PIC16-Trainer

ユーザーズマニュアル

Ver 1.01

株式会社ソリトンウェーブ

■履歴

平成 21 年 2 月 26 日	初版発行
平成 21 年 3 月 09 日	Ver 1.01 回路図を製品版に合わせてアップデート

目次

はじめに

PIC16 トレーナは、マイクロチップ社の PIC16F690 を使用した、学習キットとなっています。
本キットには、C コンパイラだけではなく、PIC デバイスの学習や開発に必要な物が全てそろっているため、購入して、直ぐに学習が開始できます。また、さまざまな周辺デバイスを搭載しているため、半田付けなどの道具や手間が必要ありません。半田付けが苦手な人や面倒な人でも、問題なくご利用いただけます。
学習用の基板には、あらかじめ、LED、スイッチ、ボリューム、7 セグメント LED など、必要最低限の回路が組み込まれています。さらに、拡張キットと組み合わせれば、キャラクタタイプの LCD の制御や、シリアル通信、ブレッドボードを使った実験などを行うことが可能になります。

※本学習キットは、弊社旧製品の、PIC16F690 を使用した「PIC トレーニングキット」の上位バージョンとなっています。旧製品からの変更点は、次の通りです。

- PIC16F690 から、より高機能な PIC16F690 に変更
- ほとんどの回路が実装済みのため、配線をせず、直ぐに学習が可能
- USB 接続タイプの PIC ライタ「PICKit2」付属（PICKit2 セットの場合）のため、シリアルポートのない PC や、ノート PC でも使用可能
- 日本語 C コンパイラ「ソースブースト」と、PIC16F690 のフルライセンス付き
- 組み込み機器でよく使われる、7 セグメント LED のダイナミック点灯の実験も可能

セット内容

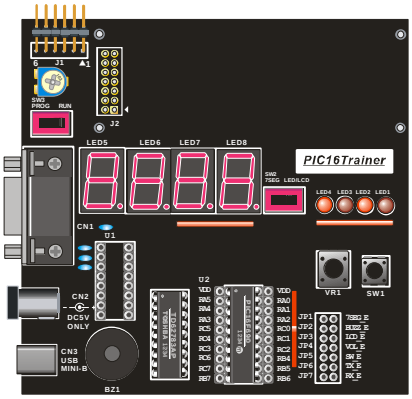
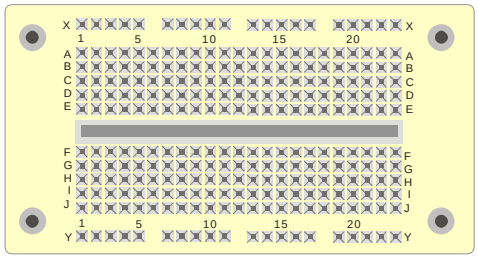
本セットには、以下の物が含まれています。添付品をご確認下さい。

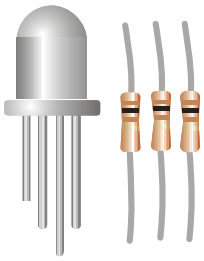
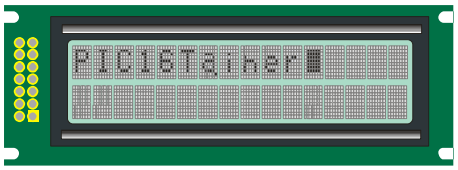
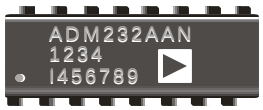
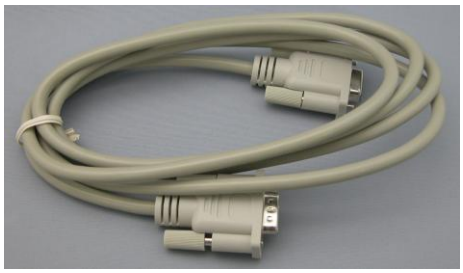
1. PIC16 トレーナー基板（PIC16F690 付き）
2. ソースブースト 6.0（日本語 PIC-C コンパイラ）（PIC16F690 フルライセンス付き）
3. サンプルソース（ソースブースト CD 内に格納）

また、拡張キットには、次のものが含まれています。

1. D-SUB9 通信ケーブル（オスーメス）
2. LCD モジュールと取り付けネジ
3. RS232 ドライバ IC
4. ブレッドボードセット（配線ケーブル付き）
5. 実験デバイス（フルカラーLED、抵抗）
6. ソースブースト 6.0 製本マニュアル

付属品とオプション

画像	内容
	ソースブースト 6.0・PIC16F690 フルライセンス付き PIC マイクロコントローラ用の日本語 C コンパイラです。
	学習キット本体です。PIC16F690 が実装されています。 基板には、スイッチ、LED、ボリューム、7 セグメント LED など、実験に必要な回路が、あらかじめ含まれています。
	PIC 書き込み器です。PICKit2 セットに付属しています。 本製品は、電源も PICKit2 から供給します。他の PIC プログラマを使用する場合は、外部電源端子を使用してください。(外部電源端子は、DC プラグもしくは MINI USB コネクタが利用可能です。どちらのコネクタも、実装されていませんので、使用する場合は、別途コネクタを用意して、実装してください。)
	ブレッドボードです。拡張キットに付属しています。 実験回路を組み立てる場合に使用します。
	ジャンパケーブルです。拡張キットに付属しています。 ブレッドボードで回路を組み立てる場合に、配線用に使用します。

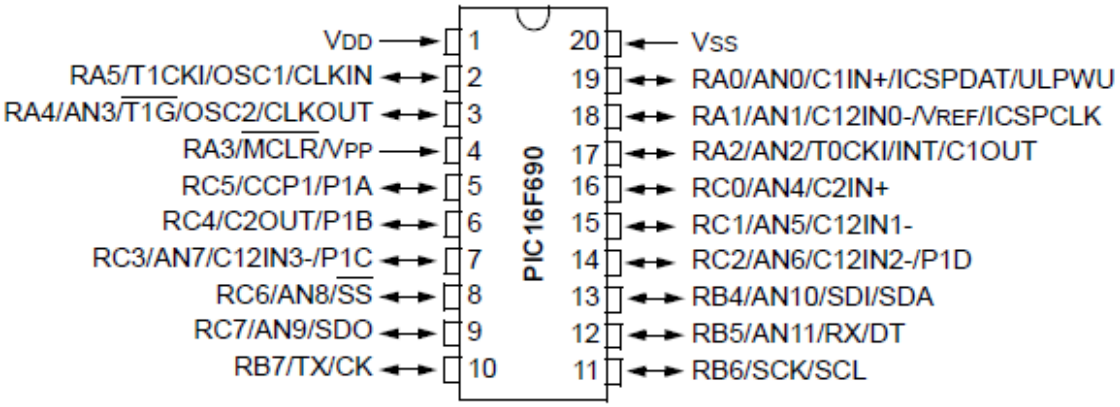
	フルカラーLED と 100 オーム抵抗です。拡張キットに付属しています。フルカラーLED の実験の際、ブレッドボードとともに使用します。
	LCD モジュールです。拡張キットに付属しています。 LCD は、付属のスペーサとネジで、PIC16 トレーナ基板に取り付けます。
	RS232 ドライバ IC です。拡張キットに付属します。 PIC16 トレーナ基板の、U2 に取り付けます。取り付ける際は、IC の向きに注意してください。
	シリアルケーブルです。拡張キットに付属しています。 シリアル通信のテストで使用します。 ケーブルは、D-SUB9 ピンの、オスーメスタイプのストレートケーブルです。

PIC16F690 の概要

ここでは、PIC16F690 の概要を説明します。詳しい説明は、PIC16F690 のデータシートや、一般の解説書をご参照下さい。
PIC16F690 の最新版のデータシート（英語）は、以下の URL からダウンロード可能です。
http://ww1.microchip.com/downloads/en/DeviceDoc/41262E.pdf

PIC16F160 は、プログラム用のフラッシュメモリを内蔵した、マイクロコントローラ（CPU）です。プログラムメモリや、RAM、各種周辺回路を内蔵し、クロックオシレータも内蔵しているため、電源を入れるだけで動作させることができます。
次の図は、PIC16F690 のピン配置図です。

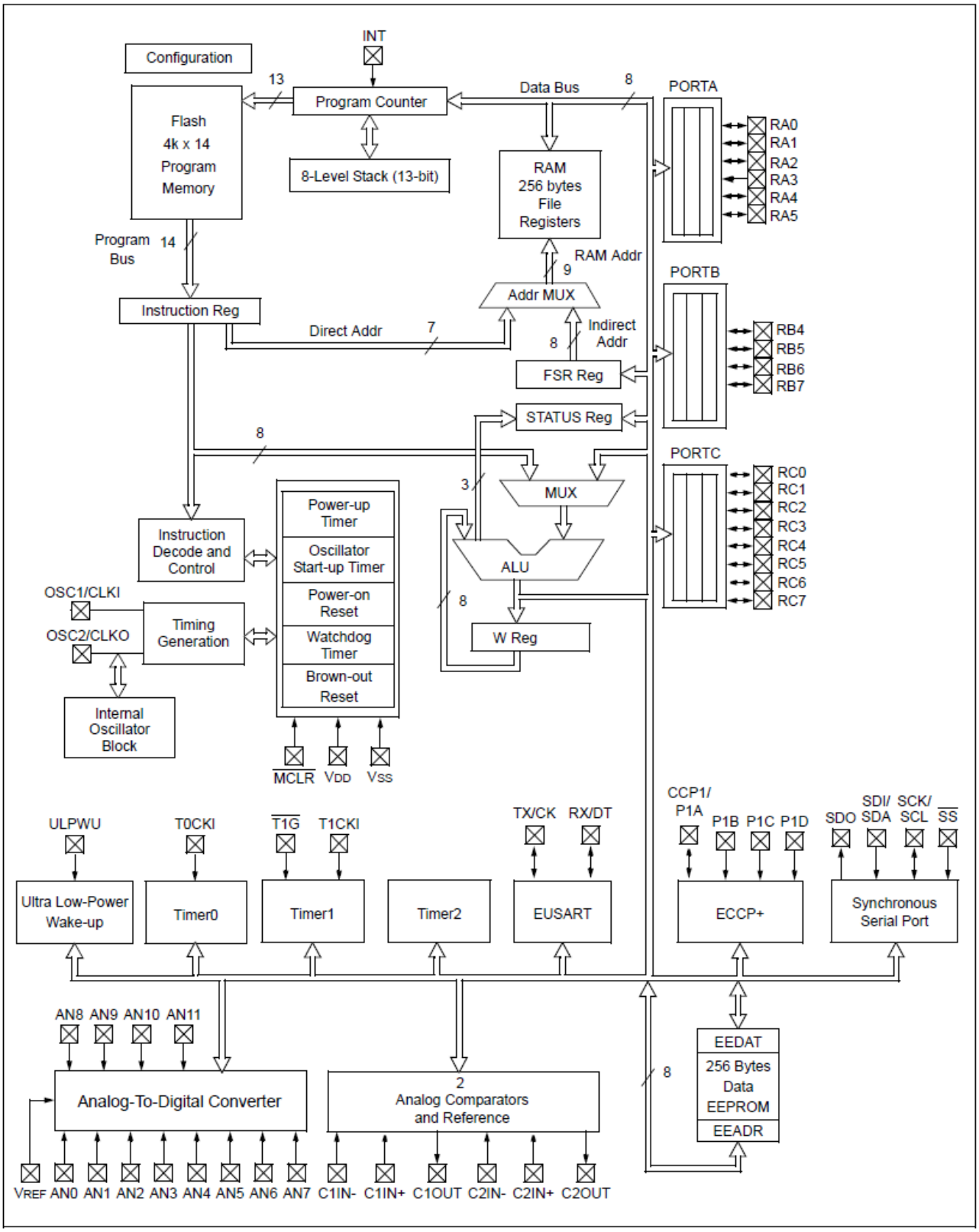
20-pin PDIP, SOIC, SSOP



PIC16F690 には、RA0-5、RB4-7、RC0-7 の 18 本の GPIO があります。GPIO は、ソフトウェアの制御により、入力ピンや出力ピンに割り当てることができるため、ソフトウェアを用意することにより、カスタムの IC のような動作をさせることができます。例えば、7 セグメントの LED デコーダのように、4 ビットの BCD 入力に対して、7 セグメントの LED を点灯させるようなデバイスが必要な場合、PIC16F690 を使えば、簡単なプログラムで実現することができます。このようなデバイスは、PAL や GAL といた、書き込み可能なハードウェアデバイスでも実現可能ですが、PIC16F690 を使う場合は、使い慣れた C 言語で記述できますし、ソフトウェア制御ですので、さらに複雑な動作をさせたい場合も、比較的簡単に行う事ができます。また、PIC16F690 には、タイマ、UART、D-A コンバータ、コンパレータ、SSP といった、内蔵の周辺デバイスがあります。これらの周辺デバイスは、GPIO のピンと同じピンに割り当てられています。1 つのピンには、複数の機能がありますが、どの機能を使うかは、ソフトウェアで切り替えられるようになっています。
各ピンの割り当ては、次の表のようになっています。

I/O	Pin	Analog	Comparators	Timers	ECCP	EUSART	SSP	Interrupt	Pull-up	Basic
RA0	19	AN0/ULPWU	C1IN+	—	—	—	—	IOC	Y	ICSPDAT
RA1	18	AN1/VREF	C12IN0-	—	—	—	—	IOC	Y	ICSPCLK
RA2	17	AN2	C1OUT	T0CKI	—	—	—	IOC/INT	Y	
RA3	4	—	—	—	—	—	—	IOC	Y(1)	$\overline{\text{MCLR}}$/VPP
RA4	3	AN3	—	$\overline{\text{T1G}}$	—	—	—	IOC	Y	OSC2/CLKOUT
RA5	2	—	—	T1CKI	—	—	—	IOC	Y	OSC1/CLKIN
RB4	13	AN10	—	—	—	—	SDI/SDA	IOC	Y	—
RB5	12	AN11	—	—	—	RX/DT	—	IOC	Y	—
RB6	11	—	—	—	—		SCL/SCK	IOC	Y	—
RB7	10	—	—	—	—	TX/CK	—	IOC	Y	—
RC0	16	AN4	C2IN+	—	—	—	—	—	—	—
RC1	15	AN5	C12IN1-	—	—	—	—	—	—	—
RC2	14	AN6	C12IN2-	—	P1D	—	—	—	—	—
RC3	7	AN7	C12IN3-	—	P1C	—	—	—	—	—
RC4	6	—	C2OUT	—	P1B	—	—	—	—	—
RC5	5	—	—	—	CCP1/P1A	—	—	—	—	—
RC6	8	AN8	—	—	—	—	$\overline{\text{SS}}$	—	—	—
RC7	9	AN9	—	—	—	—	SDO	—	—	—
—	1	—	—	—	—	—	—	—	—	VDD
—	20	—	—	—	—	—	—	—	—	VSS

また、次の図は、PIC16F690 のブロック図です。



ここでは、主なブロックについて説明します。

1）フラッシュプログラムメモリ

プログラム格納用のメモリです。実行するプログラムを書き込んで使用します。

2）RAM

RAM は、図のようにファイルレジスタの一部となっています。周辺デバイスは、00H-0BH と 80-8BH

のアドレスの範囲にあります。

3）GPIO

GPIO には、PORTA と PORTB、及び PORTC の３つのポートがあります。PORTA は、RA0-RA5 までの 6 ビット、PORTB は、RB4-RB7 の 4 ビット、PORTC は、RC0-7 の 8 ビットが使用可能です。

4）タイマ

Timer0～Timer2 の、３つのタイマが使用可能です。入力クロックには、プリスケーラを使用可能です。また、オーバーフローの割り込みが使用できますので、周期割り込みをかけることができます。

5）EEPROM

256 バイトの EEPROM を使用可能です。EEPROM は、不揮発性ですので、設定データなどを保存する場合に便利です。EEPROM はプログラム中から、自由に読み書きすることができます。

6）スタック

8 レベルのハードウェアスタックが使用できます。ハードウェアスタックですので、サイズを変更することはできません。したがって、サブルーチンをネストさせる場合は、スタックサイズを超えないように、注意する必要があります。

7）ウォッチドッグタイマ

ウォッチドッグタイマは、プログラムが暴走した場合、CPU をリセットするために使用します。これは、CPU を組み込み機器に使用した場合、万が一プログラムが暴走しても、CPU が自動復帰させ、ハングアップを防ぐための機能です。PIC のプログラム学習の場合は、この機能は、OFF にしたままでよいでしょう。

