

オペアンプ内蔵形 PIC マイコン (PIC16F1705) による

正弦波発生器 (第2版)

ー ChatGPT による PIC マイコンプログラミングー

本稿掲載の Web ページ

https://mybook-pub-site.sakura.ne.jp/Power_Electronics_Note/

令和6年3月

古橋 武

目次

第 1 章	はじめに	3
第 2 章	部品	6
2.1	PIC マイコン	6
第 3 章	実験回路	9
第 4 章	ゼロからの ChatGPT によるプログラミング	13
4.1	得られたプログラム	13
4.2	周辺モジュールのブロック図	16
4.3	実験結果	17
4.4	ChatGPT との対話の経緯	18
第 5 章	MCC で周辺モジュール設定後の ChatGPT によるプログラミング	20
5.1	得られたプログラム	20
5.2	周辺モジュールのブロック図	24
5.3	実験結果	24
5.4	ChatGPT との対話の経緯	24
第 A 章	付録	26
A.1	New Project の作成方法	26
A.2	MCC の起動	28
A.3	システムモジュール設定	30
A.4	タイマ 1 モジュールの設定	32
A.5	A-D 変換モジュールの設定	34
A.6	D-A 変換モジュールの設定	36
A.7	オペアンプモジュール 1 の設定	39
A.8	オペアンプモジュール 2 の設定	41
A.9	周辺モジュール設定関数の自動生成	42

A.10 追加の設定	43
索引	45
参考文献	45

第1章

はじめに

ChatGPT 4 を用いて、PIC16F1705 による正弦波発生器のプログラミング行いました。アプローチは以下の二通りです。

(1) ゼロからの ChatGPT を用いたプログラミング

このアプローチでは、ChatGPT を活用して、PIC16F1705 マイクロコントローラのプログラミングを始めから終わりまで行います。すなわち、初期化コードから始めて、ピン設定、周辺モジュール設定、そして正弦波を生成するための関数まで、全てを ChatGPT を用いて作成します。この方法は、マイクロコントローラの内部動作を深く理解したい場合や、特定の条件下でのカスタム動作が必要な場合に適しています。

(2) MCC で周辺モジュール設定後の ChatGPT を用いたプログラミング

MCC、つまり MPLAB[®] Code Configurator を使用するこの方法では、MCC ツールを利用して PIC16F1705 のピン設定、周辺モジュール（例えば、タイマ、A-D 変換モジュールなど）設定をします。その後、設定済みの環境に ChatGPT を用いて正弦波を生成するためのコードを作成します。このアプローチは、設定が容易で、より早くプロトタイプを作成したい場合や、周辺モジュールに関する詳細な設定を GUI を通して行いたい場合に適しています。

どちらのアプローチも PIC16F1705 を用いた正弦波発生器の開発において有効です。選択はプロジェクトの要件、時間、そして個人の好みに依存します。

以下は、両アプローチのメリットとデメリットを ChatGPT に尋ねた結果です。ほぼ、ChatGPT 4 の回答そのままです。

(1) ゼロから ChatGPT を用いたプログラミング

●メリット

インタラクティブな学習: ChatGPT は質問に答え、プログラミングの疑問を解決しながら学習できます。

コードサンプルと説明: ChatGPT は、理解を深めるために様々なコードサンプルとその説明を提供できます。

カスタマイズ性: ChatGPT によるアドバイスを基に、特定のニーズに合わせたコードを作成することができます。

● デメリット

時間消費: インタラクティブな質問応答には時間がかかり、ゼロからの開発は時間がかかることが多いです。

理解度に依存: ChatGPT が提供する情報の理解度が低いと、プログラミングの進行が遅くなる可能性があります。

(2) MCC で周辺モジュール設定後の ChatGPT を用いたプログラミング

● メリット

効率的なプログラミング: MCC による初期設定後、ChatGPT は特定の機能の実装にフォーカスしてサポートできます。

具体的な質問解決: MCC で設定した周辺モジュールに関連する具体的な問題やエラーに対して、ChatGPT がサポートを提供できます。

時間の節約: MCC の使用により基本的な設定時間を節約し、ChatGPT との対話をより高度なプログラミング課題に集中させることができます。

● デメリット

知識の偏り: MCC による自動設定は基本的な知識の習得を妨げ、ChatGPT との対話でその部分が補われない場合があります。

依存性: MCC と ChatGPT への依存が強まると、独立して問題を解決する能力が低下する可能性があります。

カスタマイズの限界: MCC で設定できる範囲外のカスタマイズが必要な場合、ChatGPT との対話だけでは解決しづらい場合があります。

これらのメリットとデメリットを考慮することで、プロジェクトや学習目標に最適なアプローチを選択できます。

本稿で解説するソースコードは、本稿掲載ページ

オペアンプ内蔵形 PIC16F1705 による正弦波発生器

の圧縮ファイル sin_wave_ChatGPT.zip にあります。ダウンロード，解凍してお使いください。

類書に，

「PIC16F1825 による DC モータの回転数制御 (最新版)

－ ChatGPT を用いたプログラミングー」

があります。こちらもお訪ねください。

第2章

部品

2.1 PIC マイコン

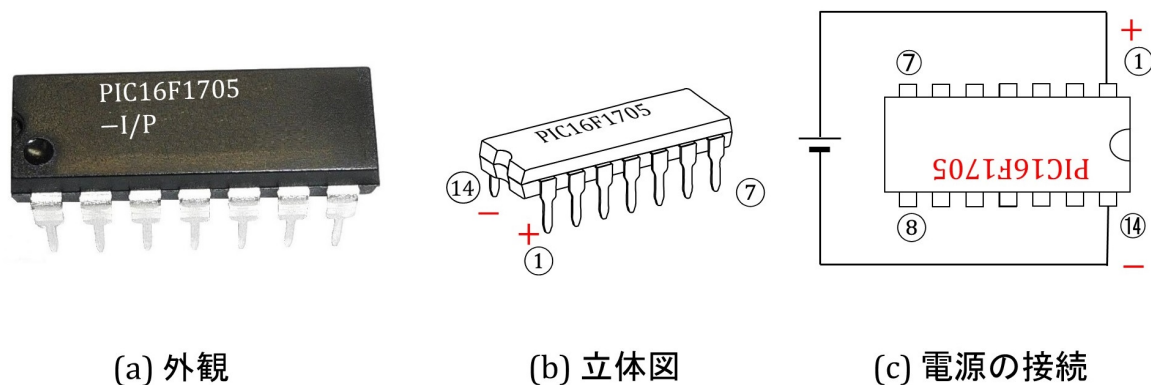


図 2.1: PIC16F1705

図 2.1 は PIC16F1705 の外観，立体図および電源接続の様子です。このマイコンは Microchip 社の製品です。PIC16F1705 の特徴は D-A 変換器と 2 個のオペアンプを持っていることです。D-A 変換器は 8 ビットです。これにより，256 段階の値を出力できます。ただし，D-A 変換器は電流を出力するようには設計されていません。10[μ A] の電流を取り出しただけでも D-A 変換器の出力電圧は低下してしまいます。そこで，マイコン内蔵のオペアンプを利用します。D-A 変換器の出力信号をオペアンプに通すことで，オペアンプからは最大 100[mA] の電流を取り出すことができます。もちろんオペアンプは汎用オペアンプとして様々な応用に使えます。

なお，文献 [5] [6] では，PIC16F1825 を使っています。このマイコンの特徴はフルブリッジインバータ用の相補型の PWM 信号を生成できることです。一方で，PIC16F1825

はD-A変換器を持っていません。このため、文献 [5] の正弦波信号を利用する製作課題ではPWM信号を使って正弦波を生成しています。PWM信号は、多くの高調波成分を含み、正弦波を用いた実験には必ずしも向いていません。

ブレッドボードで電気・電子回路実験がし易いように、PIC16F1705にはDIP(Dual In-line Package)と呼ばれるパッケージのものを選びました。電極(ピン)が2列に並んでいて、プリント基板、ブレッドボードに差し込めるタイプです。PIC16F1705は14ピンを持ち、これらのピン7個ずつが2列に配置されています。

ピン番号は、同図(b)の立体図の通り、パッケージに設けられた凹みを左に見て、その左下から反時計回りに①→②→…→⑭と付けられています。

このマイコンの場合、電源は図2.1(c)のように、①番ピンに電源の+側、⑭番ピンに電源の-側(GND: Ground)をつなぎます。

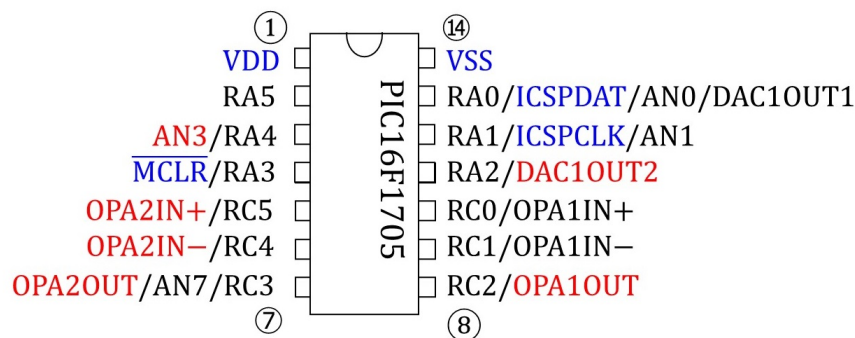


図 2.2: PIC16F1705 のピン配置

図2.2はPIC16F1825のピン配置を示します。②番ピンから⑬番ピンまでは複数の機能が割り当てられていて、プログラムにより選択できます。

RA_x, RC_x はI/O(Input/Output)ポートです。

AN_x はアナログ入力ピンであり、A-D変換モジュールをつなげられます。

DAC1OUT_x がD-A変換器の出力ピンです。

OPA_xIN_± がオペアンプの入力ピン、OPA_xOUT が出力ピンです。

MCLR は、Master Clear の略で、 $\overline{\text{MCLR}}$ とオーバーラインが付いているので、負論理入力です。このピン(4番ピン)は通常は5[V]を入力しておき、強制的にマイコンをリセット(初期化)したい場合に0[V]を入力します。その後5[V]を入力すると、マイコンはmain()関数の先頭にもどって再起動します。

ICSPDATとICSPCLKはICSPTM(In-Circuit Serial Programming)用のピンです。PICKitTM3, PICKitTM4, MPLAB SNAPなどのDebugger/Programmerを接続して、マイコンへのプログラム書き込み、デバッグを行うことができます。

図 2.2 において赤字のピンが本稿で利用するピンです。そして、青字のピンがマイコンへのプログラム書き込み、デバッグ用のピンです。

- 電源

この回路は、正と負の電源 ($V+$ と $V-$) を必要とします。これらは外部電源 E1, E2 から供給されます。電源には、2.5V の充電電池、もしくは3V の乾電池が適しています。

- 周波数指令電圧と正弦波の生成

外部からの周波数指令電圧 v_{FCOM} は AN3 ピンを通じて読み取られ、内部の A-D 変換器によりデジタル値に変換されます。この値は正弦波の周波数を設定するために使用されます。

マイコンは正弦波データを生成します。正弦波データは内部メモリにデジタル値として保持されていて、マイコンは指令電圧に基づく周期で正弦波データを順次 D-A 変換器に送ります。このデータ送付の周期を変えることで、正弦波の周波数を変えることができます。

- D-A 変換

正弦波データは、内蔵 D-A 変換器を通じてアナログ信号に変換され、DAC1OUT2 ピンから出力 (v_{DAC}) されます。

- オペアンプによる信号処理

オペアンプ 1 (ユニティゲインアンプ): DAC からの出力は、マイコン内部でオペアンプ 1 のマイナス入力に接続され、入力信号を 1 倍に増幅して OPA1OUT ピンから出力 (v_{OP1}) します。このユニティゲインアンプはバッファアンプとして働きます。D-A 変換器の出力電流容量は小さく、数 μA も流せませんが、オペアンプは最大で 100 mA 程度を出力できます。

