平成 24 年度文部科学省委託
「成長分野等における中核的専門人材養成の戦略的推進事業」

スマートグリッドエンジニア育成教材

環境・エネルギー分野のスマートグリッドエンジニア育成の調査研究プロジェクト

はじめに

この研究報告書は、平成２４年度に実施した、『環境・エネルギー分野のスマートグリッドエンジニア育成の調査研究プロジェクト』の中で、教材開発を行った研究成果をまとめた報告です。

環境に優しい、再生可能エネルギーを効率的に利用する事は、現代の大きな命題であり、そこで活躍するＩＴエンジニアを育成することは、情報系専門学校にも大変意義ある目標となっています。この研究では、身近な再生可能エネルギーを利用して、以下の事項に関連づけて、情報系ＩＴエンジニアに必要と思われる技術を学ぶことができる教材の開発を行いました。

◇ 発電（太陽光）

◇ 充電

◇ 通信

◇ 計測

◇ 制御（放電）

目次

１．再生可能エネルギーの定義と教材の対象

経済産業省・自然エネルギー庁では、次の様に説明しています。

http://www.enecho.meti.go.jp/saiene/renewable/index.html

現在わが国の主要なエネルギー源である石油・石炭などの化石燃料は限りがあるエネルギー資源です。これに対し、太陽光や太陽熱、水力、風力、バイオマス、地熱などのエネルギーは、一度利用しても比較的短期間に再生が可能であり、資源が枯渇しないエネルギーです。これらは、「再生可能エネルギー」ともいわれます。石油等に代わるクリーンなエネルギーとして、政府はさらなる導入・普及を促進します。

自然エネルギー庁 HP

上記で定義される再生可能エネルギーを教材として利用しようとする場合、次の様な条件を考慮する必要があります。

◇ 設備・機器の設置が容易

◇ 小規模導入が可能な事

◇ 実験が容易

◇ 価格が安価

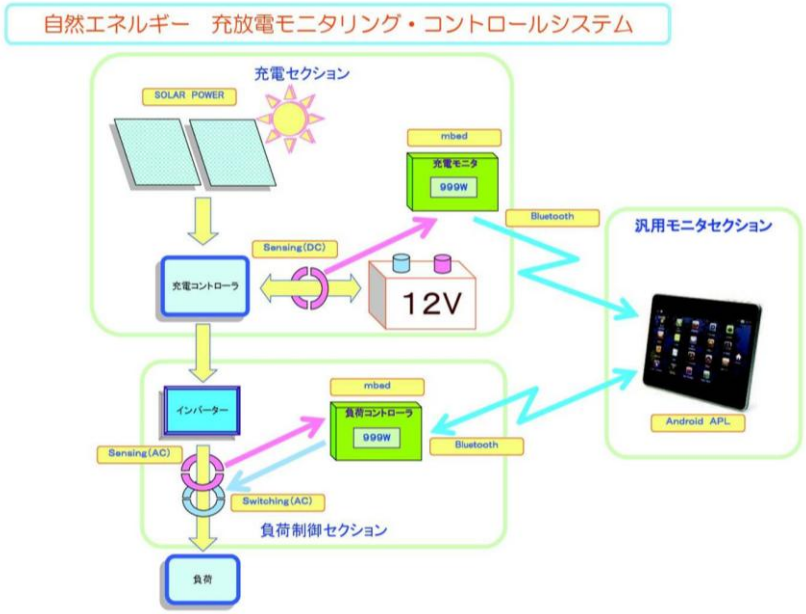
◇ 壊れにくい（機械・機構部分が無い）

身近な再生可能エネルギーを考えると、太陽光、水力、風力エネルギーが手の届く材料として有望ですが、機構部分がなく設置が容易で、手軽に導入でき電気エネルギーに変換しやすい事から、この研究では発電装置に太陽光発電パネルを採用しました。

採用した太陽光パネル

２．教材システムの概要

教材システム開発にあたり、システム全体像を次の様に構想しました。



教材システムの構想

２．１　充電セクション

太陽光発電パネルで発電した電気エネルギーを、充電コントローラを通じてバッテリーに充電します。充電コントローラでは、バッテリーの電圧を測り一定以上の電圧になると充電を停止、一定以下の電圧で充電開始となるよう、マイコンを使用した閾値制御を行います。(充電モニター)

２．２　負荷制御セクション

バッテリーから電力を供給して、インバータを介してAC100V家電品を駆動する制御をおこないます。この際、負荷に流れる交流電流を測定して、使用電力量を計算しモニターに表示します。このセクションもマイコンを使い通信、センシング、表示等をおこないます。

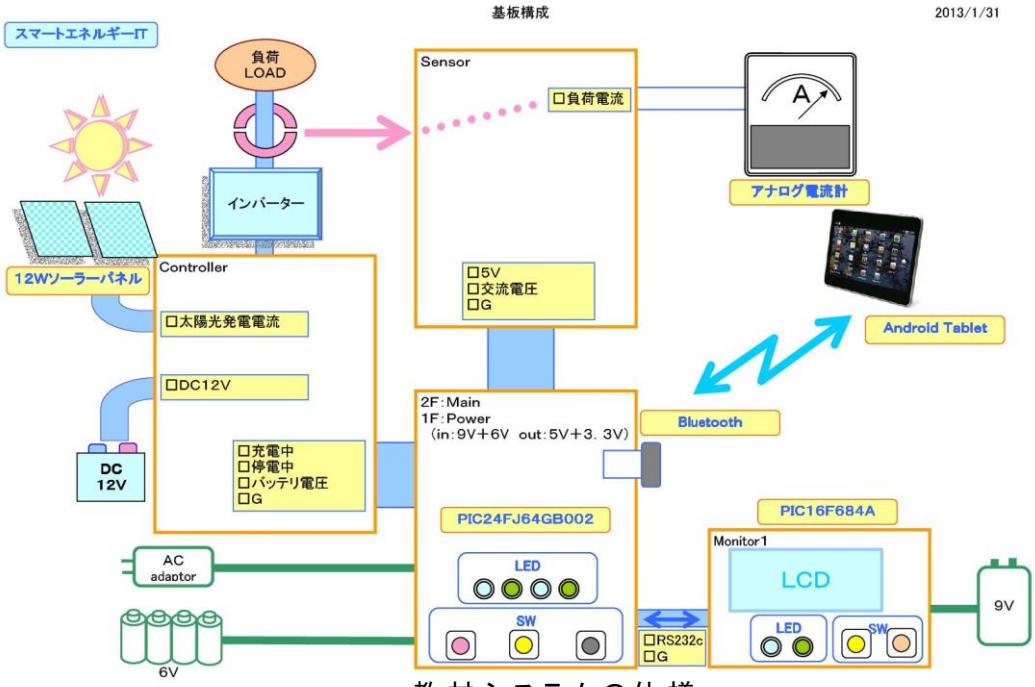
２．３　汎用モニターセクション

充電セクション・負荷制御セクションでモニタリングしている情報を Bluetooth 通信で、タブレット PC に送り、手元でモニタリングが可能とします。

タブレット PC は、OS として Android を使用したものが手元にありましたので、それをそのまま利用して開発することとしました。既に開発済みの Bluetooth 通信プログラムを改造して、今回の目的にかなう汎用的なモニタリングシステムに適応出来る様に考えました。

２．４　最終仕様

上記構想を草案として、IT 教材としての検討を行い、最終的なシステム仕様を決定しました。



教材システムの仕様

当初構想では、スマートメータを意識してモニタリングを主としたシステム開発の計画でしたが、教材として検討した結果、次に示す様

な基板構成としました。大きな特徴としては、

◇ Main ボードは、Power ボードの上段に配置しいくつかの SW により操作ができます。

◇ Power ボードは、AC アダプタとバックアップ乾電池の 2Way 電源とし、AC 電源が途絶える停電を検出する機能を持ち、乾電池バックアップに切り替わり、自動的にバッテリーから負荷に通電する。

◇ 汎用モニターがない場合や Bluetooth 通信が出来ないときでも、簡易モニターで操作ができるようにしました。当初の構想では Bluetooth の通信が 2 系統必要でしたが、タブレット PC の仕様からこれは難しいので、Main ボードがまとめて 1ch の Bluetooth 通信を行う事としました。

◇ Main ボードは、Bluetooth 通信と同時に、シリアル通信を行い簡易モニターとの間で情報のやりとりと、簡易モニターの SW による操作指示を Main ボードに指示します。IT 教材としては、Bluetooth とシリアル通信の 2 つの技術を学ぶことが出来ます。

※シリアル通信のケーブル（またはコネクタ）は、PC のシリアルポートに接続して、ターミナルソフトで通信のモニターとテストが出来る様になっており、今回の開発でも、そのようなコネクタを作り、PC で通信確認テストをおこないました。

◇ 汎用モニターは、簡易モニターが無い場合でも、全てのモニタリングと操作が出来るように Android 側のシステムを作り込みます。組込系マイコンシステムでは、小さな表示器（この研究では 8 文字×2 行の液晶表示器）や LED 等による表現方法が採れますが、タブレットの表現機能を使う事で、情報システムとしての機能が向上します。また、一般的に Android タブレットは、JAVA による開発

環境が無料で公開されており、開発環境の構築も容易です。

◇ Main ボード CPU は、PIC24FJ64GB002 を使用して、C 言語により開発します。これは、Bluetooth 通信で使用出来る様々なライブラリ群が CPU の製造メーカ Microchip 社から公開されており、開発環境も MPLAB IDE として無償公開されている環境が使えるためです。インターネット上に情報も多く存在するので、システム開発に不安がありません。

◇ 簡易モニターCPU には、PIC16F684A を用います。開発環境は、Main ボードと同様の環境でも開発出来ますが、情報 IT 系専門学校の教材として位置づける意味から、手元に既にある mikroBasic Pro を使うこととしました。このシステムを開発するにあたり、

①. Basic 言語による組込マイコン開発（簡易モニタ）

②. C 言語による組込マイコン開発（Main ボード）

③. JAVA によるモバイルシステム開発（汎用モニタ）

が学べます。

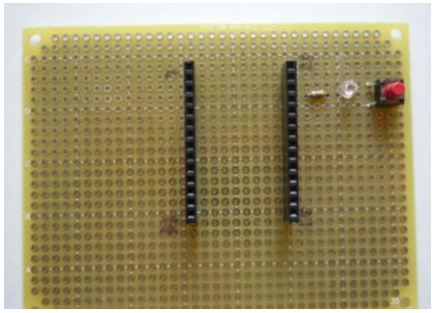
一からシステムを開発しなくても、部分的にプログラムを改造・改善する事で、細かな学習教材として使用する事が出来ます。

◇ PWM 制御の実装

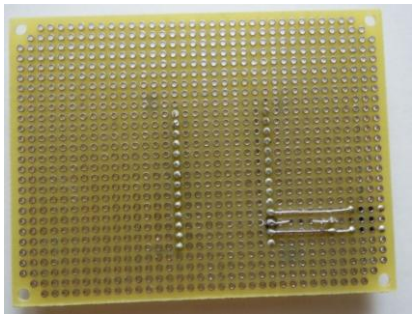
放電制御の部分では、単にバッテリーからインバータに電源供給する事は、スイッチングだけの制御で容易過ぎておもしろみに掛けます。スマートグリッドエンジニアに不可欠の技術として、Main ボード CPU のタイマー機能をフルに使った PWM 制御を実装しました。DUTY 比 0～95%の範囲で切換が可能で、0～80%は 10%刻み、80～95%は 1%刻みで制御実験が行えるようプログラムの設計をしました。

３．教材システム開発（ハードウエア）

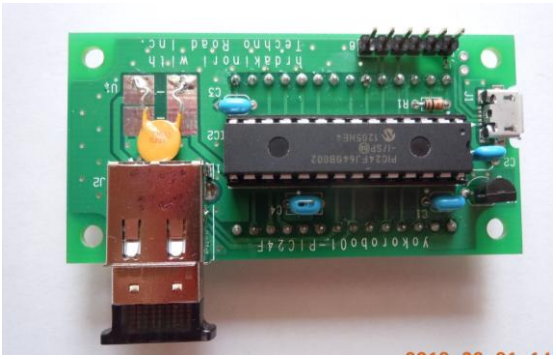
このシステムでは、Bluetooth 通信を行う事が、汎用モニタ・Main ボード CPU の大きな命題です。システムを開発するにあたり、既に開発済みのマイコン基板と、手持ちのタブレット PC、Bluetooth ドングルについて、Google 社の Android ライブラリ、マイクロチップ社の Bluetooth 通信ライブラリが使えるかどうか実験を行いました。



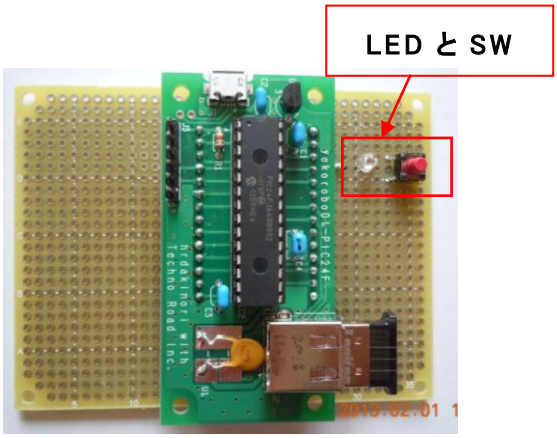
実験基板（表）



実験基板（裏）



実験用 CPU ボード



実験基板（完成）

完成した実験基板に、簡易プログラムを書き込み、Android タブレットも、実験用プログラムを作り、基板上の SW をクリックすると、Android タブレットの画面表示が変化し、タブレット側をタップすると、基板上の LED が点灯する事を確認しました。

この実験で、手持ちの Bluetooth ドングルとタブレットが使用出来る事が分かりましたので、システムの本格的な開発に入りました。

以下、この研究で開発した教材システムのハードウエアについて報告します。

３．１　Power ボード

◆設計

このシステムでは、マイコン(電源 3.3V:計測用基準電圧を兼ねる)、シリアル通信レベル変換 IC(電源 5V)、ソーラーパネル・インバーター・12V バッテリーの電源系をスイッチングする FET の制御などが必要なために、Power ボードでは 5V・3.3V の DC 電源を作ることとし、周辺基板に電源を供給します。また、停電検出を行う構想があるので、単三乾電池４個（6V）と 9V1.2A の AC アダプタという２WAY 電源としました。

停電検出の機構は簡単で、電池電源と 9V のアダプタ電源を、逆流防止用ダイオードを介して並列に接続します。このとき 9V のアダプタ出力をトランジスタの Base に接続してトランジスタを駆動しておきます。

停電すると、9V と 6V の電圧比較で低い方に電源が切り替わります。同時に、トランジスタの Base 電流が途絶えるので、この出力をマイコンのデジタル入力で読み取ると、通常時は Low レベル(=0)、停電時は High レベル(=1)と読めます。この信号をマイコン側ソフトで常時監視すれば、停電時を検出して、必要な処理を行いつつ、各通信先に伝達出来ます。

Main ボードの面積の関係で、シリアル通信用レベル変換 IC も Power ボードに載せることとしました。このために、Power ボードと Main ボードの間は、6Pin のヘッダ・ソケットで接続し、信号内訳は

①．　GND

②．　5V

③．　3.3V

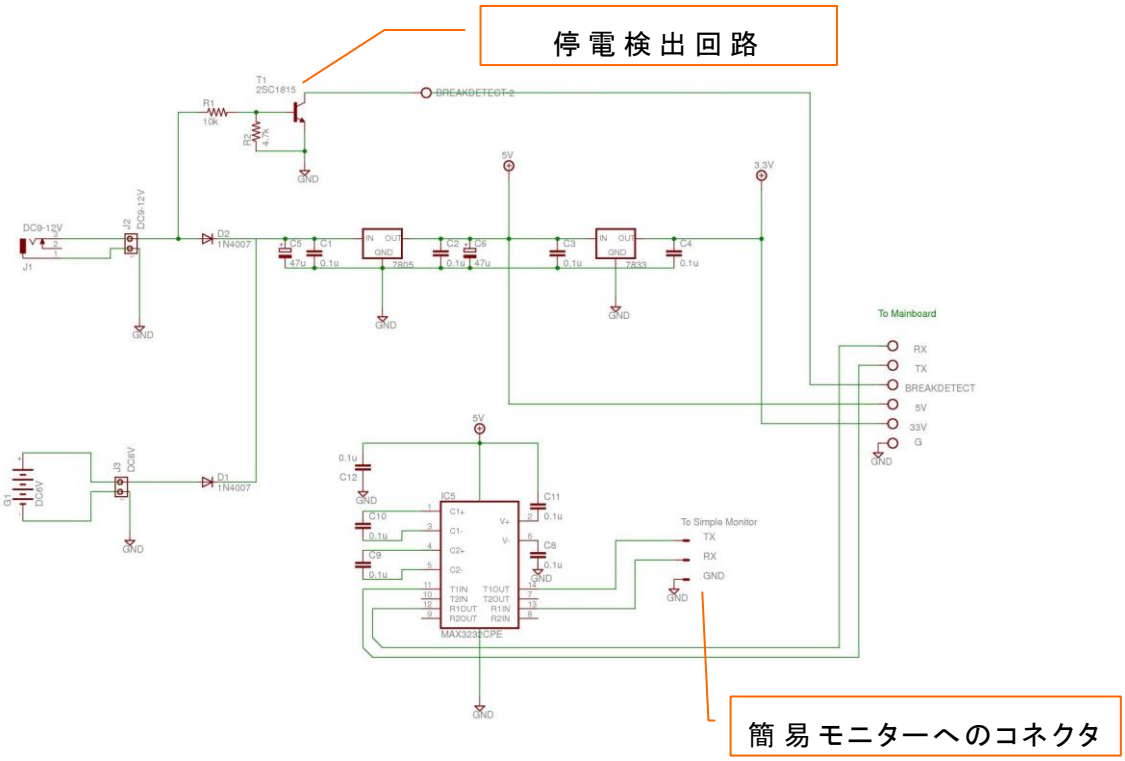
④．　停電検出信号

⑤．　シリアル通信 Tx（送信側）

⑥．　シリアル通信 Rx（受信側）

となりました。

※GND は、全てのソケットなどで位置を統一し、基板に向かって右側端の Pin が GND になるように配置しました。

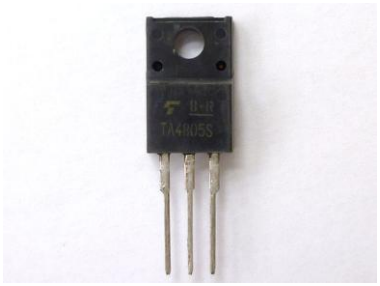


Power ボード回路図

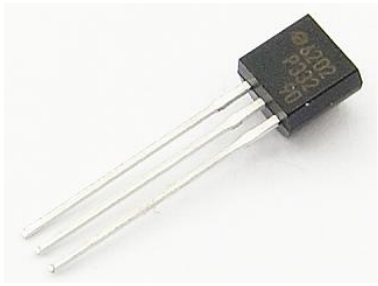
◆製作

Power ボードは、電源を作り出すことが大きな仕事です。基板に中心となる電源 IC を配置して、AC アダプタやバックアップ電池空の電源電圧を安定化させた 5V 電源にします。この 5V を使い 3.3V を作り、Main CPU の主電源とします。5V 電源 IC は、乾電池駆動が可能な定損失レギ

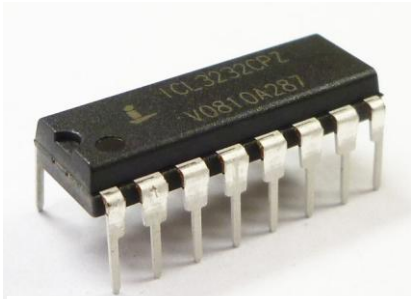
ュレータを使います。



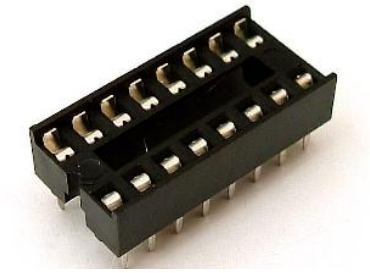
DC5V 電源 IC



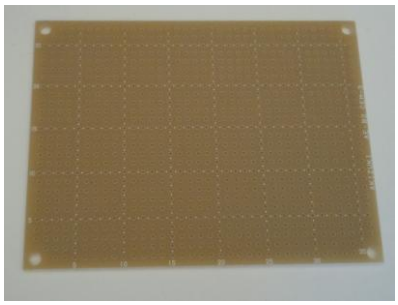
DC3.3V 電源 IC



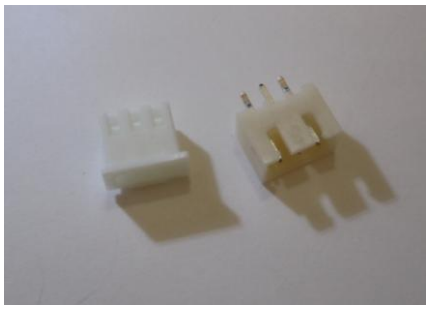
RS232c レベル変換 IC



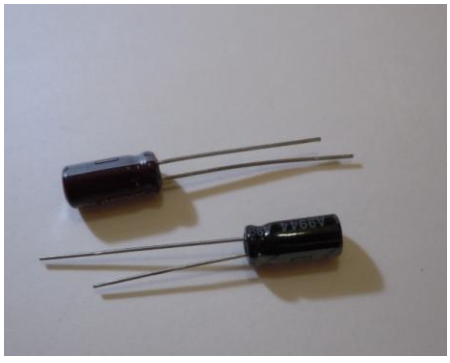
16PIN IC ソケット



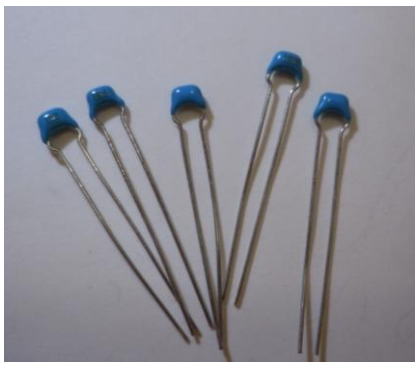
汎用基板



3ポイントコネクタ

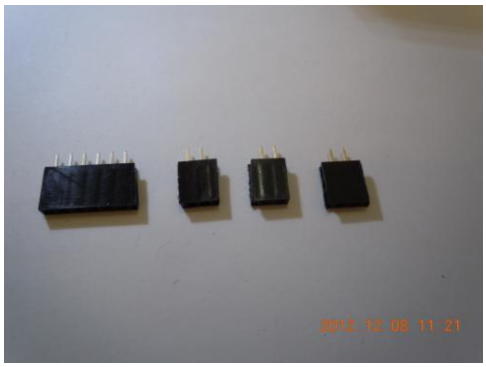


電解コンデンサ

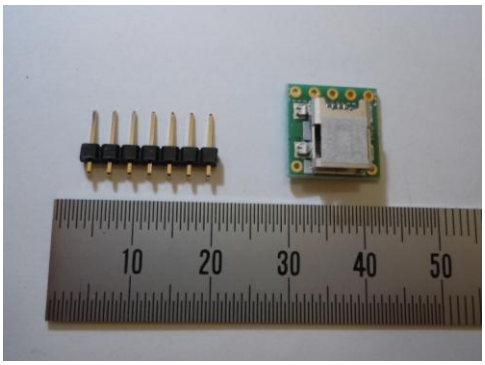


ノイズ対策用セラミックコンデンサ

Power ボードは、上層の Main ボードに電源供給しながら、シリアル通信の送受信信号の受け渡しを行いますので、Pin ヘッダとソケット使い Main ボードと接続します。また、USB ケーブル接続の出来る乾電池ケースを入手したので、miniUSB コネクタの受け側コネクタを、基板に取り付けます。



