

ChatGPT 3.5 を用いた PIC マイコン(PIC16F1825)用 PI 制御プログラムの作成経緯
(周辺モジュール既設定)

令和 6 年 3 月
古橋武

[本稿掲載の Web ページ](#)

(最初の依頼)

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) {
        loop(); // メインループ
    }
    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;    // プリスケータ 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;
}

```

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。

(2 度目の依頼)

You

integral にも上下限値を設定して

ChatGPT

integral の値に上下限値を設定して、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。

```
#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

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

// 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) {
        loop(); // メインループ
    }
    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;

  // 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;
  }

  // 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;
}

```

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。

(3 度目の依頼)

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。

```
#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

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

// 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) {
        loop(); // メインループ
    }
    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 = 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
    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;

    // 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;
    }

    // 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;
}

```

この修正では、`_XTAL_FREQ` を 32MHz に設定し、タイマー2 のプリスケアラを 1 に設定して、タイマーのカウンタ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。

(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も引用するように改良して

ChatGPT

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

```
#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) {
        loop(); // メインループ
    }
    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 = 0b00000100; // プリスケーラ 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
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;

    // 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); // サンプルング周期の遅延
}

// 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
    pwm_output = constrain(pwm_output, 0, 255);

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

```

この修正では、`constrain()`という関数を定義して、上下限値のチェックをこの関数で行います。各箇所での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。

(5 度目の依頼)

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。

```
#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) {
        loop(); // メインループ
    }
    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;     // プリスケーラ 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
CCPR1L = 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;
}

```

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。