

ChatGPTを用いたPICマイコン プログラミング

ー PIC16F1825によるDCモータ制御 ー

本稿掲載の Web ページ

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

令和6年3月

古橋 武

まえがき

ChatGPT 3.5 を用いて、PIC マイコンのプログラミングを試しました。PIC16F1825 を用いた DC モータ制御回路をブレッドボード上に構築して、このマイコン用の制御プログラムを ChatGPT により作成しました。以下の 2 通りの手順でプログラムを作成しました。

- (P1) [MCC により周辺モジュール設定を済ませた後に、PI 制御プログラムを ChatGPT により作成](#)
- (P2) [周辺モジュール設定プログラムと PI 制御プログラムの両方を ChatGPT により作成](#)

ChatGPT の能力は素晴らしく、上記の 2 通りの手順のいずれでも、PIC マイコン用 DC モータ制御プログラムを作成できました。筆者の

[「PIC16F1825 を用いた DC モータの PI 制御プログラム例を見せて」](#)

という指示（依頼）に対して、ChatGPT は、周辺モジュール設定 + PI 制御プログラムのたたき台を提示してくれました。その案に対して、プログラム改変／訂正を指示すると、ChatGPT は改訂案を提示してくれます。このやりとりを繰り返しながら、目標機能を有するプログラムにたどり着くことができました。

目標とした機能は

- (F1) [回転数指令値・検出値をアナログ値で読み込む。サンプリング周波数 1 kHz](#)
- (F2) [インバータ駆動用 PWM 電圧を出力する。PWM 周波数 32 kHz](#)
- (F3) [PI 制御を実行する。](#)

でした。ただし、手順 P1 では、機能 F1, F2 は MCC を用いた手作業で達成済みです。

ChatGPT は、まだ、周辺モジュール設定を正確にできません。ほとんどの設定には問題がないので、それだけでもありがたいのですが、一部の設定に不具合があります。その不具合を見つけるには多くの労力を要してしまいます。データシートを詳しく読む必要が生じます。

そこで、手順 P1 では、この ChatGPT の不得手な問題を回避しています。[MCC\(MPLAB® Code Configurator\)](#) はマイクロチップ社が無償で提供している、グラフィカル・ユーザ・インターフェースです。周辺モジュール設定の労力を大幅に軽減してくれる便利なツ

ルです。MCCにより周辺モジュール設定を済ませておき、目標機能をF3に絞ることで、筆者はストレスを感じることなく、ChatGPTのプログラム提案能力を楽しむことができました。筆者のような趣味レベルであれば、ChatGPTによるPICマイコンプログラミングは、現時点で、既に十分実用(?)的です。

手順P2がおそらく理想でしょう。筆者は、周辺モジュールの知識に基づいた指示をすることで、自らコードを書くこと無く、目標機能F1～F3をほぼ達成できました。ChatGPTの能力にただただ驚きました。ただし、ChatGPTとの対話で解決できない設定がいくつかあり、どのような指示/依頼/質問が適切なのを見つけれませんでした。そこで、妥協で採用したプログラム案に対して、一部に直接修正を加えました。周辺モジュール設定に詳しくないユーザであれば、プログラム案のバグに対して途方に暮れてしまうと予想します。

ChatGPTの能力は画期的です。今回の筆者の体験した不備などはそう遠くないうちに解決されると期待されます。そのときには、周辺モジュール設定を知らなくてもPICマイコンプログラミングができてしまいます。筆者は、PICマイコンを用いたモータドライブの本 [4]などを著してきましたが、さて、これからどう著していこうか？

手順P1, P2で作成したプログラムを本稿と同じWebページ

[モータドライブノート](#)

に掲載してあります。参考にしてください。

2024年3月

目次

第 1 章	PIC16F1825 による DC モータ回転数制御回路	4
1.1	DC モータ回転数制御回路の組み立て	4
第 2 章	手順 P1 : ChatGPT による PI 制御プログラムの作成	9
2.1	プログラムのブロック図	9
2.2	MCC による周辺モジュール設定後の画面	11
2.3	ChatGPT を用いて作成した PI 制御プログラム	12
2.3.1	プログラム	12
2.3.2	実験結果	14
2.4	ChatGPT を用いた PI 制御プログラムの作成経緯	15
2.4.1	最初の依頼	15
2.4.2	2 度目の依頼	18
2.4.3	3 度目の依頼	19
2.4.4	4 度目の依頼	20
2.4.5	5 度目（最後）の依頼	21
2.4.6	採用案	22
第 3 章	手順 P2 : ChatGPT による周辺モジュール設定 + PI 制御プログラムの作成	25
3.1	プログラムのブロック図	25
3.2	ChatGPT を用いて作成した周辺モジュール設定 + PI 制御プログラム	26
3.2.1	プログラム	26
3.2.2	実験結果	29
3.3	ChatGPT を用いた周辺モジュール設定 + Pi 制御プログラムの作成経緯	30
索引		31
参考文献		32

第1章

PIC16F1825 による DC モータ回転数制御回路

1.1 DC モータ回転数制御回路の組み立て

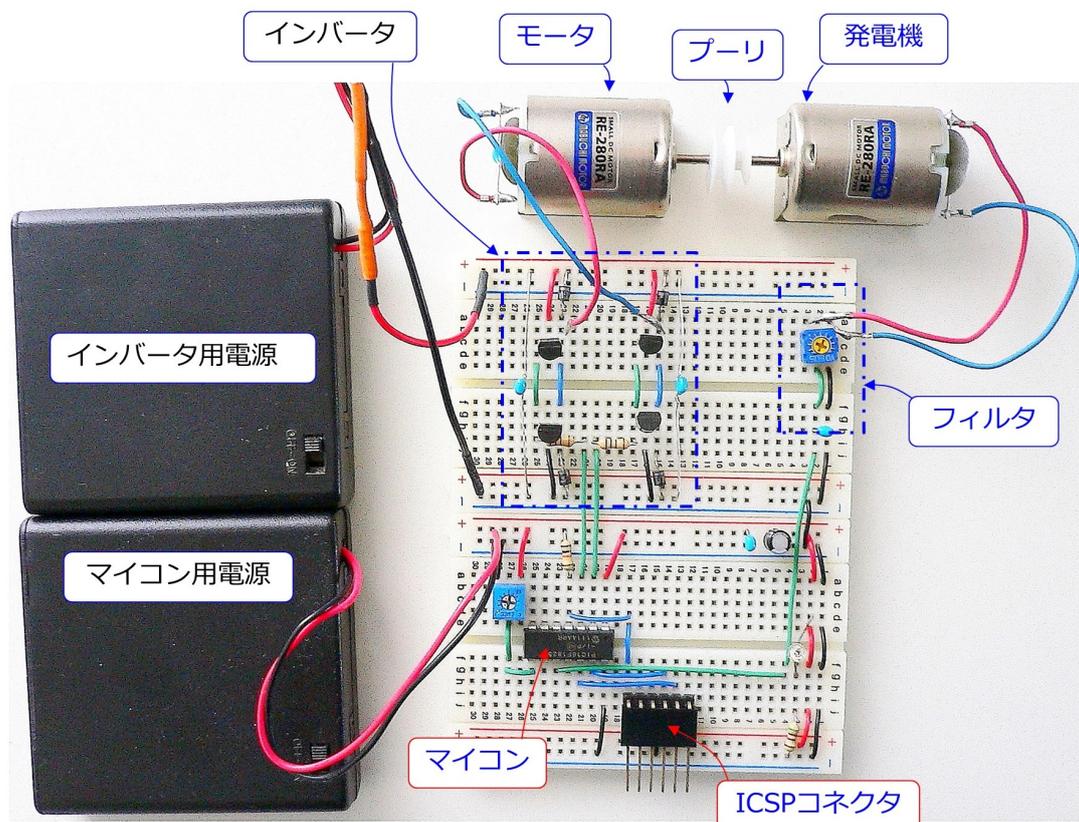


図 1.1: PIC16F1825 を用いた直流 (DC) モータの回転数制御回路 (全体写真)

