オペアンプ内蔵形PICマイコンPIC16F1705による正弦波・矩形波発生器

古橋　武

令和3年10月

目 次

第1章　はじめに

本稿は，オペアンプ内蔵形 PIC マイコン PIC16F1705 による正弦波・矩形波発生器のプログラムとその動作について解説します．ただし，途中からは詳細説明を端折っています．執筆途中で，この発生器を使う予定だったシリーズ本の執筆方針を大きく変えたため，本発生器の活用場所を自分で無くしてしまいました．しばらく，執筆を中止しておりましたが，せっかく途中まで書いたので公開いたします．部品，コンパイラなどの情報は令和 3 年 10 月時点のものです．

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・矩形波発生器

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

類書として，

「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－」

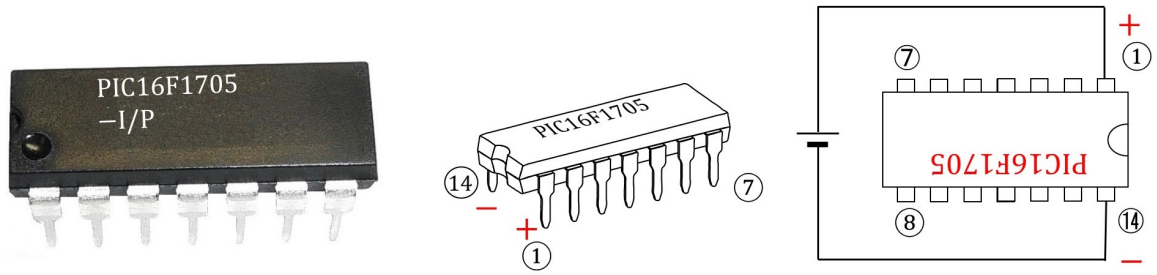
に PIC16F1825 による正弦波発生器，PWM 信号発生器のプログラムとその動作について解説しています．また，

「PIC16F1825 による DC モータの回転数制御」

に PIC16F1825 のモジュール設定関数の定義方法および（関数設定にとって直感的に分りやすい）専用語の定義方法を詳述しています．必要に応じて参照してください．

第2章　部品

2.1　PICマイコン



(a) 外観　　(b) 立体図　　(c) 電源の接続

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 変換器とオペアンプを 2 個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[μA] 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 [2][3] 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 [2] の正弦波信号を利用する製作課題には 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 Inline 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．

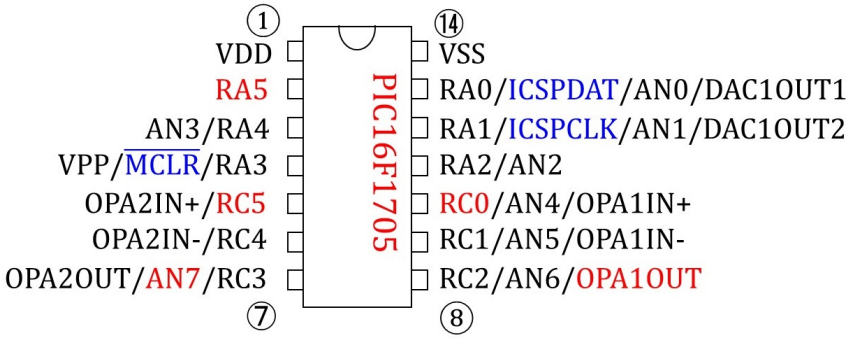


図 2.2: PIC16F1705 のピン配置

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．RAx, RCx は I/O(Input/Output) ポートです．ANx 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．DAC1OUTx が D/A 変換器の出力ピンです．OPAxIN± がオペアンプの入力ピン，OPAxOUT が出力ピンです．また，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 ICSP$^{\text{TM}}$ (In-Circuit Serial Programming) 用のピンです．PICKit$^{\text{TM}}$3, PICKit$^{\text{TM}}$4，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．

2.2　抵抗器

2.2.1　可変抵抗器

図 2.3 は可変抵抗器の外観と内部構造および記号です．可変抵抗器は抵抗値を変化させられる抵抗器です．写真のタイプは a, b, c の 3 電極を持ちます．抵抗器上面の灰色部分をネジ回しで回転させると，b 電極の先が連動して a-c 間の抵抗体表面を摺動します．これにより，a-b 間，b-c 間の抵抗値を変化させられます．a-c 間の抵抗値は固定です．図 (b) 最下図のように，灰色部分を時計方向いっぱいに回すと，a-b 間の抵抗 R_{a-b} 最小，b-c 間の抵抗 R_{b-c} 最大となります．逆に図 (b) 最上図のように，反時計方向いっぱいに回すと，R_{a-b} 最大，R_{b-c} 最小となります．抵抗器の全面に a-c 間の抵抗値が印字されています．写

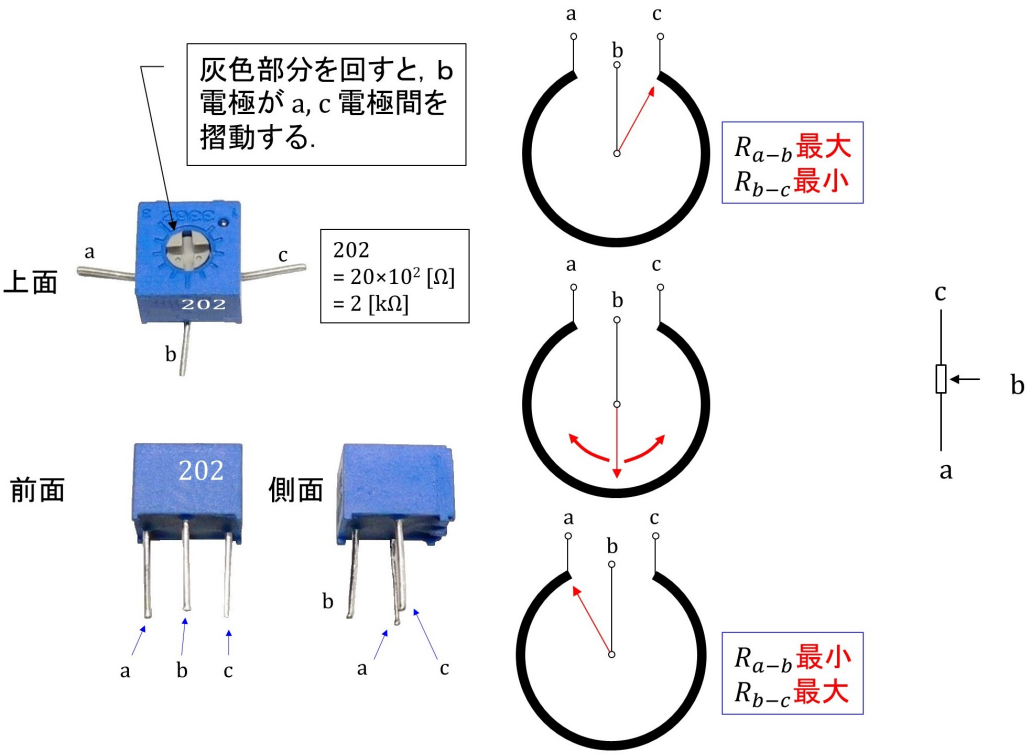


図 2.3: 可変抵抗器

真の例は 202 です．この数値の意味は

$$\begin{aligned} 202 &= 20 \times 10^2[\Omega] \\ &= 2[\mathrm{k}\Omega] \end{aligned} \tag{2.1}$$

です．したがって，R_{a-b}, R_{b-c} の変化範囲は

$$\begin{aligned} R_{a-b} &= 0 \sim 2[\mathrm{k}\Omega] \\ R_{b-c} &= 2[\mathrm{k}\Omega] - R_{a-b} \\ &= 2 \sim 0[\mathrm{k}\Omega] \end{aligned} \tag{2.2}$$

です．

写真の可変抵抗器は半固定形です．ネジ回しでなく，指でつまんで回せるタイプの可変抵抗器と区別するために，半固定形と呼ばれます．また，可変抵抗器はボリュームとも呼ばれます．写真のタイプのボリュームは半固定ボリュームとも呼ばれます．

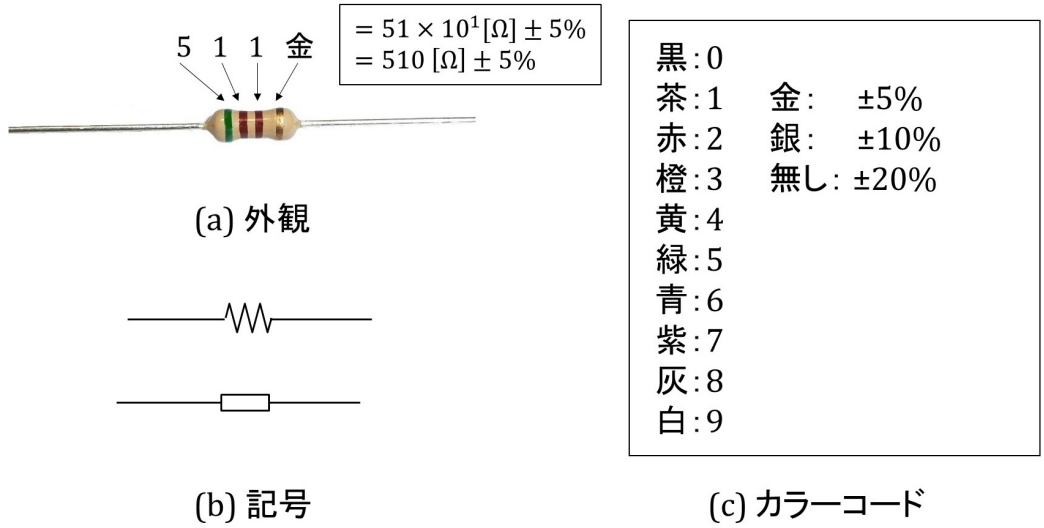


図 2.4: 抵抗器

2.2.2　固定抵抗器

図 2.4 は抵抗器の外観，記号とカラーコード表です．可変抵抗器に対して固定抵抗器とも呼ばれます．抵抗値はカラーコードで表示されています．写真の例は緑，茶，茶なので

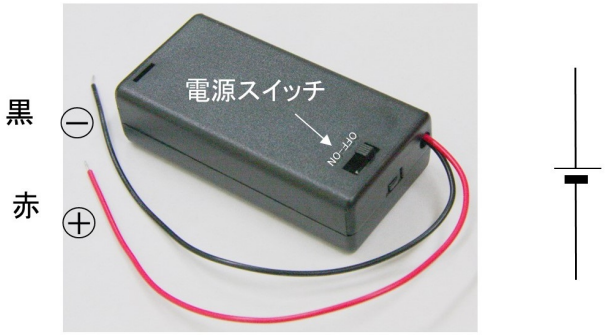
$$\begin{aligned} 511 &= 51 \times 10^1[\Omega] \\ &= 510[\Omega] \end{aligned} \tag{2.3}$$

です．右端の金色は抵抗値の精度を表します．実際の抵抗値は 510 [Ω] ± 5%の範囲にあります．写真の抵抗器はカーボン抵抗器（炭素皮膜抵抗器）と呼ばれます．

カラーコードの憶え方に「くちあだきみあむはし」と語呂合わせをする方法があります．「くちあ」という名前の滝と，「みあむ」という名前の橋があると憶えます．く（黒，0），ち（茶，1），あ（赤，2），だ（橙, 3），き（黄，4），み（緑，5），あ（青，6），む（紫, 7），は（灰，8），し（白，9）と対応付けています．

2.3　電池ボックス

図 2.5 は電池ボックスの外観と記号です．単 3 乾電池 2 本を直列接続して収納できるタイプです．出力電圧は乾電池の場合 3 [V]，充電池の場合 2.5 [V] です．赤い電線が電池の＋側，黒い電線が－側つながっています．電源スイッチが付いているので，容易に電源のオン／オフ切替ができます．



(a) 外観　　(b) 記号

図 2.5: 電池ボックス

2.4　ブレッドボード

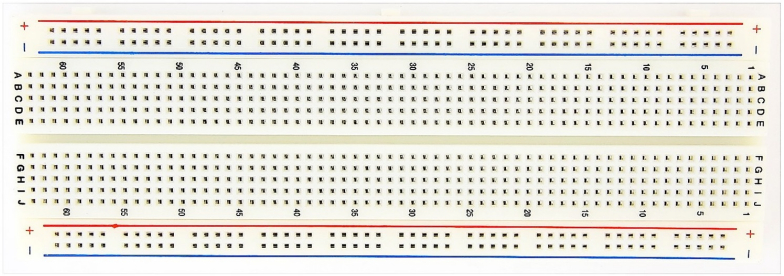
図 2.6 はブレッドボードの外観とボード内部での穴のつながりの様子を示します．このブレッドボードに電子部品を挿入することで，回路配線を行います．電子回路の配線には半田づけが使われることが多いのですが，半田づけは初心者には負担の大きな作業です．また，数十人の学生に毎回半田づけ作業を課していては，やけどなどの危険が大きくなります．ブレッドボードは半田づけを必要としないので，配線が容易ですし，やけどの危険もありません．同図 (b) 中の□印がジャンパ線を差し込める穴です．黒い線によりブレッドボード内部での穴同士のつながりを示します．最上段の 2 行と最下段の 2 行では，各行内の 50 個の穴が内部でつながれています．これら上下各 2 行の穴に挟まれて 63 列の穴があります．各列には 10 個の穴があり，A〜E，F〜J の 5 個ずつが内部でつながれています．同図 (a) の写真のように，A〜E，F〜J の上下 5 個ずつの列の中央には溝があり，この溝の部分で上下の内部配線は切れています．

2.5　ジャンパ線

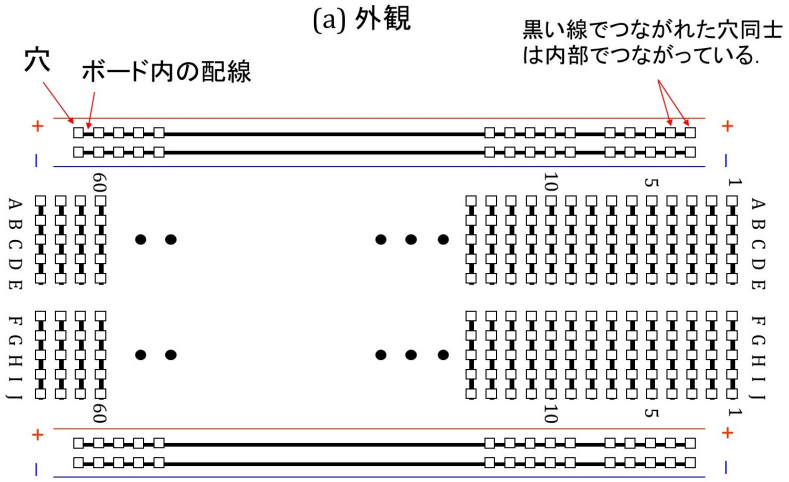
図 2.7 はジャンパ線です．ブレッドボード上に回路を組む場合に，必要に応じてジャンパ線により配線します．

図 2.8 はジャンパ線をブレッドボードに差し込んだ場合に，つながっている配線とつながっていない配線の例を示します．同図 (a) はつながっている例です．(a) の上の図は最上行の穴に 2 本のジャンパ線が挿入されています．これは同じ行内なので，ブレッドボード内部でこれら 2 本のジャンパ線がつながっています．(a) の下の図は同じ列内でつながっている例です．

一方，同図 (b) はつながっていない配線例です．(b) の上の図では行がずれています．(b) の下左の図では列がずれています．また，(b) の下右の図では列は同じですが，間に溝を挟んでいます．列同士の内部配線は溝で切れています．



(a) 外観



(b) ブレッドボード内部での穴のつながりの様子

図 2.6: ブレッドボード

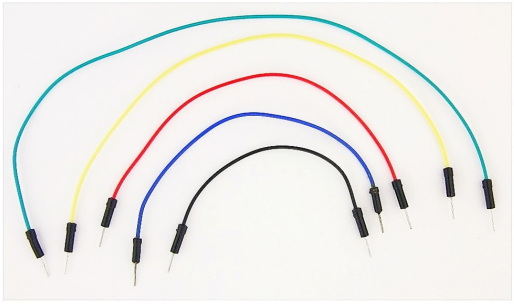
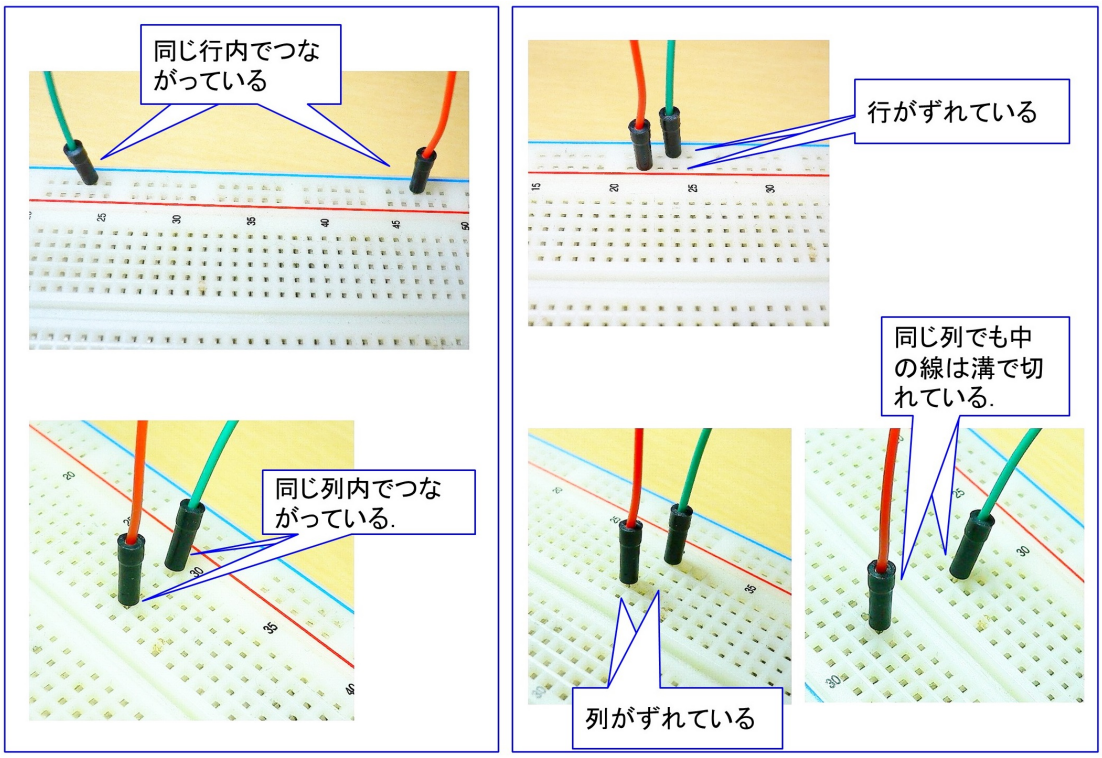


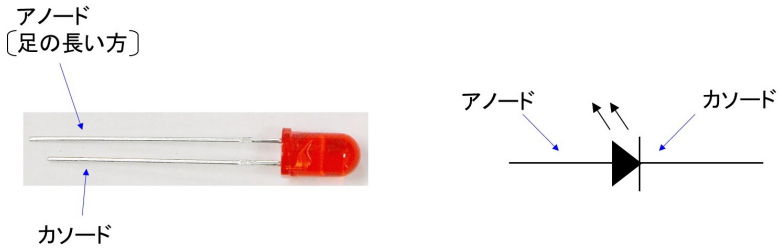
図 2.7: ジャンパ線



(a) ボード内でつながっている配線の例　　(b) つながっていない配線の例

図 2.8: ブレッドボードにおいてつながっている配線とつながっていない配線の例

2.6　LED



(a) 外観　　(b) 記号

図 2.9: LED

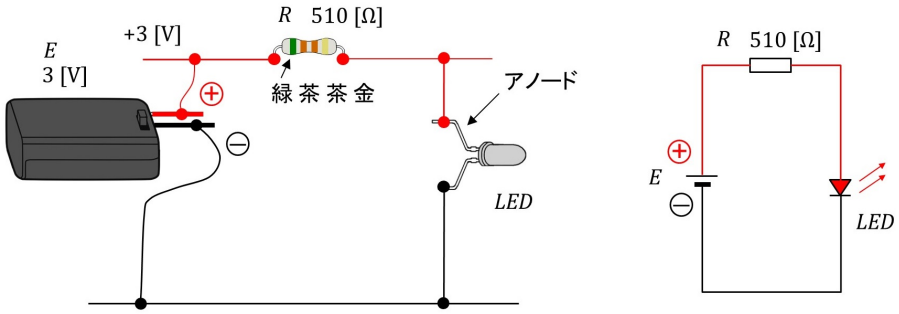
図 2.9 は LED (Light Emitting Diode : 発光ダイオード）の外観と記号を示します．図は赤色 LED（直径 5mm (5mmϕ)）の例です．LED には電極が 2 つあり，同図 (b) のようにそれぞれアノード，カソードと名付けられています．LED の記号には，ダイオードの記号に，発光を象徴する 2 本の矢印が付けられています．アノード，カソード電極は，LED 本体内部の電極形状により見分けることができます．同図 (a) の写真のようにカソード側電極はペリカンのくちばしの様な形状をしています．また，2 本の足の長さには違いが付けられています．長い方がアノード，短い方がカソードです．