オペアンプ 2 (ローパスフィルタ): オペアンプ 1 の出力は、抵抗 R1 を介してオペアンプ 2 に送られます。ここで、抵抗 R2 とコンデンサ C1 を使用したローパスフィルタが高周波ノイズを減衰させ、出力信号を滑らかにします (OPA2OUT ピン (v_{FIL}))。

- ICSP (In-Circuit Serial Programming)

ICSP コネクタを通じて、マイクロコントローラにプログラムを直接書き込むことが可能です。これにより、回路を完成させた後もソフトウェアの更新やデバッグが容易になります。ICSP は開発プロセスを大幅に加速させ、プロトタイピングや教育目的での利用において非常に便利です。

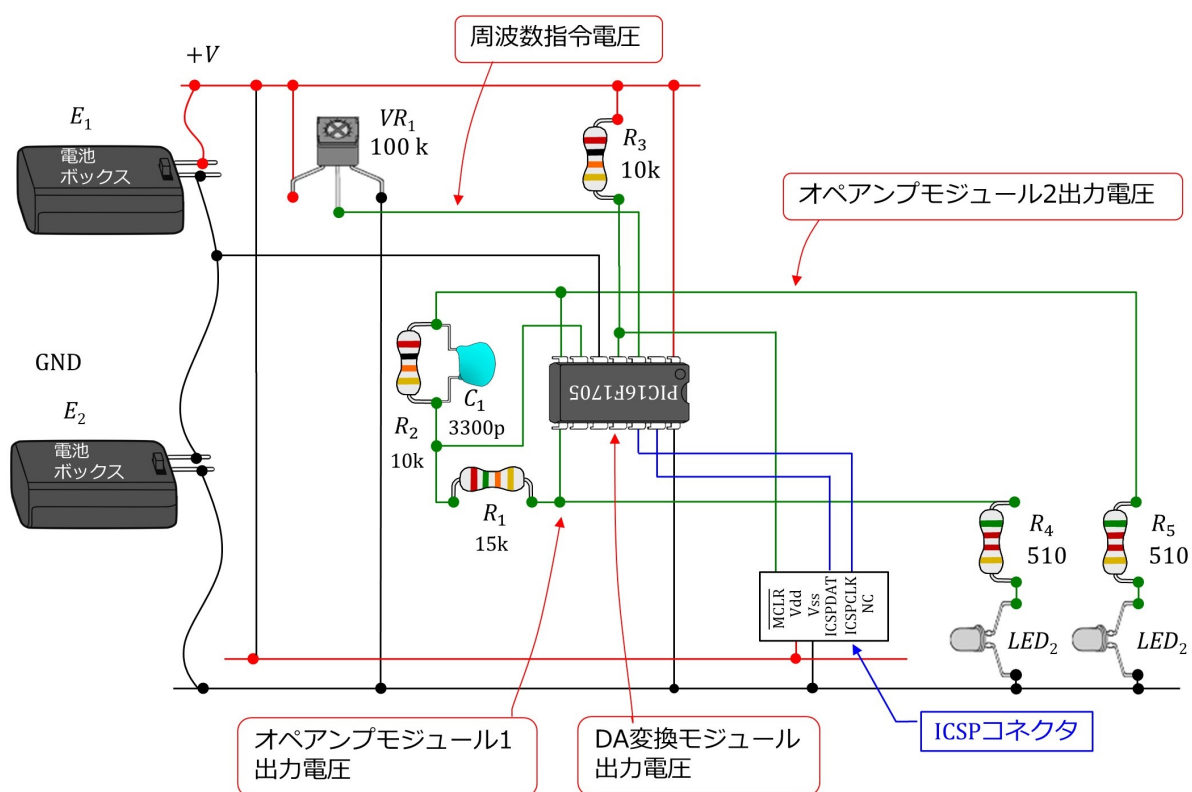


図 3.2: PIC16F1705 を用いた正弦波発生器実験回路の実体配線図

図 3.2 は正弦波発生器実験回路の実体配線図です。実体配線図（または 3D 配線図）は、実際の部品配置と接続を視覚的に表現することで、回路の構築を容易にします。以下に、実体配線図の利点をまとめます。

- 直感的な理解

実体配線図は、実際の部品の配置と接続を視覚的に示すため、回路図を読むことが難しい初心者にとっても理解しやすいです。

- 組み立ての容易さ

回路の組み立てを行う際に、どの部品がどこに配置され、どのように接続されるべきかが一目でわかるため、ミスを減らし、作業効率を向上させます。

- デバッグの効率化:

もし回路が期待通りに動作しない場合、実体配線図を参照することで問題点を迅速に特定し、修正することができます。

- 教育ツールとしての価値

教育現場や趣味のプロジェクトでの使用において、実体配線図は、電子工学の基礎から複雑な回路設計まで、幅広い知識を身に付けるのに役立ちます。

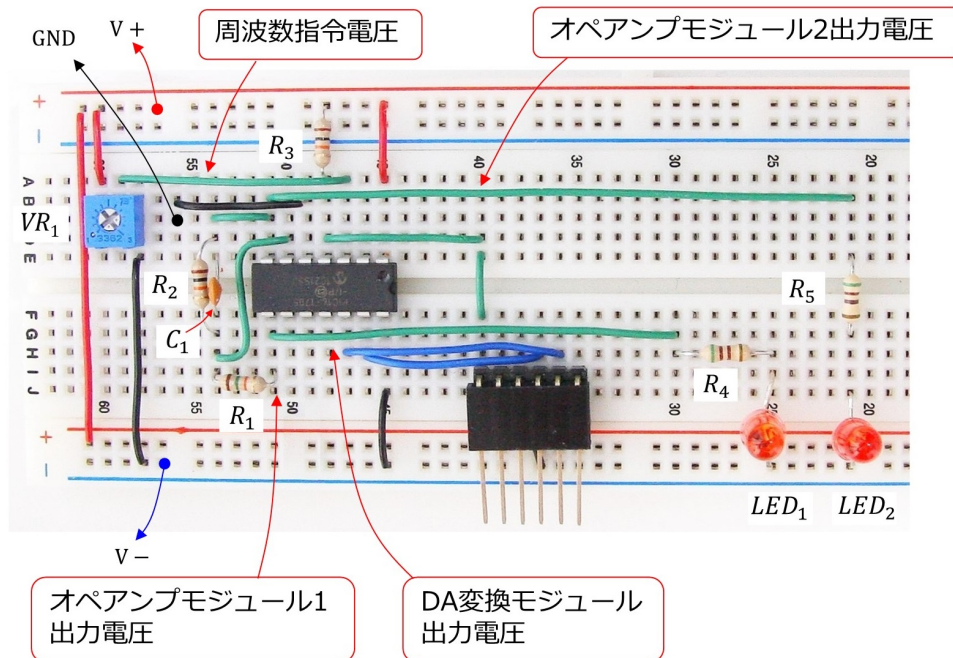


図 3.3: PIC16F1705 を用いた正弦波発生器の実験回路配線例

図 3.3 は正弦波発生器実験回路のブレッドボード上の配線例です。

第4章

ゼロからの ChatGPT によるプログラミング

本稿で解説するソースコードは、本稿と同じ

[パワーエレクトロニクスノート](#)

の「PIC16F1705用正弦波発生プログラム (ChatGPT でゼロからプログラミング) (圧縮ファイル)」に入れてあります。同フォルダをダウンロード・解凍して、「sin_wave_from_scratch_chatgpt.X」フォルダを MPLABXProjects フォルダ内に置いてください。MPLAB® X IDE により、ソースコードを使用できます。

4.1 得られたプログラム

ChatGPT 4 を活用して、PIC16F1705 マイクロコントローラのプログラミングを始めから終わりまで行いました。すなわち、初期化コードから始めて、ピン設定、周辺モジュール設定、そして正弦波を生成する関数まで、全てを ChatGPT との対話を通して作成しました。

対話の経緯は [4.4 節](#) にまとめてあります。また、プログラムを含む対話の全文は、とても長いので、本稿とは別に、

[オペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器](#)

の Web ページに掲載してあります。

ChatGPT との対話の最終版を以下に示します。図 3.1 の回路で、正弦波を v_{DAC} , v_{OP1} , v_{FIL} に出力するプログラムです。

```
#include <xc.h>

// CONFIG1
#pragma config FOSC = INTOSC // Oscillator Selection Bits (INTOSC oscillator: I/O function on CLKIN pin)
#pragma config WDTE = OFF // Watchdog Timer Enable (WDT disabled)
#pragma config PWRT = OFF // Power-up Timer Enable (PWRT disabled)
#pragma config MCLRE = OFF // MCLR Pin Function Select (MCLR/VPP pin function is digital input)
#pragma config CP = OFF // Flash Program Memory Code Protection (Program memory code protection is disabled)

#pragma config BOREN = ON // Brown-out Reset Enable (Brown-out Reset enabled)
#pragma config CLKOUTEN = OFF // Clock Out Enable (CLKOUT function is disabled. I/O or oscillator function on the CLKOUT pin)

#pragma config IESO = ON // Internal/External Switch Over (Internal External Switch Over mode is enabled)
#pragma config FCMEN = ON // Fail-Safe Clock Monitor Enable (Fail-Safe Clock Monitor is enabled)

// CONFIG2
#pragma config WRT = OFF // Flash Memory Self-Write Protection (Write protection off)
#pragma config PPS1WAY = ON // Peripheral Pin Select one-way control (The PPSLOCK bit cannot be cleared once it is set by software)
#pragma config PLLEN = ON // PLL Enable Bit (4x PLL is enabled when software sets the SPLLEN bit)
#pragma config STVREN = ON // Stack Overflow/Underflow Reset Enable (Stack Overflow or Underflow will cause a Reset)

#pragma config BORV = LO // Brown-out Reset Voltage Selection (Brown-out Reset Voltage (Vbor), low trip point selected.)

#pragma config LPBOR = OFF // Low-Power Brown Out Reset (Low-Power BOR is disabled)
#pragma config LVP = ON // Low-Voltage Programming Enable (Low-voltage programming enabled)

#define XTAL_FREQ 32000000 // 内部クロックを32MHzで使用する場合の定義です。

// 正弦波データ配列 (例として16点で1周期)
const unsigned char sine_wave[16] = {128, 176, 218, 245, 255, 245, 218, 176, 128, 79, 37, 10, 0, 10, 37, 79};
unsigned char i = 0; // ループカウンタとして割り込み内で使用

void __interrupt() isr(void) {
    if (TMR1IF) { // タイマ1の割り込みフラグをチェック
        TMR1IF = 0; // タイマ1の割り込みフラグをクリア

        // AD変換開始
        ADCON0bits.GO_nDONE = 1;
        // AD変換完了待ち
        while(ADCON0bits.GO_nDONE);

        // AD変換結果に基づいて割り込み周期を設定
        TMR1 = 65535 - (ADRES >> 2);

        DAC1CON1 = sine_wave[i]; // 正弦波の値をDACに設定
        i++; // インデックスをインクリメント
        if (i >= 16) i = 0; // インデックスが配列のサイズを超えたら0にリセット
    }
}
```

```
void main(void){
// OSCコントロール
OSCCON = 0xF0; // 内部オシレータを32MHzに設定し、PLLENをONにしてPLLを有効にします。

// DAC設定
DAC1CON0 = 0x90; // DACを有効にし、正のリファレンスをVDDに、出力はオペアンプに接続
DAC1CON1 = 0; // DACの出力値を0に初期設定

// オペアンプ設定
OPA1CON = 0xD2; // OPA1をUnity Gainモードで有効化,+入りにDAC outputを接続

// オペアンプ2設定
OPA2CON = 0xC0; // OPA2をHigh GBWPモードで起動し、非反転入力をOPA2IN+ピンに、
// 反転入力をOPA2IN-ピンに接続

// ADC設定
ANSELA = 0x10; // RA4ピンをアナログ入力に設定
ADCON0 = 0x0D; // ADCを有効にし、チャンネルをAN3(RA4)に設定、ADONビットを設定
ADCON1 = 0x20; // 左詰め結果を選択し、FOSC/32の速度で変換

// タイマ1設定
T1CON = 0x01; // タイマ1を1:1プリスケアラで起動
TMR1 = 65535 - ADRES; // 初期値設定、割り込みが発生するまでの時間
TMR1IE = 1; // タイマ1割り込みを有効化
PEIE = 1; // 周辺装置割り込みを有効化
GIE = 1; // グローバル割り込みを有効化

while (1){
// メインループでは何もしない
}
}
```