接続用ピンソケット



miniUSB コネクタとヘッダ

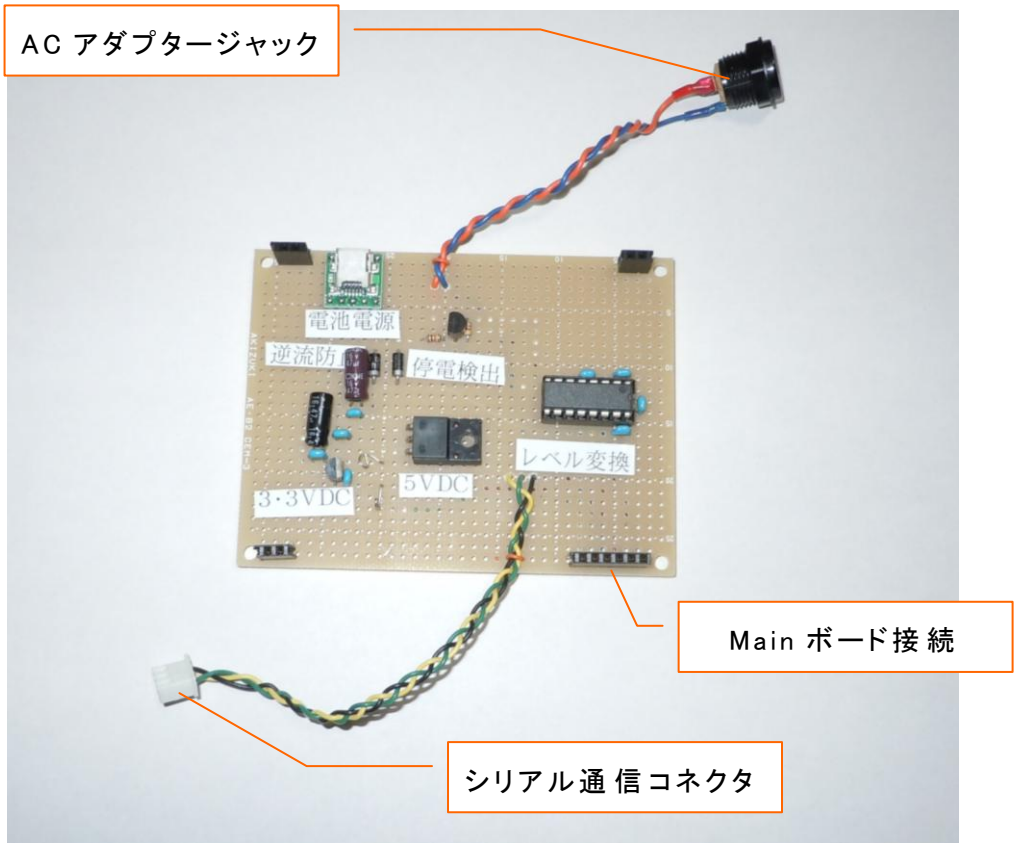
乾電池ケース外観

USB コネクタと SW が見える

miniUSB コネクタ準備

ピンソケットの加工

写真にあるように、パーツを購入しても、そのまま使える物は少ないので、準備の為に半田付けや加工が必要になりますが、安価な教材開発には不可欠な仕事です。これを旨くこなしておくと、後の製作が楽になります。



完成した Power ボード

Power ボードのシリアル通信ケーブルは、三つ編みにしてノイズ低減を図ります。RS232C レベル変換 IC を使用していますので、通信ノイズは低減されますが、それでも安心の為に GND を含めた線を撚り線にしました。実験したところ、この先に 3m ほどの LAN ケーブルを繋いでも問題無く動作しました。

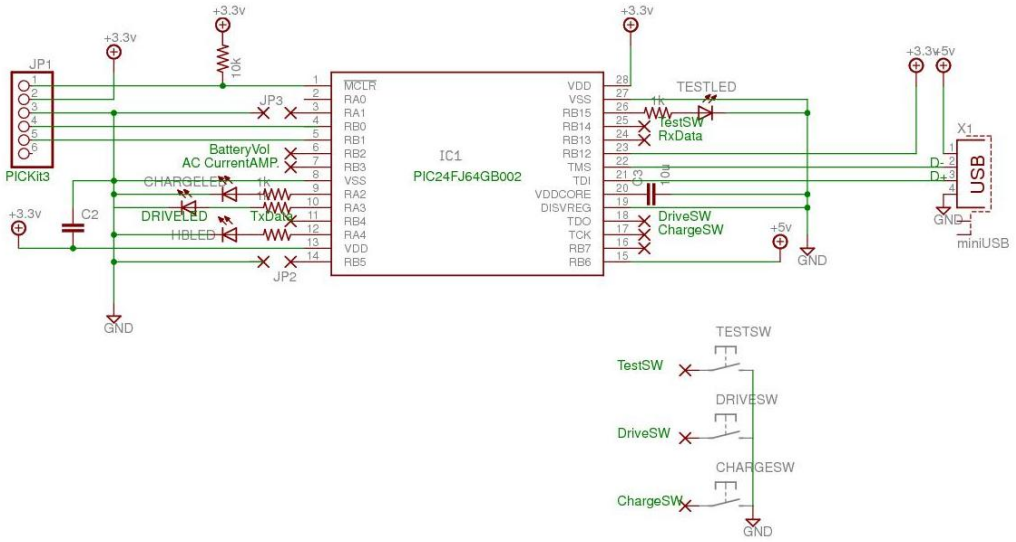
３．２　Main ボード

Main ボードは、このシステムの中枢を担う部分で、16bit マイコン（PIC24FJ64GB002）を用いました。この CPU は、USB コントローラを内蔵していて、CPU メーカから USB に装着して使用する予定の Bluetooth 用ライブラリなども豊富に提供されています。

◆設計

Main ボードは、外部との接続は、下層の Power ボードとのピンヘッダ・ソケットによる接続だけです。主要な部分は CPU なのでこれを中央に配置し、ソケットで抜き差し出来る様にします。プログラムの書き込み用に『ICSP』という I/F が内蔵されているので、CPU を基板に装着したまま内蔵フラッシュメモリのプログラムを書き換えることが出来ます。これを使う為に、ピンヘッダをつけておきます。（回路図上の PICKit3）

単体でテストが出来る様に、Bluetooth テスト用、Drive テスト（放電）用、Charge（充電）テスト用それぞれに３つの SW と LED を配置します。通常稼働時に、CPU の負荷の具合が分かるように、HB（ハートブレーク）LED を点灯します。この LED が長く点灯していると、CPU の処理が忙しいことを示します。



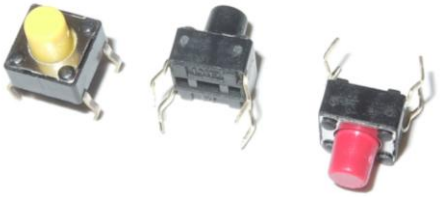
Main ボード回路図

◆製作

Main ボードに必要なパーツを写真に示します。基板取り付け用の USB コネクタが市販の Bluetooth ドングルを差し込むソケットになります。

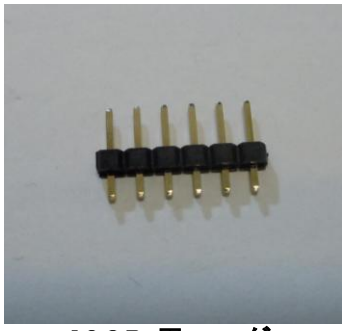
LED と電流制限用抵抗

基板取付用 USB コネクタ

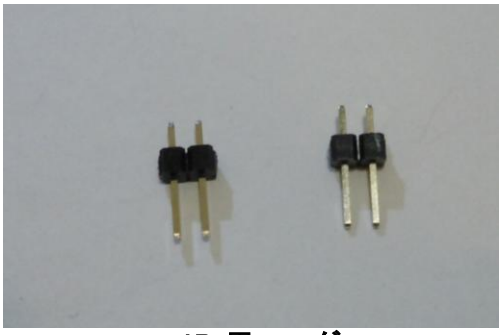


タクト SW

基準電圧調整 VR

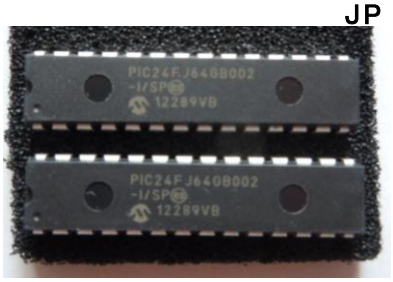


ICSP 用ヘッダ

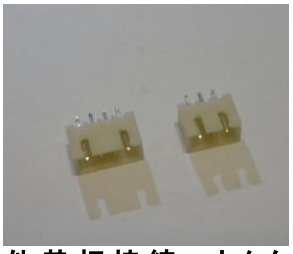


JP 用ヘッダ

CPU 用 24PIN ソケット



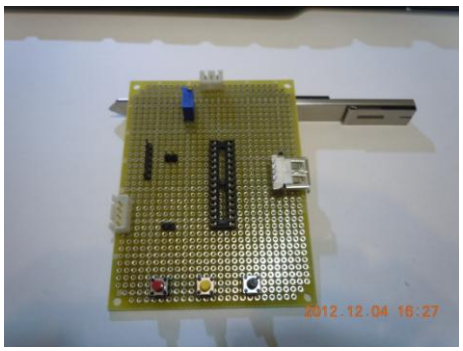
MainCPU



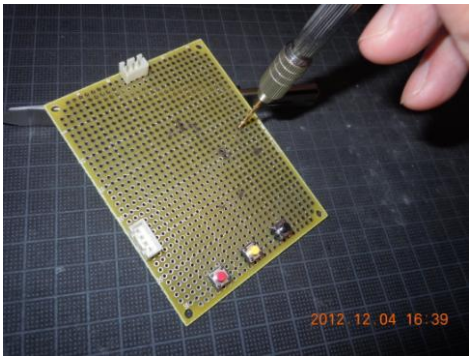
他基板接続コネクタ

始めに大間感パーツを基板に載せて、位置決めを行い、マジックなどでマークします。

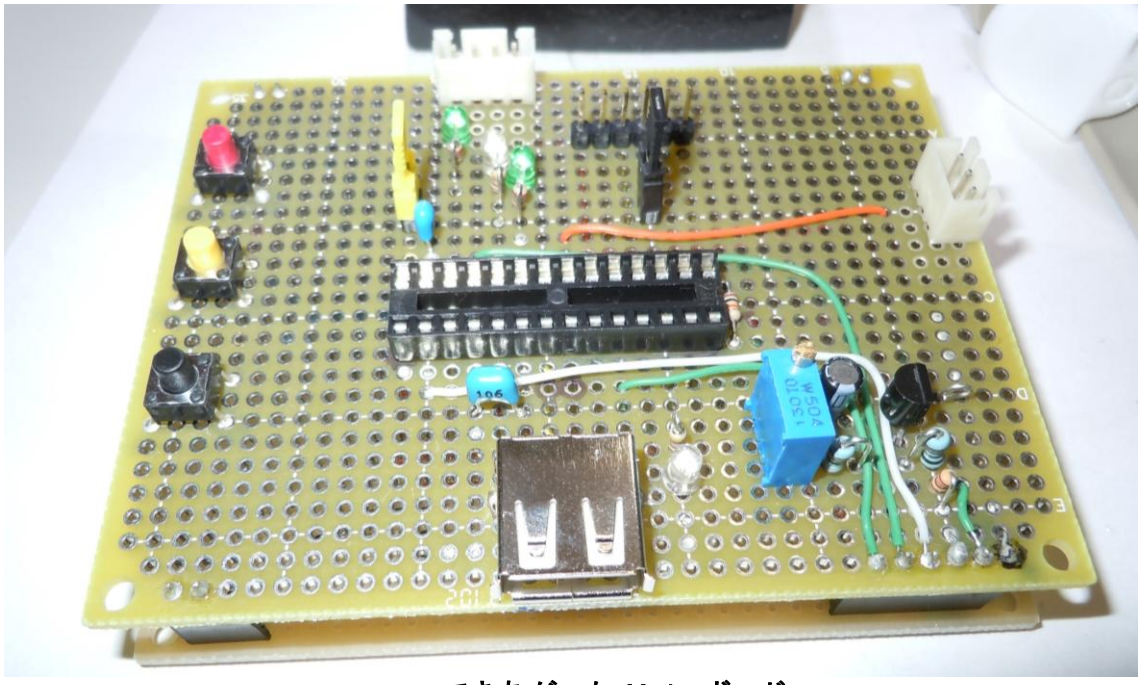
この基板には、USB コネクタがつきますが、固定用のツメが入るように、穴加工を行う必要があります。写真の様なハンドビットと呼ばれる工具とドリルで、穴空け加工します。



パーツの位置決め



穴加工



できあがった Main ボード

上の写真の右下の部分に基準電圧を作る回路がありますが、今回の計測には CPU の電源を基準としたので、未使用となっています。

下層の Power ボードとピンヘッダ・ソケットにより重なっています。

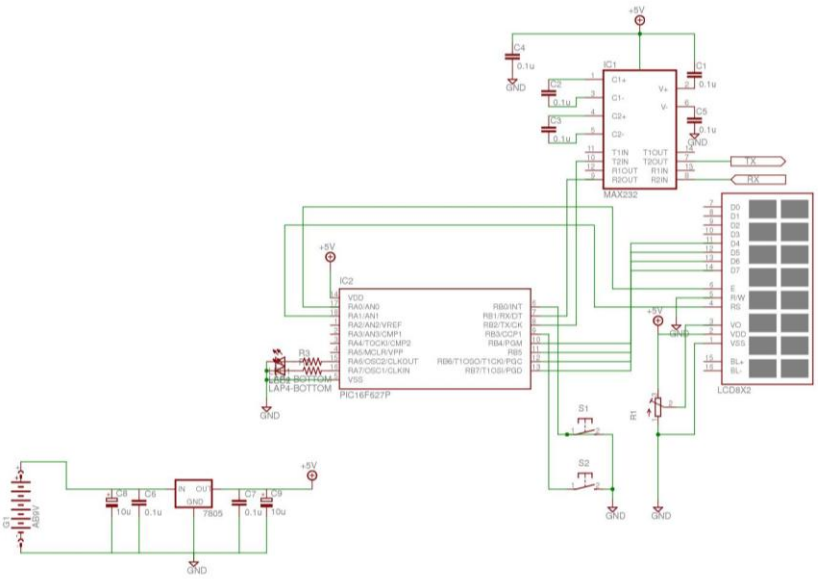
３．３　簡易モニター

簡易モニターは、Main ボード周りとは別電源で単独駆動が出来る様に、006P(9V)乾電池を電源に使います。情報表示器に８文字×2 行のバックライト付液晶表示器を取り付けます。この表示器は白抜き文字（反転文字）がバックライトで浮かび上がる仕組みになっていますので、屋外など明るい場所での文字表示に適していますが、表示文字数が少ないので、ソフトウエア開発時には、少し工夫が必要です。

またこのユニットで使う CPU は PIC16F486 という 8bit マイコンで、シリアル通信ポート（UART）を 1ch 持っていますので、通信時の処理を割り込みで記述することが出来ますので、組込系マイコンシステム開発の良い教材となります。

◆設計

回路図を示します。



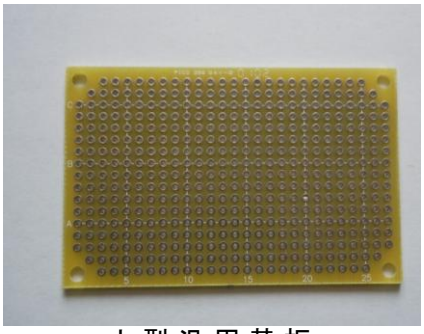
簡易モニター回路図

簡易モニターにも LED×2 個、タクト SW×2 個を取付けて、システム全体の操作ができるようにしています。

特徴として、この CPU はデータ EEPROM を持っていますので、バッテリーの上限電圧・下限電圧の閾値を Main ボードからの指示で書き込み、個別のコンフィギュレーションが出来る様にします。Main ボードで使用している CPU には、このデータ EEPROM の無い安価なモデルを使用していますので、簡易モニター CPU のシリアル通信機能と EEPROM 機能を充分活用した設計になりました。電源は、9V から 5V を作り出す電源 IC を使います。消費電流は CPU が 4MHz 駆動時で 0.9mA、LED１個で数 mA、液晶表示器で 70mA ですので、シリアル通信用レベル変換 IC の電源を取っても充分乾電池駆動できます。

◆製作

使用したパーツを示します。

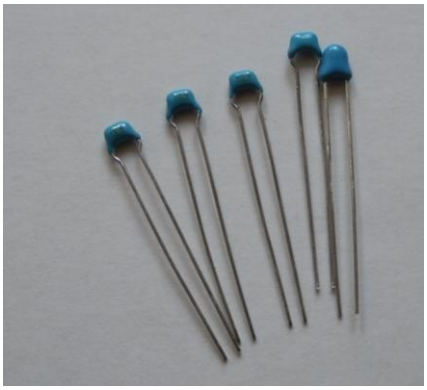


小型汎用基板

006P9V 電池スナップ

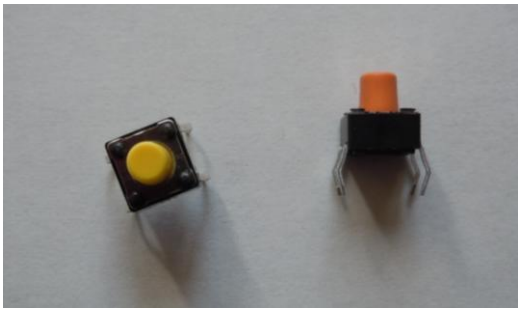
5V 用電源 IC

電源 IC 用電解コンデンサ

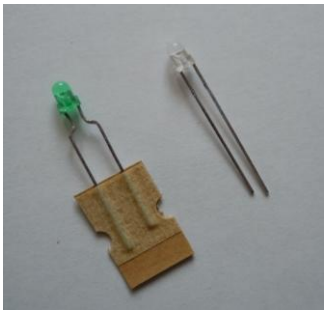


電源・RS232C 用　セラミックコンデンサ

LED 電流制限用カーボン抵抗



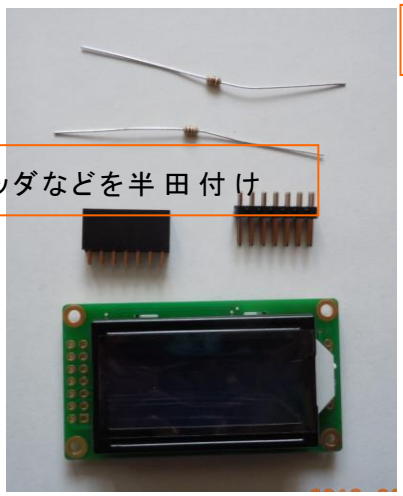
モニタ操作用タクト SW



LED 緑・赤

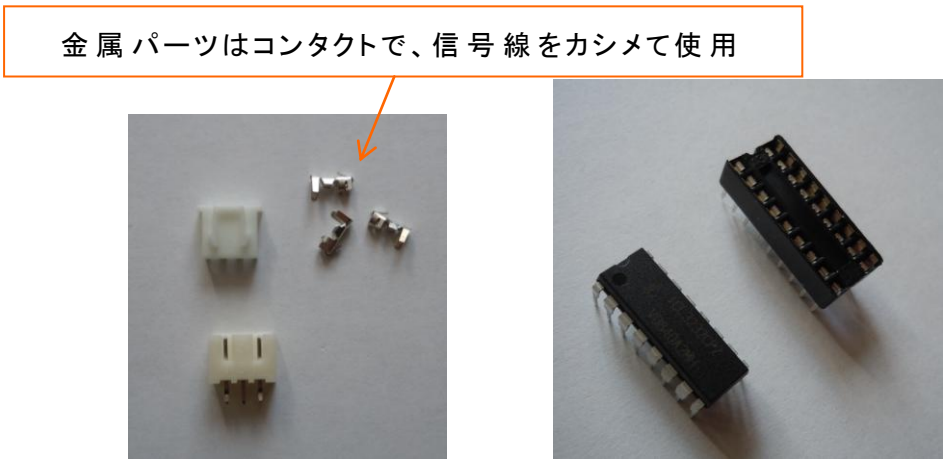
抵抗は、バックライト用

事前にヘッダなどを半田付け

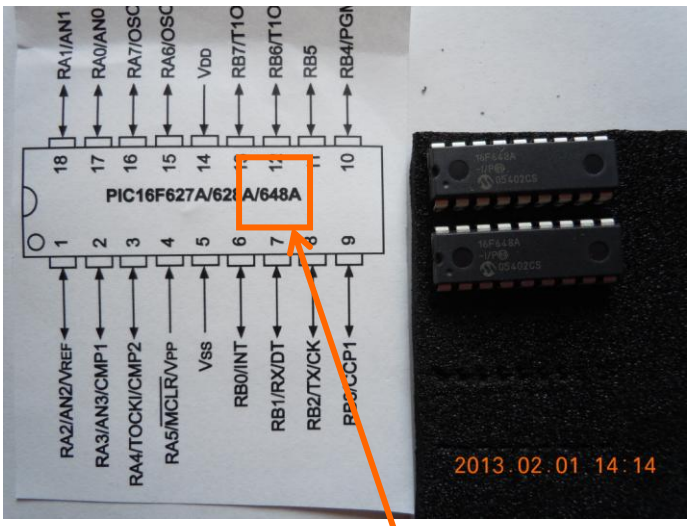


液晶表示器と構成パーツ

液晶表示器コントラスト調整用 VR



通信ケーブル用コネクタ　　RS232C 用 IC とソケット

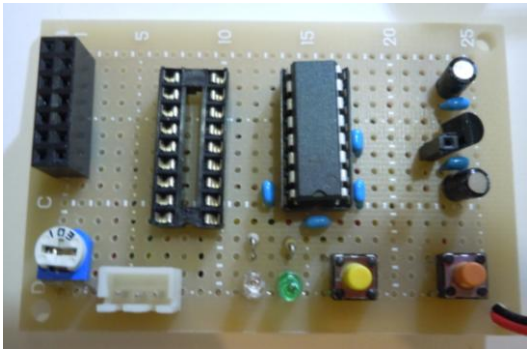


簡易モニター用 CPU 648A

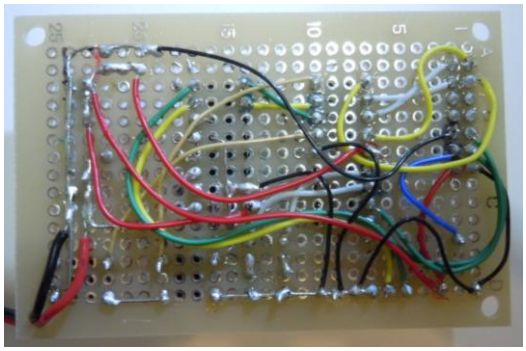
液晶表示器は、裏側にバックライト用の電流制限抵抗と、基板接続用のピンヘッダを半田付けします。基板側には、付属のピンソケットを取り付けます。LED は、電流を沢山流ば明るく光りますが、その分寿命が短くなりますので、電流制限抵抗で必要最小限の明るさにします。

