第4章　8ピンマイコンによる直流(DC)モータ回転数制御

本稿掲載のWebページ

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

古橋　武

目次

第4章

8ピンマイコンによる直流(DC)モータ回転数制御

4.1 組み立て

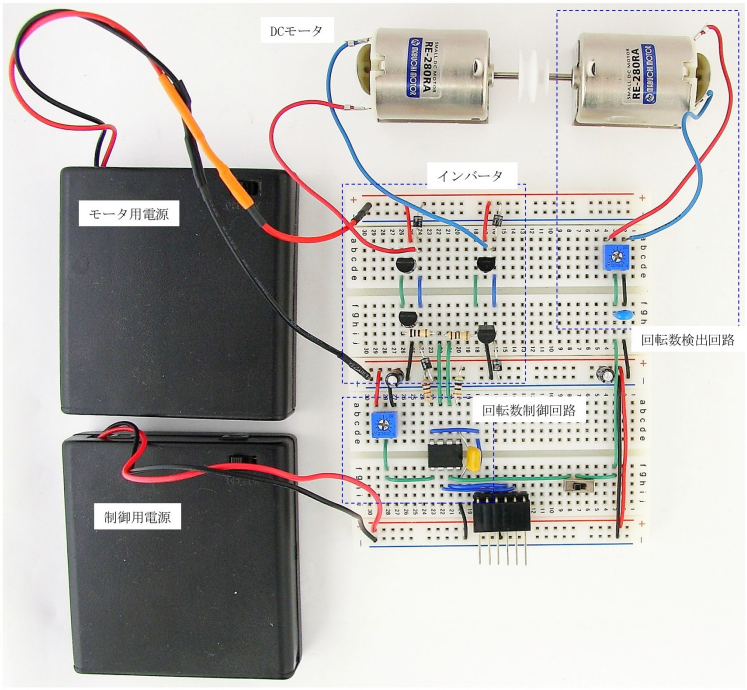


図 4.1: PIC12HV615を用いた直流(DC)モータの回転数制御回路(全体写真)

本章ではPIC12HV615を用いた直流(DC)モータの回転数制御回路を紹介する．PIC12HV615は筆者のパワーエレクトロニクスの講義「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」において長年使用していたPICマイコンである．同講義録ではPIC16F1825を用いた製作演習を紹介しているが，PIC16F1825の弱点は，乾電池4個を直列にした電源電圧が，電池が新しい内は6.3 [V] ほどあり，同マイコンの最大電圧定格6.5 [V] に近い値であったことである．一方，PIC12HV615は内部の電源回路にレギュ

レータを持つ高電圧タイプのマイコンであり，6.3 [V] の電池を使用しても壊れる心配が無い．仮に 10 [V] の電源電圧であっても，レギュレータが 5 [V] に降圧してマイコン内の回路に供給する．外付け回路は抵抗とコンデンサの簡単な回路でよい．なお，充電池 4 個を直列 (5[V]) とした場合には PIC12F615 を用いればよい．PIC12HV615 と PIC12F615 ではピン配置は全く同じである．プログラムの変更は必要ない．

PIC12HV615 を用いた直流 (DC) モータの回転数制御回路の製作例を図 4.1 に示す．その拡大写真を図 4.2 に示す．第 1 章の PIC16F1825 を用いた回路との違いは，マイコンを PIC12HV615 に置き換えたことと，抵抗 R_5，コンデンサ C_6，切り替えスイッチ SW を追加したことだけである．

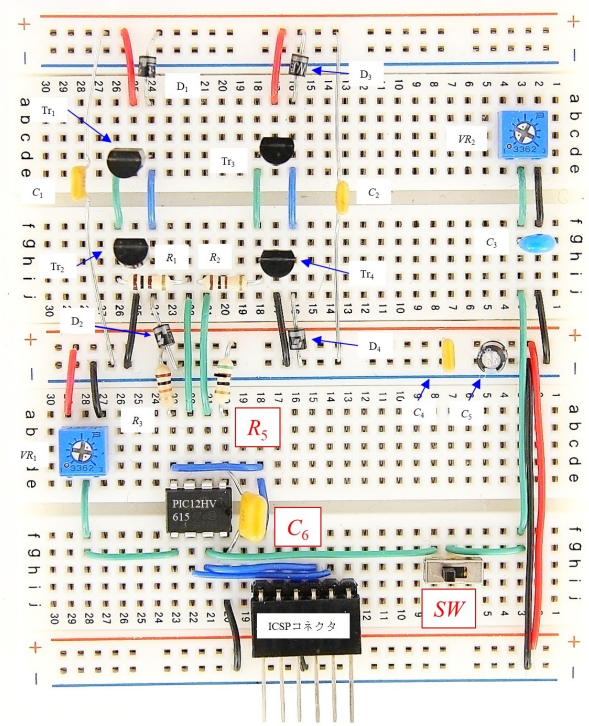


図　4.2: PIC12HV615 を用いた直流 (DC) モータの回転数制御回路 (拡大写真)

立体配線図を図 4.3 に示す．また，その回路図を図 4.4 に示す．インバータ回路は第 1 章図 1.3, 1.4 のものと全く同じである．インバータ，DC モータ，回転数検出回路などの詳細は第 1〜3 章を参照されたい．

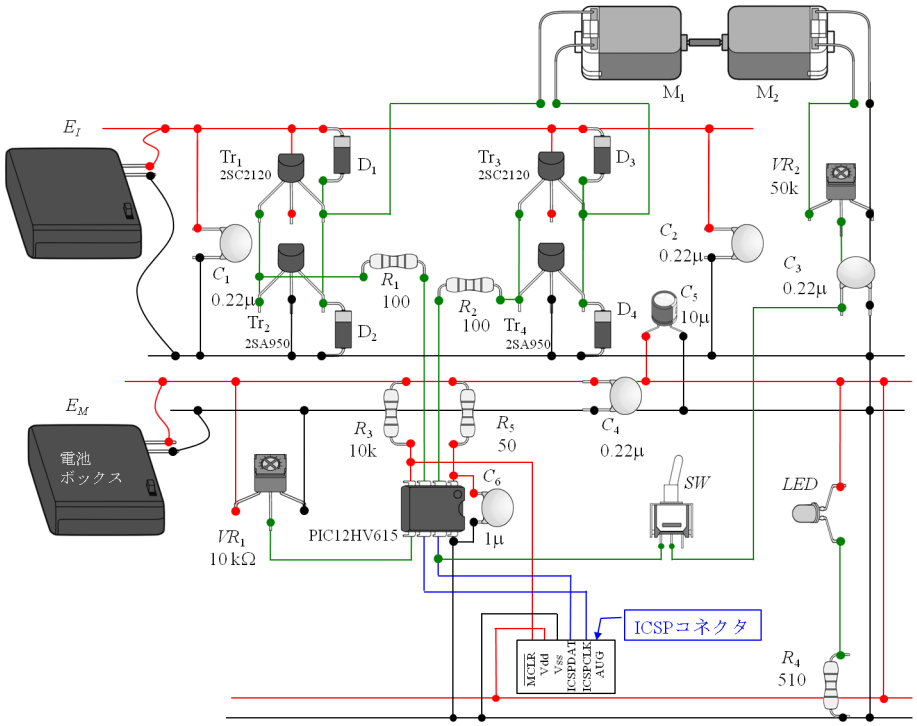


図 4.3: PIC12HV615 を用いた直流 (DC) モータの回転数制御回路 立体配線図

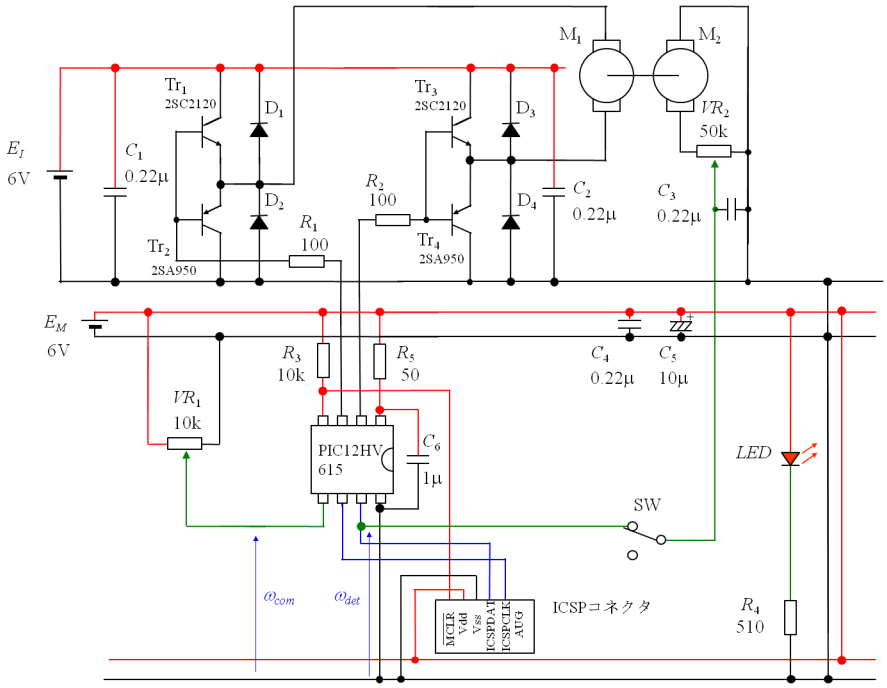


図 4.4: PIC12HV615 を用いた直流 (DC) モータの回転数制御回路図

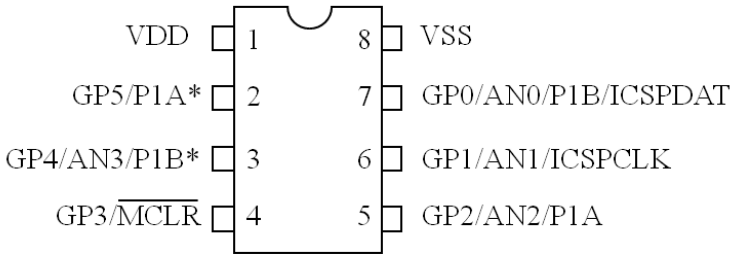


図 4.5: PIC12HV615(PIC12F615) のピン配置

図 4.5 は PIC12HV615(PIC12F615) のピン配置図である．2〜7 番ピンを汎用入出力ポート GP5〜GP0 として利用できる．また，3, 5〜7 番ピンは A/D コンバータの入力 AN3〜AN0 として利用することもできる．PWM 出力は 2, 5 番ピンのいずれかを P1A として，3, 7 番ピンのいずれかを P1B として利用できる．PICkit3, PICkit4 は 4 番ピン ($\overline{\text{MCLR}}$)，6 番ピン (ICSPCLK), 7 番ピン (ICSPDAT) と接続することで利用できる．