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

同士をプリーでつなぎ、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の乾電池4本 (6 [V]), もしくは充電電池4本 (5 [V]) 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。

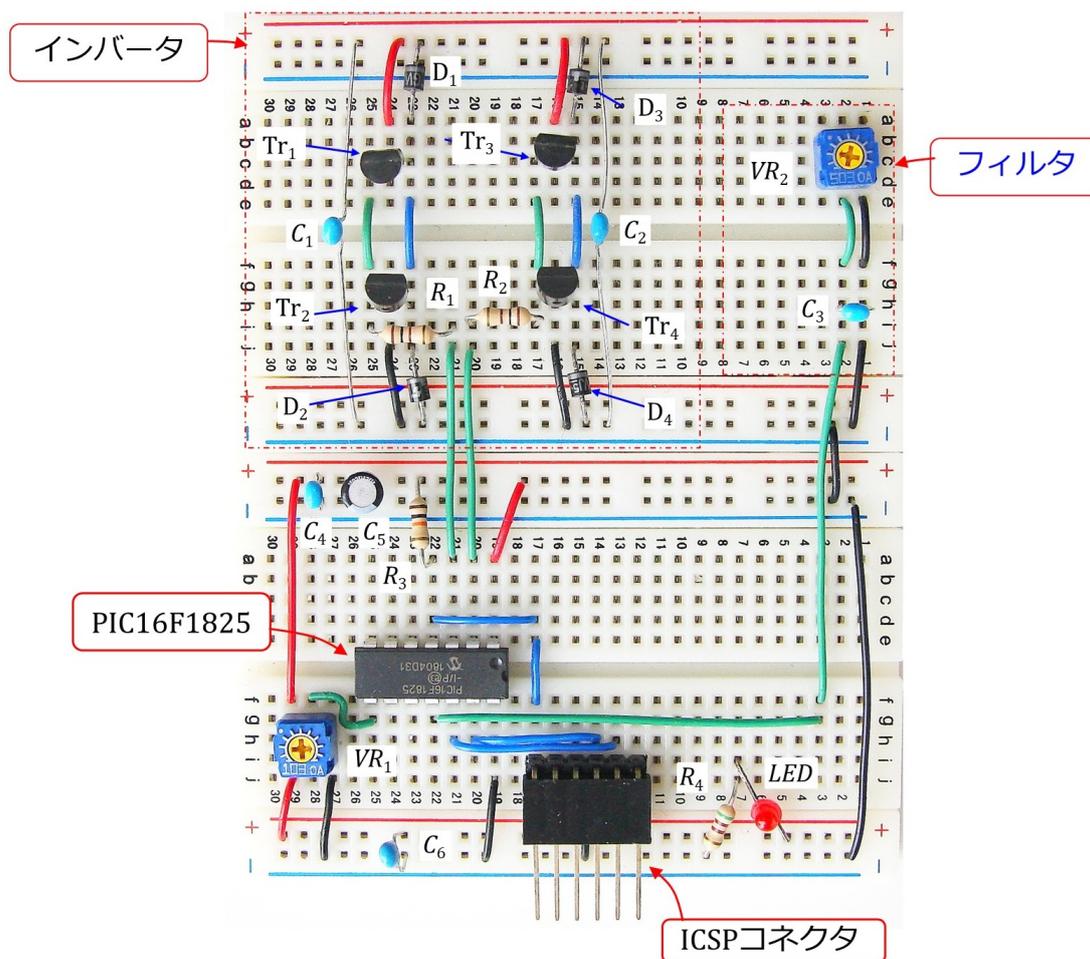


図 1.2: PIC16F1825 を用いた直流 (DC) モータの回転数制御回路 (拡大写真)