タクト SW で、ON/OFF を判断しますが、使用する CPU の I/O ポートは、内部プルアップが可能ですので、外部にプルアップ抵抗をつける必要はありません。これは Main ボードも同様にしています。

完成した簡易モニターを示します。



簡易モニター（表）



簡易モニター（裏）

液晶表示器（裏）

簡易モニター（完成）

３．４　Controller ボード

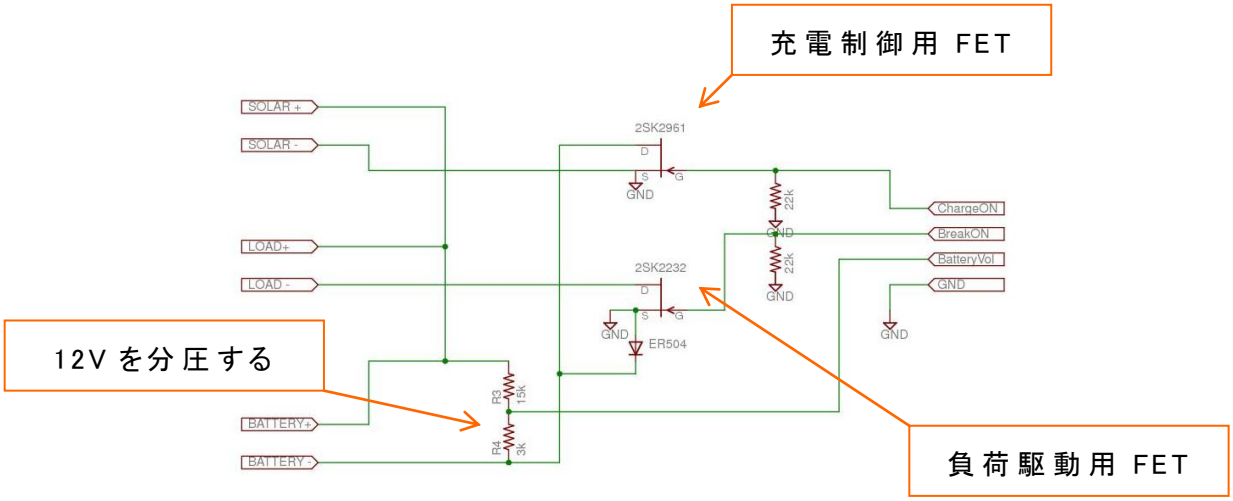
このユニットは、大きな電流が流れますので、電力系統の配線には1.25φの家庭で使う家電品の電源ケーブルと同じ電線を使います。マイコンとの信号線は、通常の細い線で構いません。これまでのユニットに比較するとパーツも少なく作りやすいはずです。

◆設計

電源電圧を基準にして AD 変換してバッテリー電圧を計測するために、12V を分圧して、最大でも 3.3V 未満にする部分を組み込みます。

Main ボードの指示により、充電 ON/OFF、放電 ON/OFF をスイッチングするために、2 つの FET を使います。

設計した回路図は次のとおりです。

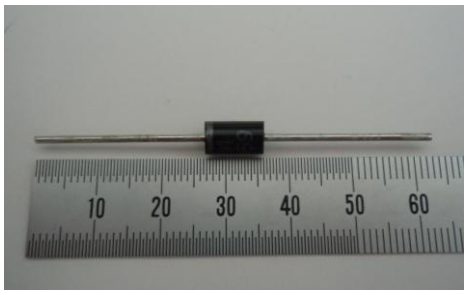


Controller ボード

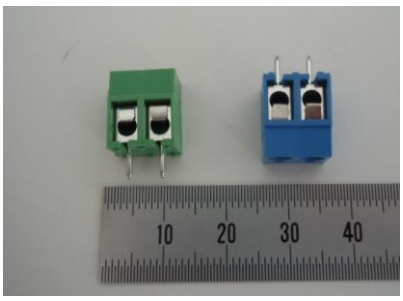
FET の Gate 側に付いている抵抗（22KΩ）は、システムが通電して Main ボードの信号が安定するまでの間に、FET が確実に OFF する様にプルダウンするための抵抗です。このことにより、不安定な状態で不用意に負荷が駆動される事がなくなります。

◆製作

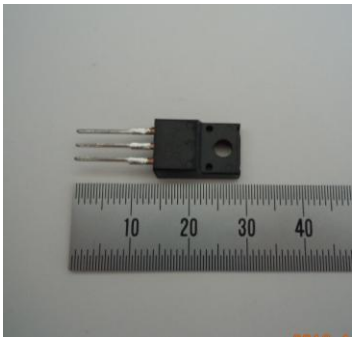
使用したパーツを示します。



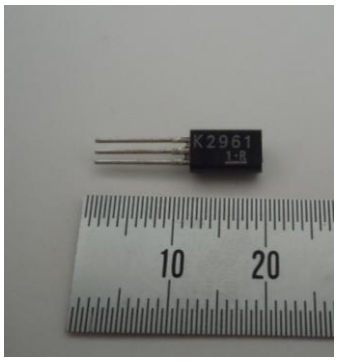
逆流防止ダイオード



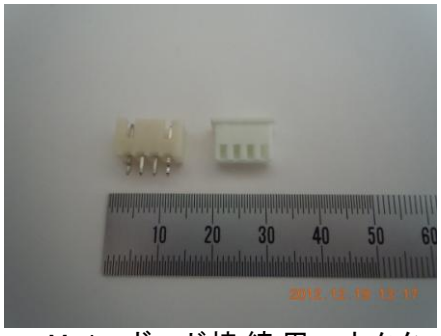
ターミナルブロック



負荷駆動用 FET



充電制御様 FET

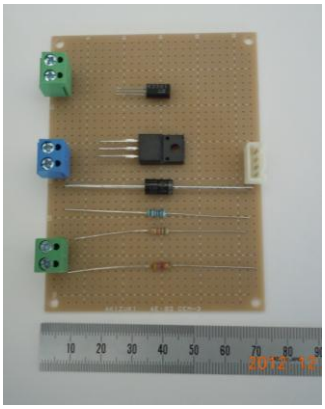


Main ボード接続用コネクタ

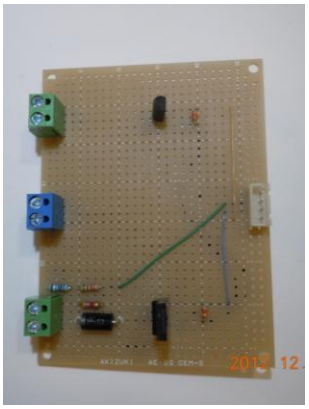
分圧用抵抗

大電流用配線

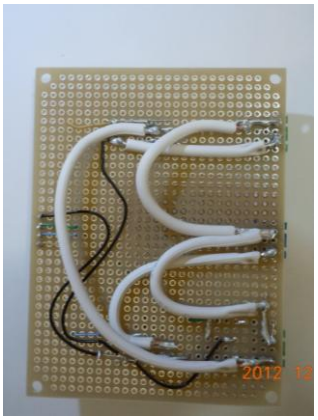
小信号用配線



事前の配置検討



完成した基板（表）



完成した基板（裏）

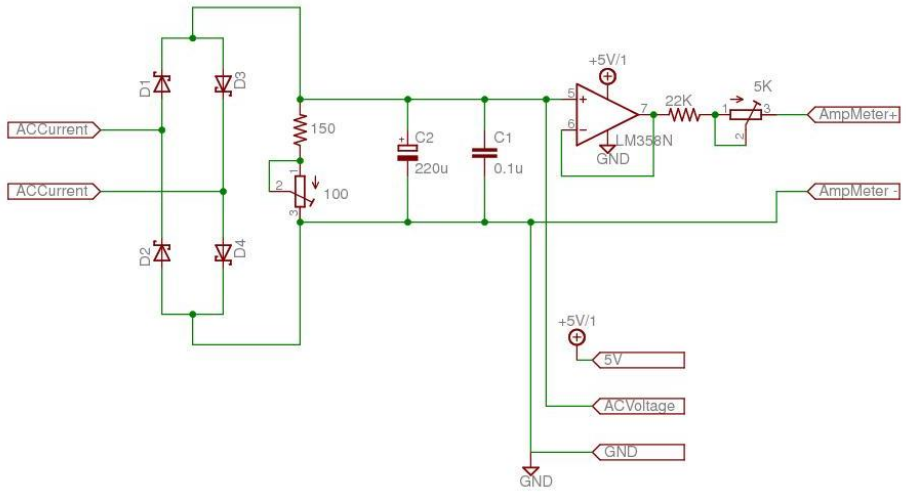
３．５　Sensor ボード

このユニットは、AC 電流センサーで駆動中のインバータの出力電流を計測します。AC 電流センサーで検出した電流に対応する電圧を Main ボードに送り AD 変換して計測します。また、電流センサからの出力電流を整流して DC 電流とし、その電流をオペアンプを介して、アナログ

電流計に送り、メータ表示します。

◆設計

回路図を示します。

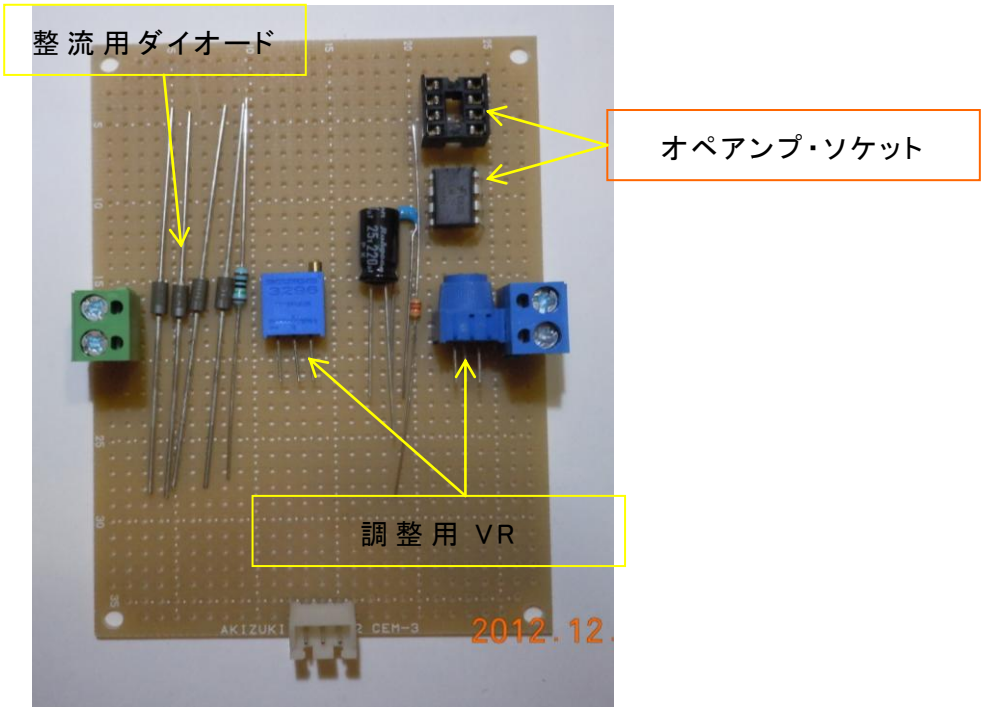


Sensor ボード

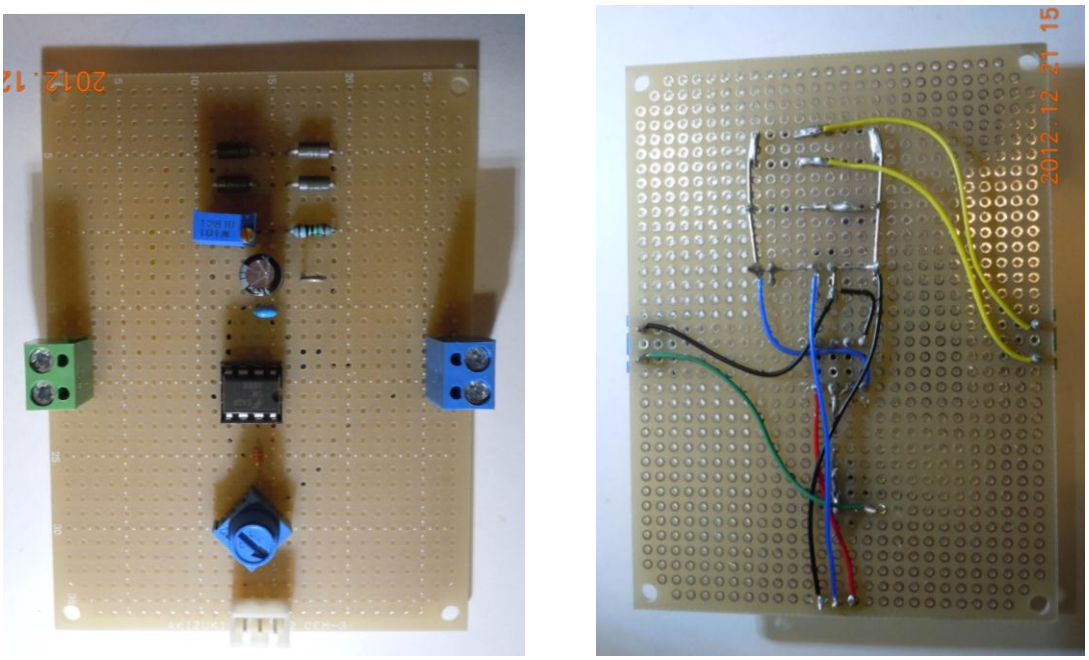
AC 電流センサーにはコイルが内蔵されています。電源線にクランプして、誘導電流による検出を行います。検出される電流は僅かなので、抵抗を介して電圧にして、それを Main ボードの AD 変換ポートに送ります。アナログメータによる表示は、システムとしては特に重要な物ではありません。同じ事が簡易モニターと汎用モニターで表示できます。

◆製作

このユニットで使用するパーツを示します。汎用基板・ターミナルブロック・整流用ダイオード・電圧調整用 VR・電流調整用 VR・コンデンサ・抵抗・オペアンプなどです。



Sensor ボードパーツ



Sensor ボード（完成）

このユニットには、AC 電流センサーとアナログメータが取り付けられますので、これらのパーツも加工します。

電流センサーは、コネクタが付いていますが、それを取り外し、リード

線を延長して、ターミナルブロックに接続出来ように加工します。

その際、リード線の延長部分には、熱収縮チューブなどで、絶縁加工しておきます。

AC 電流センサー

クランプ部分

コネクタをカット

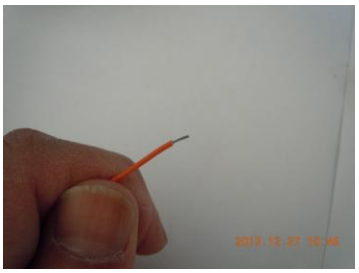
コネクタをカット

絶縁用熱収縮チューブ

延長リード用巻き線

適当な長さを準備

被覆除去



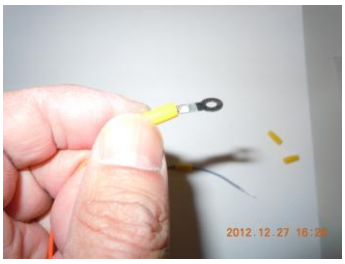
より線ハンダメッキ

絶縁チューブ加工

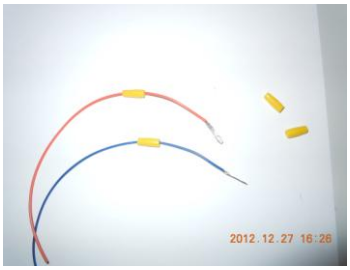
対にして固定

アナログメータ（表）

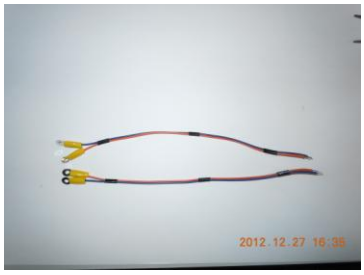
アナログメータ（裏）



メータリードの製作



メータリードの製作



メータリード（完成）

メータリードの取付

このアナログメータは、フルスケール 100μA の物ですが、今回フルスケール 50A として使用しますので、調整用ボリュームでスケールに合せます。メータはカバーが分解できるので、メモリ板を書き換えるなどして、読みやすいようにします。

３．６　その他のパーツ

◆バッテリー用接続リード

Controllボードのターミナルブロックに 12Vバッテリを接続するための、リード線を作成します。鰐口クリップでバッテリに接続出来る様にします。

鰐口クリップを 1.25φの線に半田付けして作成

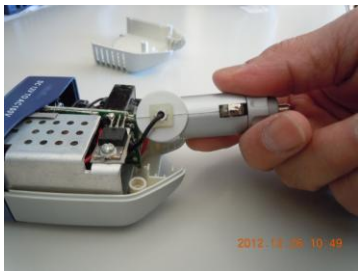
◆インバータの改造

150W 級の自動車用インバータを、Controll ボードに接続出来るようにリード線をとりつけます。


放熱ファンの付いたインバータ

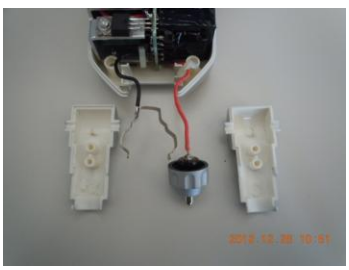


裏側のネジをはずし分解します

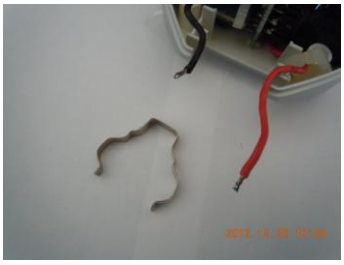


コネクタ部分のストッパーを取り外す





シガーライターソケット部分を分解

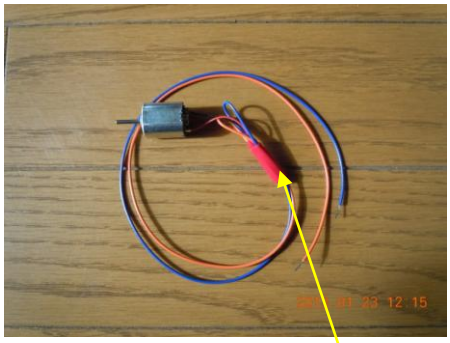




ソケット部分のコネクタを取り外し、リード線を半田付け

ビニールテープで絶縁する

ターミナルブロック側に＋マーク

◆12VDC モータの準備

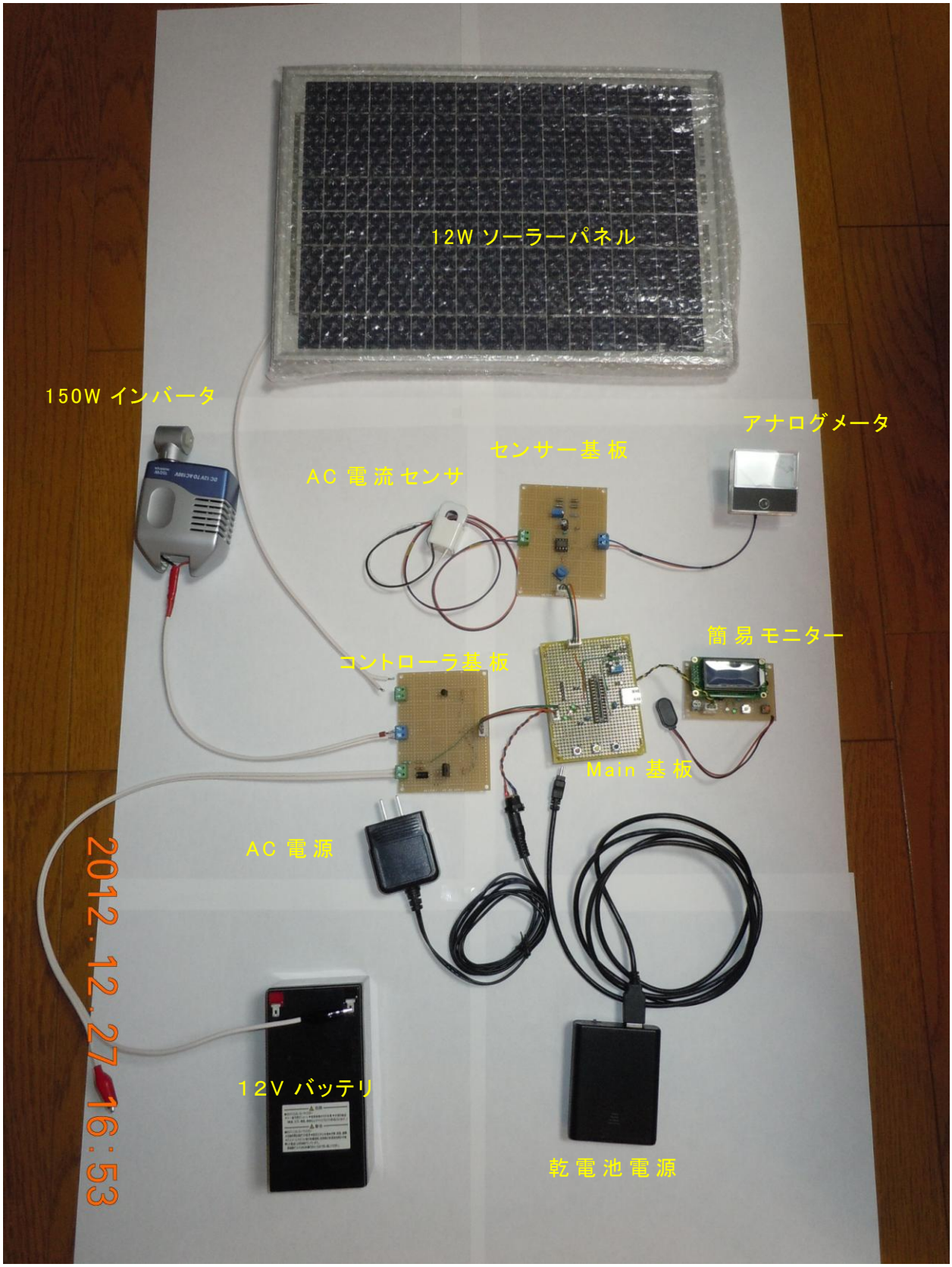


PWM 制御実験用 DC モータのリードを延長加工

◆太陽光発電パネルのリード作成

バッテリリードと同様の物を作成し、太陽光パネルを接続する。

完成した、システム全体（ハードウエア）は、次の様になりました。



システム全体

４．教材システムの開発（ソフトウエア）

４．１　簡易モニター

◆開発環境

簡易モニターの CPU は、米 Microchip 社のマイクロコントローラ【PIC】です。OS は有りませんので、組込システムとしての OS の役目をする main プログラムを何も無い状態から作り込む事になります。

