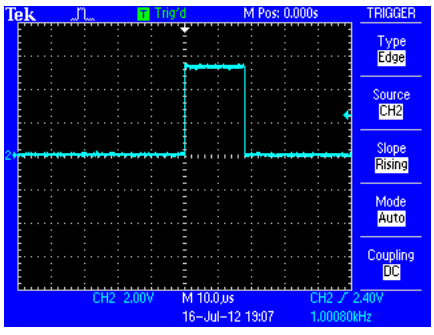


図 1.66: PWM モジュール実行時のタイマ 1 による割り込み処理ルーチンの処理時間

1.3.9　関数とヘッダファイルの作成–PWM モジュールのプログラム–

```
#define PWM_period   0xFF
unsigned int  ad_out;

static void __interrupt()          // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    RC1 = 1;                       // ポートCのRC1に1を出力する．割り込み発生のモニタリング用

    set_timer1_count_down_ini_num(0x7BF0);
                                   // タイマ1のカウントダウン値の設定．
                                   // 入力値を初期値としてカウントダウンを行い，0で割り込みを発生することとなる．
    clear_interrupt_flag_of_timer1();
                                   // タイマ1の割り込みフラグを0にして，次のタイマ1による割り込みを受付け可とする．

    start_ad_conversion();
    ad_out = read_result_of_ad_conversion();

    set_PWM_duty_cycle(ad_out);  // PWMデューティ比の設定

    RC1 = 0;                       // ポートCのRC1に0を出力する．割り込み発生のモニタリング用
}
```

図 1.67: PWM モジュールのプログラム（判読性を高めたプログラム）(PWM)

図 1.67，1.68 は，図 1.62，1.63 のプログラムの判読性を高めたプログラムである．図 1.53，1.54 のプログラムに対して，新たに追加した命令を青色で示してある．

$$\text{set\_PWM\_duty\_cycle}(\text{ad\_out}); \tag{1.51}$$

は，PWM 制御のデューティ比の設定を行う関数である．10 ビットの A/D 変換結果 ad_out を上位 8 ビットと下位 2 ビットに振り分ける操作を関数内に移動させ，関数名には ad_out を基に PWM 制御のデューティ比の設定を行うという最も重要な情報のみを記している．

メイン関数内では

$$\begin{aligned}\text{set\_PWM}(&\text{Half\_Bridge\_with\_P2A\_P2B}, \text{P2A\_B\_C\_D\_active\_high},\\ &\text{P2B\_to\_RA4}, \text{P2A\_to\_RA5}, \text{based\_on\_timer2}));\end{aligned} \tag{1.52}$$

により，各引数からこの set_PWM 関数が何をどう設定するものであるかを分かりやすくしている．

```
void main(void)
{
    TRISA = 0x00;               // ポートAを出力ポートに設定する.
    TRISC = 0x04;               // ポートCを入力ポートに設定する.


    //オシレータの設定
    set_osc(Int_OSC_Freq_8MHz, SysClockSource_detmd_by_Config);

    //  タイマー1の設定
    set_timer1(TMR1_clock_source_FOSC, T1Clock_PreScale_1_1, TMR1_ON);
    set_interrupt_by_timer1();

    // A/D変換モジュールの設定
    set_AD_Converter(select_AN6, AD_ON, right_justified, clock_1_32, neg_ref_VSS, pos_ref_FVR);

    // FVRの設定
    set_FVR(FVR_ON, ad_ref_4_096);

    // PWMモジュールの設定
    set_PWM(Half_Bridge_with_P2A_P2B, P2A_B_C_D_active_high, P2B_to_RA4, P2A_to_RA5, based_on_timer2);

    // タイマ2の設定
    set_timer2(set_Postscaler_1_1, timer2_on, set_Prescaler_1_1, PWM_period);

    for(;;)
        continue;               // 無限ループ
}
```

図 1.68: PWM モジュールのプログラム（判読性を高めたプログラム，つづき）(PWM)

set_pwm.c

```
#include <xc.h>
#include "..¥PIC16F1825_Files¥pic16f1825_s.h"

// CCP2CON(CCP2 Control Register)の設定
// APFCON1(Alternate Pin Function Control Register1)の設定
// CCPTMRS(PWM Timer Selection Control Register0)の設定
void set_PWM(unsigned int a, unsigned int b, unsigned int c, unsigned int d, unsigned int e)
{
    //PWMモジュールの設定
    CCP2CONbits.P2M = a;        // Half Bridge, P2A, P2Bを利用.   P2C, P2Dはポートピン
    CCP2CONbits.CCP2M = b;      // PWMモード  P2A, P2B active high

    // PWM出力ピンの割り当て
    APFCON1bits.P2BSEL = c;     // P2BをRA4(3番ピン)に割り当てる
    APFCON1bits.CCP2SEL = d;    // P2AをRA5(2番ピン)に割り当てる.

    // PWM用のタイマの割り当て
    CCPTMRSbits.C2TSEL = e;     // PWM用のタイマとしてタイマ2を割り当てる.
}

// PWMのデューティ比の設定
void set_PWM_duty_cycle(unsigned long int a)
{
    CCP2CONbits.DC2B = a;       // PWMデューティ比の下位2ビット指定
    CCPR2L = a >> 2;            // PWMデューティ比の上位8ビット指定
}
```

図 1.69: PWM モジュール設定関数 (set_pwm.c)(PWM)

図 1.69 は set_pwm.c ファイル内の関数を示す．式 (1.51)，(1.52) の関数を set_pwm.c のファイル内に記述してある．これら関数は図 1.63 のプログラムの main ルーチン内で行っていたレジスタ設定を移植したものである．set_pwm.c は￥PIC16F1825_Files のフォルダに入れてある．

図 1.68 において新しく追加した命令の最後は

$$\text{set\_timer2}(\text{set\_Postscaler1\_1\_}, \text{timer2\_on}, \text{set\_Prescaler1\_1\_}, \text{PWM\_period}); \qquad (1.53)$$

である．この関数は分周率，PWM 周期の設定を行っている．

set_timer2.c

```
#include <xc.h>
#include "..¥PIC16F1825_Files¥pic16f1825_s.h"

// T2CON (Timer2 Control Register)の設定
void set_timer2(unsigned int a, unsigned int b, unsigned int c, unsigned int d)
{
    // タイマ2の設定
    T2CONbits.T2OUTPS = a;      // タイマ2出力の分周率を設定
    T2CONbits.TMR2ON = b;       // タイマ2をオンとする.
    T2CONbits.T2CKPS = c;       // タイマ2の入力クロック(FOSC/4)を分周する.
    PR2 = d;                    // PWM周波数設定  FOSC/4/d
}
```

図 1.70: タイマ 2 設定関数 (set_timer2.c)(PWM)

図 1.70 は set_timer2.c のファイル内に記述した set_timer2 関数である．図 1.63 のプログラムの main ルーチン内で行っていた T2CON レジスタ設定をこの関数に移植した．

図 1.71 は PWM モジュール設定用ヘッダファイルである．式 (1.52) で用いている引数と CCP2CON, APFCON1, CCPTMRS レジスタの各設定値との対応関係を示してある．図 1.72 はタイマ 2 設定用ファッダファイルである．式 (1.53) で用いている引数と T2CON レジスタの設定値との対応関係を示してある．これらの対応関係は pic16f1825_s.h のヘッダファイルに追記してある．

図 1.73 は，図 1.67 のプログラムを用いた場合の，タイマ 1 による割り込み処理ルーチンの処理時間の計測結果である．約 26 [μs] を要している．

```
// CCP2CON(CCP2 Control Register)の設定
    // P2M (Enhanced PWM Output Configuration bits)
    // CCP2M = 11xxのとき
    #define PWM_with_P2A_only       0b00    // P2AのみPWM出力，P2B, P2C, P2Dはポートピン
    #define Full_Bridge_with_P2D    0b01    // フルブリッジPWM, P2Aは常時オン，P2DでPWM制御.
    #define Half_Bridge_with_P2A_P2B 0b10   // ハーフブリッジPWM, P2A, P2BにPWM出力，P2C, P2Dはポートピン
    #define Full_Bridge_with_P2B    0b11    // フルブリッジPWM, P2BでPWM制御，P2Cが常時オン
    // CCP2M (Enhanced CCP2 Mode Select bits)
    #define P2A_B_C_D_active_high          0b1100  // PWMモード，P2A, B, C, D が正論理
    #define P2AC_act_high_P2BD_act_low     0b1101  //          P2A, Cが正論理，P2BDが負論理
    #define P2AC_act_low_P2BD_act_high     0b1110  //          P2A, Cが負論理，P2BDが正論理
    #define P2AC_act_low_P2BD_act_low      0b1111  // PWMモード，P2A, B, C, D が負論理

// APFCON1(Alternate Pin Function Control Register1)の設定
    // P2BSEL (P2B pin selection bit)
    #define P2B_to_RC2      0b0 // P2BをRC2に割り当てる
    #define P2B_to_RA4      0b1 // P2BをRA4に割り当てる
    // CCP2SEL (P2A pin selection bit)
    #define P2A_to_RC3      0b0 // P2AをRC3に割り当てる
    #define P2A_to_RA5      0b1 // P2BARA5に割り当てる

// CCPTMRS(PWM Timer Selection Control Register0)の設定

    // C2TSEL (CCP2 Timer Selection bits)
    #define based_on_timer2     0b00    // タイマ2によりPWM周期を決定
    #define based_on_timer4     0b01    // タイマ4によりPWM周期を決定
    #define based_on_timer6     0b10    // タイマ6によりPWM周期を決定
```

図 1.71: PWM モジュール設定用ヘッダファイル (PWM)

```
// T2CON (Timer2 Control Register)の設定
    // T2OUTPS (Timer2 Output Postscaler Select bits)
    #define set_Postscaler_1_1          0b0000  // 1:1にタイマ2出力を分周
    #define set_Postscaler_1_2          0b0001  // 1:2にタイマ2出力を分周
    #define set_Postscaler_1_3          0b0010  // 1:3にタイマ2出力を分周
    #define set_Postscaler_1_4          0b0011  // 1:4にタイマ2出力を分周
    #define set_Postscaler_1_5          0b0100  // 1:5にタイマ2出力を分周
    #define set_Postscaler_1_6          0b0101  // 1:6にタイマ2出力を分周
    #define set_Postscaler_1_7          0b0110  // 1:7にタイマ2出力を分周
    #define set_Postscaler_1_8          0b0111  // 1:8にタイマ2出力を分周
    #define set_Postscaler_1_9          0b1000  // 1:9にタイマ2出力を分周
    #define set_Postscaler_1_10         0b1001  // 1:10にタイマ2出力を分周
    #define set_Postscaler_1_11         0b1010  // 1:11にタイマ2出力を分周
    #define set_Postscaler_1_12         0b1011  // 1:12にタイマ2出力を分周
    #define set_Postscaler_1_13         0b1100  // 1:13にタイマ2出力を分周
    #define set_Postscaler_1_14         0b1101  // 1:14にタイマ2出力を分周
    #define set_Postscaler_1_15         0b1110  // 1:15にタイマ2出力を分周
    #define set_Postscaler_1_16         0b1111  // 1:16にタイマ2出力を分周
    // TMR2ON (Timer2 On bit)
    #define timer2_on                   0b1     // タイマ2をオンとする.
    #define timer2_off                  0b0     // タイマ2をオフとする.
    // T2CKPS (Timer2 Clock Prescale Select bits)
    #define set_Prescaler_1_1           0b00    // 1:1にタイマ2入力を分周
    #define set_Prescaler_1_4           0b01    // 1:4にタイマ2入力を分周
    #define set_Prescaler_1_16          0b10    // 1:16にタイマ2入力を分周
    #define set_Prescaler_1_64          0b11    // 1:64にタイマ2入力を分周

//関数の宣言
void set_PWM(unsigned int a, unsigned int b, unsigned int c, unsigned int d, unsigned int e);
void set_PWM_duty_cycle(unsigned long int a);
void set_timer2(unsigned int a, unsigned int b, unsigned int c, unsigned int d);
```