8）EUSART

RS232 通信に使用する、シリアル通信モジュールです。

9）A-D コンバータ

12 チャンネル、10 ビットの A-D コンバータです。アナログデータを入力するのに使用します。

１０）　コンパレータ

プログラマブルの内部電圧と比較できる、２つのアナログコンパレータを内蔵しています。

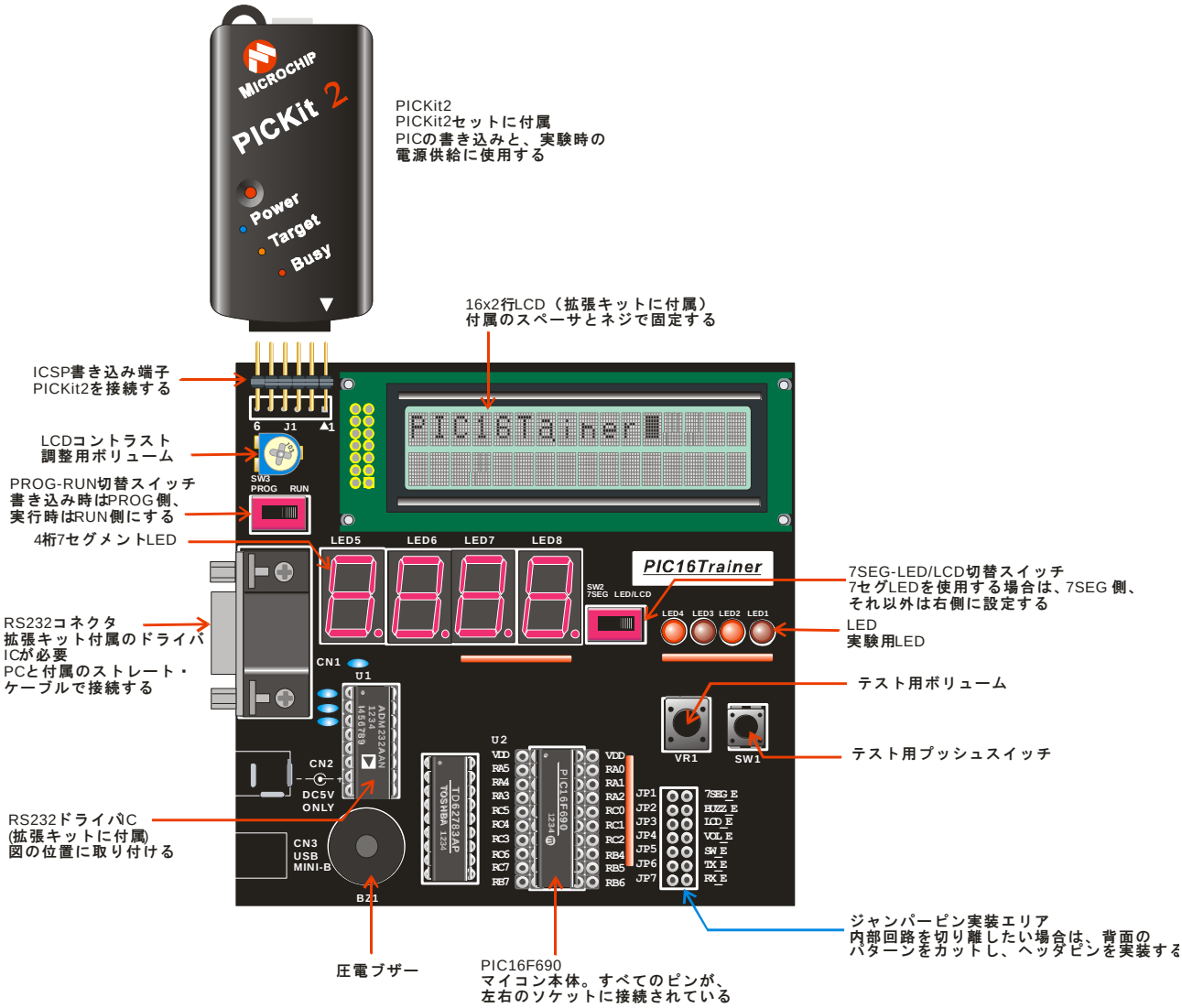
１１）　SSP

同期式シリアル通信用のモジュールです。I2C や SPI の通信に使用します。

PIC16F690 と、内蔵周辺デバイスの具体的な使い方は、後述するサンプルを参照してください。

PIC16 トレーナー基板について

次の図は、PIC16 トレーナー基板の概観図です。



この基板には、PIC16F690 のほか、スイッチ、ボリューム、LED、ブザー、7 セグメント LED、PIC 書き込み用 ICSP コネクタなどが搭載されています。また、拡張キットには、16 桁×2 行の LCD や RS232 用のドライバ IC、ブレッドボードなどが付属しています。ブレッドボードを使用すれば、独自の回路を搭載して、様々な実験を行う事ができるようになります。

各部の機能は、次の通りです。

- **ICSP コネクタ**

  PICKit2 を接続して、PIC デバイスの書き込みや、実験の際の電源供給を行います。。

- **RS232 コネクタ**

  拡張キット付属のドライバ IC を装着し、RS232 のテストを行う場合に使用します。

- **RS232 ドライバ IC**

  RS232 の信号レベルを、PIC で使用する信号レベルに変換します。拡張キットに付属しています。

- **LED**
  4 ビットの LED で、LED のテストを行う場合に使用します。
- **テスト用プッシュスイッチ**
  スイッチのテストを行う場合に使用します。
- **テスト用ボリューム**
  A-D コンバータのテストで使用します。
- **圧電ブザー**
  ブザーのテストで使用します。
- **16 桁×2 行 LCD**
  拡張キットに附属します。付属のスペーサーで取り付けます。LCD のテストに使用します。
- **LCD コントラスト調整用ボリューム**
  LCD のコントラストを調整します。
- **PROG-RUN 切替スイッチ**
  書き込み時／実行時に合わせて、スイッチを切り替えます。
- **7SEG-LED/LCD 切替スイッチ**
  7 セグ LED を使用する場合は、7SEG 側、それ以外は、LED/LCD 側に設定します。
- **ジャンパーピン実装エリア**
  PIC16 トレーナーでは、さまざまな周辺デバイスが、効率よく接続されていますが、ブレッドボードを使用して、独自回路の実験をする際、周辺デバイスを切り離したい場合があります。この場合、ジャンパーピン実装エリア背面のパターンをカットして、デバイスを切り離すことができます。この部分は、ジャンパーピンを接続すれば、再度周辺デバイスを有効にすることができます。

拡張キットの使い方

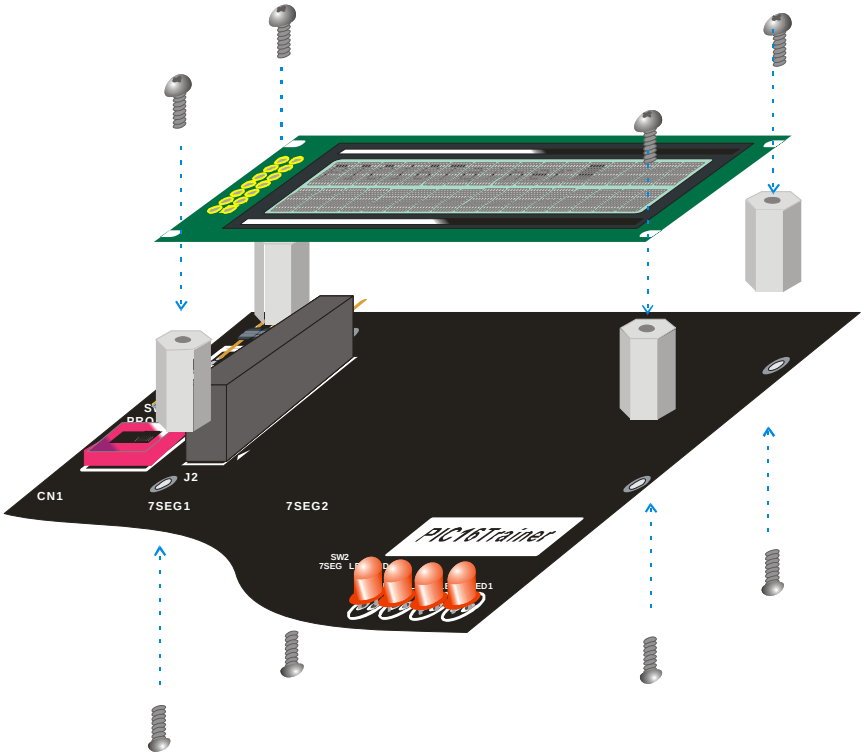
PIC16 トレーナーの拡張キットは、次のものが付属しています。

- 16 桁×2 行 LCD
- LCD 取り付け用スペーサとネジ
- RS232 ドライバ IC
- RS232 ストレートケーブル
- ブレッドボード
- ブレッドボード配線用ワイヤー
- フルカラーLED と 100 オームの抵抗×3
- ソースブースト 製本マニュアル

拡張キットを使用する場合は、LCD と RS232 ドライバ IC を、PIC16 トレーナー基板に取り付ける必要があります。基板の取り付け方法は、以下の説明を参照してください。

LCD の取り付け

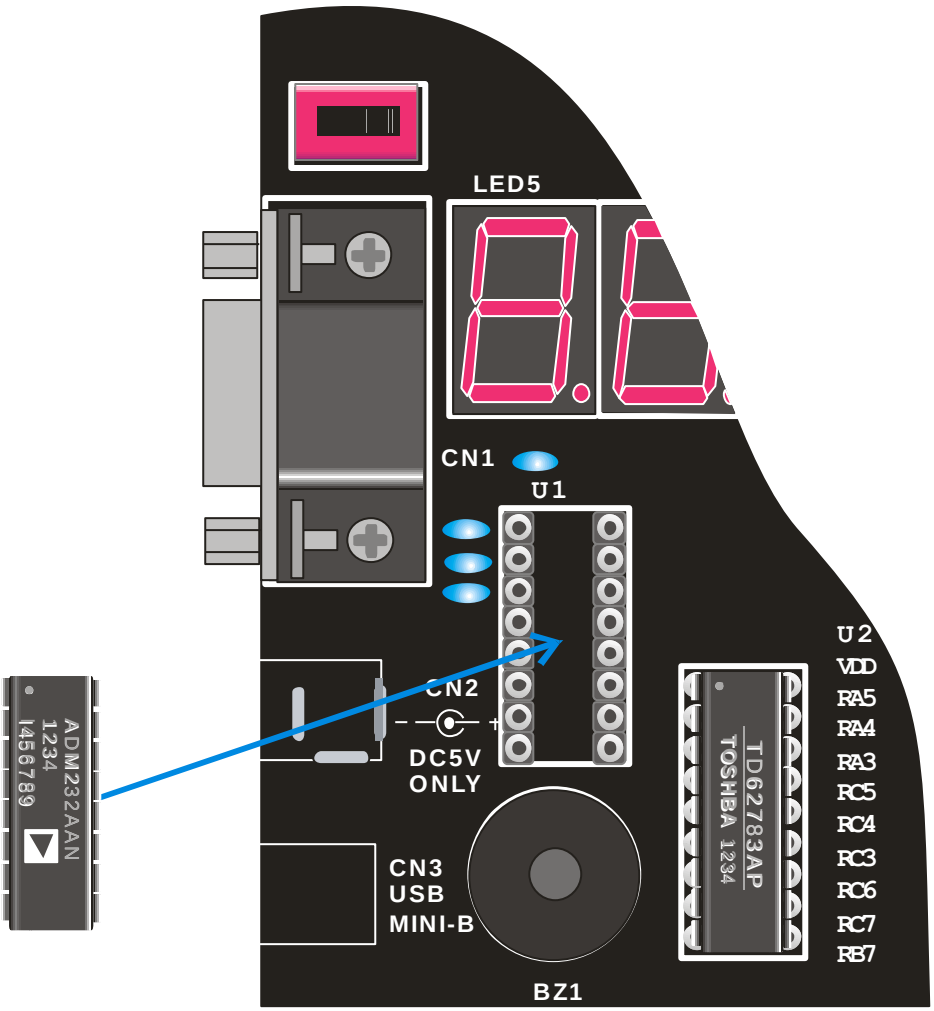
拡張キットを購入した場合は、LCD を PIC16 トレーナーに取り付けます。
LCD の取り付けは、図のように、付属のねじとスペーサを使って、トレーナー基板に取り付けてください。取り付けの際には、コネクタのすべてのピンが、正しく挿入されているかを確認してください。



※先に基板にスペーサーを取り付けてから、LCD を取り付けてください。

RS232 ドライバ IC の取り付け

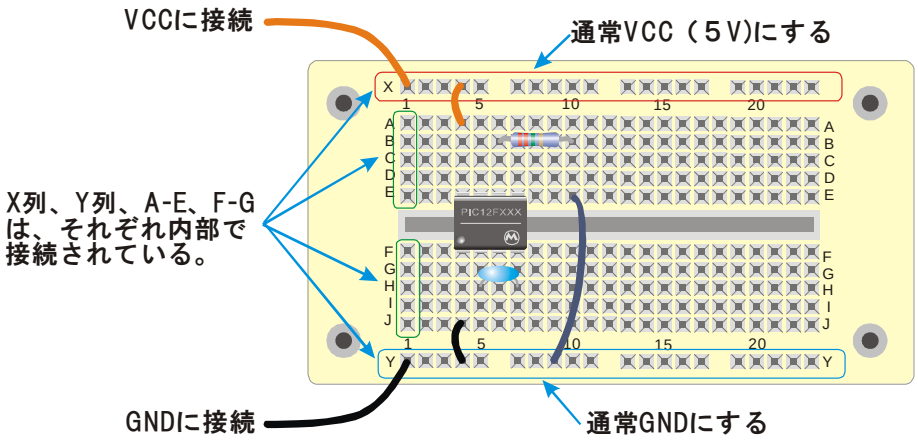
拡張キットには、RS232 のドライバ IC、ADM232AAN（または相当品）が付属しています。ドライバ IC は、図のように、PIC トレーナー基板の U1 の IC ソケットに挿入します。ドライバ IC は、保護用の黒いウレタンに刺さっていますので、出来るだけ IC ピンには直接触れないようにしてください。
ドライバ IC の挿入位置は、図のように、IC の 1 番ピン（丸印の付いたピン）が、U1 の IC ソケットの 1 番ピン（左上のピン）に来るように装着します。



IC のピンは、ソケットよりも若干広く広がっていますので、ピンを軽く内側に曲げると、うまく挿入することができます。挿入後は、IC のピンがすべて正しく IC ソケットに刺さっていることを確認してください。

ブレッドボードの使い方

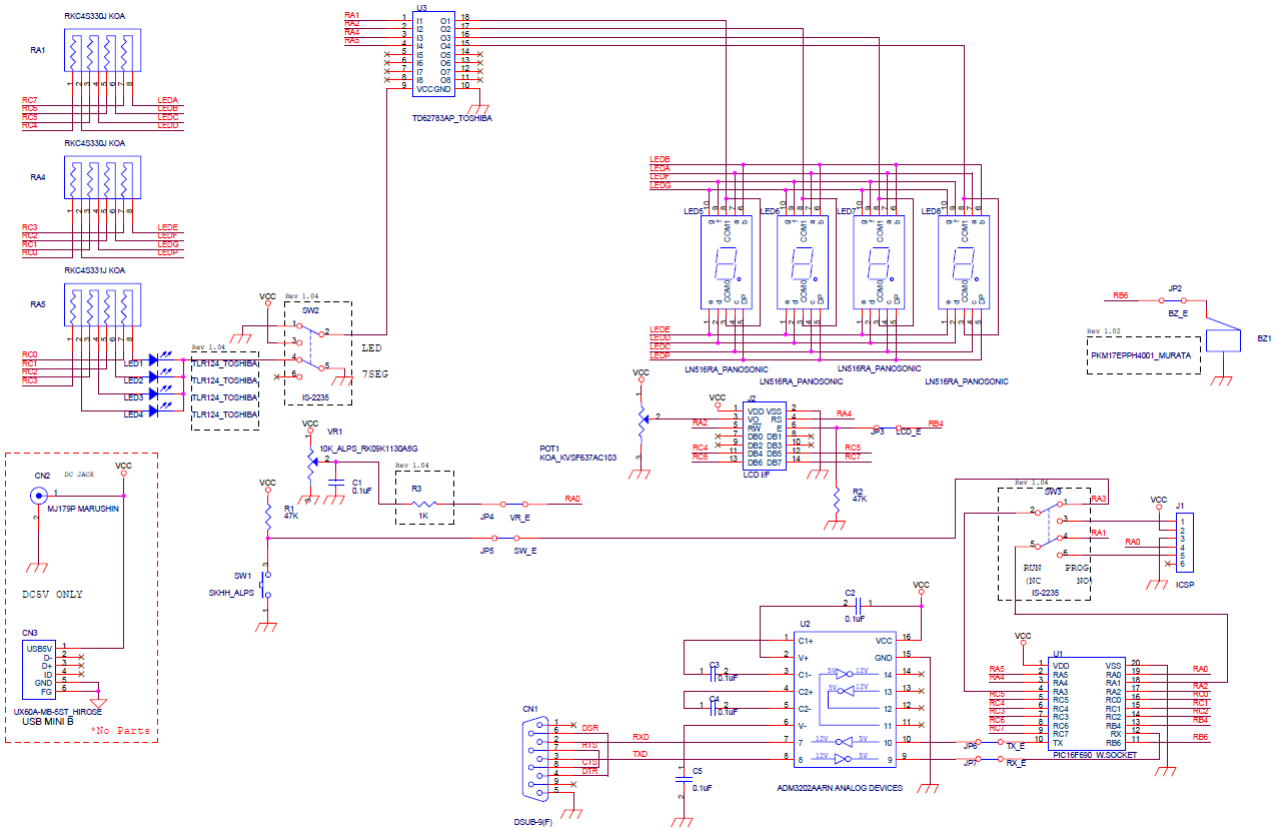
ここで、簡単にブレッドボードの使い方について説明します。
ブレッドボードは、図のように、格子状にソケットピンが配置されているボードです。



図の X 列、Y 列のピンは、それぞれ、内部で接続されています。例えば、X 列の 1 番ピンに VCC を接続すれば、X 列は全て VCC となります。また、A-E、F-G も、それぞれ内部で接続されています。
ブレッドボードのピンには、IC や抵抗、コンデンサ、ジャンパケーブルなどが接続できますので、任意の回路をボード上につくることができます。この図の例では、IC の 1 番ピンが GND、8 番が VCC、2 ピンとピンの間にコンデンサ、5 ピンは抵抗を通して GND に接続されています。VCC は、PIC1690 の 1 ピン、GND は、PIC16F690 の 20 ピンから取ることができます。

回路図

次の図は、PIC16 トレーナー基板の回路図です。



回路の各ブロックの詳細は、次の通りです。

1）電源回路

電源は、通常 ICSP に接続した PICKit2 から電源をもらって動作するようになっています。

基板上には、5V の DC ジャックと、USB のミニコネクタを実装できるようになっていますので、外部電源を別途用意したい場合は、何れかのコネクタを実装してください。

2）RS232 回路

RS232 回路は、ADM3202AA/ADM232AA/MAX232 などの RS232 レベルコンバータを使って動作しています。拡張キットには、上記の何れかのデバイス、もしくは相当品が付属していますので、IC を基板に実装してから使用してください。