開発言語は、MikroElectronika 社の『mikroBasic Pro for PIC』を使用しました。この環境は IDE（Integrated Development Environment）を含み、使用するマイコンの種類を選択するだけでプログラムの雛形を作ってくれる『フレームワーク』機能があります。基本的には、ユーザーが main 関数を作り込めば良く、言語も理解容易な Basic です。大変多くのライブラリが含まれていて、システム開発の工数削減に大いに役立ちます。筆者の開発デスクには既にこの環境が整った PC がありましたので、これを使いましたが、最新の環境は、MikroElectronika 社の HP からダウンロードできるようになっています。

ダウンロードしたファイルをインストールすれば、プログラムサイズの制限はありますが、重敏な機能の開発環境が整い、Free で使用できます。いろいろな種類の PIC に対応していますので、開発を多く手掛けるのであれば、しばらく Free で使い機能を見定めてから購入するとよいでしょう。IT 系専門学校の教材にはもってこいの環境です。

mikroBasic の HP

HP を開くと、mikroBasic Pro for PIC の画面が表示されています。これを少しスクロールすると、Download 画面がでてきます。

Dowload 画面

Download ボタンをクリックすると、以下の確認ウインドウが表示されますので、保存を選択し保存先のフォルダを適宜選択します。サイズは 26MB ほどです。無線 LAN 環境でも容易にインストール用のファイルを入手できます。

mikroBasic の Download

Download が終了すると、mikrobasic_pro_pic_2012_v580.zip（この報告書作成時の最新版）が保存されていますので、このファイルをダブルクリックしてインストールします。表示されるメッセージに従い進めれば容易に環境が構築出来ます。

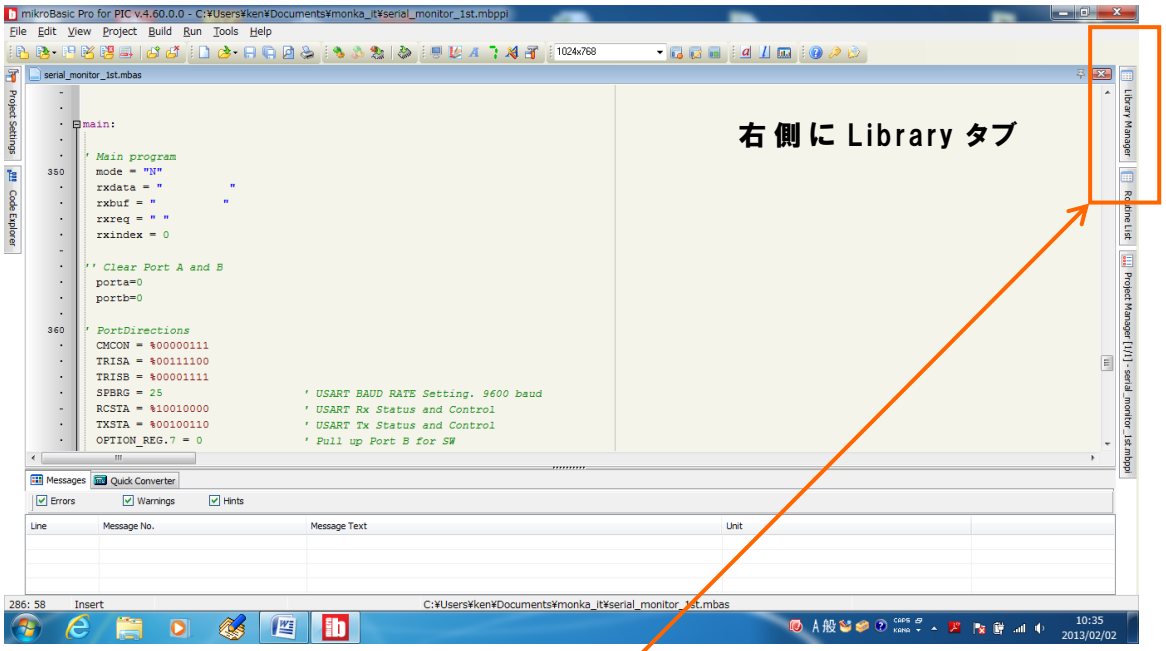
※PC の OS は、Windows Vista と Windows7(64bit)で確認しています。

Free 版で使用するとプログラムサイズに制限がありますが、簡易モニター程度であれば充分開発出来ますし、他のシステム開発にそれほど困ることはありません。大きなサイズのプログラムを作成することになったときに、ライセンス購入すれば良いでしょう。

今回使用した mikroBasic

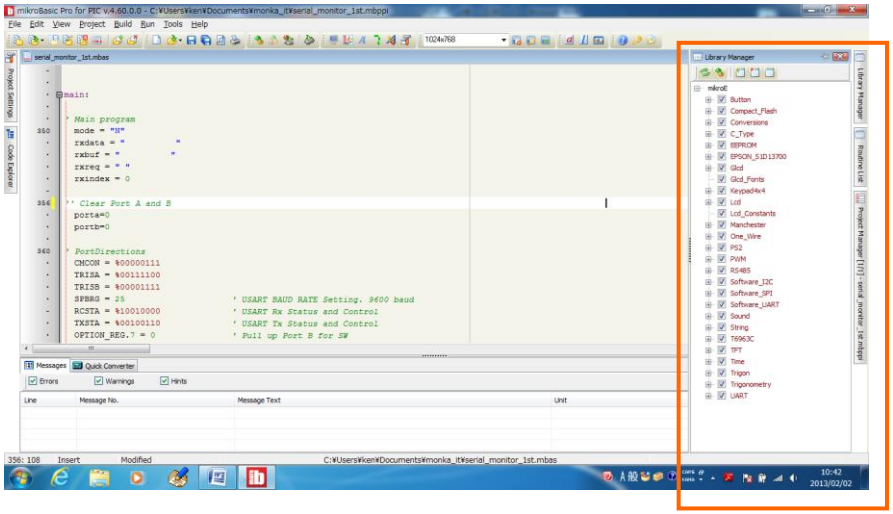
今回使用した環境は、過去の研究事業で実績のある Ver.4.60 の製品版です。使用する PIC も網羅されていて、申し分ありません。

既に開発済みの簡易モニタプロジェクト表示を示します。

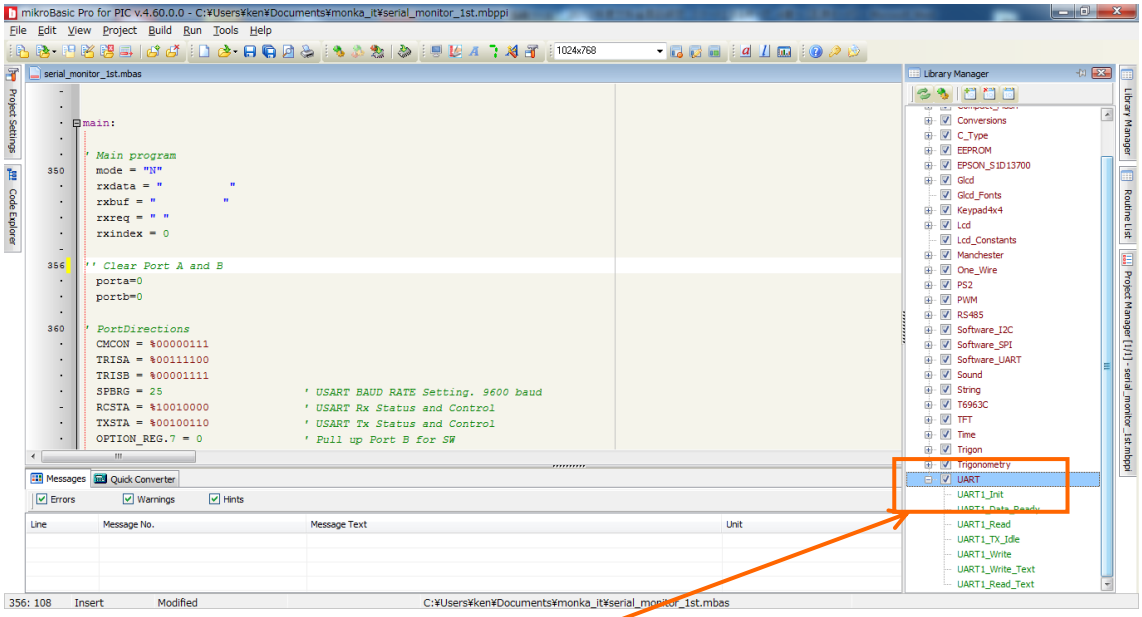


このウインドウの右側に Library Manager というタブがあります。

このタブをクリックするとタブが広がり、含まれている多くのライブラリが確認できます。ADC、CAN 通信、ボタン、LCD、シリアル通信、LCD・・・など、数多くのライブラリが含まれていることが分かります。

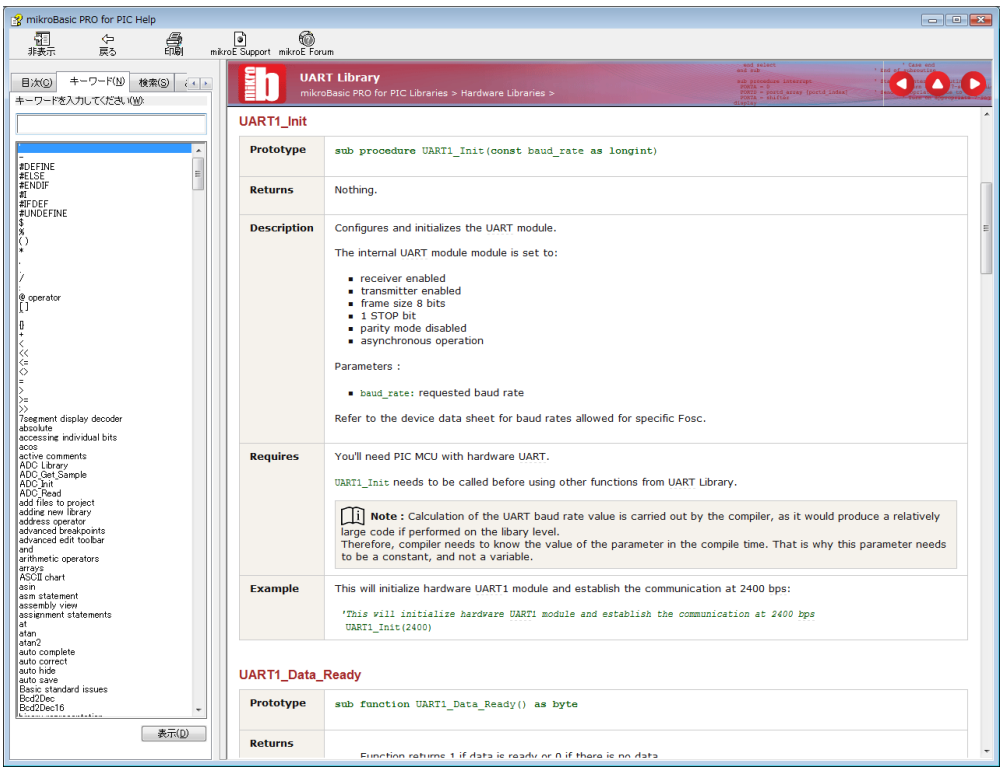


含まれている Library



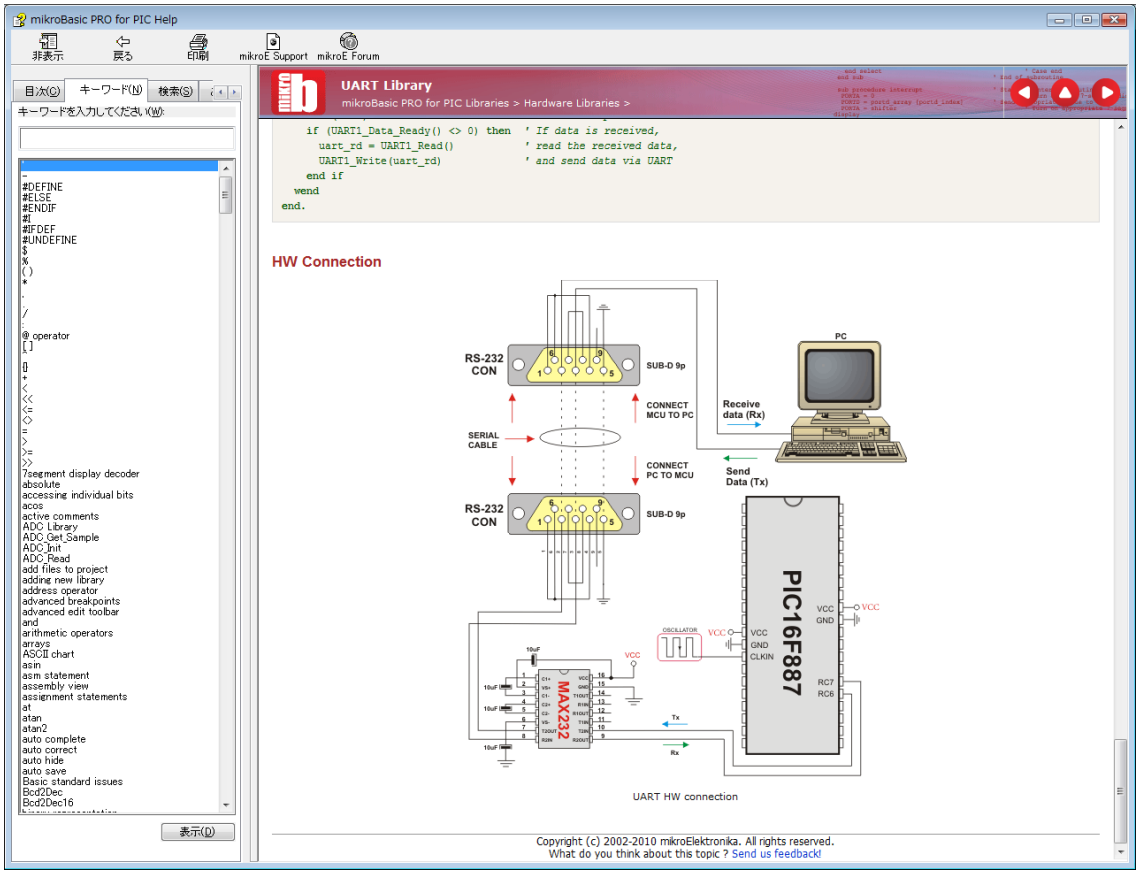
Library の Help

一番下の UART をクリックすると、さらにその関数が表示されます。一番上の UART1_Init 関数をダブルクリックすると、ライブラリ使用の Help が表示されます。



Library の内容例

表示された Help を下にスクロールすると、配線例なども示されていますので、大変参考になります。



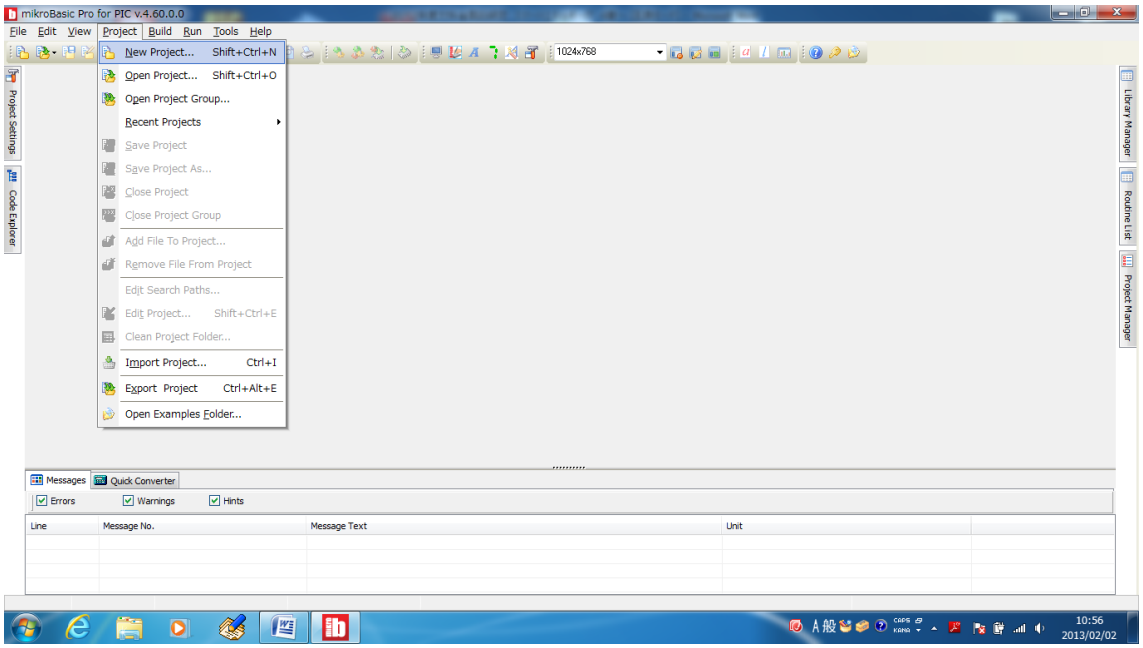
Library に含まれる配線例

情報系 IT 専門学校の学生は、ハードウエアに関する実践知識が不足がちですが、このような Help からも、使用している IC やコネクタのピン配置・接続などが分かるので、とても有意義です。

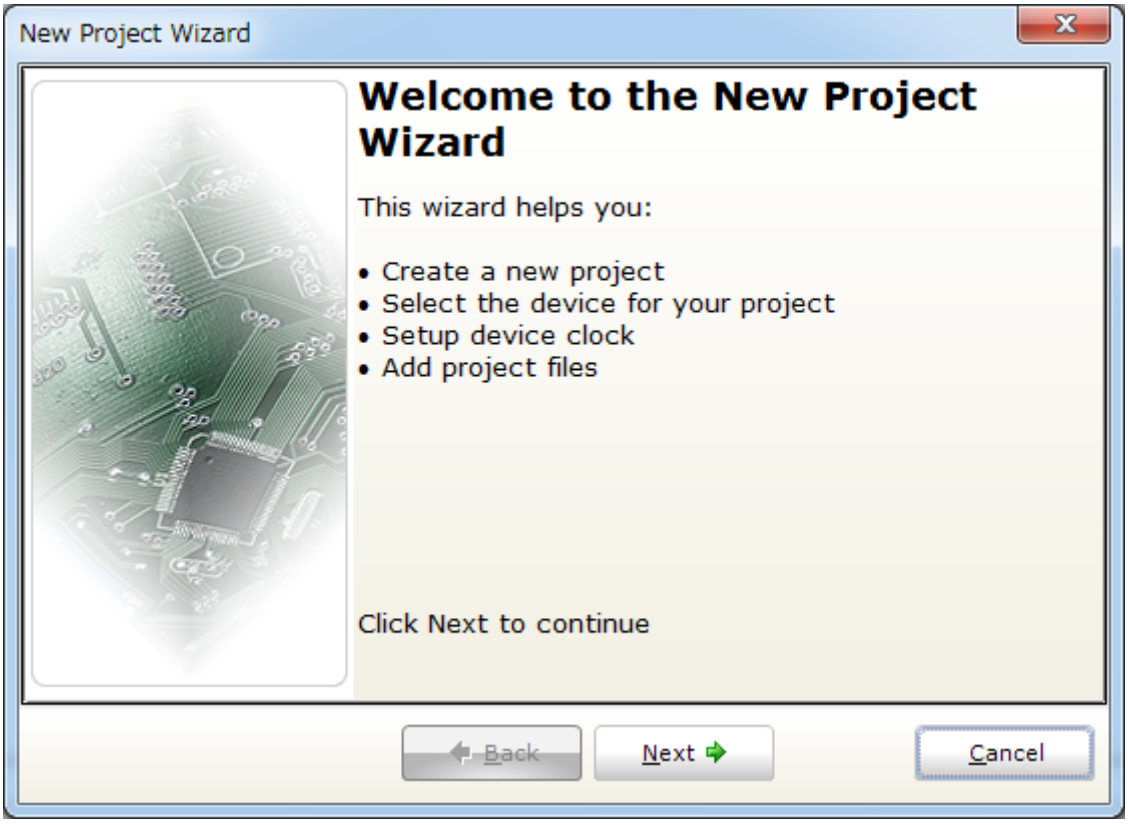
プログラムを開発するに当たり、この環境でプロジェクトを作成します。以下に手順を示します。