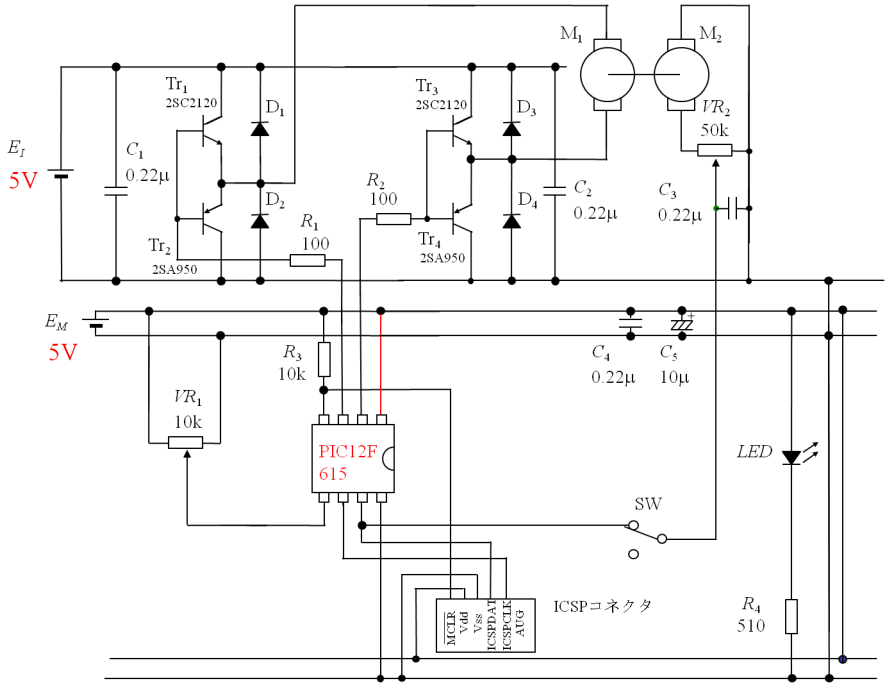


図 4.6: PIC12F615 を用いた直流 (DC) モータの回転数制御回路図

乾電池を充電池に置き代え，電圧を 5[V] として，PIC12F615 を用いた場合の回路図を図 4.6 に示す．図 4.4 から，電源を 5 [V] とし，マイコンを PIC12F615 に置き換え，抵抗 R_5，コンデンサ C_6 を取り除いてある．マイコンのプログラムは PIC12HV615 用のもの

を変更する必要はない．

なお，8 ピンの PIC マイコンでフルブリッジインバータを駆動できるものには，PIC12F615 の他に，PIC12F1822, PIC12F1840 がある．しかし，12HV タイプでは PIC12HV615 のみのようである (2012 年 12 月時点)．

4.2 プロジェクトの作成

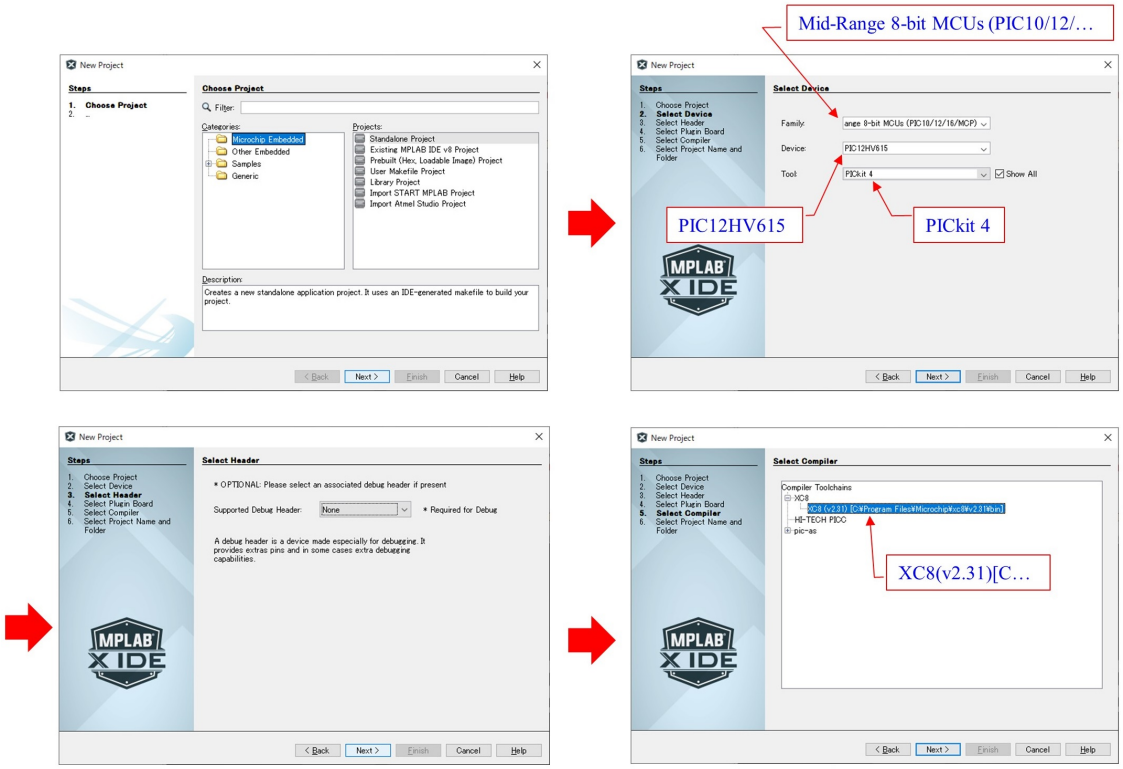


図 4.7: プロジェクトの作成

ソフトウェアの作成，マイコンへの書き込みとデバッグには Microchip 社が無償提供している統合開発環境 MPLAB® X IDE (Integrated Development Environment) を使用する[1]．MPLAB® X IDE のダウンロード、インストール，MPLAB® X IDE の立ち上げ，プロジェクトの作成，ソースファイルの作成は 1.2 節にも詳述してあるので参照されたい．

[1]MPLAB® X IDE の画面を本稿に掲載するに当たっては，マイクロチップ社の許可を得ています．同社の許可なくこれらの画像を複写することはできません．

図 4.7 はプロジェクトを作成している画面のスナップショットを示す．MPLAB X IDE のアイコンをダブルクリックすると同開発環境が立ち上がる．File → New Project とクリックすると，Choose Project のがウィンドウが開かれる．デフォルト（何もしないこと）が Microchip Embedded になっているので，そのまま Next をクリックする．Select Device のページにて Family に Mid-Range 8-bit MCUs を，Device に PIC12HV615 を，Tool に PICkit 4 を選択する．そして，Select Header は何もせず Next をクリックする．Select Compiler では XC8 (v.xxx) [C...] を選択する．

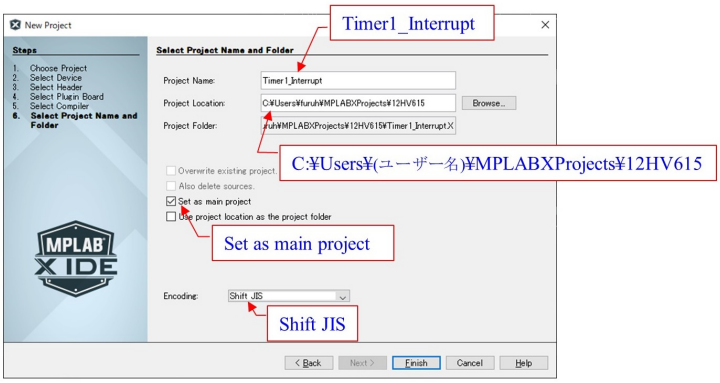


図 4.8: プロジェクトの作成（つづき）

図 4.8 は次に開かれる Select Project Name and Folder のウィンドウである．ここで，Project Location に C:￥Users￥(ユーザー名)￥MPLABXProjects￥12HV615 を入力する．Project Folder は自動的に Project Location が入力される．次に Project Name に Timer1_Interrupt を入力する．Set as main project に レ点を入れる（デフォルトで入っている）．プログラムのコメントを日本語で入力できるようにするためには，Encoding に Shift JIS を選択する．Finish をクリックすれば New Project の作成終了である．

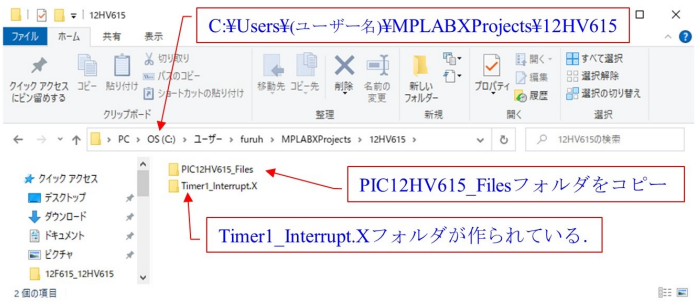


図 4.9: PIC12HV615_Files のコピー

図4.9のように，C:￥Users￥(ユーザー名)のフォルダ内に ￥MPLABXProjects￥12HV615 のフォルダが作成され，その中に Timer1_Interrupt.X のフォルダが作成されたことを確認してください．本稿で紹介するプログラムをモータドライブノート　PIC12HV615_PIC12F615 用 DC モータの回転数制御プログラムに掲載しておきます．ダウンロード・解凍して，そのフォルダの中に PICHV615_Files のフォルダと Timer1_Interrupt のフォルダがあることを確認してください．図4.9のように，C:･･･￥MPLABXProjects￥12HV615 のフォルダ内に PIC12HV615_Files のフォルダをコピーしてください．

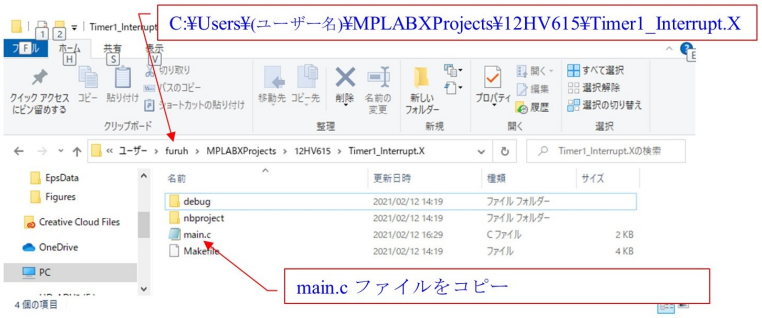


図 4.10: main.c のコピー

同様にして，図4.10のように，C:･･･￥MPLABXProjects￥12HV615￥Timer1_Interrupt.X のフォルダ内に，ダウンロードした同名のフォルダ内の main.c ファイルをコピーしてください．

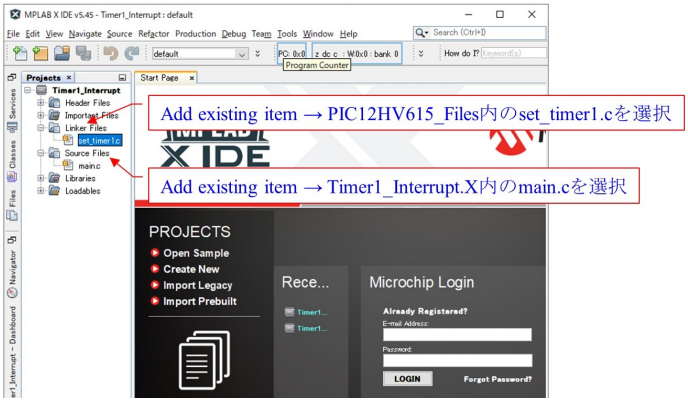


図 4.11: Add_existing_item

最後の設定はファイルのプロジェクトへの追加である．図4.11の様に，Linker Files を右クリック → Add existing item をクリック → PIC12HV615_Files フォルダ内の set_timer1.c

を選択する．Linker Files の左横の +印を左クリックすると Linker Files フォルダの下に set_timer1.c のファイルが現れる．同様にして， Source Files に Timer1_Interrupt.X フォルダ内にコピーしておいた main.c ファイルを追加する．