赤字の箇所は対話終了後に筆者が手直ししました。ADRES >> 2 は、A-D 変換結果を1/4倍して、正弦波の周波数範囲を少し狭める調整をしています。DAC1CON1 = 0x90は訂正です。対話の途中で筆者の訂正にChatGPTが応じていたのですが、その後の対話の中で元の誤設定に戻っていました。

4.2 周辺モジュールのブロック図

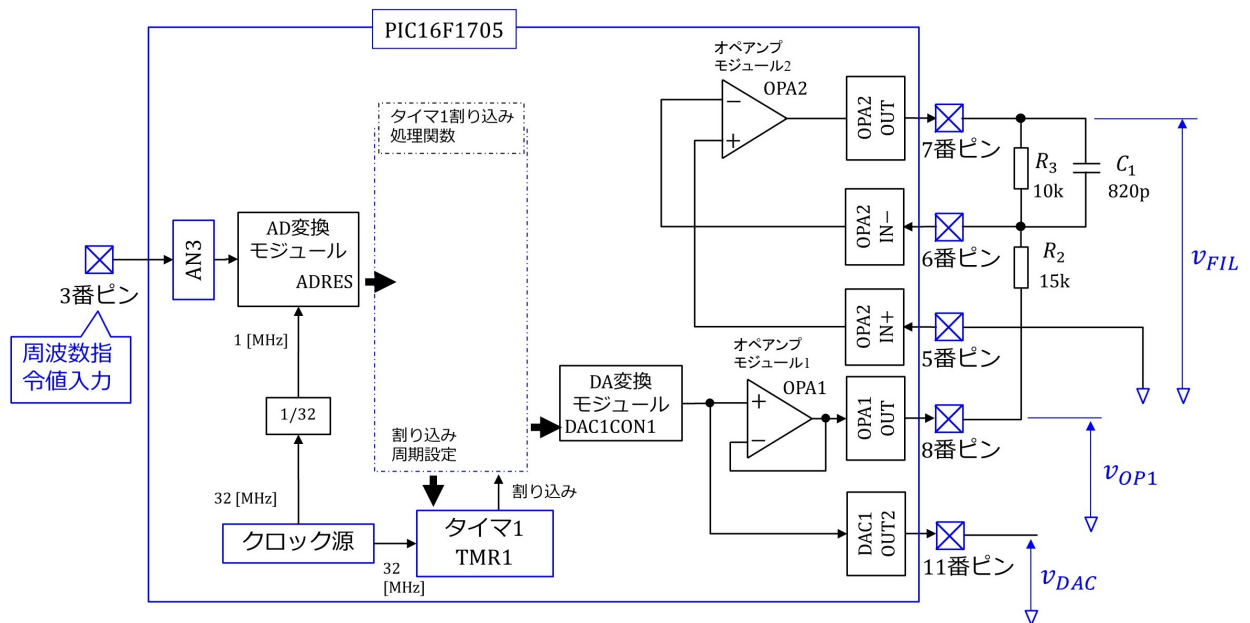


図 4.1: PIC16F1705 の周辺モジュールブロック図

図 4.1 は、前節のプログラムにより設定された周辺モジュールをブロック図にして示します。ブロック図は主に以下の部分で構成されています：

- 周波数指令値入力

周波数指令電圧が、マイコンの AN3 ピンを介して取り込まれます。この信号は A-D 変換モジュールによってデジタル化され、ADRES レジスタに格納されます。

- クロック源

マイコン内のクロック源が 32MHz で動作し、その信号は 1/32 の分周器を通して 1MHz の信号に逡減されます。この 1MHz の信号は、A-D 変換モジュールを駆動するクロック信号として使用されます。タイマ 1 には 32MHz のクロックがそのまま供給されます。

- 正弦波生成ロジック

タイマ 1 割り込み処理関数が正弦波信号を生成します。正弦波データは 16 個からなり、タイマ 1 による割り込みごとに、D-A 変換モジュールの DAC1CON1 レジスタ

に順次デジタルデータを送ります。そして、D-A変換モジュールがアナログ信号へと変換します。また、この割り込み処理関数は、ADRESレジスタの周波数指令値を読み出し、この値を用いて割り込み周期を決定し、タイマ1のTMR1レジスタに格納します。これにより、正弦波の周波数が設定されます。

- オペアンプ回路

D-A変換器からの出力信号は、2段のオペアンプ（OPA1, OPA2）を通じてさらに処理されます。最初のオペアンプ（OPA1）は、ゲイン1のアンプです。ボルテージフォロワとも呼ばれます。D-A変換モジュールの出力電流容量が小さいので、OPA1により電流容量を増大させます。2番目のオペアンプ（OPA2）は、抵抗 R_2, R_3 とコンデンサ C_1 によりフィルタとして機能し、信号から不要な高周波成分を除去します。

4.3 実験結果

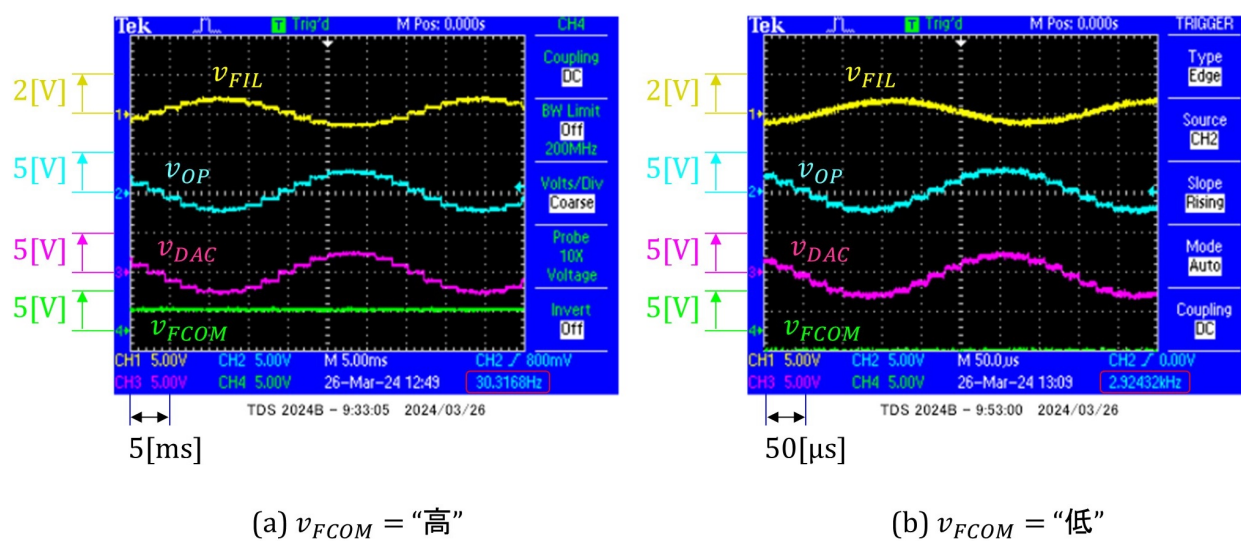


図 4.2: ゼロからのプログラミング結果の実験波形

図 4.2 は、ChatGPT との対話によるゼロからのプログラミング結果の実験波形例です。(a) は周波数指令電圧 v_{FCOM} が高い場合、(b) は低い場合です。 v_{FCOM} が高い場合の正弦波の周波数は約 30Hz、低い場合は約 2.9 kHz です。正弦波データ配列 `sin_wave[]` には 1 周期 16 点のデータが格納されています。データ数が 16 点と少ないため、正弦波形は階段状

になっています。周波数が高い場合には、オペアンプモジュール2のフィルタ効果により、 v_{FIL} の波形が滑らかになっています。

タイマ1割り込み処理関数 `interrupt()` `isr()` 関数は、割り込みの度にこの16点のデータを先頭から順次 D-A 変換器に送ります。16点を送り終わると、再び先頭データにもどって順次 D-A 変換器に送ることを繰り返します。

v_{FCOM} の値は、A-D 変換されて ADRES レジスタに格納されます。そして、

```
TMR1 = 65535 - (ADRES >> 2);
```

により、**タイマ1割り込み周期**が決定されます。TMR1はこの式の値からカウントアップして、オーバフロー時に割り込みを行います。そのため、ADRES の値が小さい (v_{FCOM} = "低") ほど割り込み周期は短くなり、正弦波周波数が高くなります。

4.4 ChatGPT との対話の経緯

以下、筆者の依頼・指示文を記します。ChatGPT 4 が回答したプログラムを含む対話の全文は、とても長いので、本稿とは別に、

オペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器
に掲載してあります。

- (1回目) (You) pic16f1705 で**正弦波を D-A 変換モジュールより出力するプログラム**を生成して。
- (2回目) (You) システムクロックを 32MHz にして、LVP は ON にして
- (3回目) (You) このプログラムと先のプログラムを統合して。
- (4回目) (You) **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。また、割り込み周期は 1ms にして。**
- (5回目) (You) このプログラムで、DAC の出力ピンを DACOUT2 ピンに設定して。
- (6回目) (You) PIC16F1705 では DAC 用レジスタは DACCON0, DACCON1 ではなくて、DAC1CON0, DAC1CON1 です。これに合わせて、このプログラムを修正してください。
- (7回目) (You) このプログラムで、DAC1CON0 の設定値とコメントが一致していません。設定値を 0xE4 ではなく、0x90 に修正してください。

- (8回目) (You) 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、A-D 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。
- (9回目) (You) このプログラムで、ADCON0=0x0D, ADCON1 = 0x20 に修正して。また、ZCD = OFF のコードは削除して。
- (10回目) (You) このプログラムで、タイマ1 割り込み処理関数内の TMR1 の式を $TMR1 = 65535 - ADRES;$ に変更して
- (11回目) (You) このプログラムで、sin_wave のデータ数を 200 点にして
- (12回目) 1つ前のプログラムにもどって、DAC の出力をオペアンプモジュール OPA1 の入力とするように変更して。また、OPA1 は Unity Gain に設定して。
- (13回目) (You) このプログラムで、OPA1 の入力を DAC の出力に変更して
- (14回目) (You) このプログラムで、OPA1CON=0xD2 に変更して
- (15回目) (You) このプログラムで、sine_wave のデータを 200 点にして。
- (16回目) (You) 1つ前のプログラムにもどって、タイマ1 を 1:1 プリスケアラで起動して
- (17回目) (You) このプログラムで、OPA2 を起動する設定に変更して。OPA2 は high GBBP mode とし、inverting input は OPA2IN-ピン、non-inverting input は OPA2IN+ピンに接続して
- (18回目) (You) OPA1 の使い方は変えないで OPA2 を追加で設定するようにして
- (19回目) (You) このプログラムで、OPA2CON = 0xC0; に変更して

以上の経緯で、前節のプログラムを得ることができました。対話の基本は、文章による依頼／指示と、ChatGPT のプログラム提示です。対話のポイントは、朱書きにした依頼文です。これだけで、ChatGPT はコードの改訂案を提示してくれました。