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、実体配線図を図 1.3 に示します。

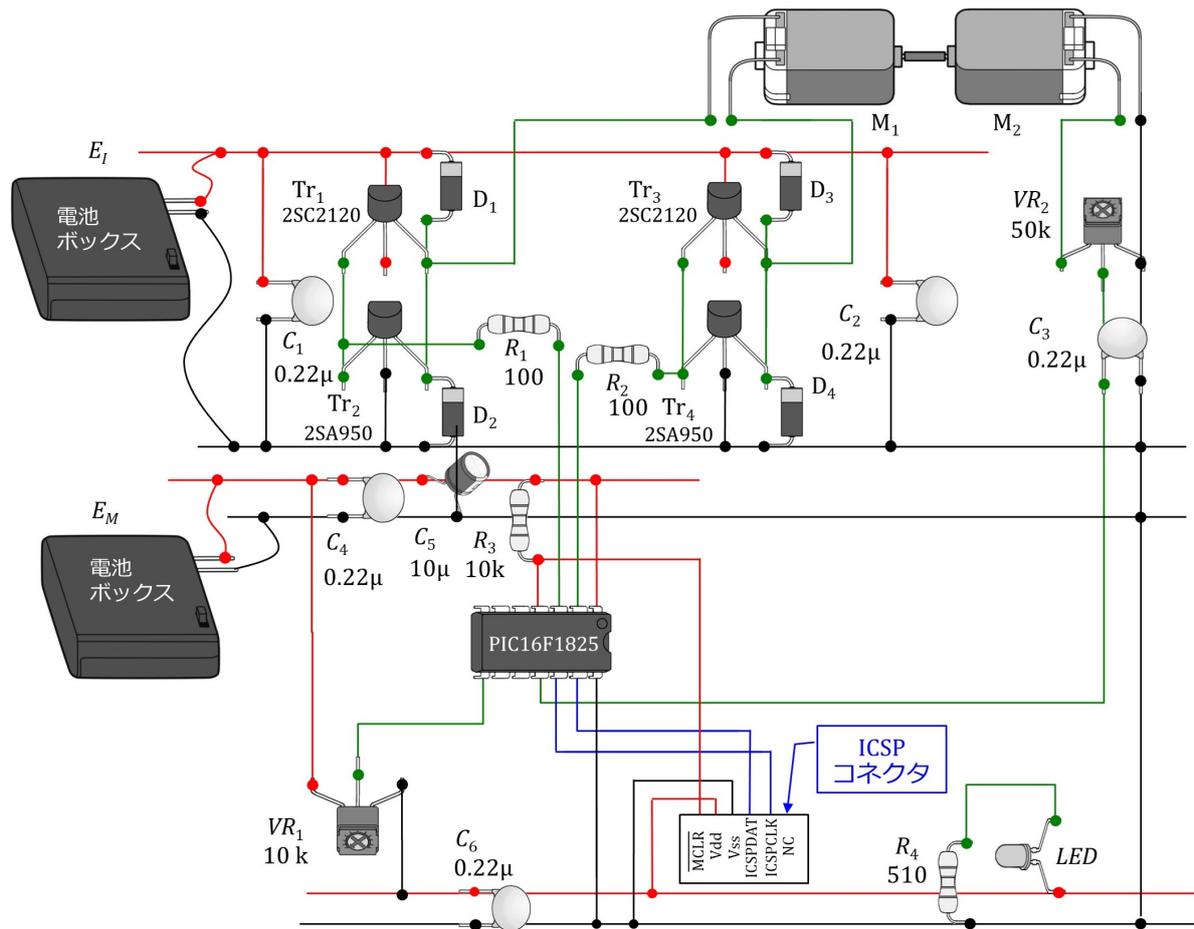


図 1.3: PIC16F1825 を用いた直流 (DC) モータの回転数制御回路実体配線図

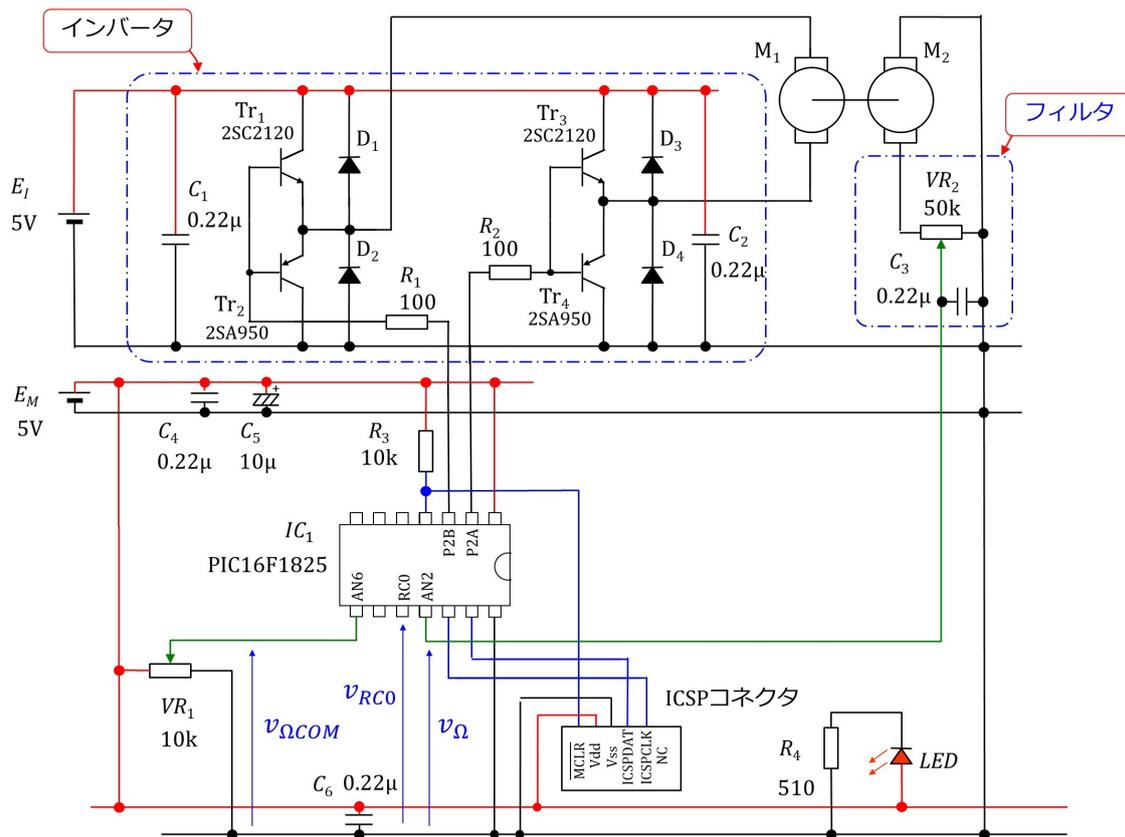


図 1.4: PIC16F1825 を用いた直流 (DC) モータの回転数制御回路図

回路図を図 1.4 に示します。マイコンは、**回転数指令値** $v_{\Omega COM}$ を 8 番 (AN6) ピンから読み込み、**回転数検出値** v_{Ω} を 11 番 (AN2) ピンから読み込みます。そして、PI 制御演算を行って、操作電圧を **PWM 電圧** として 2 番 (P2A) ピンと 3 番 (P2B) ピンから出力します。10 番 (RC0) ピンからは、タイマ 1 割り込み処理関数の **処理時間モニタ** 用電圧 v_{RC0} を出力します。

インバータは Tr_1, Tr_3 に **NPN 型トランジスタ** 2SC2120 を使い、 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 $D_1 \sim D_4$ には **ショットキーバリアダイオード** 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 は発電機として使い、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_6 はノイズ対策用、 C_5 は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

モータドライブノート

PIC16F1825 による DC モータの回転数制御 (第3版)
にて詳述しています。参考にしてください。

第2章

手順P1 : ChatGPTによるPI制御プログラムの作成

2.1 プログラムのブロック図

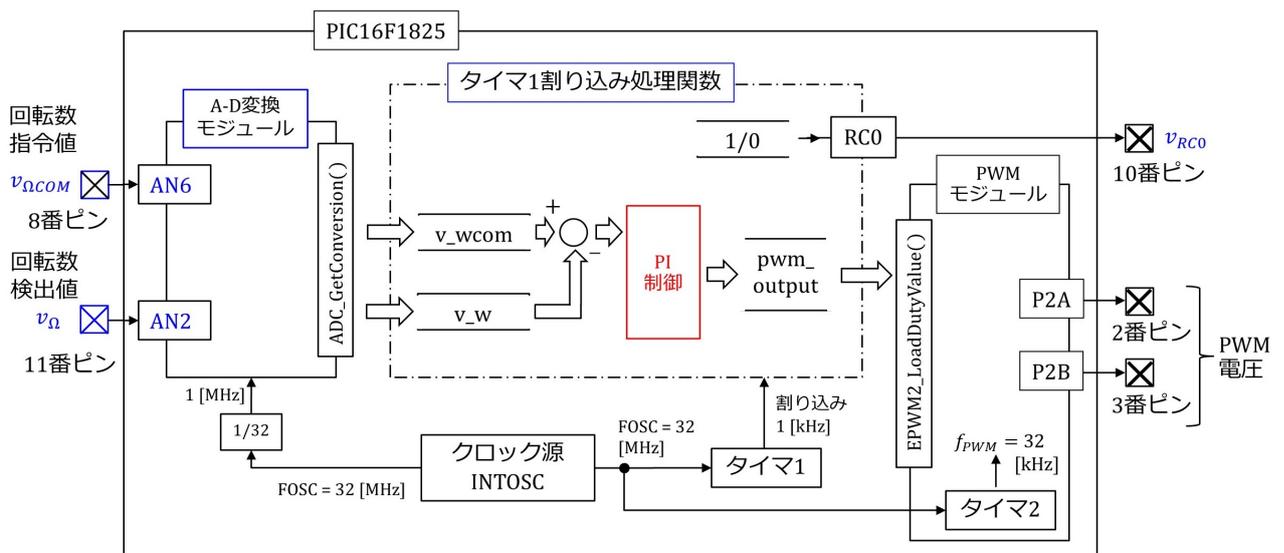


図 2.1: DC モータ制御プログラムのブロック図 (周辺モジュール設定済み)