アノードに＋電圧，カソードに－電圧を印加することで，アノードからカソードへと電流を流すことができます．逆向きに電圧を印加した場合は，カソードからアノードにわずかな漏れ電流が流れます．アノードからカソードに流れる電流は順方向電流，逆方向に流れる電流は逆方向電流と呼ばれます．また，アノード側が＋，カソード側が－の電圧は順方向電圧，逆は逆方向電圧と呼ばれます．順方向電圧を V_F，順方向電流を I_F，逆方向電圧を V_R，逆方向電流を I_R と記します．最大順方向電流（直流電流）は写真の 5mmϕ タイプの場合 30[mA] です．また，標準順方向電圧 $V_F = 2$ [V] です．逆方向電流 I_R は $V_R = 5$[V] のとき最大で 10 [μA] です．

図 2.10 は LED の電池への接続例です (a) が立体配線図，(b) が回路図です．図中の電池ボックスには乾電池（1 本当たり 1.5[V]）が 2 本入っているとします．LED を直接電池につなぐと，LED には最大順方向電流を上回る電流が流れて，LED は瞬時に壊れてしまいます．図のように抵抗と LED を直列につないで電池に接続します．このとき LED に流れる電流 I_F は

$$\begin{aligned} I_F &= \frac{V_E - V_F}{R} \\ &= \frac{3 - 2[\mathrm{V}]}{510[\Omega]} \\ &\approx 2[\mathrm{mA}] \end{aligned} \tag{2.4}$$

です．



(a) 立体配線図　　(b) 回路図

図 2.10: LED と電源の接続例

第3章　PIC16F1705へのプログラム書き込み／デバッグ回路

3.1　書き込み／デバッグ回路

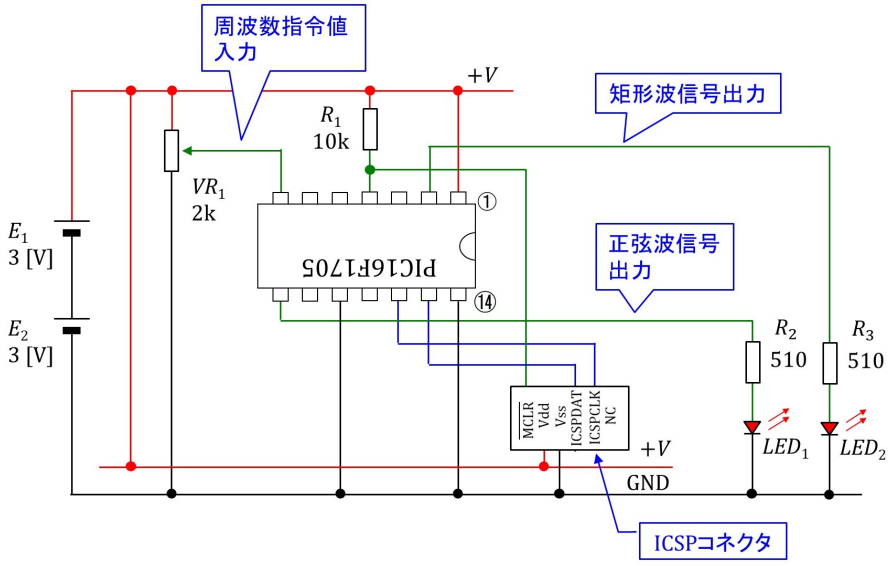


図 3.1: PIC16F1705 へのプログラム書き込み／デバッグ回路

図 3.1 は PIC マイコン PIC16F1705 へのプログラム書き込み／デバッグ回路図です．電源は図 2.5 の電池ボックス 2 個を直列接続し，それぞれに単 3 乾電池 2 個を入れて 6 [V] とします．PIC16F1705 の 1 番ピンを電池のプラス側に接続し，14 番ピンを－側に接続します．乾電池 2 個入りの電池ボックス 2 個を直列にしている理由は，このマイコンの出力を利用してオペアンプ回路実験を可能にするためです．3 [V] と 3 [V] の中間をグラウンド電位にできます．PIC16F1705 のデータシート（例えば秋月電子通商の PIC16F1705 の Web ページより無料でダウンロード可能です.）の Electrical Specifications によると，1 番ピンと 14 番ピン間の最大電圧は 6.5 [V] です．新品の乾電池は 6.3 [V] 程度の電圧を出力するので，ぎりぎり許容範囲内です．充電池を使えば，1 本 1.25 [V] なので，4 本直列で 5 [V] と適切な電圧となります．PIC マイコンへのプログラム書き込みとデバッグは ICSP® コネクタを介して行います．ICSP® コネクタの詳細は 3.2 項に記します．

本稿では，PIC16F1705 を正弦波・矩形波発生器として使用します．このプログラムは 7 番ピンをアナログ入力とし，8 番ピンを正弦波信号出力，2 番ピンを矩形波信号出力と

します．可変抵抗器 VR_1 により 7 番ピンの入力電圧を変えることで，正弦波・矩形波信号の周波数を変えられるプログラムです．

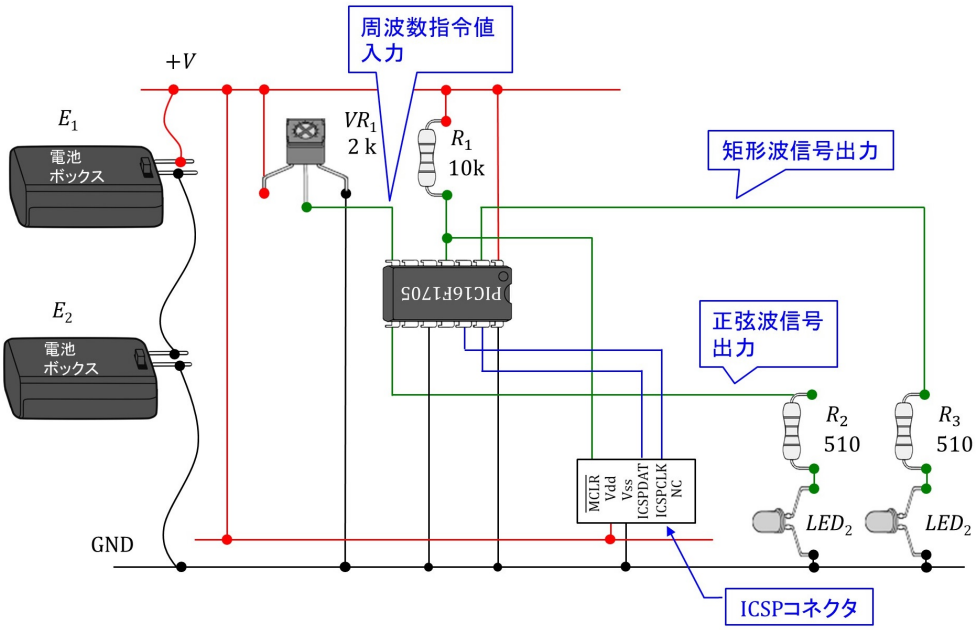


図 3.2: PIC16F1705 へのプログラム書き込み回路の立体配線図

図 3.2 は PIC マイコン PIC16F1705 へのプログラム書き込み回路の立体配線図です．

図 3.3 は PIC マイコン PIC16F1705 へのプログラム書き込み回路の作成例の写真です．写真では配線の様子を見やすくするために，ジャンパ線に単線（直径 0.5mm）の架橋ポリエチレン電線（協和ハーモネット　KQE 0.5mm 単線）を用いています．この電線は耐熱電子ワイヤーという名前でも売られています．なお，耐熱通信機器用ビニル電線（協和ハーモネット　導体径 0.65mm 単芯）を使うこともできます．工具にはニッパーとワイヤストリッパーが必要です．

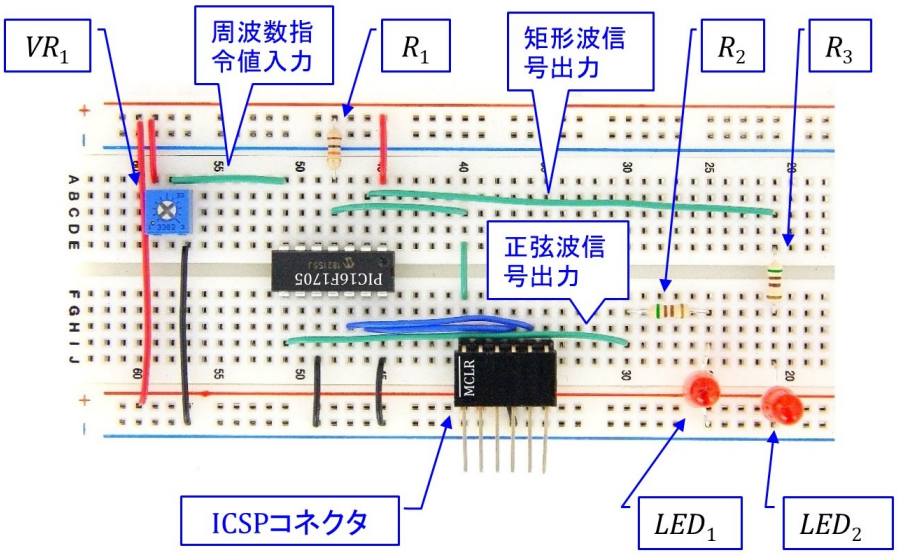


図 3.3: PIC16F1705 へのプログラム書き込み回路の作成例

3.2　ICSP® コネクタ

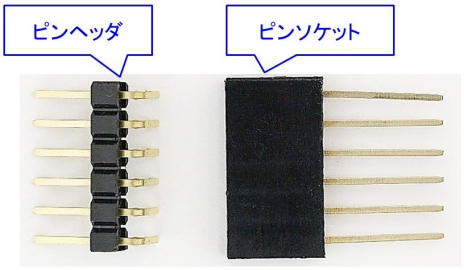


図 3.4: ピンヘッダとピンソケット

ICSP® コネクタは，図 3.4 の 6 ピンのピンヘッダとピンソケットを組み合わせて作ります．

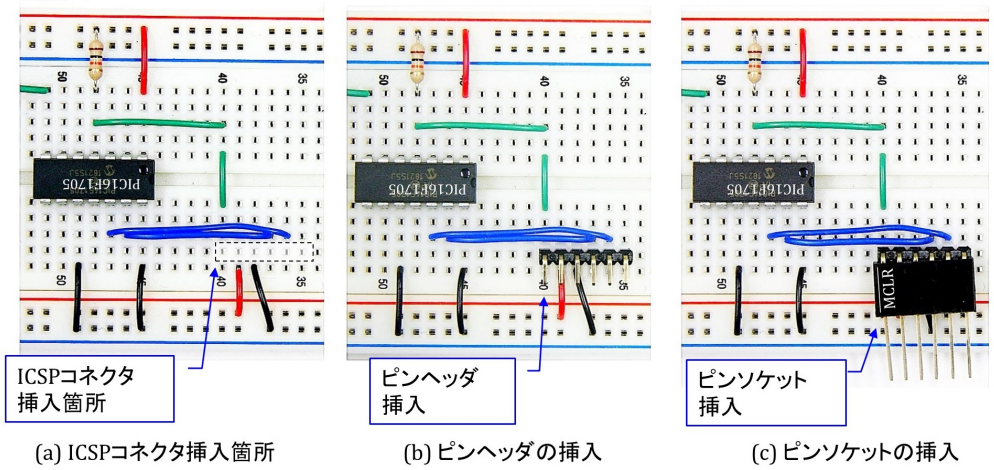


図 3.5: ICSP® コネクタの製作

Fig.3.5 は ICSP® の製作過程を示します．同図 (a) の破線で囲った 6 個の穴にピンヘッドを挿入します．同図 (b) がピンヘッダを挿入した写真です．このピンヘッダにピンソケットを挿入して ICSP® コネクタができあがります．

図 3.6 は PICkit™3 の外観とピン配置を示します．ピン穴は 6 個あり，ICSP® コネクタを挿入できます．◁ 記号側が 1 番ピンです．

図 3.7 は ICSP® コネクタに PICkit™3 を挿入した様子です．PICkit™3 は USB ケーブルによりパソコンとつなぐことができ，パソコンよりプログラム書き込みおよびデバッギングができます．なお，PICkit™4，MPLAB™Snap も使えます．これらデバッガ/プログラマのピンソケットは 8 ピンです．◁ 記号側が 1 番ピンなので，これを $\overline{\text{MCLR}}$ に合わせて，6 番ピンまでを ICSP® コネクタにつなげばよいです．

図 3.6: PICkit™3

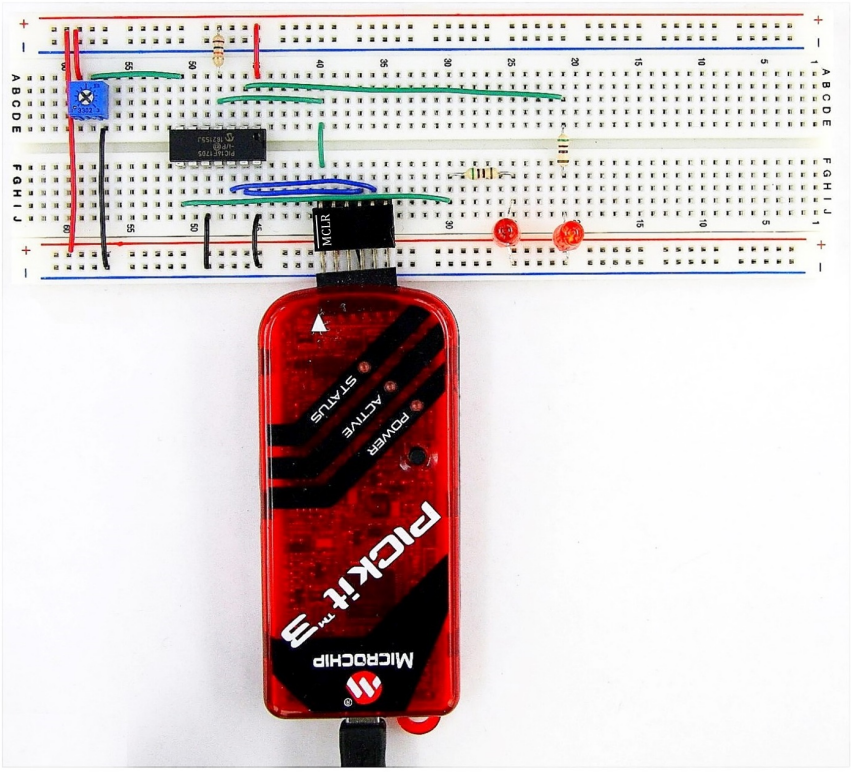


図 3.7: ICSP コネクタに PICkit™3 を接続

以上でマイコンにプログラム書き込みができるようになったのですが，たくさんのマイコン（例えば 100 個）にプログラム書き込みを行うには，マイコンのブレッドボードへの挿抜が手間です．そのようなときにはゼロプレッシャーソケットが便利です．図 3.8 はゼロプレッシャーソケットの外観です．写真のようにレバーを立てた状態では，ソケット口は広く開いていて，圧力ゼロ（ゼロプレッシャー）でマイコンをソケット口に挿入できます．レバーを倒すとソケット口が固く閉じるため，マイコンをソケットに固定できます．そこで，あらかじめゼロプレッシャーソケットだけをブレッドボードに差し込んでおきます．そして，マイコンをソケット口に挿入してレバーを倒すことで，マイコンを回路に接続できます．

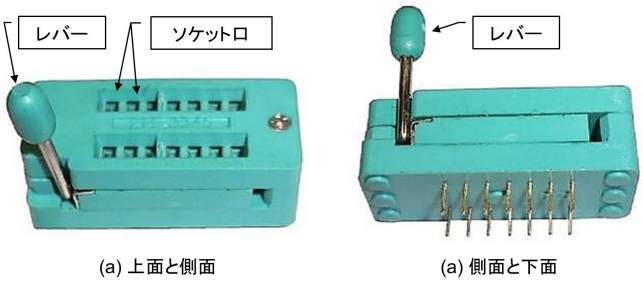


図 3.8: ゼロプレッシャーソケット

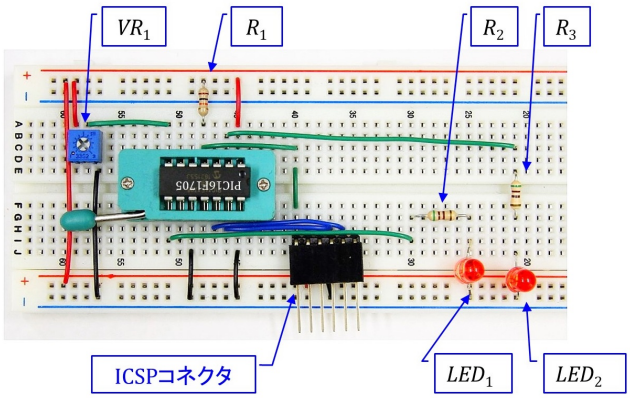


図 3.9: ゼロプレッシャーソケット使用によるマイコン挿抜の容易化

図 3.9，3.10 はゼロプレッシャーソケットによりマイコンをブレッドボード上の回路に接続した様子です．これでマイコンにプログラムを書き込むことができます．書き込み後は，レバーを立てることでマイコンを抵抗なくソケットから抜き取ることができます．

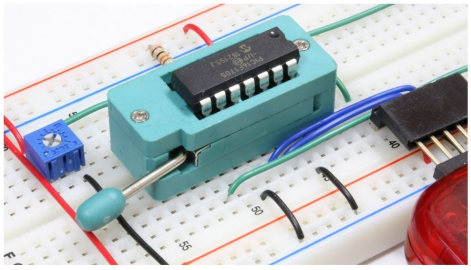


図 3.10: ゼロプレッシャーソケット使用によるマイコン挿抜の容易化（その 2）

第4章　MPLAB® X IDE, XC8 コンパイラ，New Project，デバッガ

4.1　MPLAB® X IDE, XC8 コンパイラのインストール方法

本章では Microchip 社が無償提供している統合開発環境 MPLAB® X IDE (Integrated Development Environment : 統合開発環境，および，MPLAB® XC8 コンパイラのダウンロード，インストール方法と使用方法の概要を，Windows の場合について，紹介します．

MPLAB® X IDE は Microchip 社のホームページ → Tools and Software → MPLAB X IDE → MPLAB X IDE Windows Download とたどることでインストーラ (MPLABX-v5.50-windows-installer.exe) をダウンロードできます (2021 年 10 月時点)．このインストーラを立ち上げ，インストーラの推奨通りに Next ボタンを押していくことで，MPLAB® X IDE をインストールできます．無事インストールに成功すれば，C:¥Program Files (x86) のフォルダ内に Microchip という名前のフォルダ，ドキュメントフォルダ内に MPLABX-Projects という名前のフォルダが作られます．

同様に，MPLAB® XC8 コンパイラは Microchip 社のホームページ → Tools and Software → MPLAB® XC Compilers → Compiler Downloads → MPLAB® XC8 Compiler v2.32 (2021 年 10 月時点) とたどることでインストーラ (xc8-v2.32-full-install-windows-x64-installer.exe) をダウンロードできます．このインストーラを立ち上げ，推奨通りに Next ボタンを押していくことで，XC8 コンパイラをインストールできます．無事インストールに成功すると，Microchip フォルダ内に xc8¥v2.32 という名前のフォルダが作られます．v2.32¥pic¥include フォルダ内に本稿で用いるヘッダファイル xc.h がダウンロードされます．