なお、令和6年3月時点では、ChatGPT は PIC マイコンの周辺モジュール設定に関する情報を十分には持っていません。筆者の黒字の指示文には、プログラムの誤り訂正が含まれています。

第5章

MCCで周辺モジュール設定後のChatGPTによるプログラミング

本章で解説するソースコードは、本稿と同じ

[パワーエレクトロニクスノート](#)

の「PIC16F1705用正弦波発生プログラム（ChatGPTでMCC設定後のプログラミング）（圧縮ファイル）」に入れてあります。

5.1 得られたプログラム

前章のゼロからのプログラミングでは、PIC16F1705の周辺モジュール設定に筆者の訂正指示が必要でした。この指示のためにはデータシートを読み込まなければならず、PICマイコンプログラミングの大きなハードルが解消されたとは、まだ言えない状況です。それならば、ゼロからではなく、MCCにより周辺モジュール設定を済ませておき、残りのコードをChatGPTとの対話で作成してみよう、という考えに至りました。

ちなみに、MCCの特徴、メリット、デメリットをChatGPT 4に尋ねたところ、以下の回答が返ってきました。

特徴

- [グラフィカルユーザーインターフェイス（GUI）](#)

MCCは、マイクロチップ社のPICマイクロコントローラ用の設定をグラフィカルに行えるツールです。これにより、ハードウェアの設定、ピン割り当て、周辺モジュールの設定などが直感的に行えます。

- [自動コード生成](#)

設定に基づいて、C言語のコードを自動で生成します。これにより、開発時間を大幅に短縮できます。

- **周辺モジュールライブラリ**

多くの内蔵周辺モジュールに対応したライブラリを持っており、それぞれの機能に合わせたコードの生成が可能です。

- **MPLAB X IDE との統合**

MCCはMPLAB X IDEと完全に統合されており、開発の全過程を一つの環境内で完了できます。

メリット

- **開発時間の短縮**

初期設定や周辺モジュールの設定にかかる時間が大幅に短縮されます。

- **エラーの削減**

GUIを使用した設定は、手動でのコード入力時に比べてエラーが発生しにくいです。

- **学習曲線の緩和**

初心者でも容易にPICマイクロコントローラを使い始めることができます。

- **プロジェクトの再利用性の向上**

生成されたコードや設定は他のプロジェクトにも再利用しやすいです。

デメリット

- **カスタマイズの制限**

自動生成されたコードは、特定のパターンに従っているため、高度なカスタマイズが必要な場合は制限があります。

- **リソース使用量**

自動生成されるコードは最適化されていない場合があり、リソース使用量が多くなることがあります。

- 学習の遮断

初心者がMCCに頼りすぎると、低レベルのコードに対する理解が深まらない可能性があります。

- 対応していない機能やデバイス

全てのPICマイクロコントローラや周辺モジュールがMCCに対応しているわけではありません。

ChatGPT 4との対話により得られたプログラムを以下に示します。図3.1の回路で、正弦波を v_{DAC} , v_{OP1} , v_{FIL} に出力するプログラムです。MCCによる周辺モジュール設定は付録Aを参照してください。

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

// 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 TMR1_ISR(void)
{
    DAC1CON1 = 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
    ADC_GetConversion(0x03);

    // Adjust Timer1 interval based on ADC result
    unsigned int newTimer1Value = 65535 - (ADRES >> 2); // Example scaling, adjust as necessary
    TMR1 = newTimer1Value;

    TMR1IF = 0; // Clear Timer1 interrupt flag
}

void main(void)
{
    // initialize the device
    SYSTEM_Initialize();