図 1.72: タイマ 2 設定用ヘッダファイル (PWM)

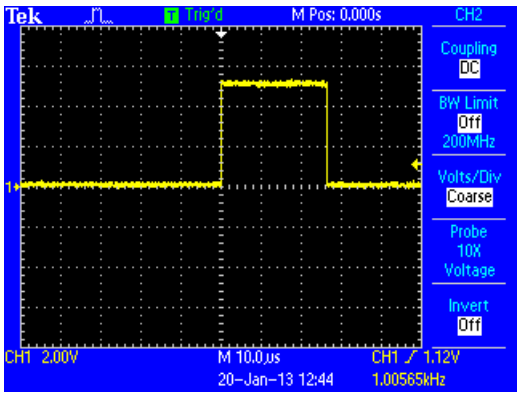


図 1.73: PWM モジュール実行時のタイマ 1 による割り込み処理ルーチンの処理時間（判読性を高めたプログラム）

1.4　DC モータの回転数制御

1.4.1　組み立て

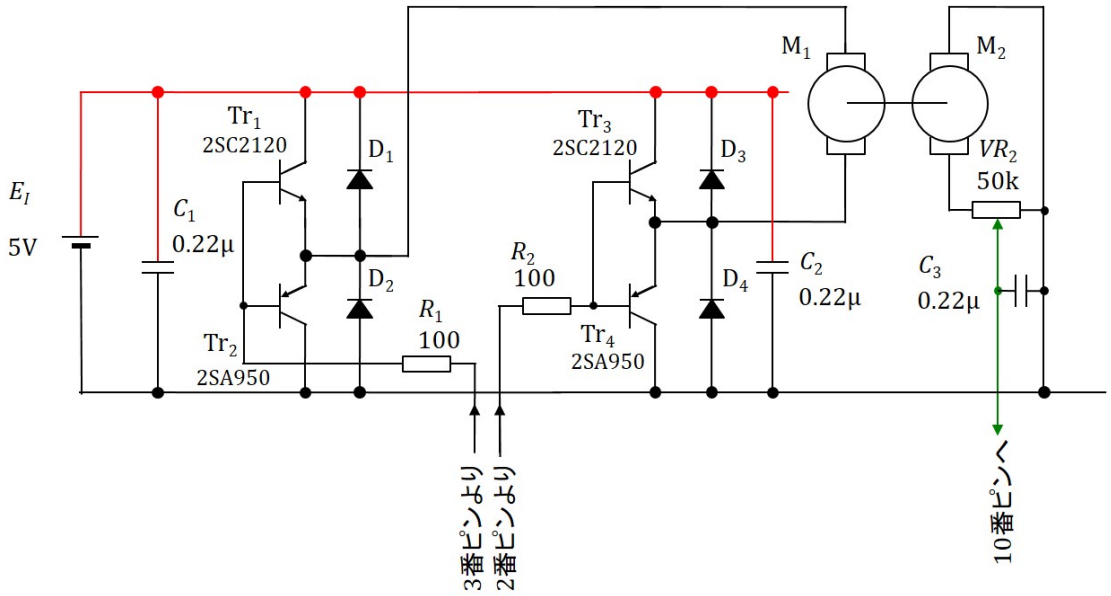


図 1.74: インバータと回転数検出回路

表 1.3: 部品表（追加分）

(2019年9月時点)

品名	型式	個数	単価	値段	入手先の例
トランジスタ	2SC2120 (20個入り)	1	110	110	秋月電子通商
	2SA950（20個入り）	1	110	110	〃
整流用ショットキーダイオード	30 V, 1A	4	20	80	〃
積層セラミックコンデンサ	0.22μF 50V (10個入り)	3	20	60	〃
電解コンデンサ	10μF 16V	1	10	10	〃
半固定ボリューム	50kΩ	1	40	40	〃
電池ボックス	単3×4本 フタ付きプラスチック・スイッチ付き	1	110	110	〃
抵抗	100Ω, 1/4W（100個入り）	1	100	100	〃
DCモータ	マブチ RE-280RA モーターベース付き	2	260	520	アマゾン
			総計	1140	円

図 1.24 の回路に図 1.74 のインバータと回転数検出回路を追加することで図 1.4 の回路を構成できる．前節までで A/D 変換モジュールと PWM モジュールを使えるようになったので，いよいよ DC モータの回転数制御を実現する段階となった．図 1.74 の回路を製作するには，表 1.3 の部品を必要とする．

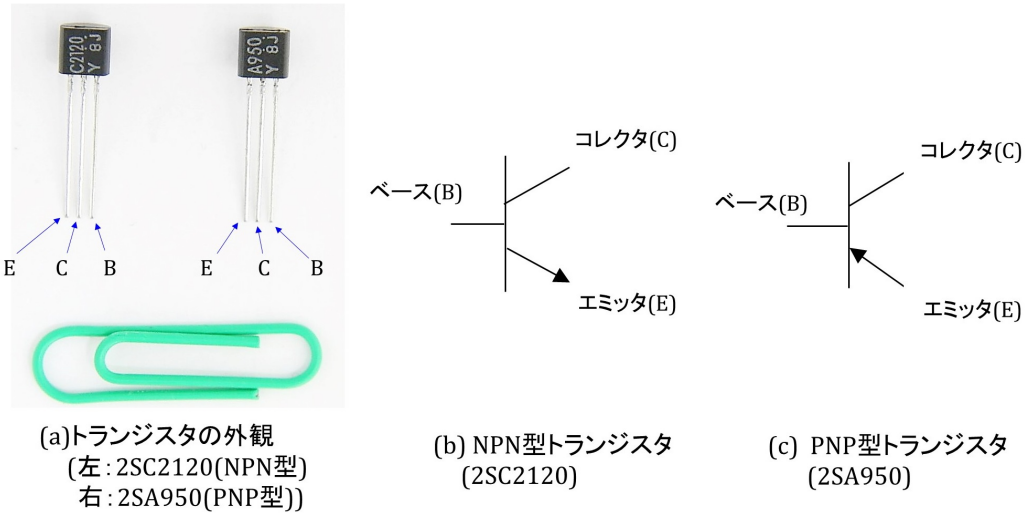


図 1.75: トランジスタの外観と記号

トランジスタ

図 1.75 はトランジスタ Tr_1〜Tr_4 の外観と記号を示す．同図 (a) の左側が 2SC2120 であり，NPN 型トランジスタである．右側は 2SA950 であり，PNP 型トランジスタである．これらはバイポーラトランジスタと呼ばれる．3 つの電極には呼び名があり，写真のようにラベルを上にしたときに，左からエミッタ (E; Emitter)，コレクタ (C: Collector)，ベース (B: Base) である．E, C, B を「えくぼ」と憶えておくと忘れない．同図 (b), (c) はそれぞれ NPN 型, PNP 型トランジスタの記号である．両者はエミッタ電極の矢印の向きで区別されている．これはエミッタ電流の流れる向きを示している．

ダイオード

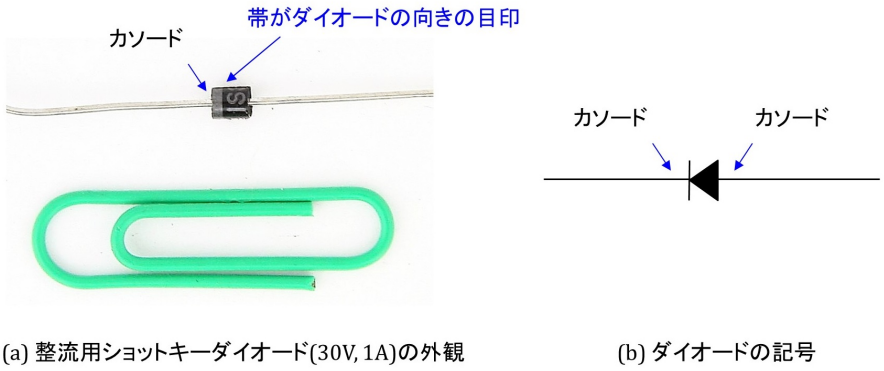


図 1.76: 整流用ショットキーダイオードの外観と記号

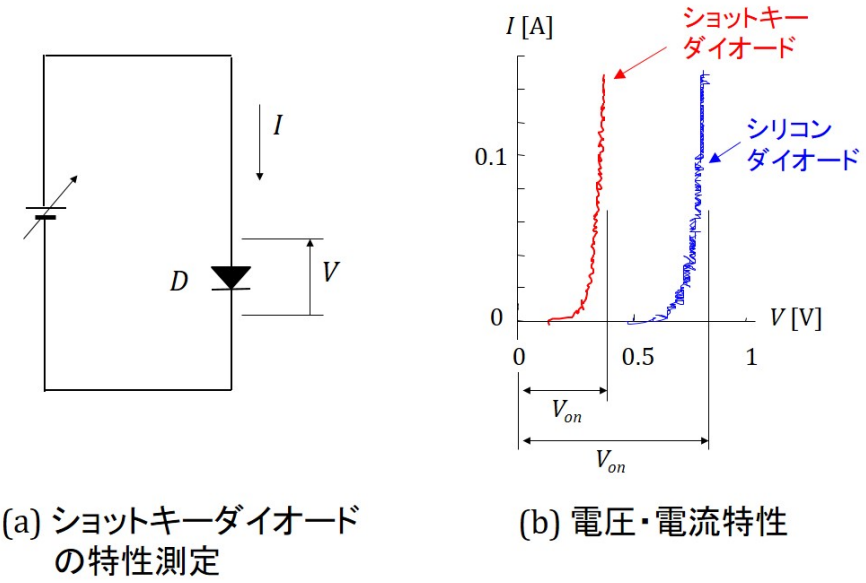


図 1.77: 整流用ショットキーダイオードの特性

図 1.76 は整流用ショットキーダイオードの外観と記号を示す．ダイオードにはアノード電極とカソード電極がある．ダイオードの円筒部分に帯がある側がカソード電極である．ショットキーダイオードの特徴はオン電圧が低いことである．図 1.77(a) はダイオード D の両端電圧 V と電流 I の向きの定義を示し，同図 (b) はその測定結果の例を示す．シリコンダイオードは 100[V], 1[A] の定格のものである．ショットキーダイオードの定格は 30[V], 1[A] である．オシロスコープの測定結果であるためノイズが乗っているが，それぞれのオン電圧 V_{on} に差があることが見て取れる．ショットキーダイオードはオン電圧が低いことから，シリコンダイオードより損失が少ない．