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・矩形波発生器

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

4.2　New Project の作成方法

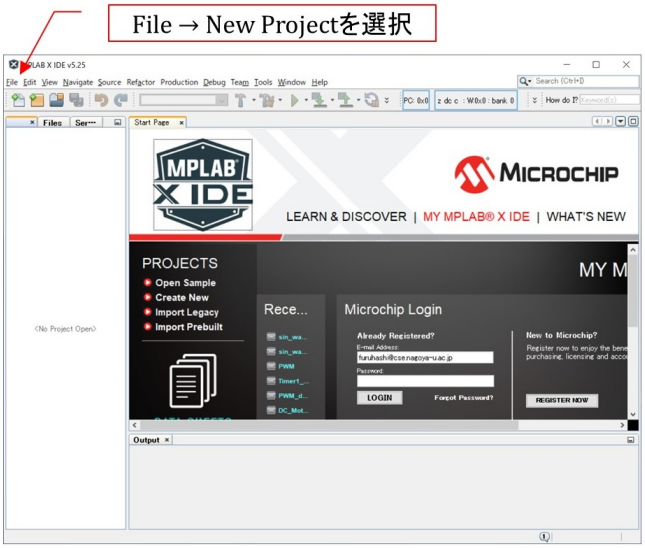


図 4.1: MPLAB® X IDE の立ち上げと New Project の設定

第 6 章で詳述するファイルを用いて，MPLAB® X IDE による編集，マイコンへの書き込み方法を記します．

MPLAB® X IDE のアイコンを左ダブルクリックすることで，この統合開発環境を立ち上げることができます．図 4.1 の画面が立ち上がったら，File → New Project を選択します．

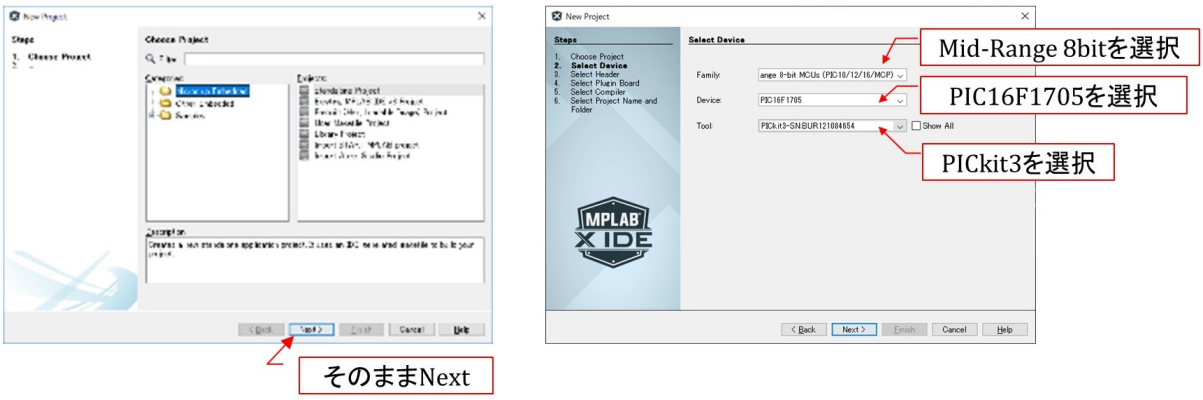


図 4.2: MPLAB® X IDE: Device, Tool 選択

次に図 4.2 のように進み，デバイスとして PIC16F1705 を選択します．PICkit3 をパソコンにつないでおくとツールのプルダウンメニューに PICkit3 が選択肢として表示されるので，これを選択します．No Tool のままでもよいです．後にプログラムをマイコンへ書

き込むときに Tool の指定を要求されるので，その段階で PICkit3 などのデバッガ/プログラマをパソコンにつないで，当該デバッガ/プログラマを指定してもよいです．

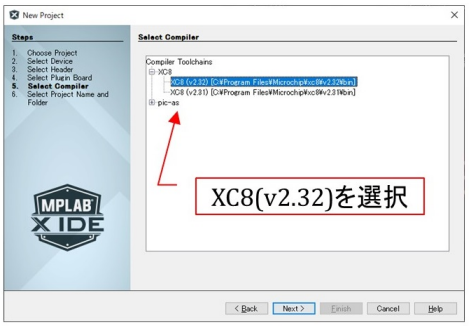


図 4.3: MPLAB® X IDE: compiler 選択

その後は図 4.3 のようにコンパイラに XC8(vx.xx).... を選択します．

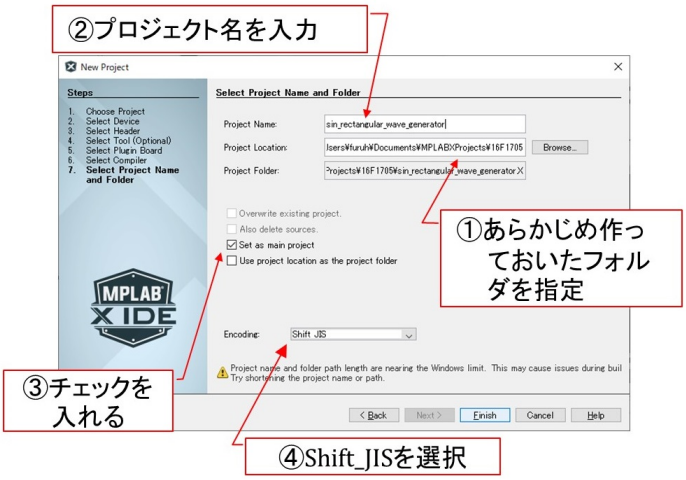


図 4.4: MPLAB® X IDE: project name, folder, 言語選択

図 4.4 は次に表示される画面です．あらかじめ作っておいたフォルダ（例えば，MPLABX-Projects フォルダ内に PIC16F1705 という名前のフォルダを作っておきます.）をブラウズし，プロジェクト名を自分で決めて（例えば，sin_rectangular_wave_generator とします.）入力し，”Set as main project”にチェックを入れて，言語に Shift_JIS を選択します．

4.3　ファイルのプロジェクトへの追加

以上が完了した段階で，（上の例では PIC16F1705 のフォルダ内に sin_rectangular_wave_generator という名前の）フォルダが作られています．図 4.5 のように，このフォルダの

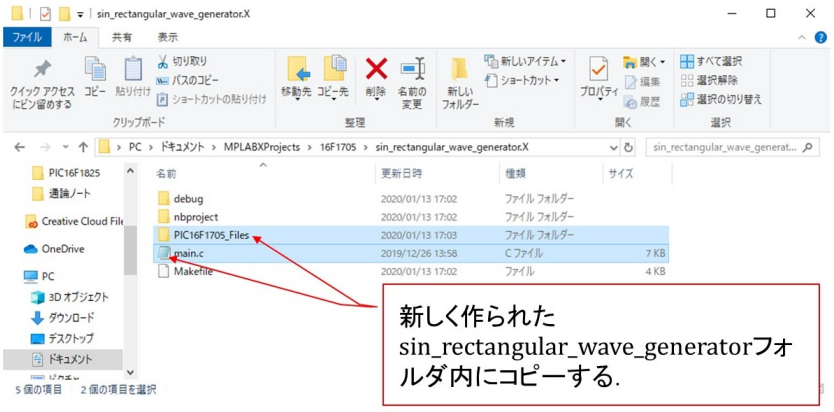


図 4.5: main.c，PIC16F1705_Files を sin_rectangular_wave_generator フォルダ内へコピー

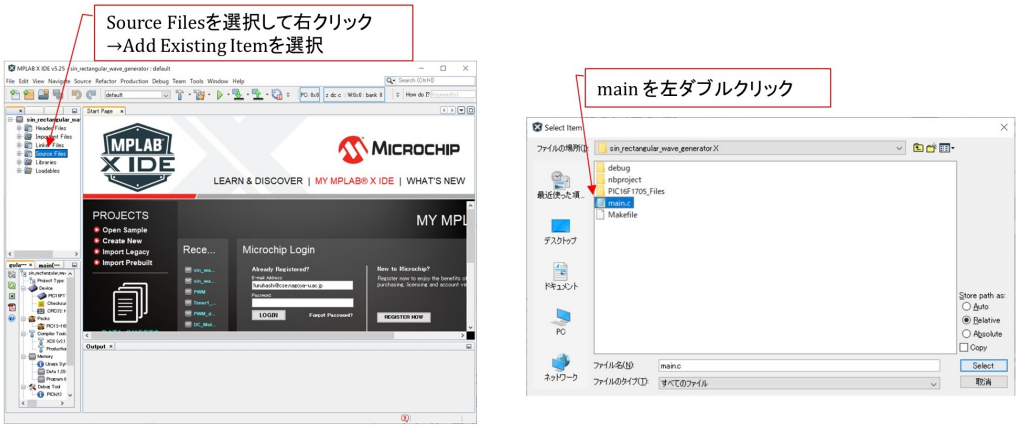


図 4.6: MPLAB® X IDE: Source file の追加

中へ

オペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器

からダウンロードしておいた「sin_rectangular_wave_generator_for_Web_up」フォルダ内のソースファイル (main.c) および関連ファイルの入っているフォルダ (PIC16F1705_Files) をコピーします．

次に，これらのファイルをプロジェクトに追加します．図 4.6 のように，Source Files のフォルダを右クリックし，Add Existing Item を選択します．そして，同図右のように main.c を選択します．次に，図 4.7 のように，Linker Files を選択して右クリックし，Add Existing Item を選択します．そして，同図右のように，set_ad_converter.c, set_da_converter.c, set_op_amp.c, set_osc.c, set_timer1.c を選択します．

最後に，図 4.8 の画面のように，Header Files のフォルダを右クリックして，ヘッダファイル pic16f1705_s.h, f_mul_data.h, sin_data.h を追加します．

図 4.7: MPLAB® X IDE: Linker files の追加

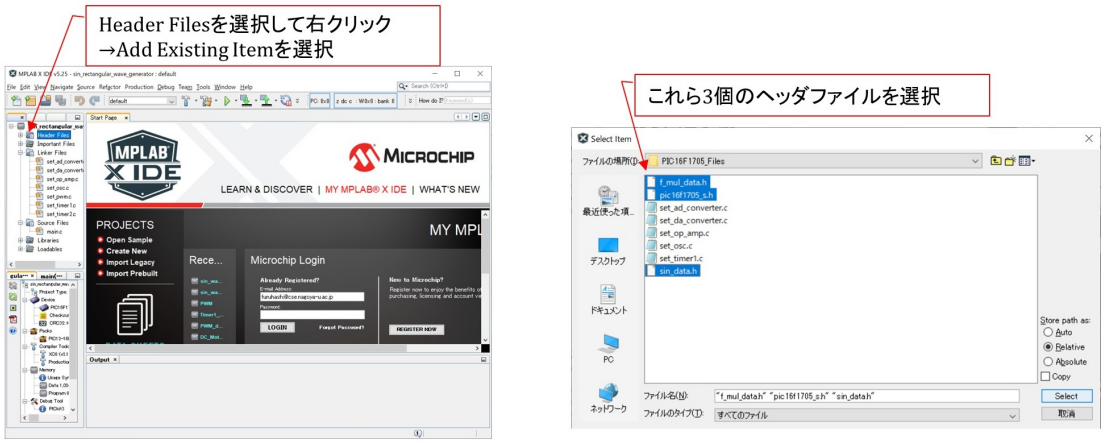


図 4.8: MPLAB® X IDE: Header files の追加

4.4　プログラムのビルド，書き込み

以上で MPLAB® X IDE によるプログラムの編集，PIC マイコンへの書き込み準備が完了しました．各電池ボックスに 2 本ずつ乾電池 (計 6[V]) もしくは充電池 (計 5[V]) を入れて，ブレッドボードの＋電源および GND ライン間に接続して電圧を印加します．ICSP® コネクタに PICkit™3 もしくは PICkit™4, MPLAB™Snap を接続し，デバッガ/プログラマとパソコンを USB ケーブルで接続します．

図 4.9 のように，画面内の main.c を左ダブルクリックすることで，このファイル内のプログラムを開くことができます．そして，Build Main Project ボタンをクリックして，同図下のように

BUILD SUCCESSFUL (total time: xxs)

のメッセージが出れば，全ての準備が完了です．

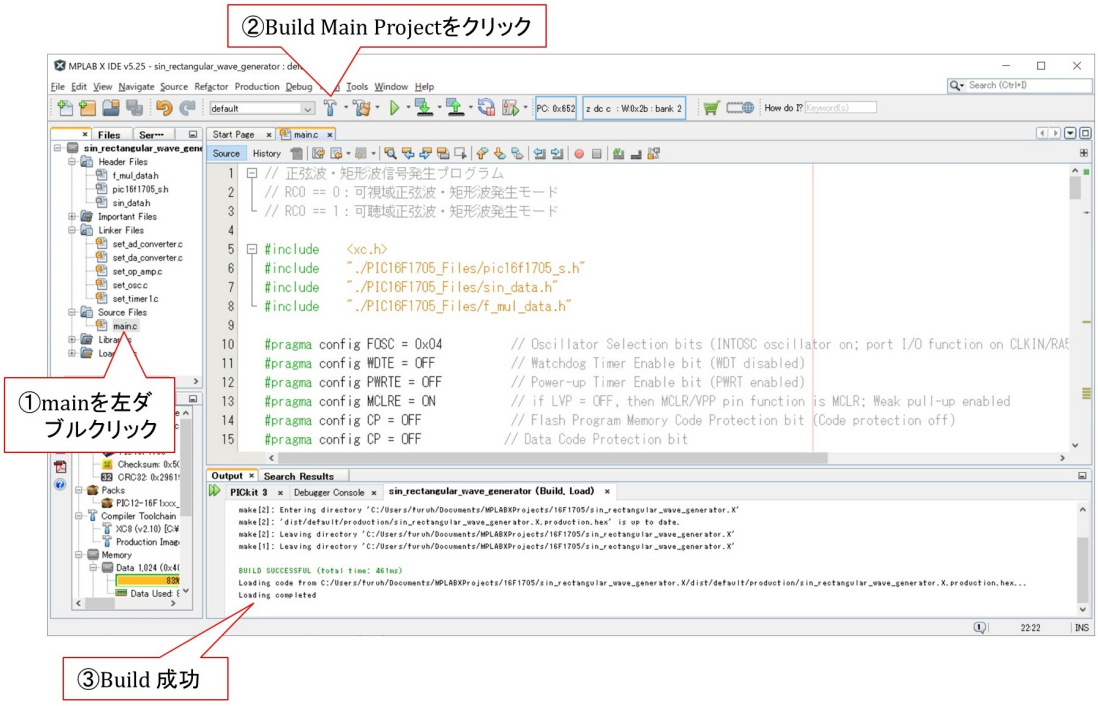


図 4.9: MPLAB® X IDE: Build

図 4.10 のように ▷ ボタンを押すとプログラムの Build とマイコンへの書き込みを実行します．同図下の画面のように

Programming/Verify complete

が表示されれば，書き込みが完了し，LED の明るさが変化するはずです．可変抵抗器上面のつまみをネジ回しで回すと，LED の明かりの変化速度が変わります．

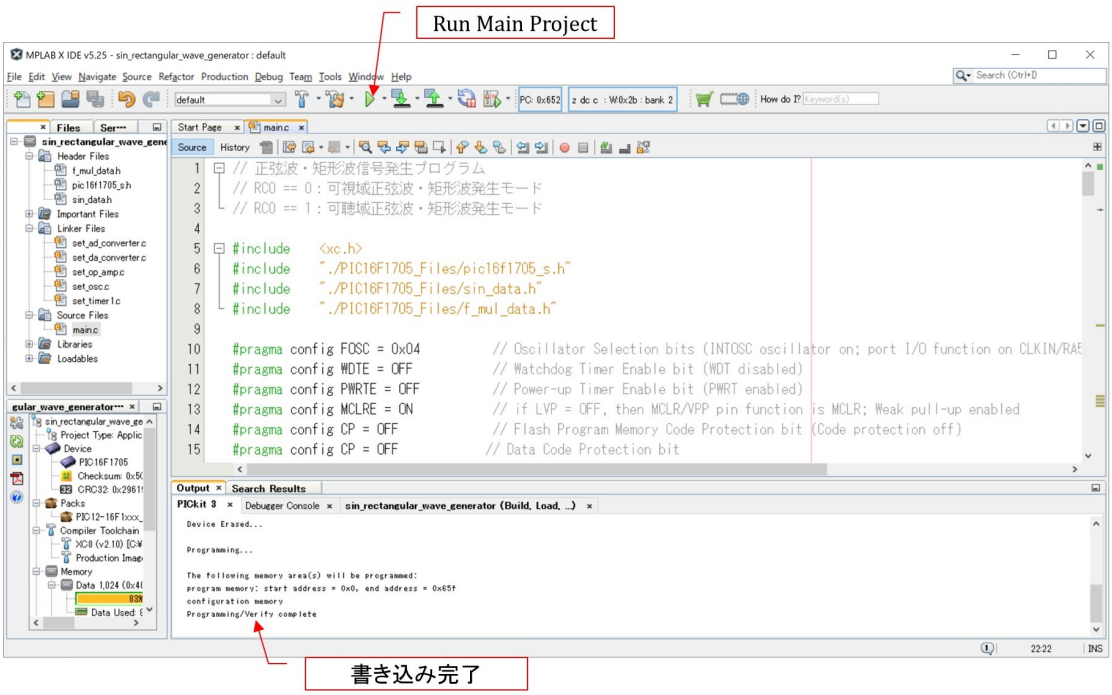


図 4.10: MPLAB® X IDE: Build とマイコンへの書き込み

4.5　デバッガの使用方法

4.5.1　デバッガの起動

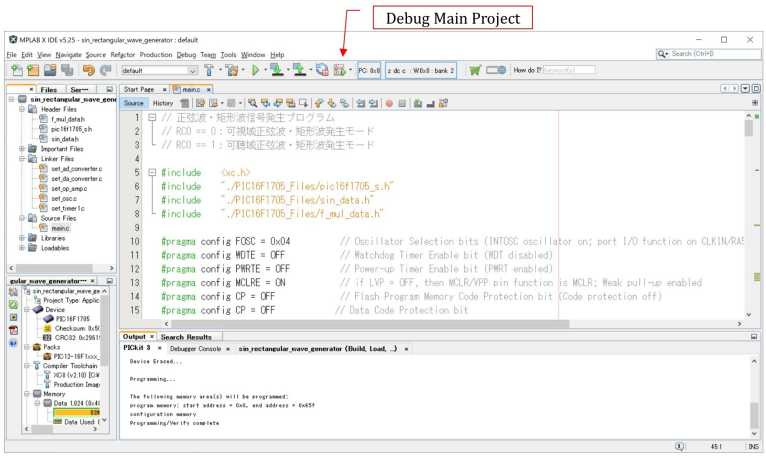


図 4.11: デバッガの起動

プログラムを改変したいときにはデバッガが便利です．図 4.11 の Debug Main Project ボタンを左クリックすることで，プログラムのビルドとマイコンへの書き込み，デバッガの起動を実行します．