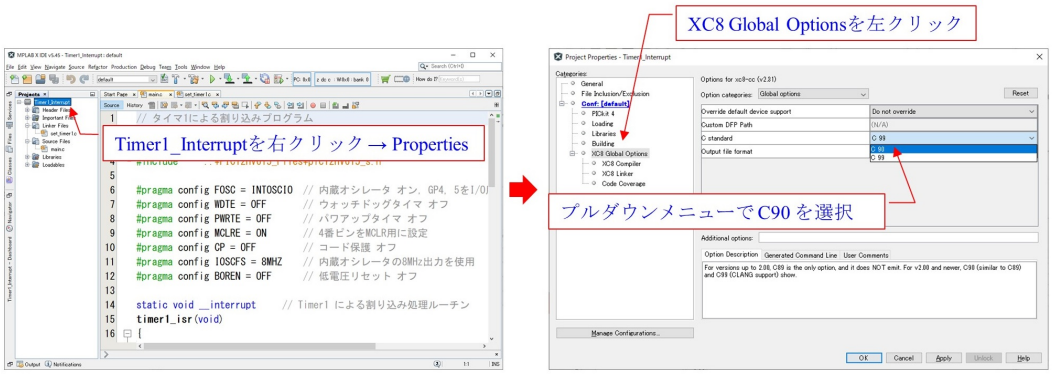


図 4.12: C Standard 設定 を C99 から C90 へと変更

これで設定終了とばかりに，IDE のツールバーにある金槌ボタンを左クリックもしくはキーボードの F11 キーを押すと，“C99 compliant libraries are currently not available for baseline or mid-range devices” とエラーメッセージが出る．C99 は ANSI-C の 1999 年規格である．MPLAB XC8 v2.xx コンパイラはデフォルトで C99 を使用する設定になっているが，PIC12HV615 などの mid-range devices 用 C99 ライブラリはまだ使えないとのことである．そこで，C90(1990 年規格）を使用する設定へと変更する．図 4.12 に示すように，Timer1_Interrupt のプロジェクト名を右クリックし，プルダウンメニュー最下段の Propeties を左クリックする．すると，Project Properties のウィンドウが開かれるので，XC8 Global Options を左クリックする．Optons for xc8-cc のページにおいて C Standard の欄が C99 になっているので，プルダウンメニューから C90 を選択する．これで準備完了である．金槌ボタンを左クリックもしくは F11 キーを押すと “BUILD SUCCESSFUL” とメッセージが出る．

プログラムをマイコンに書き込む際には図 4.4 の *SW* をオフにする．In-Circuit Debugger の例えば PICkit 4 をパソコンと USB ケーブルで接続し，もう一端の ICSP コネクタをブレッドボード上の ICSP コネクタ・ピンに挿入して，マイコンの電源を投入する．▷ ボタンを左クリックすると，使用 Tool を聞いてくる．プルダウンメニューから PICkit 4 を選択すると，プログラムのコンパイルとマイコンへの書き込みを行う．成功すれば “Programming/Verify complete” とメッセージが出る．PICkit 4 を ICSP コネクタ・ピンから抜き，*SW* をオンにして，インバータ回路の電源を投入すればモータ制御が開始される．図 4.4 の可変抵抗器 VR_1 を小さなねじ回しで回せば，回転数を変えられる．もし，

モータが暴走するようであれば，モータ M_1 もしくは発電機,M_2 の極性が間違っているので，いずれかの配線を反転させる．

4.3 部品

表 4.1: 部品表

(2021年2月)

品名	型式	個数	単価	値段	入手先の例
PICマイコン	PIC12F615-I/P	1	80	80	秋月電子通商
〃	PIC12HV615-I/P	1	132	132	Digi-key

表 4.1 に PIC マイコン (PIC12HV615, PIC12F615) の価格（令和 3 年 2 月時点) と入手先の例を記してある．ただし，送料は含まれていない．いずれもネット通販である．

回路内の各部品の詳細，ICSP コネクタと PICKit3 との接続法は第 1 章 1.3.2 項を参照されたい．

マイコン内蔵の電圧レギュレータ用外付け部品

図 4.4 の抵抗 R_5，コンデンサ C_6 は PIC12HV615 内蔵の電圧レギュレータ用外付け部品である．データシート (PIC12F615/HV615_Data_Sheet) の RSER LIMITING REGISTER によると，R_5 の上限値 R_{5MAX} と下限値 R_{5MIN} は

$$R_{5MAX} = \frac{VUMIN - 5[\mathrm{V}]}{1.05 \times (4[\mathrm{mA}] + \mathrm{I_{LOAD}})} \tag{4.1}$$

$$R_{5MIN} = \frac{VUMAX - 5[\mathrm{V}]}{0.95 \times 50[\mathrm{mA}]} \tag{4.2}$$

により与えられる．$VUMIN, VUMAX$ はそれぞれ電源電圧の最小値と最大値，また，I_{LAOD} はマイコンの最大出力電流である．$VUMIN = 5.9$ [V], $VUMAX = 6.3$ [V], また，

$$I_{LOAD} = 10[\mathrm{mA}] \tag{4.3}$$

を仮定すると，

$$R_{5MAX} = \frac{5.9 - 5[\mathrm{V}]}{1.05 \times (4[\mathrm{mA}] + 10[\mathrm{mA}])} = 61[\Omega] \tag{4.4}$$

$$R_{5MIN} = \frac{6.3 - 5[\mathrm{V}]}{0.95 \times 50[\mathrm{mA}]} = 27[\Omega] \tag{4.5}$$

となる．そこで，図 4.4 では $R_5 = 50[\Omega]$ としている．

コンデンサ C_6 は，アプリケーションノート AN1035 に上限値の求め方が記載されている．AN1035 はマイクロチップ社の Web ページからダウンロードできる．これによると C_{6MAX} は

$$\begin{aligned} C_{6MAX} &= -\frac{42[\mathrm{ms}]}{R_5 \times ln\left(\frac{2.1}{5}\right)} \\ &= -\frac{42 \times 10^{-3}}{50 \times ln\left(\frac{2.1}{5}\right)} \\ &= 968[\mu\mathrm{F}] \end{aligned} \tag{4.6}$$

と求められる．図 4.4 では $C_6 = 1[\mu\mathrm{F}]$ としている．

4.4 タイマ 1 による割込プログラム

4.4.1 実験回路

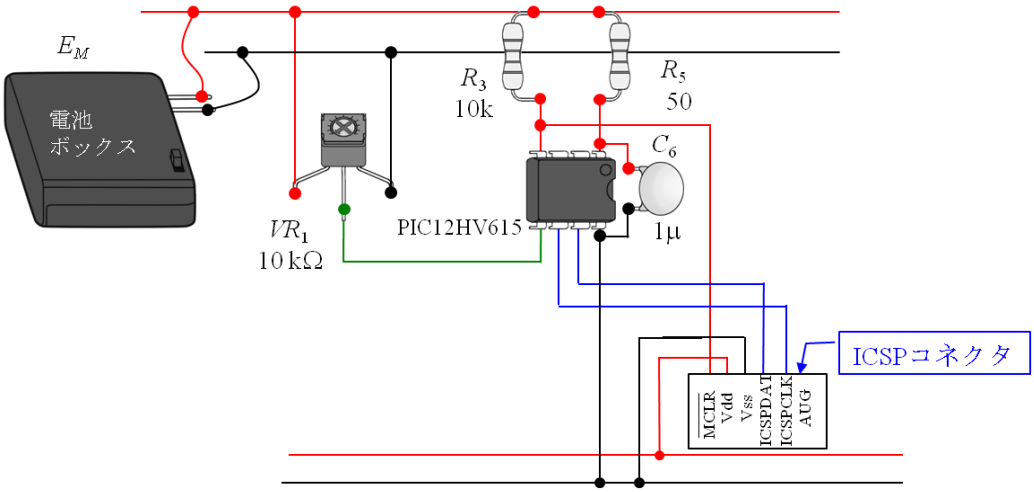


図 4.13: タイマ 1，PWM モジュール，A/D コンバータモジュール実験回路の立体配線図

回転数制御に必要な PIC マイコン内のモジュールはタイマ 1，A/D(Analog/Digital) コンバータモジュールと PWM モジュールである．本節ではタイマ 1 モジュールとその利用法ついて解説する．

図 4.13 はブレッドボード上に製作した実験回路の立体配線図，図 4.14 は回路図である．PIC12HV615 を用いて，内蔵のタイマ 1 モジュール，A/D コンバータモジュールと 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 (PIC12HV615)，可

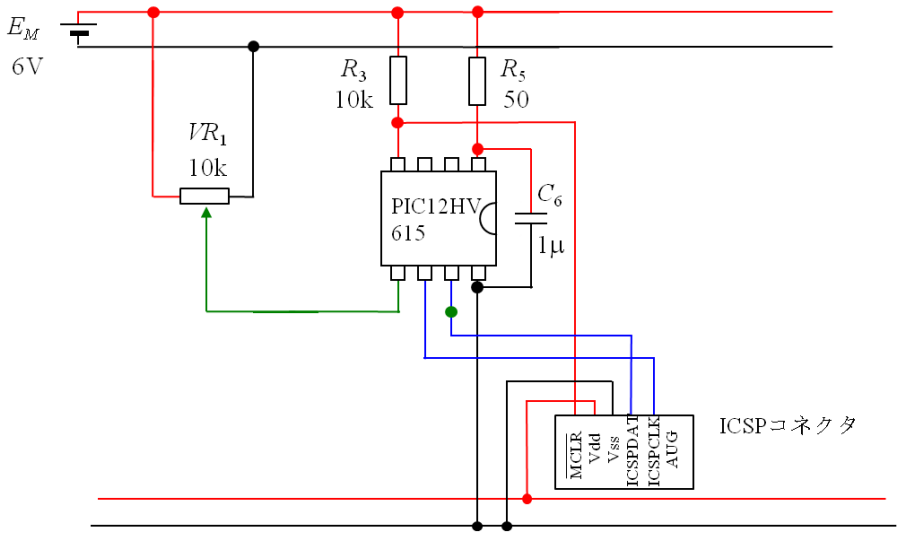


図 4.14: タイマ 1，PWM モジュール，A/D コンバータモジュール実験回路図

変抵抗器 VR_1，ICSP コネクタとブレッドボードである．可変抵抗器はマイコンの A/D コンバータ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用であり，本章ではこのコネクタに PICkit 3, PICkit 4 を接続して，プログラムの書き込みを行う．電源には充電池 4 本を使用して，約 6 [V] の電圧を得ている．

4.4.2 ヘッダファイルのインクルードとコンフィギュレーション

データシート (PIC12F615/HV615_Data_Sheet) の Timer0, 1, 2 Modules の説明によると，PIC12HV615 には 8 ビットのタイマモジュール (Timer0, 2) と 16 ビットのタイマモジュール (Timer1) がある．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 4.15〜図 4.20 にコンフィギュレーション設定，タイマ 1 による割り込み処理関数，メインプログラム，タイマ 1 のヘッダファイル，タイマ 1 の設定関数を示す．マゼンタ色の図説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 ページの PIC12HV615_PIC12F615 用 DC モータの回転数制御プログラム[2]に圧縮フォルダを掲載してあるので，ダウンロードして試みられたい．

PIC マイコンのプログラムにおいて最初に行うのが，図 4.15 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 PIC12HV615 であるので，

$$\#\text{include} < \text{pic12hv615.h} > \tag{4.7}$$

[2]これらのプログラム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る．しかし，利用するに当たっては，読者の責任で行ってください．

```
#include      <xc.h>
#include      "..¥PIC12HV615_Files¥pic12HV615_s.h"

#pragma config FOSC = INTOSCIO      // 内蔵オシレータ オン，GP4, 5を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 IOSCFS = 8MHZ        // 内蔵オシレータの8MHz出力を使用
#pragma config BOREN = OFF          // 低電圧リセット オフ
```

図 4.15: タイマ 1 による割り込みプログラム（コンフィギュレーション設定）PIC12HV615_PIC12F615 用 DC モータの回転数制御プログラム（￥Timer1_Interrupt.X ￥main.c)

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必要がある．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￥Microchip￥xc8￥v2.31￥pic￥include フォルダの中に自動的に保存される．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，MPLAB® XC8 C Compiler User's 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

xc.h (4.8)

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．

本節で新たに導入した設定関数 (図 4.18) の引き数の定義をヘッダファイル (図 4.20) に記してある．このヘッダファイルは図 4.15 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．

#include"..\PIC12HV615_Files\pic12HV615_s.h" (4.9)

上式中の pic12HV615_s.h は筆者が作成したヘッダファイルをである．筆者が作成したヘッダファイルと関数は全て PIC12HV615_PIC12F615 用 DC モータの回転数制御プログラムに入れておくので，ダウンロードして利用されたい．

ヘッダファイルのインクルードの後に必要な設定がコンフィギュレーションである．この設定の説明はデータシートの CONFIGURATION WORD REGISTER にある．多く