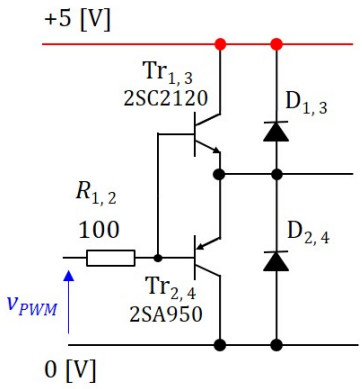


図 1.78: インバータの基本回路

PNP 型と NPN 型のトランジスタをペアとし，それぞれにダイオードを逆方向に接続することでインバータの基本を構成できる．図 1.78 に基本回路を示す．図中の電圧 v_{PWM}

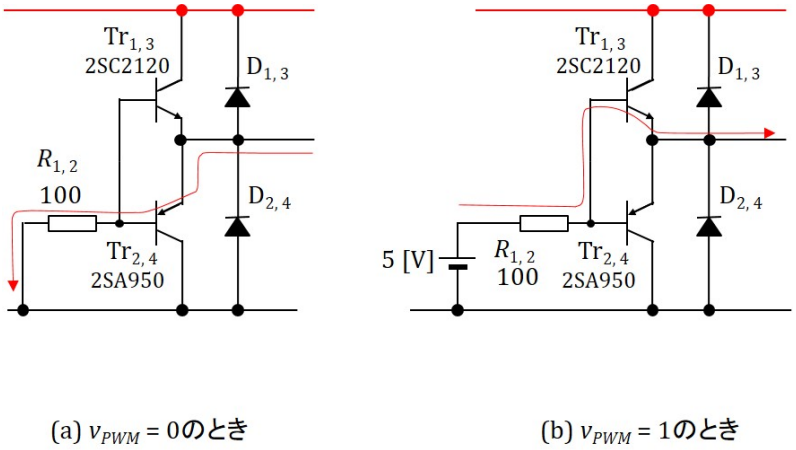
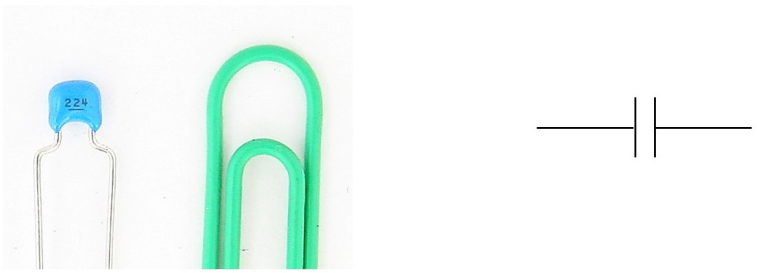


図 1.79: トランジスタの駆動

はマイコンの PWM モジュールの出力電圧であり，図 1.65 の v_{P2A} もしくは v_{P2B} である．マイコンの電源電圧 $V_{E_I} = 5$ [V] とすると，$v_{PWM} = 0$ or 5 [V] の 2 値をとる．トランジスタの駆動の様子を図 1.79 に示す．同図 (a) は $v_{PWM} = 0$ のときであり，(b) は $v_{PWM} = 5$ [V] のときである．$v_{PWM} = 0$ のときは図の経路で下側のトランジスタにベース電流が流れ，このトランジスタがオンとなる．$v_{PWM} = 5$ [V] のときには上側のトランジスタがオンとなる．

積層セラミックコンデンサ



(a) 積層セラミックコンデンサの外観　　(b) コンデンサの記号

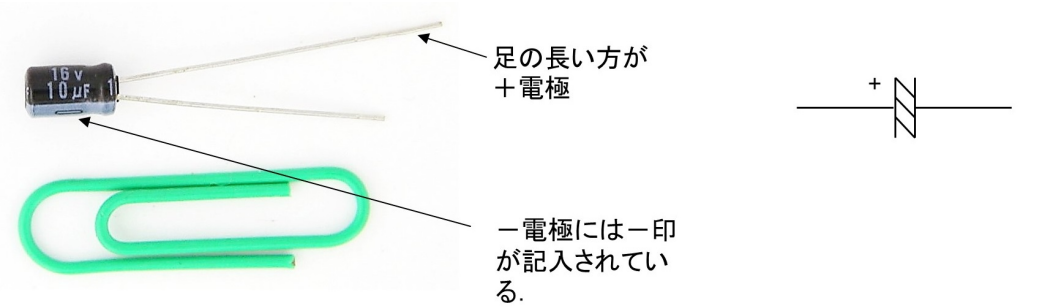
図 1.80: 積層セラミックコンデンサの外観と記号

図 1.80 は積層セラミックコンデンサの外観と記号を示す．セラミックコンデンサはノイズ対策用のコンデンサとして良く用いられる．積層タイプはセラミックコンデンサの静電容量を大きくしたもので，10[μF] の積層セラミックコンデンサもある．図は 0.22[μF] のものである．表面に印字された 224 は

$$\begin{aligned} 224 &= 22 \times 10^4 [\mathrm{pF}] \\ &= 0.22 [\mu \mathrm{F}] \end{aligned} \tag{1.54}$$

を意味する．図 1.74 の回路ではインバータのオン・オフ（スイッチングと呼ばれる）によるノイズ（スイッチングノイズと呼ばれる）の除去用 (C_1, C_2)，および DC モータを発電機として用いた場合の発電電圧に含まれるノイズ除去用 (C_3) として用いられている．

電解コンデンサ



(a) 電解コンデンサ（10μF)の外観　　(b) 電解コンデンサの記号

図 1.81: 電解コンデンサの外観と記号

図 1.81 は電解コンデンサの外観と記号を示す．大容量コンデンサの代表であり，電源電圧の安定化などに良く用いられる．図 1.4 の回路においてマイコン用電源の＋の線（電

源ラインと呼ばれる）と－の線（これも電源ラインと呼ばれる）の間に入れてある．電解コンデンサとセラミックコンデンサを並列にして電源ライン間に入れることで，ノイズに対して電源を強化してある．

DC モータ

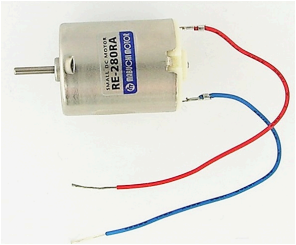


図 1.82: DC モータの外観

図 1.82 は本章で用いる DC モータ（マブチモータ, RE-280RA）の外観を示す．最大効率運転時で 3 [V], 0.87 [A]，出力 1.61 [W] のモータである．

1.4.2　インバータ回路

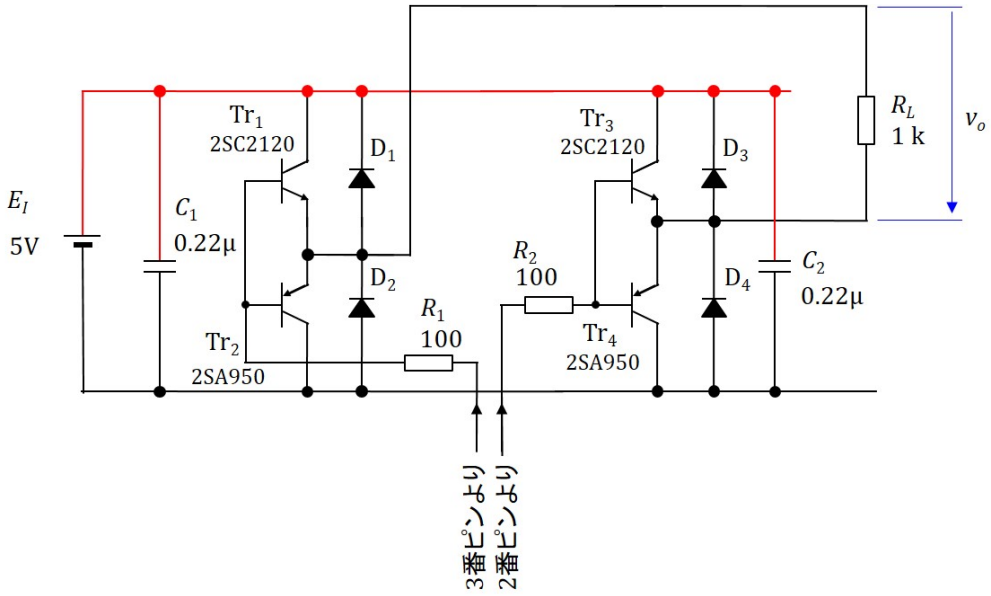


図 1.83: インバータの実験回路

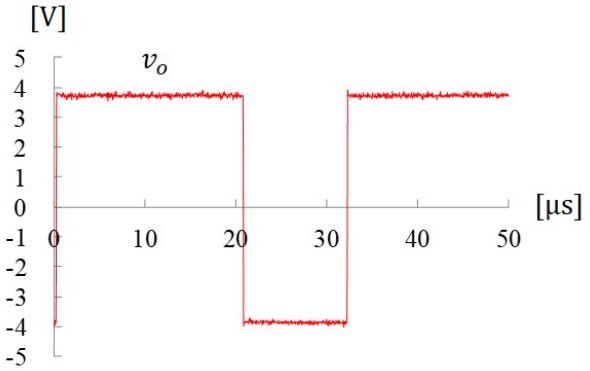


図 1.84: インバータの実験 _ 出力電圧波形（抵抗負荷）

図 1.83 はインバータの実験回路を示す．インバータの出力に抵抗 R_L を接続し，出力電圧 v_o を計測する．図 1.62〜1.63 のプログラムによりマイコンの 2 番ピンに v_{P2A} を出力させて，これにより $\mathrm{Tr}_3, \mathrm{Tr}_4$ を駆動させ，3 番ピンに v_{P2B} を出力させて，これにより $\mathrm{Tr}_1, \mathrm{Tr}_2$ を駆動させる．得られた電圧波形を図 1.84 に示す．PWM 周期は式 (1.39) より 32 [μs] であり，図からも読み取ることができる．また，電圧振幅は約 3.8 [V] である．この値は，$v_{\mathrm{P2A}} = 5$ [V]，$v_{\mathrm{P2B}} = 0$ [V] のとき Tr_3 と Tr_2 がオンとなるが，Tr_3 のベース・エミッタ間電圧を

