

MCC による周辺モジュール設定後に ChatGPT を用いたプログラム作成経緯

令和 6 年 3 月

古橋武

本稿掲載の Web ページ

User

pic16f1705 で、正弦波を DA 変換モジュールより出力するプログラムを生成して

GPT

PIC16F1705 マイクロコントローラを使用して正弦波を DA 変換モジュールから出力するためのプログラムの基本的な構造は以下の通りです。この例では、正弦波の値を配列に格納し、それを周期的に DAC モジュールに送ることで、アナログの正弦波を生成します。

```
#include <xc.h>

// PIC16F1705 の設定 (FOSC, WDTE, etc.に合わせて変更してください)
#pragma config FOSC = INTOSC, WDTE = OFF, MCLRE = OFF, CP = OFF

#define _XTAL_FREQ 500000          //内部クロックの周波数を設定

// 正弦波のサンプル (例として 16 点の正弦波)
const unsigned char sine_wave[] = {127, 176, 218, 245, 255, 245, 218, 176, 127, 78, 36, 9, 0, 9, 36, 78};

void main(void) {
    // オシレータ設定 (必要に応じて変更)
    OSCCON = 0x70;                // 内部クロックを 8MHz に設定

    // DAC 設定
    DACCON0 = 0x80;              // DAC を有効にし、Vdd を参照電圧とする
    DACCON1 = 0;                 // DAC 出力を 0 に初期化

    // メインループ
    while (1) {
        for (int i = 0; i < sizeof(sine_wave); i++) {
            DACCON1 = sine_wave[i]; // DAC に正弦波のサンプル値を設定
            _delay_ms(2);          // 適切なディレイで波形の更新速度を調整
        }
    }
}
```

このプログラムは、指定された正弦波データを使用して、PIC16F1705 の DA 変換モジュールを介して正弦波を生成します。正弦波データ (sine_wave 配列) は、実際の要件に応じて変更することができます。また、ディレイの長さ (_delay_ms 関数内の値) を変更することで、出力される正弦波の周波数を調整できます。

User

タイマ 1 割り込み処理関数を設定して、正弦波の値を DAC に設定するコードを、この割り込み処理関数内に移して

GPT

PIC16F1705 でタイマ 1 を使用して割り込み処理を行い、正弦波を生成するコードは以下のようになります。このコードでは、タイマ 1 割り込みを使って定期的に DAC の値を更新し、滑らかな正弦波を生成します。

```
#include <xc.h>

// PIC16F1705 の設定
#pragma config FOSC = INTOSC, WDTE = OFF, MCLRE = OFF, CP = OFF

#define _XTAL_FREQ 500000          // 内部クロックの周波数

// 正弦波のサンプル
const unsigned char sine_wave[] = {127, 176, 218, 245, 255, 245, 218, 176, 127, 78, 36, 9, 0, 9, 36, 78};
unsigned char sine_index = 0;      // 正弦波配列のインデックス

void setup_timer1() {
    T1CON = 0x31;                  // タイマ 1 を 1:8 プリスケールで設定
    TMR1IE = 1;                   // タイマ 1 の割り込みを有効にする
    TMR1IF = 0;                   // タイマ 1 の割り込みフラグをクリア
    TMR1 = 65536 - 625;           // 割り込み頻度を設定 (例: 1kHz で割り込みが発生)
    PEIE = 1;                     // 周辺装置割り込みを有効にする
    GIE = 1;                      // グローバル割り込みを有効にする
}

void main(void) {
    // オシレータと DAC の設定
    OSCCON = 0x70;                // 内部クロックを 8MHz に設定
    DACCON0 = 0x80;               // DAC を有効にし、Vdd を参照電圧とする
    DACCON1 = 0;                  // DAC 出力を 0 に初期化

    setup_timer1();              // タイマ 1 の設定

    // メインループ
    while (1) {
        // ここは空にしておき、正弦波の生成は割り込みで行う
    }
}

void _interrupt() isr() {
    // タイマ 1 の割り込みかどうかチェック
```

```
if (TMR1IF) {  
    DACCON1 = sine_wave[sine_index];    // 正弦波の次の値を DAC に設定  
    sine_index = (sine_index + 1) % sizeof(sine_wave); // インデックス更新  
    TMR1IF = 0;                          // 割り込みフラグをクリア  
    TMR1 = 65536 - 625;                   // タイマ 1 を再設定  
}  
}
```