    while (1)
    {
        // Add your application code
    }
}
```

これは、main.c ファイル内のコードです。TMR1_ISR() 関数がタイマ1割り込み処理関数です。青字の部分（TMR1_ISR() 関数の先頭行と後尾の TMR1IF = 0; の行）が、MCCにより tmr1.c ファイル内に自動生成されたものです。これを main.c ファイル内へとカットアンドペーストし、青字の間の黒字のコードを ChatGPT との対話により作成しました。

赤字の箇所は、ChatGPTが呈示した案から、最後に筆者が書き直しました。ADC_GetConversion()関数は、A-D変換の起動、変換終了待ちを行う関数です。0x03はA-D変換モジュールへの入力ピンをAN3に指定します。この関数はMCCによりadc.cファイル内に自動生成されています。

ADDRESS >> 2は、周波数指令電圧に対する感度を調整しています。

5.2 周辺モジュールのブロック図

周辺モジュールのブロック図は図 4.1 と同じです。

5.3 実験結果

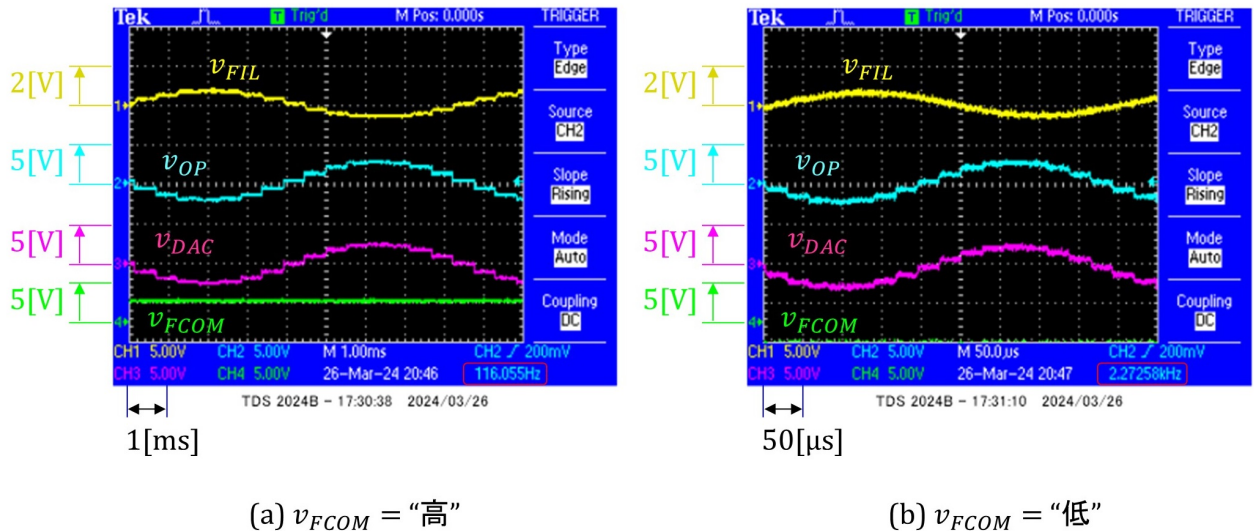


図 5.1: MCC により周辺モジュール設定後のプログラミング結果の実験波形

図 5.1 は、MCC により周辺モジュール設定後に、ChatGPT 4 との対話によりプログラミングを行った結果の実験波形例です。(a) は周波数指令電圧 v_{FCOM} が高い場合、(b) は低い場合です。図 4.2 の波形に近い結果が得られました。

5.4 ChatGPT との対話の経緯

以下、筆者の依頼・指示文を記します。ChatGPT 4 が回答したプログラムを含む対話の全文は、本稿とは別に、

オペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器

に掲載してあります。

(1回目) (You) pic16f1705 で、正弦波を D-A 変換モジュールより出力するプログラムを生成して

(2回目) (You) 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

(3回目) (You) 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，A-D 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．

以上，3回の対話で目的のプログラムを得ることができました．ChatGPT は，周辺モジュールの設定コードも提示してくれますが，その正誤を無視して，正弦波発生とその周波数調整に関するコードのみに注目しました．

第 A 章

付録

A.1 New Project の作成方法

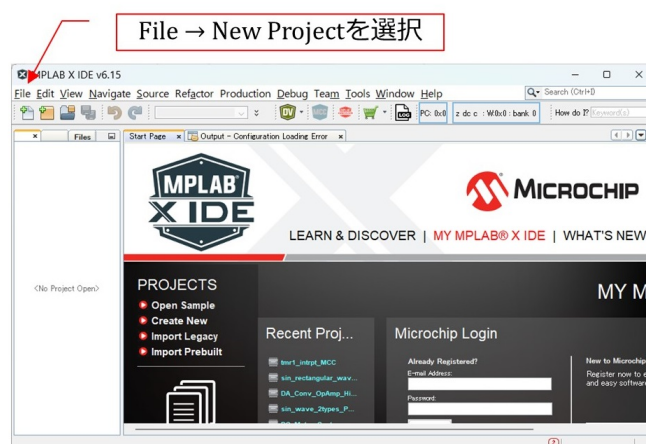


図 A.1: New Project の作成開始

新しいプロジェクトを立ち上げる手順について説明します。MPLAB[®] X IDE のアイコンを左ダブルクリックすることで、統合開発環境を立ち上げられます。図 A.1 の画面が立ち上がったら、File → [New Project](#) を選択します。

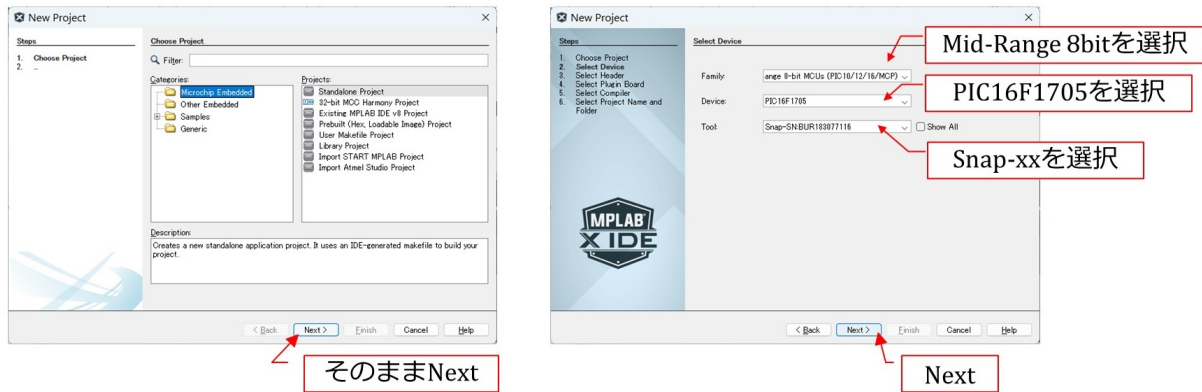


図 A.2: Device 選択

次に図 A.2 のように進み、Device に PIC16F1705 を選択します。

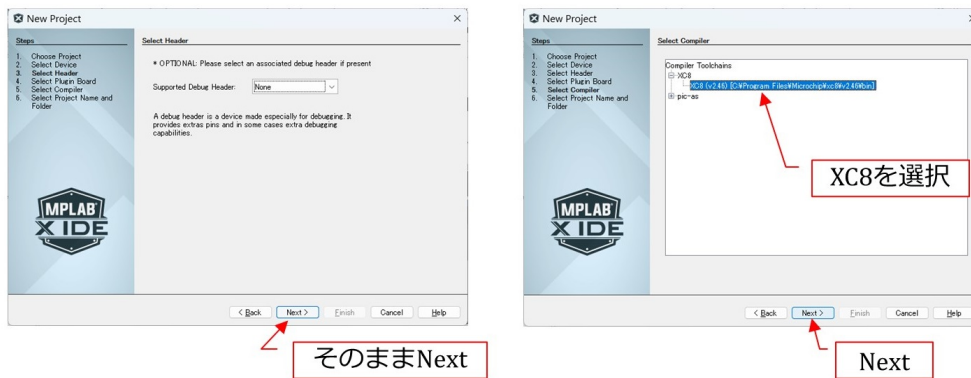


図 A.3: Compiler 選択

その後は図 A.3 のようにコンパイラ (Compiler) に XC8(v2.46) を選択します。

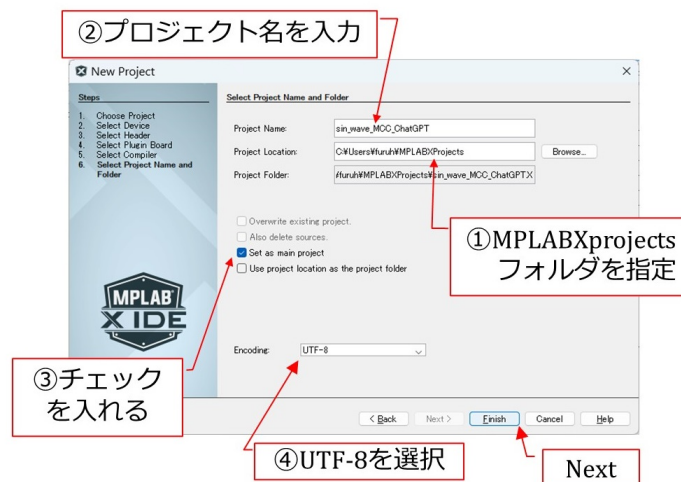


図 A.4: Project Name, コード選択

図 A.4 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、¥MPLABXProjects とします。そして、Project Name を、例えば、sin_wave_MCC_ChatGPT とします。文字コード (Encoding) に UTF-8 を選択して、Finish ボタンをクリックします。

A.2 MCC の起動



図 A.5: MCC の起動

図 A.4 までの手順により、新しいプロジェクトの設定が完了すると、図 A.5 の画面になります。画面左上には sin_wave_MCC_ChatGPT のプロジェクトツリーが表示されています。

す。この時点では、ツリー内のフォルダは空です。MCC 起動ボタンを左クリックします。

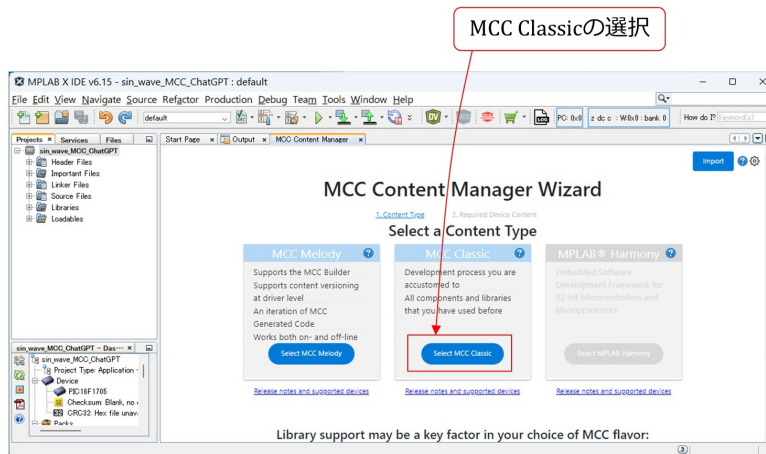


図 A.6: MCC Classic の選択

図 A.6 の画面が現れたら、MCC の Content Type を選びます。PIC16F1705 の場合は MCC Melody と MCC Classic が選べます。ただし、令和 6 年 3 月の時点で、筆者のパソコンでは (?), MCC Melody を立ち上げることができませんでした。そこで、MCC Classic を選択しました。

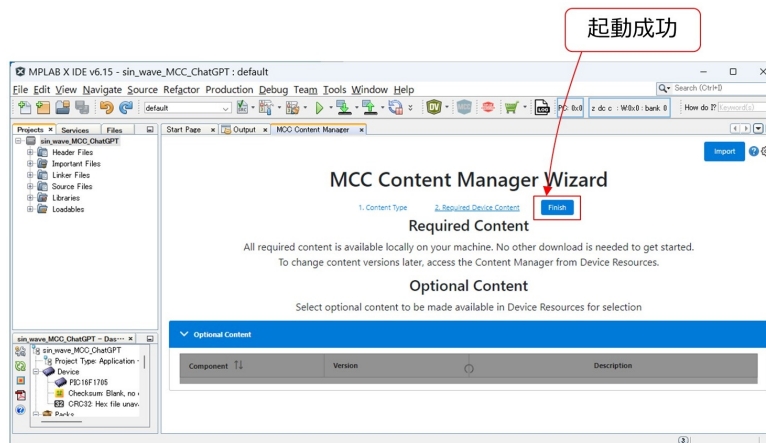


図 A.7: MCC の起動成功

図 A.7 の画面に切り替わったら、Finish ボタンをクリックします。

A.3 システムモジュール設定

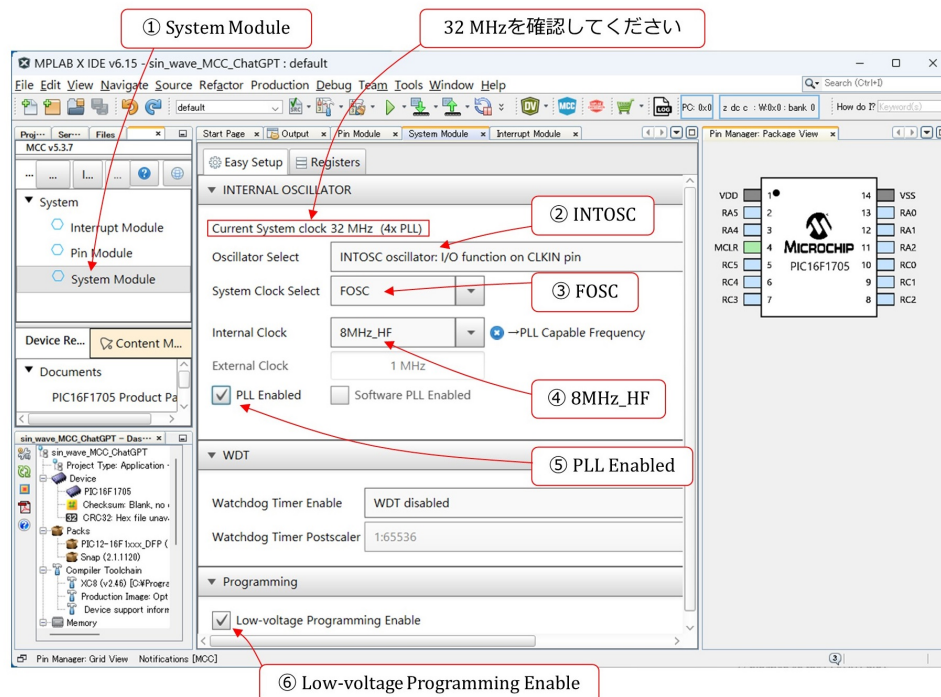


図 A.8: システムモジュール設定

まず、**System Module** の設定から始めます。図 A.8 のように、画面左上段の Project Resources エリア内の System Module の文字を左クリックします。すると、画面中央の Composer エリアに INTERNAL OSCILLATOR 等の設定画面が開かれ、右側の Pin Manager エリアにピン配置のパッケージ表示が示されます。以下の手順で設定を進めます。

(2) INTOSC oscillator を選択

マイコン内蔵のオシレータ (INTOSC oscillator) を使用します。本稿の製作ではクロックに高い精度を必要としないので、マイコン内蔵の精度の低い ($\pm 1\%$) クロックを使います。精度の高いクロックを必要とする場合には、マイコンに水晶発振子を外付けする方法があります。

(3) FOSC を選択

システムクロック源を FOSC とします。

(4) 8MHz_HF を選択

内蔵オシレータのクロック周波数を8MHzとします。この周波数を選ぶと4通倍PLLが使用可能になります。選択肢には16MHzがありますが、16MHzでは4通倍PLLは使えません。

(5) PLL Enabled をクリック

4通倍PLLを使って、内蔵オシレータのクロック周波数を4倍にできます。これにより、FOSCを $4 \times 8 \text{ MHz} = 32 \text{ MHz}$ とします。この周波数がこのマイコンの最高動作周波数です。

(6) Low-voltage Programming Enable にチェックマークが入っていることを確認

Low-voltage Programming(LVP)は、デフォルトでチェックマークが入っています。MPLAB[®] SNAPはLVPでしか使えません。このチェックマークを外して、MPLAB[®] SNAPでマイコンへの書き込みを行おうとすると、エラーが出ます。チェックは外さないでください。