4.5.2　デバッガの一時停止

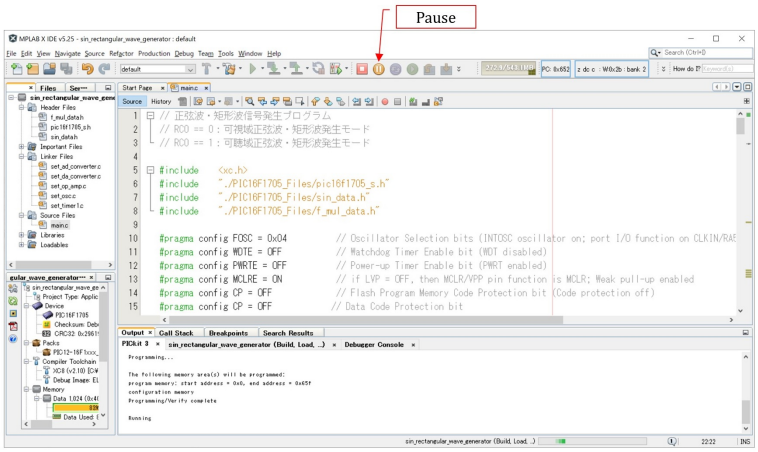


図 4.12: デバッガの一時停止

図 4.12 の Pause ボタンを左クリックすることで，を一時停止できます．

4.5.3　データの表示

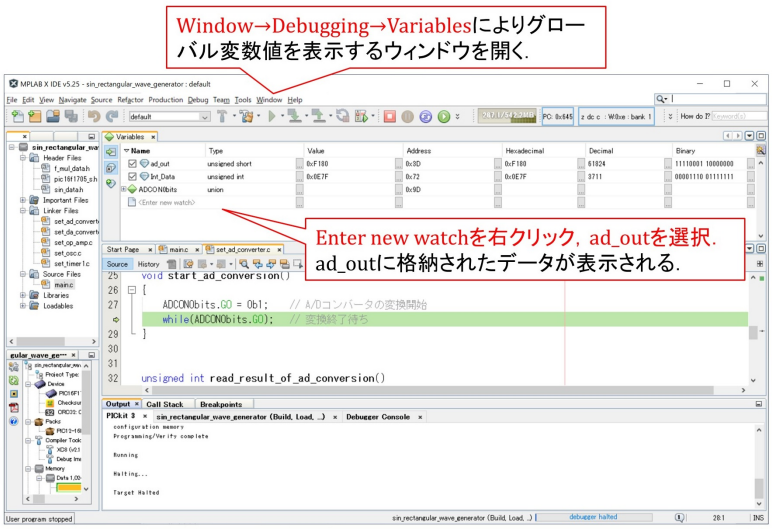


図 4.13: 表示データの指定

そして，この停止時のグローバル変数の値，レジスタの値などを表示させることができます．例えば，第 6 章のプログラムは，7 番ピンの入力電圧を A/D 変換して 10 ビットの値で読み出し，グローバル変数 ad_out に格納します．図 4.13 はマイコンの一時停止中に，ad_out の値を表示している画面です．Enter new watch を右クリックすると，プルダウンメニューが表示されます．New Watch を選択すると，表示可能なレジスタおよび変数の一覧が表示されます．その中から ad_out を選択して OK ボタンを押すと，図のように ad_out の値が表示されます．値の表示形式は 10 進，16 進，2 進を選べます．

第5章　sin_rectangular_wave_generator による信号発生

5.1　可視域モード

本章では第4章にてマイコンに書き込んだプログラム（sin_rectangular_wave_generator）の実行例を記します．

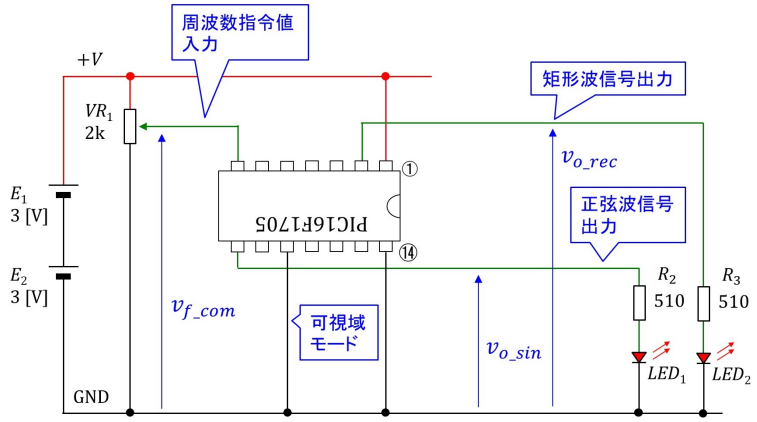
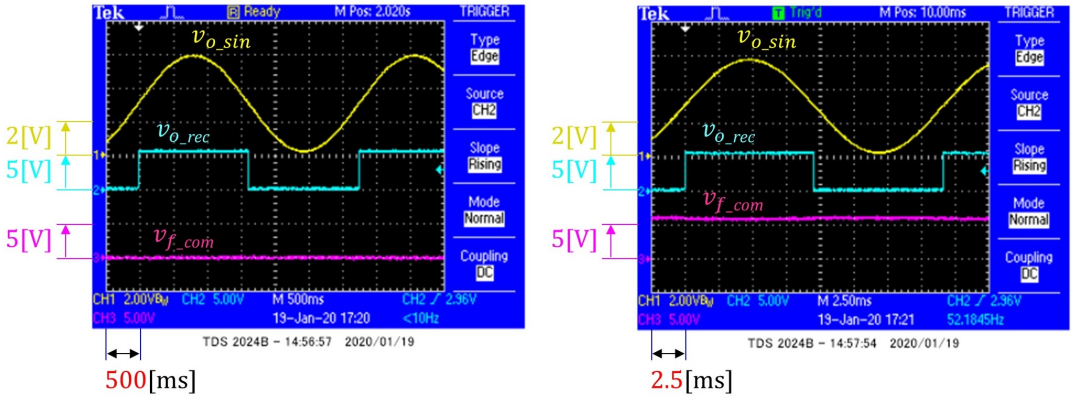


図 5.1: 正弦波・矩形波発生回路による実験（可視域モード）

図5.1は書き込み済みマイコンの実験回路です．7番ピンの入力電圧（周波数指令電圧）v_{f_com} により出力電圧の周波数を変えられます．8番ピンからは正弦波信号 v_{o_sin} が出力され，2番ピンからは矩形波信号 v_{o_rec} が出力されます．図では10番ピンを電源の－側につないであります．これにより，可視域モードのプログラムが実行されます．

図5.2は可聴域モードにおける入出力波形例です．同図 (a) は周波数指令電圧 $v_{f\_com} \approx 0$ [V] のときで，出力の正弦波信号 v_{o_sin}，矩形波信号 v_{o_rec} の周波数が最も低く，約0.3 [Hz] です．(b) は $v_{f_com} \approx +$V のときで，各信号の周波数が最も高く，約52 [Hz] です．v_{f_com} が低いとき，LED_1 の明るさは，肉眼で見られる速さで，ゆっくりと変化します．LED_2 の明るさはステップ的に変化し，その点滅を肉眼で見られます．v_{f_com} が高いときには，LED の明るさの変化は肉眼では見られなくなります．



(a) $f \approx 0.3$ [Hz]　　(b) $f \approx 52$ [Hz]

図 5.2: 正弦波・矩形波発生回路の周波数指令値，正弦波・矩形波信号波形（可視域モード）

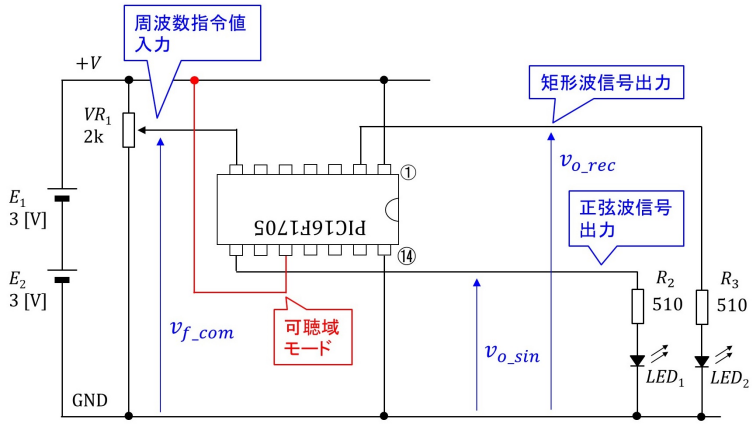


図 5.3: 正弦波発生回路による実験（可聴域モード）

5.2　可聴域モード

図 5.3 のように，10 番ピンを電源の＋側に接続して電源をオンにすると，可聴域モードのプログラムが実行されます．

図 5.4 は，可聴域モードにおける正弦波信号 v_{o_sin} の波形例です．可聴域モードでは矩形波信号 v_{o_rec} は出力されません．同図 (a) は周波数指令電圧 $v_{f\_com} \approx 0$ [V] のときで，v_{o_sin} の周波数が最も低く，約 110 [Hz] です．(b) は $v_{f\_com} \approx 3$ [V] のときで，v_{o_sin} の周波数は約 2.7 [kHz] です．(c) は $v_{f_com} \approx +$V のときで，v_{o_sin} の周波数は最も高くて約 11 [kHz] です．v_{f_com} の全範囲で正弦波信号の基本周波数は可聴域にあります．可聴域モードでは，マイコンの処理速度の限界から，正弦波データを間引くことで信号の周波数を上げています．このため，周波数が高いほど階段状の変化が顕著になっています．本プログラムでは，正弦波 1 周期を 800 分割して，各時点の正弦波データを用意しておきます．そして，

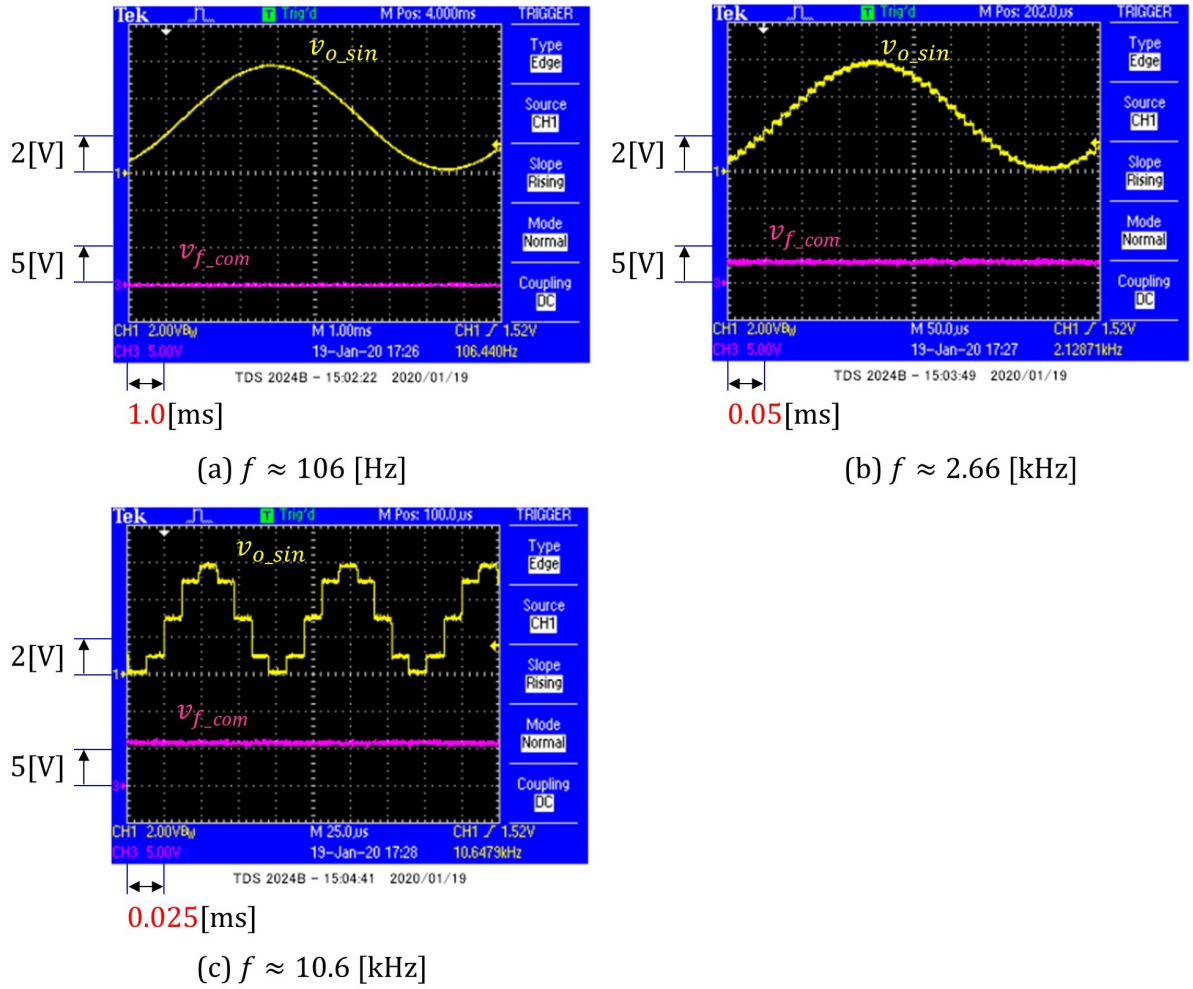


図 5.4: 正弦波・矩形波発生回路の周波数指令値，正弦波信号波形（可聴域モード）

一定時間間隔（T_{o_samp} とします.）で，800 個のデータを順次 D/A 変換器に出力します．図 5.4(a) がその出力結果です．可視域モードでは T_{o_samp} 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．可聴域モードでは，処理速度が間に合わないため，(a) において T_{o_samp} はすでに最短設定です．出力正弦波の周波数を上げるには，800 個のデータを一定個数ずつ跳ばし読みして D/A 変換器へ出力します．(b) では 1 周期を 32 分割して，800 個のデータを 25 個ずつ飛ばし読みしています．(c) では 100 個ずつ跳ばし読みしています．(c) では，波形の最大値と最小値の間にはわずか 5 段階の電圧値しかありません．11 [kHz] の基本波に対して，第 8 高調波（88 [kHz]）が基本波の次に大きな成分として含まれています．増幅回路でスピーカを駆動する実験にこの出力電圧を利用しても 88 [kHz] の高調波は聞こえないので，支障はありません．

第6章　プログラム

本章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・矩形波発生器
の圧縮ファイル sin_rectangular_wave_generator_for_Web_up.zip にあります．zip ファイル内には以下のフォルダがあります．

timer1_interupt
AD_Conv
DA_Conv
OP_Amp
sin_rectangular_wave_genrator
sin_rectangular_wave_genrator_filter

本章では　これらファイル内のコードを用いて，PIC16F1705 のタイマ 1 モジュール設定とタイマ 1 による割り込み，A/D 変換モジュール設定と A/D 変換，D/A 変換モジュール設定と正弦波信号出力，オペアンプモジュール設定と正弦波信号出力を順次解説します．

6.1　タイマ 1 モジュール設定とタイマ 1 による割り込み

4.2 節と同様にして，/MPLABXProjects/PIC16F1705 フォルダ内にプロジェクト名が timer1_interrupt である New Project を作成してください．同フォルダ内に timer1_interrupt.X という名前のフォルダが作られます．この timer1_interrupt.X フォルダ内に本稿掲載ページオペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器
からダウンロード・解凍しておいた「timer1_interrupt」フォルダ内の main.c ファイルと PIC16F1705_Files フォルダをコピーしてください．そして，4.3 節と同様にして，Source Files に main.c を，Linker Files に PIC16F1705_Files フォルダ内の set_osc.c と set_timer1.c ファイルを，Header Files に pic16f1705_s.h ファイルを追加して下さい．

6.1.1　実験回路

図 6.1 はタイマ 1 モジュールの実験回路です．5 番ピンに，割り込み処理の所要時間をモニタするために，割り込み処理プログラム実行中に＋ V [V]，それ以外のときに 0 [V] の信号を出力します．この電圧を v_{o_chk} とします．

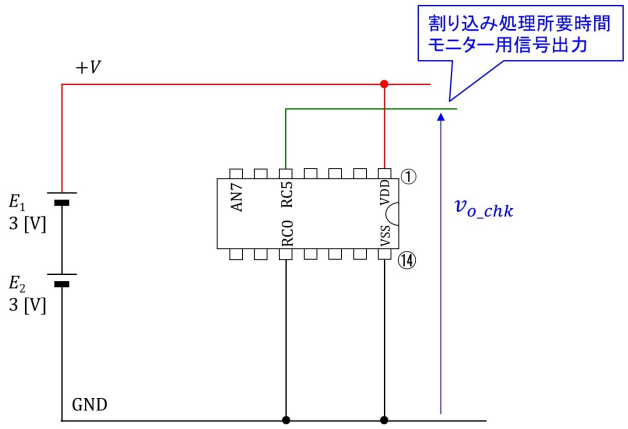


図 6.1: タイマ 1 モジュール実験回路

6.1.2　ヘッダファイルのインクルード（読み込み）

main.c の先頭はヘッダファイルのインクルードを行う命令です．

```
#include    <xc.h>
#include    "./PIC16F1705_Files/pic16f1705_s.h"
```

図 6.2: ヘッダファイルのインクルード

図 6.2 はヘッダファイル xc.h と pic16f1705_s.h をインクルード（読み込み）するプリプロセス命令です．プリプロセス命令は以降のプログラムのコンパイルに先だって実行されます．xc.h は 4.1 節に述べたように，XC8 コンパイラをインストールしたときに xc8￥v2.10￥pic￥include フォルダ内にダウンロードされています．#include <> と山括弧で囲うことでコンパイラがこのヘッダファイルを探し出します．xc.h は PIC16F1705 のデータシート内の各種用語を定義しています．したがって，xc.h のインクルードにより，これら用語を以降のプログラムで利用できるようになります．pic16f1705_s.h は筆者自作の用語定義ファイルです．このファイルは main.c と同じフォルダ内の PIC16F1705_Files フォルダ内にあります．#indlude "./PIC16F1705_Files/..." と 2 重引用符で囲うことでファイルの所在場所を指定します．.（ピリオド）は main.c ファイルと同じフォルダ内を指定します．このヘッダファイル内の用語定義については，本章の各節で必要に応じて順次解説します．

6.1.3　デバイスコンフィギュレーション（デバイス設定）

図 6.3 はプログラムの先頭で，ヘッダファイルのインクルードの次に実行しなければならないデバイスコンフィギュレーション（デイバイス設定）です．PIC16F1705 のデータシートは，Microchip 社の Web ページから無料でダウンロードできます．同社のホームページ

```
#include    <xc.h>
#include    "./PIC16F1705_Files/pic16f1705_s.h"

#pragma config FOSC = 0x04          // 内蔵オシレータオン，RA5をクロック入力用ではなく，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 BOREN = OFF          // 低電圧リセット オフ
#pragma config CLKOUTEN = OFF       // クロック信号出力ピン（RA4）オフ
#pragma config IESO = OFF           // 電源立ち上げ時の内蔵オシレータ→外部オシレータ切替 オフ
#pragma config FCMEN = OFF          // フェイルセーフクロックモニタオフ
#pragma config WRT = OFF            // フラッシュメモリ書き込み保護 オフ
#pragma config ZCDDIS = 1           // ゼロクロス検出停止
#pragma config PPS1WAY = 1          // PPS (Peripheral Pin Select) LOCKを掛ける．本プログラムではI/Oピンへの
                                    // 周辺モジュールの割り当てをプログラム実行中に変えることはしない．）
#pragma config PLLEN = ON           // PLL オン
#pragma config STVREN = OFF         // スタックオーバー／アンダーフローリセット オフ
#pragma config BORV = HI            // 低電圧リセットの設定電圧を2.7[V]に設定 (LOW=1.9 [V])
#pragma config LVP = ON             // 低電圧（例えば3.3 [V]）でのプログラミング オン
```

図 6.3: デバイスコンフィギュレーション（デバイス設定）

から Document Library → Data Sheets → Search by Products → PIC16F1705 とたどることで見つけられます．“PIC16F1705” のキーワードでネット検索をかけた方が手っ取り早いでしょう．このデータシートの DEVICE CONFIGURATION のページに CONFIG1 レジスタと CONFIG2 レジスタの各ビットの名称と説明が載っています．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－（文献 [3]）の 4.2 節には PIC16F1825 のデバイスコンフィギュレーションを記してあります．本稿の PIC16F1705 における相違点を緑字で示します．PIC16F1705 にはゼロクロス点検出と I/O ピンへの周辺モジュールの動的割り当て（プログラム実行中にピン割り当てを変えられる）機能があります．ただし，本稿ではこれらの機能は使いません．これら 2 つの設定を除けば，本稿のデバイスコンフィギュレーションは文献 [3] の PIC16F1825 の設定と同じで，

FOSC = 0x04 : 内蔵オシレータオン, RA5（2 番ピン）をクロック入力用ではなく，I/O 用に設定

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

PLLEN = ON : 内蔵オシレータにより得られたクロックを PLL により 4 倍可とする

BORV = HI : 低電圧リセット電圧を 2.7[V] に設定

と設定しています．実験室レベルでは，これで十分です．

図 6.4 はデバイスコンフィギュレーションにより設定されたオシレータモジュールです．内蔵オシレータと 4 逓倍 PLL (Phase Locked Loop) を利用可能とします．内蔵オシレータの発信周波数は 16 [MHz] で精度は ±1 %です．クロックに高い精度を必要とする場合には，水晶（クリスタル）発振子などをマイコンに外付けして使用します．4 逓倍 PLL はクロック周波数を 4 倍します．図より，内蔵オシレータを使用する場合，8 [MHz] のクロッ