□メニューから Project→New Project と選択します。

『New Project Wizard』 ウインドウが表示されますので、その指示にしたがって Project を作ります。

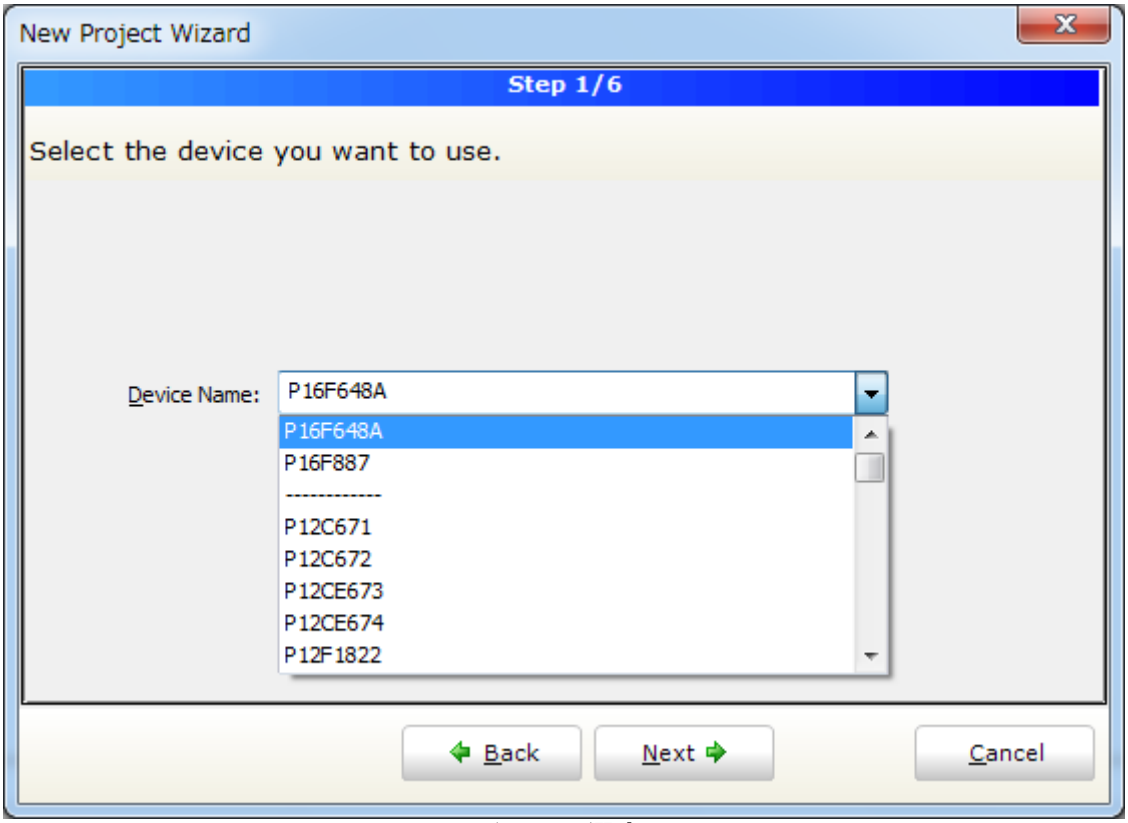


Project 作成開始



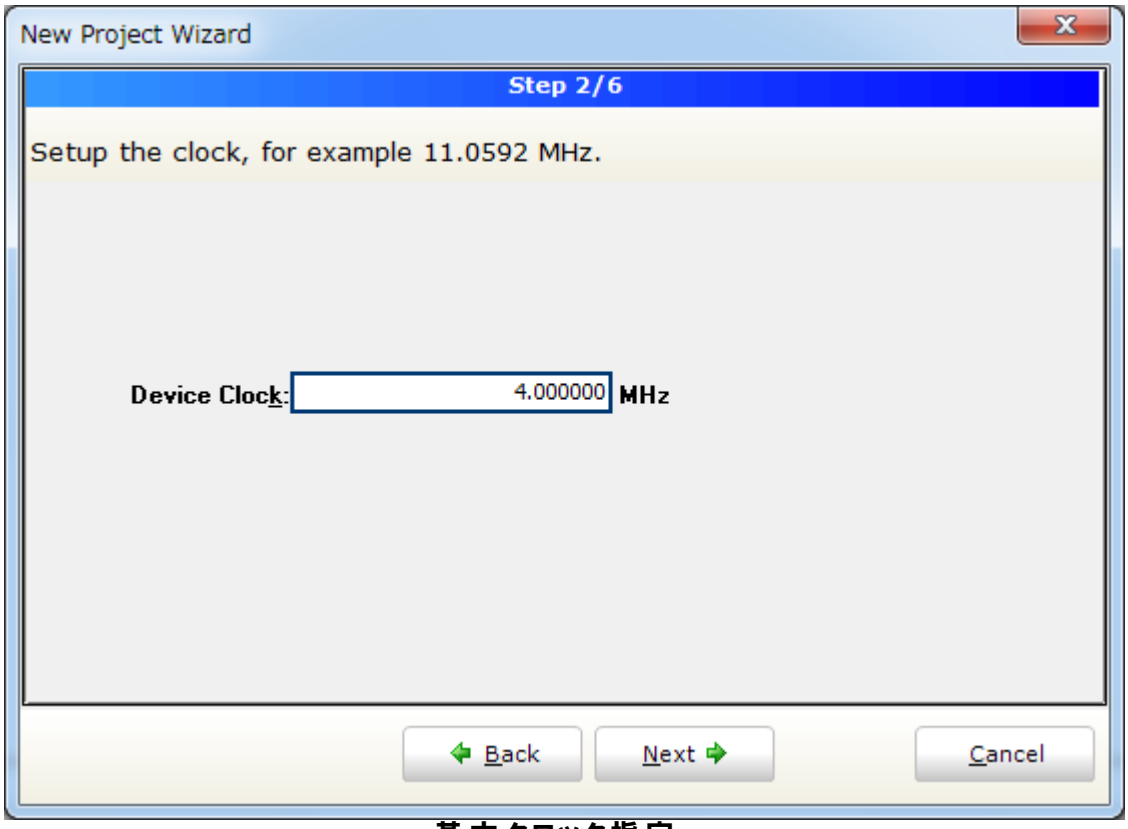
Project 作成ウイザード

□ Wizard が開いたら、Next を選択します。



デバイス選択

□Device を使用する PIC の型番を選択して Next をクリックします。

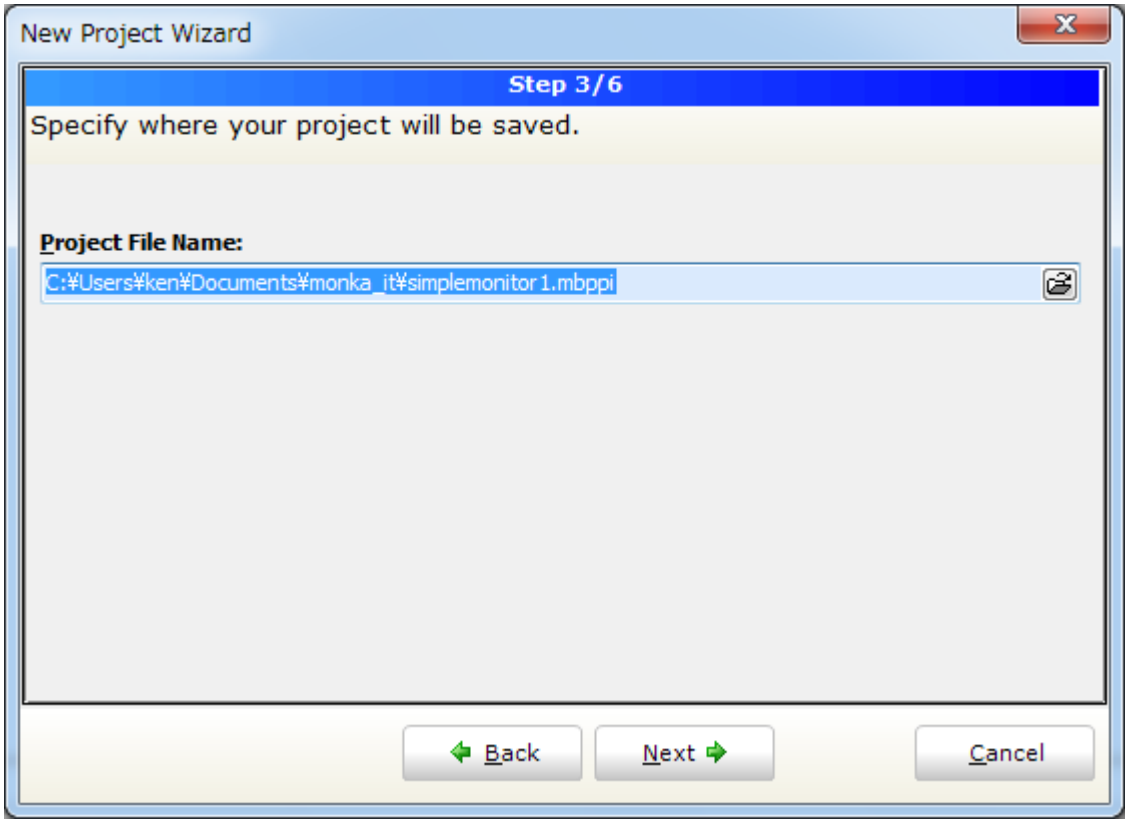


基本クロック指定

□Device Clock の指定画面になります。マイコンは PIC に限らず基本クロックが重要で、この周波数に応じてプログラムの処理速度やタイマー基本周波数、シリアル通信の基本スピードなどが変わります。ここでは内部クロック 4MHz を指定します。デフォルトは 8MHz です。