3）16 桁×2 行 LCD

拡張キットには、16 桁×2 行のキャラクタ LCD が付属しています。LCD はトレーナー基板に、スペーサーとネジで固定して使用します。

4）その他の PIC 周辺回路

その他の PIC 周辺回路には、プッシュスイッチ、LED、ブザー、7 セグメント LED が接続されています。7 セグメント LED を使用する場合は、7SEG-LED/LCD 切り替えスイッチを、7SEG 側に

切り替えます。それ以外の場合は、LED/LCD 側に設定します。7 セグメント LED を使用する場合は、LCD や LED は使用できませんので、注意してください。

5）ICSP 端子

ICSP 端子は、PICKit2 を接続して、PIC デバイスに書き込みや、動作時の電源供給を行います。

PIC のピンアサイン

PIC16 トレーナーの、PIC16F690 のピンアサインは、次のようになっています。

ポート名	ピン番号	LED/LCD	7SEG	入出力	ICSP
RA0	19	ボリューム	ボリューム	入力	ICSPDAT
RA1	18	-	7セグ-1有効	出力	ICSPCLK
RA2	17	LCD-RW	7セグ-2有効	出力	
RA3	4	SW	SW	入力	VPP
RA4	3	LCD-RS	7セグ-3有効	出力	
RA5	2	-	7セグ-4有効	出力	
RB4	13	LCD-E	-	出力	
RB5	12	RS232-RX	RS232-RX	入力	
RB6	11	ブザー	ブザー	出力	
RB7	10	RS232-TX	RS232-TX	出力	
RC0	16	LED1	LEDP	出力	
RC1	15	LED2	LEDG	出力	
RC2	14	LED3	LEDF	出力	
RC3	7	LED4	LEDE	出力	
RC4	6	LCD-D4	LEDD	出力	
RC5	5	LCD-D5	LEDC	出力	
RC6	8	LCD-D6	LEDB	出力	
RC7	9	LCD-D7	LEDA	出力	
ー	1	VDD	VDD		VDD
ー	20	VSS	VSS		VSS

※LED/LCD の列は、スイッチを LED/LCD 側にした場合、7SEG の列は、7SEG 側にした場合の設定です。

開発環境の準備とサンプルプログラム

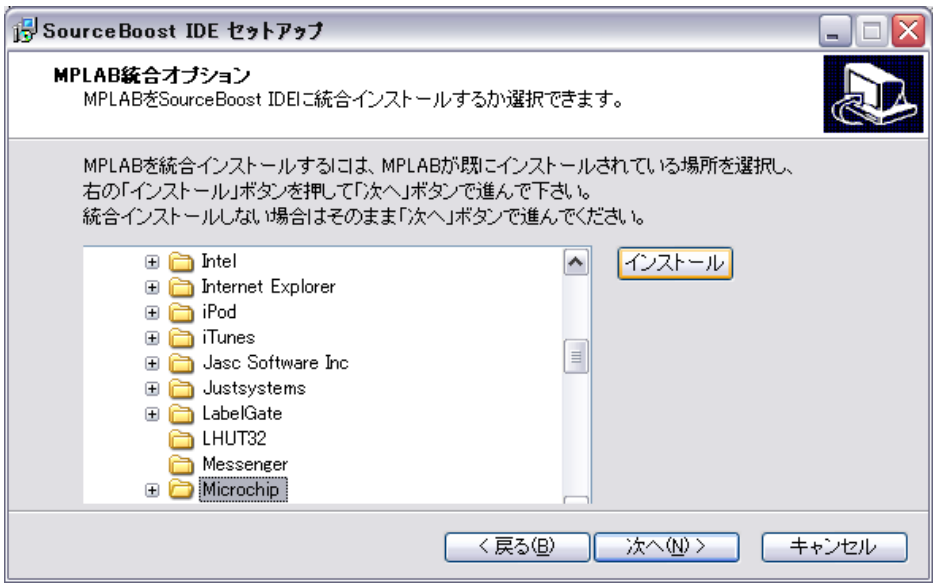
PIC の開発には、コンパイラとプログラマが必要になります。はじめに、これらの環境を整えることにします。コンパイラは、付属の日本語 C コンパイラ「ソースブースト 6.0」を使用しますが、開発／実行時には、MPLAB IDE を使用すると便利です。また、プログラマには、PICKit2 を使用します。PICKit2 以外のプログラマを利用する場合は、ご使用になるプログラマの説明書を参照してください。また、PIC16 トレーナー基板は、実行時も PICKit2 から、電源をもらって動作するように設計されています。PICKit2 以外のプログラマを使用する場合は、電源コネクタを実装して、外部から 5V を供給するようにしてください。
開発環境のセットアップは、以下の手順で行ってください。

MPLAB IDE のインストール

はじめに、MPLAB IDE をインストールします。PIC16 トレーナー付属の CD を起動すると、メニューが起動しますので、MPLAB IDE を選択して、インストールを行います。インストールは、インストールプログラムの指示に従って、デフォルト設定のままインストールしてください。途中、PICC LITE のインストールをするかどうかの確認がありますが、PIC16 トレーナーでは、ソースブーストを使用しますので、インストールしなくて構いません。MPLAB のインストールでは、同時に PICKit2 のドライバもインストールされますので、PICKit2 の設定は、特に必要ありません。

ソースブーストのインストール

次に、ソースブーストのインストールを行います。MPLAB と同様に、CD のメニューから、ソースブーストを選択します。ソースブーストのインストールも、デフォルト設定のまま行ってください。ただし、途中で画面のように、MPLAB との統合を行うかどうかの確認画面が出ますので、”インストール”ボタンを押して、MPLAB との統合を行ってください。



インストールボタンを押すと、ボタンがグレー表示になり、統合が完了しますので、”次へ”ボタンを押して、インストールを完了してください。

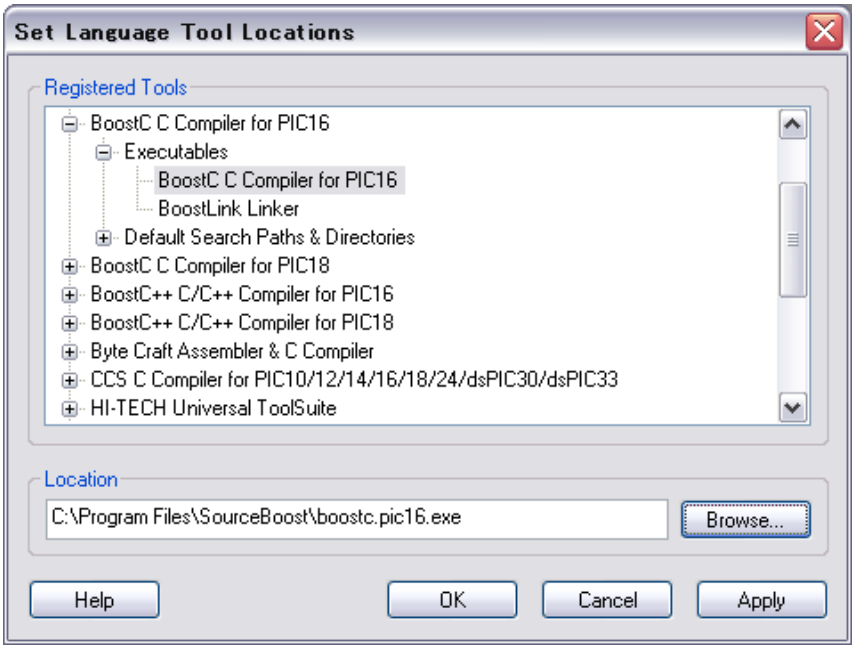
サンプルプログラムのインストール

最後にサンプルプログラムをインストールします。同じように、CD メニューから、PIC16 トレーナーサンプルを選択して、インストールを行います。インストールフォルダは、C:¥Pic16Samp となります。

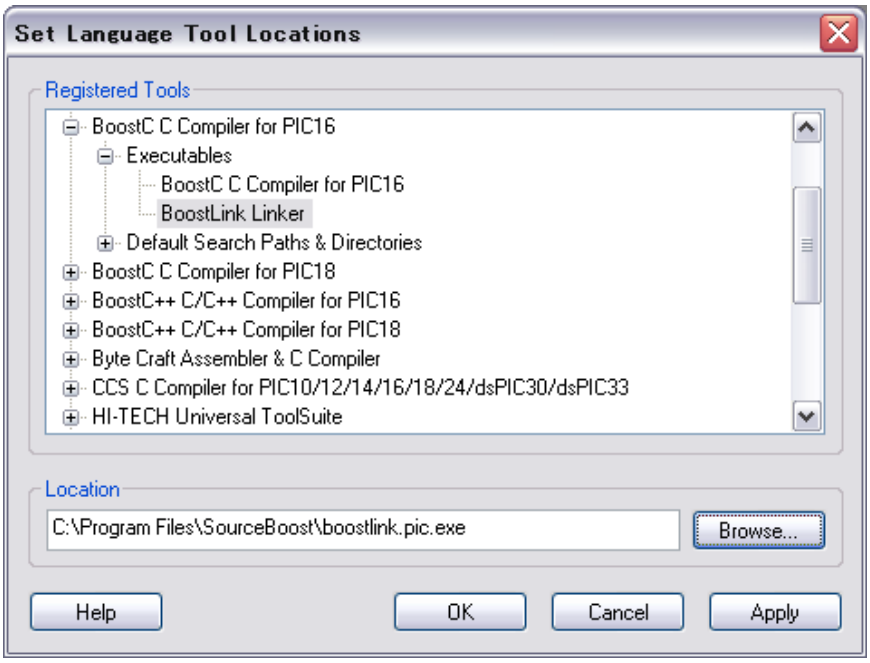
MPLAB の環境設定

開発環境のインストールが終わったら、MPLAB の初期設定を行います。MPLAB は、標準で、マイクロチップのアセンブラを搭載していますが、今回は、ソースブーストをコンパイラとして使用しますので、あらかじめ設定を行う必要があります。MPLAB の環境設定は、MPLAB を起動して、次の手順で行います。

1. 1.”Project”メニューから、”Set Language Tool Locations…”を選択して、”Set Language Tool Locations”のダイアログを開く”BoostC C Compiler for PIC16”を展開して、さらに”Executables”を展開する
2. ”Executabules”の項目の、”BoostC C Compiler for PIC16”の Location を、C:¥Program Files¥SourceBoost¥boostc.pic16.exe に設定し、Apply ボタンで確定する。



3. 同様にリンカの”BoostLink Linker”のパスを、
C:¥Program Files¥SourceBoost¥boostlink.pic.exe に設定し、Apply ボタンで確定する。



(

以上の設定で、MPLAB のソースブースト用の設定は完了です。

MPLAB IDE とソースブースト IDE

プログラムの開発は、MPLAB の IDE でも、ソースブーストの IDE でも行うことができます。どちらの環境も、ソースをカラー表示する高度なエディタがあり、コンパイルからデバッグまで、統合された環境で行うことができます。ただし、どちらの環境も、次のように、一長一短がありますので、用途によって使い分けてください。

1．　ソースブースト IDE

ソースブーストの IDE では、仮想デバイスを使ったシミュレーション・デバッグが可能です。仮想デバイスを使ったシュミレーションでは、スイッチや LED など、よく使うデバイスを Windows の画面上に配置して、作成したプログラムを、仮想的に動作させることができます。仮想デバイスは、別売のプラグインで増やすことができます。追加のプラグインでは、7 セグメント LED や、LCD モジュールなどを使用することができます。仮想デバイスを使ったシュミレーションでは、ハードウェアがなくても、プログラムのデバッグが可能ですので、ハードウェアと並行して、プログラム開発を行わなければならない場合などに非常に便利です。また、シミュレータ上でデバッグできるので、任意の場所でプログラムを停止させ、その時のレジスタの状態を確認したり、値を変更して、プログラムの動きを確認するといった使い方ができます。

2．　MPLAB IDE

MPLAB は、PIC の開発元のマイクロチップ社の製品のため、マイクロチップ社製のプログラマをサポートしています。サポートプログラマの中には、PICKit2 も含まれているため、PICKit2 を使って、開発／デバッグを行う場合、IDE 上ですべての作業が可能になります。また、ソフトウェア・シミュレータとして、MPSIM が付属しています。MPSIM は、ソースブーストのシミュレータのように、仮想デバイスを使ったシミュレーションはできませんが、純正品のため、サポートデバイスが豊富で、新しいデバイスにも、いち早く対応できるというメリットがあります。

このマニュアルでは、PIC16 トレーナーでの動作確認を主に行うため、MPLAB を使った環境で、開発を行うことにします。また、後の章で、ソースブーストの仮想デバイス・シミュレーションについても、説明いたします。どちらの環境も、ソースファイルがあれば、プロジェクトファイルを作るだけで、簡単に環境の移行が可能です。

以下の章から、実際にプログラムを作りながら、PIC のプログラム開発を学んできます。

スイッチと LED のテスト

■概要

スイッチ押すたびに、LED が ON./OFF するプログラムを作成する

■サンプルプロジェクトの格納フォルダ

Switch

（フォルダ名は、インストールフォルダの下のフォルダ名です。デフォルトのインストールでは、c:¥Pic16Samp¥Switch が、実際のフォルダです。）

■手順

PIC16 トレーナー基板には、テスト用のスイッチと LED が付属していますので、今回は、これをそのまま使用します。7SEG-LED/LCD のスイッチが、LED/LCD 側になっていることを確認してください。

作成するプログラムは、次のようなものです。

- スイッチを 1 回押すと、LED が点灯する
- 再度スイッチを押すと、LED が消灯する
- スイッチを押す度に、上記の動作を繰り返す
- スイッチのチャタリングを防止する

それでは早速、ソースブーストを使って、プログラムを作ってみましょう。

プロジェクトフォルダの準備

最初に、プロジェクトを作成するフォルダを用意します。サンプルをインストールすると、すでにプロジェクトファイルが格納されていますが、プロジェクトの作成方法を学ぶため、ここでは新しいフォルダを使って、プロジェクトを作成してみることにします。

なお、このマニュアルで取り上げるプログラムのプロジェクトは、サンプルのインストール時に指定したフォルダ（通常は、c:¥Pic16Samp）の下にインストールされています。作成したプロジェクトがうまく動作しない場合は、こちらのフォルダのソースを参照してください。

ここでは、プロジェクトの格納用に、次のフォルダを作成してください。

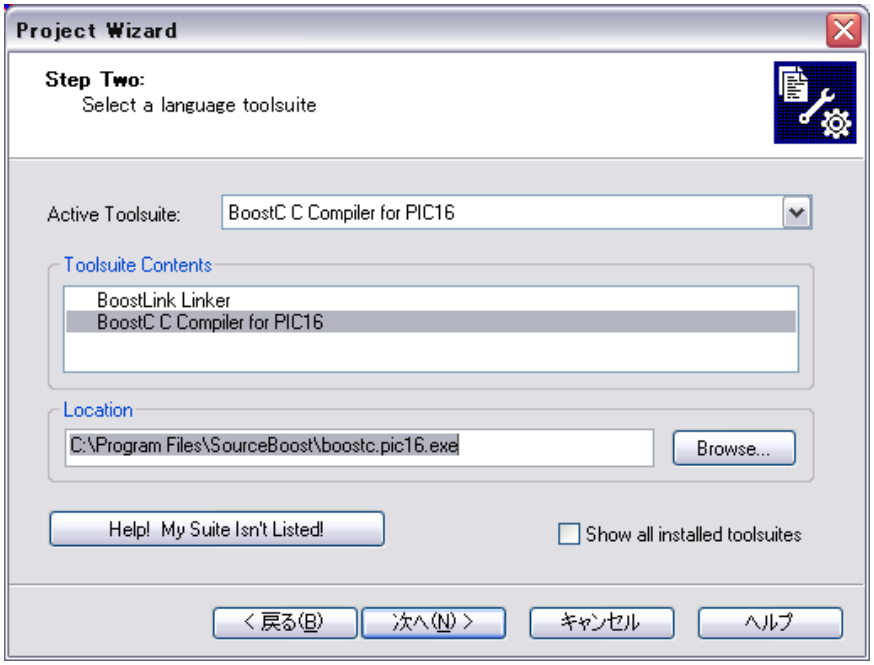
c:¥PicTest¥Switch

フレームワークの作成

MPLABのプロジェクトウィザードを使って、フレームワーク（雛形）を作成します。
まず MPLAB を起動して、”Project”メニューから、”Project Wizard”を起動します。MPLAB の画面は英語ですが、難しいことはありません。プロジェクトウィザードの画面は、ステップ 1 から順に、”次へ”ボタンを押して行き、必要な項目を設定するだけで、プロジェクトファイルが完成します。
プロジェクトウィザードの、各ステップでの設定内容は、次の通りです。

Step One:
使用するデバイスを選択します。PIC16 トレーナーでは、PIC16F690 を使用しますので、Device 欄で、PIC16F690 を選択します。

Step Two:
使用するコンパイラとリンカを選択します。Active Toolsuite:で、”BoostC C Compiler for PIC16”を選択すると、図のような画面になります。