の設定項目がある中で，本節では

FOSC_INTOSCIO : 内蔵オシレータを使用する．GP4, 5(3, 2 番ピン) は入出力ピンに設定する．

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

IOSCFS_8MHz : 内蔵オシレータの 8MHz 出力を使用する

のみ設定している．実験室レベルでは，これで十分である．

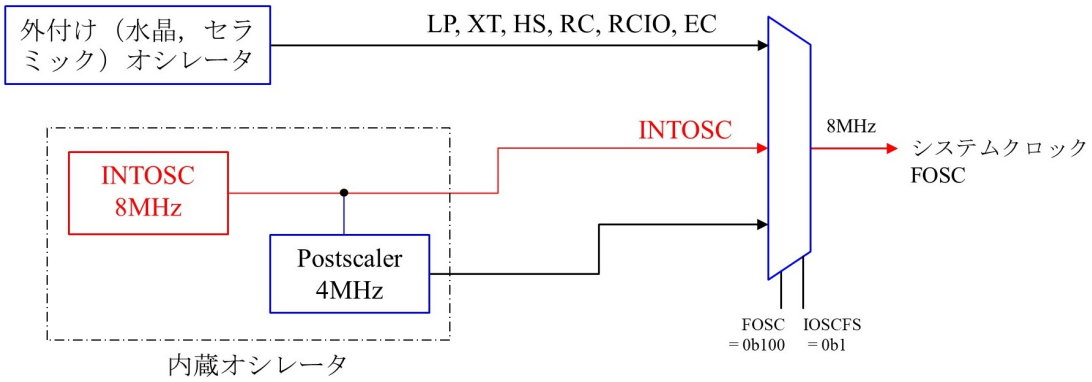


図 4.16: クロックのブロック図

図 4.16 は PIC12HV615 のクロック源のブロック図を示す．上記のコンフィギュレーションにより内蔵オシレータの 8MHz 出力がシステムクロック FOSC となる．

4.4.3 タイマ 1 割り込み処理関数

図 4.17 はタイマ 1 による割り込み処理関数である．図 1.35 と異なる箇所を朱書きで示す．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．

前項で述べたように，ヘッダファイル pic12hv615.h をインクルードすることにより，データシート (PIC12HV615 Data Sheet) の用語を用いて，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．

$$\text{GP5} = 1 \tag{4.10}$$

は，データシートの GPIO REGISTER より，GPIO レジスタの GP5 ビットに 1 を書き込んでいる．これは GP5(PIC12HV615 の 2 番ピン) に 1 を出力する命令である．この関

```
static void __interrupt        // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    GP5 = 1;                   // I/OポートのGP5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(0x1EB0);
                               // 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

    GP5 = 0;                   // I/OポートのGP5に0を出力する. 割り込み発生のモニタリング用
}
```

図 4.17: タイマ 1 による割り込みプログラム（タイマ 1 割り込み処理関数）PIC12HV615_PIC12F615 用 DC モータの回転数制御プログラム (￥Timer1_Interrupt.X ￥main.c)

数の最後では

$$\mathrm{GP5} = 0 \tag{4.11}$$

により，GP5 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，この関数の処理に要する時間を計測することができる．

図 4.18 はタイマ 1 の設定関数である．図 4.17 中の

$$\mathrm{set\_timer1\_count\_down\_ini\_num(0x1EB0);} \tag{4.12}$$

は，TMR1(Timer1 Register) に

$$\mathrm{0xFFFF} - \mathrm{0x1EB0} + 1 \tag{4.13}$$

の値を書き込む関数である．式 (4.12) では，分かりやすくするためにカウントダウン値を設定し，set_timer1_count_down_ini_num 関数内にてカウントアップの初期値に換算している．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 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 となることが分かる．タイマ 1 の値は割り込みをかけた瞬間には 0 となっている．再設定をしないと，タイマ 1 は 0 を初期値としてカウントアップを再開する．後述するようにメイン関数内にてタイマ 1 のクロックを 8MHz に設定しているので，

$$\mathrm{0x1F40}_{(16)} = 8000_{(10)} \tag{4.14}$$

```
#include      <xc.h>
#include      "pic12HV615_s.h"

// T1CON(Timer1 Control Register)の設定（ヘッダファイルpic12f615_s.h参照）
void set_timer1(unsigned int a,unsigned int b, unsigned int c, unsigned int d)
{
      T1CONbits.TMR1CS = a;
      T1CONbits.T1CKPS = b;
      T1CONbits.TMR1ON = c;
      CMCON1bits.T1ACS = d;
}

// タイマ1で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換算を行っている．
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 4.18: タイマ 1 の設定関数 (￥PIC12HV615_Files￥set_timer1.c)

より，カウントダウンの初期値を 0x1F40 とすればよい．しかし，実際には 0x1F40 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 0x1EB0 とし 0 までのカウントダウン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．0x1EB0 の値は試行しながら探した．

図 4.17 中の

```
clear_interrupt_flag_of_timer1();
```
(4.15)

は，図 4.18 に定義しているように，PIR1 REGISTER の TMR1IF ビットに 0 を書き込むことで，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．

4.4.4 メイン関数

```
void main()
{
    TRISIO = 0x00;                  // ポートGPIOを出力ポートに設定する.

    //  タイマー1の設定
    set_timer1(TMR1_clock_source_FOSC, T1Clock_PreScale_1_1, TMR1_ON, TMR1_Alternate_Clock_FOSC);
                                    // 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 4.19: タイマ 1 による割り込みプログラム（メイン関数）(￥Timer1_Interrupt.X￥main.c)

図 4.19 はメイン関数である．レジスタ，タイマの初期設定を行っている．図 1.36 と異なる箇所を朱書きで示す．

$$\text{TRISIO} = 0\text{x}00; \tag{4.16}$$

はデータシートの GPIO TRI-STATE REGISTER より TRISIO レジスタに 0x00 を書き込んでいる．これにより I/O PORT を出力ポートに設定している．

$$\text{set\_timer1}(\text{TMR1\_clock\_source\_FOSC}, \text{T1Clock\_PreScale\_1\_1}, \text{TMR1\_ON}, \text{TMR1\_Alternate\_Clock\_FOSC}); \tag{4.17}$$

の定義も図 4.18 にある．図 1.43 の set_timer1 関数と異なる箇所は CMCON1 レジスタへの書き込みである．この関数は，以下のように T1CON(TIMER1 CONTROL REGISTER) レジスタおよび CMCON1(COMPARATOR CONTROL REGISTER 1) 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．

$$\text{T1CONbits.TMR1CS} = 0\text{b}0; \tag{4.18}$$

$$\text{T1CONbits.T1CKPS} = 0\text{b}00; \tag{4.19}$$

$$\text{T1CONbits.TMR1ON} = 0\text{b}1; \tag{4.20}$$

$$\text{CMCON1bits.T1ACS} = 0\text{b}1; \tag{4.21}$$

各式右辺の具体的な値は，式 (4.17) の関数内の引数で決められている．これら引数は略語である．

```
pic12HV615_s.hヘッダファイル

// 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_T1CkI     0b1       // 外部クロック入力(T1CKIより)を利用する.
    #define TMR1_clock_source_FOSC      0b0       // システムクロック(FOSC or 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. )

// CMCON1(Comparator Gating Timer1 Register)の用語の設定

    #define TMR1_Alternate_Clock_FOSC           0b1   //タイマ1のクロックをFOSCとする.
    #define TMR1_Alternate_Clock_FOSC_1_4       0b0   //タイマ1のクロックをFOSC/4とする.

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

図 4.20: タイマ 1 ヘッダファイル _T1CON, CMCON1(￥PIC12HV615_Files￥pic12HV615_s.h)

略語と具体的数値との関係は図 4.20 のヘッダファイルで定義している．これらの略語は，データシートの T1CON レジスタと CMCON1 レジスタの説明をもとに筆者が独自に（勝手に）定義したものである．図 1.40 と異なる箇所を朱書きで示す．

図 4.21 はタイマ 1 のブロック図である．データシートによると，COMPARATOR CONTROL REGISTER 1 の T1ACS(Timer1 Alternate Clock Select bit) に 0b1 を書き込むことで，タイマ 1 にはシステムクロック FOSC = 8MHz が供給される．TIMER1 CONTROL REGISTER の TMR1CS(Timer1 Clock Source Select bits) に 0b0 を書き込むことで，タイマ 1 のクロックソースをシステムクロック (FOSC = 8MHz) に設定する．なお，データシートでは FOSC の用語は CONFIGURATION WORD REGISTER 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 に設定する．これにより，タ

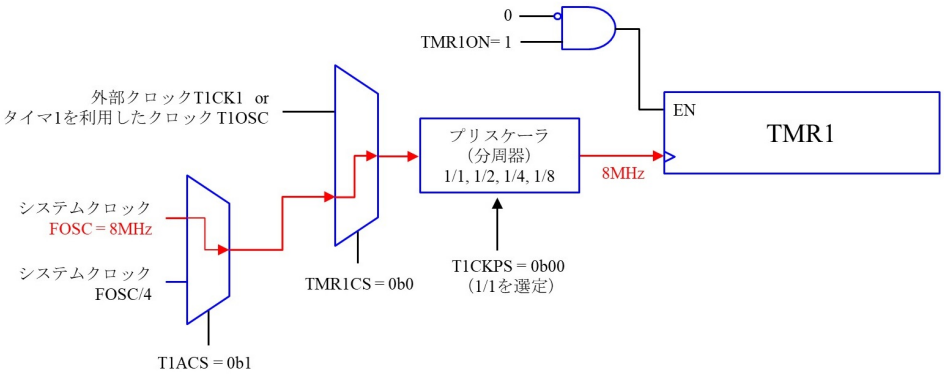


図 4.21: タイマ 1 のブロック図

イマ 1 はシステムクロックを分周しないで 8MHz のまま用いる．TMR1ON はタイマ 1 を起動する．

図 4.19 の

$$\text{set\_interrupt\_by\_timer1}(); \tag{4.22}$$

は，タイマ 1 による割り込みを設定する関数である．この関数の定義は図 4.18 にある．この関数は，以下のように PIE1(PERIPHERAL INTERRUPT ENABLE REGISTER 1) レジスタおよび INTCON1(INTERRUPT CONTROL REGISTER) レジスタへの書き込みを行い，タイマ 1 による割り込み設定を行っている．

$$\text{PIE1bits.TMR1IE} = 1; \tag{4.23}$$

$$\text{INTCONbits.PEIE} = 1; \tag{4.24}$$

$$\text{INTCONbits.GIE} = 1; \tag{4.25}$$

データシートの PERIPHERAL INTERRUPT ENABLE REGISTER 1 の 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 の PEIE (Peripheral Interrupt Enable bit) は，タイマ，AD コンバータ，PWM 制御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，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．なお，SUMMARY OF REGISTERS ASSOCIATED WITH TIMER1 より，TMR1IF ビットは電源投入時には 0 に初期設定されているので，メインプログラム内でこのビットを 0 に設定する必要はない．

図 4.19 のメイン関数の最後は