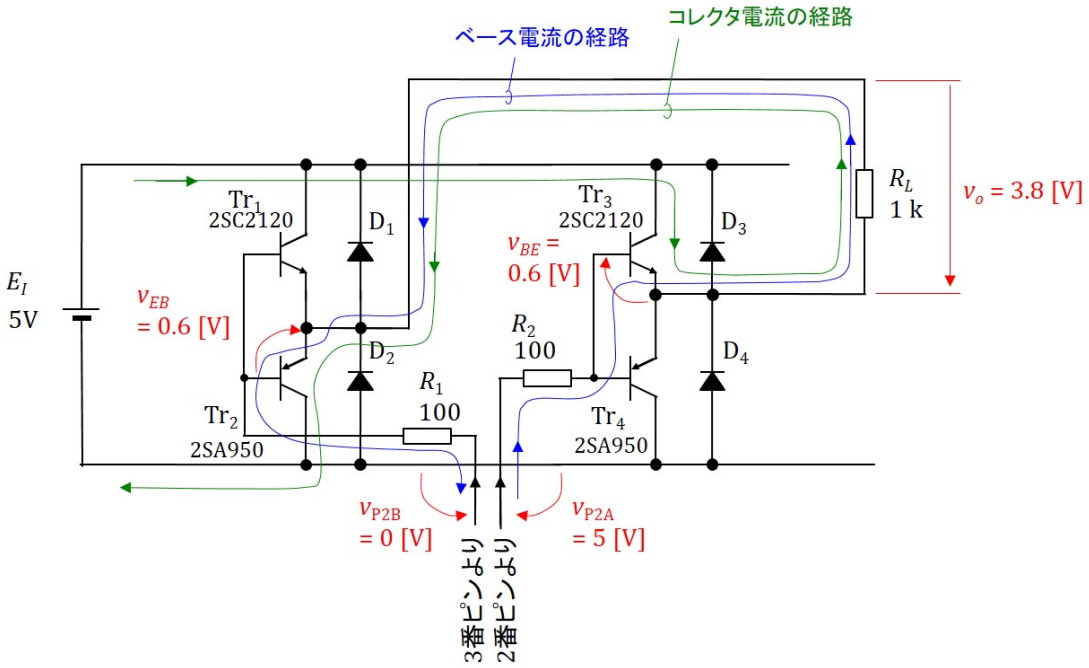


図 1.85: インバータの実験 _v_{BE} の出力電圧への影響

v_{BE}，Tr_2 のエミッタ・ベース間電圧を v_{EB} とすると

$$\begin{aligned} v_o &= v_{\mathrm{P2A}} - v_{BE} - v_{EB} - v_{\mathrm{P2B}} \\ &= 5 - v_{BE} - v_{EB} \\ &\approx 3.8[V] \end{aligned} \tag{1.55}$$

により与えられる．v_{BE} と v_{EB} の和は約 1.2[V] であったことが分かる．この様子を　図 1.85 に示す．青色で示す経路がベース電流の流れる経路である．2 番ピンから 8 番ピンまでの経路には Tr_3 の v_{BE}，出力電圧 v_o，Tr_2 の v_{EB}　$(= -v_{BE})$ がある．トランジスタのベース・エミッタ間の特性は図 1.77 のシリコンダイオードの特性と同じである．同図は NPN 型トランジスタのベース・エミッタ間電圧対ベース電流特性に相当する．PNP 型トランジスタの場合は電圧，電流の極性が反対となる点を除けば，ほぼ同じ特性である．Tr_3 のベース・エミッタ間に 0.6 [V] 程度の電圧がかかることで，ベース電流が数十 [μA] 流れる．それによってコレクタ電流 I_C が

$$\begin{aligned} I_C &= \frac{3.8[V]}{1[\mathrm{k}\Omega]} \\ &= 3.8[\mathrm{mA}] \end{aligned} \tag{1.56}$$

流れる．コレクタ電流の経路を図 1.85 に緑色で示す．

このインバータの駆動方式の場合，インバータの電源電圧 V_{E_I} を約 4.2 [V] 程度にまで下げても，インバータの出力電圧 v_o はほとんど変わらず約 3.8 [V] を維持する．その様子を図 1.86 に示す．この場合も式 (1.55) の関係は変わらない．変わるのはコレクタ電流の

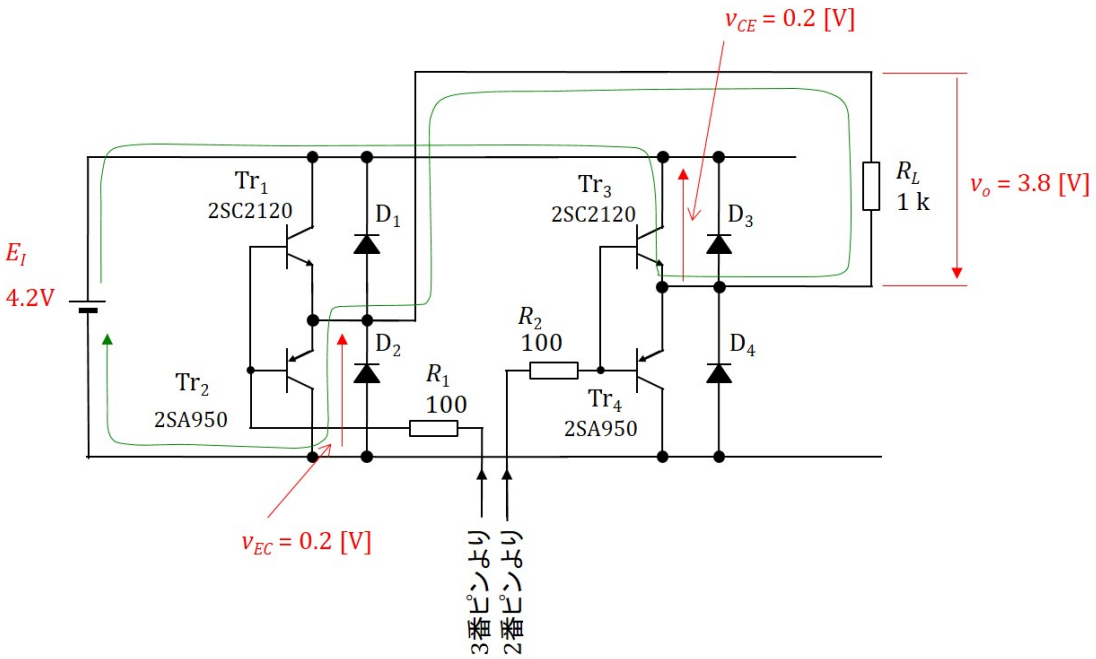


図 1.86: インバータの実験 _v_{CE} の出力電圧への影響

流れる経路における Tr$_3$ のコレクタ・エミッタ間電圧 v_{CE} と Tr$_2$ のエミッタ・コレクタ間電圧 v_{EC} の値である．コレクタ電流の経路上で

$$V_{E_I} = v_{EC} + v_o + v_{CE} \tag{1.57}$$

の関係が成立し，2SC2120 と 2SA950 では $v_{EC} \approx v_{CE}$ が成立している．よって，

$$v_{EC} \approx v_{CE} \approx 0.2[V] \tag{1.58}$$

である．V_{E_I} をさらに低下させると，v_{EC}, v_{CE} はほぼこの値を保ったまま，出力電圧 v_o が低下する．逆に V_{E_I} を図 1.85 の 5 [V] よりも増大させた場合は，出力電圧の 3.8 [V] はほとんど変わらず，v_{EC}, v_{CE} が増大する．

図 1.87 は図 1.83 の負荷抵抗 R_L の替わりに DC モータを接続した場合の出力電圧 v_o，出力電流 i_o の測定例である．モータは回転しないように軸を指先でつまんで固定して測定した．また，波形はオシロスコープにより測定したため，ノイズが目立っている．同図 (a) はデューティ比 $\delta \approx 2/3$ のときであり，(b) は $\delta \approx 0.5$ のときである．(a) の (1) の区間は図 1.85 と同じ経路でコレクタ電流が流れ，出力電圧 $v_o \approx 3.8$ [V] となっている．(2) の区間では出力電流 i_o が正であるが，出力電圧 v_o は負である．出力電流 i_o の経路を図 1.88 に示す．DC モータは電機子抵抗 R_a，電機子インダクタンス L_a の等価回路で表してある．i_o は電源 E_I からダイオード D_4，DC モータ，ダイオード D_1 を通って E_I に戻っている．出力電圧は電源電圧 V_{E_I}= 5[V] に二つのショットキーダイオードのオン電圧 $V_D \approx 0.45$ [V] を加えた値が v_o の矢印とは逆向きに印加されている．その結果として $v_o \approx -5.9$[V] である．(2) の区間の出力電流 i_o は DC モータの電機子インダクタンス L_a が流している．こ

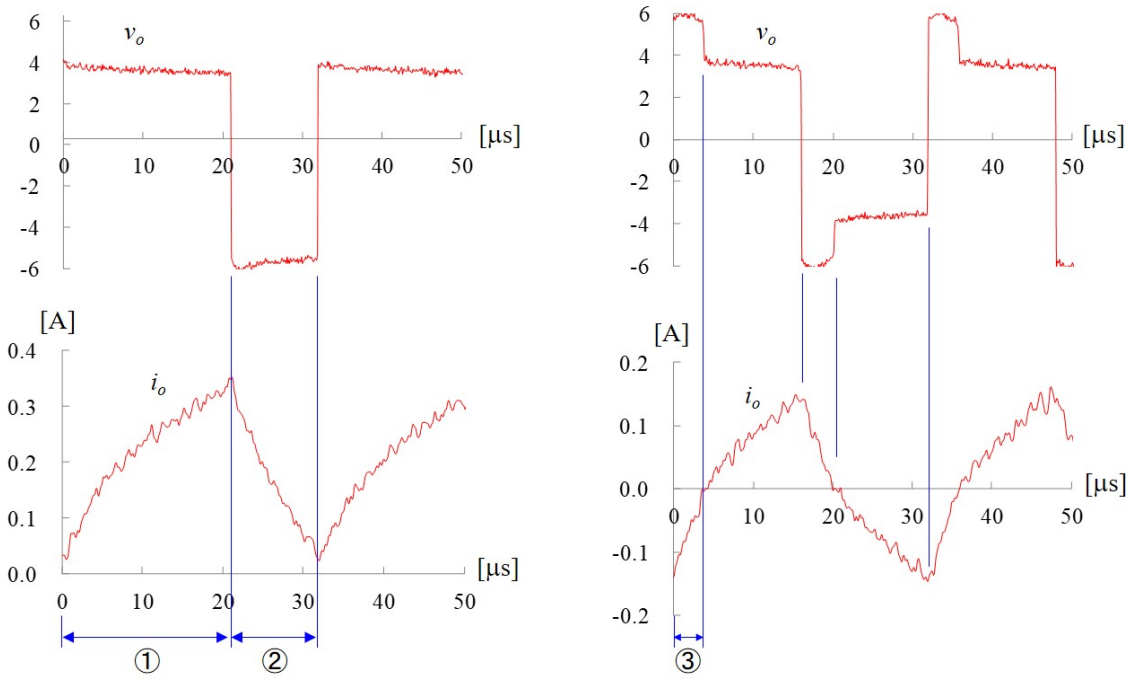


図 1.87: インバータの実験 _ 出力電圧・電流波形（DC モータ負荷）

の間 L_a の磁気エネルギは電源 E_I に電気エネルギとして戻されている．これはエネルギ回生と呼ばれる．エネルギ回生を行っている回路モードは回生モードと呼ばれる．