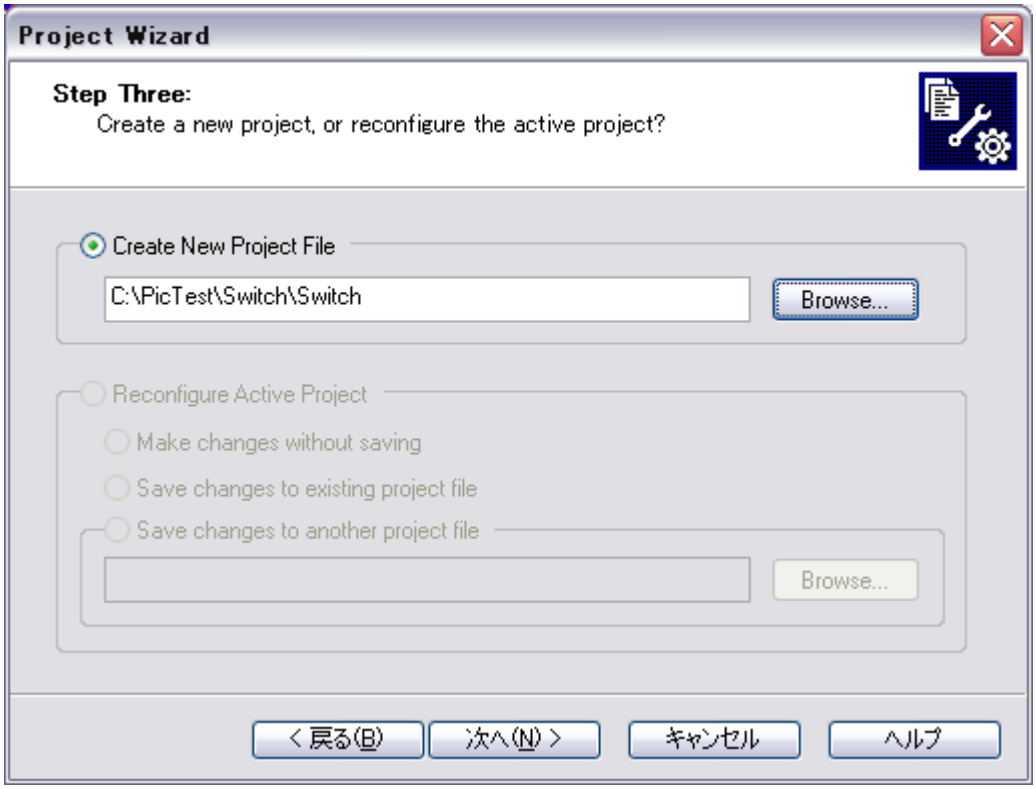
ここでは、MPLABの環境設定で設定した、コンパイラとリンカのパスが表示されています。環境設定が正しく行われていれば、上記のように表示されますが、Locationが正しく表示されていない場合は、環境設定で行った通り、コンパイラとリンカを、次のよう設定してください。

1) BoostC C Compiler for PIC16
   c:¥Program Files¥SourceBoost¥boostc.pic16.exe
2) BoostLink Linker
   C:¥Program Files¥SourceBoost¥boostlink.pic.exe

Step Three:

プロジェクトファイルの作成フォルダを指定して、プロジェクトファイルを作成します。

Browseボタンを押して、先ほど作成したc:¥PicTest¥Switchフォルダを指定し、プロジェクトファイル名として、Switchを入力して、保存ボタンを押します。

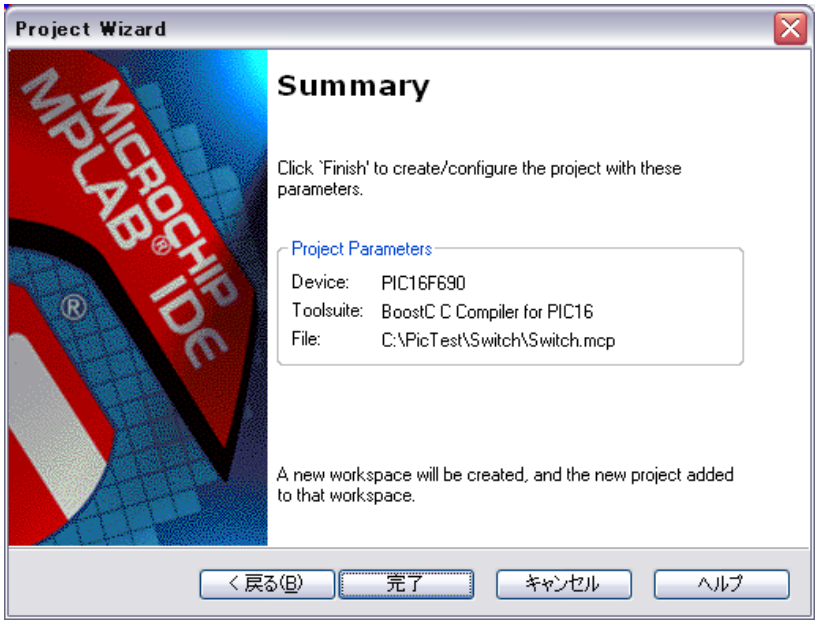


Step Four:

すでに作成されたソースファイルをプロジェクトに追加する場合は、ここでファイルを選択して、追加することができます。今回は、新規に作成するため、そのまま“次へ”ボタンを押します。

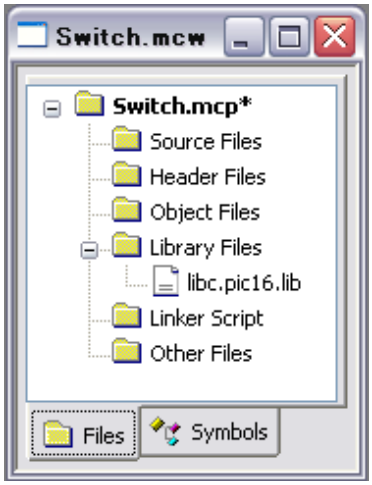
Step Five:

最後にサマリが表示されるので、内容を確認して、完了ボタンを押します。



ライブラリの追加

これでプロジェクトファイルが作成されましたので、プロジェクトファイルの一覧を見ることにします。プロジェクトファイルの一覧は、プロジェクトウィンドウで、ファイルツリーの形で見ることができます。画面にプロジェクトウィンドウが表示されていなければ、Viewメニューから、Projectメニューにチェックを入れると、プロジェクトウィンドウを表示することができます。
ソースブーストでは、ライブラリ関数がありますので、これを使えるようにしておくと便利です。ソースブーストのIDEを使用する場合は、ライブラリは自動でリンクされるので、ライブラリの追加操作は必要ありませんが、MPLAB上でソースブーストを使用する場合は、プロジェクトのファイルとして、ライブラリを追加しておく必要があります。ライブラリの追加は、ライブラリ関数を使わなければ必要ありませんが、いろいろ便利な関数がありますので、忘れないように、最初にライブラリの追加を行っておきます。
ライブラリの追加をするには、プロジェクトウィンドウの、Library Filesのアイコンで、マウスを右クリックして、Add Filesを選択します。ファイル選択のダイアログが出たら、次のファイルを追加します。
C:¥Program Files¥SourceBoost¥Lib¥libc.pic16.lib
ライブラリのフォルダには、いくつかのライブラリがありますので、間違えないようにしてください。
ファイルが追加されると、プロジェクトウィンドウは、次のように、追加したライブラリが表示されます。

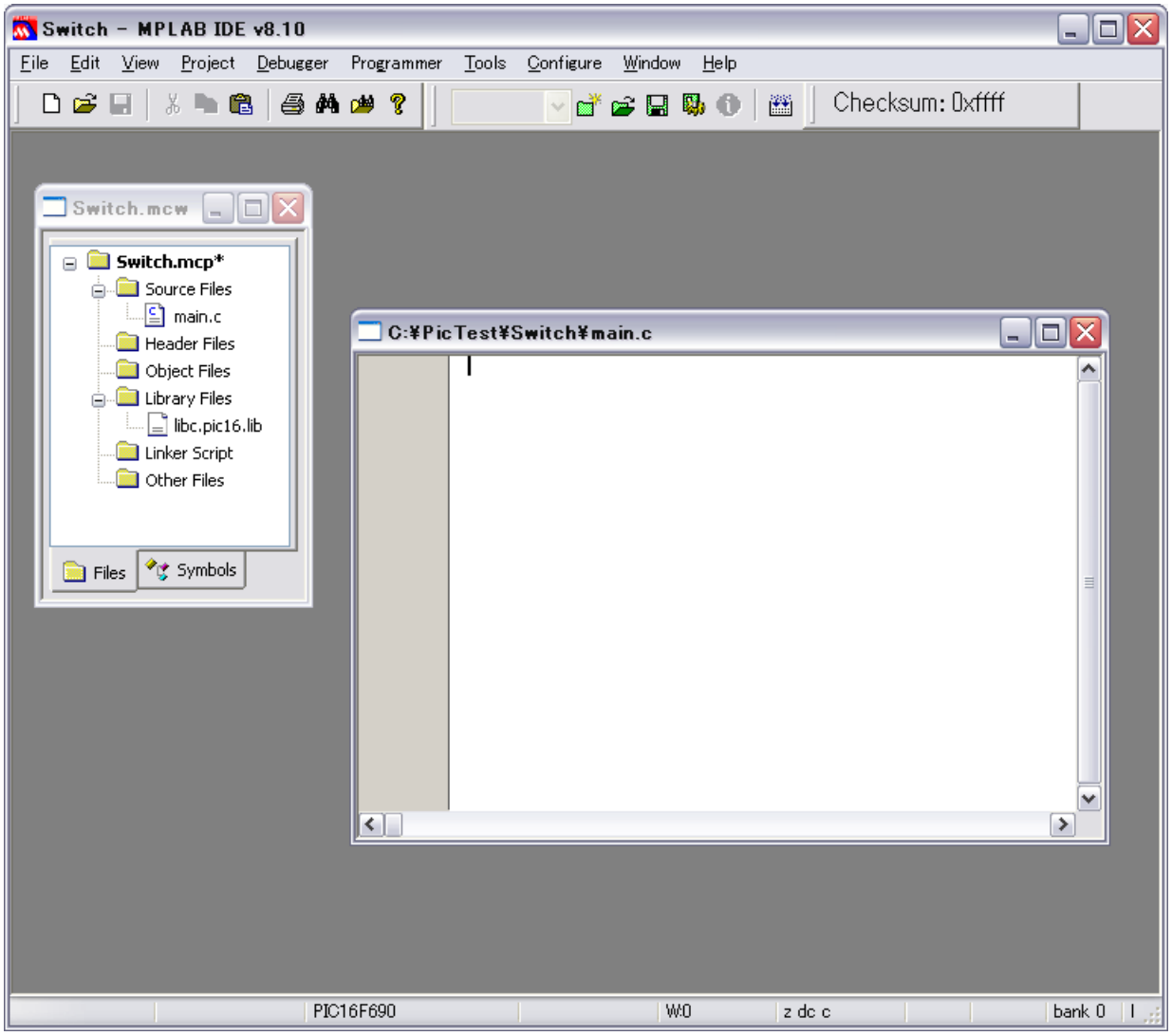


ソースプログラムの作成

ライブラリの追加が終わったら、いよいよプログラムの作成です。
ここでは、新しく、main.cというファイルを作成することにします。main.cの作成手順は、
次の通りです。

1. Fileメニューから、Newを選択する
2. Untitledという編集ウィンドウが表示されるので、一旦Fileメニューから、Save Asを選択して、c:¥PicTest¥Switchフォルダに、main.cというファイル名で保存する。

3. プロジェクトウィンドウの、Source Filesのアイコンを、マウスで右クリックして、Add Filesを選択します。ファイル選択のダイアログが出たら、作成したmain.cを追加します。これでプロジェクトに、main.cが追加されました。プロジェクトウィンドウには、次のようにmain.cが追加されます。



ソースプログラムを記述する

今までの手順で、プロジェクトの準備ができましたので、main.cにソースを記述します。
今回使用するのは、スイッチとLEDです。PIC16トレーナーでは、基板上にスイッチとLED
がありますので、それを使って実験を行います。LEDは4個ありますが、そのうちRC0に接
続されているLEDを使用することにします。
今回使用する回路のI./Oの設定は、次のようになります。

GPIO	機能	入出力
RA3	プッシュスイッチ	入力
RC0	LED	出力

PICのGPIOポートは、PORTA、PORTB・・・という名前ですが、ポートの特定ビットを表すのに、RA3やRC0のように、ポート名とビット番号を使って表すことがよくあります。RA3はPORTAのビット3、RC0はPORTCのビット0を表します。

main.cに記述するプログラムは、次のようになります。

```
==========================================================================
#include <system.h>

#pragma DATA _CONFIG,_INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_OFF & _MCLRE_OFF & _CP_OFF & _BOR_OFF & _IESO_OFF
& _FCMEN_OFF;
#pragma CLOCK_FREQ  8000000
void main()
{
    char sw=0;
    osccon=0x70;    //CLK=8MHz(Internal)
    trisa=0x04;
    trisc=0;
    portc=0x00;
    while(1){
        while(test_bit(porta,3));   //スイッチが押されるのを待つ
        delay_ms(30);               //チャタリング防止
        //LED の反転
        if(sw)
            clear_bit(portc,0);
        else
            set_bit(portc,0);
        sw=!sw;
        while(!test_bit(porta,3));  //スイッチが放されるのを待つ
        delay_ms(30);               //チャタリング防止
    }
}
==========================================================================
```

以下で、プログラムの簡単な説明を行います。

＃include　<system.h>

最初に、system.hのインクルードを行います。このファイルは、PICのデバイスの種類に合わせて設定

を行うためのもので、ソースブーストでPICのプログラムを書く場合は、必ず記述します。

#pragma DATA_CONFIG

#pragma DATA_CONFIGは、コンフィグレーションビットの設定を行うためのものです。コンフィグレーションビットは、プログラム書き込み時に、PICの初期動作を決定するためのものです。たとえば、PIC16F690は、内部クロックと外部クロックのどちらでも使用できますが、クロックは、PICが動作するために必要なため、プログラムを実行する前に、どちらのクロックを使用するかを設定しておく必要があります。また、RA3は、リセットにも入力ポートにも使用できますが、これもプログラムの起動前に設定できていないと、リセットが入らず、プログラムが動作しなくなってしまう可能性があります。このように、プログラム実行前に設定する項目などを、コンフィグレーションレジスタという、特殊なレジスタに書き込んで、PICの動作を決定するようになっています。

#pragma CLOCK_FREQ

動作クロック周波数を、コンパイラに通知します。今回は、内蔵クロックの8MHzを使用するので、8000000を設定しています。

ハードウェアの初期化

メインプログラムの最初で、次のようにハードウェアを初期化しています。

```
osccon=0x70;
trisa=0x04;
trisc=0;
portc=0x00;
```

最初のosccon の設定は、内部クロック周波数の選択です。8MHzにするには、この値を設定します。
また、trisaとtriscは、それぞれPORTAとPORTCの方向レジスタで、8ビットのデータのうち、1のビットが入力、0のビットが出力となります。これは、アルファベットのIとOを読み替えればよいので、非常に覚えやすいです。ここでは、RA3のみが入力となるので、trisaに4を設定しています。
また、LEDの初期値を決めるため、PORTCを0にしています。PIC16トレーナーは、1を出力したときに、LEDが点灯するようになっています。

メインプログラム

スイッチの状態を調べるには、test_bitという関数を使用します。この関数は、ポートの指定ビットの状態を返します。スイッチが押されていない状態では、ポートはプルアップされているため、１となり、押されると０となります。したがって、最初にスイッチが押されるまで待つには、

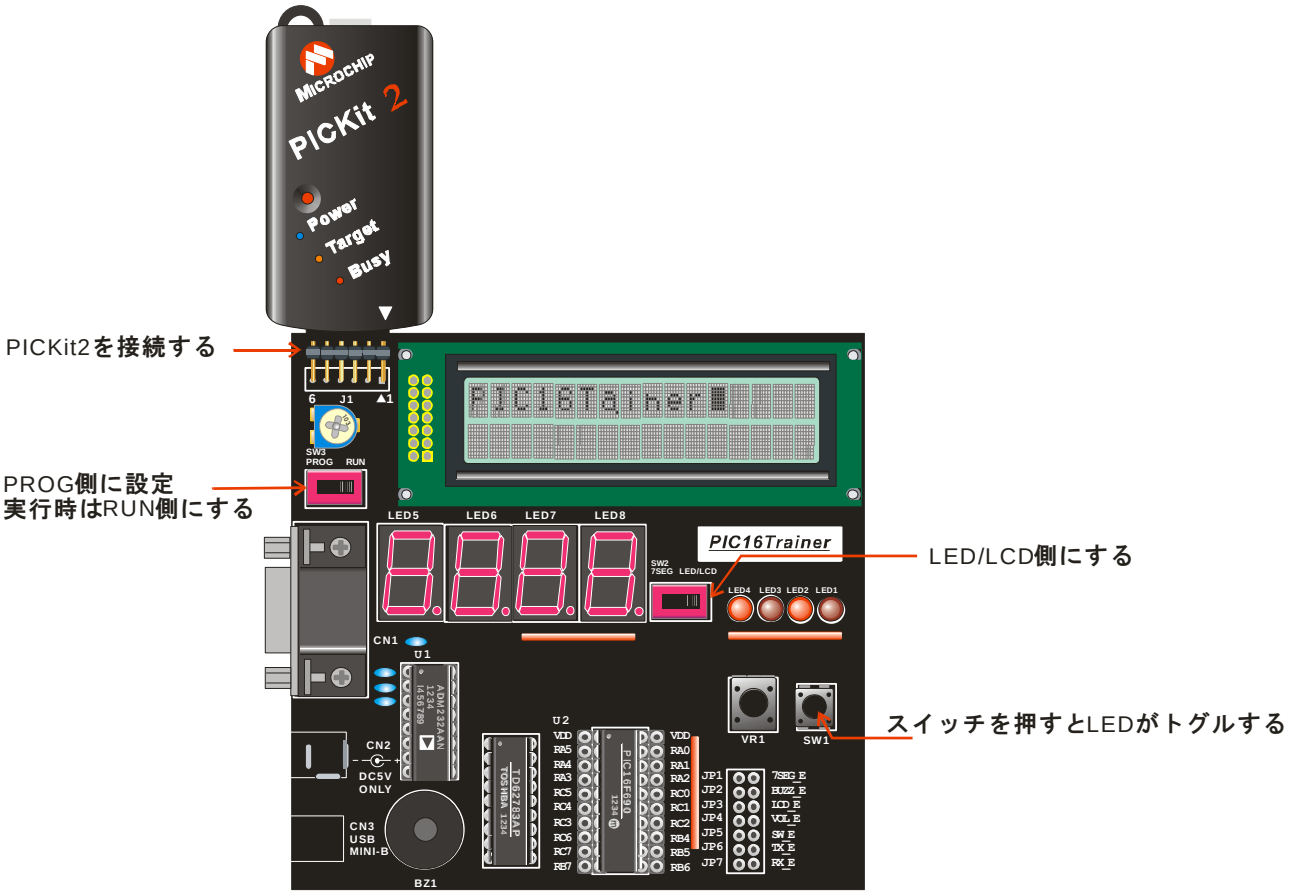
```
while(test_bit(porta,3);
```

という処理で可能になります。スイッチが押されたら、LEDを反転する処理を行います。
LEDの状態を保存している、SWという変数の値を調べ、LEDが点灯中かどうかを調べます。点灯中なら消灯し、消灯中なら点灯します。最後にSW変数を反転させ、新しい状態を記憶します。
LEDの点灯／消灯の処理が終わったら、スイッチが放されるまで待ちます。
スイッチの状態を調べた後に、チャタリング防止のために、ディレイ関数を呼び出しています。チャタリングは、スイッチのON/OFF時に、スイッチのばねの振動や、接触表面の状態により、短い時間（通