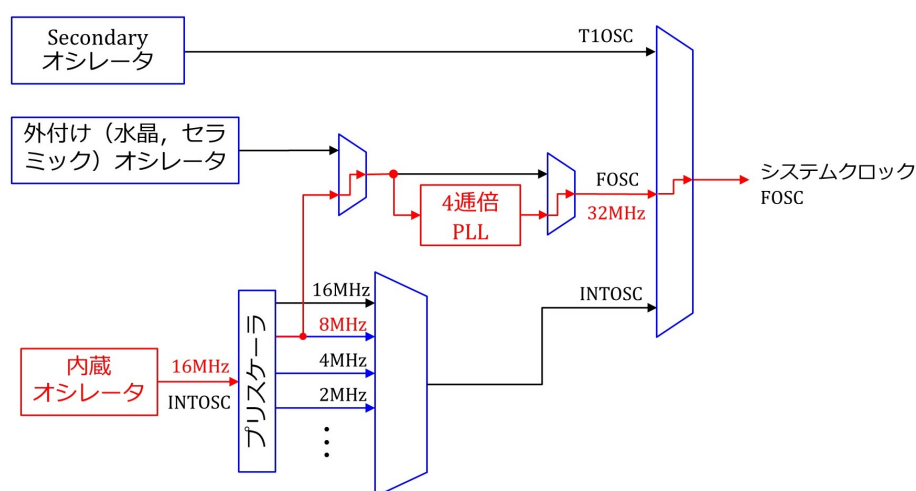


図 A.9: クロックソースのブロック図

以上の設定により、システムクロックが定まります。システムクロックは周辺モジュールに供給されます。図A.9は、クロックソースのブロック図です。上記設定により定まったシステムクロックの生成経路を赤線で示します。内蔵オシレータINTOSCは16MHzのクロックを生成します。Internal Clockの8MHz設定により、プリスケール出力が8MHzのクロックになります。そして、4通倍PLLにより、この8MHzクロックは、4通倍されて32MHzのクロックになります。最後に、FOSC選択により、システムクロックはPLL出力の32MHzクロックになります。

A.4 タイマ 1 モジュールの設定

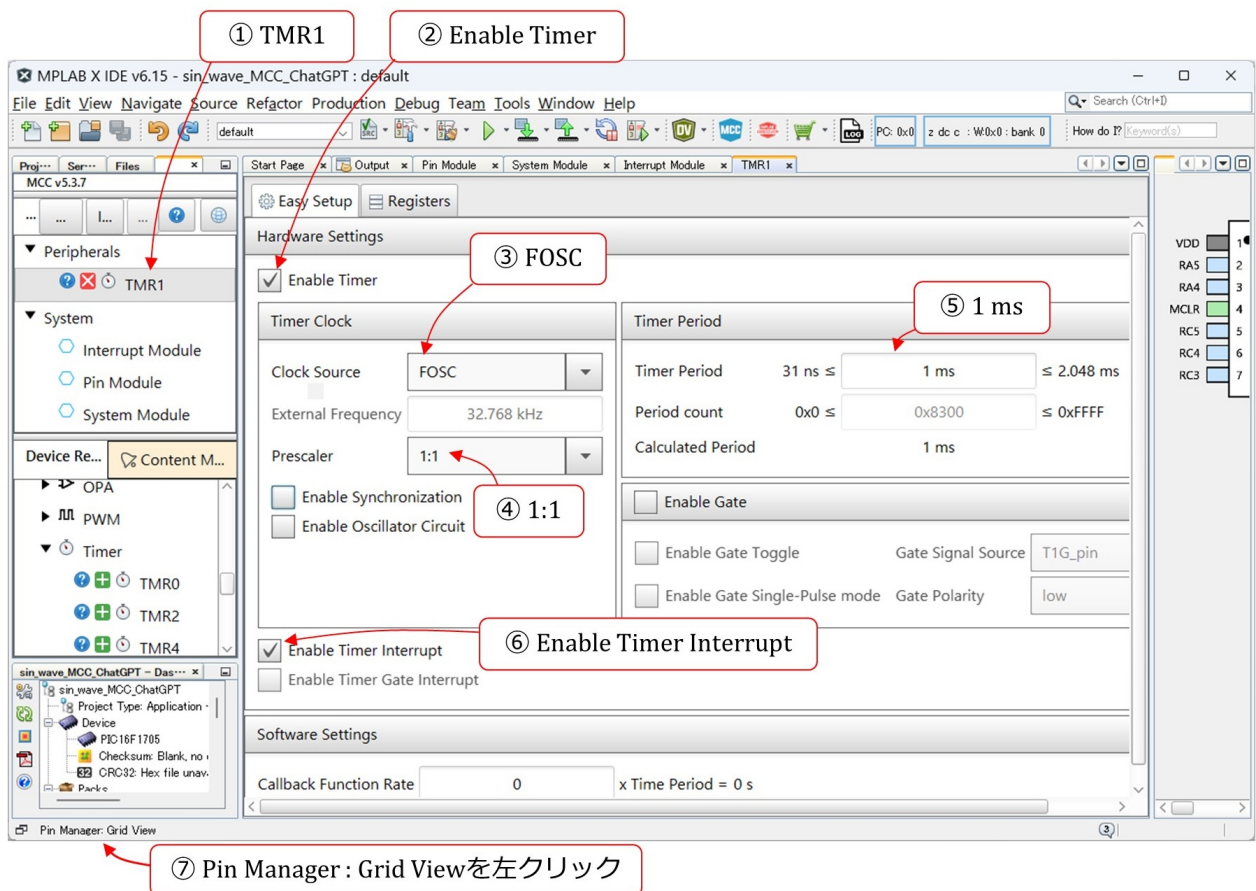


図 A.10: タイマ 1 モジュールの設定

図 A.10 はタイマ 1 の設定画面です。画面左中段の Device Resources エリアの **TMR1** を選択すると、左上段の Project Resources エリアに TMR1 の文字が現れ、画面中央の Composer エリアにタイマ 1・モジュールの設定項目が表示されます。一定時間間隔でタイマ 1 による割り込みを発生させるように、タイマ 1 を設定します。設定手順は以下のとおりです。

(2) Enable Timer を左クリック

タイマ 1 モジュールを起動します。

(3) FOSC を選択

タイマ 1 モジュールのクロック周波数を FOSC(= 32 MHz) とします。

(4) 1:1 を選択

クロックを 1/1 に分周します。クロック周波数は FOSC のまま不変です。

(5) 1 ms に設定

タイマ1による割り込み周期をとりあえず 1 [ms] に設定しておきます。TMR1レジスタには、初期値として 0x8300 が書き込まれます。TMR1は16ビットレジスタです。クロックによりカウント・アップして、オーバーフローしたところで割り込み処理関数を起動します。オーバーフローまでのカウント・アップ回数は

$$\begin{aligned} 2^{16} - 0x8300 &= 2^{16} - 8 \times 16^3 - 3 \times 16^2 \\ &= 32000 \end{aligned} \quad (\text{A.1})$$

です。FOSC = 32 MHz なので、オーバーフローまでの所要時間は 1 ms です。

(6) Enable Timer Interrupt を左クリック

タイマ1による割り込み機能を起動します。

(7) Pin Manager: Grid View を左クリック

Pin Manager エリア：テーブル表示を開きます。

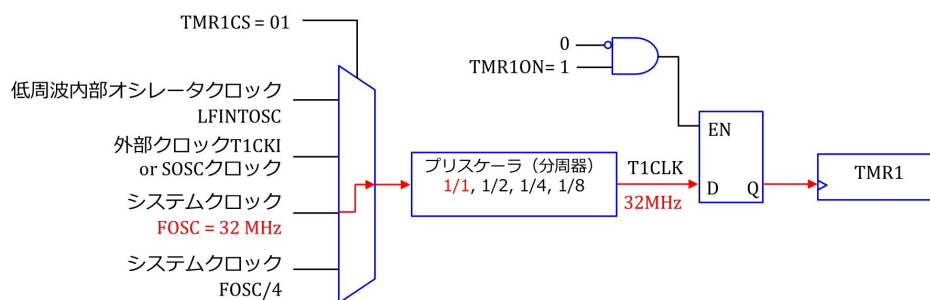


図 A.11: タイマ1のクロック源のブロック図

図 A.11 は、タイマ1のクロック源（クロックソース）のブロック図（抜粋）です。上記の設定により、タイマ1モジュールにおけるクロックの経路が図示のようになります。システムクロック FOSC (= 32 MHz) が取り込まれ、プリスケータにより 1 倍されます。ENABLE Timer により、TMR1ON = 1 となり、TMR1 レジスタに 32 MHz のクロックが供給されます。

A.5 A-D 変換モジュールの設定

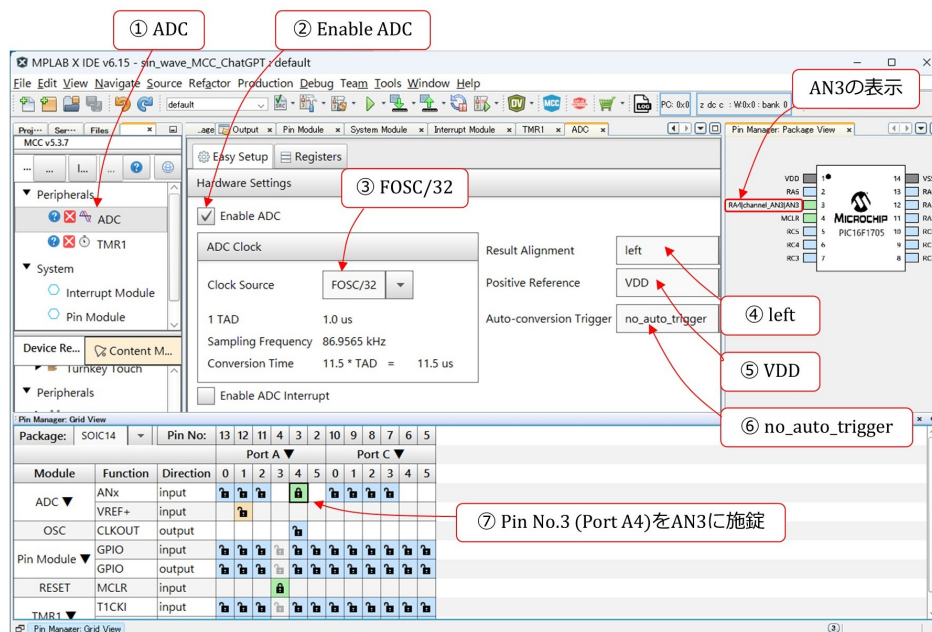


図 A.12: A-D 変換モジュールの設定

図 A.12 のように、左上段の Project Resources エリアの Peripherals 欄に ADC の文字が表示され、中央の Composer エリアに A-D 変換モジュールの設定項目が表示されます。以下の手順で、設定を行ってください。

(2) Enable ADC を左クリック

A-D 変換モジュールを起動します。

(3) FOSC/32 を選択

A-D 変換モジュールのクロックをシステムクロックの $1/32$ に設定します。データシートによると、PIC16F1705 の A-D 変換モジュールのクロック周期 TAD は、 $1TAD \geq 1 [\mu s]$ を満たさなければなりません。FOSC = 32 MHz なので、FOSC/32 により $1TAD = 1 [\mu s]$ とします。

(4) left を選択

PIC16F1705 の A-D コンバータは 10 ビットです。left を選択することで、A-D 変換結果は、16 ビットのレジスタに左寄せで得られます。

(5) VDD を選択

A-D 変換の正側の基準値を VDD（1 番ピンの入力電圧値）とします。

(6) VSS を選択

負側の基準値を VSS（14 番ピンの入力電圧値）とします。以上の基準値設定により、A-D コンバータは、VDD - VSS 間の電圧を 10 ビットのデジタル値に変換します。

(7) Pin No.38(Port A4) の ANx 欄を左クリック

3 番ピン (Port A4) をアナログ入力ピンに設定します。ANx 欄の当該箇所を左クリックすると、錠の画が施錠状態に変わります。この変化と同時に、右上のパッケージの 3 番ピンに AN3 の文字が表示されます。

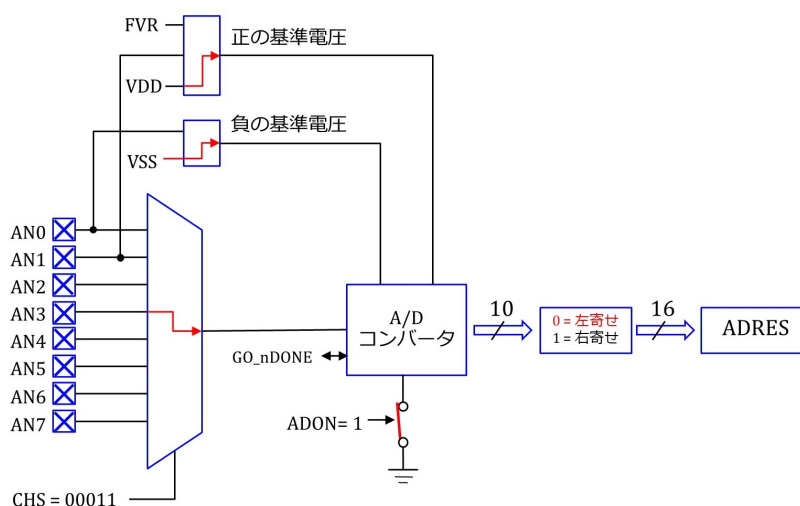


図 A.13: A-D 変換モジュールのブロック図

図 A.13 は、A-D 変換モジュール設定結果のブロック図です。以下に関連用語を説明します。

- CHS

入力チャネルを選定します。この設定では AN3 を入力とします。5.1 節のプログラムにおいて、ADC_GetConversion(0x03) 関数の引数 0x03 は CHS ビットを設定します。

- GO_nDONE

4.1 節のプログラムにおいて、A-D 変換開始のコードは、GO_nDONE = 1 です。A-D 変換が終了すると、GO_nDONE = 0 とリセットされます。while 文により、A-D 変換

換が終了するまで待ちます。5.1 節のプログラムの ADC_GetConversion(0x03) 関数も同じ操作・待機をします。

- **ADRES**

10 ビットの変換結果が 16 ビットの ADRES レジスタに格納されます。ADRES レジスタは、8 ビットの ADRESH と ADRESL の 2 個のレジスタからなります。left (左寄せ) の場合、変換結果の上位 8 ビットが ADRESH の全 8 ビットに格納され、変換結果の下位 2 ビットが ADRESL の上位 2 ビットに格納されます。

A.6 D-A 変換モジュールの設定

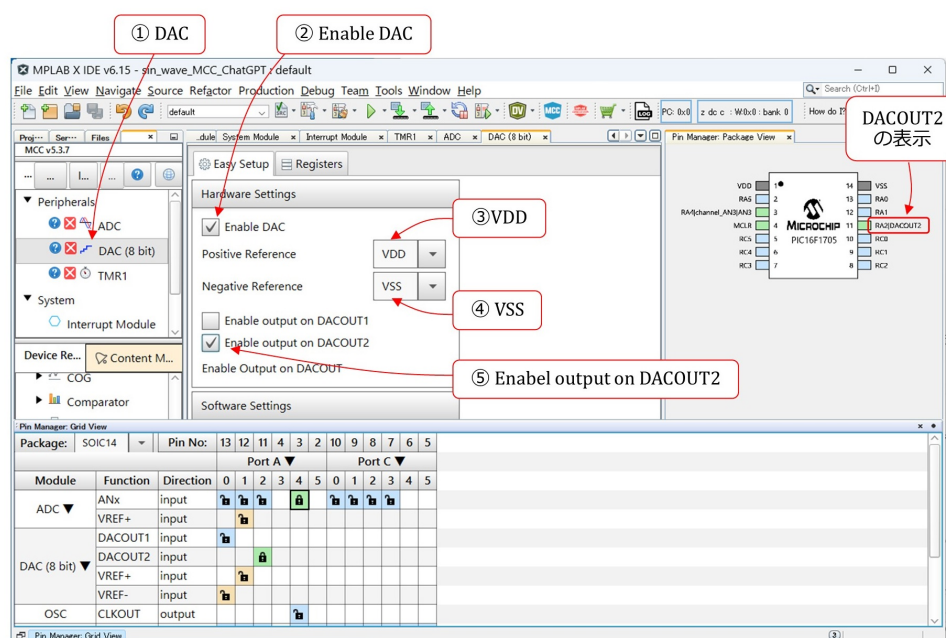


図 A.14: D-A 変換モジュールの設定

図 A.14 は、D-A 変換モジュールの設定画面です。以下の手順で設定します。

(2) Enable DAC

DAC を有効にするオプション。このチェックボックスを選択することで、D-A 変換モジュールがオンになります。

(3) VDD

正側の参照電圧として選択します。DAC が生成するアナログ信号の最大値がこの電圧になります。

(4) VSS

負側の参照電圧として選択します。これは DAC が生成するアナログ信号の最小値を示し、通常はグラウンド (0V) に設定されます。この設定により、アナログ出力の範囲が 0V から VDD までの範囲に定められます。

(5) Enable output on DACOUT2