図 1.89 は図 1.87(b) の区間 (3) における出力電流 i_o の経路を示す．この区間では i_o は負であり，v_o が正である．この区間も電機子インダクタンス L_a が i_o を流す回生モードである．

インバータ回路とその動作の詳細は拙著 [5] を参照されたい．

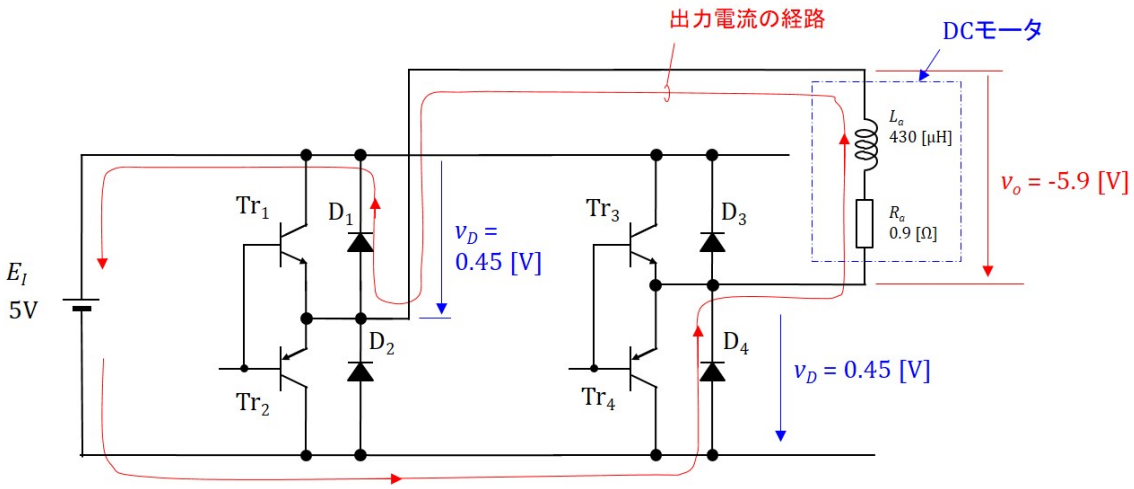


図 1.88: インバータの実験 _ 回生モード（区間 (2)，DC モータ負荷）

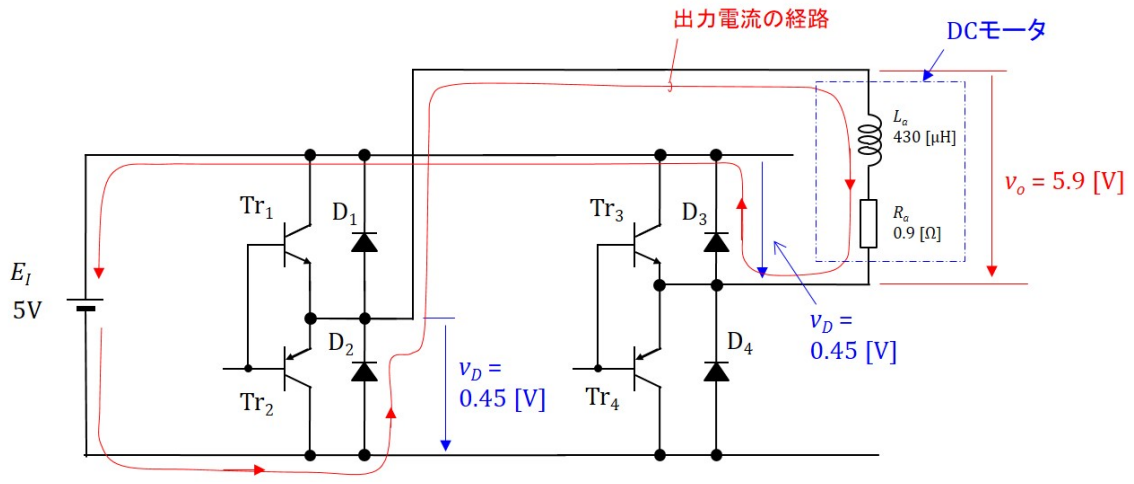


図 1.89: インバータの実験 _ 回生モード（区間 (3)，DC モータ負荷）

1.4.3　フィルタ回路

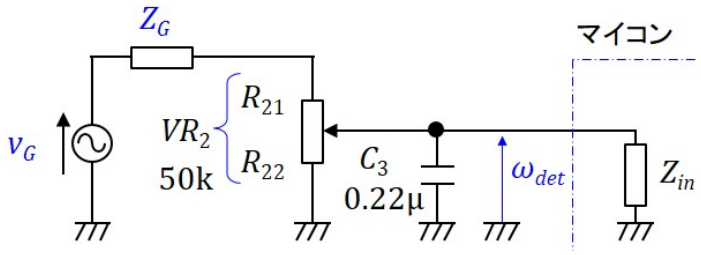


図 1.90: フィルタ回路

図 1.74 の発電機 M_2，可変抵抗 VR_2 とコンデンサ C_3 からなる回転数検出回路において，M_2 は DC モータを発電機として用いている．後述の式 (2.10) より，発電電圧（モータの逆起電力）E_a はモータの回転数に比例するので，E_a により回転数を検出できる．ただし，これも後述するように，モータの逆起電力には図 2.28 のように大きな脈動成分（リップルと呼ばれる.）が含まれている．VR_2 と C_3 からなるフィルタ回路はこのリップルをとるための回路である．図 1.90 はフィルタ回路の等価回路である．v_G は発電機の発電電圧であり，ω_{det} はフィルタ回路の出力電圧である．VR_2 の可動電極の上側の抵抗値を R_{21}，下側のそれを R_{22} とする．発電機の内部インピーダンス Z_G は電機子抵抗が約 1[Ω]，電機子インダクタンスが約 400[μH] であるので，$VR_2 = 50$ [kΩ] と比べて小さく無視できる．マイコン側の入力インピーダンス Z_{in} は，A/D コンバータの入力であり，データシートの ANALOG INPUT MODEL より，静電容量は 15[pF]，漏れ電流は 100[nA] 程度であるので，フィルタ回路のインピーダンスに比べて大きく無視できる．

ω_{det} と v_G の関係は

$$\begin{aligned}\omega_{det} &= \frac{\frac{R_{22}}{1+j\omega C_3 R_{22}}}{R_{21}+\frac{R_{22}}{1+j\omega C_3 R_{22}}} v_G \\ &= \frac{R_{22}}{R_{21}+R_{22}+j\omega C_3 R_{21} R_{22}} \\ &= \frac{\frac{R_{22}}{R_{21}+R_{22}}}{1+j\frac{\omega C_3 R_{21} R_{22}}{R_{21}+R_{22}}}\end{aligned} \tag{1.59}$$

である．よって，この回路のカットオフ周波数 f_c は

$$f_c = \frac{R_{21}+R_{22}}{2\pi C_3 R_{21} R_{22}} \tag{1.60}$$

と求められる．$R_{21} = 10[\mathrm{k\Omega}], \mathrm{R}_{22} = 40[\mathrm{k\Omega}]$ とすると，$C_3 = 0.22[\mu\mathrm{F}]$ であるので，$f_c \approx 90$[Hz] である．

図 1.91 はフィルタ回路の入力電圧 v_G と出力電圧 ω_{det} の測定例である．この例ではリップルの繰り返し周波数は約 430 [Hz] であり，フィルタ回路はこの成分を大きく低減している．

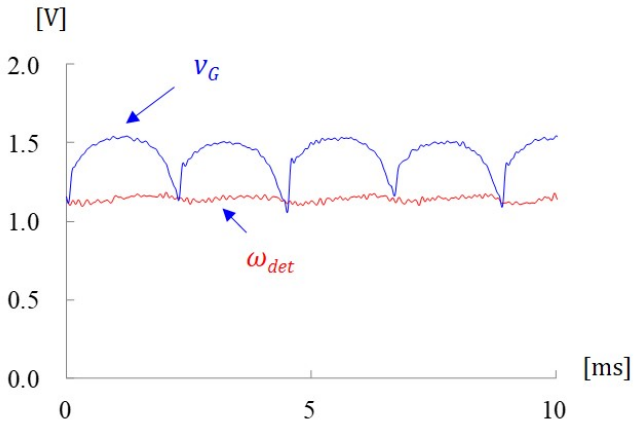


図 1.91: フィルタ回路の入出力

1.4.4　PI 制御と実験結果

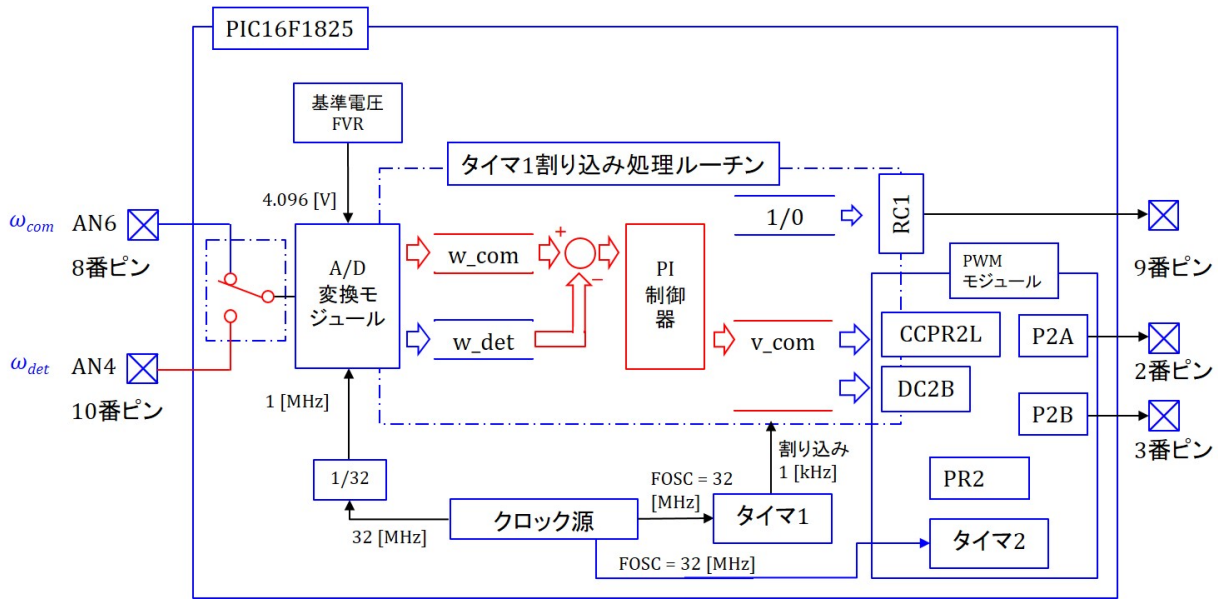


図 1.92: DC モータの回転数制御プログラムのブロック図

以上で 1.1 節の DC モータ回転数制御回路のプログラミングと実験の準備が整った．図 1.92 は作成したプログラムのブロック図を示す．図 1.59 の PWM モジュールプログラムのブロック図に対して新たに追加した箇所を朱書きで示す．図 1.4 に示すように回転数指令値 ω_{com} を 8 番ピンに入力し，回転数検出値 ω_{det} を 10 番ピンに入力している．PIC16F1825 内には A/D コンバータは 1 つしかないので，入力を切り替えてそれぞれの電圧を検出する．変換結果は ω_{com} を w_com に，ω_{det} を w_det にそれぞれ格納する．両者の差を PI 制御器の入力とし，出力を電圧指令値 v_com に格納する．

```
#define _XTAL_FREQ 32000000

#define Kp          32          // 比例ゲイン
#define Kidt         1          //  積分ゲイン

signed int w_com;               //回転数指令値
signed int w_det;               //回転数検出値
signed int w_diff;              //回転数差分
signed int v_com;               //電圧指令値
signed int w_diff_integ = 0;    //積分項
```