常は、数ミリ～30ms程度）の間、ON/OFFが繰り返される現象です。チャタリングが発生しても、誤動作しないよう、ここでは30msの間、スイッチを無視するようにしています。
この処理は、while文で無限ループになっているため、スイッチが押されるたびに、LEDがON／OFFするようになります。

プログラムのコンパイル

プログラムの記述が終わったら、早速プログラムをコンパイルしてみることにします。
PC の USB コネクタに PICKit2 を接続し、また PICKit2 を PIC16 トレーナーに接続します。また、SW3 は PROG 側、SW2 は、LED/LCD 側になっていることを確認してください。



接続が正しくなっている事を確認したら、Project メニューから Build を選択して、コンパイルを行います。エラーが出てしまったら、どこかに記述ミスがありますので、ソースを再度確認してみてください。プログラムが正常にコンパイルできたら、プログラムを書き込んで、実行を行います。

プログラムの書き込みと実行

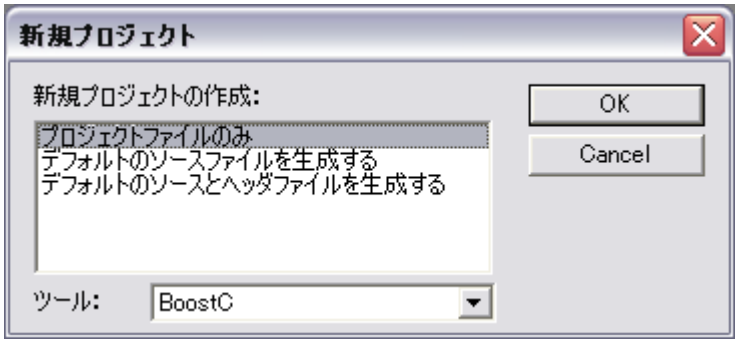
プログラムを書き込む前に、プログラマの設定を行います。Programmer メニューで、Select Programmer から、PICKit2 を選択してください。これで、プログラマの設定は終わりですので、実際に PIC16F690 に書き込みを行います。
Programmer メニューで、Program を選択すると、ターゲット基板の PIC にプログラムを書き込む事ができます。エラーが出る場合は、PICKit2 が正しく接続されているか、また SW3 が PROG 側になっているかを確認して、再度プログラムを行ってください。
プログラムが正常に終わったら、最後に実行して、テストを行います。SW3 を RUN 側にして、SW1 を押す度に、LED が反転している事を確認してください。うまく動作しない場合は、SW2 が LED/LCD 側になっているか、また、SW3 が RUN 側になっているか、また PICKit2 の Target の LED（オレンジ色）が点灯しているかを確認してください。PICKit2 の Target の LED が点灯していない場合は、MPLAB の Programmer メニューから、Set Vdd On を選択してください。

以上で、一通りのプログラムの動作確認は完了です。
今回作成した、main.c のソースは、main 関数以外の部分は、他のプログラムでも、共通に使えますので、他のプログラムを書く場合、このソースをコピーして、main 関数部分を書き換えるようにすると、ソースコードの記述の手間が省けますし、ソースコードの記述時の間違いも防ぐことができます。

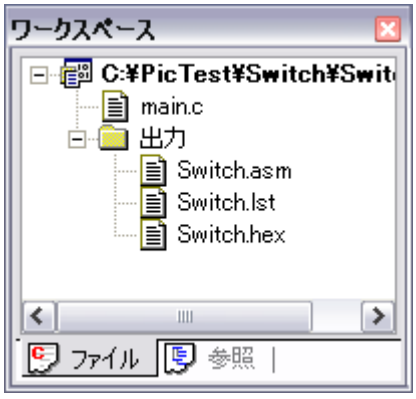
ソースブースト IDE でのシミュレーション

次のサンプルに入る前に、ソースブーストで、スイッチと LED のテストプログラムをシミュレーションしてみましょう。
プロジェクトの作り方は、MPLAB の場合とよく似ていて、最初に空のプロジェクトを作成し、その後ファイルの追加を行います。プロジェクトの作成手順は、次の通りです。

1．　プロジェクトメニューから、新規プロジェクトを選択する
2．　プロジェクトファイルのパス名を入力するダイアログが開くので、c:¥PicTest¥Switch フォルダを開き、ファイル名を“Switch”とする。
3．　次のような画面が表示されるので“プロジェクトファイルのみ”を選択して、ツールが BoostC になっていることを確認して、OK を押す



4．　プロジェクトウィンドウのツリーで、マウスを右クリックし、“ファイルをプロジェクトに追加”を選択して、main.c を追加する



5．　設定メニューで、ターゲットデバイスを”PIC16F690”、クロック周波数を 8000000 に設定する。

以上の操作でプロジェクトの作成は完了です。MPLAB と同様に、プロジェクトウィンドウの main.c をマウスでダブルクリックすれば、編集用のウィンドウが開き、プログラムソースの編集ができます。なお、ソース中に日本語のコメントを入れたい場合は、フォントの設定を MS ゴシックなどの、日本語フォントに変更してください。

プログラムのコンパイル

C 言語のプログラムは、プログラムをコンパイルした後、リンクを行って、初めて実行用のモジュールが作成されます。今回のように、単一のソースの場合は、コンパイルとリンクを行うのは、2 度手間のように感じてしまいますが、1 つのプログラムが、たくさんのソースファイルで構成されている場合は、このような構成の方が、かえって手間がかかりません。1 度のコンパイルで、実行モジュールを作ってしまうと、1 つのソースのみ変更しても、全部のプログラムをコンパイルし直さなければなりませんが、コンパイルとリンクが別になっていると、変更のあったプログラムのみコンパイルし直して、リンクを行えば良いためです。しかしながら、コンパイルをしてリンクをするという作業は、毎回発生してしまい、これも煩雑ですので、通常はビルドという操作を行い、この一連の作業を自動化できるようになっています。次の図は、ソースブーストの、コンパイルに関するボタンです。


ビルドボタン


コンパイルボタン


リンクボタン

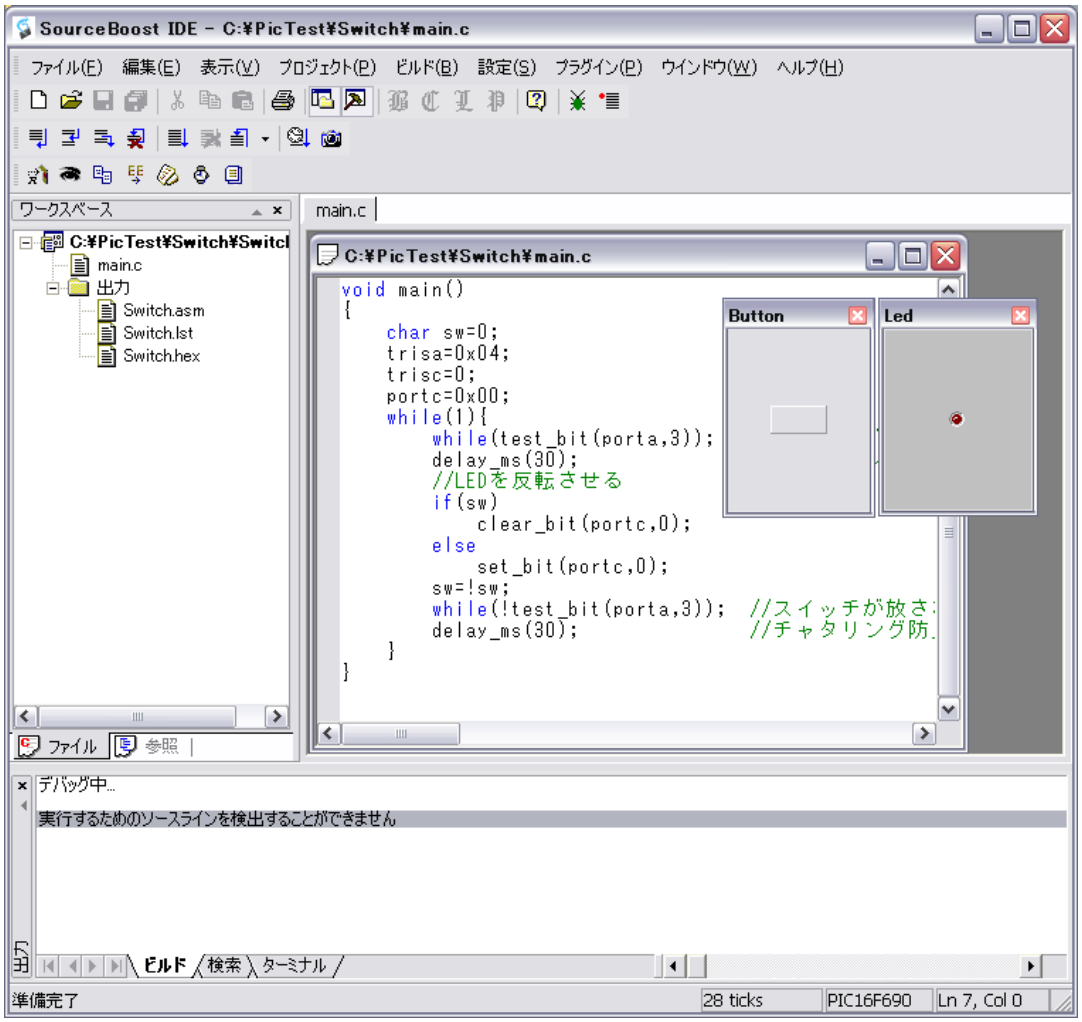
手動でコンパイルとリンクを行う場合は、コンパイルボタンでコンパイルした後、リンクボタンでリンクを行い、実行モジュールを作成します。ビルドボタンは、この 2 つの操作を自動で行います。ただし、ビルドでコンパイルされるファイルは、最後にコンパイルされてから、変更のあったファイルのみです。ソースファイルを変更せず、オプションのみ変更した場合は、ビルドボタンを押しても、ソースに変更がないため、再コンパイルされない場合があります。このような場合は、手動でコンパイルボタンを押して、コンパイルし直してください。

仮想デバイスによるシミュレーション

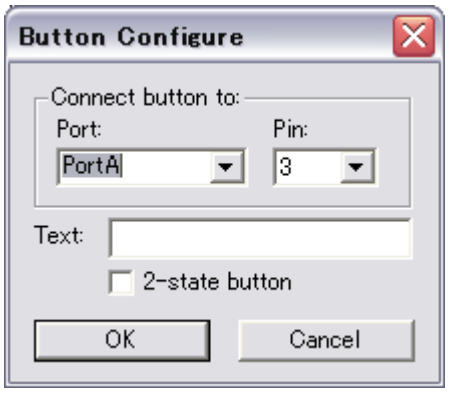
正常にビルドを行ったソースファイルは、仮想デバイスのシミュレーションを使って、PC 上でシミュレーションを行うことができます。

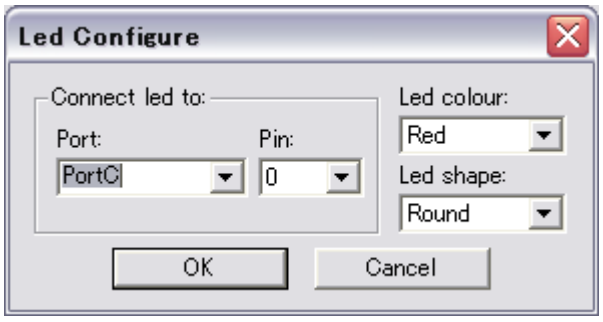
シミュレーションは、シミュレーションボタン  で開始します。

シミュレータが起動すると、シミュレーションボタンのアイコンが、緑色に変わります。再度このボタンを押すと、シミュレータは終了し、もとの赤いテントウ虫になります。
今回のプログラムは、ボタンと LED が 1 つづつありますので、シミュレータを起動したら、プラグインから、ボタンと LED を起動してください。次の図は、ボタンと LED のプラグインを起動した画面です。



仮想デバイスは、ウィンドウ上でマウスを右クリックすると、構成を変更することができますので、ボタンと LED の構成を、次の図のように、ボタンを PORTA のビット 3、LED を PORTC のビット 0 に設定します。





ボタンを設定したら、実行ボタン  を押して、プログラムを実行してみてください。Button ウィンドウのボタンを押す毎に、Led ウィンドウ上の LED が点灯、消灯して、プログラムが正しく動作していることが確認できます。

そのほか、シミュレータでは、プログラムをステップ実行や、変数やレジスタのウォッチなど、便利な機能があります。詳しくは、ソースブーストのマニュアルを、ご参照ください。

割り込みテストプログラム

■概要

周期割り込みを使って、LED を点滅させる

■サンプルプロジェクトの格納フォルダ

Interrupt

■ポイント

タイマと割り込みの使い方を理解する

■テスト回路とプログラム

テスト回路は、スイッチと LED のテストと同じです。ただし、スイッチは使用しません。
ここでは、タイマを使用して、LED を約 0.5 秒毎に ON/OFF させてみることにします。

■ソフトウェアの説明

PIC16F690 には、3 個のタイマが内蔵されていますが、このうちタイマ 0 は 8 ビットのタイマで、最も基本的なタイマです。このタイマは、内部のシステムクロックや、プリスケーラの出力、あるいは外部のクロックで動作させることができます。
PIC のシステムクロックは、発振周波数の 1/4 ですので、今回のボードでは、8/4=2MHz がシステムクロックとなります。
タイマ割り込みは、タイマがカウントアップして、オーバーフローとなり、FFh から 00h に変わるところで発生します。例えば、タイマの入力クロックがシステムクロックであれば、1 クロックの周期は、1/2MHz で、0.2μs となります。オーバーフローは、256 回に 1 回発生しますので、割り込みの周期は、0.5μs×256=128μs です。これで LED を点滅させると、あまりにも速すぎて、人間の目には、点滅しているようには見えません。そこで、プリスケーラを使って、さらに点滅間隔を遅くすることにします。
プリスケーラは、クロックの分周器で、システムクロックを分周して、タイマに入力することができます。PIC16F690 では、最大で 256 分周まで行う事ができます。256 分周すると、割り込み周期は 256 倍されますので、割り込み周期は、128μs×256≒33ms となります。割り込み周期が 33ms とすると、15 回の割り込み毎に LED を反転させれば、約 0.5 秒の周期で LED が点滅することになります。

割り込みを使ったソースプログラム

割り込みを使ったプログラムは、interrupt という特別な名前の関数を用意します。この関数は、割り込みが発生すると自動的に呼び出されます。

次のリストは、割り込みを使ったプログラムのソースです。

```
#include <system.h>

#pragma DATA _CONFIG,_INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_OFF & _MCLRE_OFF & _CP_OFF & _BOR_OFF & _IESO_OFF
& _FCMEN_OFF;
#pragma CLOCK_FREQ  8000000

//CLK=20MHz,PreScaler=1/256,TimerOverflow=256,TimerInterval=3.27ms/4=13.1ms
#define LED_BIT 0
#define SEC_CNT 15      //33msx15=0.5Sec

//Timer 0 handler
static void OnTimer( void )
{
    static char tcnt=0;
    static char sw=0;
    tcnt++;
    if(tcnt>=SEC_CNT){
        if(sw)
            clear_bit(portc,LED_BIT);
        else
            set_bit(portc,LED_BIT);
        sw=!sw;
        tcnt=0;
    }

}


void interrupt( void )
{

    //Handle timer0 interrupt
    if( intcon & (1<<T0IF) )
    {
        OnTimer(); //call timer 0 handler

        clear_bit( intcon, T0IF ); //clear timer 0 interrupt bit
    }

}

void main( void )
{
    //Configure port A
    trisa = 0x00;
    //Configure port B
    trisc = 0x00;

    //Initialize port A
    porta = 0x00;
```

```
    //Initialize port B
    portc = 0x00;
    //Set Timer0 mode
    clear_bit( option_reg, T0CS ); //configure timer0 as a timer
    //Set prescaler assignment
    clear_bit( option_reg, PSA ); //prescaler is assigned to timer0
    //Set prescaler rate
    set_bit( option_reg, PS2 ); //prescaler rate 1:256
    set_bit( option_reg, PS1 );
    set_bit( option_reg, PS0 );
    //Set timer0 source edge selection
    set_bit( option_reg, T0SE ); //increment on high-to-low transition on RA4/T0CKI pin


    //Enable interrupts (Timer0)
    intcon = 0xA0;


    //Endless loop
    while( 1 );
}
```