DACOUT2 の出力を有効にするオプションをチェックします。これにより、DACOUT2 ピンからアナログ信号が出力されます。マイクロコントローラのピン配置図には、このピンが RA2/DACOUT2 としてマークされます。

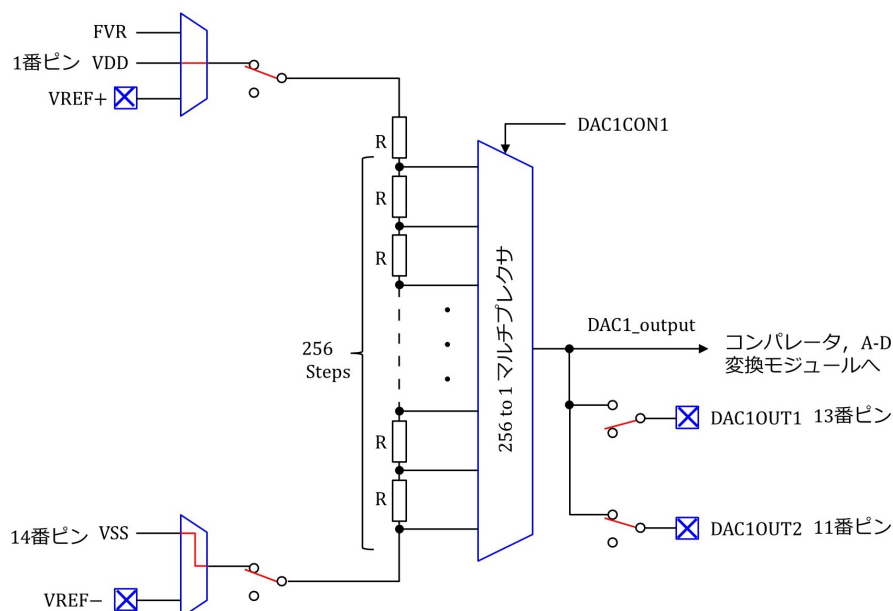


図 A.15: D-A 変換モジュールのブロック図

図 A.15 は、D-A 変換モジュール設定結果のブロック図です。赤線がモジュールの設定を示しています。各設定とマルチプレクサの働きについて以下にまとめます。

● モジュール設定

- 参照電圧 (VREF+, VREF-)

正の参照電圧 (VREF+) は VDD に接続されています。これは DAC が出力できる最大電圧を VDD に設定することを意味します。負の参照電圧 (VREF-) は VSS (通常はグランド) に接続されています。これは DAC の出力電圧範囲の最小値を 0V に設定することを意味します。

- DAC の分解能

DAC は 256 ステップの分解能を持っています。これは、入力デジタル値に基づいて DAC が生成できる異なる電圧レベルの段数を示しており、DAC の精度を表します。

- DAC1CON1 レジスタ:

このレジスタへの値の書き込みにより、DAC の出力電圧が決定されます。

- マルチプレクサの働き

マルチプレクサは、複数の入力信号の中から 1 つを選択して、単一の出力チャンネルに送ることができるデバイスです。このブロック図では、マルチプレクサの出力が DAC1.output です。DAC1OUT1 (13 番) ピンと DAC1OUT2 (11 番) ピンが出力オプションとしてあります。この設定では DAC1OUT2 ピンが選択され、アナログ信号の出力に使用されます。なお、この DAC1OUT1, DAC1OUT2 はデータシートの表記です。MCC では DACOUT1, DACOUT2 と表記されています。

A.7 オペアンプモジュール1の設定

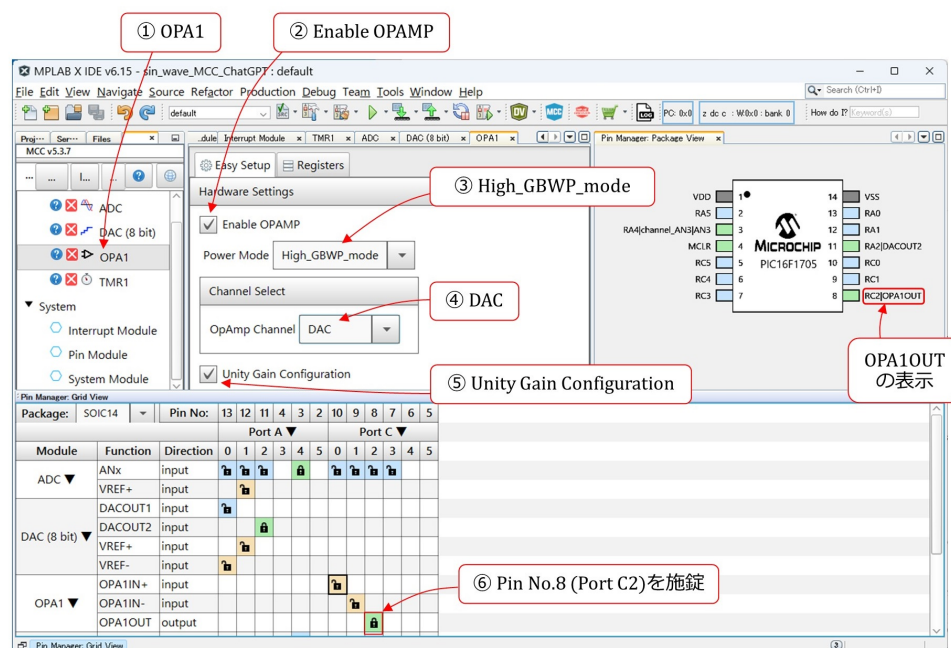


図 A.16: オペアンプモジュール1の設定

図 A.16 は、オペアンプモジュール1の設定画面です。以下の手順で設定します。

(2) Enable OPAMP

OPAMP を有効にします。

(3) High_GBWP_mode

オペアンプの動作モードを **High GBWP mode**（高利得帯域幅積モード）に設定します。このモードは、高い利得帯域幅積を必要とするアプリケーションに適しており、オペアンプがより広い周波数範囲で効果的に動作することを可能にします。

(4) DAC

オペアンプのチャンネル選択でDACを指定します。オペアンプの入力信号としてDACの出力を使用します。

(5) Unity Gain Configuration

オペアンプを**ユニティゲイン**に設定します。これは、オペアンプの利得が1であることを意味します。この設定は、信号のバッファリングやインピーダンス変換に使用します。

(6) Pin No.8 (Port C2) を施錠

出力ピンとして C2 (8 番) ピンを施錠します。オペアンプの出力はこのピンから提供されます。画像の右側のパッケージ図に OPA1OUT の文字が表示されます。

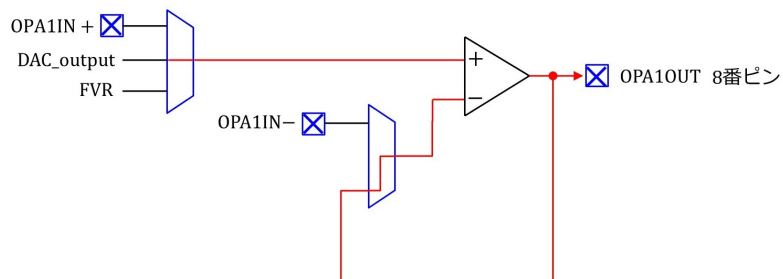


図 A.17: オペアンプモジュール1のブロック図

図 A.17 は、オペアンプモジュール1 設定結果のブロック図です。赤線がモジュールの設定経路を示しています。

オペアンプの+入力が DAC_output につながれ、-入力がオペアンプの出力につながれています。これにより、この回路は **Unity Gain Amp** として機能します。本稿での用途は、**インピーダンス変換**です。すなわち、この回路は入力インピーダンスがとても大きく、出力インピーダンスが小さいです。

D-A 変換モジュールの出力電流容量は小さく、数 μA の電流でも、出力電圧が歪んでしまいます。入力インピーダンスの大きな Unity Gain Amp は、入力にほとんど電流をとりません。したがって、このアンプを D-A 変換モジュールの出力につなげば、D-A 変換モジュールの出力電圧は歪みません。一方、このアンプは 100 mA の電流を出力しても、出力電圧は歪みません。

A.8 オペアンプモジュール 2 の設定

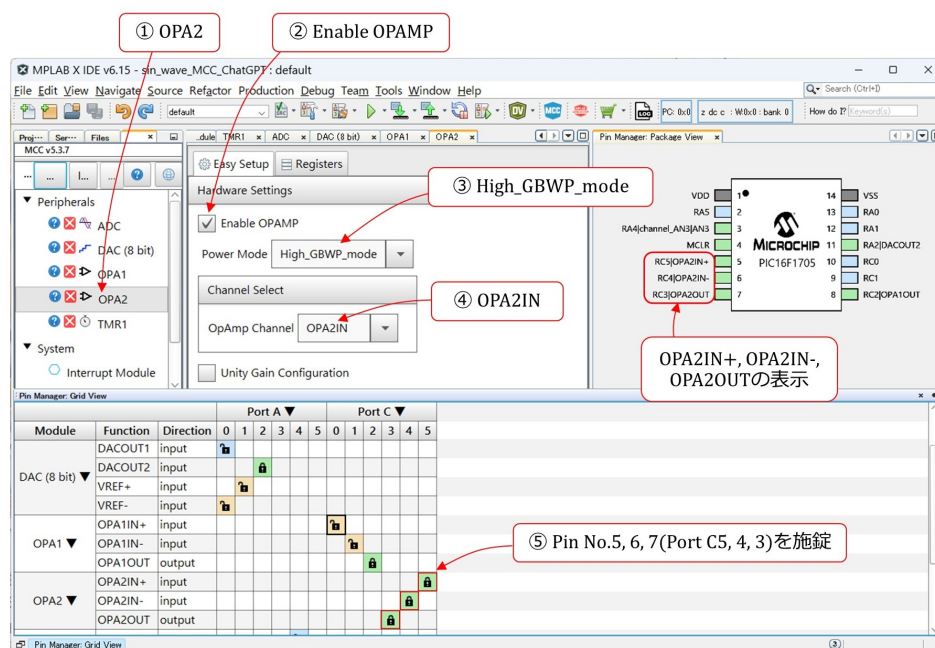


図 A.18: オペアンプモジュール 2 の設定

図 A.18 は、オペアンプモジュール 2 の設定画面です。以下の手順で設定します。

(2) Enable OPAMP

OPAMP を有効にします。

(3) High_GBWP_mode

オペアンプの動作モードを High GBWP mode (高利得帯域幅積モード) に設定します。

(4) OPA2IN

オペアンプの+入力を OPA2IN ピンにつなぐ指定です。

(5) Pin No.5, 6, 7 (Port C5, 4, 3) が施錠されていることを確認

上のオペアンプのチャンネル選択で OPA2IN を選択すると、これらのピンが施錠されます。画像の右側のパッケージ図に OPA2IN+, OPA2IN-, OPA2OUT の文字が表示されます。

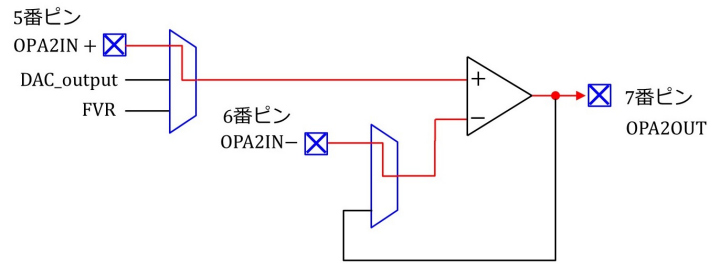


図 A.19: オペアンプモジュール2のブロック図

図 A.19 は、オペアンプモジュール2 設定結果のブロック図です。赤線がモジュールの設定経路を示しています。このオペアンプ回路の機能は5, 6, 7 番ピンに外付けされる抵抗, コンデンサなどの素子によって決まります。

A.9 周辺モジュール設定関数の自動生成

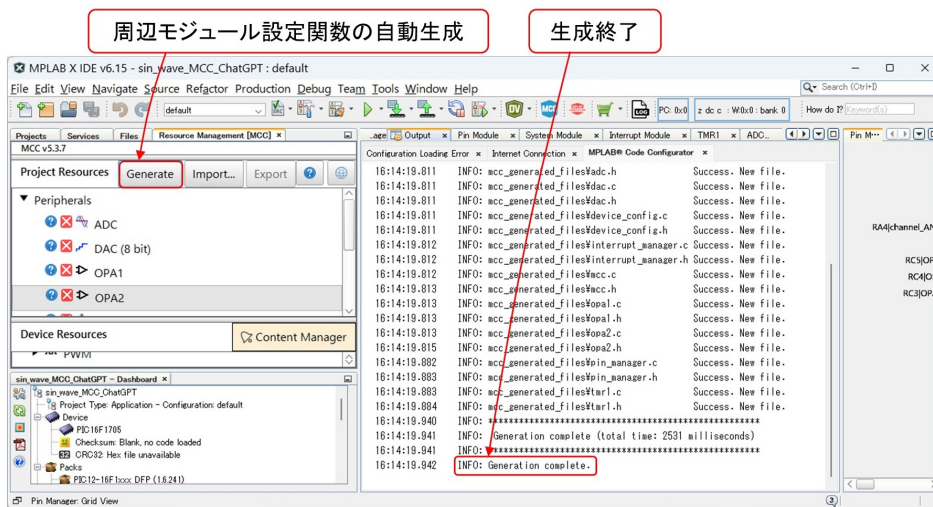


図 A.20: 周辺モジュール設定関数の自動生成

前節までで、周辺モジュールの設定を終えたので、それら設定関数の自動生成を実行します。図 A.20 はその自動生成を行った画面です。画面左上段の Project Resources エリアの **Generate** ボタンを左クリックします。中央上の Composer エリアに “Generation complete.” の文字が現れれば、生成終了です。

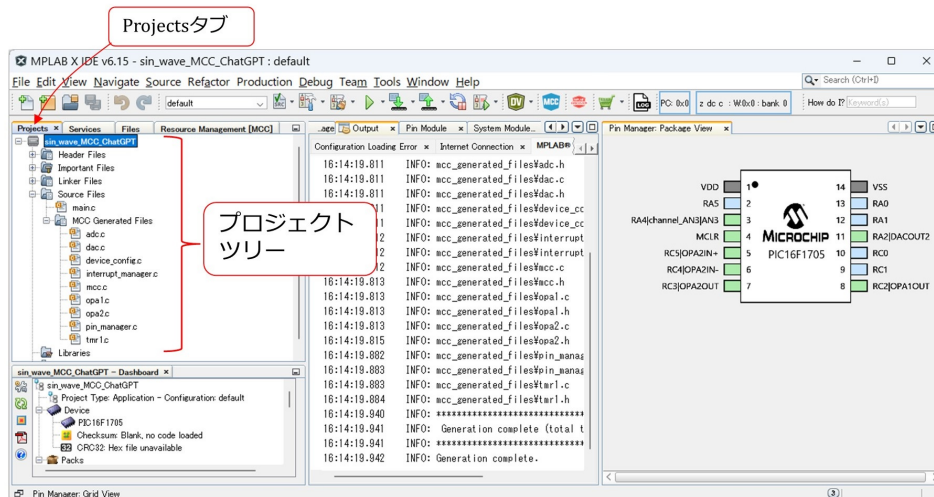


図 A.21: 自動生成されたプロジェクトツリー

図 A.21 のように、Project Resources エリアの **Projects** タブを左クリックすると、同エリア内に自動生成された**プロジェクトツリー**が表示されます。

A.10 追加の設定

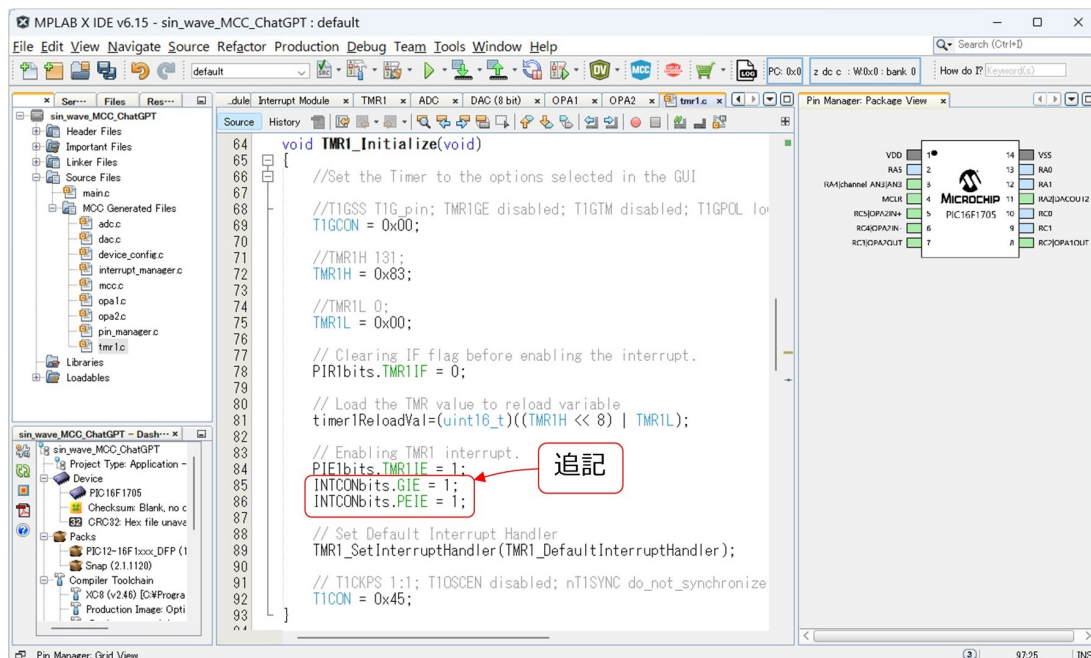


図 A.22: 追加の設定 (GIE, PEIE)

タイマ 1 モジュールでは、追加で [GIE\(Global Interrupt Enable\)](#) ビットと [PEIE\(PERipheral Interrupt Enable\)](#) ビットを “1” に設定しなければなりません。図 [A.22](#) は、GIE と PEIE を 1 に設定するコードとその追加箇所を示します。プロジェクトツリー内の `tmr1.c` ファイルを左クリックします。すると、Composer エリアに同ファイルが展開されます。スクロールすると [TMR1_Initialize\(\)](#) 関数が現れます。 `PIE1bits.TMR1IE = 1` の下に