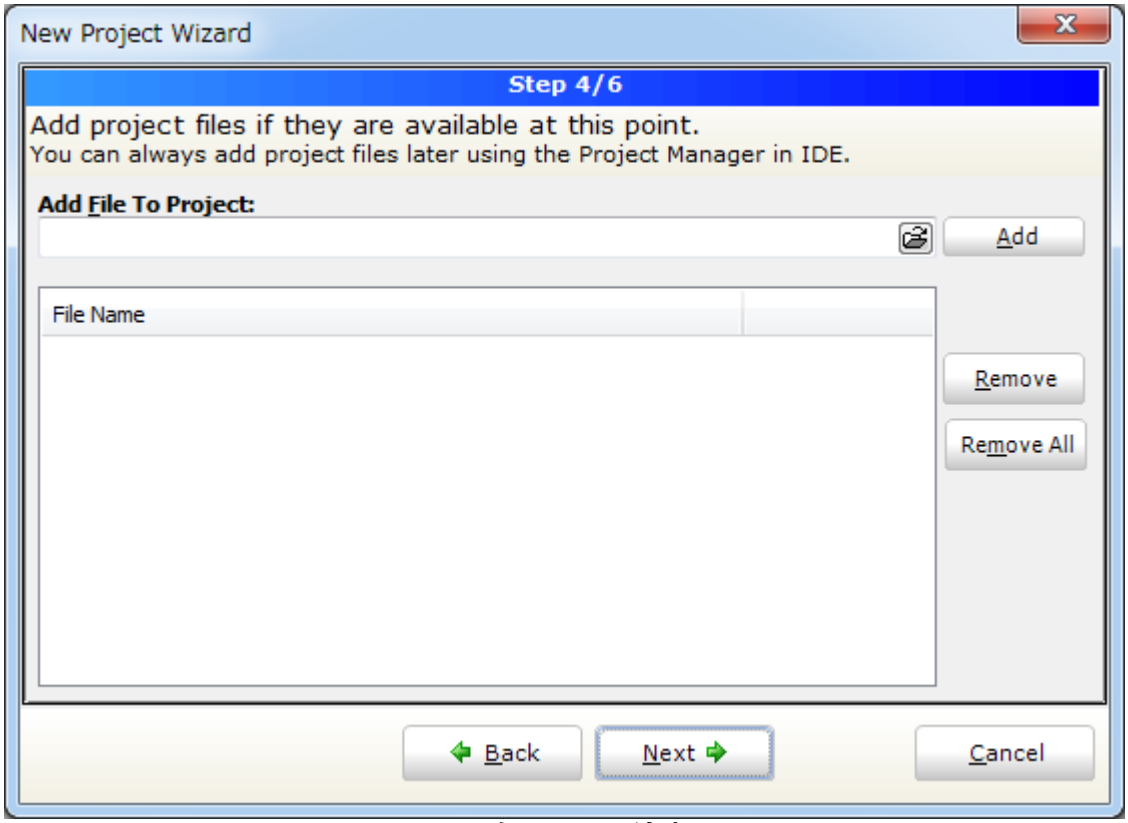
□次に、プロジェクトの名称と保存用 Path を指定します。

※Path に日本語が含まれないようにしたほうが良いでしょう。



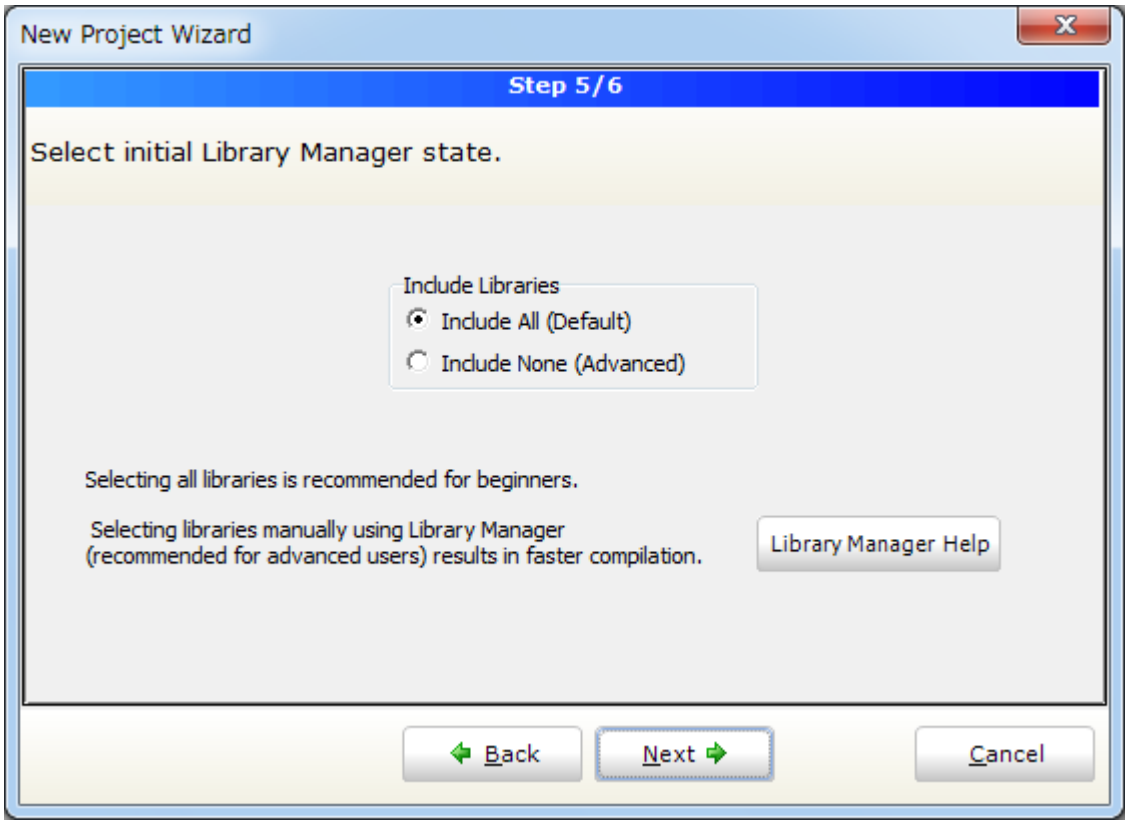
Project の path 指定

□プロジェクトに追加するファイルを指定できますが、ここでは、そのまま Next をクリックして先に進みます。

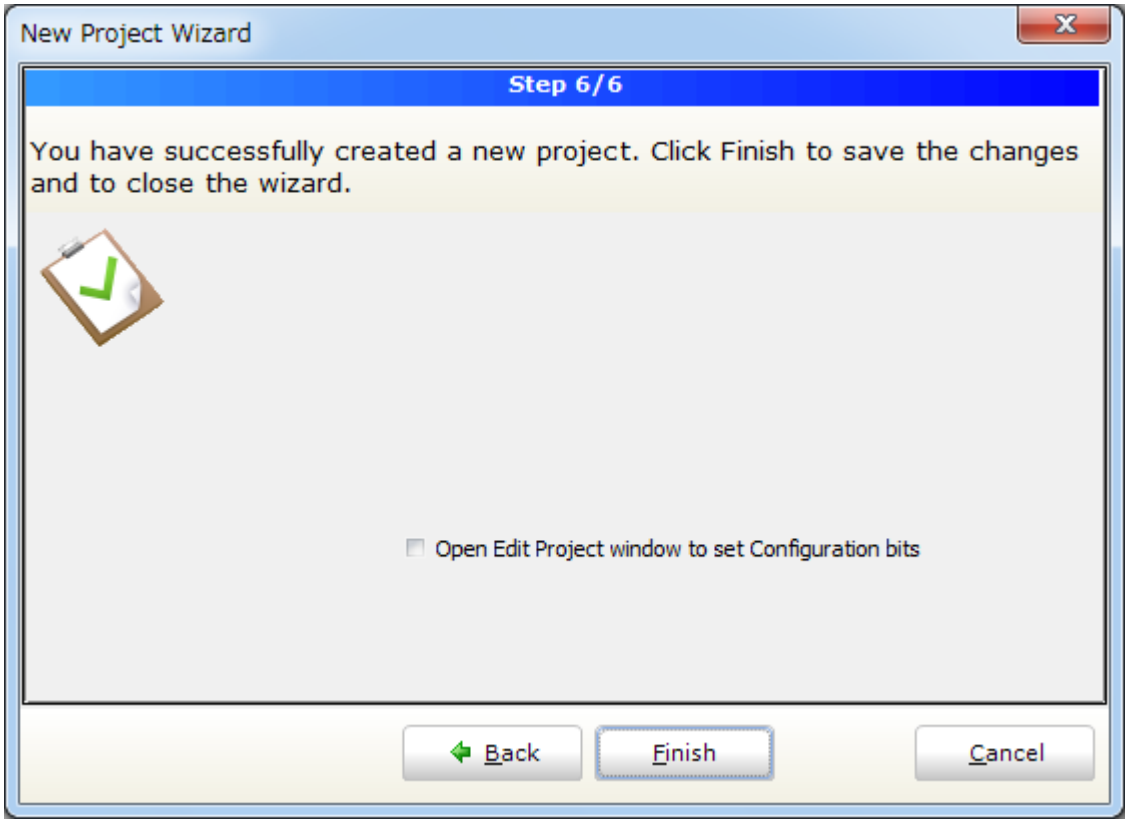


Projectc にファイル追加

□使用するライブラリの指定です Default で Next をクリックします。

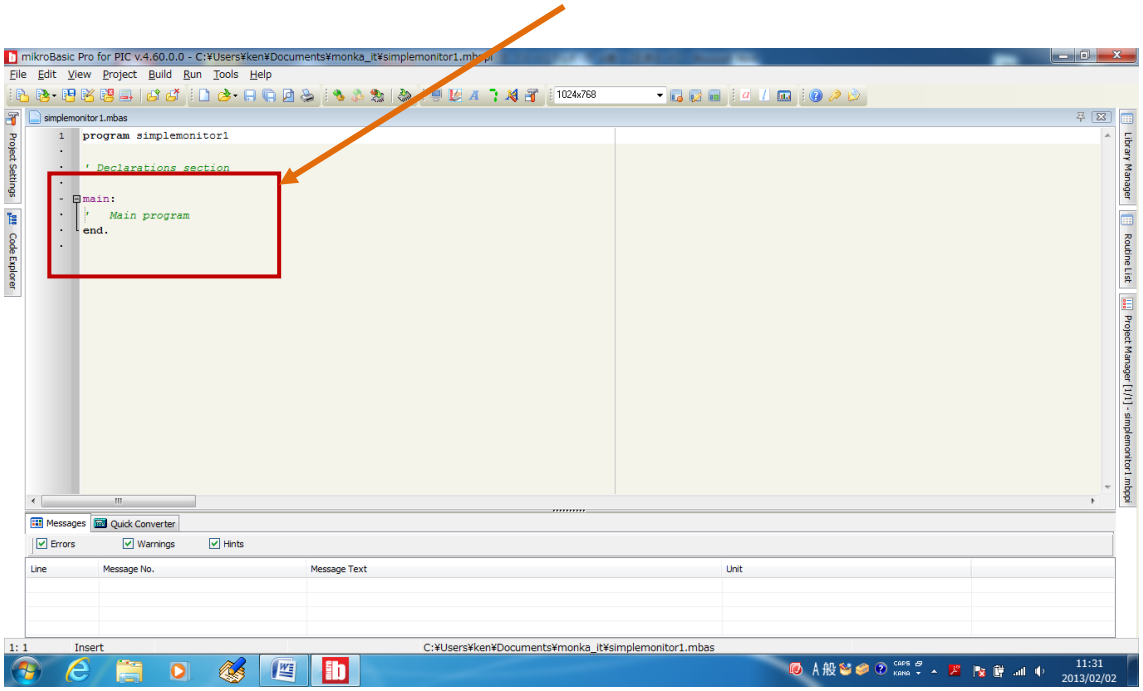


初期の Library 指定

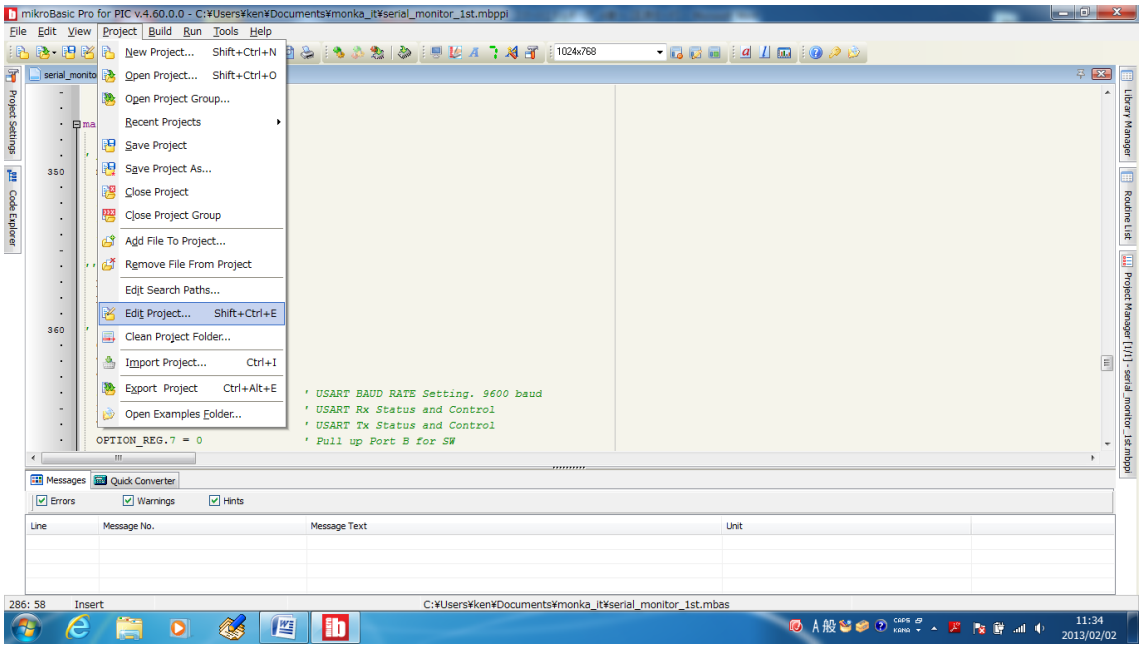


Finish 画面

□Finish で Project が作られて main 関数のひな型が表示されます。



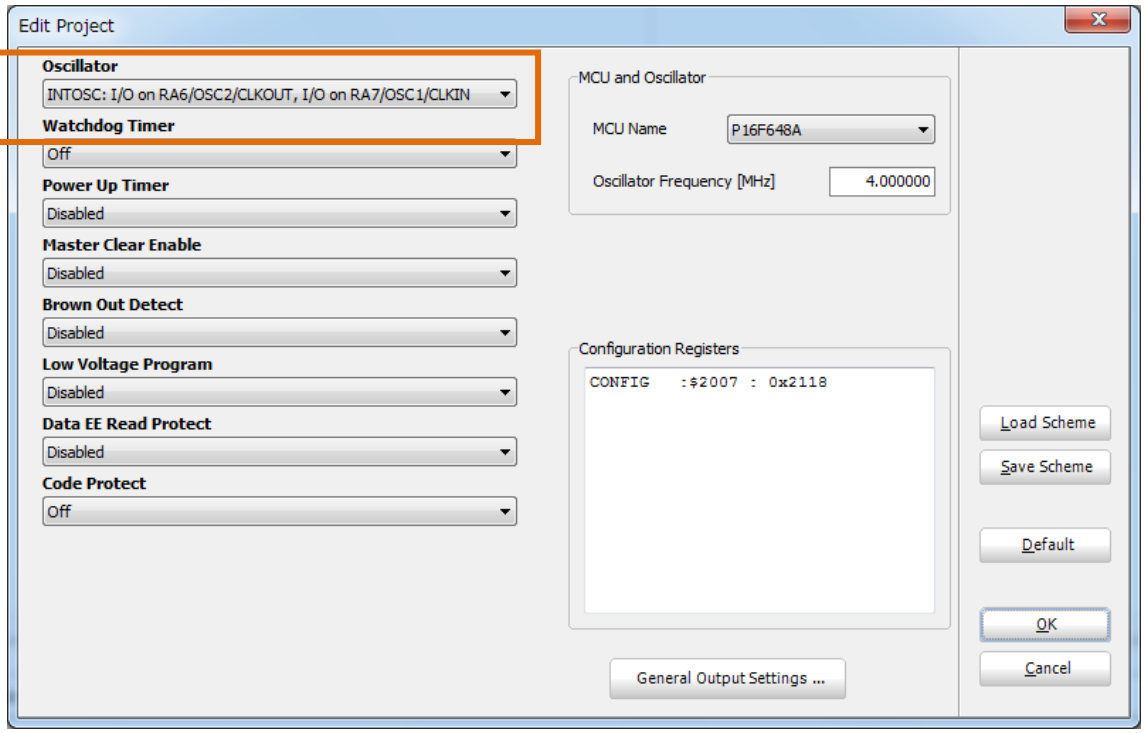
作成されたmain処理



Configuration選択

□メニューの Project→Edit Project で PIC マイコンの Configuration の設定を行います。下記を参考に指定してください。

※Oscillator の項目で内部発振を指定します。この設定が誤っていて動作しないことがよくあります。



Configuration画面

◆プログラム作成

□UART によるシリアル通信は、

9600bps、8bit data、1stop bit、non parity とします。

□送信電文は、1 文字として、次のような電文とします。

“S”：ステータスリクエスト（計測データ要求）

“M”：モード変更要求

※ドライブモード（D）放電テスト

PWM モード（P）PWM 駆動

“D”：PWM 駆動時の DUTY 比変更要求

□LCD 表示

簡易モニターは、8 文字 2 行の LCD を持っています。効率の良い表示をおこなうために、送信時には、上段先頭に『@』を付けた送信項目名を表示し、下段にその内容を表示することとします。これに対して受信時は、上段先頭に『＊』を付けた項目名として、受信したことを示します。

このユニットのプログラムは、main 関数の先頭部分で、PIC の I/O ポート・UART シリアルポートの設定をおこない、Loop に入ります。Loop では、2 つあるタクト SW の処理と Main ボードから受信した電文の処理をおこないます。EEPROM に書き込んである、バッテリ電圧の上限・下限電圧の閾値のうち、下限値を 1 度だけ Main ボードに送信する処理を加えています。

割り込みは UART の受信割り込みを使用しています。通信の電文は半角 ASCII キャラクターとし、1 文字受信するたびに割り込みが発生しますので、受信バッファから取り出して、電文バッファに書きくわえるプ

ログラムを sub procedure interrupt として記述します。

この環境では、PIC の割り込みはすべてこの名前の関数で処理します。

ソースプログラムは、割り込み処理部分を含めて全体で 450 行程度です。たいへん効率良く開発ができたと思います。

例として、SW2 クリック時のモードセット電文送信時は、次のように記述します。送信時なので、1 行目先頭に@マークがを表示しています。

```
Lcd_Out(1,1,"@MODE   ")   'LCD１行目１カラムから表示（常に８文字）
Lcd_Out(2,1,"    SET")    'LCD２行目１カラムから表示（常に８文字）
UART1_Write_Text("M")      'モード変更要求電文の送信
```

受信時の電文は、LCD 表示文字列そのものを、先頭に"U"または"L"をつけて Main ボードが送信します。先頭１文字で上段か下段かの判断をして受信した電文をそのまま表示します。

プログラムは次のような構成です。

※プログラム全体は、Appendix に掲示します。

1). dim で始まる定義ブロック

PIC の I/O ポートの用途ごとの割当て設定を使用するライブラリ用に設定します。また、ポート名称を何レジスタの何 bit 目、という表現では、分かりにくいので、用途に応じた名称に定義します。

また、システムで使用するワークエリア用メモリの設定・定数の定義などもこのブロックで行います。

2). 関数定義ブロック

sub Procedure XXXX()～end sub で囲まれたブロックで、各関数（サ

ブルーチン）を記述した部分です。

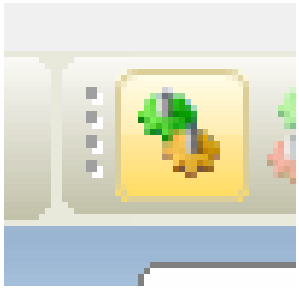
sub Procedure interrupt XXXX()～end sub で囲まれたブロックは、割り込み処理のプログラムです。各 PIC で使用できる割り込みは、すべてこの記述をした関数内で処理することとなるので、この関数の冒頭は、どの割り込みなのかを判断することです。簡易モニターでは、UART シリアル通信の受信割り込みだけ使用しているので、該当する割り込みであることを示す bit がセットされているかを調べています。

3). main()～end.ブロック

主たる処理のプログラムを記述した部分です。ここまでに宣言、定義した変数・定数や、記述した関数を使いながら、PIC がリセットされた直後からの処理を記述しますので、この関数の冒頭には、変数の初期設定や、I/O ポートの方向設定などを記述します。

◆ Build

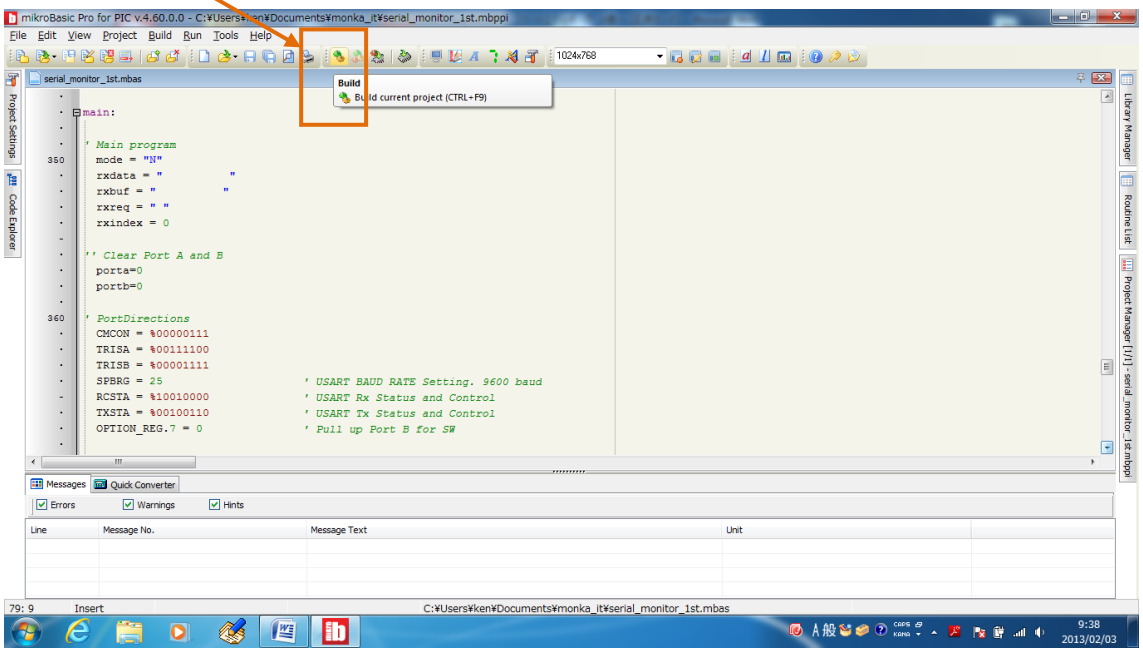
コンパイル等一連の実行可能ファイルを生成する手続きを Build と呼んでいます。mikroBasic のウインドウにある該当のボタンをクリックすれば、コンパイル～リンク～hex ファイル生成までをおこなってくれます。



Buildボタン

ウインドウの上部にある、歯車がかみ合っているボタンをクリックすれば、Build がおこなわれます。

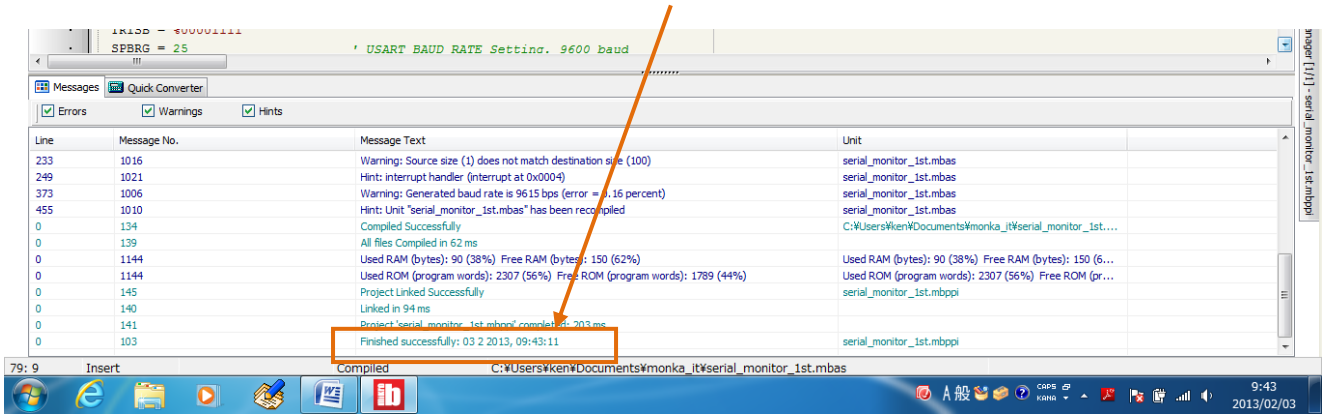
上部の Build ボタン



ProjectのBuild

□ウインドウ下部に Build の進捗が示され、エラーがあれば赤く表示されます。エラー該当部分は、ファイル名と行番号が示されていますので、そこをダブルクリックすれば、ソースプログラムの該当部分にカーソルが移動します。

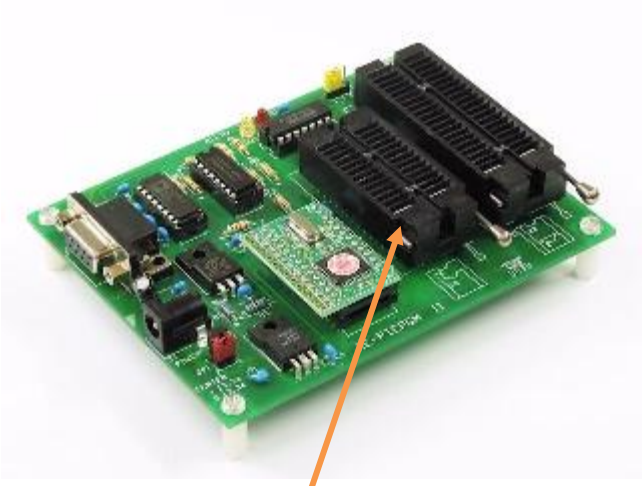
エラーがなくコンパイル・リンク・hex ファイル生成まで終了するとウインドウ下部の最後の行に『Finished successfully』と表示されます。



Build結果表示

◆書き込み

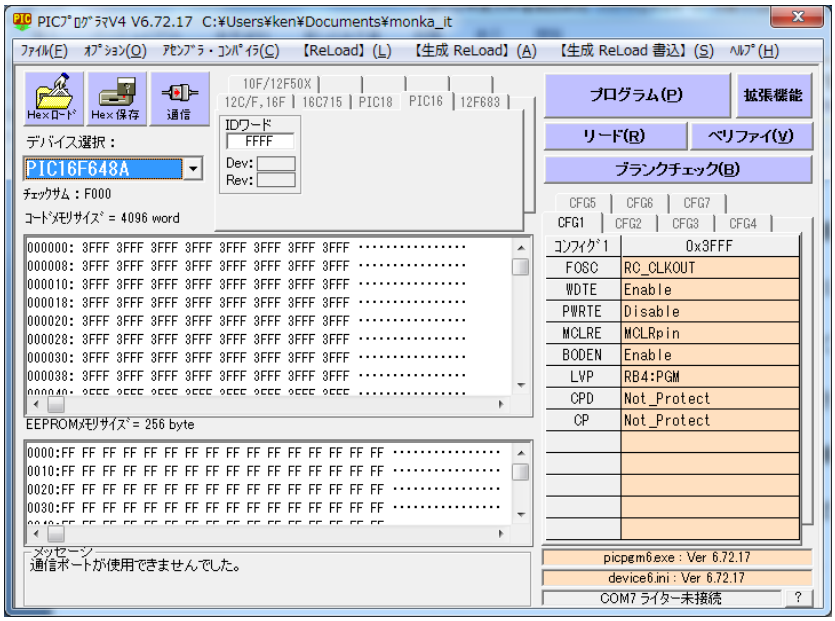
生成された hex ファイルを PIC に書き込みます。この作業には、『PIC プログラマー』と呼ばれるツールを使います。（写真）



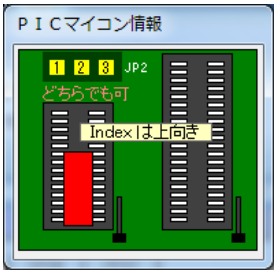
プログラマ

この基板を PC にシリアルケーブル（RS232C）経由で接続して、PIC に hex ファイルを書き込みます。書き込む際は、専用の書き込みソフトがありますので、これを使います。

基板上のソケットに PIC を差し込んで、書き込みソフトのプログラムボタンを操作して書き込みを行います。ソケットは 2 つありますが、書き込みソフトでどちらを使用するか指示が出ますので、それに従い PIC を差し込みます。



書き込み操作画面



ソケット使用指示

このプログラマ以外にも書き込みの方法がありますが、それは Main ボードの項で解説します。

４．２　Main ボード

◆開発環境

Main ボードは、PIC の開発元『Microchip』社が公開している無償の開発環境を使用し開発することにしました。

簡易モニターも mikroBasic を使わず、こちらで説明する環境を使用して開発することも可能です。IT 専門学校の教材としての選択肢として、この研究では意識して 2 つの環境を使っていますが、いずれも無償で使えます。簡易モニターが Basic 言語を用いたのに対して、Main ボードは、組込マイコンソフトではごく普通に使われる C 言語で開発します。コンパイラも PIC の種類に応じて複数公開されていますので、使用する PIC に対応したものをダウンロードして使います。

以下、環境作成の手順を示します。

【注意事項】この研究に使用した環境は、現行のものより一世代前のものです。現在は更新されて、旧版は入手できなくなっていますので、ここでは最新版の IDE について、入手とインストールの解説をしています。

最新の環境は統合開発環境として『MPLAB X IDE』という名称で、Microchip 社の HP から入手します。

□HP の TOP メニューからダウンロードに移動します。

□ダウンロードメニューの『ツール/ユーザガイド』を選択します。

Microchip 社の HP

□表示される画面を下にスクロールします。

ツール/ユーザーガイド

□製品一覧タブがあります。

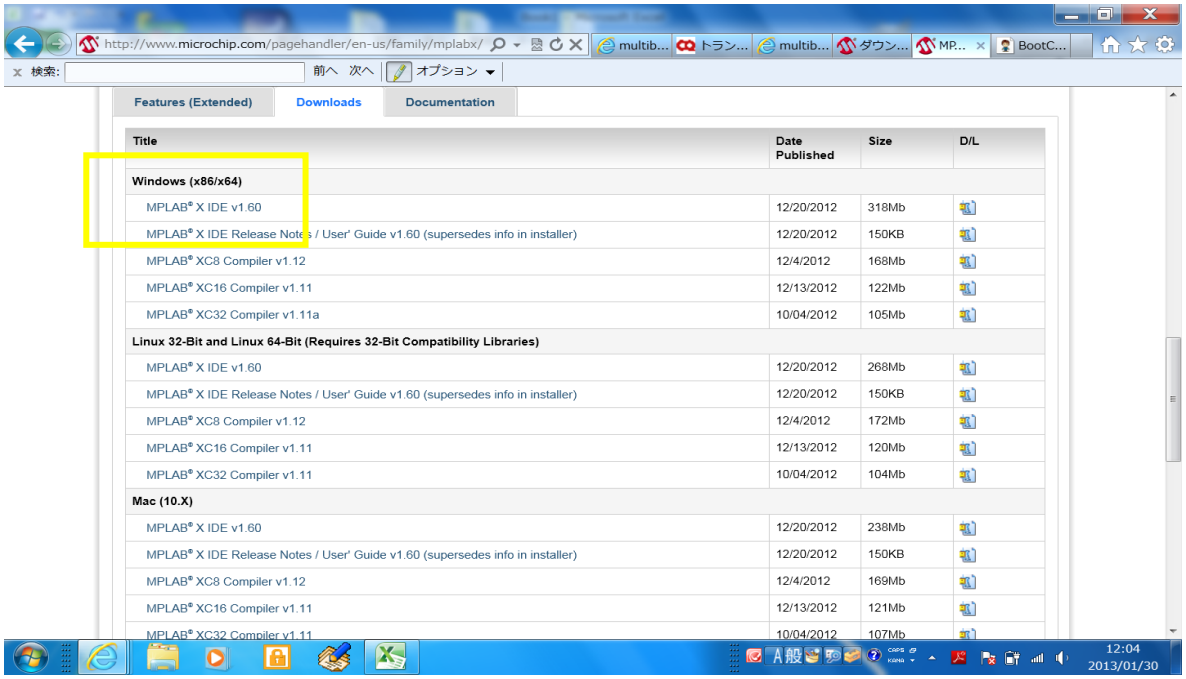
ここで開発ツールグループの MPLAB X IDE を選択します。

開発ツールグループ

□MPLAB X FREE DOWNLOAD を選択します。

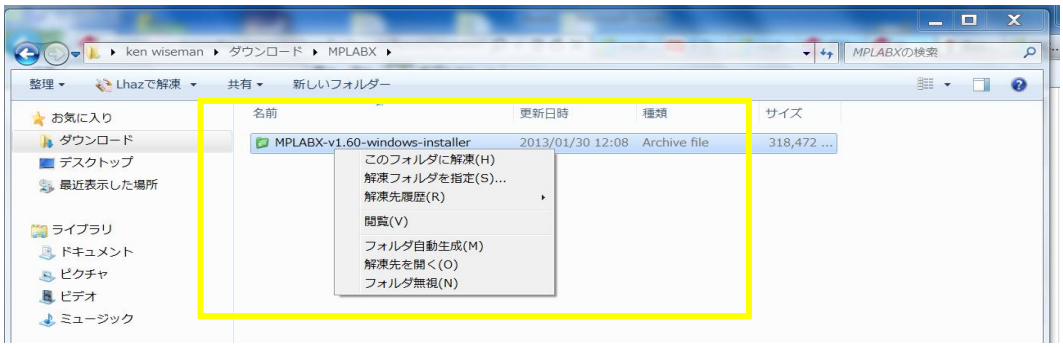
DOWNLOAD 選択

□対象 OS ごとにファイルが分かれていますので、ここでは Windows 版を選択してダウンロードします。保存フォルダを確認するウインドウが表示されますので、適宜指定します。