メインプログラムでは、初期化を行った後は、while(1);という永久ループの処理で終わっています。
LED の点滅は、すべて割り込みルーチン内で行っているためです。
先ほどと同じ要領で、PIC16 トレーナーにプログラムを書き込んでテストしてみてください。
LED が約 0.5 秒間隔で点滅すれば、テストは完了です。
なお、このプログラムを、シミュレータでテストする場合は、点滅が非常に遅くなりますので、注意が必要です。ソフトウェアのシミュレーションは、構造上どうしても、実際のデバイスよりも動作が遅くなります。これは、デバイスの各種レジスタの状態やピンの状態など、全ての状態をシミュレーションする必要があるためです。このため、今回のようなプログラムのシミュレーションを行うと、点滅が遅すぎて、実際にシミュレーションにならない場合があります。このようの場合、デバッグ時のみ、割り込み周期を短くすると、PC 上でもシミュレーションが出来るようになります。

RS232 のテスト

■概要
UART モジュールを使って RS232 通信を実現する

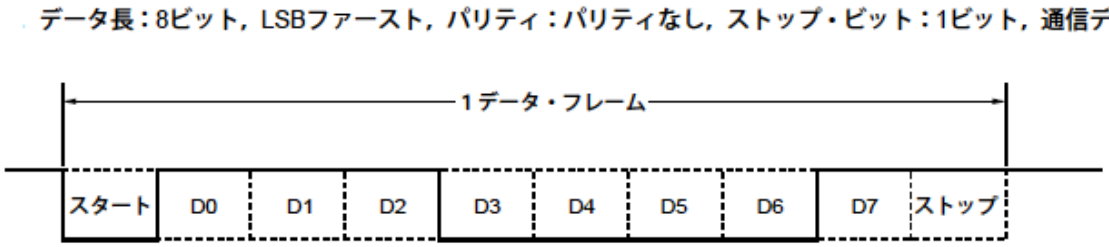
■サンプルプロジェクトの格納フォルダ
RS232

■ポイント
・ RS232 の使用を理解する
・ ソフトウェアで RS232 通信を実現する手法を学ぶ

※このサンプルには、拡張キットが必要です。

RS232 の通信は、PC とマイコンとの通信を行うのに、最も簡単な手段です。RS232 通信は、PC からマイコンを制御したり、マイコンで取得したデータを PC に取り込む場合に、非常に便利です。PIC16F690 には、UART モジュールがあるため、簡単に RS232 のシリアル通信を行うことができます。RS232 は、多くのデスクトップの PC には、9 ピンの D-SUB コネクタが内蔵されています。また、一部のノート PC のように、RS232 ポートが無いものもありますが、市販の USB-COM 変換アダプタを使えば、簡単に使うことができます。

■RS232 の仕様
RS232 は、比較的単純な、非同期のシリアル通信です。信号レベルは、PIC で使用している 5V のレベルとは異なりますが、レベル変換は、上記回路の MAX232 が行います。
次の図は、一般的な RS232 通信のデータフォーマットです。

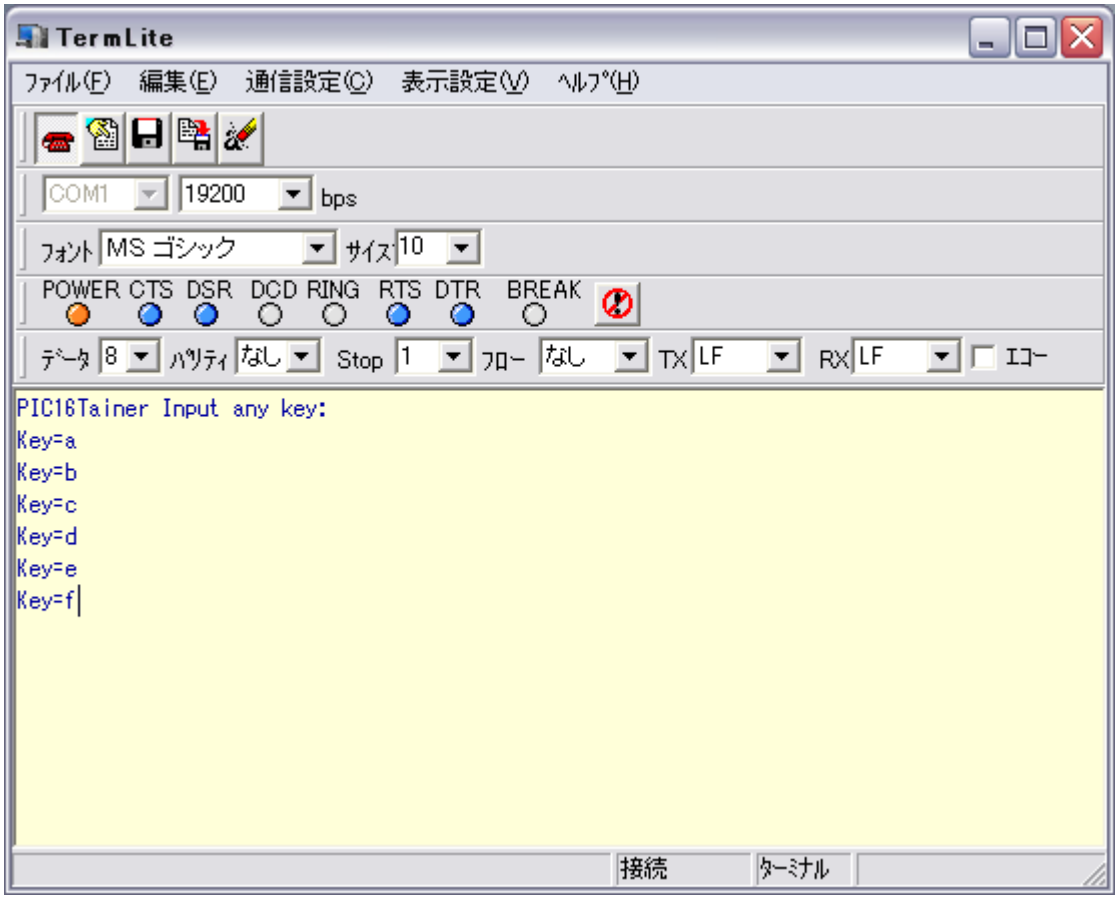


1 ビットのデータの幅は、通信速度で決まります。例えば、9600bps の場合は、1 ビットの幅は、1/9600sec ですので、約 104μs のパルス幅となります。

■ソフトウェアの説明
このテストでは、起動時に、"PIC16Tainer Input any key:"と表示し、その後は、受信した文字を表示するようになっています。ハイパーターミナルなどのターミナルソフトを PIC16 トレーナーの RS232 コネクタに接続して、テストしてみてください。

通信速度の設定は、19200bps、8 ビット、パリティなし、フロー制御なしに設定します。
なお、RS232 のサンプルは、main.c と Serial.c の 2 つのソースに分かれています。RS232 関係のソースは、Serial.c にまとめられています。新規にプロジェクトを作成する場合は、この 2 つのソースをプロジェクトに含める必要があります。

■ターミナルソフトについて
ターミナルソフトは、RS232 通信などを行うアプリケーションで、フリーウェアもいくつか存在します。Windows には、ハイパーターミナルというターミナルソフトが付属していますが、WindowsVISTA には付属していません。
PIC16 トレーナーには、TermLite というターミナルソフトが入っていますので、ターミナルソフトがない場合は、こちらをお使いください。
次の図は、TermLite で、サンプルを動作させている画面です。



EEPROM のテスト

■概要

RS232 通信で受信したデータを EEPROM に書き込み、リスタート時にそれを表示する

■サンプルプロジェクトの格納フォルダ

EEPROM

■ポイント

・ EEPROM の使い方を理解する

このテストでは、RS232 テストで使用した RS232 モジュールををそのまま使用します。

※このサンプルには、拡張キットが必要です。

■EEPROM の概要

PIC16F690 には、データ格納用の、256 バイトの EEPROM が内蔵されています。EEPROM は、電源が切断されても、データが消えたいため、初期化用のデータや、設定値、あるいは製品毎のシリアルナンバーなどを保存する場合に便利です。

■ソフトウェアの説明

EEPROM のアクセスは、次の 4 つのレジスタで行います。

レジスタ名	機能
eedata	EEROM のデータレジスタ。EEPROM へのアクセスは、すべてこのレジスタを通して行います。
eeadr	EEPROM のアドレスレジスタ。eedata への読み出しまたは、eedata の書き込みのアドレスを指定します。
eecon1	EEPROM の制御レジスタ。EEPROM の読み書きの制御を行います。
eecon2	EEPROM の制御レジスタ。不用意に EEPROM を書き換えないため、書き込みのためのシーケンスを制御するために使用します。

1）EEPROM からの読出し

EEPROM の読出しシーケンスは、次のようになります。

・ eeadr に、読み出したいデータのアドレスをセットする

・ eecon1 の RD ビットを 1 にする

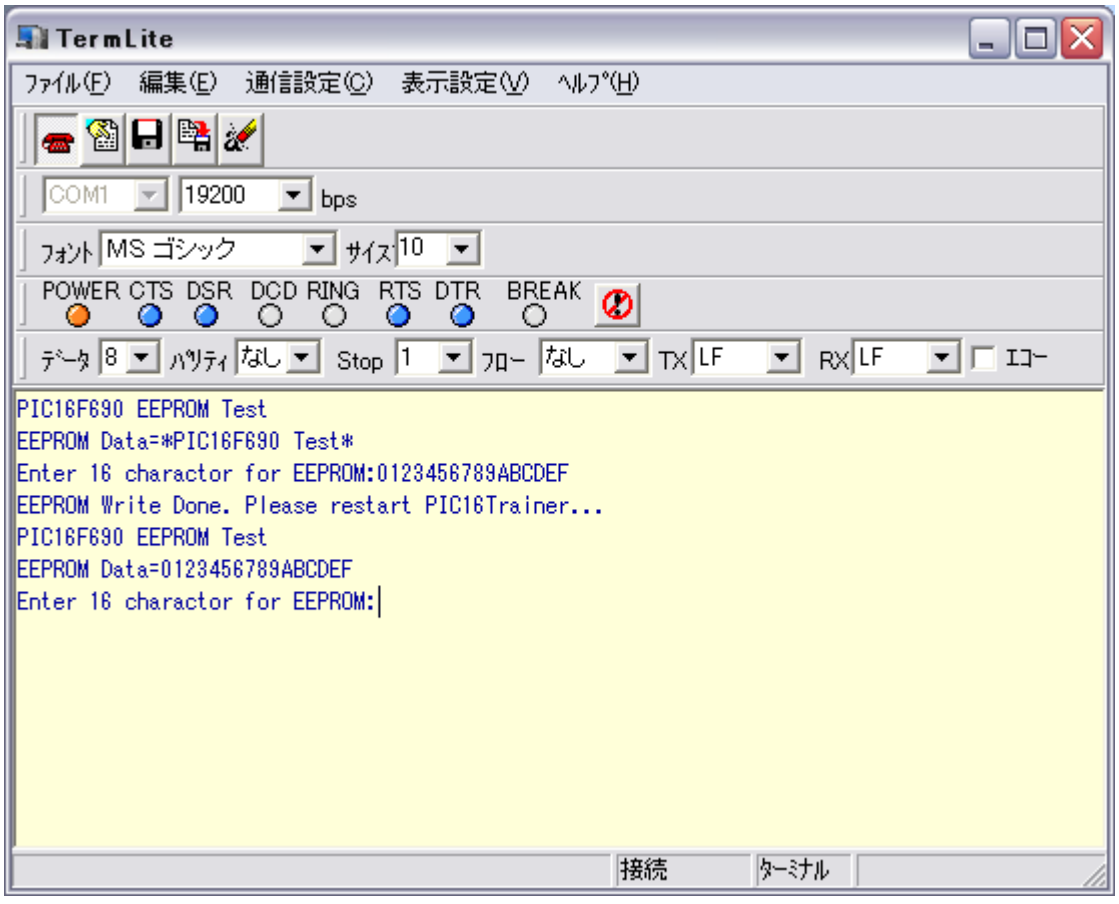
・ 設定したアドレスのデータが、eedata にセットされるので、eedata を読み出す

以上のように、EEPROM の読出しは、非常に簡単に行う事ができます。

また、EEPROM への書き込みは、次のようになります。

- eecon1 の WREN ビットを 1 にして、EEPROM の書き込みを有効にする
- eeadr に、データを書き込むアドレスをセットする
- eedata に、書き込むデータをセットする
- eecon2 に、0x55 をセットする（書き込みシーケンス 1）
- eecon2 に、0xaa をセットする（書き込みシーケンス 2）
- eecon1 の WR ビットを 1 にして、書き込みを開始する
- eecon1 の WR ビットが、0 になるまで待つ（書き込みの完了待ち）

EEPROM への書き込みは、通常数 ms 必要としますので、必ず書き込みの完了待ちを行う必要があります。
サンプルのテストプログラムでは、16 文字の書き込みをテストします。デフォルトでは、EEPROM に、"*PIC16F690 Test*"という文字が書き込まれています。PC にシリアルケーブルで接続し、ターミナルソフトを起動して、通信状態にしておきます。通信設定は、RS232 テストと同じで、19200bps の、8 ビットパリティなし、ストップビット 1 で、フロー制御無しにしておきます。
ターミナルソフトの接続状態で、PIC16 トレーナーボードの電源を入れると、起動メッセージの後に、"*PIC16F690 Test*"とう文字が表示されます。(MPLAB の Programer メニューで、Vdd On と Vdd Off を切り替えると、電源の ON/.OFF ができます。)
次に、文字の入力が促されますので、16 個の適当な文字を打ち込みます。最後に、書き込みが完了したメッセージが表示されますので、トレーナーボードの電源を入れなおしてください。
起動時に表示されるメッセージが、今打ち込んだ文字に変わっていたら、実験は成功です。
次の画面は、EEPROM テストで、16 進の文字を入力した結果です。



BEEP サウンド

■概要
ブザーを使って、さまざまな周波数の音を発生させる
ブザー音の周波数を、連続的に変化させて、サイレンのような音を出す

■サンプルプロジェクトの格納フォルダ
Sound

■ポイント
・ タイマを使って周波数制御をする

■テストの内容
このプログラムは、タイマ割り込みで、一定周期の割り込みを発生させ、BEEP サウンドを発生させます。PIC16 トレーナーの RB6 には、圧電ブザーが接続されており、これを使って音を鳴らすことができます。
音階を出すには、特定の周期でポートを ON/OFF すれば良いので、サンプルでは、この周期を作り出すために、タイマ割り込みを使用しています。
このサンプルでは、システムクロックをプリスケーラで 16 分周し、タイマ割り込みを発生させています。タイマの割り込みルーチンでは、タイマの値を任意の値に初期化するようにしています。タイマの値が 0 であれば、割り込みの周波数は、8MHz/4/256/16=488Hz の周波数であり、この割り込みの度にブザーを ON/OFF すると、実際の音の周波数は、この半分の 244Hz となります。タイマを再初期化する際、タイマの値を 0 以外の値にすると、タイマ割り込みの周期を変化させることができます。再初期化をしない場合は、タイマは、256 カウントしたところで、次のオーバーフロー割り込みを発生しますが、再初期化で 128 を設定すると、残りの 128 カウントでオーバーフロー割り込みが発生するので、割り込みの周期は半分になり、再生される音は、2 倍の周波数となります。タイマの再初期化の値を連続的に変化させれば、再生音も連続的に変化させることができます。
このサンプルでは、タイマの再初期化の値を、0-128 の間で、インクリメントし、128 になると、今度はデクリメントして行き、0 になったらまたインクリメントを開始するようにしています。
なお、タイマの再初期化の値を、あまり大きくすると、割り込みの周期が短くなりすぎて、プログラムが暴走してしまいますので注意してください。これは、タイマ割り込みの処理中に、次の割り込みが発生してしまうためで、これを繰り返すと、スタックオーバーフローが発生し、プログラムが暴走してしましまいます。

ADC のテスト

■概要

A-D コンバータモジュールを使ってボリューム(VR1)の値を 7 セグ LED(LED8)に表示する

■サンプルプロジェクトの格納フォルダ

ADC

■ポイント

・ ADC モジュールの使い方を習得する

■テストの内容

PIC16F690 には、10 ビットの AD コンバータが内蔵されています。AD の入力ピンは、AN0~AN11 まであり、シリアル線や GPIO 線などの他の機能とマルチプレクスされています。AD コンバータ使用の手順は以下の通りです。上から順に実行します。

	内容説明	操作
初期設定	アナログ入力するポートの向きをインプットに設定	TRISA レジスタの該当ポートピンを立てる
	ポートを A/D どちらで使用するかを選択	ANSEL レジスタの該当ビットを立てる
	ADC で使用するチャンネルを設定	ADCON0 レジスタの CHS ビットを設定
	ADC リファレンス電圧を設定	ADCON0 レジスタの VCFG ピンを設定
	ADC のクロック設定	ADCON1 レジスタの ADCS ピンを設定
	ADC 取得値のフォーマットを設定(ADRESH/ADRESL に左/右詰めのどちらで格納するかを決める)	ADCON0 レジスタの ADFM ピンを設定
	ADC モジュールを起動	ADCON0 レジスタの ADON ピンを立てる
AD 変換部	アクイジションタイムを確保	下記「アクイジションタイム」を参照
	ADC 作業をスタート	ADCON0 レジスタの GO/DONE ピンと ADON ビットを立てる
	AD が終わるまで待つ	ADCON0 レジスタの GO/DONE ピンを監視
	AD 変換値を取得	ADRESH もしくは ADRESL の値を取得
	アナログ入力するポートの向きをインプットに設定	(上記 4 つを繰り返す)

ADC モジュールでは割り込みを利用した変換も可能ですが、このサンプルでは、割り込みは使用していません。初期設定の部分はメイン関数中の IoInit()で行い、AD 変換部分は while 文内で行っています。本サンプルでは、ADC 計測値を ADRESH レジスタに左詰めで格納し、ADRESH を 4 ビット右にシフ