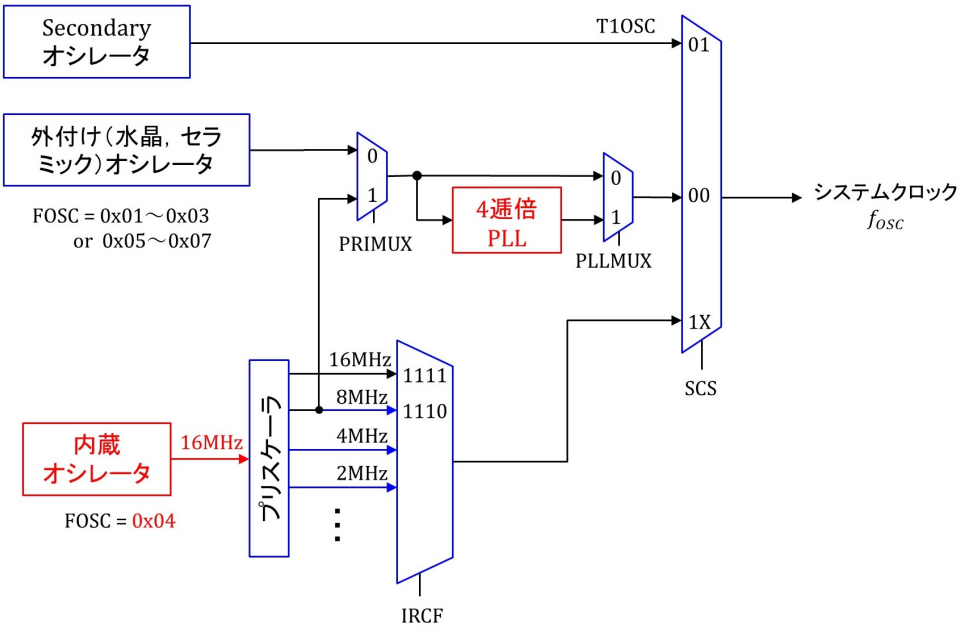


図 6.4: デバイスコンフィギュレーションによるオシレータモジュール設定

クを 4 逓倍 PLL の入力に入れることができます．出力には 32 [MHz] のクロックが得られます．オシレータモジュールの出力がシステムクロック f_{OSC} です．

6.1.4　main 関数

```
void main()
{
        TRISA = 0x00;           // RAxを出力ポートに設定
        TRISC = 0x09;           // RC3（7番ピン），RC0（10番ピン）を入力ポートに設定
        ANSELA = 0x00;
        ANSELC = 0x00;          // RC3（7番ピン），RC0（10番ピン）をディジタル入力に設定

        if(RC0 == 1)            //オシレータの設定
          {
              set_osc(Int_OSC_Freq_8MHz, SysClockSource_detmd_by_Config);
                                // 内蔵オシレータ8MHzとPLLEN = ONで fosc = 32MHzに設定する.
                                // デバイスコンフィギュレーションにてPLLEN = ONとしておくと，
                                // FOSCは内蔵オシレータ8MHzのときだけPLLにより4倍される.
         }else{
              set_osc(Int_OSC_Freq_16MHz, SysClockSource_detmd_by_Config);
                                // 内蔵オシレータ16MHzで，fosc = 16MHzに設定する.
         }

        set_timer1(TMR1_clock_source_FOSC, T1Clock_PreScale_1_1, TMR1_ON);
                                // 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(;;){}               // 無限ループ
}
```

図 6.5: メイン関数

図 6.5 はメイン関数です．RC0(10 番ピン) の入力が 1(5 [V]) のとき $f_{OSC} = 32$ [MHz]，0 （0 [V]) のとき 16 [MHz] に設定しています．

6.1.5　set_osc 関数

set_osc.c

```
#include    <xc.h>
#include    "../PIC16F1705_Files/pic16f1705_s.h"

// OSCCON(Oscillator Control Register)の設定（ヘッダファイルPIC16F1705_s.h参照）
void set_osc(unsigned char x, unsigned char y)
{
        OSCCONbits.IRCF = x;
        OSCCONbits.SCS = y;
}
```

図 6.6: set.osc 関数

メイン関数内の set_osc 関数を図 6.6 に示します．この関数は OSCCON レジスタの IRCF ビットと SCS ビットを設定します．IRCF, SCS などのレジスタに関する用語は PIC16F1705

のデータシートで定義されています．メイン関数では set_osc 関数の引数を Int_OSC_Freq_8MHz などの用語で与えています．これら用語と IRCF ビット，SCS ビットとの対応関係はヘッダファイル pic16f1705_s.h 内で定義しています．

```
// OSCCON(Oscillator Control Register)

    // IRCF  (内蔵オシレータの発信周波数設定)
    #define   Int_OSC_Freq_16MHz    0b1111
    #define   Int_OSC_Freq_8MHz     0b1110
    #define   Int_OSC_Freq_4MHz     0b1101
    #define   Int_OSC_Freq_2MHz     0b1100
    #define   Int_OSC_Freq_1MHz     0b1011
    #define   Int_OSC_Freq_500kHz   0b1010

    // SCS  (システムクロック(FOSC)にどのクロックを利用するかを設定する. )
    #define   SysClockSource_int_OSC_block      0b11
                                        // 内蔵オシレータブロックの出力を用いる.
                                        // このとき，IRCFレジスタで設定した値がfOSCとなる.
                                        // PLLを使用しない設定.
    #define   SysClockSource_32_768kHz          0b01
                                        // T1OSI(2番ピン), T1OSOピン(3番ピン)に32.768kHzの
                                        // 水晶発振器を接続して，その出力を用いる.
    #define   SysClockSource_detmd_by_Config  0b00
                                        // コンフィギュレーション設定のFOSC = INTOSCとPLLEN = ON
                                        // の設定に従う.
                                        // IRCF設定がInt_OSC_Freq_16MHzであれば fOSC= 16 MHzとなる.
                                        //  Int_OSC_Freq_8MHz であれば，PLLにより fOSC = 8 × 4 = 32 MHz
                                        // となる.
//関数の宣言
void set_osc(unsigned char x, unsigned char y);
```

図 6.7: OSCCON レジスタの用語とビットとの対応関係

図 6.7 は OSCCON レジスタに関連する用語と各ビットとの対応関係を示します．

図 6.8 は，set_osc 関数により $f_{OSC} = 16$ MHz] とするオシレータモジュール設定とクロック信号の経路を示します．内蔵オシレータにより 16 [MHz] のクロック信号が生成され，その信号が赤線の経路を通ってそのままシステムクロックと成っています．

図 6.9 は，$f_{OSC} = 32$ MHz] とするオシレータモジュール設定とクロック信号の経路を示します．内蔵オシレータにより生成された 16 [MHz のクロック信号は，プリスケーラにより一旦 8 [MHz] へと下げられた後，PLL により 4 逓倍されて $f_{OSC} = 32$ MHz] が得られます．

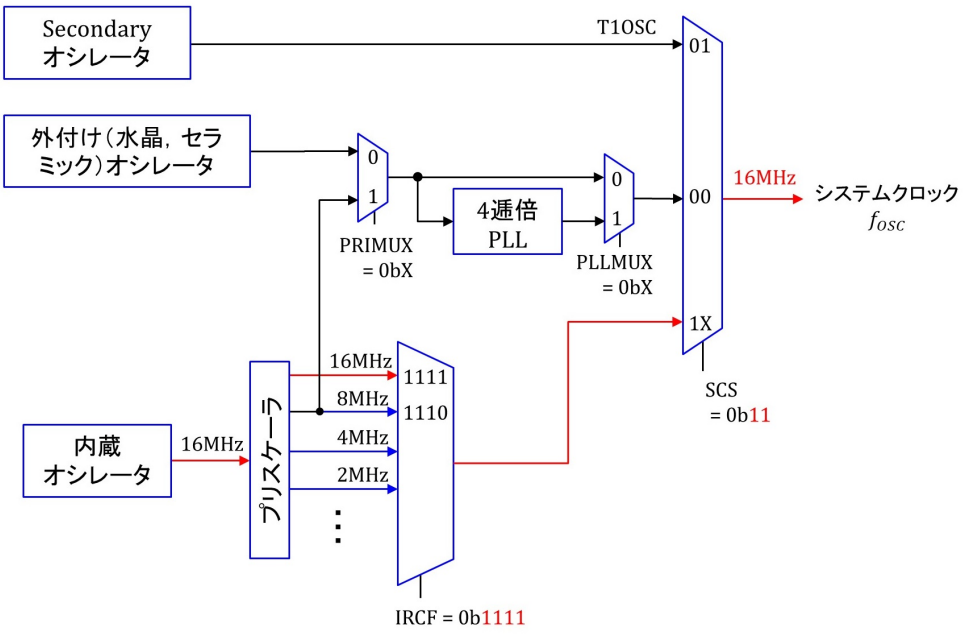


図 6.8: システムクロック $f_{OSC} = 16$ [MHz] とするオシレータモジュール設定

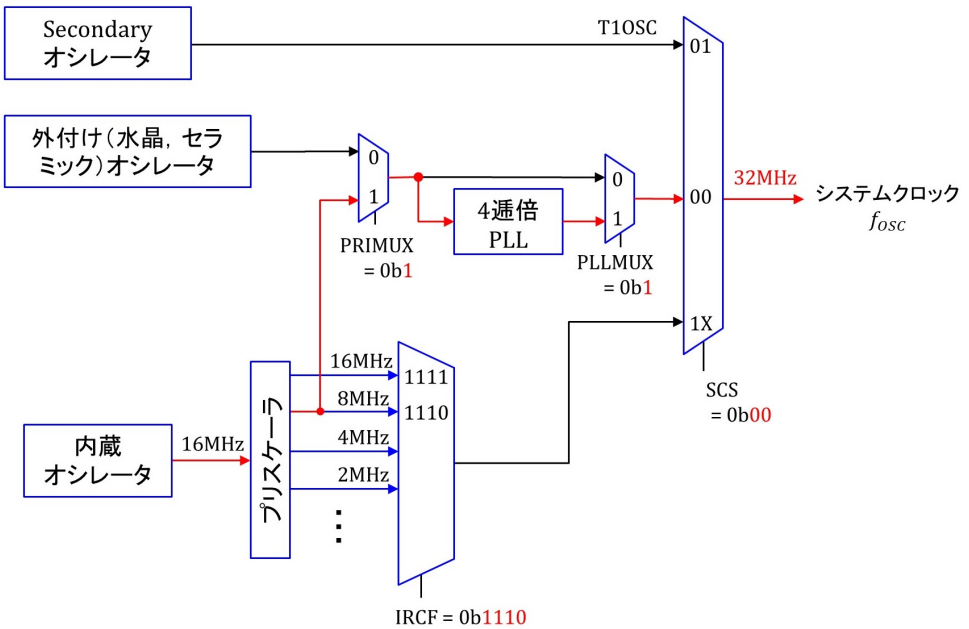


図 6.9: システムクロック $f_{OSC} = 32$ [MHz] とするオシレータモジュール設定

SCS	FOSC	PLLEN	IRCF	PRIMUX	PLLMUX
00	100	0	x	1	0
00	100	1	1110	1	1
00	100	1	≠ 1110	1	0
00	≠ 100	0	x	0	0
≠ 00	x	x	x	x	x

図 6.10: PRIMUX と PLLMUX は SCS, FOSC, PLLEN, IRCF により自動設定される

図 6.9 中の PRIMUX, PLLMUX はプログラム内で設定していません．set_osc 関数により IRCF ビットと SCS ビットを設定し，デバイスコンフィグレーションにより FOSC ビット，PLLEN ビットを設定することで，図 6.10 のように自動設定されます．

6.1.6　set_timer1 関数

```
set_timer1.c
#include <xc.h>
#include "../PIC16F1705_Files/pic16f1705_s.h"

// T1CON(Timer1 Control Register)の設定（ヘッダファイルpic16f1705_s.h参照）
void set_timer1(unsigned char a, unsigned char b, unsigned char c)
{
    T1CONbits.TMR1CS = a;
    T1CONbits.T1CKPS = b;
    T1CONbits.TMR1ON = c;
}
```

図 6.11: set_timer1 関数

図 6.11 はメイン関数内の set_timer1 関数です．この関数は T1CON レジスタの設定をします．set_timer1.c ファイル内にあります．

```
// T1CON(Timer1 Control Register)

  // TMR1CS(タイマ1のクロックソースを設定する)
  #define   TMR1_clock_source_LFINTOSC   0b11   // Low Freq Internal Oscillatorのクロックを利用する.
  #define   TMR1_clock_source_T1Ck_SOSC  0b10   // 外部クロック入力(T1CKI)or SOSCクロックを利用する.
  #define   TMR1_clock_source_FOSC       0b01   // システムクロック(FOSC)を利用する.
  #define   TMR1_clock_source_FOSC_1_4   0b00   // システムクロック(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. )

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

図 6.12: T1CON レジスタの用語とビットとの対応関係

図 6.12 は T1CON レジスタに関連する用語と各ビットとの対応関係を示します．

図 6.13 は set_timer1 関数によるタイマ 1 モジュール設定とクロック信号の経路を示します．システムクロック FOSC の 32 MHz] もしくは 16 [MHz] はプリスケーラをそのまま通りタイマ 1 のクロックとなっています．

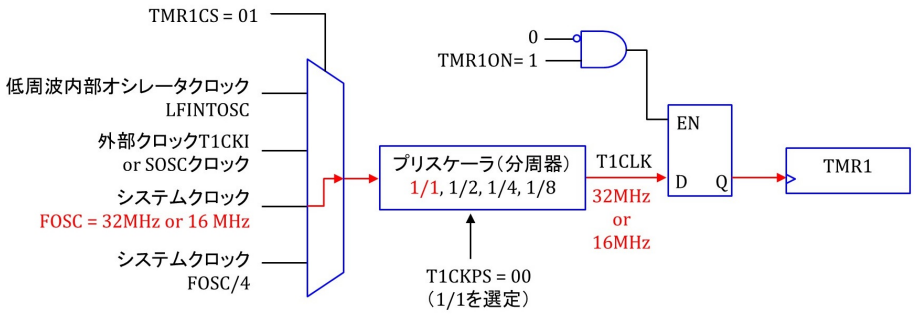


図 6.13: タイマ 1 モジュールの設定

```
set_timer1.c
// 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 6.14: set_interrupt_by_timer1 関数

6.1.7　set_interrupt_by_timer1 関数

図 6.14 はメイン関数内の set_interrupt_by_timer1 関数です．この関数はタイマ 1 による割り込み設定をします．set_timer1.c ファイル内にあります．

6.1.8　interrupt 関数

```
static void __interrupt()          // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    clear_interrupt_flag_of_timer1();
                        // 割り込みフラグクリア．タイマ1においてX からのカウントダウン値が 0 に到達した
                        // 時点で1にセットされ，再割り込みがなされる．
    RC5 = 1;            // ポートCのRC5(5番ピン)に1を出力する．割り込み処理時間のモニタリング用

    set_timer1_count_down_ini_num(3064);
            // タイマ1のカウントダウン値の設定．
            // ..._num(X)の X からカウントダウンを行い，0で再割り込みを発生
            // 割り込み周波数  f_intrpt = 32MHz/(X+136)．X = 3064のとき，f_intrpt =10 [kHz]
            //                 f_intrpt = 16MHz/(X+136)．X = 3064のとき，f_intrpt =5 [kHz]．
            // 割り込み時にAutomatic Context Saving(レジスタ値のスタックへの退避・復帰)と
            // clear_..., RC5 = 1, set_...の実行に計136ステップを要している．
            // Xminは割り込み処理プログラムの処理時間に依存する．このプログラムではXmin = 28．
            // これよりXを小さくしても割り込み周波数は上がらない．

    RC5 = 0;            // ポートCのRC5(5番ピン)に0を出力する．割り込み処理時間のモニタリング用
}
```

図 6.15: インタラプト関数

図 6.15 は main.c ファイル内にあるインタラプト関数です．タイマ 1 の割り込みフラグをクリアして再割り込みを可能とし，タイマ 1 のカウントダウン値を設定するだけのプログラムです．clear_interrupt_flag_of_time1 関数と set_timer1_count_down_ini_num 関数は set_timer1.c ファイルの中に記載してあります．

6.1.9　clear_interrupt_flag_of_timer1 関数

```
set_timer1.c
// 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;
}
```

図 6.16: タイマ 1 による割り込みフラグクリア関数

図 6.16 はタイマ 1 による割り込みフラグクリア関数です．set_timer1.c ファイル内にあります．

```
set_timer1.c
// 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;
}
```

図 6.17: タイマ 1 のカウントダウン値の設定関数

6.1.10　set_timer1_count_down_ini_num 関数

図 6.17 はタイマ 1 のカウントダウン値の設定関数です．set_timer1.c ファイル内にあります．

6.1.11　タイマ 1 による割り込みプログラムのブロック図と実験結果

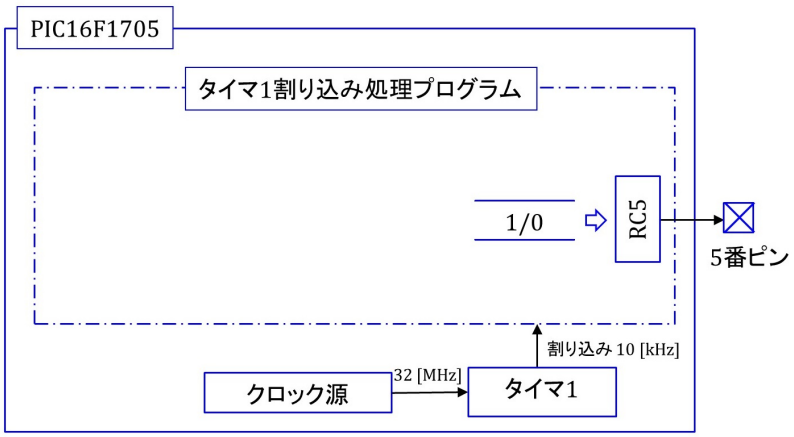


図 6.18: タイマ 1 による割り込みプログラムのブロック図

図 6.18 は本節のタイマ 1 による割り込みプログラムがマイコンの外に対して行っていることをまとめたブロック図です．interrupt 関数がタイマ 1 により 10 [kHz] もしくは 5[kHz] の頻度で 5 番ピンに 1 と 0 を出力します．

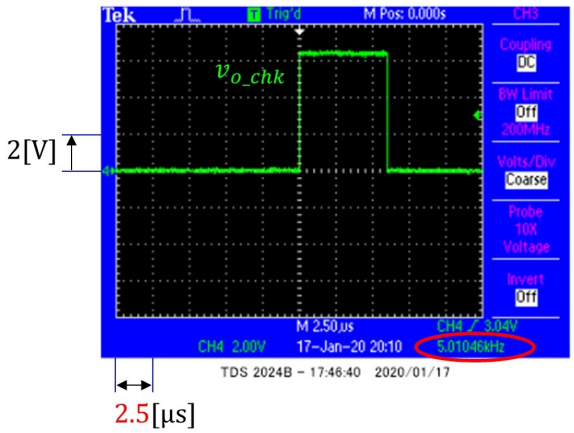


図 6.19: タイマ 1 による割り込みプログラムの実験結果

図 6.19 はタイマ 1 による割り込みプログラムの実験結果です．図 6.1 の電圧 V_{O_CHK} を観測したオシロスコープ画面のスナップショットです．$V_{O\_CHK} \approx 6.3$ [V] である期間が約 6[μs] です．interrupt 関数が 5 番ピンに 1 を出力してから set_timer1_count_down_ini_num 関数を実行した後に 0 を出力するまでに要した時間です．画面の右下にはこのパルス状波形の繰り返し周波数 = 5.01 [kHz] と表示されています．図 6.1 の回路における割り込み周波数は 5.01 [kHz] でした．