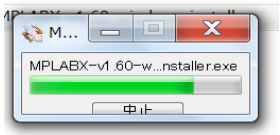
MPLAB X IDE 選択

□指定したフォルダに、最新版の IDE ファイルが保存されていますので、ダブルクリックして解凍します。

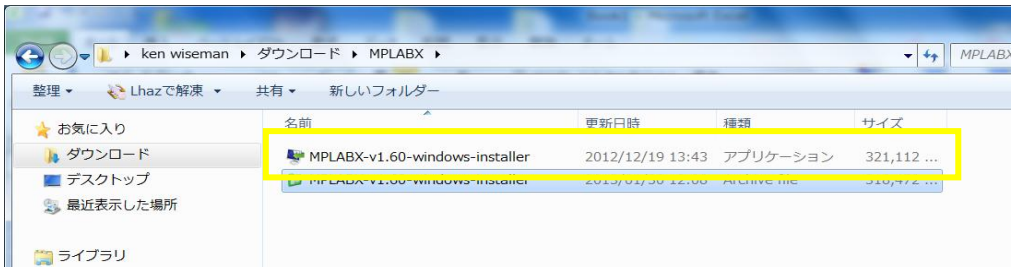


解凍

□解凍時にインジケータが表示されます。



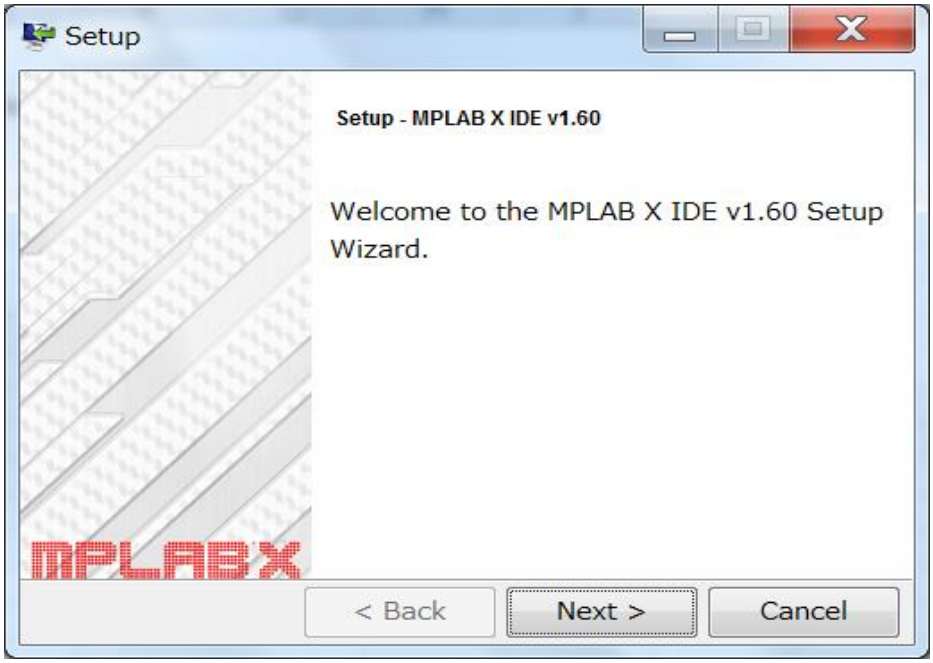
□解凍されたファイルができますので、そのファイルをダブルクリックしてインストールを始めます。



インストーラー開始

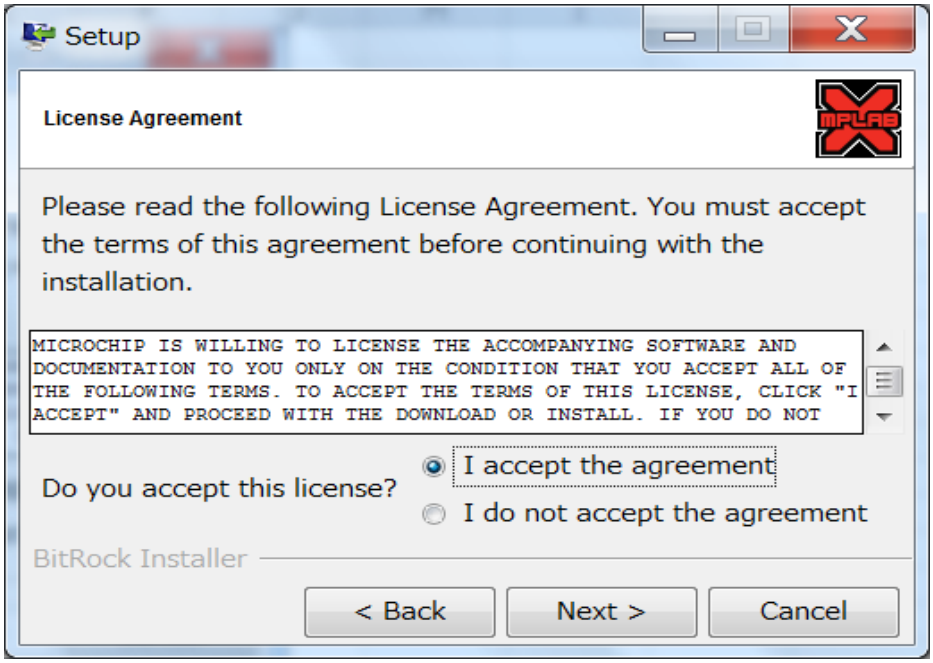
以下、インストールの手順です。

□Setup Wizard が開始されます。そのまま Next をクリックします。



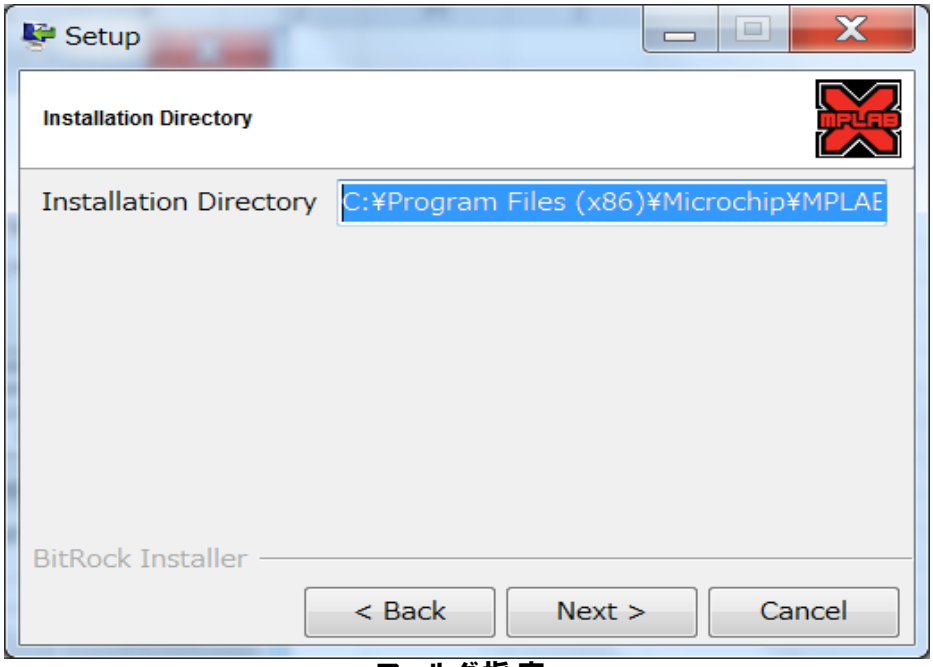
Setup Wizard 開始

□ライセンス確認画面では、『I accept・・・』を選択して Next ボタンをクリックします。



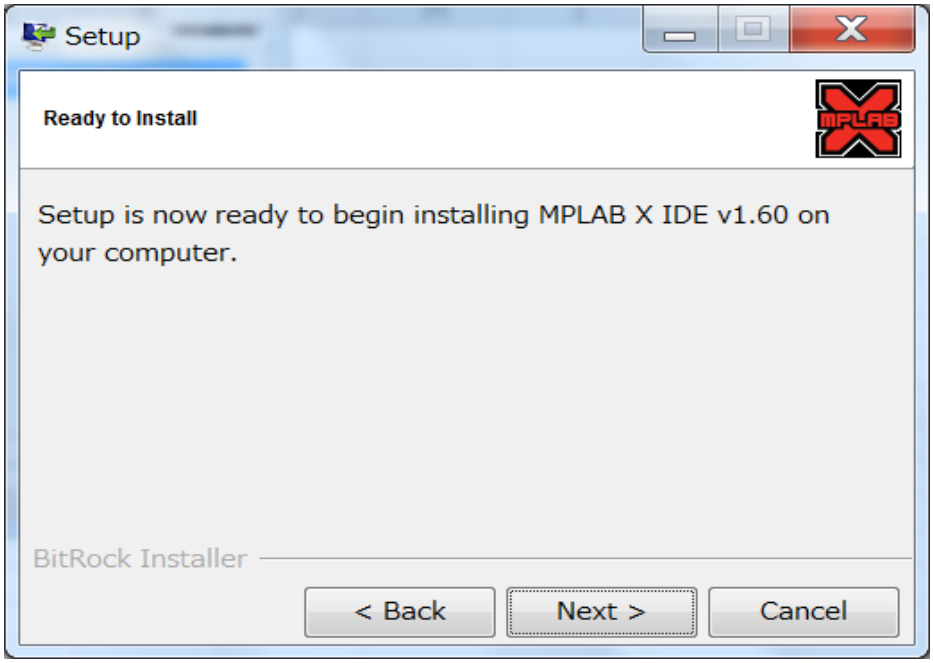
ライセンス確認

□インストールするフォルダの指定です。ここではそのまま Next ボタンをクリックします。



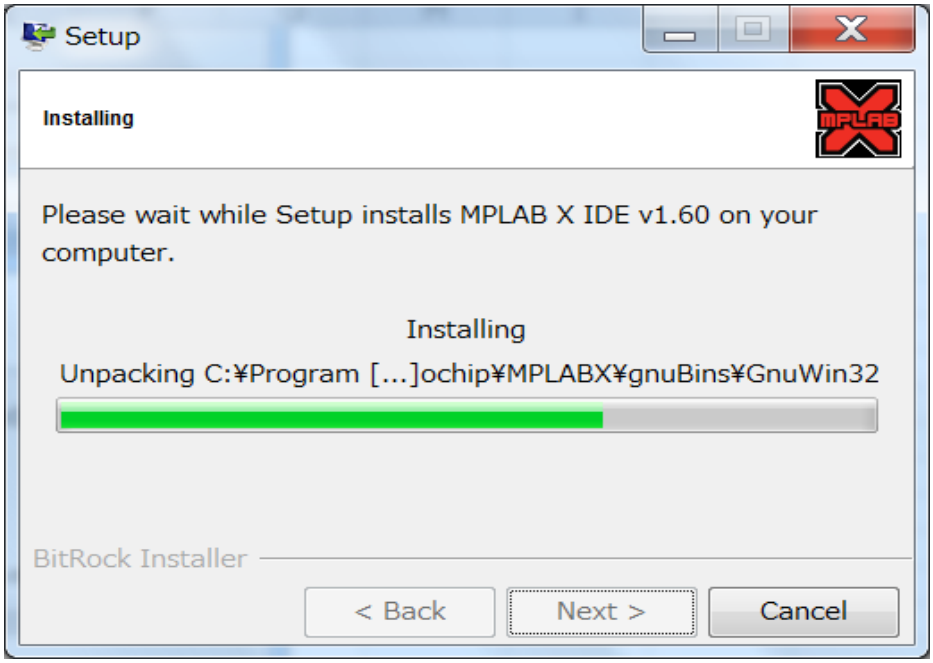
フォルダ指定

□事前の指定が終わりました。Next ボタンをクリックしてインストールが始まります。



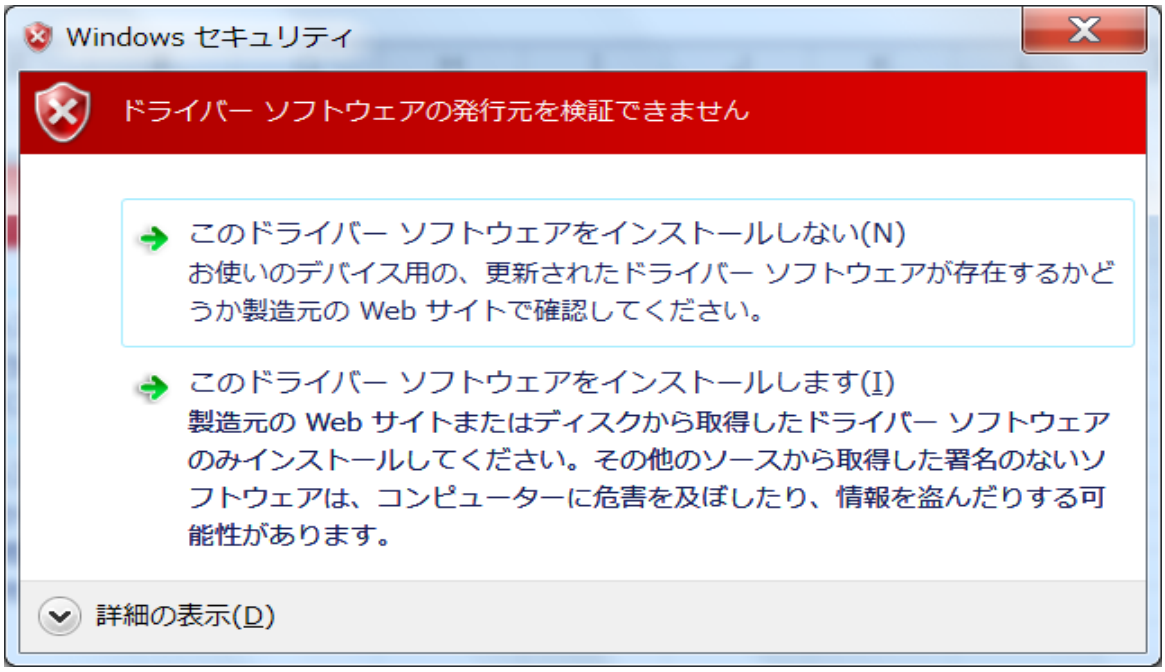
インストールの開始確認

□インジケータが進みます・・・



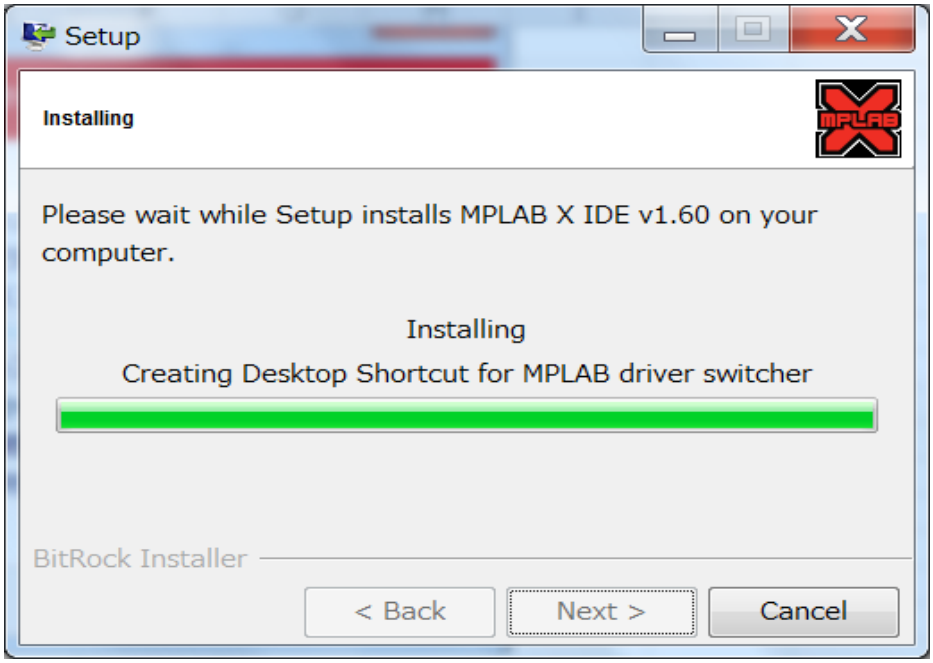
インストールインジケータ

□セキュリティ画面が表示されたら『インストールします』を選択して先に進みます。



セキュリティ確 認

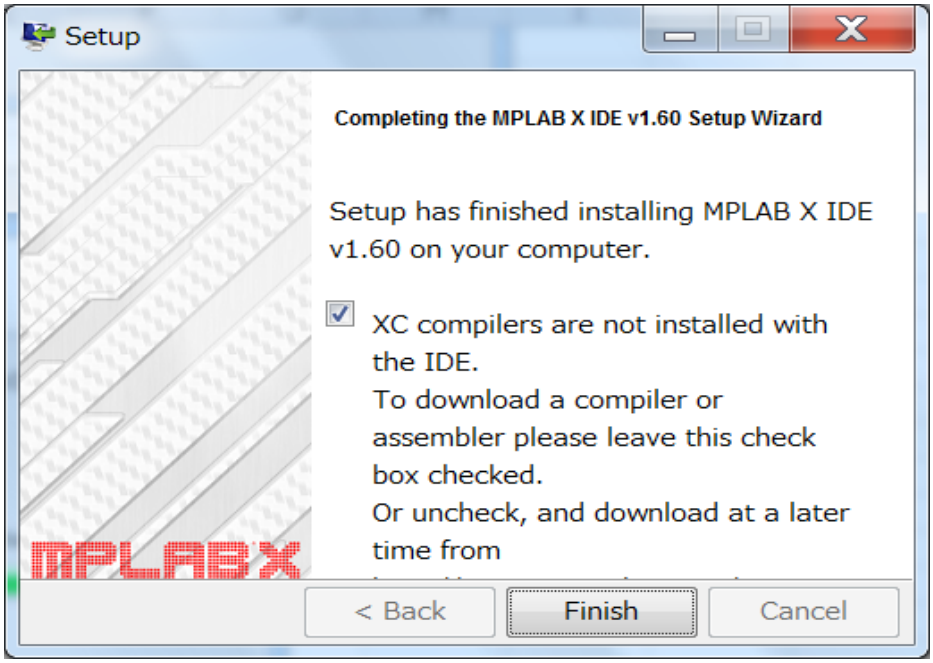
□さらにインジケータが進みます。



インストールインジケータ

□しばらくすると、インストール完了です。

Finish ボタンをクリックして、インストール完了です。



インストール完了

次に、コンパイラを入手します。

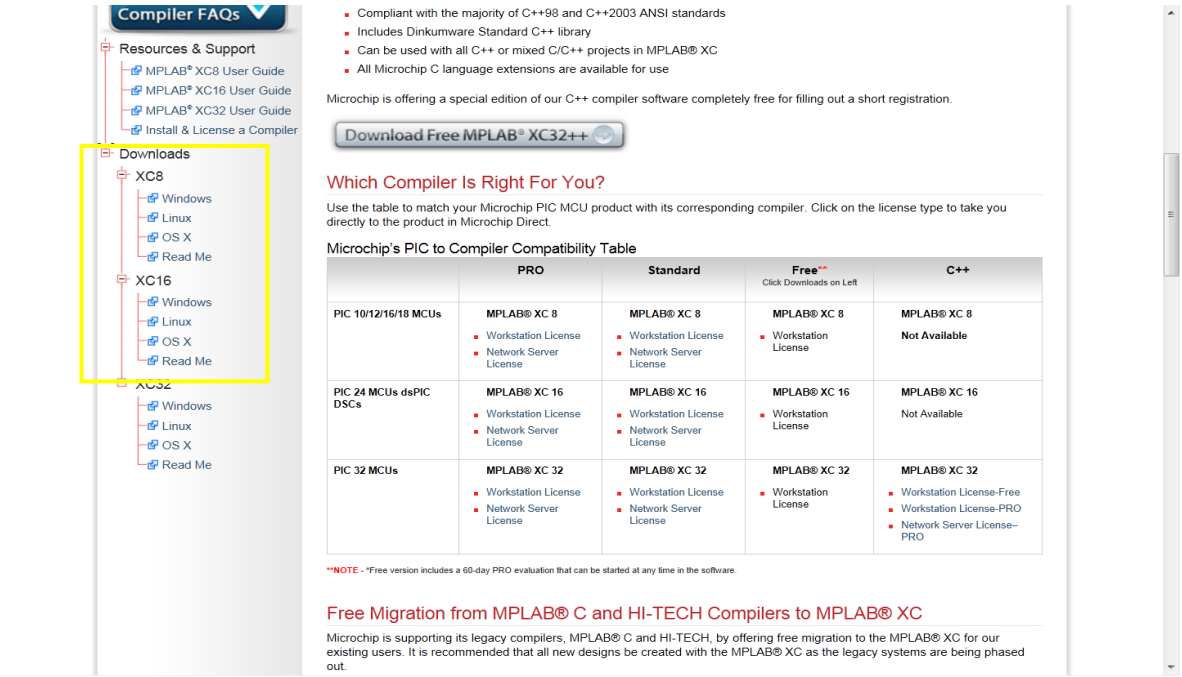
□Microchip 社 HP の開発ツールグループから、コンパイラを選択します。