このプログラムでは、タイマ 1 の割り込みを利用して、一定の間隔で DAC の出力を更新して
います。割り込みハンドラ内で、sine_wave 配列から次の値を読み取り、DAC に設定すること
で、連続した正弦波を生成します。割り込み間隔（TMR1 に設定する値）は、所望の正弦波の周波数
に応じて調整してください。

User

このプログラムのタイマ 1 割り込み処理関数内で、RA4 ピンからアナログ値を読み込んで、AD 変換を行い、その変換結果で割り込み周期を変えるプログラムに変更して。

GPT

以下のプログラムでは、PIC16F1705 を使って、タイマ 1 割り込み処理関数内で RA4 ピンからアナログ値を読み取り、AD 変換を行い、その変換結果を基に割り込み周期を変えるように変更しています。この例では、AD 変換結果を用いてタイマ 1 のオーバーフロー時間を動的に変更しています。

```
#include <xc.h>

// PIC16F1705 configuration settings
#pragma config FOSC = INTOSC, WDTE = OFF, MCLRE = OFF, CP = OFF

#define _XTAL_FREQ 500000          // Set the internal oscillator frequency

// Sine wave samples
const unsigned char sine_wave[] = {127, 176, 218, 245, 255, 245, 218, 176, 127, 78, 36, 9, 0, 9, 36, 78};
volatile unsigned char sine_index = 0; // Index for sine wave array

void setup_adc() {
    ADCON0 = 0x11;           // Enable ADC, channel AN4 (RA4)
    ADCON1 = 0x80;           // ADC result is right-justified
    ADRESL = 0;              // Clear AD result register
    ADRESH = 0;
    ADON = 1;                // Turn on the ADC module
}

void setup_timer1() {
    T1CON = 0x31;           // Configure Timer1 with 1:8 prescale
    TMR1IE = 1;             // Enable Timer1 interrupt
    TMR1IF = 0;             // Clear Timer1 interrupt flag
    PEIE = 1;               // Enable peripheral interrupts
    GIE = 1;                // Enable global interrupts
}

void main(void) {
    // Oscillator, DAC and ADC configuration
    OSCCON = 0x70;          // Set internal oscillator to 8MHz
    DACCON0 = 0x80;         // Enable DAC, set Vdd as reference voltage
    DACCON1 = 0;           // Initialize DAC output to 0

    setup_adc();            // Setup ADC
    setup_timer1();         // Setup Timer1

    // Main loop
    while (1) {
        // Main loop left empty intentionally, sine wave generation and ADC read are handled by interrupt
    }
}

// Interrupt service routine
void __interrupt() isr() {
    // Check if it is a Timer1 interrupt
    if (TMR1IF) {
        DACCON1 = sine_wave[sine_index]; // Set DAC to next value of the sine wave
    }
}
```

```

sine_index = (sine_index + 1) % sizeof(sine_wave); // Update the sine wave index

// Start an ADC conversion
GO_nDONE = 1;
while (GO_nDONE); // Wait for the conversion to finish

// Adjust Timer1 interval based on ADC result
unsigned int newTimer1Value = 65536 - (ADRES * 4); // Example scaling, adjust as necessary
TMR1 = newTimer1Value;

TMR1IF = 0; // Clear Timer1 interrupt flag
}
}

```

このプログラムでは、タイマ 1 の割り込みの度に、RA4 ピン（AN4 チャンネル）でアナログ値を読み取り、AD 変換を実施しています。AD 変換の結果は、タイマ 1 の新しいプリロード値を計算するのに使用されます。このプリロード値はタイマの割り込み周期を制御し、結果的に正弦波の更新速度に影響します。この例では、ADRES の値に基づいてタイマのプリロード値を動的に調整していますが、必要に応じてスケーリング係数を調整してください。