```
INTCONbits.GIE = 1;
INTCONbits.PEIE = 1;                                     (A.2)
```

を追記します。GIE ビットと PEIE ビットは [INTCON レジスタ](#) 内にあります。GIE は PEIE を包含し、PEIE は TMR1IE を包含します。TMR1 による割り込みを有効にするには、それを包含する PEIE と GIE を “1” にする必要があります。この追記をしないと、タイマ 1 割り込み処理関数は起動されません。

索引

- 4 過倍 PLL, 31
- ADC_GetConversion(), 23
- ADRES, 16, 17, 36
- AD 変換モジュール, 34
- CHS, 35
- Compiler, 27
- DAC1CON1, 16, 38
- DAC1OUT2, 10, 38
- DACOUT2, 37
- DAC_output, 38
- D-A 変換モジュール, 36
- Device, 27
- DIP, 7
- Encoding, 28
- FOSC, 30
- Generate, 42
- GIE, 44
- Global Interrupt Enable, 44
- GND, 7
- GO_nDONE, 35
- High GBWP mode, 39
- ICSP コネクタ, 10
- INTOSC oscillator, 30
- INTCON レジスタ, 44
- I/O ポート, 7
- left, 34
- Low-voltage Programming, 31
- LVP, 31
- MCC Classic, 29
- MCC Melody, 29
- MCC の起動, 28
- MCLR, 7
- New Project, 26
- OPA1OUT, 10, 40
- OPA2IN, 41
- OPA2OUT, 10
- PEIE, 44
- Peripheral Interrupt Enable, 44
- PIC16F1705, 6
- Project Location, 28
- Project Name, 28
- Projects タブ, 43
- System Module, 30
- TAD, 34
- TMR1, 17, 32
- TMR1.Initialize() 関数, 44
- Unity Gain Amp, 40
- UTF-8, 28
- v_{DAC} , 10
- v_{FCOM} , 10
- v_{FIL} , 10
- v_{OP1} , 10
- インピーダンス変換, 40

オペアンプ, 17
オペアンプモジュール 1, 39
オペアンプモジュール 2, 41
クロック源, 16, 33
クロックソース, 33
周波数指令電圧, 10, 16
正弦波生成ロジック, 16
タイマ 1 割り込み処理関数, 16
タイマ 1 割り込み周期, 18
タイマ 1 割り込み処理関数, 18, 22
バッファアンプ, 10
左寄せ, 34
ピン配置, 7
ピン番号, 7
分周器, 16
プロジェクトツリー, 43
ボルテージフォロワ, 17
マルチプレクサ, 38
ユニティゲイン, 39
ユニティゲインアンプ, 10
ローパスフィルタ, 10
クロックソース, 31
システムクロック, 31

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.
- [2] 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
- [3] パワーエレクトロニクスノート
- [4] モータドライブノート
- [5] 古橋武「PIC マイコン (PIC16F1825) による正弦波発生器，PWM 信号発生器（第2版）-MCC(MPLAB Code Configurator) による周辺モジュール設定関数の自動生成-」無料配布ファイル，2024
- [6] 古橋武「パワーエレクトロニクスノート - PIC16F1825 による正弦波発生器，PWM 信号発生器 -」無料配布ファイル，2019.

著者

古橋 武

名古屋大学名誉教授（令和2年4月より）

furuhashi.takeshi*

*に @gmail.com を付けてください.