トすることで上位 4 ビットを取り出し、その結果を 16 進数表記で 7 セグ LED 一桁に表示しています。また、基準電圧を電源電圧の 5V としているので、ADC の分解能は、その電圧を 10 ビット、すなわち、1024 で割った値となります。つまり、1 ビットあたり、5/1024≒4.88mV になります。

■ADC のレジスタ

ADCON0：A/D 制御レジスタ 0

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ADFM	VCFG	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON

CHS3-0:　　A/D チャンネルの選択

GO/DONE:　　1:A/D 変換中　0:アイドル

ADON:　　1: A/D モジュール有効　0: A/D モジュール無効

ADCON1：A/D 制御レジスタ 1

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
-	ADCS2	ADCS1	ADCS0	-	-	-	-

ADCS2-0:　　ADC クロックの選択

ANSEL：アナログ選択レジスタ Low

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
ANS7	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0

ANS7-0:　　1:アナログインプット　0:デジタル I/O

その他のレジスタに関して、また詳しいレジスタの説明に関しては PIC16F690 のデータシートを参照下さい。

■アクイジションタイム

ADCON0 でチャンネルを選択すると、選択したチャンネルスイッチがつながり、そのチャンネルのサンプルホールドコンデンサへの充電が開始されます。この充電時間のことをアクイジションタイムといいます。十分な充電時間を確保しないと、正しい電圧が得られず、ADC の精度が下がることになってしまいます。よって、本サンプルではこの時間を多めに、100ms とってあります。

■プログラム実行時の注意

電源を PICkit 2 から流用している場合、PICKit2 の制限により、AD 変換値の最大値が 0xF まで上がらない場合があります。外部電源を利用している場合は、最大値が 0xF となります。

7 セグ LED の使用方法に関しては、後続のストップウォッチの製作を参照して下さい。

フルカラーLED の調光

■概要

ソフトウェアで LED の各色の明るさを制御して、ネオンサインのようなカラフルな色を表示させます。

■サンプルプロジェクトの格納フォルダ

RGB_LED

■ポイント

・ PWM を使って、明るさを制御する

※このサンプルには、拡張キットが必要です。

■RGB LED の結線

このテストでは、ブレッドボードを使って、実験を行います。

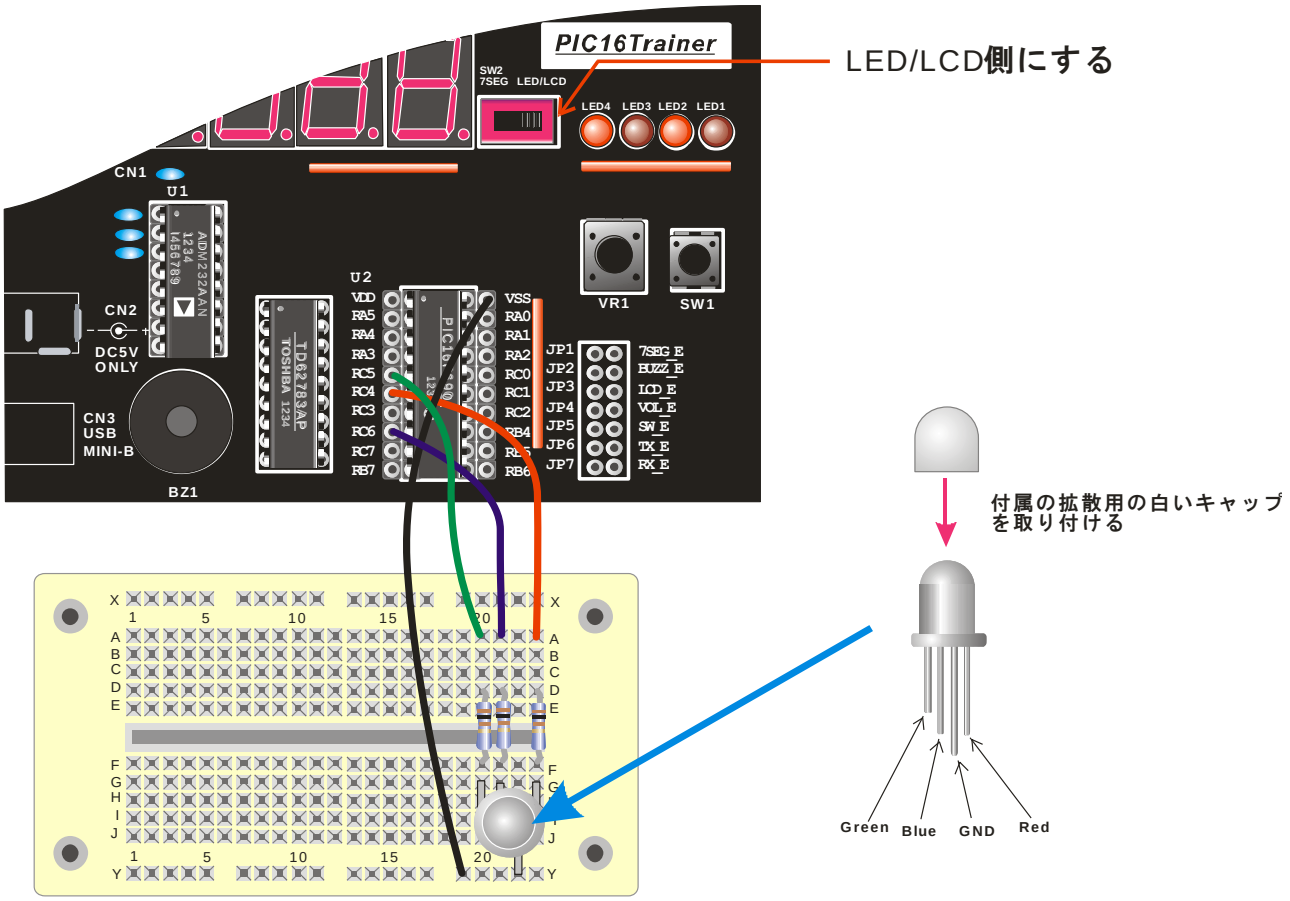
次の図は、このテストの結線図です。

赤→RC4(6pin)

緑→RC5(5pin)

青→RC6(8pin)

グランド(20pin)

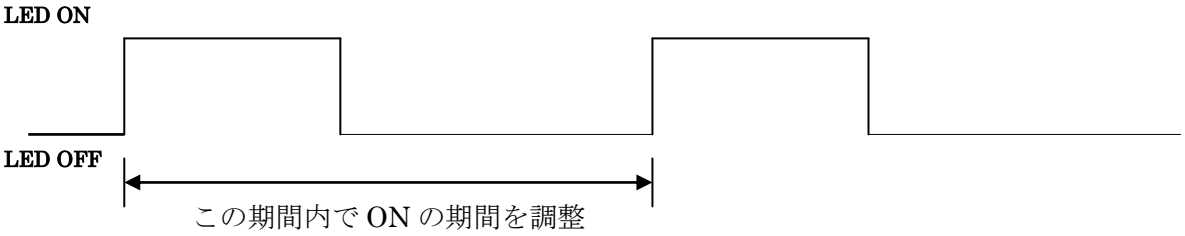


■プログラムの解説

青色 LED の発明により、LED も赤、青、緑の 3 色が揃い、これらの色の明るさを加減することで、フルカラーの発色を行う事ができるようになりました。

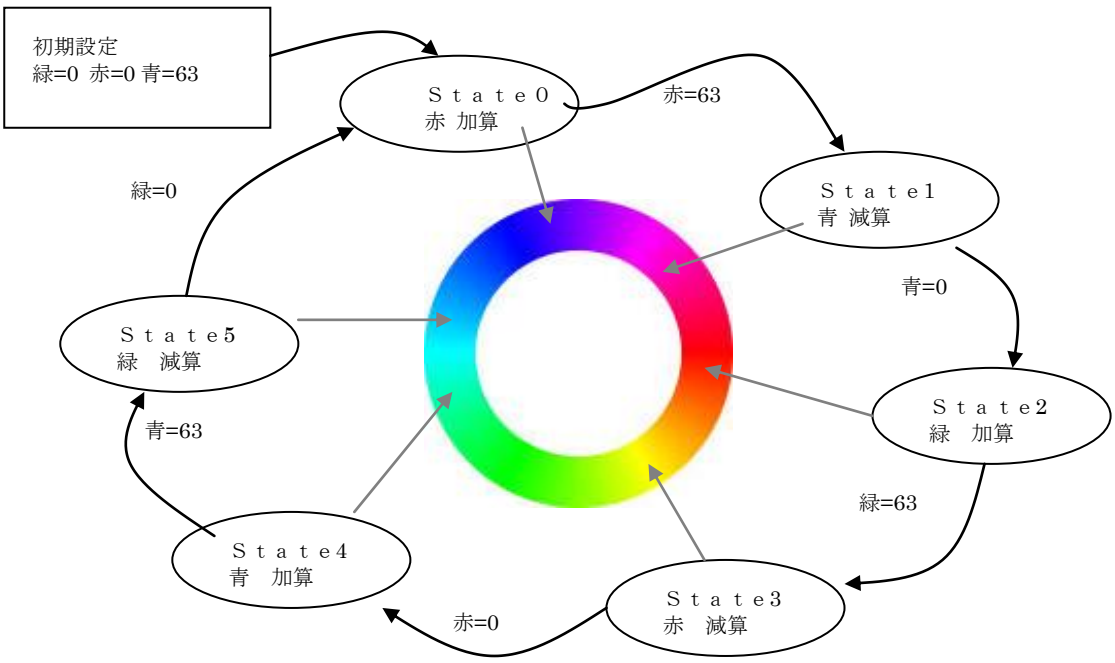
ただし、通常の方法で、PIC で LED を点灯すると、ON と OFF しか制御できないため、7 色しか表示できません。そこで、PWM のテクニックを使って、明るさの制御を行います。

PWM 制御では、図のようにある一定の時間の間に、LED を ON にする時間を制御します。



この方法は、実際には、LED は高速で点滅するのですが、点滅が早いと、人間の目には残像作用のため、連続的に発光しているように見えます。この場合 ON の期間が短ければ、暗く点灯しているように見え、ON の期間が長ければ、明るく見えます。したがって、RGB それぞれの LED の点滅を調整すれば、フルカラーで LED を点灯することができます。

サンプルプログラムでは、割り込みを使って PWM の制御を行っています。初期設定では、明るさを 0～63 までの 64 段階で調整し、割り込みのエントリーポイントでは、1 回割り込みが入る毎に、カウンタをインクリメントします。カウンタは、0～63 までカウントし、指定された明るさとカウンタの値を比較して、カウンタの値が指定された明るさよりも小さい間は、LED を ON にするようにしています。また、このループカウンタと(デフォルトは 63)、色変化の速度(デフォルトは 5)はマクロで定義できるようになっていますので、微調整に便利です。下記は色を連続的に変化させるためのアルゴリズムです。これによって滑らかに色が変化します。



LCD モジュールの表示

■概要

LCD モジュールにキャラクタを表示させる

■サンプルプロジェクトの格納フォルダ

LCD

■ポイント

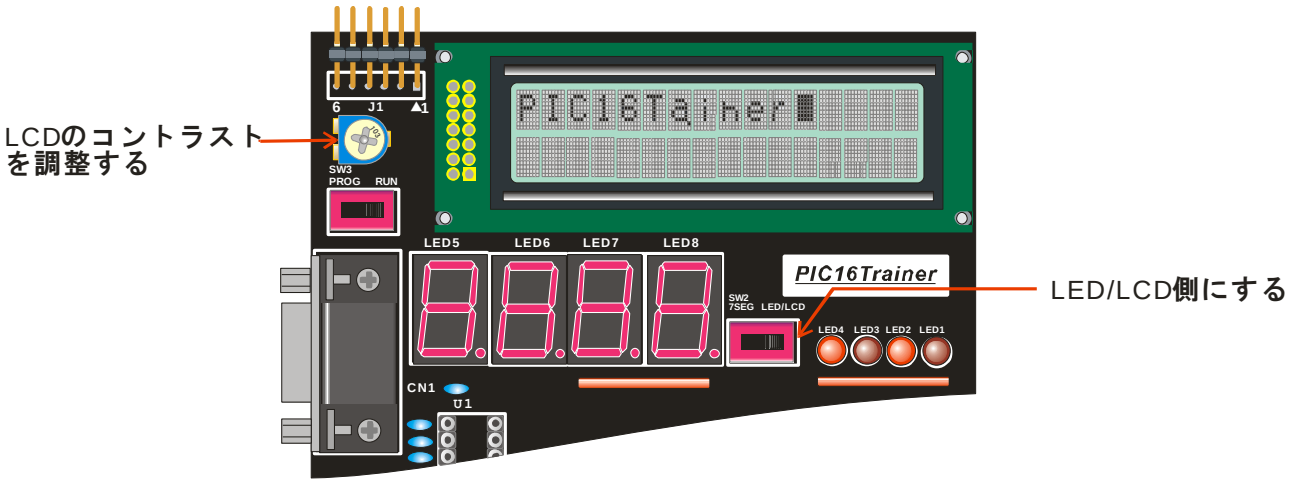
・ LCD モジュールを使って、キャラクタを表示させる

※このサンプルには、拡張キットが必要です。

※実行時は 7seg-LED/LCD 切り替えスライダを LCD 側に切り替えてください。

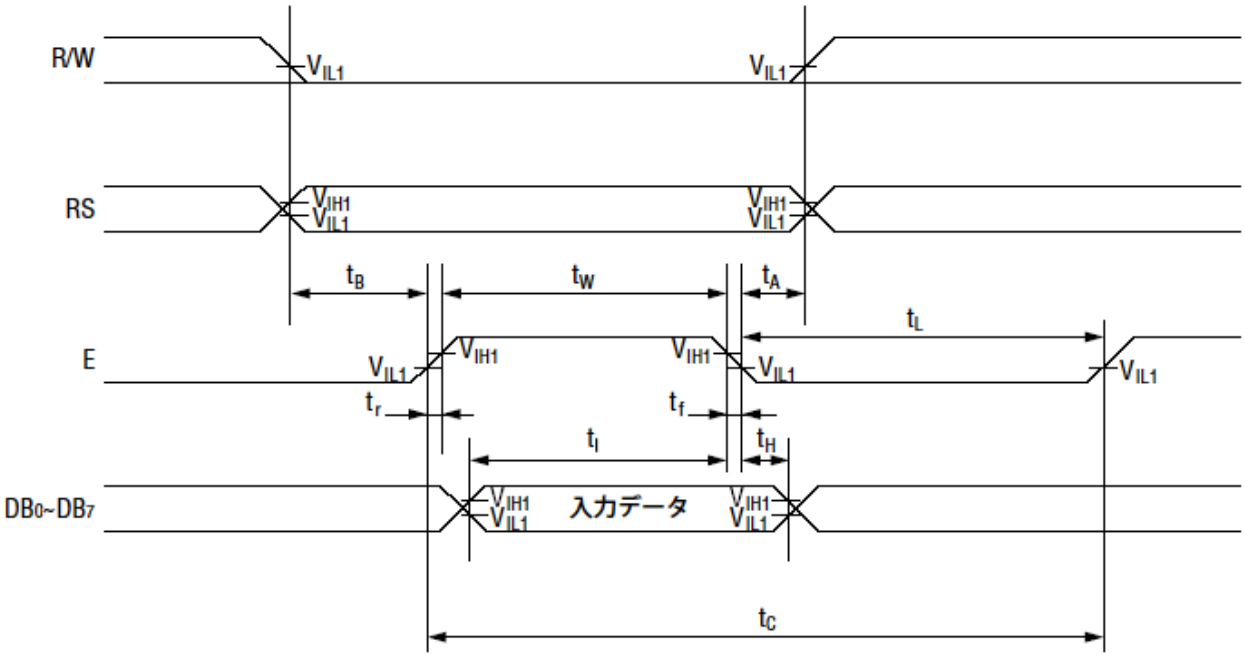
■基板の設定

次の図は、このテストを行うための、PIC16 トレーナー基板の設定方法です。



LCD 左側の POT1 は、LCD のコントラスト調整用のボリュームです。小型のドライバで回して、LCD のコントラストを調整してください。拡張キット付属の LCD モジュールは、SC1602B（又は相当品）です。これは 16 桁×2 行の表示が可能な LCD です。このモジュールには、日立製の LCD コントローラ IC、HD44780 互換の LCD コントローラが搭載されています。この種の LCD モジュールは、各社から発売されていて、LCD の桁数や行数は様々ですが、ほとんどの場合、同じコントローラを搭載しており、初期化部分を書き換えれば、同じように使用することができます。

HD44780 は、データバスのほか、RS、R/W、E の 3 つの信号で制御を行います。RS レジスタセレクトで、コマンド（ステータス）レジスタとデータレジスタの何れかを指定し、R/W は、リードかライトかを指定します。これらの信号のタイミングは次の図のようになっています。



プログラムで GPIO を使って制御する場合は、ソフトウェアで、これらのタイミングを作成します。データバスは、8 ビットですが、上位 4 ビットを使って、8 ビットのデータを 2 回に分けてアクセスすることも可能です。PIC16 トレーナーの回路では、信号線の数を節約するために、4 ビットアクセスを使用しています。

■初期化方法

HD44780 の初期化は、多少癖があり、いくつか、一定時間のウェイトを入れながら処理しなければなりません。初期化ルーチンは、次のようになっています。

LcdInit は、引数の mode が 4 の時、4bit アクセスになるように初期化します。

```
void LcdInit(char mode)
{
	//ポートの初期化
	LCDDIRECTION=LCDWRITE;
	LCDPORT=0;			//0 で初期化
	SetBit(RSPORT,RSBIT,CMD);
	msdelay(15);			//起動後、15ms 待つ
	CmWrite(0x30);			//最初に 8 ビットモードにする
	msdelay(5);			//5ms 待つ
	CmWrite(0x30);			//再度 8 ビットモードにする
	mcsdelay(100);			//100μs 待つ
	CmWrite(0x30);			//再度 8 ビットモードにする
	mcsdelay(100);			//100μs 待つ
	if(mode==4){
		Is4BitMode=true;
		CmWrite(0x20);		/4 ビットモードにする
		mcsdelay(40);
		LcdCmd(0x28);		//4BitMode Set,5*7font,1/16Duty
	}else{
		Is4BitMode=false;
		CmWrite(0x38);		//8BitMode Set,5*7font,1/16Duty
		mcsdelay(40);
	}

	//ここから正規のアクセスになる
	LcdDisplayMode(false,false,false); //表示を OFF にする
	LcdCls();
```

```
        LcdCmd(0x06);       //EntryMode Set。書き込んだ際にアドレスインクリメント、カーソル右移動
        mcsdelay(40);
        LcdDisplayMode(true,true,true);  //表示を ON にする
}
```