図 2.1 は作成するプログラムのブロック図です。図中の PI 制御プログラムを ChatGPT 3.5 を用いて作成します。周辺モジュールは MCC を用いて設定済とします。MCC による周辺モジュール設定の詳細は

PIC16F1825 による DC モータの回転数制御 (第 3 版)
を参照してください。

クロック源にはマイコン内蔵のクロック INTOSC を使用し、システムクロック FOSC を 32 MHz にします。

A-D変換モジュールは、回転数指令値 $v_{\Omega COM}$ を 8 番 (AN6) ピンから読み込み、回転数検出値 v_{Ω} を 11 番 (AN2) ピンから読み込みます。PIC16F1825 内には A-D 変換モジュールが 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します。A-D 変換モジュールの動作周波数の上限値は 1 MHz です。そこで、システムクロックを $1/32$ 分周して、これを A-D 変換モジュールのクロックとします。

タイマ 1 割り込み処理関数が回転数指令値 v_{wcom} と検出値 v_w を入力として PI 演算を行い、その結果の v_{com} を PWM モジュールのデューティ比とします。タイマ 1 割り込み処理関数は、タイマ 1 により起動されます。割り込み周波数を 1 kHz とします。引数をチャンネル番号 (AN6 は 6, AN2 は 2) として `ADC_GetConversion()` 関数を実行することで、戻り値に A-D 変換結果を得ます。 $v_{\Omega COM}$ の変換結果を v_{wcom} に、 v_{Ω} のそれを v_w にそれぞれ格納します。PI 演算結果の `pwm_output` を引数として `EPWM2_LoadDutyValue()` 関数を読み出すことで、PWM モジュールのデューティ比を設定できます。

タイマ 1 割り込み処理関数の処理時間モニタ用として、割り込みのたびに 10 番 (RC0) ピンに “1” = “High”, “0” = “Low” の値を出力します。

PWM モジュールは、デューティ比に応じて、2 番 (P2A) ピンと 3 番 (P2B) ピンに PWM 電圧を出力します。PWM モジュールの動作クロックは FOSC (= 32MHz) です。PWM 周波数 f_{PWM} は、本稿では、32 kHz とします。これにより、`pwm_output` の上限値 `pwm_output_max` は

$$\begin{aligned} \text{pwm_output_max} &= \frac{\text{FOSC}}{f_{PWM}} - 1 \\ &= \frac{32\text{MHz}}{32\text{kHz}} - 1 \\ &= 999 \end{aligned} \tag{2.1}$$

です。

2.2 MCCによる周辺モジュール設定後の画面

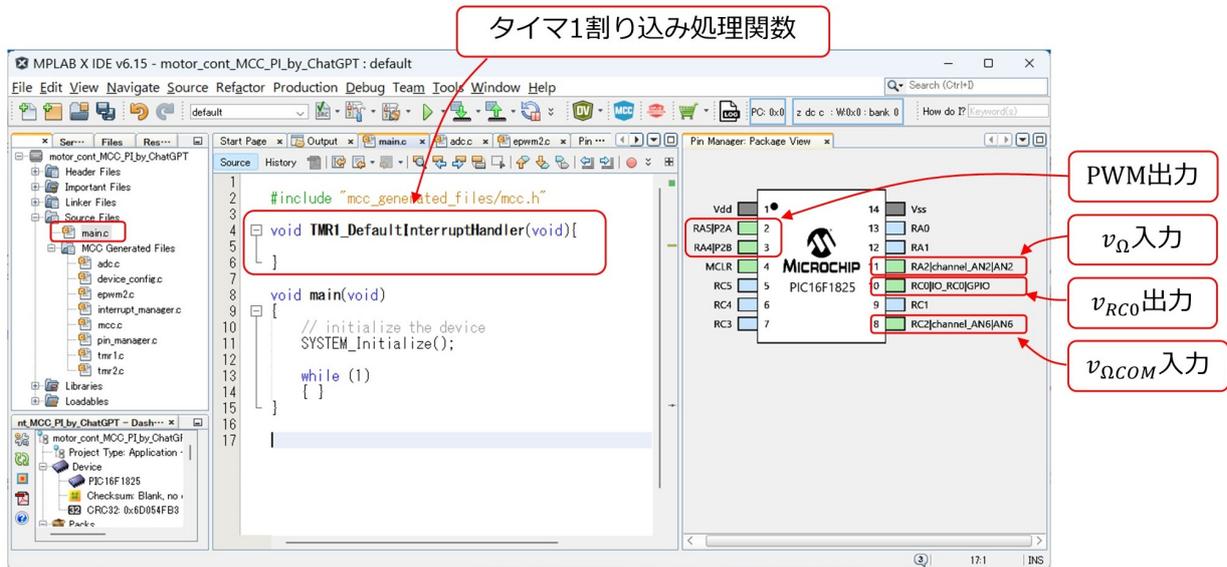


図 2.2: MCCによる周辺モジュール設定後の画面

図 2.2 は、MPLAB[®] X IDE の画面のスナップショットです。MCC により周辺モジュールの設定を終えた画面です。main.c ファイル内の関数が画面中央のエリアに表示されています。

TMR1_DefaultInterruptHandler() 関数がタイマ 1 割り込み処理関数です。これは、MCC により自動生成された tmr1.c ファイル内にあった関数を main.c ファイル内に移動させたものです。この関数内に、2.4 節で ChatGPT を用いて作成する PI 制御関数を移植します。

main() 関数では、プログラム起動時に SYSTEM_Initialize() 関数により、タイマ 1, 2 モジュール、A-D 変換モジュール、PWM モジュールを初期化して、その後に while 文により無限ループを実行します。

2.3 ChatGPTを用いて作成したPI制御プログラム

2.3.1 プログラム

結論から先に述べます。図 2.3, 図 2.4 が, ChatGPT を用いて作成した PI 制御プログラムです。筆者が依頼をして, ChatGPT がそれに答えることを 5 回繰り返した結果が 2.4.6 項の採用案です。この案の青字のコードを図 2.2 の main.c ファイル内に移植しました。2.4.6 項の採用案からの変更箇所を赤字で示します。

```
#include "mcc_generated_files/mcc.h"

// PI制御パラメータ
#define Kp 2 // 比例ゲイン
#define Ki 1 // 積分ゲイン

// PI制御用変数
int error = 0; // 目標値と現在値の差
int integral = 0; // 積分項
int pwm_output = 0; // PWM出力

// 上下限值設定関数
int16_t constrain(int value, int min, int max) {
    if (value > max) {
        return max;
    } else if (value < min) {
        return min;
    }
    return value;
}
```

図 2.3: ChatGPT を用いて作成した PI 制御プログラム (その 1)

図 2.3 は, 定数, グローバル変数, 上下限值設定関数です。PI 制御パラメータの $K_p = 2$ と $K_i = 1$ は, 実際にモータ制御を実施して, モータ回転数の応答を見ながら調整しました。

```
void TMR1_DefaultInterruptHandler(void){
    int    v_wcom, v_w;

    // RC0ピンに1を出力
    RC0 = 1;

    //回転数検出値の読み込み
    v_w = (int)ADC_GetConversion(0x02);
    //回転数指令値の読み込み
    v_wcom = (int)ADC_GetConversion(0x06);
    // 目標値と現在値の差を計算
    error = v_wcom - v_w;

    // 積分項の計算
    integral += error;

    // 積分項の上下限値のチェック
    integral = constrain(integral, -1000, 1000);

    // PI制御アルゴリズム
    pwm_output = (Kp * error) + (Ki * integral);

    // PWM出力の制約条件
    pwm_output = constrain(pwm_output, 0, 999);

    // PWM出力の更新
    EPWM2_LoadDutyValue((unsigned int)pwm_output);
    // RC0ピンに0を出力
    RC0 = 0;
}
```