```
for(;;)
    continue;
```
(4.26)

の無限ループである．メインプログラムは，マイコンの電源をオフにするまで，このループを実行し続ける．この間，タイマ 1 による割り込みにより，タイマ 1 割り込み関数が 1 [ms] の周期で起動される．

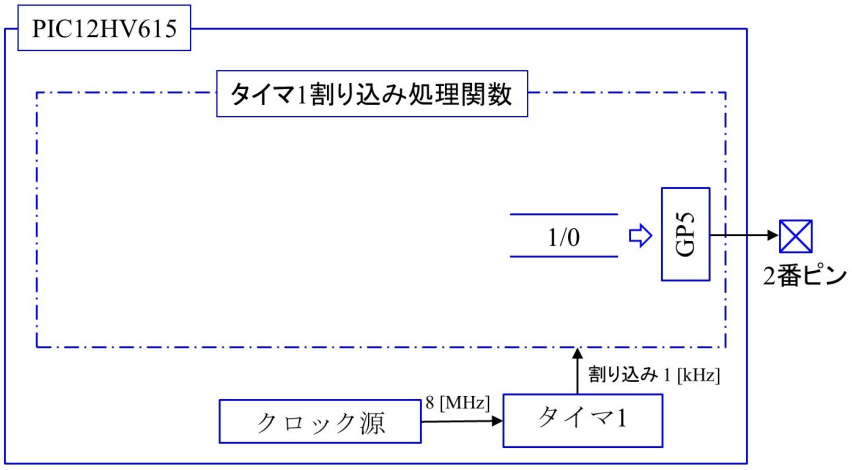


図 4.22: タイマ 1 による割り込みプログラムのブロック図

図 4.22 は図 4.15，4.17，4.19 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 8 [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 割り込み処理関数を起動する．この関数は処理開始時に 2 番ピンに 1 を出力し，処理終了時に 0 を出力する．

4.4.5 実験結果

図 4.23 は，図 4.15〜4.19 のプログラムを PIC12HV615 に書き込み・実行させたときの，GP5(2 番ピン) の電圧波形の計測結果である．横軸は 5 [μs/div] なので，一目盛りが 5 [μs] 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 1.0035 [kHz] であった．また，タイマ 1 割り込み関数の処理時間は約 14 [μs] であった．

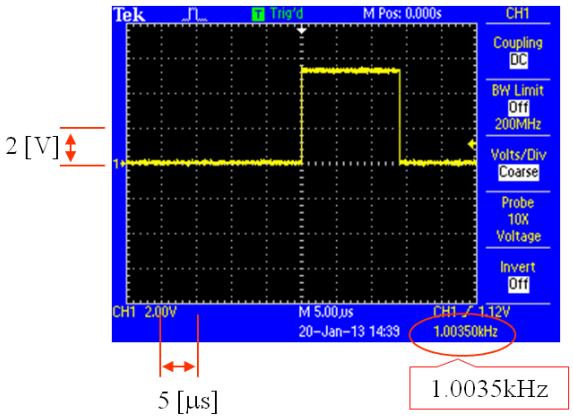


図 4.23: タイマ 1 による割り込みの実験波形

4.5 PWM モジュール

4.5.1 プロジェクトの作成とプログラムのブロック図

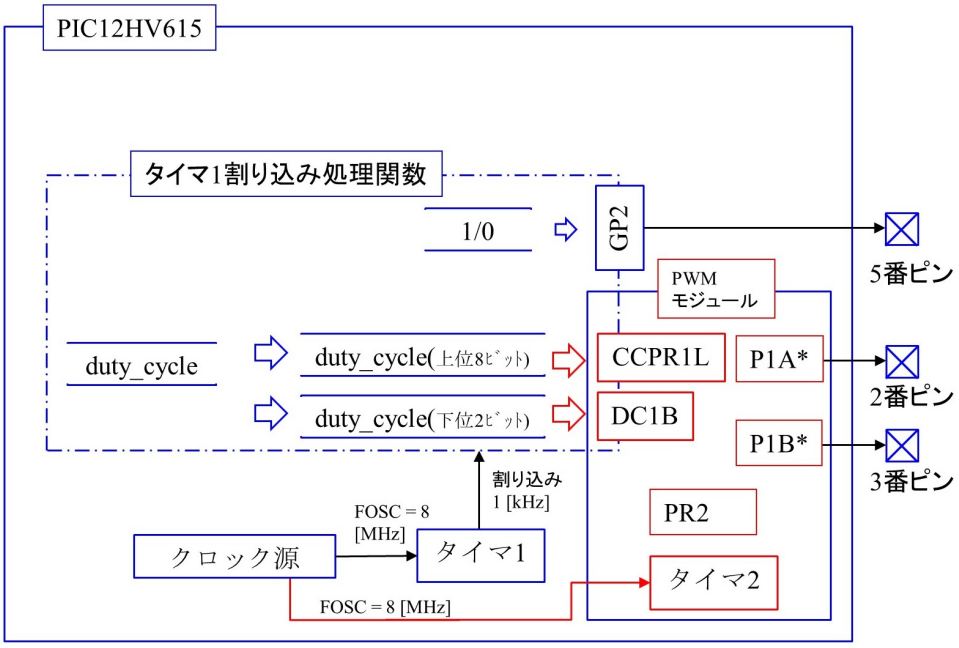


図 4.24: PWM 制御プログラムのブロック図

第 1 章ではタイマ 1 の割り込みプログラムの次に A/D コンバータモジュールを使用するプログラムを解説した．PIC16F1825 の場合は A/D 変換結果を PICKit 3, PICkit 4 のデバッグ機能を利用してパソコン上で読み取ることができた．しかし，PIC12HV615 のデータシートの In-Circuit Debugger の節によると，デバッグには Special debugging adapter が必要であると記されている．MPLAB IDE の debugger を起動すると A debug header is required のメッセージが出てくる．無視してデバッグを実行しようとするとエラーが出る．AC162083 という Debug header が要る．本稿ではデバッガの使用は諦めて先に進むことにする．そこで，A/D コンバータモジュールより先に PWM モジュールを使用するプログラムを解説する．

新しいプロジェクト「PWM」を作成してください．そして，新たに作られた C:￥Users￥(ユーザー名)￥MPLABXProjects￥12HV615￥PWM.X フォルダへ，￥12HV615_モータドライブプログラム￥PWM.X フォルダから main.c ファイルをコピーしてください．次に，Source Files に main.c を “add” し，Linker Files に図 4.9 でコピーした C:￥Users￥(ユーザー名)￥MPLABXProjects￥12HV615￥PIC12HV615_Files フォルダから

set_pwm.c，set_timer1.c，set_timer2.c を “add” してください．

図 4.24 は PWM 制御プログラムのブロック図を示す．変数 duty_cycle の値により 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 が割り当てられている．データシート (TIMER2 MODULE) よりタイマ 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 = 2 [MHz] により駆動されると記されている．しかし，PWM 制御モードではタイマ 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 = 8 [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．

4.5.2 PWM モジュールのブロック図

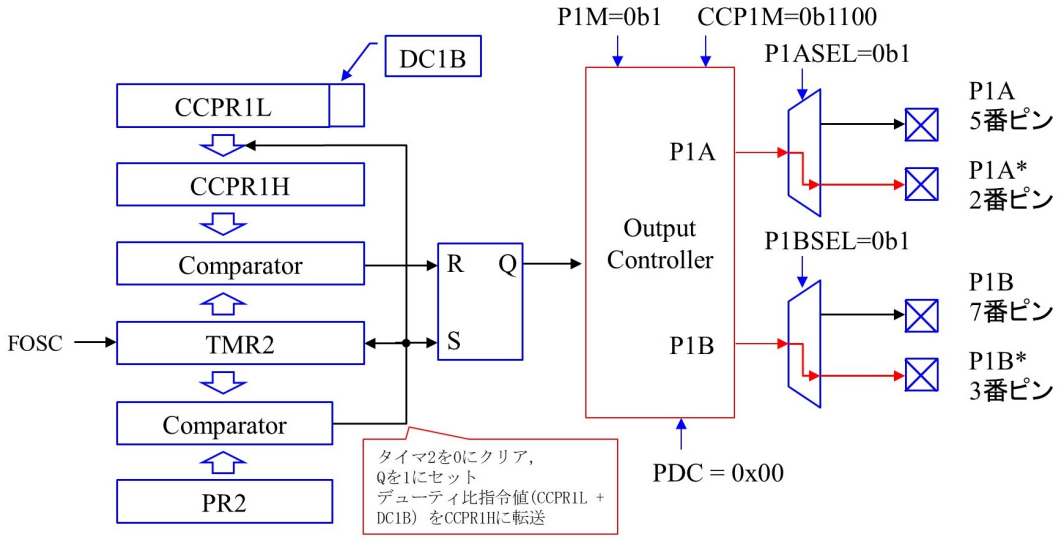


図 4.25: PWM モジュールのブロック図

図 4.25 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 は常にシステム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 ($Q = 1, \overline{Q} = 0$) し，また，CCPR1H (Capture/Compare/PWM Register 1 High Byte) レジスタ (10 ビットレジスタ) の上位 8 ビットに CCPR1L レジスタ (8 ビットレジスタ) の値を転送し，下位 2 ビットに CCP1CON (CCP1 CONTROL REGISTER) レジスタの DC1B (2 ビット) の値を転送する．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 CCPR1H 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, $\overline{\mathrm{Q}} = 1$) する．

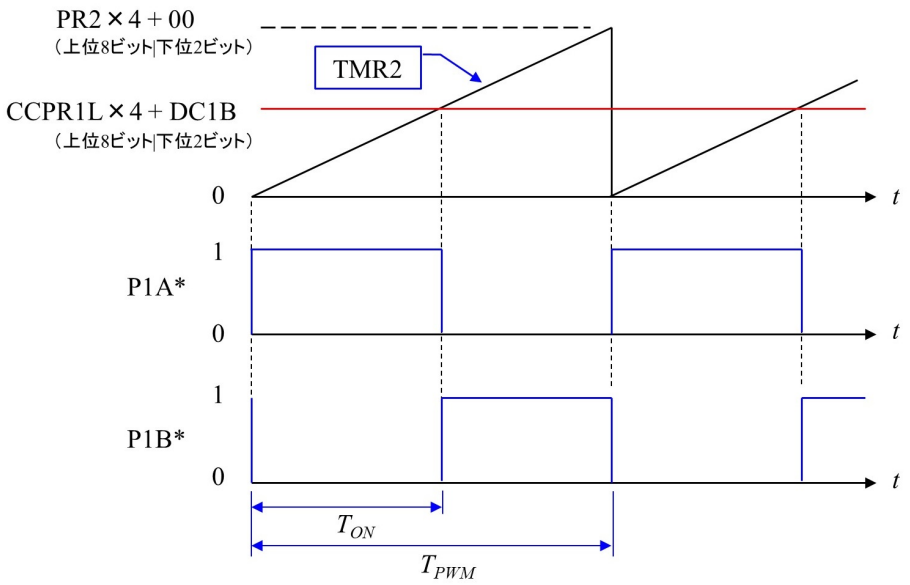


図 4.26: PWM モジュールの動作

以上の動作のイメージを図 4.26 に示す．TMR2 レジスタの値が 0 に初期化されたとき，PWM 制御出力である P1A* = Q = 1, P1B* = $\overline{\mathrm{Q}}$ = 0 とセットされる．TMR2 の値は FOSC のカウントアップにより，時間に比例して増加する．

$$\mathrm{TMR2} == \mathrm{CCPR1L} \times 4 + \mathrm{DC1B} \tag{4.27}$$

となったとき，P1A* = Q = 0, P1B* = $\overline{\mathrm{Q}}$ = 1 にリセットされる．

$$\mathrm{TMR2} == \mathrm{PR2} \times 4 \tag{4.28}$$

となったとき，TMR2 = 0，P1A* = 1, P1B* = 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，P1A*, P1B*の値の平均値は CCPR1L × 4 + DC1B の値に比例する．このように P1A*, P1B*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，P1A* = 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 = 8 [MHz] のとき，PR2 = 0x3F とすると，PWM 周期 T_{PWM} は

$$T_{PWM} = \frac{1}{\frac{\text{FOSC}}{(\text{PR2}+1)\times 4}} = 32[\mu\text{s}] \tag{4.29}$$

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 $1/((\text{PR2}+1)\times 4) = 1/256$ である．

4.5.3 タイマ 1 割り込み処理関数と PWM 制御

```
//PWM プログラム

#define PWM_period   0x3F      // PWM周期の決定 ( = FOSC/4/(PWM_period+1) = 8MHz/4/128 = 31.25 kHz

unsigned int   duty_cycle;

static void interrupt                  // Timer1 による割り込み処理関数
timer1_isr(void)
{
      GP2 = 1;                         // GP2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(0x1EB0);
                                       // 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

      duty_cycle = 0x1FF;