※HD44780 の詳しい使い方は、HD44780 のデータシートを参照してください。

■その他の関数

その他、このサンプルでは、次のような関数を用意しています。詳しい動作は、サンプルソースを参照してください。

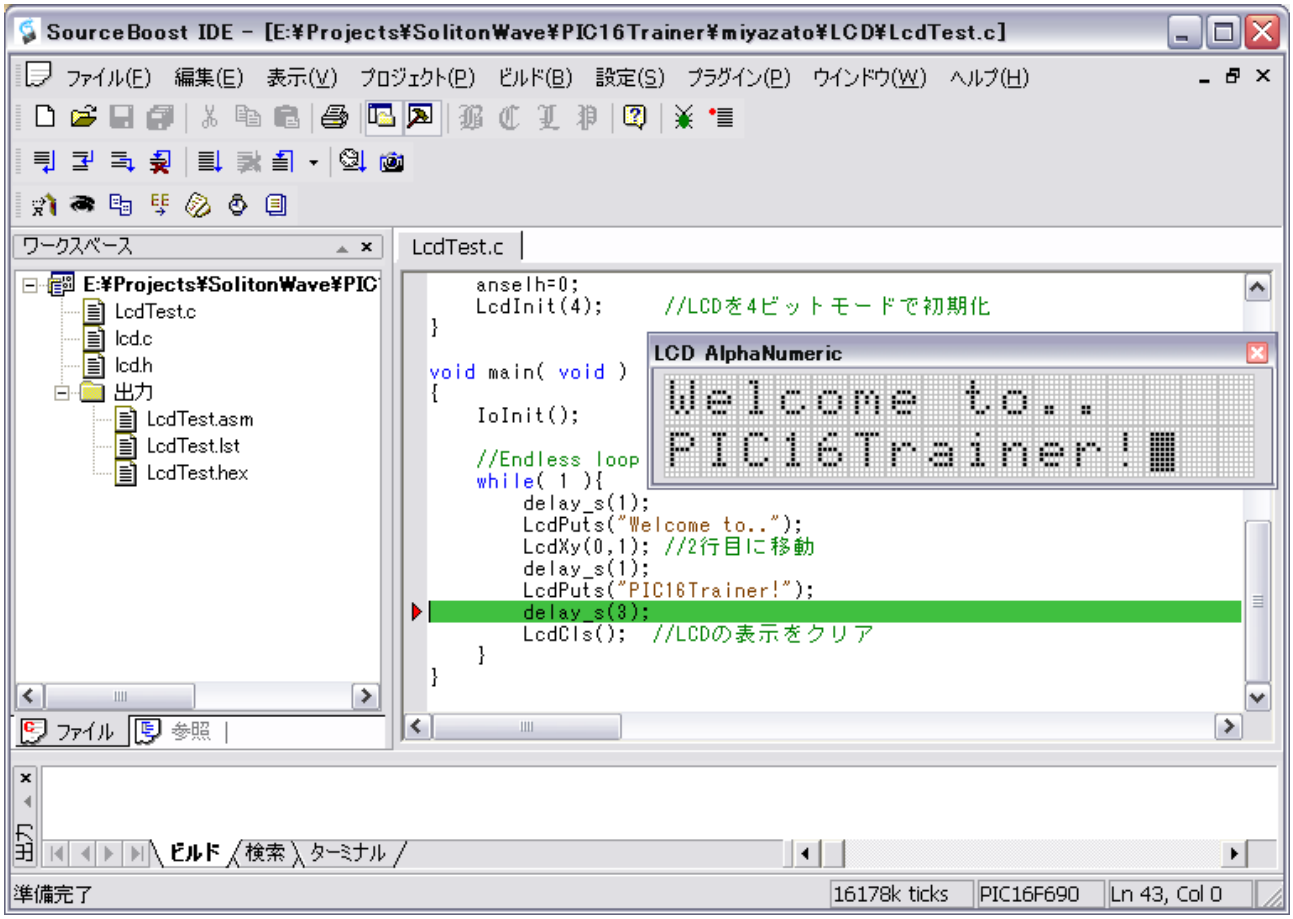
```
/*=========================================================================================
関数名：        LcdCls
機能：                  LCD の表示をクリアする。
=========================================================================================*/
void LcdCls();
/*=========================================================================================
関数名：        LcdDisplayMode
機能：                  LCD の表示モードを設定する
                        disp:           LCD 全体の表示の ON/OFF
                        cursor:         カーソル表示の ON/OFF
                        blink           カーソル位置の文字のブリンクの ON/OFF
=========================================================================================*/
void LcdDisplayMode(char disp, char cursor,char blink);
/*=========================================================================================
関数名：        LcdInit
機能：                  LCD の初期化。起動時に 1 回だけ呼び出す
                                引数の mode==4 の時、4bit アクセス。それ以外は 8bit アクセス
=========================================================================================*/
void LcdInit(char mode);
/*=========================================================================================
関数名：        LcdPutc
機能：                  現在設定されている LCD の DDRAM アドレスへ 1 文字出力する
=========================================================================================*/
void LcdPutc(char c);
/*=========================================================================================
関数名：        LcdPuts
機能：                  現在設定されている LCD の DDRAM アドレスへ文字列を出力する
=========================================================================================*/
void LcdPuts(char *str);
/*=========================================================================================
関数名：        LcdInit
機能：                  DDRAM のアドレスをセットする
                                x:列（0～7）
                                Y:行（0～1）
=========================================================================================*/
void LcdXy(char x,char y);

/*=========================================================================================
関数名：        LcdWaitReady
機能：                  BUSY な間ポーリングして待機する
=========================================================================================*/
LCDWaitReady();
/*=========================================================================================
関数名：        LcdPuts
機能：                  引数 x,y で示された位置へ文字列 s を表示する
=========================================================================================*/
LcdOutXy(x,y,s);
```

■サンプルの実行

サンプルプログラムを実行すると、LCD モジュールに“Welcome to..PIC16Trainer!”という文字列が 2 行に渡って表示されます。

ソースブーストには、別売の拡張プラグインがあり、この中に、LCD の仮想デバイス・モジュールがあります。このプラグインを使用すると、LCD モジュールを使った回路のデバッグも、PC 上で容易に行う事ができます。
次の図は、今回のサンプルを、仮想デバイスで表示している様子です。



ストップウォッチの製作

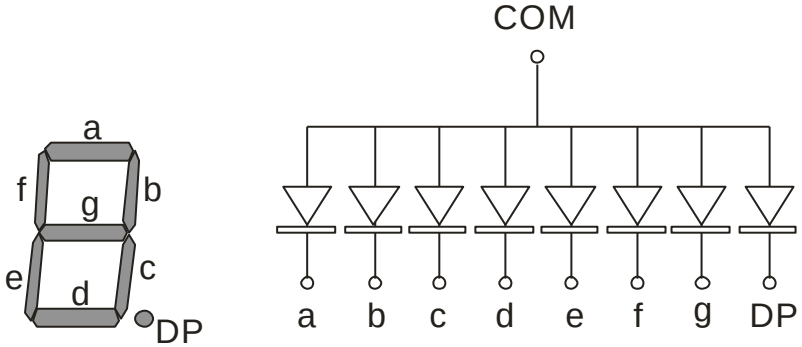
■概要
7 セグ-LED を使って、ストップウォッチを作成する

■サンプルプロジェクトの格納フォルダ
StopWatch

■ポイント
・ タイマ 1 の使い方を習得する
・ 7 セグ LED の使い方を習得する

■プログラムの動作
このサンプルは、1/100 秒(10ms)のストップウォッチとして動作します。
最初に起動した時は、表示はリセット状態の”00:00”となっています。表示は、秒：１０ミリ秒という形式です。ここで、黒いスイッチを 1 回押すと、ストップウォッチの計測が開始され、表示がカウントアップして行きます。再度スイッチを押すと、計測は停止し、計測時間が表示されます。もう一度スイッチを押すと、リセットされ、最初の”00:00”の状態に戻ります。
このように、このプログラムには、３つの状態があり、リセット状態→計測状態→停止状態の３つが繰り返されます。サンプルでは、この状態を、enum の STWT_STOP, STWT_RUN, STWT_RESET、それらを格納する変数を利用して判断しています。

■7 セグ LED の使い方
７セグ LED 1 つを駆動するには、通常 8 つのデジタル I/O が必要です。このサンプルで使用する 7 セグ LED は 4 つです。つまり、8×4＝３２個の I/O が必要ということになります。しかし、PIC690 ではそれほど多くの I/O を持っていません。よって、今回はダイナミック点灯という手法を使って、7 セグ LED4 つを 12 本の I/O で行っています。この手法は端的に言うと、４つの 7 セグ LED を高速で 1 つずつ切り替えて表示するという手法です。切り替えの間隔は 5ms なので、人間の目には同時に映っているように見えます。
次の図は、7 セグメントの LED の内部構造です。



この図からわかるように、1 つの 7 セグ LED は内部で 8 つの LED から構成されています。従って、1 桁の 7 セグ LED を駆動するには LED8 つ分の電流が必要ということになりますが、これは PIC のドライブ能力を超えていますので、PIC16 トレーナーでは、トランジスタアレイを用いて各 7 セグ LED の COM 側を駆動しています。

■タイマと割り込み

PIC のタイマモジュールはハードウェアで構成されたカウンタで、プログラムの動きに関係なく、クロックにあわせて一定の間隔でカウントアップします。よって、ストップウォッチの様に正確なクロックが必要な場合には、タイマの利用が適しています。今回はタイマ 1 を利用します。

タイマのクロックとプログラムをつなぐ機能が、割り込みです。割り込みは、プログラムの実行中に、特定のハードウェアのイベントによって呼び出されるサブルーチンです。サブルーチンは実行中のメインルーチンを強制的に一時停止し、サブルーチンを実行します。サブルーチンの実行が終わると、一時停止した場所からメインルーチンが再開されます。今回のプログラムでは、タイマ 1 をカウントアップしていき、最大のカウント数に達すると、割り込みが発生するようにしています。この仕組みを利用することで、ストップウォッチプログラムを構成する 3 つのパート、時間の計測、LED の表示、スイッチの入力を同時進行で実現することができます。

また、割り込みを利用する際の注意点として、割り込みルーチン内に多くの処理を担当させると、時間がかかってしまい、次の割り込みが入って、割り込み処理が正しく動作しない可能性があります。よって割り込みルーチン内は基本的にコンパクトなプログラムが理想的です。

■1/100 秒毎の割り込みの作り方

10msec を経過したら割り込みを発生するように、タイマ 1 のカウンタ値を初期化します。システムクロックは 8MHz で、設定したプリスケーラは 1:1 ですので、カウンタがインクリメントするサイクルは 1/(8MHz * (1/4)) = 0.5usec となります。従って、10msec の間隔をあけるにはゼロクリアするまでに 20000 回のカウントが必要となります。タイマ 1 は 16bit のカウンタですので、上位 8bit、下位 8bit のそれぞれに初期値を設定します。設定値は、20000 回目のカウントで、タイマの値が、FFFFｈから 0000h になる値ですので、65536(10000h)-20000=B1E0h ということになります。

```
TMR1H = B1h                 // Timer1 の初期値を B1E0h に設定
TMR1L = E0h                 // B1E0h = (65536-20000)
```

ただし、実際には、タイマの値を更新している間でも、タイマカウンタが進んでしまうため、この値は、多少の微調整が必要です。

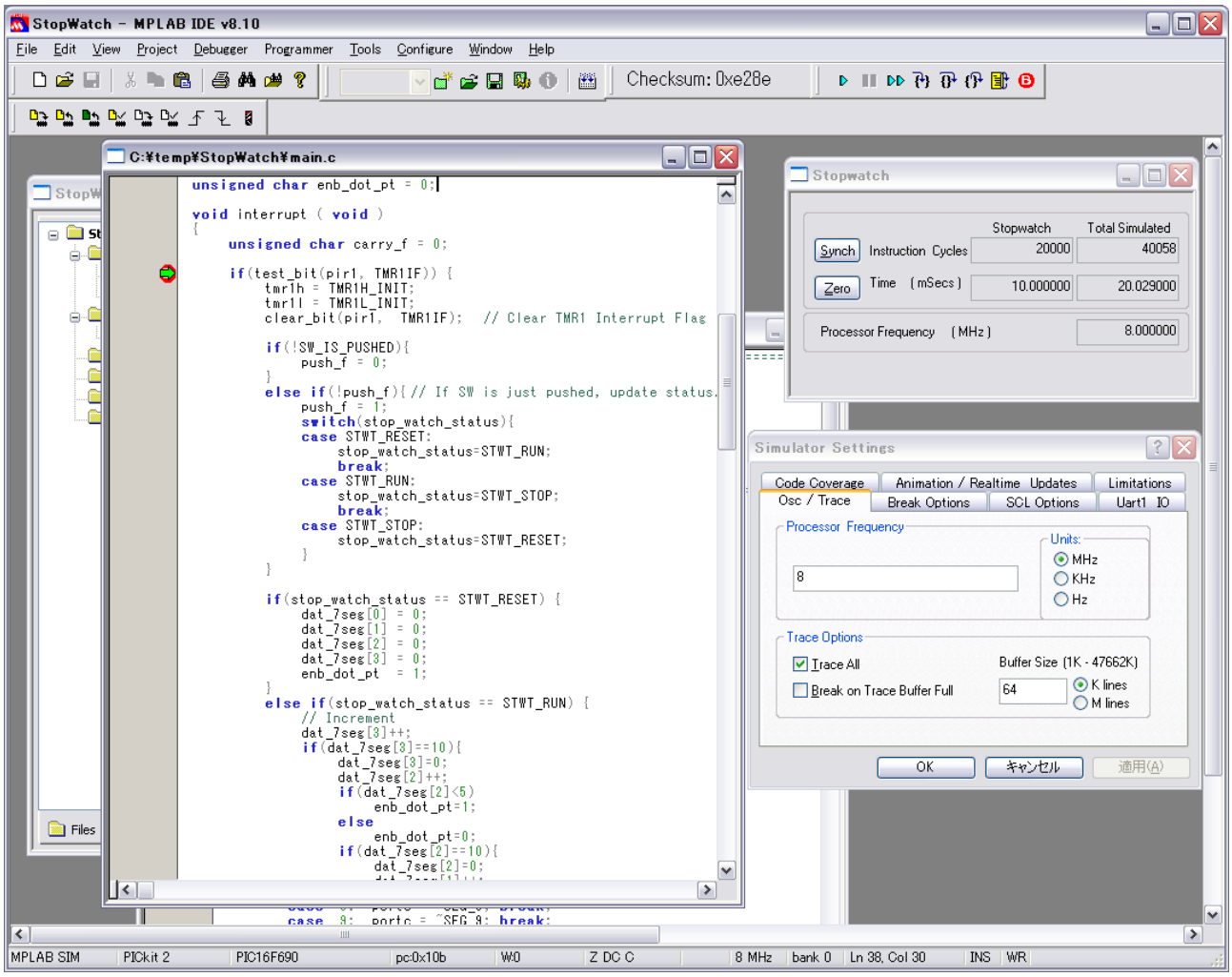
■誤差の修正

タイマを再初期化する際の設定値は、65536-20000 ですが、実際には、値がタイマに再設定されるまで、若干の遅れが発生するため、微調整が必要です。今回作成したプログラムでは、

```
#define WAIT_CYCLE (20000-24)
```

という箇所で、微調整を行っています。

この値を、正確に設定するには、MPLAB に付属の、ソフトウェアシミュレータを使うと便利です。ソフトウェアシミュレータは、PIC 用のソフトウェアによるシミュレータで、PC 上で、プログラムソースのシミュレーションやデバッグが行えます。以下はその際の MPLAB 画像と、手順です。



- MPLAB でストップウォッチのプロジェクトをコンパイルする
- Debugger メニューから、Select Tool で、MPLAB SIM を選択する
- main.c のソースを開き、割り込み処理ルーチンの最初の if 分の行をダブルクリックすると、ブレークポイントを示す、赤い丸が行頭に表示される
- Debugger メニューから、Settings...を選択し、Osc/Trace タブで、周波数を 8MHz に設定する
- Debugger メニューから、Stop Watch を選択する
- Debugger メニューから、Run を選択すると、ブレークポイントのところで、一旦プログラムが停止する
- Stopwatch のダイアログで、Zero ボタンを押し、再度 Debugger メニューから、Run を選択する
- 再度ブレークポイントのところでプログラムが停止し、この間の経過時間が Stopwatch ダイアログに表示される
- ダイアログの表示を見て、Time(mSecs)の値が 10ms になるように、数値を調整する

Stopwatch ダイアログの表示は、Instruction Cycles と Time の 2 つが表示されいます。Time(mSecs)の値がちょうど 10ms になると、調整が完了です。Instruction Cycles には、実際にかかった命令サイクル数が表示されているので、この値を見て、数値を調整します。たとえば、この値が 20002 となっていると、2 インストラクションサイクルだけ、余計にかかっていることになりますので、タイマの再初期化の値を、2 だけ減らすと、正しく設定されることになります。

■サンプルの実行

サンプルプログラムを書き込んで、実行してみてください。LCD モジュールのテストと同様、最初に表示が正しくでない場合は、リセットボタンを押してみてください。