MCCにより生成された関数の利用

MCCにより生成された関数の利用

図 2.4: ChatGPT を用いて作成したPI制御プログラム (その2)

図 2.4 は、タイマ1 割り込み処理関数内に移植したコードです。回転数検出値・指令値を読み込んで、PI 演算を行い、PWM モジュールのデューティ比を設定しています。

変更点は、MCC により生成された `ADC_GetConversion()` 関数を利用して回転数検出値・指令値を読み込むコードと、`EPWM2_LoadDutyValue()` 関数を用いてデューティ比を設定するコードです。

2.3.2 実験結果

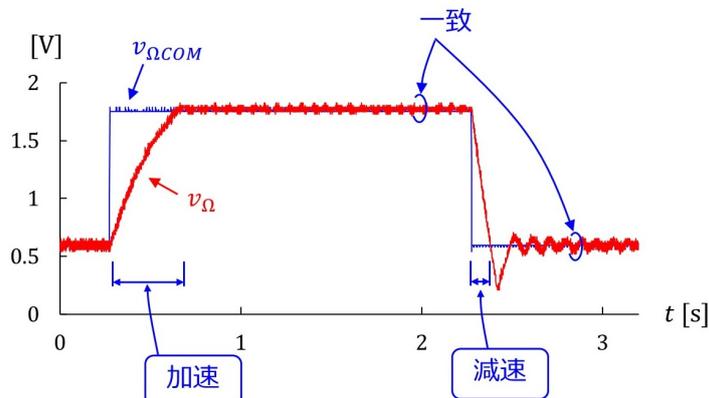


図 2.5: 実験結果 (回転数波形例)

図 2.5 は、実験結果の回転数波形例です。横軸は時間 t [s], 縦軸は電圧 [V] です。青線が回転数指令値 $v_{\Omega COM}$, 赤線が回転数検出値 v_{Ω} です。 $v_{\Omega COM}$ のステップ的立ち上がりで DC モータが加速され、立ち下がりで減速されています。モータの摩擦の影響で、減速の方が加速よりも所要時間が短くなりました。加減速期間が過ぎた後で、 v_{Ω} は $v_{\Omega COM}$ に一致しました。PI 制御の積分項の効果が現れていました。

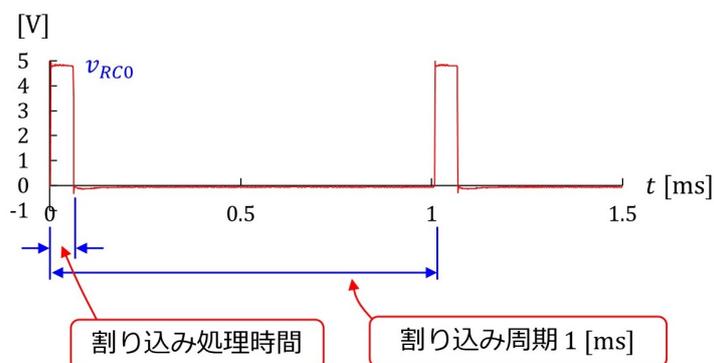
図 2.6: RC0 ピン電圧 v_{RC0} 波形例

図 2.6 は RC0 ピン電圧 v_{RC0} の波形例です。横軸は時間 t [ms], 縦軸は電圧 [V] です。図 2.4 のタイマ 1 割り込み処理関数が起動された最初に $v_{RC0} \approx 5$ [V] を出力し、同処理関数の最後に $v_{RC0} = 0$ を出力しています。これより、割り込み周期とタイマ 1 割り込み処理関数の処理時間がわかります。割り込み周期は 1 [ms], 割り込み処理時間は約 60 [μ s] でした。

2.4 ChatGPTを用いたPI制御プログラムの作成経緯

前節のプログラムの作成経緯を、本節で示します。ChatGPTは、筆者の指示（依頼）に応じて少しずつプログラム案を改訂していきます。最初のプログラム案は全コードを載せていますが、その後は改訂箇所のみを抜粋して示します。抜粋で改訂箇所が分かり難い場合は、

ChatGPTを用いたPI制御プログラムの作成経緯（周辺モジュール既設定）

を参照してください。ChatGPTが提示したプログラム案の全コードを省略せずに載せてあります。

2.4.1 最初の依頼

筆者とChatGPT 3.5とのやりとりの文章を枠で囲って示します。「You」の直下の文章が、筆者の依頼文です。それに対するChatGPTの回答が、「ChatGPT」直下の文章、その後続くプログラム、そして、プログラム後の枠中の文章です。

You

PIC16F1825用のPI制御プログラム例を示して

ChatGPT

PIC16F1825を使用したPI制御プログラムの例を示します。以下のコードは、センサーからの入力を使用してモーターを制御するための基本的なPI制御ループです。この例では、PWM信号を使用してモーターの速度を制御し、センサーの値をフィードバックしてPI制御アルゴリズムを実装します。

```
#include <xc.h>

// マクロ定義
#define _XTAL_FREQ 8000000 // ピックの動作周波数 (ここでは8MHzを仮定)

// 入力ピンと出力ピンの設定
#define SENSOR_PIN RA0 // センサー入力ピン
#define PWM_PIN RC2 // モーターPWM出力ピン

// PI制御パラメータ
#define Kp 0.5 // 比例ゲイン
#define Ki 0.05 // 積分ゲイン

// PI制御用変数
int error = 0; // 目標値と現在値の差
int integral = 0; // 積分項
int pwm_output = 0; // PWM出力

// 関数プロトタイプ
void setup();
void loop();
void updatePWM();

// メイン関数
int main() {
    setup(); // 初期設定
    while(1) { // メインループ
        loop();
    }
    return 0;
}

// 初期設定
void setup() {
    // ピンの設定
    ANSELAbits.ANSA0 = 1; // アナログ入力設定
    TRISC2 = 0; // RC2を出力に設定

    // タイマー2の設定
    T2CON = 0b00000110; // プリスケーラ 16, タイマー2 ON
    PR2 = 249; // PWM周期設定
    CCP1CON = 0b00001100; // PWMモード
    CCPR1L = 0; // PWM初期値
}

// メインループ
void loop() {
    // センサー値の読み取り
    int sensor_value = SENSOR_PIN;

    // 目標値と現在値の差を計算
    error = 100 - sensor_value; // 100は目標値

    // 積分項の計算
    integral += error;

    // PWM出力の更新
    updatePWM();

    _delay_ms(100); // サンプリング周期の遅延
}
```

```
// PWM出力の更新
void updatePWM(){

    // PI制御アルゴリズム
    pwm_output = (Kp * error) + (Ki * integral);

    // PWM出力の制約条件
    if (pwm_output > 255) {
        pwm_output = 255;
    } else if (pwm_output < 0) {
        pwm_output = 0;
    }

    // PWM出力の更新
    CCPR1L = pwm_output;
}
```

(ChatGPT 回答文のつづき)