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

      GP2 = 0;                         // GP2に0を出力する. 割り込み発生のモニタリング用
}
```

図 4.27: PWM 制御のプログラム _ 割り込み処理関数 (￥PWM.X￥main.c)

図 4.27 は PWM 制御プログラムの割り込み処理関数を示す．図 1.67 と異なる箇所を朱書きで示す．なお，ヘッダファイルのインクルードと CONFIG については図 4.15 と同じである．

set_PWM_duty_cycle 関数を図 4.28 に示す．PWM 制御のデューティ比 δ は duty_cycle により設定している．図 4.25 で述べたように，CCPR1L レジスタは 8 ビット，DC1B は 2 ビットである．PIC12HV615 は 10 ビット A/D コンバータを持つ．A/D コンバータモジュールの節で述べるように，A/D 変換結果を CCPR1L+DC1B レジスタに格納してデューティ比を決定する．10 ビットの AD 変換結果は 16 ビットレジスタ ADRES(ADC RESULT REGISTER) レジスタに右寄せで格納する予定である．

```
set_pwm.c (PWM設定関数ファイル）

#include     <xc.h>
#include     "pic12HV615_s.h"

// CCP1CON(CCP1 Control Register)の設定
// APFCON(Alternate Pin Function Control Register)の設定
void set_PWM(unsigned int a, unsigned int b, unsigned int c, unsigned int d)
{
      //PWMモジュールの設定
      CCP1CONbits.P1M = a;        // Half Bridge, P1A, P1Bを利用 or PWM with P1A only
      CCP1CONbits.CCP1M = b;    // PWMモード  P1A, P1B active high/low

      // PWM出力ピンの割り当て
      APFCONbits.P1ASEL = c;      // P1AをGP5 or GP2に割り当てる
      APFCONbits.P1BSEL = d;      // P1BをGP4 or GP0に割り当てる.
}

// PWMのデューティ比の設定
void set_PWM_duty_cycle(signed int a)
{
      CCP1CONbits.DCB = (a >>2) & 0b00000011; // PWMデューティ比の下位2ビット指定
      CCPR1L = a >> 4;                                  // PWMデューティ比の上位8ビット指定
}

// PWMのデッドタイムの設定
void set_PWM_dead_time(unsigned int a)
{
      PWM1CONbits.PDC = a;        // PWMのデッドタイムの指定
}
```

図 4.28: PWM モジュールの設定関数 (￥PIC12HV615_Files￥set_pwm.c)

PWM 周期を

$$\#\text{define PWM\_period } 0\text{x3F} \tag{4.30}$$

と設定すると，これが PR2 レジスタに格納され，TMR2 の上位 8 ビットと比較される．すなわち図 4.25 の下側の Comparator は TMR2 の値が 0b0011111100 となったときに TMR2 を 0 にクリアし，Q を 1 にセットする．すなわち，デューティ比を決める CCPR1L+DC1B レジスタにとって 0b0011111100 より大きな値は無効である．従って，A/D 変換結果は，ADRES レジスタの値を 2 ビット右シフトさせて 0b0011111111 以下の値にして，図 4.25 の上側の Comparator にて比較させるようにすればよい．0b0011111101 ～ 0b0011111111 の値は無効となるがわすかなのでよしとする．そこで，duty_cycle に A/D 変換結果が格納されることを前提に，set_PWM_duty_cycle 関数では

$$\text{CCP1CONbits.DCB} = (\text{a} >> 2)\ \&\ 0\text{b00000011;} \tag{4.31}$$

により，duty_ cycle の下から 2, 3 ビット目を DCB に格納している．なお，DCB は図 4.25 の DC1B のことである．pic12hv615.h のヘッダファイルで DC1B が DCB と記述さ

れているため，多少困惑しつつもそのまま採用している．また，

$$\text{CCPR1L} = \text{duty\_cycle} >> 4; \tag{4.32}$$

により，duty_cycle を右に 4 ビットシフトさせることで，4〜9 ビット目の値を CCP1RL に格納している．これにより，10 ビットの A/D 変換結果の上位 8 ビットによりデューティ比を決定する．

4.5.4 メイン関数

```
void main()
{
	TRISIO = 0x00;			// ポートGPIOを出力ポートに設定する.

	//  タイマー1の設定
	set_timer1(TMR1_clock_source_FOSC, T1Clock_PreScale_1_1, TMR1_ON, TMR1_Alternate_Clock_FOSC);
					// 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. .

	// PWMモジュールの設定
	set_PWM(Half_Bridge_with_P1A_P1B, P1A_B_active_high, P1A_to_GP5, P1B_to_GP4);

	// 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 4.29: PWM 制御のプログラム _ メイン関数 (￥PWM.X￥main.c)

図 4.29 は PWM 制御のメイン関数を示す．図 1.68 のメイン関数と異なる箇所を朱書きで示す．PWM モジュールの設定は

set_PWM(Half_Bridge_with_P1A_P1B, P1A_B_active_high, P1A_to_GP5, P1B_to_GP4);

により行っている．この関数の定義は図 4.28 にある．図 1.69 との違いを朱書きで示してある．引数の定義を図 4.30 に示す．

PWM モジュールの設定は，図 4.28 の CCP1CON レジスタ (CCP1 CONTROL REGISTER) により行う．CCP は Capture, Compare, PWM 制御のイニシャルである．P1A と P1B を利用するためには CCP1FCON レジスタの設定を必要とする．

$$\text{CCP1CONbits.P1M} = 0\text{b1}; \tag{4.33}$$

```
pic12HV615_s.h（ヘッダファイル）

// CCP1CON(CCP1 Control Register)の設定

    // P1M (PWM Output Configuration bits)
    // CCP1M = 11xxのとき
    #define PWM_with_P1A_only          0b0     // P1AのみPWM出力, P1Bはポートピン
    #define Half_Bridge_with_P1A_P1B   0b1     // ハーフブリッジPWM, P1A, P1BにPWM出力

    // CCP1M (Enhanced CCP1 Mode Select bits)
    #define P1A_B_active_high          0b1100  // PWMモード, P1A, B が正論理
    #define P1A_act_high_P1B_act_low   0b1101  //          P1A が正論理, P1Bが負論理
    #define P1A_act_low_P1B_act_high   0b1110  //          P1A が負論理, P2Bが正論理
    #define P1A_act_low_P1B_act_low    0b1111  // PWMモード, P1A, B が負論理


// APFCON(Alternate Pin Function Control Register)の設定

    // P1ASEL (P1A pin selection bit)
    #define P1A_to_GP5      0b1  // P1AをGP5に割り当てる
    #define P1A_to_GP2      0b0  // P1BをGP2に割り当てる

    // P1BSEL (P1B pin selection bit)
    #define P1B_to_GP4      0b1  // P1BをGP4に割り当てる
    #define P1B_to_GP0      0b0  // P1BをGP0に割り当てる
```

図 4.30: PWM モジュール用のヘッダファイル (￥PIC12HV615_Files￥pic12HV615_s.h)

により，P1M(PWM Output Configuration bit) に 0b1 を書き込むことで，P1A, P1B を利用する設定としている．また，

$$\text{CCP1CONbits.CCP1M} = 0\text{b1100}; \tag{4.34}$$

により，P1A, P1B を正論理 (active high) に設定している．これを

$$\text{CCP1CONbits.CCP1M} = 0\text{b1111}; \tag{4.35}$$

として，P1A, P1B ともに負論理 (active low) に設定すると，図 4.26 において P1A, P1B の値は 1, 0 が反転する．

P1A, 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．P1A は 2 番ピンか 5 番ピン, P1B は 3 番ピンか 7 番ピンのいずれかを選択できる（図 4.5 参照）．この選択は APFCON(ALTERNATE PIN FUNCTION CONTROL REGISTER) レジスタにより行う．

$$\text{APFCONbits.P1ASEL} = 0\text{b1}; \tag{4.36}$$

により，P1A を GP5(2 番ピン) に割り当て，

$$\text{APFCONbits.P1BSEL} = 0\text{b1}; \tag{4.37}$$

により，P1B を GP4(3 番ピン) に割り当てている．

```
// PWM1CON (Enhanced PWM Control Register)の設定

// PDC (PWM Delay Count bits)
#define _0ns     0b0000000  // デッドタイム    0 [ns]
#define _500ns  0b0000001  //              500 [ns]
#define _1us    0b0000010  //             1000 [ns]
```

図 4.31: デッドタイム設定用ヘッダファイル (￥PIC12HV615_Files￥pic12HV615_s.h)

なお，図 4.28 の set_PWM_dead_time 関数により，図 3.20 と同様にデッドタイムを設けることができる．引数の定義を 4.31 に示す．FUNCTTION REGISTER SUMMARY BANK 0 によると，デッドタイム設定用の PWM1CON レジスタの電源投入時の値は 0 であるので，本章では設定を省略している．

4.5.5 タイマ 2 の設定

```
// 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.TOUTPS = a;        // タイマ2出力の分周率を1:1に設定（分周しない）
    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1:1に分周する.
    PR2 = d;                     // PWM周波数設定  FOSC/4/256 = 8[MHz]/4/256 = 7.8[kHz]
}
```

図 4.32: タイマ 2 の設定関数 (￥PIC12HV615_Files￥set_timer2.c)

PWM 制御用のタイマにはタイマ 2 が割り当てられている．図 4.29 の set_timer2 関数はタイマ 2 の設定をお行う．set_timer2 関数の定義を図 4.32 に示す．この関数は図 1.70 の関数と異なるところはない．引数の定義を 4.33 に示す．

タイマ 1 の設定との大きな違いは，タイマ 1 ではクロック源を選択できるのに対して，タイマ 2 の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．図 4.25 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，PR2(Timer2

```
// T2CON (Timer2 Control Register)の設定

    // TOUTPS (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
```

図 4.33: タイマ 2 用ヘッダファイル (￥PIC12HV615_Files￥pic12HV615_s.h)

Module Period Register) を

$$\mathrm{PR2} = \mathrm{0x3F} \tag{4.38}$$

とすると，PWM 周波数 f_{PWM} は

$$f_{PWM} = \mathrm{FOSC}/4/(63+1) = 8[\mathrm{MHz}]/4/64 = 31.25[\mathrm{kHz}] \tag{4.39}$$

と求められる．逆に，31.25 [kHz] の f_{PWM} を得るためには，

$$\mathrm{PR2} = 31.25[\mathrm{kHz}]/(\mathrm{FOSC}/4) - 1 = 31.25[\mathrm{kHz}]/8[\mathrm{MHz}]/4 - 1 = 63 \tag{4.40}$$

と設定すればよい．

4.5.6 実験結果

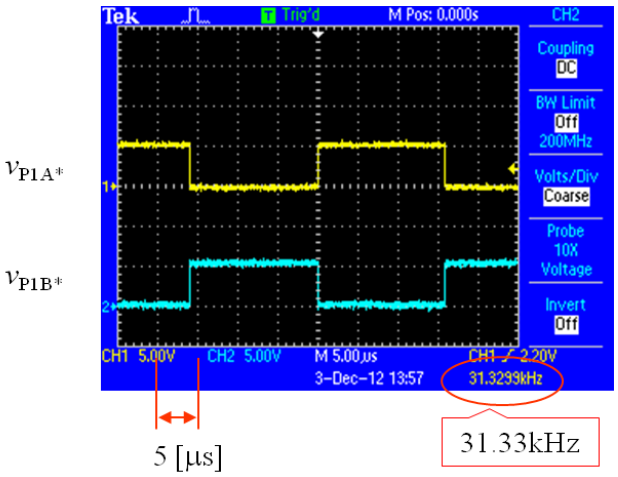


図 4.34: PWM モジュール実行時の入出力波形

図 4.34 は，図 4.14 の実験回路に PICkit 3 もしくは PICkit 4 を接続し，MPLAB により以上のプログラムを PIC12HV615 に書き込んでマイコンにプログラム実行をさせたときの，2, 3 番ピンの PWM 制御出力電圧 v_{P1A*}, v_{P1B*} の実験波形例を示す．duty_cycle = 0x1FF と設定したことで，デューティ比がほぼ 0.5 の PWM 波形が得られた．また，PWM 周波数 $f_{PWM} \approx 31.33$ [kHz] であることが分かる．