図 1.93: DC モータ回転数制御プログラム (DC_Motor_Cont_direct)

制御プログラムを図 1.93，1.95，1.96 に示す．図 1.62，1.63 の PWM モジュールのプログラムに対して新たに追加した箇所を青色にしてある．main プログラムの変更は

$$\text{TRISC} = 0\text{x}05; \quad //\text{ポート RC0, RC2 を入力ポートに設定する．} \tag{1.61}$$

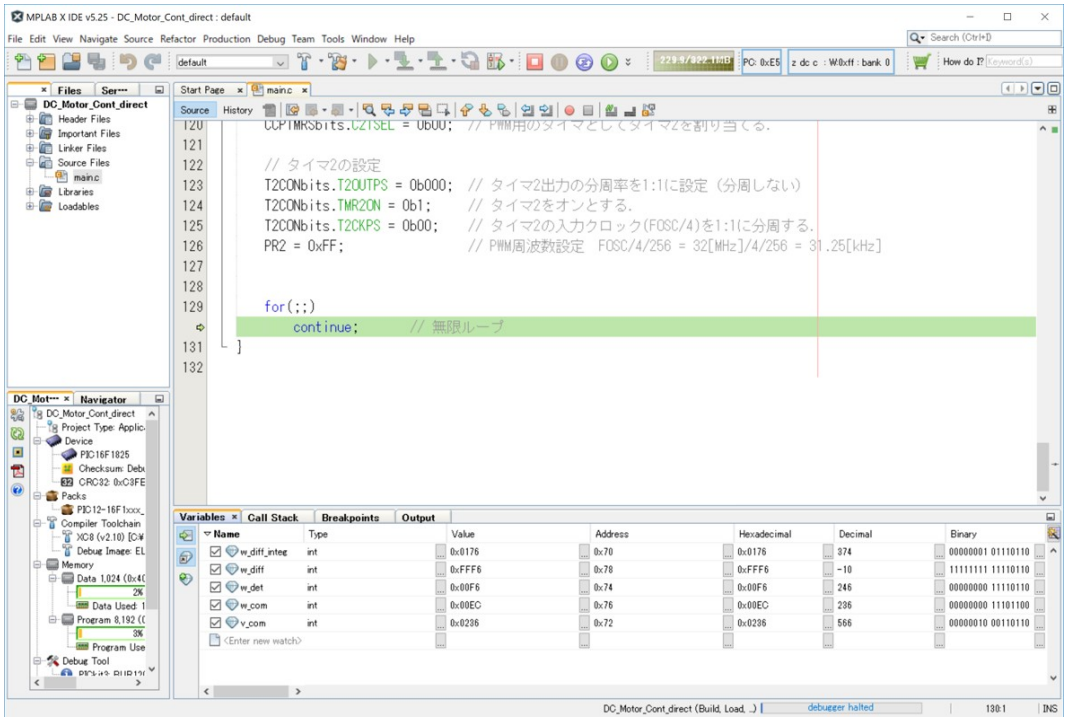


図 1.94: MPLAB X の Variables ウィンドウ

の一カ所のみであるので図には載せていない．上述のようにこのプログラムでは A/D 変換モジュールの入力を 8 番ピンと 10 番ピンで切り替える（チャネルを AN6 と AN4 で切り替える）ことを行う．チャネルの切り替え後は A/D コンバータの入力電圧が安定するまで待ってからでないと A/D 変換を精度良く行えない．データシートの A/D Acquisition Requirements より待ち時間 (Acquisition time) は 5 [μs] とれば十分であることが分かる．ただし，アナログ信号源の内部抵抗（図 1.4 の回路では VR_1, VR_2）が 10 [kΩ] 以下という条件がある．$VR_1 = 10[\mathrm{k}\Omega]$ であるので，可変抵抗器の可動電極の位置がどこであってもこの条件を満たすことができる．一方，$VR_2 = 50[\mathrm{k}\Omega]$ であるので，可動電極の位置によってはこの条件を満たすことができない．VR_2 と C_2 によるフィルタ回路のカットオフ周波数の設定次第である．図 1.91 の例では $R_{21} = 10[\mathrm{k}\Omega], \mathrm{R}_{22} = 40[\mathrm{k}\Omega]$ であり，発電機側の内部抵抗はほぼ $R_{21} = 10[\mathrm{k}\Omega]$ であるので，データシートの条件を満たすことができる．

時間待ちは

```
__delay_us(5);
```　(1.62)

により実行できる．この delay 関数は XC8 コンパイラの組み込み関数である．XC8 コンパイラをダウンロードすれば自動的に利用可能となる．このコンパイラの Web ページから無償でダウンロードできる XC8 User's Guide の Libray Functions の __DELAY_US の説明によると，この設定で 5 [μs] の時間待ちができる．また，同説明には delay 関数を利用するには事前に _XTAL_FREQ を定義しなければならないとある．そこで，システムクロック FOSC が 32[MHz] であるので

```
#define  _XTAL_FREQ  32000000
```　(1.63)

としている．次に，PI 制御系の比例ゲイン Kp と積分ゲイン Kidt を

$$\begin{aligned} &\#\text{define} \quad \text{Kp} \quad 32 \\ &\#\text{define} \quad \text{Kidt} \quad 1 \end{aligned} \tag{1.64}$$

と定義している．

タイマ 1 による割り込み処理ルーチン内で用いる変数 w_com ～ w_diff_integ は全てグローバル変数として定義している．いずれも 16 ビット符号付き整数である．グローバル変数とするのは，デバッグ時にそれぞれの値を見られるようにするためである．図 1.94 は MPLAB X の Variables ウィンドウにこれらの変数値を表示した例を示す．Debug Project → Pause → Variables → Enter new watch(右クリック) → New Watch → （プルダウンメニューから変数選択）により表示する変数を選択できる．

```
static void __interrupt()          // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    RC1 = 1;                       // ポートCのRC1に1を出力する．割り込み発生のモニタリング用

    TMR1 = 0x8330;                 // タイマ1のカウントアップ値の設定．
                                   // 入力値を初期値としてカウントアップを行い，オーバーフローで割り込みを発生
    PIR1bits.TMR1IF = 0;           // タイマ1の割り込みフラグを0にして，次のタイマ1による割り込みを受付け可とする．

    ADCON0bits.CHS = 0b00110;  // アナログチャネルをAN6(8番ピン)に設定
    __delay_us(5);             // チャネル切り替え後にA/Dコンバータの入力電圧が安定するまでの待ち時間
    ADCON0bits.GO = 0b1;       // A/Dコンバータの変換開始
    while(ADCON0bits.GO);      // 変換終了待ち
    w_com = ADRES;             // 回転数指令値の読み出し

    ADCON0bits.CHS = 0b00100;  // アナログチャネルをAN4(10番ピン)に設定
    __delay_us(5);             // チャネル切り替え後にA/Dコンバータの入力電圧が安定するまでの待ち時間
    ADCON0bits.GO = 0b1;       // A/Dコンバータの変換開始
    while(ADCON0bits.GO);      // 変換終了待ち
    w_det = ADRES;             // 回転数検出値の読み出し
```

図 1.95: DC モータ回転数制御プログラム（つづき 1）(DC_Motor_Cont_direct)

図 1.95，1.96 はタイマ 1 による割り込み処理ルーチン内のプログラムである．図 1.95 では回転数指令値と回転数検出値を A/D 変換して，マイコンに取り込んでいる．データシートの A/D CONTROL REGISTER 0 より

$$\text{ADCON0bits.CHS} = \text{0b00110;} \tag{1.65}$$

により A/D 変換モジュールの入力にチャネル AN6 を選択できる．その後，式 (1.62) の delay 関数により，A/D コンバータの入力電圧が安定するまでの 5 [μs] の時間待機させている．この待機はチャネルを切り替える度に必要となる．

図 1.96 は PI 制御を行うプログラムである．PI 制御の出力値を v_{com} とすると，この値は次式により与えれる．回転数指令値を ω_{com}，回転数検出値を ω_{det}，比例ゲインを K_p，積分ゲインを K_i として，

$$v_{com} = K_p(\omega_{com} - \omega_{det}) + K_i \int (\omega_{com} - \omega_{det})dt \tag{1.66}$$

```
    w_diff = w_com - w_det;          // 回転数の差分
    w_diff_integ += w_diff;          // 積分項

    if(w_diff_integ >= 1023)         //  リミッタ    0<= w_diff_integ <= 1023
    {
        w_diff_integ = 1023;
    } else if(w_diff_integ < 0)
    {
        w_diff_integ = 0;
    }

    v_com = Kp * w_diff  + Kidt * w_diff_integ + 512;  // PI制御出力の計算

    if(v_com >= 1023)                //  リミッタ     0<= w_com <= 1023
    {
        v_com = 1023;
    } else if(v_com < 0)
    {
        v_com = 0;
    }

    CCP2CONbits.DC2B = v_com  & 0b00000011;    // DC2B に電圧指令値の下位2ビットを格納する.
    CCPR2L = v_com >>2;                        // CCPR2L に電圧指令値の上位8ビットを格納する.

    RC1 = 0;              // ポートCのRC1に0を出力する. 割り込み発生のモニタリング用
}
```

図 1.96: DC モータ回転数制御プログラム（つづき 2）(DC_Motor_Cont_direct)

である．マイコンの中ではタイマ 1 の割り込みにより 1 [ms] の周期で ω_{com}，ω_{det} がサンプリングされているので，上式を離散式で近似する．時刻 k におけるサンプル値をそれぞれ $\omega_{com}(k)$，$\omega_{det}(k)$ とすると，出力値 $v_{com}(k)$ は

$$v_{com}(k) = K_p(\omega_{com}(k) - \omega_{det}(k)) + K_i \Delta t \sum_{j=1}^{k} (\omega_{com}(j) - \omega_{det}(j)) \tag{1.67}$$

となる．Δt はサンプリング周期である．積算 $\sum_{j=1}^{k}$ の開始時点はマイコンの制御開始時点とする．プログラム中では

$$\begin{aligned} \text{w\_diff} &= \text{w\_com} - \text{w\_det}; \\ \text{w\_diff\_integ} &+= \text{w\_diff}; \end{aligned} \tag{1.68}$$