このプログラムでは、`setup()` 関数でピンの設定とタイマー 2 の設定を行い、`loop()` 関数でセンサーの値を読み取り、PI 制御アルゴリズムを適用して PWM 出力を更新します。PWM 出力の更新は `updatePWM()` 関数で行われます。

この例では、センサー値の目標値を 100 としており、センサーからの入力为目标値からどれだけずれているかを計算して、それを元に PWM 出力を制御しています。また、積分項を使ってオーバーシュートを減らすための積分制御も行われています。

ChatGPT は周辺モジュール設定コードも提示していますが、これらは使いません。図 2.2 のタイマ 1 割り込み処理関数への移植に関わるコードを青字でします。

2.4.2 2度目の依頼

ChatGPT が提示した最初のプログラム案に対する、筆者の依頼と ChatGPT の回答です。積分項 `integral` に上下限值が設定されていないので、これの改善を求めています。

You

`integral` にも上下限值を設定して

ChatGPT

`integral` の値に上下限值を設定して、オーバーフローやアンダーフローを防ぐように修正したコードを以下に示します。

```
// 積分項の上下限值
#define INTEGRAL_MAX 1000
#define INTEGRAL_MIN -1000

// メインループ
void loop(){

    // 積分項の計算
    integral += error;

    // 積分項の上下限値のチェック
    if (integral > INTEGRAL_MAX) {
        integral = INTEGRAL_MAX;
    } else if (integral < INTEGRAL_MIN) {
        integral = INTEGRAL_MIN;
    }

}
```

(ChatGPT 回答文のつづき)

この修正では、積分項に上限値と下限値を設定し、その範囲内で積分項が保持されるようにしています。これにより、積分項が大きくなりすぎたり小さくなりすぎたりすることが防がれます。

最初のプログラムからの変更点のみを抜粋して朱書きで示してあります。提示された上下限値の適正值は、積分ゲイン K_i の値によっても変わるので、実験時に調整することになります。

2.4.3 3度目の依頼

たった2回の依頼で、十分に使えるプログラム案が得られました。以降は、ChatGPT がどこまで応えてくれるかを試しました。

前項の案ではマクロ定義でシステムクロック FOSC を 8MHz にし、また、タイマ2の設定で、プリスケアラを 16 にして、PR2 に 249 を格納しています。この場合の PWM 周波数 f_{PWM} は

$$\begin{aligned} f_{PWM} &= \frac{8\text{MHz}}{4 \times 16 \times 250} \\ &= 500\text{Hz} \end{aligned} \quad (2.2)$$

です。本稿の DC モータ制御では低すぎるので、PWM 周波数を 32kHz とするように依頼しました。

You

システムクロックを 32MHz にして、PWM 周波数を 32kHz にして

ChatGPT

システムクロックを 32MHz に設定し、PWM 周波数を 32kHz に設定するためには、プリスケアラやタイマーの設定を変更する必要があります。以下に修正したコードを示します。

```
// マクロ定義
#define XTAL_FREQ 32000000 // ピックの動作周波数 (ここでは32MHzを仮定)

// PWM周波数設定
#define PWM_FREQUENCY 32000

// 初期設定
void setup() {
    // タイマ-2の設定
    T2CON = 0b00000100; // プリスケアラ1, タイマ-2 ON
    PR2 = (XTAL_FREQ / (4 * PWM_FREQUENCY)) - 1; // PWM周期設定
    CCP1CON = 0b00001100; // PWMモード
    CCPR1L = 0; // PWM初期値
}
```

(ChatGPT 回答文のつづき)

この修正では、XTAL_FREQ を 32MHz に設定し、タイマ-2のプリスケアラを 1 に設定して、タイマーのカウント周波数をシステムクロックと同じにしています。また、PR2 の設定により PWM 周期を 32kHz に設定しています。

この改訂案中の PR2 は $32\text{MHz}/4/32\text{kHz} - 1 = 249$ です。PR2 を 249 に設定することで、(2.2) 式より、 $f_{PWM} = 32\text{kHz}$ となります。

2.4.4 4 度目の依頼

You

上下限值設定が2箇所にあるので、上下限值設定関数を定義して、この関数を各箇所
で引用するように改良して

ChatGPT

上下限值の設定を関数化し、その関数を各箇所呼び出すように改良したコードを以
下に示します。

```
// 上下限值設定関数
int constrain(int value, int min, int max) {
    if (value > max) {
        return max;
    } else if (value < min) {
        return min;
    }
    return value;
}

// メインループ
void loop() {

    // 積分項の計算
    integral += error;

    // 積分項の上下限值のチェック
    integral = constrain(integral, -1000, 1000);

}

// PWM出力の更新
void updatePWM() {
    // PI制御アルゴリズム
    pwm_output = (Kp * error) + (Ki * integral);

    // PWM出力の制約条件
    pwm_output = constrain(pwm_output, 0, 255);

    // PWM出力の更新
    CCPR1L = pwm_output;
}
```

(ChatGPT 回答文のつづき)

この修正では、constrain() という関数を定義して、上下限值のチェックをこの関数で行います。各箇所この関数を呼び出すことで、コードの可読性が向上し、修正が簡略化されます。

この案の `pwm_output` の上限値 255 は、(2.1) 式の上限値 `pwm_output_max = 999` の $1/4$ です。これは次の理由によります。タイマ 2 を PWM モジュールのタイムベースとした場合、タイマ 2 は、通常の 8 ビットタイマから、10 ビットタイマに機能拡張されます。しかし、上記抜粋コード最下行の PWM 出力更新法 (`CCPR1L = pwm_output`) は、タイマ 2 の拡張機能を利用せず、PWM 電圧を 8 ビットの分解能で決定します。一方、MCC により自動生成された `EPWM2_LoadDuty()` 関数 (図 2.4) は、PWM 電圧を 10 ビットの分解能で決定します。このため、MCC に基づく場合は、`pwm_output_max` を 4 倍にできます。

2.4.5 5 度目 (最後) の依頼

You

`loop()` 関数の最初に RC0 ピンに 1 を出力するコードを挿入して、最後に RC0 ピンに 0 を出力するコードを挿入して

ChatGPT

`loop()` 関数の最初に RC0 ピンに 1 を出力するコードを挿入し、最後に RC0 ピンに 0 を出力するコードを挿入した修正済みのコードは以下の通りです。

```
//メインループ
void loop(){
  //RC0ピンに1を出力
  RC0 = 1;

  // RC0ピンに0を出力
  RC0 = 0;
}
```

(ChatGPT 回答文のつづき)

この修正では、`setup()` 関数で RC0 ピンを出力に設定し、`loop()` 関数の先頭で RC0 ピンに 1 を出力し、最後で RC0 ピンに 0 を出力するコードを挿入しました。