6.2　A/D 変換

A/D 変換モジュールの設定と A/D 変換について解説します．4.2 節と同様にして，/MPLABX-Projects/PIC16F1705 フォルダ内にプロジェクト名が AD_Conv である New Project を作成してください．同フォルダ内に AD_Conv.X という名前のフォルダが作られます．この AD_Conv.X フォルダ内に本稿掲載ページ

オペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器

からダウンロード･解凍しておいた「AD_Conv」フォルダ内の main.c ファイルと PIC16F1705_Files フォルダをコピーしてください．そして，4.3 節と同様にして，Source Files に main.c を，Linker Files に PIC16F1705_Files フォルダ内の set_ad_converter.c，set_osc.c と set_timer1.c ファイルを，Header Files に pic16f1705_s.h ファイルを追加して下さい．

6.2.1　実験回路

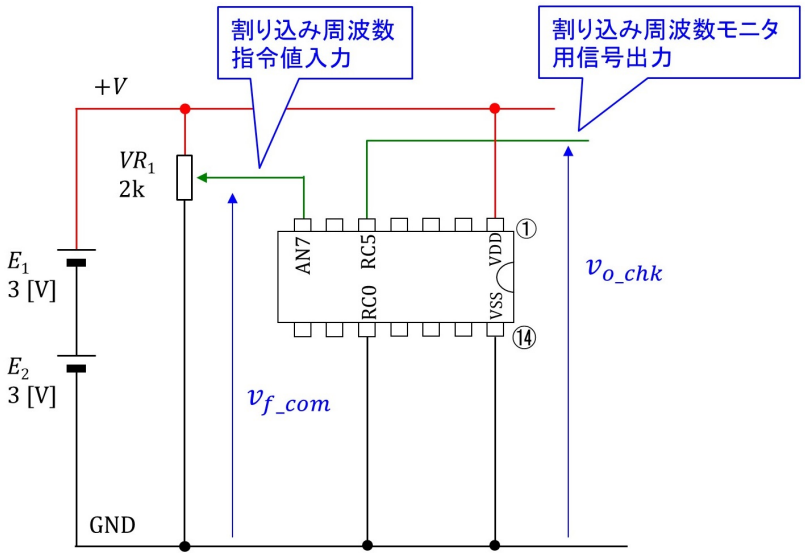


図 6.20: A/D 変換モジュール実験回路

図 6.20 は A/D 変換モジュール実験回路です．図 6.1 の実験回路に可変抵抗 VR_1 を付加した回路です．VR_1 の b 電極（可動電極）のアナログ電圧 v_{f_com} をマイコンの 7 番ピンの入力としています．7 番ピンには A/D 変換モジュールのアナログ入力ピン AN7 が割り当てられています．本節では，v_{f_com} により前節のタイマ 1 による割り込みの周波数を変える実験を行います．

図 6.21 は set_AD_Converter 関数による A/D 変換モジュール設定を示します．正の基準電圧を VDD (図 6.20 の＋ V)，負の基準電圧を VSS (同 GND) として，7 番ピンの AN7 ピンよりアナログ電圧を入力して，10 ビットのデジタル値を得る設定です．変換結果の格

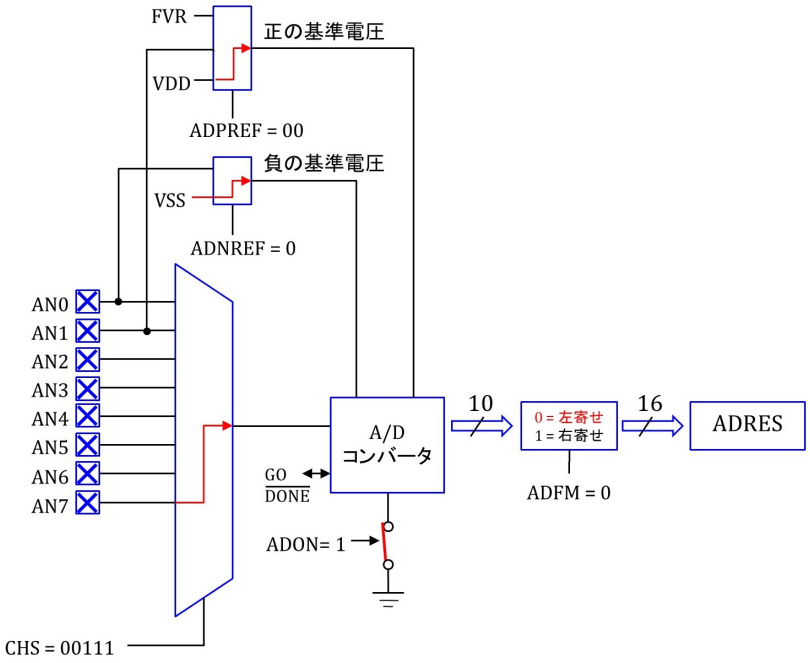


図 6.21: A/D 変換モジュールの設定

納先 ADRES レジスタは 16 ビットであるため，この設定では左寄せ（上位 10 ビットに変換値を入れ，下位 6 ビットは 0 と）しています．

6.2.2　main 関数

```
void main()
{
        （省略）

    // A/Dコンバータの設定
    set_AD_Converter(select_AN7, AD_ON, left_justified, clock_1_32, neg_ref_VSS, pos_ref_VDD);

    for(;;)
    {
        //A/D変換
        start_ad_conversion();                  // A/D変換開始
        ad_out = read_result_of_ad_conversion(); // A/D変換結果を ad_out に格納
        intrpt_period = 0xFFFF - ad_out;        // 割り込み周期の設定
    }
}
```

図 6.22: メイン関数 (A/D 変換)

図 6.22 はメイン関数です．set_AD_Converter 関数により A/D コンバータ（A/D 変換モジュール）を設定しています．そして，for(;;){} の無限ループ内では，A/D 変換の実行とその変換結果から割り込み周期設定を繰り返しています．

6.2.3　初期設定，変換開始，変換結果読み出し関数

```
set_ad_converter.c
#include <xc.h>
#include "../PIC16F1705_Files/pic16f1705_s.h"

// A/D コンバータの設定
void set_AD_Converter(unsigned char a, unsigned char b, unsigned char c, unsigned char d,
                      unsigned char e, unsigned char f)
{
   ADCON0bits.CHS      = a;    // アナログチャネル設定
   ADCON0bits.ADON     = b;    // A/Dコンバータをオン/オフ設定
   ADCON1bits.ADFM     = c;    // 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.
   ADCON1bits.ADCS     = d;    // A/Dコンバータのクロック設定
   ADCON1bits.ADNREF   = e;    // 基準電圧の−側の設定
   ADCON1bits.ADPREF   = f;    // 基準電圧の＋側の設定
}

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 = ADRES;              // 変換結果の読み出し
   return(a);
}
```

図 6.23: A/D 変換モジュール初期設定，変換開始，変換結果読み出し関数

図 6.23 は main 関数内で使用している set_AD_Converter 関数，start_ad_conversion 関数，read_result_of_ad_conversion 関数です．set_ad_converter.c ファイル内にあります．

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

    // CHS(Analog Channel Select bits)
    #define select_AN0   0b00000      //AN0を選定
    #define select_AN1   0b00001      //AN1
    #define select_AN2   0b00010      //AN2
    #define select_AN3   0b00011      //AN3
    #define select_AN4   0b00100      //AN4
    #define select_AN5   0b00101      //AN5
    #define select_AN6   0b00110      //AN6
    #define select_AN7   0b00111      //AN7

    // 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

// ADCON1(A/D Control Register 1)

    // 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

    // ADCS(A/C Conversion Clock Select bits)
    #define    clock_1_2      0b000    // A/Dコンバータのクロックを 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.
```

図 6.24: ADCON0 レジスタ

set_AD_Converter 関数は ADCON0, ADCON1 レジスタを設定します．これらレジスタの各ビットと用語の対応関係を図 6.24，図 6.25 に示します．pic_16f1705_s.h のヘッダファイル内で定義されています．

```
// ADCON1(A/D Control Register 1)

    // ADNREF(A/D Negative Voltage Reference Configuration bit)
    #define    neg_ref_VSS          0b0    // A/Dコンバータの基準電圧の−側をVSSとする.
    #define    neg_ref_exVREFM      0b1    //                              を外部VREF-ピンとする

    // ADPREF(A/D Positive Voltage Reference Configuration bit)
    #define    pos_ref_VDD          0b00   // A/Dコンバータの基準電圧の+側をVDDとする.
    #define    pos_ref_exVREFP      0b10   //                              を外部VREF+ピンとする
    #define    pos_ref_FVR          0b11   // 基準電圧にFVR(Fixed Voltage Reference)の電圧を使用

    (FVR設定は省略)

//関数の宣言
void           set_AD_Converter(unsigned char a, unsigned char b, unsigned char c, unsigned char d,
                                unsigned char e, unsigned char f);
void           start_ad_conversion(void);
unsigned int   read_result_of_ad_conversion(void);
void           start_ad_conversion_ch_select(unsigned int a);
```

図 6.25: ADCON1 レジスタ

6.2.4　interrupt 関数

```
unsigned int   ad_out;
unsigned int   intrpt_period;

static void __interrupt()       // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
    clear_interrupt_flag_of_timer1();
                                // 割り込みフラグクリア．タイマ1において X からのカウントダウン値が 0 に到達した
                                // 時点で1にセットされ，再割り込みがなされる．
    RC5 = 1;                    // ポートCのRC5(5番ピン)に1を出力する．割り込み処理時間のモニタリング用

    set_timer1_count_down_ini_num(intrpt_period);
                    // タイマ1のカウントダウン値の設定．
                    // ..._num(X)の X からカウントダウンを行い，0で再割り込みを発生
                    // 割り込み周波数  f_intrpt = 32MHz/(X+136)．X = 3064のとき，f_intrpt =10 [kHz]．
                    // 割り込み時にAutomatic Context Saving(レジスタ値のスタックへの退避・復帰)と
                    // clear_..., RC5 = 1, set_...の実行に計136ステップを要している．
                    // Xminは割り込み処理プログラムの処理時間に依存する．このプログラムでは Xmin = 28．
                    // これよりXを小さくしても割り込み周波数は上がらない．

  RC5 = 0;                      // ポートCのRC5(5番ピン)に0を出力する．割り込み発生のモニタリング用
}
```

図 6.26: インタラプト関数 (A/D 変換)

図 6.26 はタイマ 1 による割り込み処理関数です．図 6.15 の割り込み処理関数との違いは set_timer1_count_down_ini_num 関数の引数を A/D 変換結果から設定した割り込み周期 intrpt_period にしている点だけです．

6.2.5　A/D 変換プログラムのブロック図と実験結果

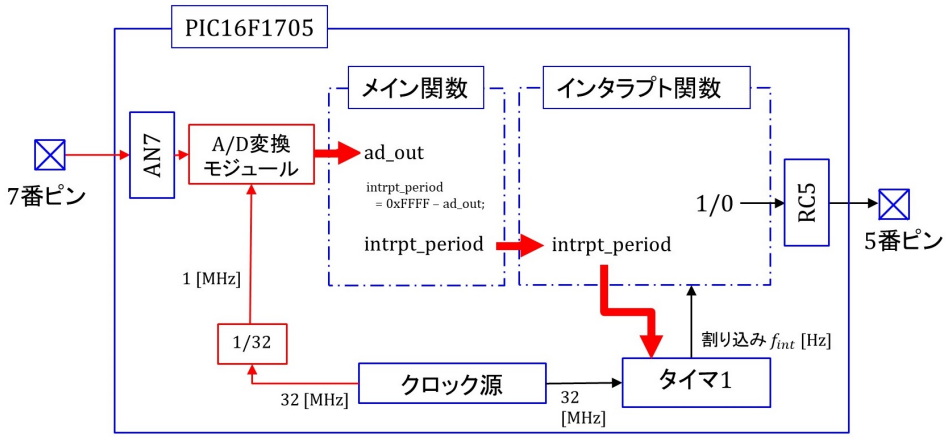


図 6.27: A/D 変換プログラムのブロック図

図 6.27 は本節の A/D 変換プログラムが行っていることをまとめたブロック図です．7 番ピンからのアナログ電圧値を A/D 変換モジュールによりディジタル値に変換してマイコンに取り込み，タイマ 1 による割り込み周期を設定しています．

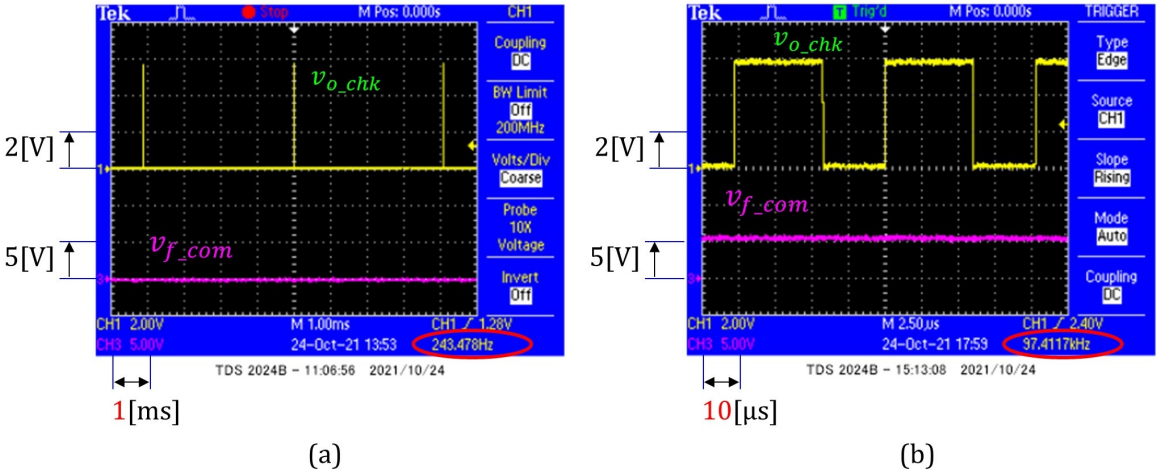


図 6.28: A/D 変換プログラムの実験結果

図 6.28 は A/D 変換プログラムの実験結果です．(a) は横軸が 1 [ms] です．v_{f_com} が最も低く，v_{o_chk} のパルス状波形の繰り返し周波数が約 240 [Hz] の場合，(b) は横軸が 10[μs] です．v_{f_com} が最も高く，繰り返し周波数が約 97 [kHz] です．

6.3　D/A 変換

D/A 変換モジュール設定と D/A 変換について解説します．4.2 節と同様にして，/MPLABX-Projects/PIC16F1705 フォルダ内にプロジェクト名が DA_Conv である New Project を作成してください．同フォルダ内に DA_Conv.X という名前のフォルダが作られます．この DA_Conv.X フォルダ内に本稿掲載ページ

オペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器

からダウンロード･解凍しておいた「DA_Conv」フォルダ内の main.c ファイルと PIC16F1705_Files フォルダをコピーしてください．そして，4.3 節と同様にして，Source Files に main.c を，Linker Files に PIC16F1705_Files フォルダ内の set_da_converter.c，set_ad _converter.c，set_osc.c と set_timer1.c ファイルを，Header Files に sin_data.h と pic16f1705_s.h ファイルを追加して下さい．

6.3.1　実験回路

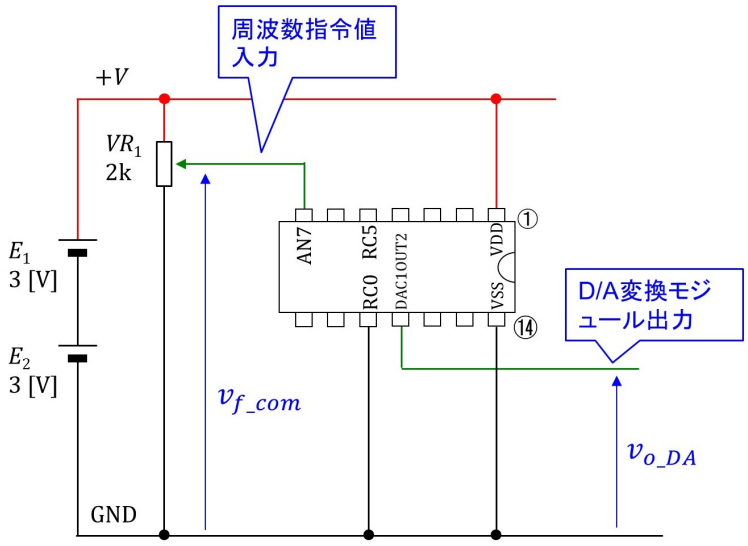


図 6.29: D/A 変換モジュール実験回路

図 6.29 は D/A 変換モジュール実験回路です．図 6.20 と同じ回路です．D/A 変換結果を 11 番ピン (DAC1OUT2 ピン) に出力します．この電圧を v_{o_DA} とします．

図 6.30 は set_DA_Converter 関数による D/A 変換モジュール設定を示します．正の基準電圧を VDD (図 6.29 の＋ V)，負の基準電圧を VSS (同 GND) として，256 個の抵抗 R により両電圧間を 256 段階に分圧しておきます．DAC1R ビットの値に応じてマルチプレクサにより 256 段階の電圧から 1 つの段階の電圧を DAC1OUT2 ピンに出力します．

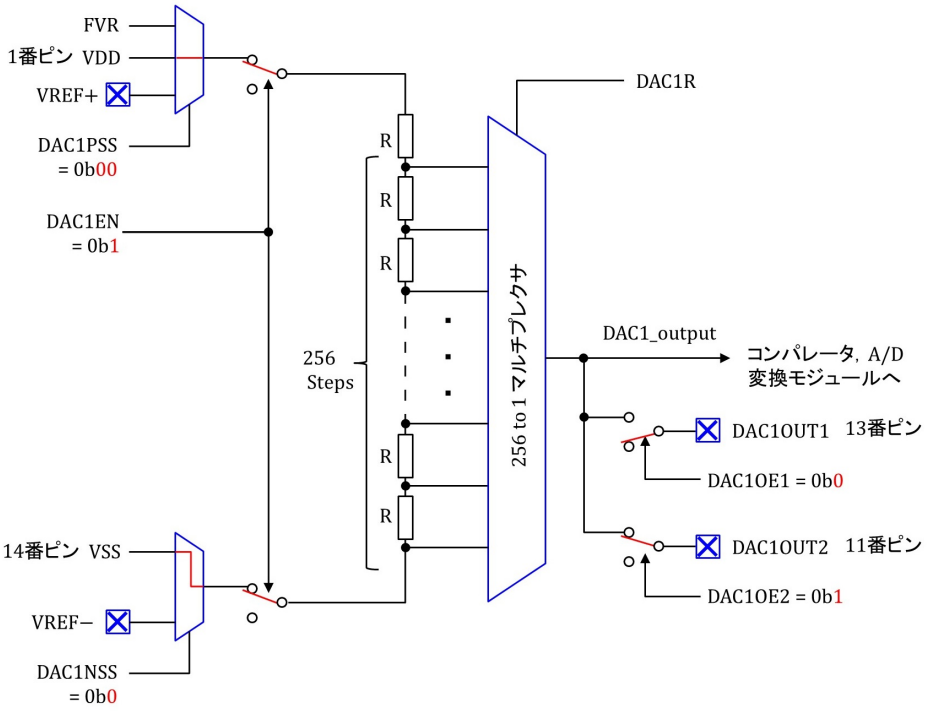


図 6.30: D/A 変換モジュールの設定

6.3.2　main 関数

図 6.31 はメイン関数です．set_DA_Converter 関数により D/A コンバータ（D/A 変換モジュール）を設定しています．そして for(;;) の無限ループ内では，intrpt_period の下限値を max_func 関数により 172 に設定しています．これより小さな数値を intrpt_period に格納しても，タイマ 1 による割り込み周期は短くならないためです．

図 6.32 は max_func 関数です．

6.3.3　set_DA_Converter 関数

図 6.33 は set_DA_Converter 関数です．set_da_converter.c ファイル内にあります．

```
void main()
{
        （省略）

    // D/Aコンバータの設定
    set_DA_Converter(DAC_on, DAC_not_to_DAC1OUT1, DAC_to_DAC1OUT2, DAC_Positive_Source_to_VDD,
                     DAC_Negative_Source_to_VSS);

    for(;;)
    {
        //A/D変換
        start_ad_conversion();                          // A/D変換開始
        ad_out = read_result_of_ad_conversion();   // A/D変換結果を ad_out に格納
        intrpt_period = max_func(0xFFFF - ad_out, 172);
                                                        // 割り込み周波数の設定．intpr_periodの下限値を172に設定
    }
}
```

図 6.31: メイン関数 (D/A 変換)

```
// maximum値計算関数
unsigned int max_func(unsigned int a, unsigned int b)
{
    if(a > b)
    {
    return(a);
    }else{
    return(b);
    }
}
```

図 6.32: 大きい方を求める関数

set_da_converter.c

```
// D/A変換モジュールの設定
void set_DA_Converter(unsigned char a, unsigned char b, unsigned char c, unsigned char d,
                      unsigned char e)
{
    DAC1CON0bits.DAC1EN  = a;      // D/A変換モジュールをオン／オフ設定
    DAC1CON0bits.DAC1OE1 = b;      // D/A変換モジュール出力をDAC1OUT1に接続／非接続設定
    DAC1CON0bits.DAC1OE2 = c;        // D/A変換モジュール出力をDAC1OUT2に接続／非接続設定
    DAC1CON0bits.DAC1PSS = d;      // 正電源選定
    DAC1CON0bits.DAC1NSS = e;        // 負電源選定
}
```

図 6.33: set_DA_Converter 関数

```
// DAC1CON0(Voltage Reference Control Register 0)

  // DAC1EN (DAC1 Enable bit)
  #define   DAC_on      0b1       // DA変換器をオンとする.
  #define   DAC_off     0b0       //              オフとする.