4.6 A/D コンバータモジュール

4.6.1 プロジェクトの作成とプログラムのブロック図

新しいプロジェクト「AD_Conv」を作成してください．そして，新たに作られた C:￥Users￥(ユーザー名)￥MPLABXProjects￥12HV615￥AD_Conv.X フォルダへ，￥12HV615_モータドライブプログラム￥AD_Conv.X フォルダから main.c ファイルをコピーしてください．次に，Source Files に main.c を “add” し，Linker Files に図 4.9 でコピーした C:￥Users￥(ユーザー名)￥MPLABXProjects￥12HV615￥PIC12HV615_Files フォルダから set_ad_converter.c，set_pwm.c，set_timer1.c，set_timer2.c を “add” してください．

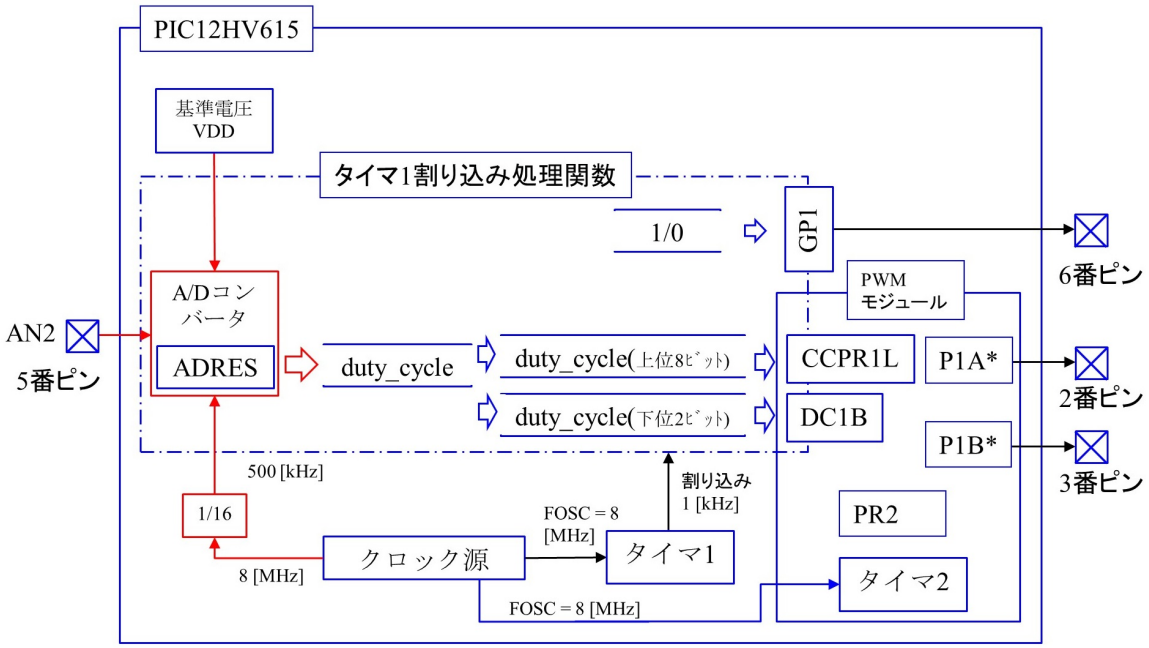


図 4.35: A/D コンバータプログラムのブロック図

図 4.35 は，図 4.24 の PWM 制御プログラムのブロック図に A/D(Analog/Digital) コンバータモジュールを追加したブロック図を示す．5 番ピンからデューティ比の指令値を入力して，P1A*, P1B*から出力される PWM 波形のデューティ比を制御する．図 4.14 の実験回路を用いる．この回路では可変抵抗 VR_1 より PIC12HV615 の 5 番ピンに可変のアナログ電圧を印加している．図 4.35 において，タイマ 1 により 1 [ms] ごとに割り込みがかかり，割り込み処理関数はその度に A/D コンバータを起動する．A/D コンバータは 5 番ピンからアナログ電圧信号を読み込み，ADRES(ADC RESULT REGISTER) レジスタに変換結果を格納する．割り込み処理関数は A/D 変換終了を受けて，変数 duty_cycle 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 (FOSC = 8 [MHz]) を 16 分周して 500[kHz] 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，システムクロック (FOSC) が 8[MHz] のとき推奨周期が 2 or 4[μs](推奨クロックが 500 or 250[kHz]) であることによる．また，A/D コンバータの基準電圧は電源電圧 VDD としている．

4.6.2 タイマ 1 割り込み処理関数と A/D 変換

```
//ADコンバータプログラム

#define PWM_period      0x3F // PWM周期の決定 ( = FOSC/4/(PWM_period+1) = 8MHz/4/128 = 31.25 kHz

unsigned int    duty_cycle;

static void interrupt               // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    GP1 = 1;                        // GP1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(0x1EB0);
                                    // 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();
    duty_cycle = read_result_of_ad_conversion();

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

    GP1 = 0;                        // GP1に0を出力する. 割り込み発生のモニタリング用
}
```

図 4.36: A/D コンバータのプログラム _ 割り込み処理関数 (￥AD_Conv.X￥main.c)

割り込み処理関数を図 4.36 に示す．図には図 1.53 のプログラムに対して変更した命令を朱書きで示してある．割り込み処理関数では，割り込み発生のモニタリング用の出力を GP1 に割り当てている．

$$\text{start\_ad\_conversion();} \tag{4.41}$$

の関数の定義を図 4.37 に示す．この関数内では

$$\text{ADCON0bits.GO} = 0\text{b1} \tag{4.42}$$

```
// A/D コンバータの設定(p.156)
void set_AD_Converter(unsigned int a, unsigned int b, unsigned int c, unsigned int d, unsigned int e, unsigned int f)
{
	ADCON0bits.ADFM = a;	// 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.
	ADCON0bits.VCFG = b;	// 基準電圧の＋側の設定
	ADCON0bits.CHS  = c;	// アナログチャネル設定
	ADCON0bits.ADON = d;	// A/Dコンバータをオン/オフ設定

	ANSELbits.ADCS  = e;	// A/Dコンバータのクロック設定
	ANSELbits.ANS   = f;	// アナログ入力ピンの設定
}

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 = ADRESH;		// 変換結果の読み出し
	a = (a<<8) + ADRESL;

	return(a);
}
```

図 4.37: A/D コンバータの設定関数 (￥PIC12HV615_Files￥set_ad_converter.c)

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 CONVERSION 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，図 4.38 のメイン関数内の set_AD_Converter() 関数により，TAD = 2[μs] と設定している．よって，ADRES レジスタの値を読み出すには A/D 変換を開始させてから 24 [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，

$$\text{while(ADCON0bits.GO)} \tag{4.43}$$

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 変換終了後には read_result_of_ad_conversion() 関数により，ADRES レジスタの値を duty_cycle に読み出している．なお，図 1.53 では PWM 制御を行っていないので set_PWM_duty_cycle 関数が無い．

4.6.3 メイン関数

```
void main()
{
	TRISIO = 0b00000100;		// GP2を入力ポート. 他を出力ポートに設定する.

	//  タイマー1の設定
	set_timer1(TMR1_clock_source_FOSC, T1Clock_PreScale_1_1, TMR1_ON, TMR1_Alternate_Clock_FOSC);
						// 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コンバータの設定
	set_AD_Converter(right_justified, pos_ref_VDD, select_AN2, AD_ON, clock_1_16, GP2_ANinput);

	// PWMモジュールの設定
	set_PWM(Half_Bridge_with_P1A_P1B, P1A_B_active_high, P1A_to_GP5, P1B_to_GP4);

	// 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 4.38: A/D コンバータのプログラム _ メイン関数 (￥AD_Conv.X￥main.c)

メイン関数を図 4.38 に示す．図には図 1.54 のプログラムに対して変更した命令を朱書きで示してある．メイン関数では，

$$\text{TRISIO} = 0\text{x}04 = 0\text{b}0000\ \ 0100 \tag{4.44}$$

により，5 番ピン (GP2/AN2) を入力ポートに設定している．

A/D コンバータモジュールの設定は

$$\begin{aligned}\text{set\_AD\_Converter}(&\text{right\_justified}, \text{pos\_ref\_VDD}, \text{select\_AN2},\\ &\text{AD\_ON}, \text{clock\_1\_16}, \text{GP2\_ANinput});\end{aligned} \tag{4.45}$$

により行っている．この関数の定義は図 4.37 に示してある．この関数の引数を定義しているヘッダファイルを図 4.39，4.40 に示す．

```
// ADCON0(A/D Control Register 0)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

    // VCFG(Voltage Reference bit)
    #define pos_ref_exVREF   0b1        // A/Dコンバータの基準電圧の+側を外部VREF+ピンとする.
    #define pos_ref_VDD      0b0        //                                 をVDDとする.

    // CHS(Analog Channel Select bits)
    #define select_AN0   0b000          //AN0を選定
    #define select_AN1   0b001          //AN1
    #define select_AN2   0b010          //AN2
    #define select_AN3   0b011          //AN3
    #define select_AN4   0b100          //CVREF
    #define select_AN5   0b101          //0.6V Reference
    #define select_AN6   0b110          //1.2V Reference
    #define select_AN7   0b111          //Do not use

    // 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
```

図 4.39: A/D コンバータのヘッダファイル (￥PIC12HV615_Files￥pic12HV615_s.c)

```
// ANSEL(Analog Select Register)の用語の設定
    // ADCS(A/D Conversion Clock Select bits)
    #define clock_1_2       0b000    // A/Dコンバータのクロック 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.

// ANS(Analog Select Between Analog or Digital Function on Pins GP4, GP2, GP1, GP0, respectively)
    #define GP0_ANinput     0b0001   // GP0をアナログ入力ピンとする.
    #define GP1_ANinput     0b0010   // GP1
    #define GP10_ANinput    0b0011   // GP1,0
    #define GP2_ANinput     0b0100   // GP2
    #define GP20_ANinput    0b0101   // GP2,0
    #define GP21_ANinput    0b0110   // GP2,1
    #define GP210_ANinput   0b0111   // GP2,1,0
    #define GP4_ANinput     0b1000   // GP4
    #define GP40_ANinput    0b1001   // GP4,0
    #define GP41_ANinput    0b1010   // GP4,1
    #define GP410_ANinput   0b1011   // GP4,1,0
    #define GP42_ANinput    0b1100   // GP4,2
    #define GP420_ANinput   0b1101   // GP4,2,0
    #define GP421_ANinput   0b1110   // GP4,2,1
    #define GP4210_ANinput  0b1111   // GP4,2,1,0