2.4.6 採用案

5度目の依頼に対する ChatGPT のプログラム案の全文を以下に示します。青字のコードを図 2.2 の main.c ファイル内に移植しました。定数，グローバル変数，上下限值設定関数は割り込み処理関数の前に，以降の PI 制御演算をタイマ1 割り込み処理関数内に移植しました。

```
#include <xc.h>

// マクロ定義
#define _XTAL_FREQ 32000000 // ピックの動作周波数（ここでは32MHzを仮定）

// 入力ピンと出力ピンの設定
#define SENSOR_PIN RA0 // センサー入力ピン
#define PWM_PIN RC2 // モーターPWM出力ピン

// PI制御パラメータ
#define Kp 0.5 // 比例ゲイン
#define Ki 0.05 // 積分ゲイン

// PI制御用変数
int error = 0; // 目標値と現在値の差
int integral = 0; // 積分項
int pwm_output = 0; // PWM出力

// PWM周波数設定
#define PWM_FREQUENCY 32000

// 上下限值設定関数
int constrain(int value, int min, int max) {
    if (value > max) {
        return max;
    } else if (value < min) {
        return min;
    }
    return value;
}
```

```
// 関数プロトタイプ
void setup();
void loop();
void updatePWM();

// メイン関数
int main() {
    setup();                // 初期設定
    while(1) {              // メインループ
        loop();
    }
    return 0;
}

// 初期設定
void setup() {
    // ピンの設定
    ANSELAbits.ANSA0 = 1;    // アナログ入力設定
    TRISC0 = 0;              // RC0を出力に設定
    TRISC2 = 0;              // RC2を出力に設定

    // タイマー2の設定
    T2CON = 0b00000100;      // プリスケアラ 1, タイマー2 ON
    PR2 = (_XTAL_FREQ / (4 * PWM_FREQUENCY)) - 1; // PWM周期設定
    CCP1CON = 0b00001100;    // PWMモード
    CCP1L = 0;               // PWM初期値
}

// メインループ
void loop() {
    // RC0ピンに1を出力
    RC0 = 1;

    // センサー値の読み取り
    int sensor_value = SENSOR_PIN;

    // 目標値と現在値の差を計算
    error = 100 - sensor_value;    // 100は目標値

    // 積分項の計算
    integral += error;

    // 積分項の上下限値のチェック
    integral = constrain(integral, -1000, 1000);

    // PWM出力の更新
    updatePWM();

    __delay_ms(100);              // サンプリング周期の遅延

    // RC0ピンに0を出力
    RC0 = 0;
}
```

```
//PWM出力の更新
void updatePWM() {
    //PI制御アルゴリズム
    pwm_output = (Kp * error) + (Ki * integral);

    //PWM出力の制約条件
    pwm_output = constrain(pwm_output, 0, 255);

    //PWM出力の更新
    CCPR1L = pwm_output;
}
```

第3章

手順P2 : ChatGPTによる周辺モジュール設定+PI制御プログラムの作成

3.1 プログラムのブロック図

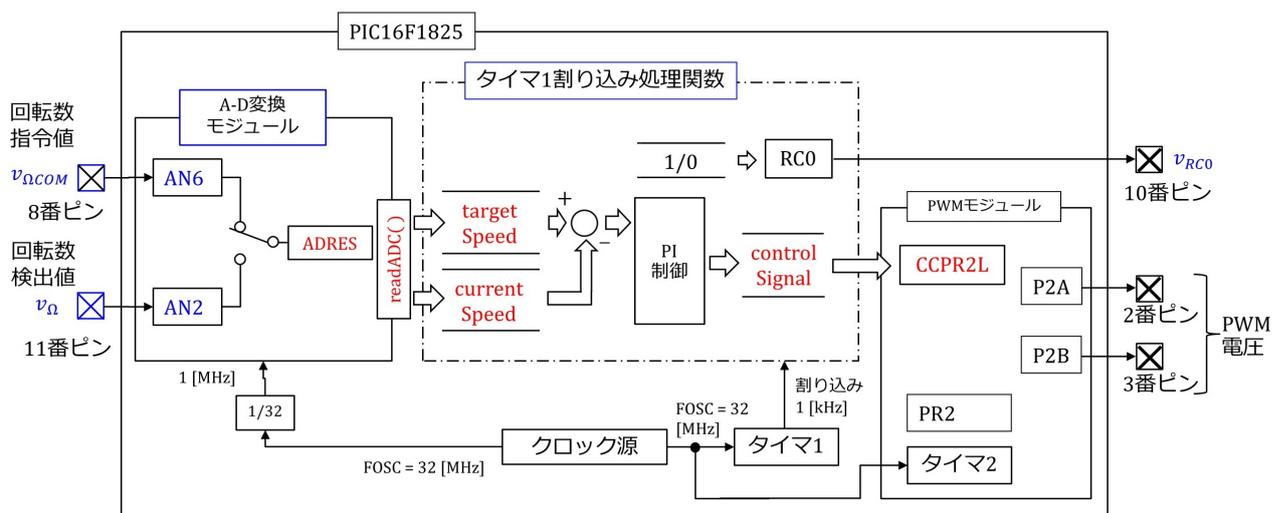


図 3.1: DC モータ制御プログラムのブロック図 (周辺モジュール設定+PI制御プログラムを ChatGPT により生成した場合)

図 3.1 は ChatGPT により作成したプログラムのブロック図です。周辺モジュール設定プログラムと PI 制御プログラムの両方を ChatGPT 3.5 を用いて作成した場合です。図 2.1 の PI 制御プログラムのみを ChatGPT により作成した場合のブロック図との違いを朱書きで示します。ブロック上での主な相違点は、関数名 `readADC()`、変数名 `targetSpeed`, `currentSpeed`, `controlSignal` です。

3.2 ChatGPTを用いて作成した周辺モジュール設定+PI制御プログラム

3.2.1 プログラム

以下に、採用案を示します。

```
#include <xc.h>

// コンフィギュレーションビット設定
#pragma config FOSC = INTOSC // 内部OSCを使用
#pragma config WDTE = OFF // ウォッチドッグタイマ無効
#pragma config LVP = ON // 低電圧プログラミング有効
#pragma config MCLR = OFF // MCLRピン機能無効

#define _XTAL_FREQ 32000000 // 内蔵OSCの周波数設定 (32MHz)

int Kp = 2; // 比例ゲイン
int Ki = 1; // 積分ゲイン
int integral = 0; // 積分値
int currentSpeed = 0; // 現在の速度 (フィードバック)
int targetSpeed = 500; // 目標速度 (例)

// 上下限値の設定
const int integral_max = 255;
const int integral_min = -255;
const int controlSignal_max = 249; // PR2と同じ値に設定
const int controlSignal_min = 0;

int controlSignal; // 制御信号

void setupADC(void) {
    // ADCの設定
    ADCON0 = 0; // ADCをオフにして初期化
    ADCON1 = 0b11100000; // 右詰め結果、Fosc/32
    ADCON0bits.ADON = 1; // ADCを有効化
}

void selectChannel(unsigned char channel) {
    // ADCチャンネルを選択
    ADCON0 &= 0b11000011; // チャンネル選択をリセット
    ADCON0 |= (channel << 2); // 新しいチャンネルを設定
}
```

```

unsigned int readADC(void) {
    // ADC変換の開始
    ADCON0bits.GO = 1;
    // 変換の完了を待つ
    while(ADCON0bits.GO_nDONE);
    // 結果を返す (右詰めのため、ADRESの値をそのまま返す)
    return ADRES;
}

void setup(void) {
    // クロックの初期設定
    OSCCON = 0b11110000; // IRCF = 1111 (32MHz), SCS = 00 (クロック選択はコンフィグワードに依存)

    // タイマ1設定
    T1CON = 0b00110001; // 1:1プリスケラ, FOSC/4, タイマ1 ON
    TMR1H = (65536 - 1000) >> 8; // 割り込み周期を設定 (1ms)
    TMR1L = (65536 - 1000) & 0xFF; // 割り込み周期を設定 (1ms)
    // PWM設定 (CCP2)
    CCP2CON = 0b10111100; // ハーフブリッジモード設定
    APFCON1bits.CCP2SEL = 1; // CCP2(P2A)をRA4に割り当て
    APFCON1bits.P2BSEL = 1; // P2B(PWM出力ピン)をRA5に割り当て
    TRISA4 = 0; // RA4(P2B)ピンを出力に設定
    TRISA5 = 0; // RA5(P2A)ピンを出力に設定
    PR2 = 249; // PWM周期 (32kHzにするための設定)
    T2CON = 0b00000100; // タイマ2設定 (プリスケラ1:4、タイマ2 ON)
    CCP2L = 0; // PWMデューティ比初期値 (0%)

    // RC0ピンをデジタル出力に設定
    TRISC0 = 0;

    // ADCの初期設定を追加
    setupADC();

    // 割り込み設定
    PIE1bits.TMR1IE = 1; // タイマ1割り込み有効
    INTCONbits.PEIE = 1; // ペリフェラル割り込み有効
    INTCONbits.GIE = 1; // グローバル割り込み有効
}

void updateMotorSpeed(void) {
    // AN2から速度フィードバック値を読み取り
    selectChannel(2); // AN2を選択
    currentSpeed = readADC(); // 現在の速度をADC値として読み取る

    // AN6からモータ速度設定値を読み取り
    selectChannel(6); // AN6を選択
    targetSpeed = readADC(); // 目標速度をADC値として読み取る

    // PI制御の計算
    int error = targetSpeed - currentSpeed; // 誤差
    int Pout = Kp * error; // 比例項
    integral += error; // 積分項
    // 上下限値のチェック
    if (integral > integral_max) integral = integral_max;
    if (integral < integral_min) integral = integral_min;

    int Iout = Ki * integral;
    controlSignal = Pout + Iout; // PI制御出力
    // 上下限値のチェック
    if (controlSignal > controlSignal_max) controlSignal = controlSignal_max;
    if (controlSignal < controlSignal_min) controlSignal = controlSignal_min;

    // PWMデューティ比を設定
    CCP2L = (unsigned char)(controlSignal); // デューティ比を設定
}