  // DAC1OE1 (DAC1 Voltage Output 1 Enable bit)
  #define   DAC_to_DAC1OUT1         0b1    // D/Aコンバータ出力をDAC1OUT1ピンに接続
  #define   DAC_not_to_DAC1OUT1     0b0    //                    DAC1OUT1ピンに非接続

  // DAC1OE2 (DAC1 Voltage Output 1 Enable bit)
  #define   DAC_to_DAC1OUT2         0b1    // D/Aコンバータ出力をDAC1OUT2ピンに接続
  #define   DAC_not_to_DAC1OUT2     0b0    //                    DAC1OUT2ピンに非接続

  // DAC1PSS (DAC1 Positive Source Select bit)
  #define   DAC_Positive_Source_to_FVR      0b10  // D/Aコンバータの正電圧源をFVRとする.
  #define   DAC_Positive_Source_to_Vref_P   0b01  //                          Vref+とする.
  #define   DAC_Positive_Source_to_VDD      0b00  //                          VDDとする.

  // DAC1NSS (DAC1 Negative Source Select bit)
  #define   DAC_Negative_Source_to_Vref_N  0b1    // D/Aコンバータの負電圧源をVref-とする.
  #define   DAC_Negative_Source_to_VSS     0b0    //                          VSSとする.

//関数の宣言
void    set_DA_Converter(unsigned char a, unsigned char b, unsigned char c, unsigned char d,
                         unsigned char e);
void    set_DA_Output(char a);
```

図 6.34: DAC1CON0 レジスタ

図 6.34 は DAC1CON0 レジスタの各ビットと用語の対応関係を示します．pic_16f1705_s.h のヘッダファイル内で定義されています．

6.3.4　interrupt 関数

```
#include <xc.h>
#include "./PIC16F1705_Files/pic16f1705_s.h"
#include   "./PIC16F1705_Files/sin_data.h"

（省略）

// 正弦波形生成用データ
#define   sin_div       800          // 1周期間のデータ数
unsigned int   sin_step = 0;         //正弦波データの番地カウンタ

static void __interrupt()            // Timer1 による割り込み処理ルーチン
timer1_isr(void)
{
      clear_interrupt_flag_of_timer1();       // 割り込みフラグクリア
      RC5 = 1;                                // ポートCのRC5(5番ピン)に1を出力する.
      set_timer1_count_down_ini_num(intrpt_period);
                  //（省略）
                  // Xminは割り込み処理プログラムの処理時間に依存する.
                  // このプログラムでは Xmin = 172. これよりXを小さくしても割り込み周波数は上がらない.
                  // また, intrpt_periodの更新ができなくなる暴走が起きる.

      set_DA_Output(sin_data[sin_step++]);      // D/A変換モジュールに正弦波データをセット

      if(sin_step >= sin_div)                   // sin_step = sin_divにてsin_step = 0 にリセット
      {
            sin_step = 0;
      }

      RC5 = 0;                                  // ポートCのRC5(5番ピン)に0を出力する.
}
```

図 6.35: インタラプト関数 (D/A 変換)

図 6.35 は D/A 変換プログラムのインタラプト関数です．D/A 変換器により正弦波を出力するために正弦波データのテーブルを sin_data.h のヘッダファイルに用意しておきます．sin_data[sin_step] ($0 \leq$ sin_step ≤ 255) により，sin_data.h ファイル内のデータを読み出すことができます．sin_step は正弦波データの番地です．set_DA_Output 関数により DAC1R ビットに正弦波データを格納します．その後の if 文では sin_step が 799 を超えたら番地を 0 にもどしています．

図 6.36 は set_DA_Output 関数です．set_da_converter.c ファイル内にあります．DAC1R ビットは DAC1CON1 レジスタ内にあります．

6.3.5　D/A 変換プログラムのブロック図と実験結果

図 6.37 は本節の D/A 変換プログラムが行っていることをまとめたブロック図です．割り込み毎に D/A 変換モジュールの DAC1R ビットの正弦波データを更新している点が A/D 変換プログラムと異なります．

set_da_converter.c

```
// D/A変換モジュールの出力値設定
void set_DA_Output(char a)
{
    DAC1CON1bits.DAC1R = a;        // D/A変換モジュールの出力値設定
}
```

図 6.36: set_DA_Output 関数

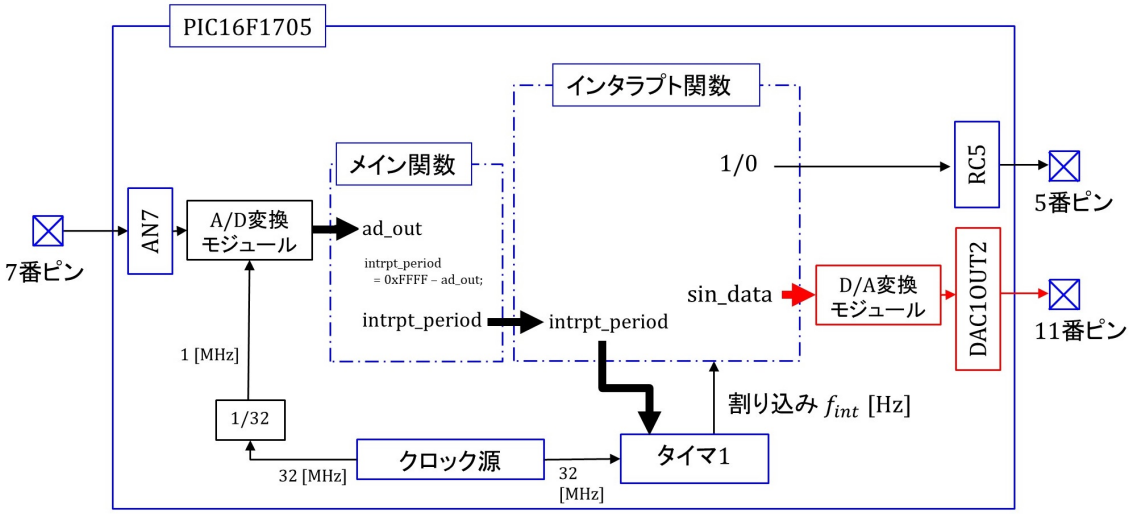


図 6.37: D/A 変換プログラムのブロック図

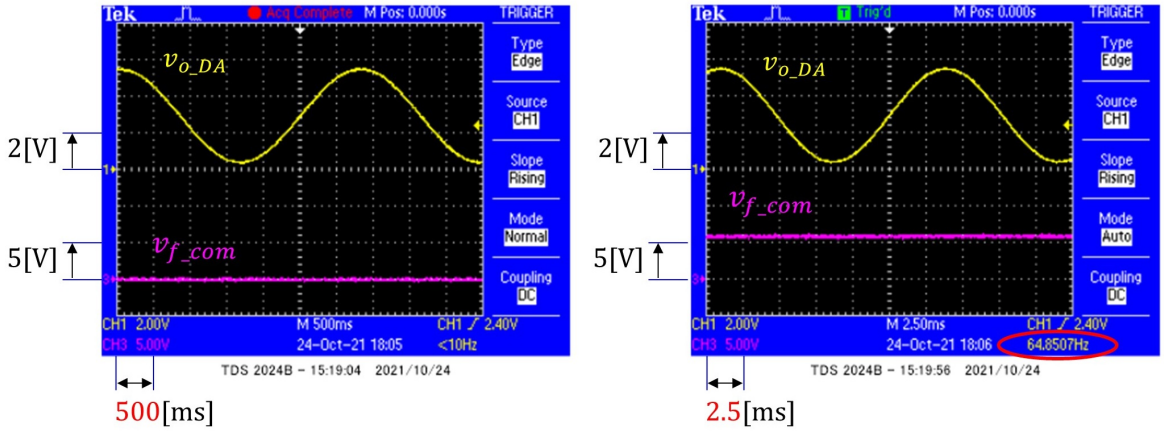


図 6.38: D/A 変換プログラムの実験結果

図 6.38 は D/A 変換プログラムの実験結果です．(a) は横軸が 500 [ms] です．v_{f_com} が最も低く，D/A 変換器の出力電圧 v_{o_DA} の正弦波の周波数が約 0.3 [Hz] の場合，(b) は横軸が 2.5 [ms] です．v_{f_com} が最も高く，繰り返し周波数が約 65 [Hz] です．

6.4　オペアンプの利用

オペアンプモジュールの設定とオペアンプ利用について解説します．4.2 節と同様にして，/MPLABXProjects/PIC16F1705 フォルダ内にプロジェクト名が OP_Amp である New Project を作成してください．同フォルダ内に OP_Amp.X という名前のフォルダが作られます．この OP_Amp.X フォルダ内に本稿掲載ページ

オペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器

からダウンロード･解凍しておいた「OP_Amp」フォルダ内の main.c ファイルと PIC16F1705_Files フォルダをコピーしてください．そして，4.3 節と同様にして，Source Files に main.c を，Linker Files に PIC16F1705_Files フォルダ内の set_op_amp.c，set_da_converter.c，set_ad_converter.c，set_osc.c と set_timer1.c ファイルを，Header Files に sin_data.h と pic16f1705_s.h ファイルを追加して下さい．

6.4.1　実験回路

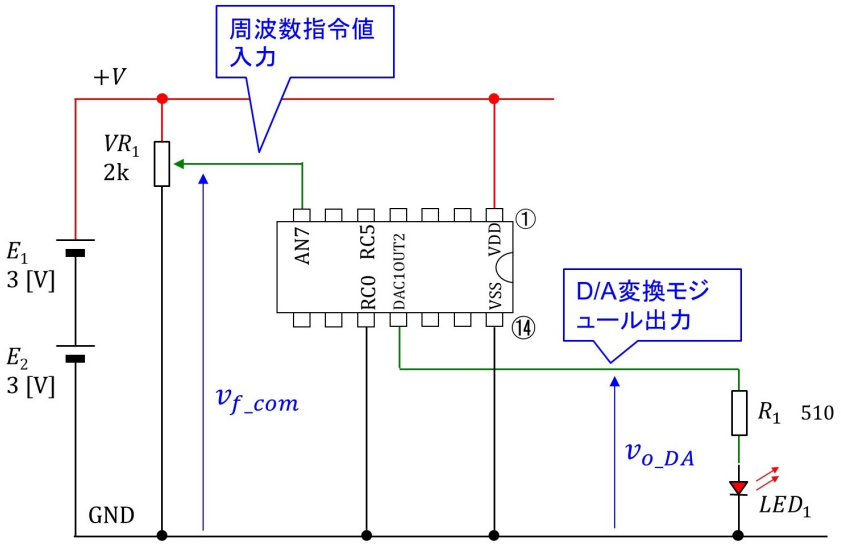


図 6.39: D/A 変換モジュール実験回路 (LED 負荷)

図 6.40 は D/A 変換モジュール実験回路に抵抗 R_1 と発光ダイオード LED_1 をつないだ回路です．

図 6.40 は図 6.40 の実験結果です．v_{o_DA} は大きく歪んでしまいました．

図 6.41 はオペアンプモジュールの実験回路です．8 番ピンに OPA1OUT ピンが割り当てられています．このピンの出力電圧を $v_{o\_OP\_Amp}$ とします．

図 6.42 は set_Op_Amp 関数によるオペアンプモジュール設定を示します．オペアンプをバッファアンプとして使用する設定です．

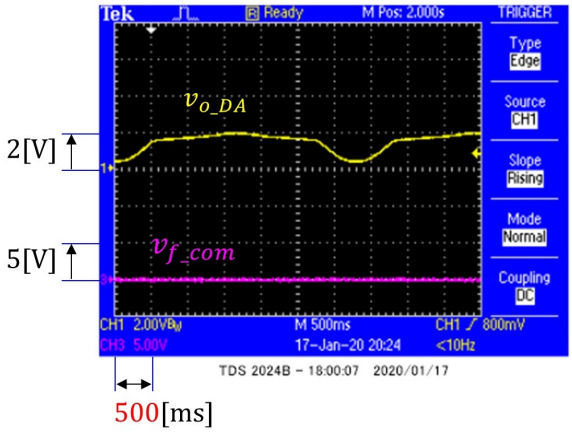


図 6.40: D/A 変換モジュール実験回路 (LED 負荷)

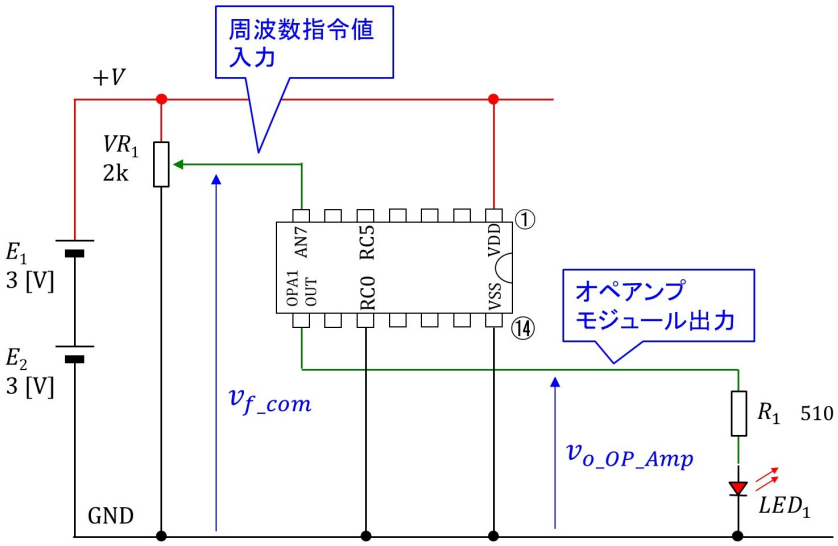


図 6.41: オペアンプモジュール実験回路

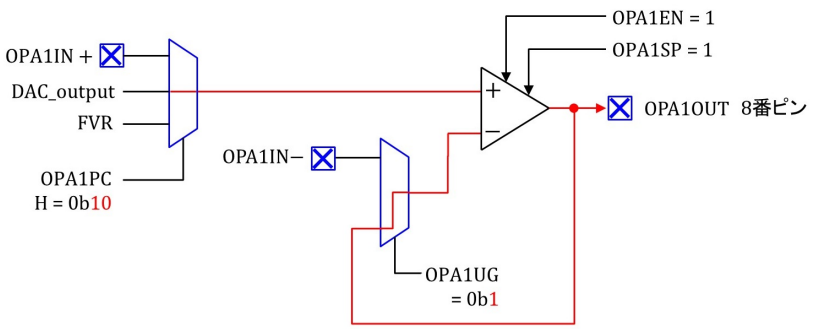


図 6.42: オペアンプモジュールの設定

6.4.2　main 関数

```
void main()
{
        （省略）

    // オペアンプの設定
    set_Op_Amp1(Op_Amp1_on, Op_Amp1_High_GBWP, Op_Amp1_Unity_Gain,
                Op_Amp1_Pos_Input_to_DAC_output);

    for(;;)
    {
        //A/D変換
        start_ad_conversion();                          // A/D変換開始
        ad_out = read_result_of_ad_conversion();   // A/D変換結果を ad_out に格納
        intrpt_period = max_func(0xFFFF - ad_out, 172);
                                                        // 割り込み周波数の設定．intpr_periodの
                                                        //下限値を172に設定
    }
}
```

図 6.43: メイン関数 (オペアンプ)

図 6.43 はメイン関数です．set_Op_Amp1 関数によりオペアンプモジュールを設定しています．

6.4.3　set_Op_Amp1 関数

set_op_amp.c

```
// オペアンプの設定
void    set_Op_Amp1(unsigned char a, unsigned char b, unsigned char c, unsigned char d)
{
    OPA1CONbits.OPA1EN =  a;    // オン／オフ設定
    OPA1CONbits.OPA1SP =  b;    // 高ゲイン×バンド幅モードで使用／不使用
    OPA1CONbits.OPA1UG =  c;    // ボルテージフォロワで使用／－入力をOPA1IN−に接続
    OPA1CONbits.OPA1PCH = d;    // ＋入力をFVR/DAコンバータ出力/OPA1IN＋に接続
}
```

図 6.44: set_Op_Amp1 関数

図 6.44 は set_Op_Amp1 関数です．OPA1CON レジスタの各ビットを設定します．

```
// OPA1CON (Op Amp Control Register)

  // OPA1EN (Op Amp1 Enable bit)
  #define   Op_Amp1_on        0b1                 // Op Amp1をオンとする.
  #define   Op_Amp1_off       0b0                 // Op Amp1をオフとする.

  // OPA1SP (Op Amp1 Speed/Power Select bit)
  #define   Op_Amp1_High_GBWP     0b1             // Op Amp1を高ゲイン×バンド幅モードに設定
  #define   Op_Amp1_not_H_GBWP  0b0               // データシートには0は使用しないとある.