//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 start_ad_conversion();
unsigned int read_result_of_ad_conversion();
void start_ad_conversion_ch_select(unsigned int a);
```

図 4.40: A/D コンバータのヘッダファイル（つづき）(￥PIC12HV615_Files￥pic12HV615_s.c)

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 Conversion Result Format Select bit) により，ADRES レジスタの左側に寄せて上位 10 ビットに格納するか，右側に寄せて下位 10 ビットに格納するかを選択できる．right_justified を選択することで，ADFM = 1 とし，右寄せに設定している．10-BIT A/D CONVERSION FORMAT によるとこの設定では ADRES レジスタの上位 6 ビットは 0 となる．

pos_ref_VDD により，VCFG(Voltage Reference bit) = 0 とし，A/D コンバータの基準電圧の+側を VDD に設定している．

select_AN2 により，CHS(Analog Channel Select bits) = 0b010 とし，A/D コンバータの入力を AN2(5 番ピン) に設定している．

AD_ON により，ADON(ADC Enable bit) = 1 とし，A/D コンバータモジュールを起動する．

clock_1_16, GP2_ANinput は ANSEL レジスタ内のビットの設定に関する引数である．

clock_1_16 により，A/D Conversion Clock Select bits) = 0b101 とし，

$$\mathrm{TAD} = \frac{1}{\frac{\mathrm{FOSC}}{16}} = \frac{1}{\frac{8[\mathrm{MHz}]}{16}} = 2[\mu\mathrm{s}] \tag{4.46}$$

と設定している．

GP2_ANinput により，ANS(Analog Select Between Analog or Digital Function on Pins bits) = 0b0100 とし，GP2 をアナログ入力ピンに設定している．

4.6.4 A/D コンバータモジュールのブロック図

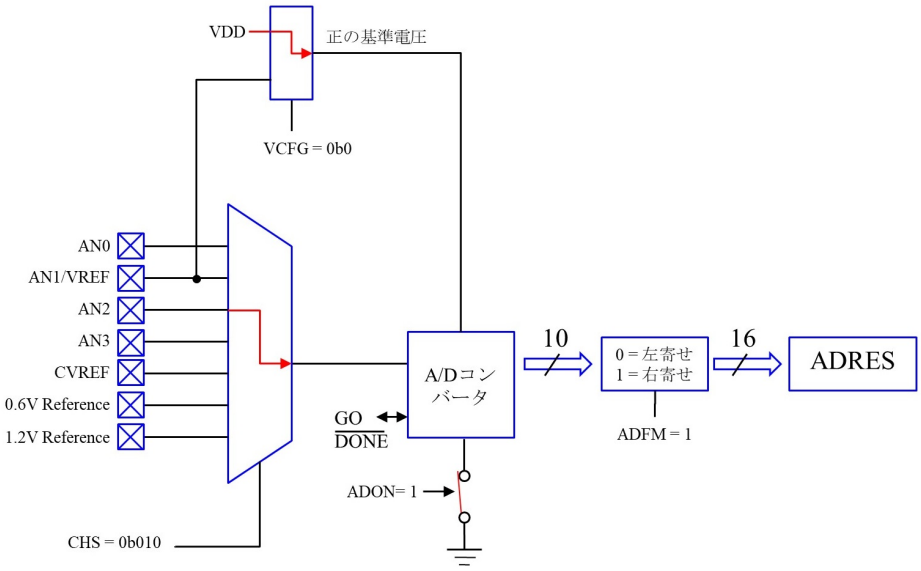
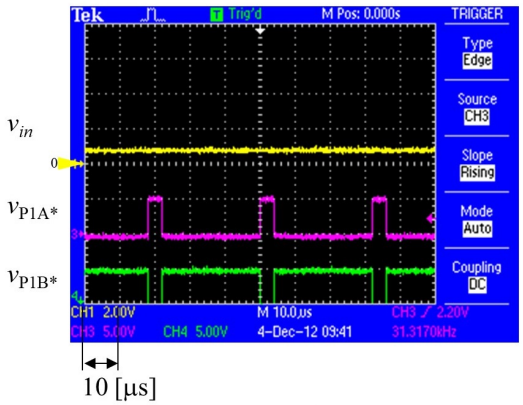


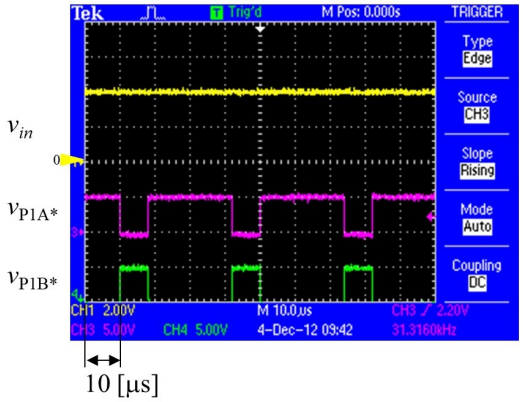
図 4.41: A/D コンバータモジュールのブロック図

図 4.41 に A/D コンバータモジュールのブロック図 (データシート ADC BLOCK DIAGRAM) を示す．このモジュールの設定は主に ADCON0 (A/D CONTROL REGISTER 0), ANSEL(ANALOG SELECT REGISTER) レジスタにより行う．図中の経路は式 (4.45) の関数の設定による．

4.6.5 実験結果



(a) v_{in} 低



(b) v_{in} 高

図 4.42: A/D コンバータモジュールの入力電圧と PWM 出力電圧

図 4.42 は A/D コンバータモジュールの入力電圧 v_{in}(5 番ピン) と PWM 出力電圧 v_{P1A*}(2 番ピン)，v_{P1B*}(3 番ピン) の波形例である．同図 (a) は v_{in} が低い場合であり，(b) は高い場合である．v_{in} により PWM 波形のデューティ比の制御ができている．

4.7 DC モータの回転数制御

4.7.1 プロジェクトの作成とプログラムのブロック図

新しいプロジェクト「DC_Motor_Control」を作成してください．そして，新たに作られた C:￥Users￥(ユーザー名)￥MPLABXProjects￥12HV615￥DC_Motor_Control.X フォルダへ，￥12HV615_モータドライブプログラム￥DC_Motor_Control.X フォルダから main.c ファイルをコピーしてください．次に，Source Files に main.c を “add” し，Linker Files に図 4.9 でコピーした C:￥Users￥(ユーザー名)￥MPLABXProjects￥12HV615￥PIC12HV615_Files フォルダから PI_contoller.c，set_ad_converter.c，set_pwm.c，set_timer1.c，set_timer2.c を “add” してください．

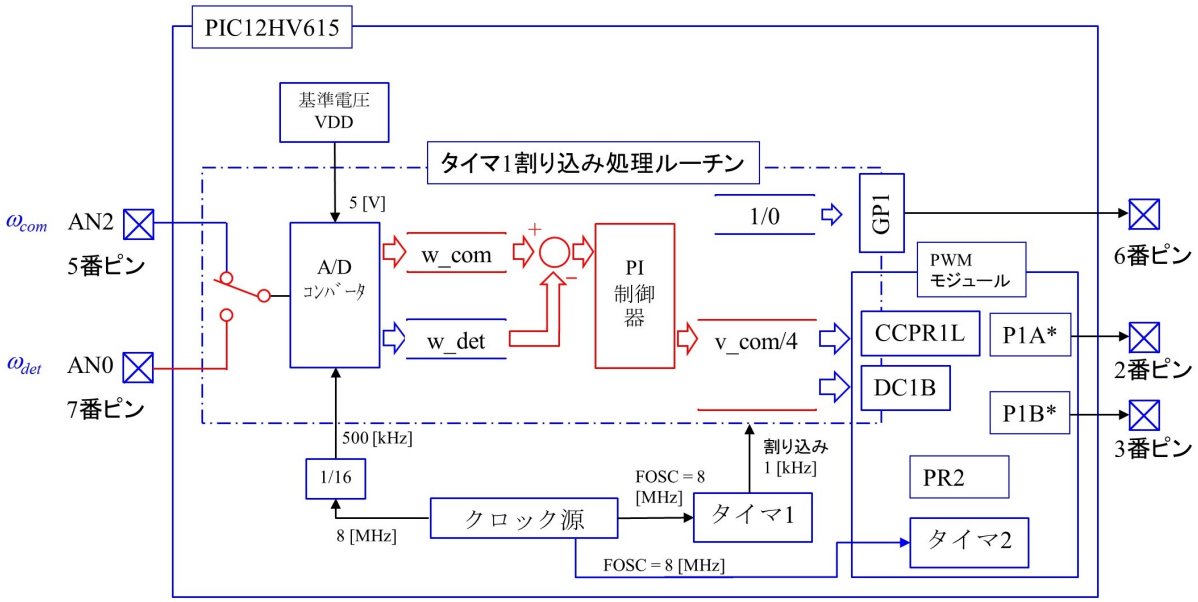


図 4.43: DC モータの回転数制御プログラムのブロック図

以上で 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 4.43 は作成したプログラムのブロック図を示す．図 4.35 の PWM コンバータ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 4.4 に示すように回転数指令値 ω_{com} を 5 番ピンに入力し，回転数検出値 ω_{det} を 7 番ピンに入力している．PIC12HV(F)615 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．図 4.4 において，ICSP コネクタを通して PICKit 3 もしくは PICkit 4 によりプログラムを書き込む場合には，切り替えスイッチ SW をオフとし，プログラムを実行して DC モータの回転数制御を行うときには PICkt x を ICSP コネクタから抜いて，SW をオンとする．

4.7.2 回転数制御プログラム

図 4.44〜図 4.48 に DC モータの回転数制御用プログラムを示す．これらは図 1.101, 1.102 と同じである．ただし，

$$\#\text{define } \_\text{XTAL\_FREQ} \quad 8000000 \tag{4.47}$$

と変更している．

```
#define PWM_period      0x3F        // PWM周期の決定 ( = FOSC/4/(PWM_period+1)
                                    //                = 8MHz/4/128 = 31.25 kHz)

signed int      Kp =  32;           // 比例ゲイン
signed int      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;   //積分値
```

図 4.44: DC モータ回転数制御プログラム _ 変数宣言 (￥DC_Motor_Control.X￥main.c)

```
static void interrupt                          // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    GP1 = 1;                                   // GP1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(0x1EB0);     // 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で割り込みが発生する.
    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による割り込み可とする.

    start_ad_conversion_ch_select(select_AN2); // アナログチャネルをGP2(5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_AN0); // アナログチャネルをGP0(7番ピン)に設定
    w_det = read_result_of_ad_conversion();    // 回転数検出値の読み出し

    PI_controller();                           // PIコントローラ

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

    GP1 = 0;                                   // BP11に0を出力する. 割り込み発生のモニタリング用
}
```

図 4.45: DC モータ回転数制御プログラム _ 割り込み処理関数) (￥DC_Motor_Control.X ￥main.c)

```
#define _XTAL_FREQ 8000000

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);           // 5μs 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
}
```

図 4.46: DC モータ回転数制御プログラム _ 設定関数 (￥PIC12HV615_Files￥PI_controller.c)

```
// PIコントローラ
void PI_controller()
{
	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;
	}
}
```

図 4.47: DC モータ回転数制御プログラム _ 設定関数 (つづき）(￥PIC12HV615_Files￥PI_controller.c)

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

//PIコントローラの変数と関数宣言
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

void PI_controller();
```

図 4.48: DC モータ回転数制御プログラム _ ヘッダファイル）(￥PIC12HV615_Files￥pic12HV615_s.c)

4.7.3 実験結果

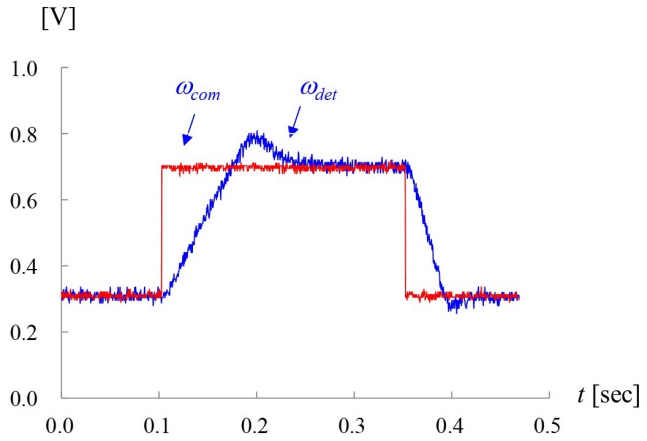


図 4.49: DC モータ回転数制御の実験結果 ($K_p = 32, K_{idt} = 1$)

図 4.49 は図 4.1 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 4.44〜4.48 のプログラムを用い，$K_p = 32, K_{idt} = 1$ の場合である．図 1.104 よりも速い応答が得られている．主な原因は第 1 章執筆時の DC モータが不調となり，本章で新品に交換したことが挙げられる．

参考文献

[1] 古橋武「パワーエレクトロニクスノート」コロナ社，2008.

索引

2012 年 11 月

2021 年 2 月改訂

著者

古橋　武

名古屋大学名誉教授・工学博士