```

```
void _interrupt() isr(void){
    static int toggle = 0;           // RC0のトグル用変数
    RC0 = 1;

    if (PIR1bits.TMR1IF){           // タイマ1割り込みフラグチェック
        PIR1bits.TMR1IF = 0;       // タイマ1割り込みフラグをクリア

        TMR1H = (65536 - 1000) >> 8; // 割り込み周期を設定 (1ms)
        TMR1L = (65536 - 1000) & 0xFF; // 割り込み周期を設定 (1ms)

        updateMotorSpeed();        // モータ速度の更新
    }
    RC0 = 0;
}

void main(void){
    setup();

    while(1){
        // メインループで行う処理
        // この例では、すべての処理は割り込みによって行われるため、ここは空です。
    }
}
```

赤字のコードは、ChatGPTに依頼することを諦めて、筆者が修正しました。また、赤字のコメントも筆者が訂正しました。

3.2.2 実験結果

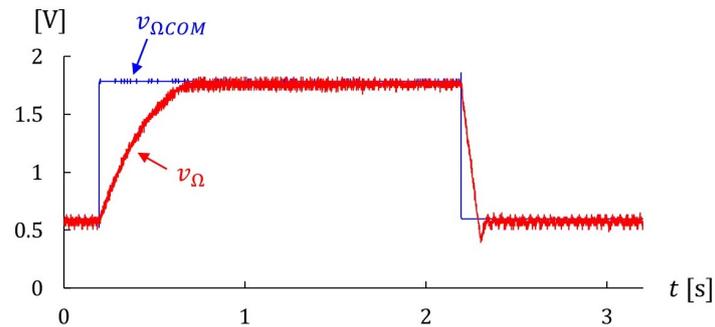


図 3.2: 実験結果（回転数波形例）

図3.2は、実験結果の回転数波形例です。横軸は時間 t [s]，縦軸は電圧 [V] です。青線が回転数指令値 $v_{\Omega COM}$ ，赤線が回転数検出値 v_{Ω} です。図2.5とほぼ同様の波形が得られました。

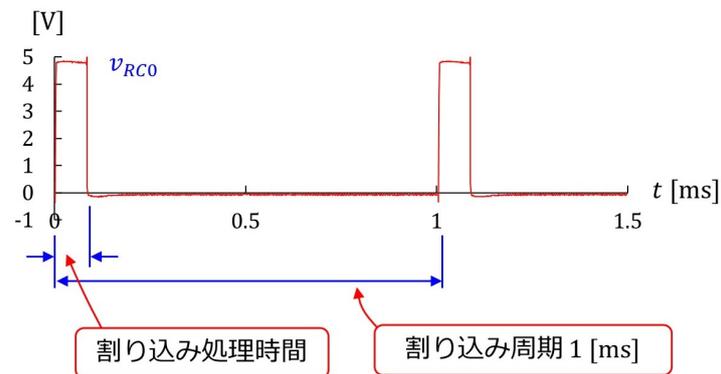


図 3.3: RC0 ピン電圧 v_{RC0} 波形例

図3.3はRC0ピン電圧 v_{RC0} の波形例です。割り込み周期は1 [ms]，割り込み処理時間は約 $85[\mu\text{s}]$ でした。

3.3 ChatGPTを用いた周辺モジュール設定+Pi制御プログラムの作成 経緯

ChatGPTに対して指示／依頼をして，ChatGPTが回答することを繰り返して，周辺モジュール設定+PI制御プログラムを作成しました．筆者は，採用案に対する一部訂正を除き，コードの記述は行いませんでした．

作成過程が長すぎるため，以下のファイルに掲載してあります．

[ChatGPTを用いた周辺モジュール設定+PI制御プログラムの作成経緯](#)
参照してください．

索引

- (MPLAB[®] Code Configurator, 1
- ADC_GetConversion() 関数, 10, 13
- A-D 変換モジュール, 10
- controlSignal, 25
- currentSpeed, 25
- EPWM2_LoadDutyValue() 関数, 10
- EPWM2_LoadDutyValue() 関数, 13
- FOSC, 10
- ICSP コネクタ, 5
- INTOSC, 10
- main 関数, 11
- MCC, 1
- NPN 型トランジスタ, 7
- PI 演算, 10
- PI 制御プログラム, 12
- PNP 型トランジスタ, 7
- PWM 周波数, 10
- PWM 制御, 7
- PWM 電圧, 7
- PWM モジュール, 10
- RC フィルタ, 7
- readADC, 25
- SYSTEM_Initialize() 関数, 11
- targetSpeed, 25
- TMR1_DefaultInterruptHandler() 関数, 11
- インバータ, 4
- インバータ用電源, 5
- 回転数検出値, 7, 10
- 回転数指令値, 7, 10
- クロック源, 9
- システムクロック, 10
- ショットキーバリヤダイオード, 7
- 処理時間モニタ, 7, 10
- 上下限值設定関数, 12
- タイマ 1, 10
- タイマ 1 割り込み処理関数, 10, 11
- デューティ比, 10
- フィルタ, 5
- フィルタ回路, 7
- 割り込み周波数, 10
- マイコン用電源, 5

参考文献

- [1] 後閑哲也「C 言語 & MCC による PIC プログラミング大全」技術評論社，2023.
- [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
- [7] ラジオノート

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.