により積算を行い，w_diff_integ に積算値が格納される．リミッタはこの積算値を $0 \leq \text{w\_diff\_integ} \leq 1023$ の範囲内に抑えている．

PI 制御の出力値 v_com は

$$\text{v\_com} = \text{Kp} * \text{w\_diff} + \text{Kidt} * \text{w\_diff\_integ} + 512; \tag{1.69}$$

により得ている．式 (1.67) 中の $K_i \Delta t$ を Kidt としている．v_com は PWM 制御のデューティ比 δ の設定値である．$0 \leq \delta \leq 1023$ の制限があるので，リミッタにより $0 \leq \text{v\_com} \leq 1023$ としている．このことから，w_diff_integ の上限値は 1023/Kidt の小数点以下を切り捨てた値を用いると良い．

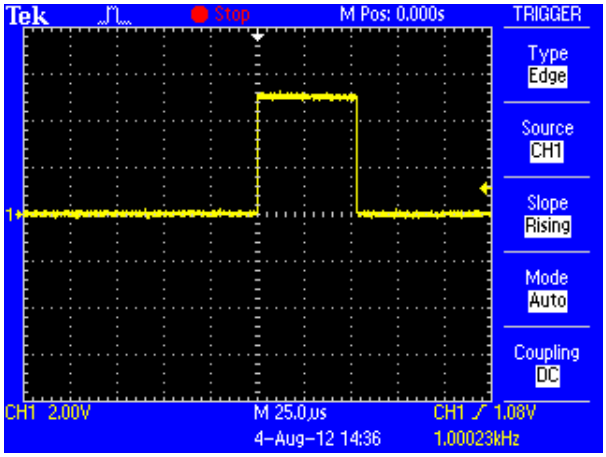


図 1.97: DC モータの回転数制御プログラムの処理時間 ($K_p = 32, K_{idt} = 1$)

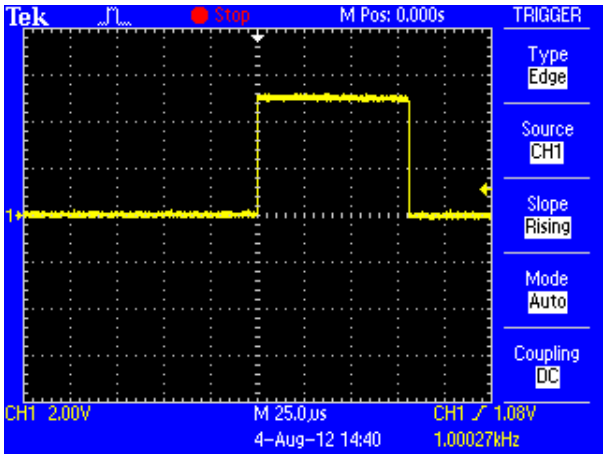


図 1.98: DC モータ回転数制御プログラムの処理時間 ($K_p = 31, K_{idt} = 1$)

図 1.97 は図 1.93〜1.96 のプログラムにおいて，タイマ 1 による割り込み処理ルーチンの処理時間である．横軸は 25[μs/div] である．約 55[μs] を要している．これは比例ゲイン $K_p = 32$，積分ゲイン $K_{idt} = 1$ の場合である．式 (1.69) の計算に要する時間は，K_p，K_{idt} の値によって大きく異なる．図 1.98 は，$K_p = 31$，$K_{idt} = 1$ とした場合のタイマ 1 による割り込み処理ルーチンの処理時間である．約 80[μs] を要している．K_p を 32 から 31 へと変えたことで処理時間は約 25[μs] 延びている．さらに，$K_p = 31$，$K_{idt} = 3$ として，K_{idt} も変えた結果が図 1.99 である．処理時間は約 115[μs] となり，$K_p = 32$，$K_{idt} = 1$ のときより約 60[μs] 増えている．かけ算は 2 のべき乗（$2^n, n = 0, 1, 2, 4, \cdots$ ）をかける場合には処理に要する時間が短い．

図 1.100 は図 1.1 の DC モータの回転数制御回路による実験結果を示す．制御プログラムは図 1.93〜1.96 のプログラムを用い，$K_p = 32, K_{idt} = 1$ の場合である．図 1.4 の回路における回転数指令値 ω_{com} と回転数検出値 ω_{det} を表示してある．図からは ω_{com} がステップ的に変化した場合のモータの回転数の応答の様子が分かる．制御回路は ω_{com} がステップ

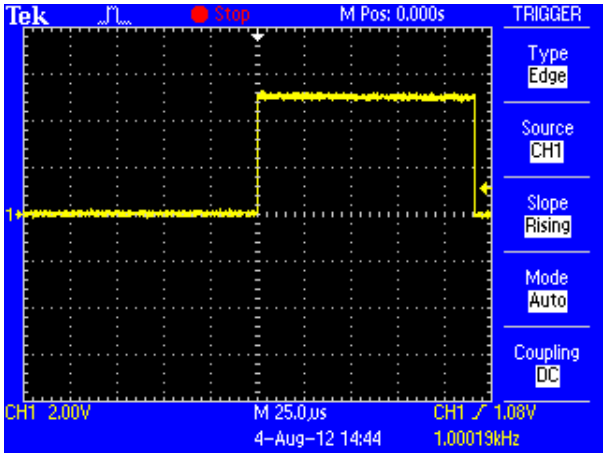


図 1.99: DC モータ回転数制御プログラムの処理時間 ($K_p = 31, K_{idt} = 3$)

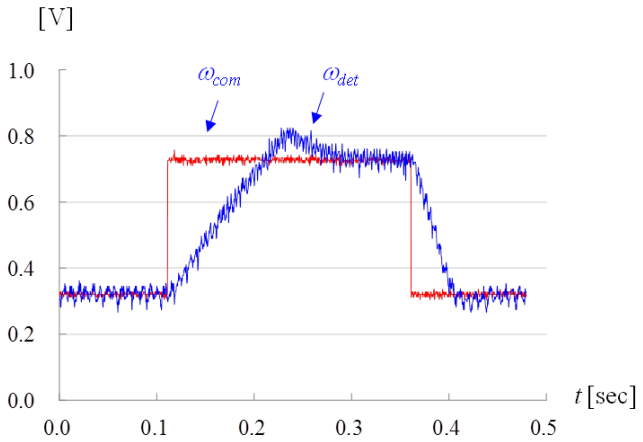


図 1.100: DC モータ回転数制御の実験結果 ($K_p = 32, K_{idt} = 1$)

的に変化した直後は絶対値が最大の電圧を印加してモータを加速／減速している．この回転数の上昇／下降率は主に電源電圧（この実験装置では 5[V]) 及び後述するインバータの出力電圧-電流特性で決まる．減速率の方が加速率よりも大きいが，これは減速時には電源電圧とモータの逆起電力を合わせた電圧により大きな電機子電流を流せることによる．なお，ω_{det} には発電電圧の脈動成分が見られる．図 1.90 の VR_2 と C_3 からなるフィルタ回路ではこの脈動成分を取りきれていないことが分かる．

1.4.5　関数とヘッダファイルの作成

```
timer1_isr(void)

    start_ad_conversion_ch_select(select_AN6); // アナログチャネルをAN6(8番ピン)に設定
    w_com = read_result_of_ad_conversion();    // 回転数指令値の読み出し

    start_ad_conversion_ch_select(select_AN4); // アナログチャネルをAN4(10番ピン)に設定
    w_det = read_result_of_ad_conversion();    // 回転数検出値の読み出し
```

(a) タイマ1による割り込み処理ルーチン

```
set_ad_converter.c

#define _XTAL_FREQ 32000000

void start_ad_conversion_ch_select(unsigned int a)
{
    ADCON0bits.CHS = a;      // アナログチャネル設定
    __delay_us(5);           // 5us 待ち  チャネル変更後にA/D変換器の入力電圧が安定するまでの待ち時間
    ADCON0bits.GO = 0b1;     // A/Dコンバータの変換開始
    while(ADCON0bits.GO);    // 変換終了待ち
}
```

(b) set_ad_converter.c（A/D変換モジュール設定用関数のファイル）内での関数定義

```
pic16f1825_s.h

// 関数の宣言
void start_ad_conversion_ch_select(unsigned int a);
```

(c) ヘッダファイル内の関数宣言

図　1.101:　DC モータ回転数制御プログラムの抜粋（判読性を高めたプログラム）(DC_Motor_Cont)

図 1.101 は DC モータの回転数制御プログラムにおいて新たに定義した start_ad_conversion_ch_select() 関数に関する部分の抜粋を示す．チャネル切り替え直後に A/D コンバータの入力電圧が安定するまでの時間待ちはこの関数内で実行している．同図 (a) はタイマ 1 による割り込み処理ルーチン内に記述した同関数である．この関数の引数で選択するチャネルを指定する．(b) に同関数内の命令を示す．チャネル設定，電圧安定までの時間待ち，A/D 変換開始，変換終了待ちを実行する．(c) のヘッダファイル内では関数宣言を行っている．これを宣言しておかないと，コンパイル時に conflicting declaration のエラーが出される．

図 1.102 は PI_controller() 関数に関する部分の抜粋を示す．図 1.96 の PI 制御のプログラムをこの関数内で実行している．同図 (a) は比例ゲイン Kp と積分ゲイン Kidt を変数として定義し直している．同図 (b) はタイマ 1 による割り込み処理ルーチン内に記述した PI_controller() 関数である．同図 (c) は同関数内の命令を示す．比例ゲイン Kp，積分ゲイン Kidt，回転数指令値 w_com，回転数検出値 w_det，回転数差分 w_diff，回転数差分の積分値 w_diff_integ を extern 変数と宣言することで，PI_controller() 関数のファイル内でこれらのグローバル変数を共有できるようにしている．その後に，図 1.96 のプログラムと同じ処理を行っている．同図 (d) のヘッダファイル内では関数宣言を行っている．

図 1.103 は図 1.101，1.102 のプログラムを含む DC モータの回転数制御プログラムを実行した場合の，タイマ 1 による割り込み処理ルーチンの処理時間である．約 95[μs] を要している．これに対して，図 1.97 の判読性を高める前のプログラムにおけるタイマ 1 による割り込み処理ルーチンの処理時間は約 55[μs] であった．図 1.102 のプログラムにおいて $K_p = 31$，$K_{idt} = 1$ とした場合と $K_p = 31$，$K_{idt} = 3$ とした場合で処理時間はいずれも約 95[μs] でほとんど変わらなかった．

図 1.104 は図 1.1 の DC モータの回転数制御回路による実験結果を示す．制御プログラムは図 1.101，1.102 のプログラムを用い，$K_p = 32, K_{idt} = 1$ の場合である．図 1.100 とほぼ同じ結果が得られている．

```
signed int      Kp  = 32;           // 比例ゲイン
signed int      Kidt = 1;          // 積分ゲイン
```