  // OPA1UG (Op Amp1 Unitiy Gain Select bit)
  #define   Op_Amp1_Unity_Gain              0b1   // Op Amp1をボルテージフォロワに設定
  #define   Op_Amp1_Neg_Input_to_OPA1IN   0b0     // Op Amp1の－入力をOPA1IN-に接続

  // OPA1PCH (Op Amp1 Non-Inverting Channenl Selecton bits)
  #define   Op_Amp1_Pos_Input_to_FVR          0b11 // ＋入力をFVRに接続
  #define   Op_Amp1_Pos_Input_to_DAC_output   0b10 //          D/Aコンバータ出力に接続
  #define   Op_Amp1_Pos_Input_to_OPA1IN_P     0b00 //          OPA1IN+に接続

// 関数の宣言
void    set_Op_Amp1(unsigned char a, unsigned char b, unsigned char c, unsigned char d);
```

図 6.45: OPA1CON レジスタ

図 6.45 は OPA1CON レジスタの各ビットと用語の対応関係を示します．pic_16f1705_s.h のヘッダファイル内で定義されています．

6.4.4　オペアンププログラムのブロック図と実験結果

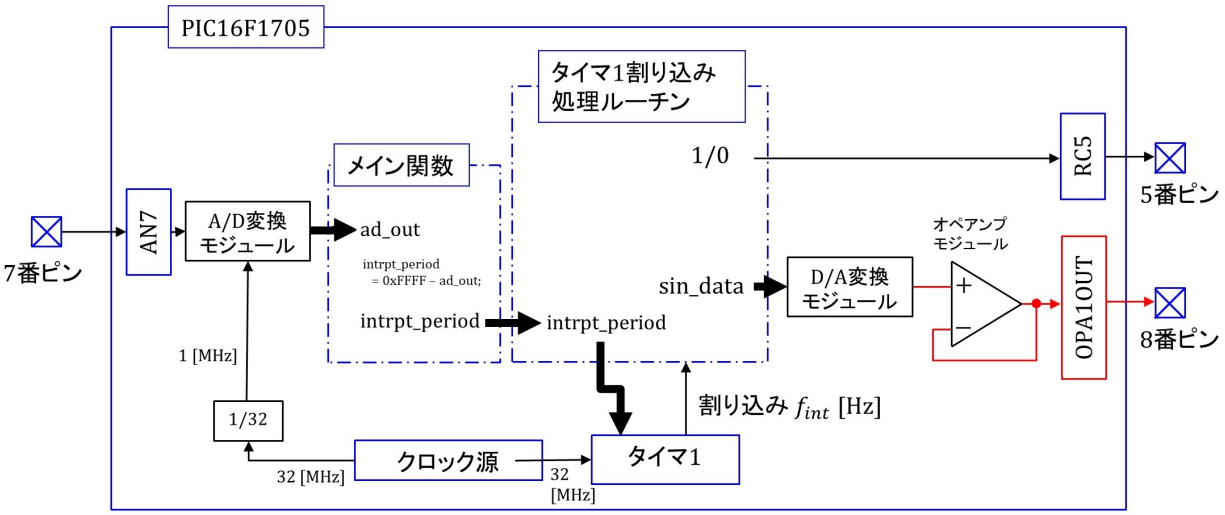


図 6.46: オペアンププログラムのブロック図

図 6.46 は本節のオペアンププログラムが行っていることをまとめたブロック図です．D/A 変換モジュールの出力をオペアンプによるバッファアンプに通して OPA1OUT ピンより出力しています．

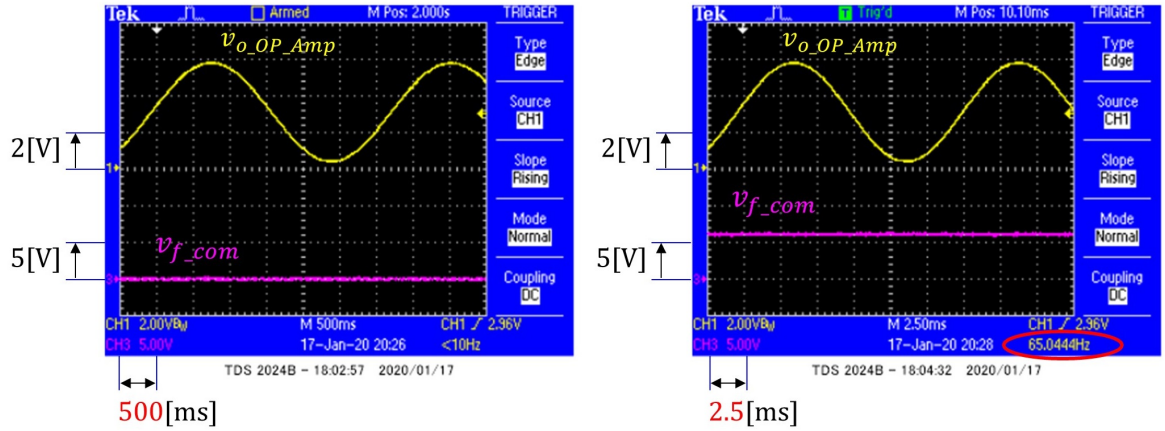


図 6.47: オペアンププログラムの実験結果

図 6.47 はオペアンププログラムの実験結果です．LED 負荷がつながっていても $v_{o\_OP\_Amp}$ の波形に歪みは見られませんでした．なお，オペアンプの出力電流は最大約 8 [mA] でした．

6.5　オペアンプによるローパスフィルタ

以上で，タイマ，A/D 変換，D/A 変換，オペアンプの各モジュールの本稿での使い方を解説しました．第 5 章の sin_rectangular_wave_ generator では新しく使うモジュールはありません．main 関数内の for(;;) 文の無限ループにおいて，可聴域モードでは正弦波データの間引き方と周波数指令値にヒステリシスを持たせる工夫，可視域モードでは低周波域と高周波域でボリューム感度を変える工夫をしてあります．ヒステリシス効果はヒステリシス幅 th_hys を 0 にすると違いが分かります．ボリューム感度調整の効果は ad_out_mode = ad_out に変更すると違いが分かります．

本稿の最後にオペアンプによるフィルタ回路の応用例を紹介します．図 5.4(c) の実験波形では，1 周期 800 個の正弦波データから 100 個ずつ跳ばし読みして，1 周期を 8 個のみのデータで波形を生成するため，階段形状が顕著に現れていました．なお，同図 (a) は 800 個のデータを 1 個ずつ読み出し，(b) は 20 個ずつ跳ばし読みした結果です．PIC16F1705 はオペアンプを 2 個内蔵しているので，もう一つのオペアンプを使ってローパスフィルタを構成し，この階段状波形の成形を行います．

4.2 節と同様にして，/MPLABXProjects/PIC16F1705 フォルダ内にプロジェクト名が sin_rectangular_wave_generator_filter.X である New Project を作成してください．同フォルダ内に sin_rectangular_wave_generator_filter.X という名前のフォルダが作られます．このフォルダ内に本稿掲載ページ

オペアンプ内蔵形 PIC16F1705 による正弦波・矩形波発生器

からダウンロード・解凍しておいた「sin_rectangular_wave_generator_filter.X」フォルダ内の main.c ファイルと PIC16F1705_Files フォルダをコピーしてください．そして，4.3 節と同様にして，Source Files に main.c を，Linker Files に PIC16F1705_Files フォルダ内の set_op_amp.c，set_da_converter.c，set_ad_converter.c，set_osc.c と set_timer1.c ファイルを，Header Files に f_mul_data.h, sin_data.h と pic16f1705_s.h ファイルを追加して下さい．

6.5.1　実験回路

図 6.48 はオペアンプによるローパスフィルタの実験回路です．8 番ピンが 1 個目のオペアンプの出力ピンです．このピンから図 5.4 の階段波形が出力されます．これを v_{o_sin} とします．もう 1 個のオペアンプは 5 番ピンが＋入力ピン，6 番ピンが－入力ピン，そして 7 番ピンが出力ピンです．この出力電圧を v_{o_fil} とします．周波数指令値は 3 番ピン（AN3 ピン) へと移してあります..

図 6.49 はフィルタ用のオペアンプモジュール設定です．5 番ピンを＋入力，6 番ピンを－入力とし，出力を 7 番ピンにつなぎます．

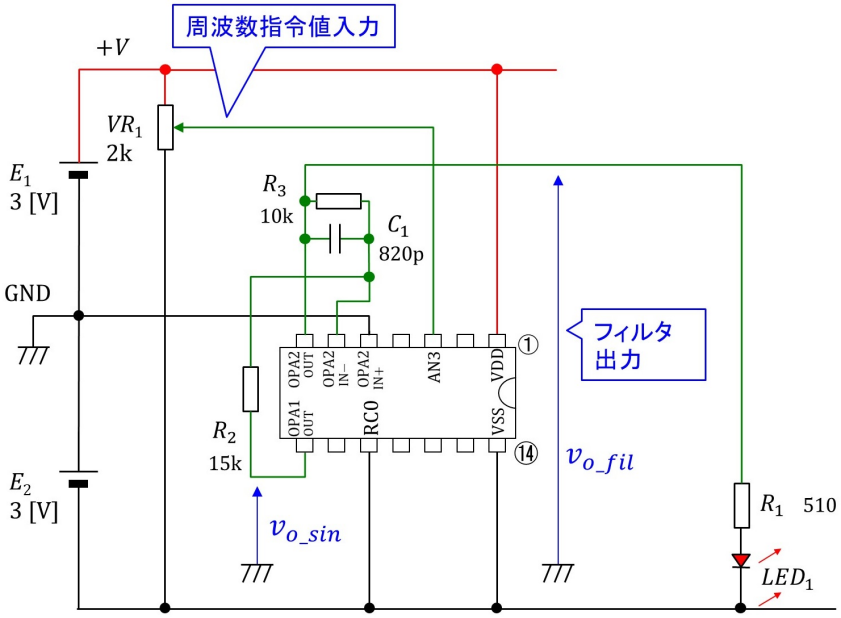


図 6.48: オペアンプによるローパスフィルタの実験回路

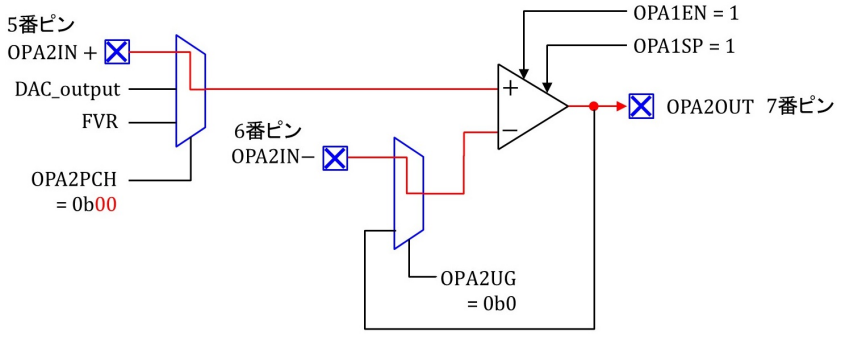


図 6.49: オペアンプ 2 モジュールの設定

6.5.2　ローパスフィルタの周波数特性

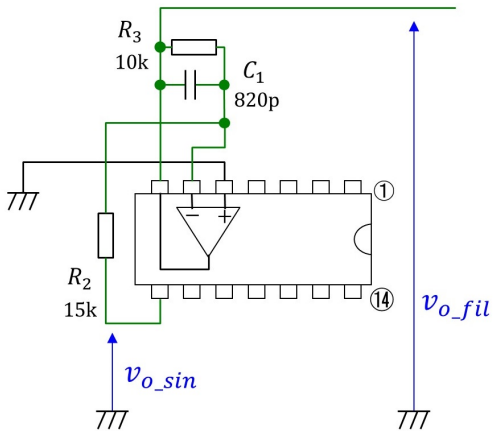


図 6.50: オペアンプによるローパスフィルタ

図 6.50 は 5, 6, 7 番ピンにつながるオペアンプを記号で示します．v_{o_fil}, v_{o_sin} の実効値をそれぞれ V_{o_fil}, V_{o_sin} とすると，両者の関係は

$$V_{o\_fil} = -\frac{R_3}{R_1}\frac{1}{1+j\omega C_1 R_3} \tag{6.1}$$

となります．ローパスフィルタのゲイン G は

$$\begin{aligned} G &= 20log_{10}\frac{|V_{0\_fil}|}{|V_{o\_sin}|} \\ &= 20log_{10}\frac{R_3}{R_2}\frac{1}{\sqrt{1+(\omega C_1 R_3)^2}} \end{aligned} \tag{6.2}$$

です．また，v_{o_fil} の v_{o_sin} に対する位相差 $\varphi_{fil-sin}$ は

$$\varphi_{fil-sin} = -\pi - \tan^{-1}\omega C_1 R_3 \tag{6.3}$$

です．カットオフ周波数 f_c は

$$\begin{aligned} f_c &= \frac{1}{2\pi C_1 R_3} \\ &= \frac{1}{2\pi \times 820 \times 10^{-12} \times 10^4} \\ &= 19.4[\mathrm{kHz}] \end{aligned} \tag{6.4}$$

と求まります．$f \ll f_c$ にて

$$\begin{aligned} G &\approx 20log_{10}\frac{R_3}{R_2} \\ &= 20log_{10}\frac{10}{15} \\ &= -3.52[\mathrm{dB}] \\ \varphi_{fil-sin} &= -\pi \end{aligned} \tag{6.5}$$

です．$f = f_c$ にて

$$\begin{aligned} G &\approx 20log_{10}\frac{R_3}{R_2}\frac{1}{\sqrt{2}} \\ &= -6.53[\mathrm{dB}] \\ \varphi_{fil-sin} &= -\frac{3\pi}{2} \end{aligned} \tag{6.6}$$

となります．

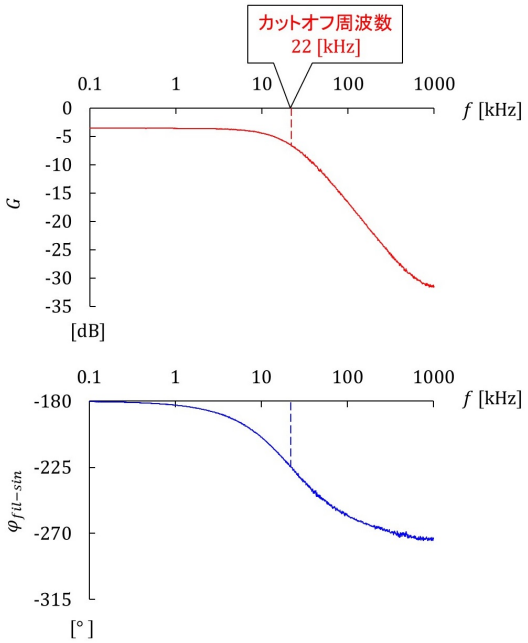


図 6.51: ローパスフィルタの周波数特性の実験結果

図 6.51 はオペアンプによるローパスフィルタの周波数特性の実験結果です．横軸は周波数で，対数目盛です．縦軸は上のグラフが G，下のグラフが $\varphi_{fil-sin}$ です．$f \ll f_c$ にて

$$\begin{aligned} G &= -3.56[\mathrm{dB}] \\ \varphi_{fil-sin} &= -180[^\circ\ ] \end{aligned} \tag{6.7}$$

でした．理論値に近い結果が得られました．一方，f_c は $G = -6.56[\mathrm{dB}], \varphi = -225[^\circ\ ]$ となる辺りから

$$f_c \approx 22[\mathrm{kHz}] \tag{6.8}$$

と得られました．これは理論値より少し大きな値となりました．実験に使用した R_3, C_1 の値を LCR メータで測定したところ，10 [kHz] の信号による計測結果が $R_3 = 10.3[\mathrm{k\Omega}], C_1 = 734$ [pF] でした．この計測値より計算される f_c は

$$\begin{aligned} f_c &= \frac{1}{2\pi C_1 R_3} \\ &= 21.1[\mathrm{kHz}] \end{aligned} \tag{6.9}$$

です．実験値はこの計算値に近い値でした．

6.5.3　main 関数

```
void main()
{
        （省略）

  // オペアンプの設定
  set_Op_Amp2(Op_Amp2_on, Op_Amp2_High_GBWP, Op_Amp2_Neg_Input_to_OPA2IN,
        Op_Amp2_Pos_Input_to_OPA2IN_P);

        （省略）

}
```

図 6.52: main 関数（ローパスフィルタ）

図 6.52 は main 関数です．省略箇所は sin_rectangular_wave_generator.X の main 関数と同じです．

6.5.4　set_Op_Amp2 関数

```
set_op_amp.c

// オペアンプ2の設定
void set_Op_Amp2(unsigned char a, unsigned char b, unsigned char c, unsigned char d)
{
  OPA2CONbits.OPA2EN = a;    // オペアンプ2をオン／オフ設定
  OPA2CONbits.OPA2SP = b;    // オペアンプ2を高ゲイン×バンド幅モードで使用／不使用
  OPA2CONbits.OPA2UG = c;    // オペアンプ2をボルテージフォロワで使用／入力をOPA2IN-につないで使用
  OPA2CONbits.OPA2PCH = d;   // オペアンプ2の＋入力をFVR/DAコンバータ出力/OPA2IN+に接続
}
```

図 6.53: set_Op_Amp2 関数

図 6.53 は set_Op_Amp2 関数です．set_op_amp.c ファイル内にあります．

図 6.54 は OPA2CON レジスタです．pic_16f1705_s.h のヘッダファイル内で定義されています．

```
// OPA2CON (Op Amp Control Register)

    // OPA2EN (Op Amp2 Enable bit)
    #define Op_Amp2_on   0b1                    // Op Amp2をオンとする.
    #define Op_Amp2_off  0b0                    // Op Amp2をオフとする.

    // OPA2SP (Op Amp2 Speed/Power Select bit)
    #define Op_Amp2_High_GBWP    0b1            // Op Amp2を高ゲイン×バンド幅モードに設定
    #define Op_Amp2_not_H_GBWP   0b0            // データシートには0は使用しないとある.

    // OPA2UG (Op Amp2 Unitiy Gain Select bit)
    #define Op_Amp2_Unity_Gain          0b1     // Op Amp2をボルテージフォロワに設定
    #define Op_Amp2_Neg_Input_to_OPA2IN  0b0    // Op Amp2の-入力をOPA2IN-に接続

    // OPA2PCH (Op Amp2 Non-Inverting Channenl Selecton bits)
    #define Op_Amp2_Pos_Input_to_FVR        0b11    // ＋入力をFVRに接続
    #define Op_Amp2_Pos_Input_to_DAC_output  0b10   //          D/Aコンバータ出力に接続
    #define Op_Amp2_Pos_Input_to_OPA2IN_P    0b00   //          OPA2IN+に接続

// 関数の宣言
void set_Op_Amp2 (unsigned char a, unsigned char b, unsigned char c, unsigned char d);
```

図 6.54: OPA2CON レジスタ

6.5.5　フィルタプログラムのブロック図と実験結果

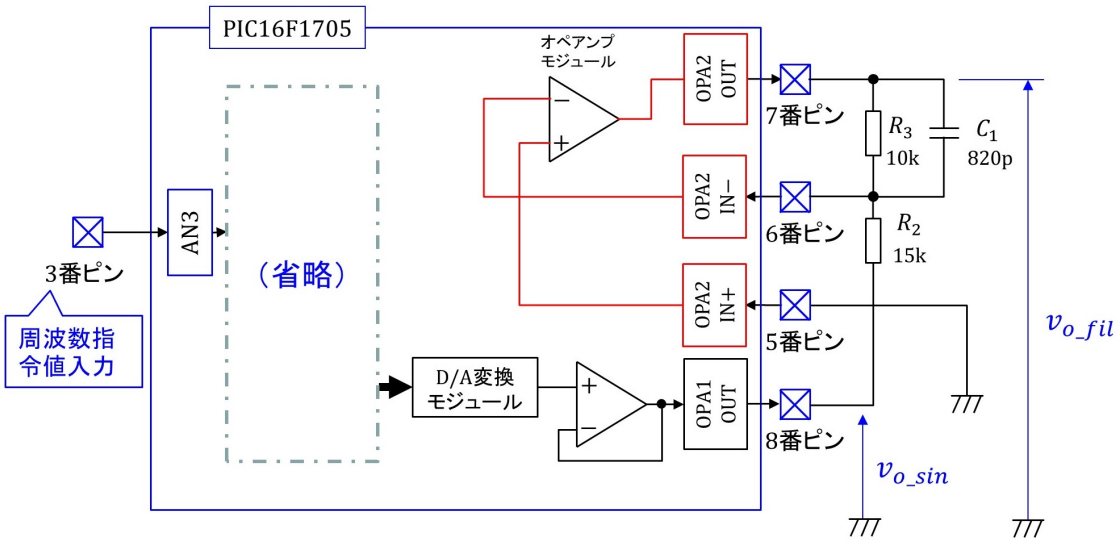


図 6.55: オペアンプによるフィルタプログラムのブロック図

図 6.55 がオペアンプによるフィルタプログラムのブロック図です．省略箇所は図 6.46 と同じです．

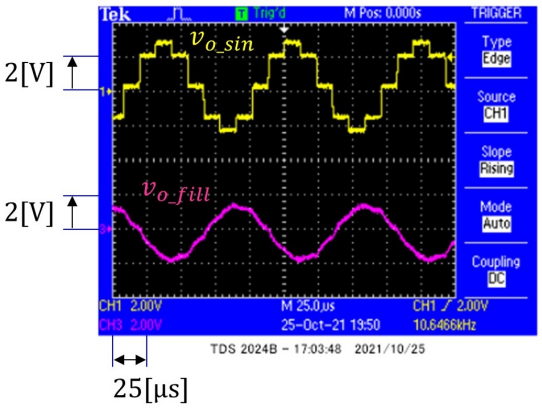


図 6.56: オペアンプによるローパスフィルタプログラムの実験結果

図 6.56 はオペアンプによるローパスフィルタプログラムの実験結果です．上の波形がフィルタ入力電圧 v_{o_sin} です．図 5.4(c) と同じ波形です．下の波形がフィルタ出力電圧 v_{o_fil} です．ローパスフィルタにより階段状の波形が正弦波に近く成形されました．

索 引

関連図書

[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」アマゾン Kindle 版，2019.

[3] 古橋武「パワーエレクトロニクスノート　ー PIC16F1825 による正弦波発生器，PWM 信号発生器ー」無料配布ファイル，2019.

[4] 古橋武「PIC16F1825 による DC モータの回転数制御」無料配布ファイル, 2019.

著者
古橋　武
名古屋大学名誉教授