(a) 比例ゲイン，　積分ゲインの宣言

```
timer1_isr(void)

    PI_controller();
```

(b) タイマ1　による割り込み処理ルーチン

```
// PI controller
void PI_controller()
{
 //PIコントローラの定数，変数，関数の宣言
   extern signed int     Kp;          //比例ゲイン
   extern signed int     Kidt;        //積分ゲイン
   extern signed int     w_com;       //回転数指令値
   extern signed int     w_det;       //回転数検出値
   extern signed int     w_diff;      //回転数差分
   extern signed int     v_com;       //電圧指令値
   extern signed int     w_diff_integ; //積分値

    w_diff = w_com - w_det;           // 回転数の差分
    w_diff_integ += w_diff;           // 積分項

    if(w_diff_integ >= 1023)          // リミッタ    0<= w_diff_integ <= 1023
    {
        w_diff_integ = 1023;
    } else if(w_diff_integ < 0)
    {
        w_diff_integ = 0;
    }

    v_com = Kp * w_diff + Kidt * w_diff_integ + 512; // PI制御出力の計算

    if(v_com >= 1023)                 // リミッタ      0<= v_com <= 1023
    {
        v_com = 1023;
    } else if(v_com < 0)
    {
        v_com = 0;
    }
}
```

(c) PI_controller.c（PIコントローラ関数のファイル）内での関数定義

```
pic16f1825_s.h

// 関数の宣言
void PI_controller(void);
```

(d) ヘッダファイル内の関数宣言

図 1.102: DC モータ回転数制御プログラムの抜粋（つづき）（判読性を高めたプログラム）(DC_Motor_Cont)

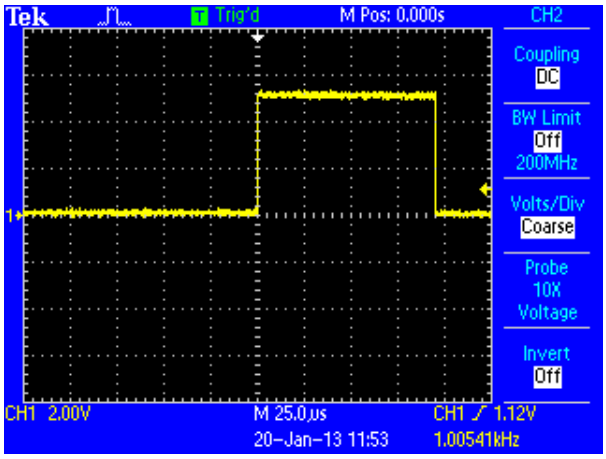


図 1.103: DC モータの回転数制御プログラム（判読性を高めたプログラム）の処理時間 $(K_p = 32, K_{idt} = 1)$

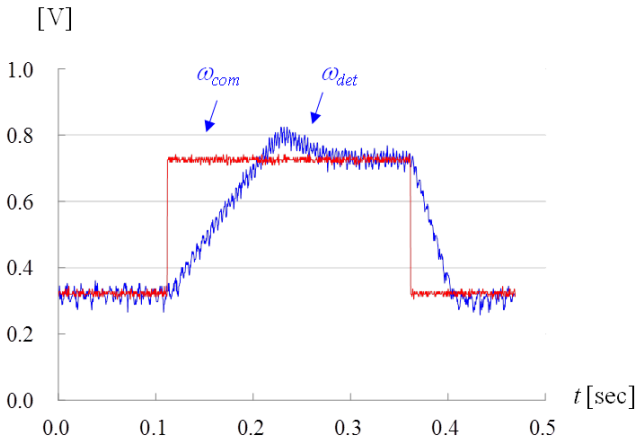


図 1.104: DC モータ回転数制御の実験結果 $(K_p = 32, K_{idt} = 1)$（判読性を高めたプログラムの場合）

索 引

include, 30
_XTAL_FREQ, 83
4 逓倍 PLL, 40

A/D 変換, 23, 45
Acquisition time, 83
ADCON0 レジスタ, 47
ADCON1 レジスタ, 47
ADFM, 47
ADPREF, 47
ANSI C, 30
APFCON1 レジスタ, 60, 66

Build, 18

C2TSEL, 60
CCP2CON レジスタ, 59, 66
CCP2M, 59
CCP2SEL, 60
CCPR2H レジスタ, 55
CCPR2L レジスタ, 55
CCPTMRS レジスタ, 60, 66
CCPxCON レジスタ, 59
CHS, 84
Comparator, 55
Compiler, 14
conflicting declaration, 88

DC2B, 55
Debug, 21
Debugger, 21
delay() 関数, 83
Device, 14

Encoding, 15

FOSC, 34, 55
FOSC_ INTOSC, 32, 34
FVR, 45
FVRCON, 47

GIE, 34

Header Files, 17

ICSP コネクタ, 9, 23
IRCF, 34, 39
isr, 32

LED, 23, 28
Linker Files, 17

MCLRE_ ON, 32
MPLAB®X IDE, 30
MPLAB® Snap, 10
MPLAB® X IDE, 12
MPLAB® XC8 コンパイラ, 12

New Watch, 21
NPN 型トランジスタ, 6, 70

OSCCON，ヘッダファイル, 39
OSCCON レジスタ, 33

P2A, 56
P2B, 56
P2BSEL, 60
P2M, 59
Pause, 21
PEIE, 34
pic16f1825_s.h, 53, 66
PICkit3, 23
PIC マイコン, 26
PI 制御, 84
PLLEN_ON, 32, 34, 39

PNP 型トランジスタ, 6, 70
PORTC レジスタ, 32
PR2, 60
PR2 レジスタ, 55
pragma, 31
Program Speed, 19
Project Location, 15
Project Name, 15
PWM 周期, 56
PWM 周波数, 56, 60
PWM 制御, 6
PWM モジュール出力電圧, 61
PWM 制御法, 56
PWM モジュール, 55

R-S フリップフロップ, 55
RC1, 32
RC フィルタ, 6
Run, 19

SCS, 34, 39
set_ad_converter.c, 50
set_osc.c, 41
set_pwm.c, 65
set_timer1.c, 41
Source Files, 16
SPLLEN, 34, 39

T1CKPS, 34, 41
T1CON，ヘッダファイル, 41
T1CON レジスタ, 34
T2CON レジスタ, 66
TAD, 48
Timer1, 30
Timer2, 55
TMR1CS, 34, 41
TMR1IE, 34
TMR1IF レジスタ, 33
TMR1ON, 34, 41
TMR1 レジスタ, 33
Tool, 14
TRISA レジスタ, 33
TRISC レジスタ, 33

Variables, 21
Variables ウィンドウ, 84
VDD, 26
VSS, 26

XC8 C Compiler, 30
XC8 C Compiler ユーザーズガイド, 30
XC8 ユーザーズガイド, 83
xc.h, 31

アノード, 28, 71
アノード・カソード間電圧, 29

インクルードファイル, 30
インバータ, 69
インバータ用電源, 5

エネルギ回生, 78
エミッタ, 70
エミッタ・コレクタ間電圧, 77
エミッタ電流, 70
エミッタ・ベース間電圧, 76

オン電圧, 29, 71

回生モード, 78
回転数検出回路, 69, 80
回転数検出値, 82
回転数指令値, 82
開発環境, 30
カソード, 28, 71
カットオフ周波数, 80
可変抵抗器, 23, 27
カラーコード表, 28
関数 clear_interrupt_flag_of_timer1(), 37
関数 delay(), 83
関数 PI_controller(), 89
関数 read_result_of_ad_conversion(), 50
関数 set_AD_Converter(), 50
関数 set_FVR(), 50
関数 set_interrupt_by_timer1(), 37
関数 set_osc(), 37

関数 set_PWM(), 63
関数 set_PWM_duty_cycle(), 63
関数 set_timer1(), 37
関数 set_timer1_count_down_ini_num(), 37
関数 set_timer2(), 65
関数 start_ad_conversion_ch_select(), 88
関数 start_ad_conversion(), 50

基準電圧, 45

組み込み関数, 30, 83
グラウンド, 27
グローバル変数, 46, 84
クロック源, 40
クロック周期, 48
クロックソース, 40

コレクタ, 70
コレクタ・エミッタ間電圧, 77
コレクタ電流, 76
CONFIGURATION WORD 1, 2, 31

サンプリング周期, 85

システムクロック, 34, 55
出力ポート, 33
ショットキーダイオード, 6, 71

推奨クロック, 45
スイッチング, 73
スイッチングノイズ, 73

積層セラミックコンデンサ, 73
積分ゲイン, 84

タイマ 1 による割り込み設定, 34
タイマ 2, 55
タイマ 2 の設定, 60
タイマモジュール, 30

抵抗器, 28
データシート, 27
データメモリ, 26
デバイスコンフィギュレーション, 30, 31
デバッガ, 21, 48
デフォルト設定, 41
デューティ比, 56
電解コンデンサ, 73
電機子インダクタンス, 77, 80
電機子抵抗, 77, 80
電源ライン, 73

等価回路, 77
統合開発環境, 30
動作クロック, 45
トランジスタ, 70

入力ポート, 46

バイポーラトランジスタ, 70

比較器, 55
引き数, 37
比例ゲイン, 84
ピン, 26
ピンソケット, 9
ピン配置, 26
ピン番号, 26
ピンヘッダ, 9

ファイル set_ad_converter.c, 50
ファイル set_osc.c, 41
ファイル set_pwm.c, 65
ファイル set_timer1.c, 41
関数 set_timer2(), 66
フィルタ回路, 6, 80
符号付き整数, 84
プリプロセッサ, 32
ブレッドボード, 23, 27
プログラムメモリ, 26
分解能, 56

ベース, 70
ベース・エミッタ間電圧, 75
ベース電流, 76
ヘッダファイル, 30, 37
ヘッダファイル，OSCCON, 39
ヘッダファイル pic16f1825_s.h, 53, 66

ヘッダファイル，T1CON, 41
ヘッダファイル xc.h, 31
ヘッダファイル，デバイス用, 30

マイコン用電源, 5
待ち時間, 83
マブチモータ, 74

脈動成分, 80

無限ループ, 35

メインプログラム, 30

リップル, 80
リミッタ, 85

レジスタ, 32

割り込み, 30
割り込み周期, 33
割り込み処理ルーチン, 32
割り込みフラグ, 33

関連図書

[1] 後閑哲也「改訂版 C 言語による PIC プログラミング入門」技術評論社，2009.

[2] 古橋武「トラ技 Jr. 特集記事　基本回路 10 選！ブレッドボード実験室」 2022 年冬号 (通巻 48 号)

[3] 「ジュニアのためのブレッドボード実験室 (4) 第 4 回はじめての PIC マイコン DC モータ制御」 トランジスタ技術 2022 年 6 月号

[4] モータドライブノート

[5] 古橋武「パワーエレクトロニクスノート」コロナ社，2008.

[6] 古橋武「パワーエレクトロニクスノート II: 製作演習付き講義の実践記録」Kindle 本，Amazon

著者

古橋　武
名古屋大学名誉教授
furuhashi.takeshi*

*に @gmail.com を付けてください．