コンパイラ選択

□移動先画面をさらに下にスクロールします。

XC コンパイラ

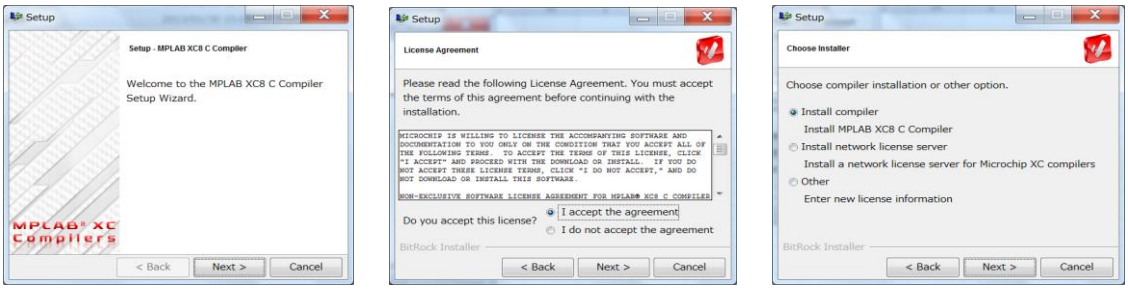
□画面左側表示の XC8 または XC16 を選択します。



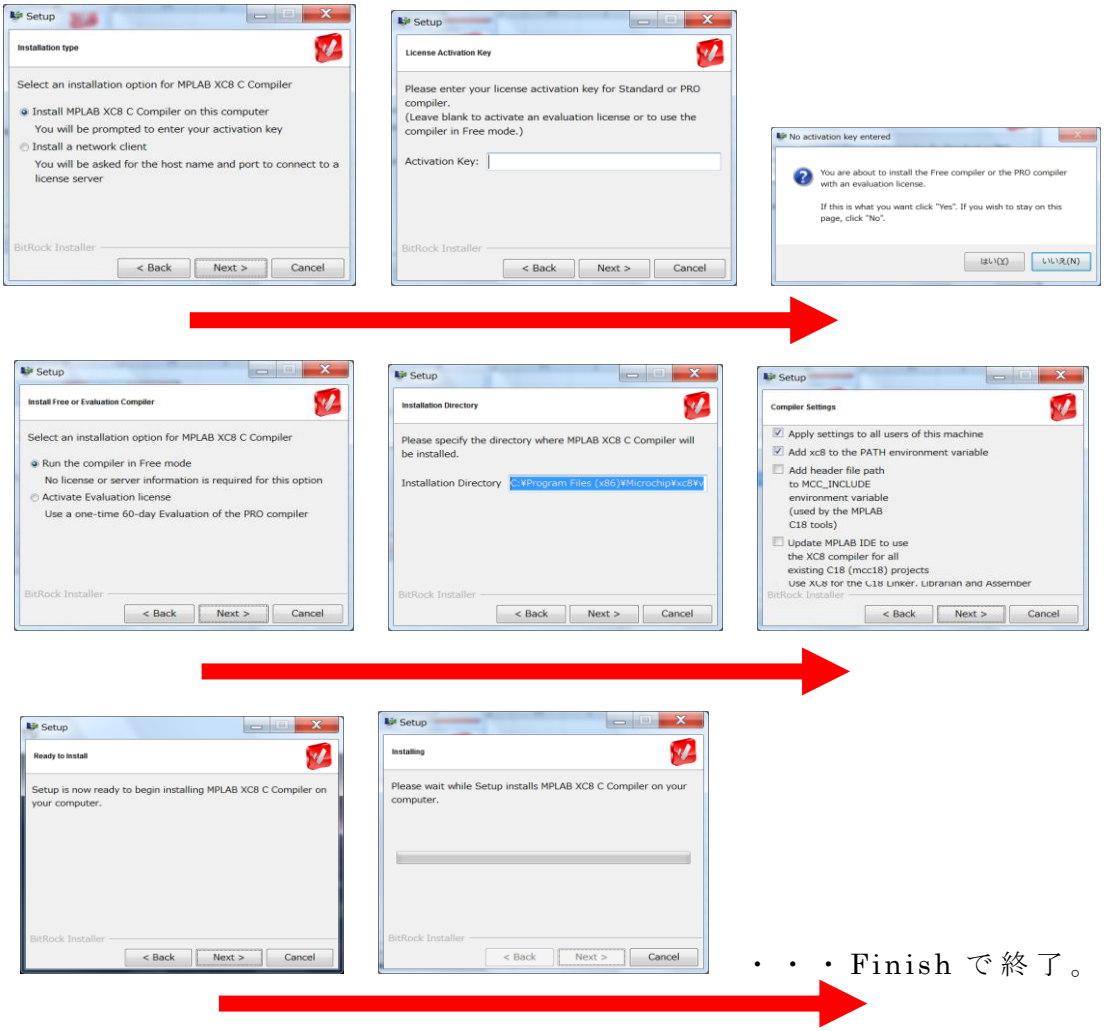
XC コンパイラ選択

※XC8 は 8bitPIC 用、XC16 は 16bit 用 PIC 用です。簡易モニターの PIC は XC8 で、Main ボードの PIC は XC16 で開発できます。

インストールは、ダウンロードしたファイルを解凍してできるファイルを実行するだけです。XC8、XC16 と操作の違いはありませんので、ここでは、XC8 を例に Setup 開始からの画面遷移を列記します。



XC コンパイラインストール進行



・・・Finish で終了。

XC コンパイラインストール進行

◆プログラム作成

PIC マイコンを使用して Bluetooth 経由で Android タブレットと通信するシステムのサンプルは、PIC メーカからもライブラリが公開されていて、それを利用したサンプルが WEB 上に数多く公開されています。PIC で USB に取り付けた Bluetooth を使うには、2 つの大きなプログラムが必要です。

①. USB 制御ソフト

②. Bluetooth 制御ソフト

いずれも、何もない状態から開発するとたいへんな工数がかかりこの研究の主題がずれてしまいますので、公開されているシステムをうまく使い、追加作成する部分の設計をおこないます。

main()関数に次のタスクを作成します。

①. Bluetooth 受信タスク

対向 Android から Bluetooth 経由で電文を受信します。

②. 通信テストタスク

Main ボードの通信テスト SW が操作されたことを対向 Android に通知します。

③. 放電テストタスク

放電テスト SW の操作を処理するタスクです。SW が操作されたことに対する内部処理と、簡易モニター・Android への通知が処理の内容です。

④. 充電テストタスク

充電テスト SW の操作を処理するタスクです。SW が操作されたことに対する内部処理と、簡易モニター・Android への通知が処理の内容です。

⑤. ハートビート LED タスク

main()関数の処理の進み具合を表示する LED の点滅を制御します。

⑥. シリアル受信タスク

UART 経由の受信処理をおこないます。対向機は簡易モニターです。

⑦. main 制御タスク

バッテリの電圧を閾値と比較しながら充電・放電の制御を、おこないます。また、停電検出信号の監視も行います。

⑧．SPP 受信処理

このシステムは、Bluetooth 通信を『SPP プロファイル』（Serial Port Profile）という方式で使用します。この方式は、Bluetooth で接続される 2 つの機器の通信をシリアル通信でエミュレートして、無線であることを意識せずに使用することができて大変便利です。この通信の受信の部分（Android からの受信電文処理）をおこないます。

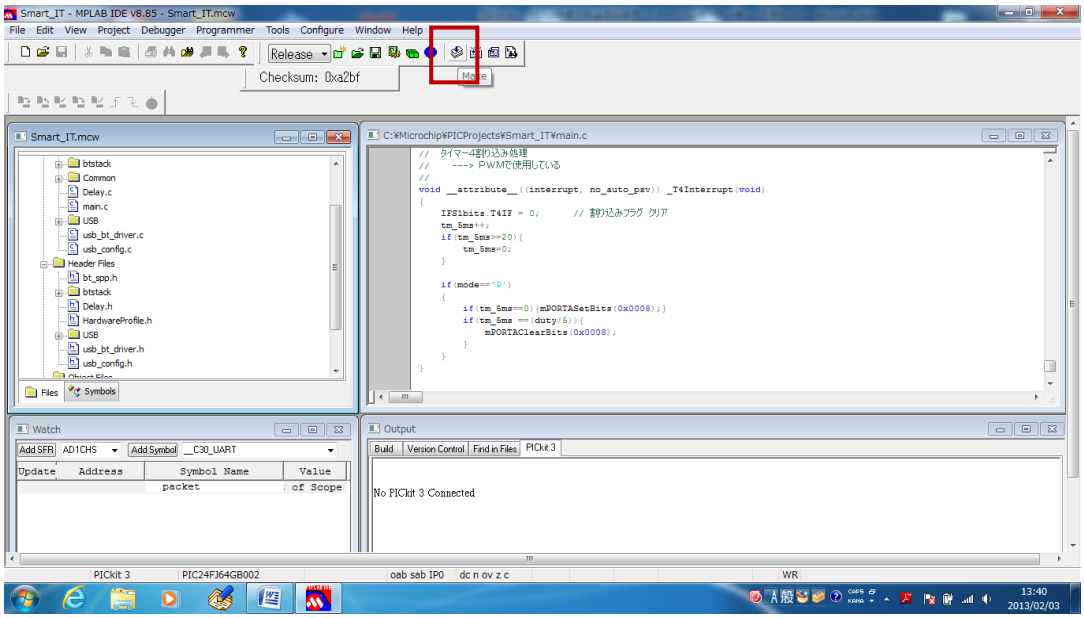
⑨．上記プログラムを全体的に制御する main()関数

以上、9 つの部分を作成します。

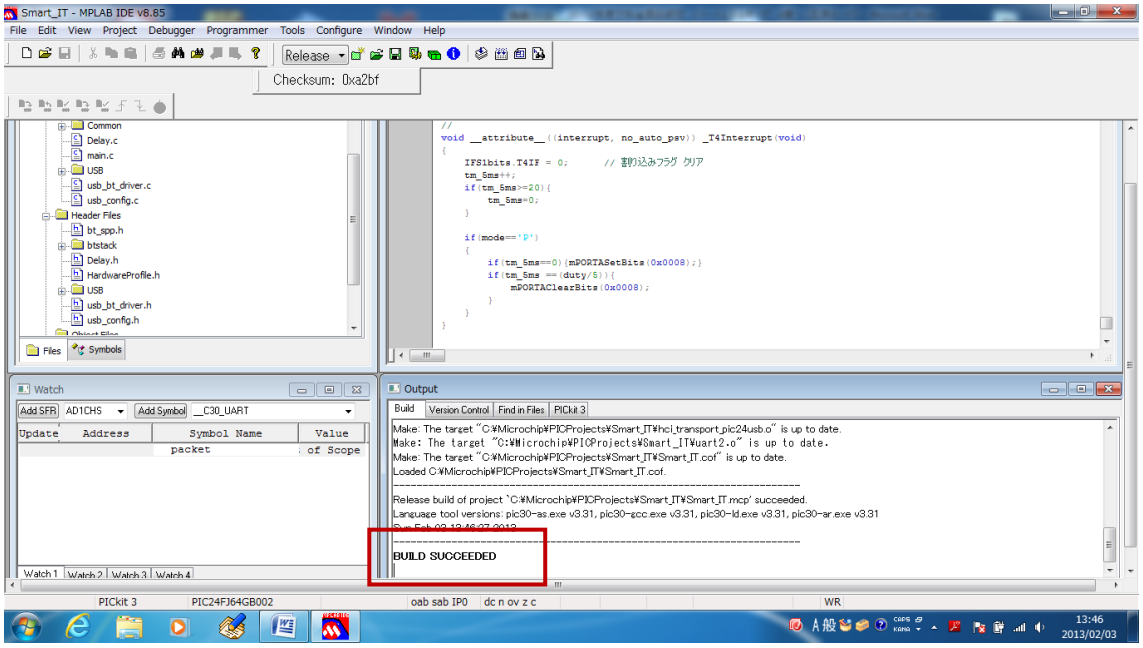
⑧を除く①~⑨は、main()関数と同じファイル main.c に記述しています。⑧は、bt_spp_recieve_callback()関数として作成しています。

◆Make

簡易モニターの開発環境では Build でしたが、こちらの環境では Make となっています。開発時に使用した MPLAB IDE のウインドウの上部にある Make ボタンをクリックすれば、コンパイル~リンク~hex ファイル生成まで一連の作業を実行します。



Make ボタン



Make 成功

Make によるエラー検出がなければ、右下 Output ウインドウの最後に、『BUILD SUCCESSED』と表示されます。

◆書き込み

出来上がった hex ファイルは、やはりプログラマという装置で PIC に書き込みます。MainCPU では、簡易モニターとは違った装置で書き込みます。

MainCPU は、ICSP(In Circuit Serial Programming)という手法が使えます。書き込みに使用する装置は、Microchip 社の PICKIT3 というもので、CPU を基板に装着したまま、hex ファイルの書き込みができます。

ICSP による書き込みの様子

書き込み操作は PICKIT3 を USB 接続して、使用している IDE ウインドウで、上部にあるメニューから Programmer→Program と選択します。数秒で書き込みが完了します。

４．３　Android タブレット

◆開発環境

Android の開発環境も、無償で提供されているものが使えます。次の 4 つのプログラムを順にインストールして環境を作成します。

①．Java 開発キット（JDK）のインストール

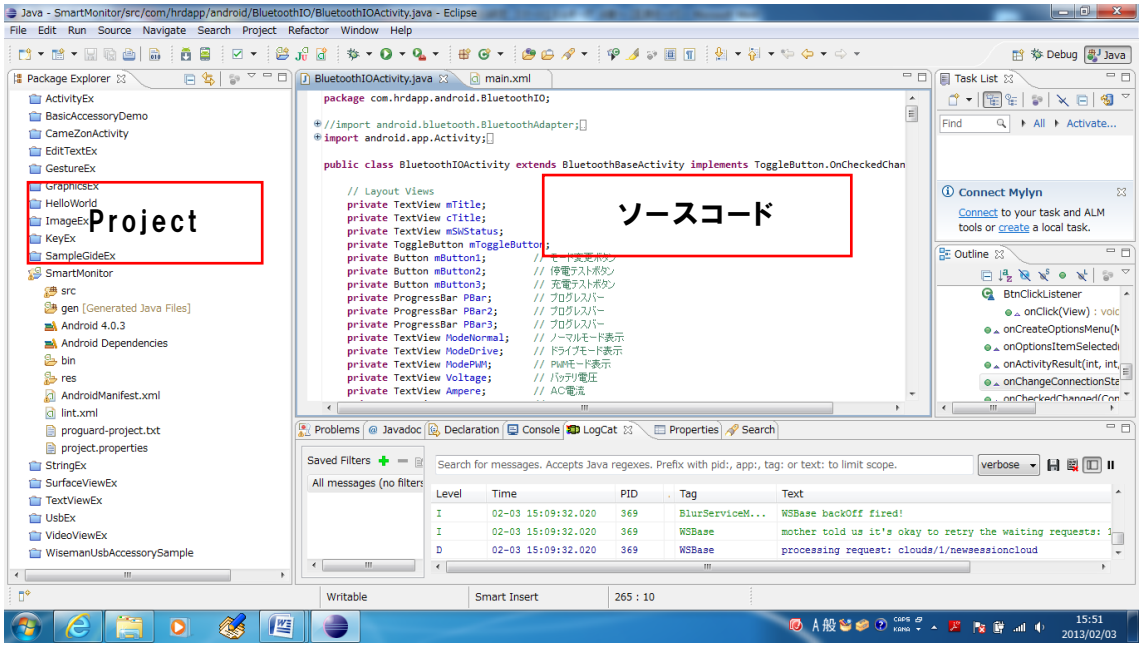
②．統合開発環境（Eclipse）のインストール

③．ADT（Android Developer Tools）のインストール

④．Android SDK のインストール

開発環境のバージョンアップが頻繁に行われているので、どんどん使いやすくなっている半面、具体的な手順をここで紹介しても、短時間の間に陳腐化してしまいます。WEB や参考書籍（Android プログラミングバイブル：ソシム：布留川英一著）などを参考に環境を整えてください。

開発に使用した Eclipse ウインドウを示します。



Eclipse による IDE

◆プログラム作成

PIC を使用する簡易モニターと Main ボードのプログラムと違い、Android 側のプログラムは、ほぼ出来上がっているものが公開されていますので、画面設計と Bluetooth 通信電文に応じた追加をおこなえば完成できます。

画面は次のような構成としました。配置した Object を示します。

①．タイトル（固定表示）

②．モード表示（NORMAL）

③．モード表示（DRIVE）

④．モード表示（PWM）

※該当時モード名が青色になります。

⑤．バッテリ電圧

⑥．消費電流（AC）：インバータを介して出力している電流のみ

⑦．DUTY 比（PWM モード時）

⑧．受信メッセージ：Main ボードから受信した Bluetooth 通信メッセージを表示します。

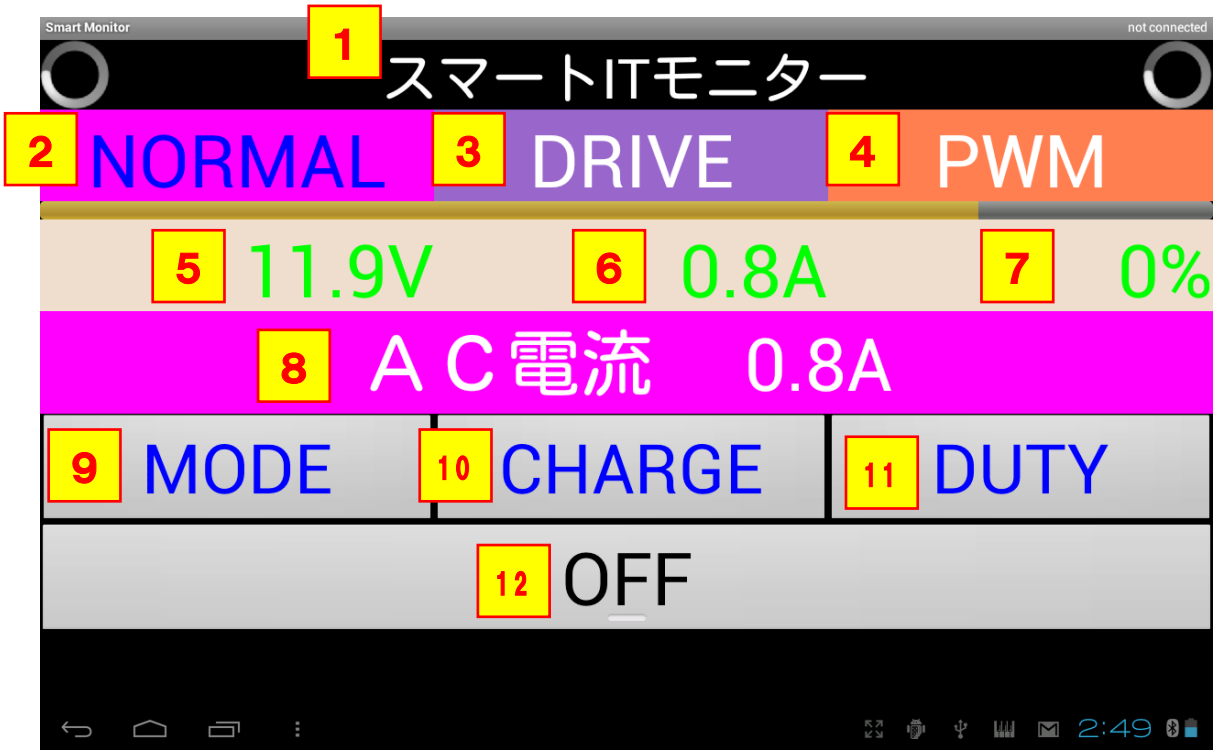
⑨．モード変更ボタン

⑩．充電テストボタン

⑪．DUTY 変更ボタン

⑫．Bluetooth 通信テスト

実際の画面は、次のとおりです。



Android タブレットの画面設計

プログラムは、主となる Activity クラスに各 Object の部分を追加して作成します。

主な部分は、以下の通りです。

□ BtnClickListener

⑨⑩⑪のボタンがクリックされたときに処理をおこないます。

※Main ボードへの Bluetooth 通信電文は、

⑨MODE：モード変更＝M

⑩CHARG：充電テスト＝C

⑨DUTY：DUTY 変更＝D

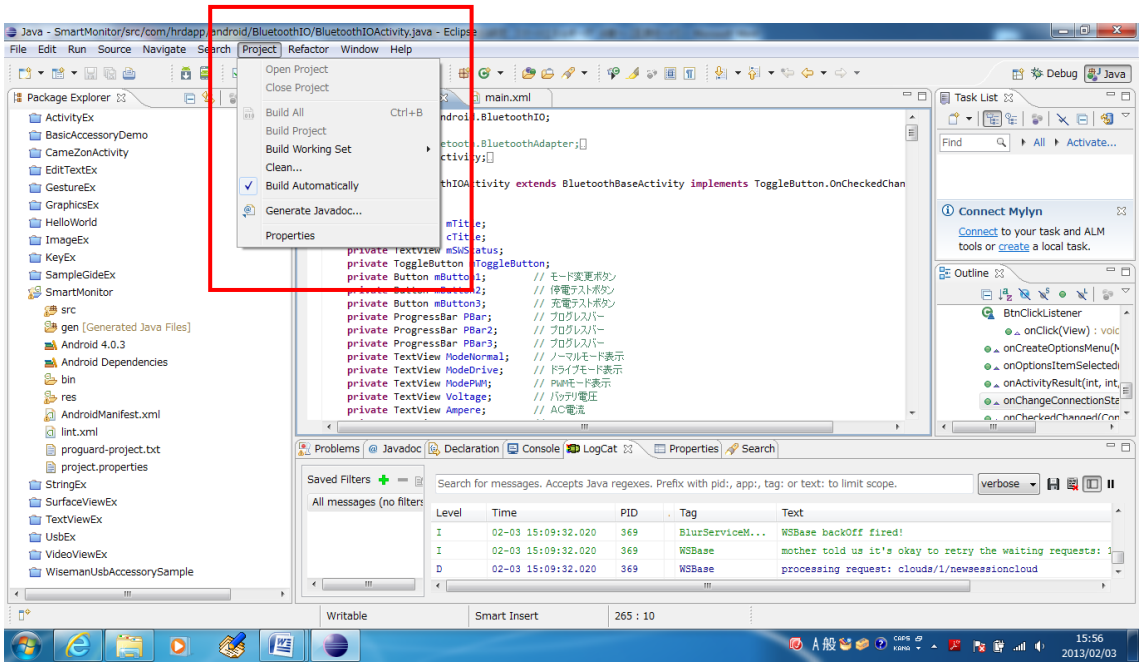
の、各１文字の電文とします。

□ onReviceMessage

Bluetooth 通信により受信した Main ボードからのメッセージを処

理する部分です。モード変更メッセージの場合は、②③④のモード表示を変更します。また、DUTY 変更メッセージの場合は、DUTY の％表示とプログレスバーの割合を DUTY に合わせて変更します。

◆ Build



Auto Build

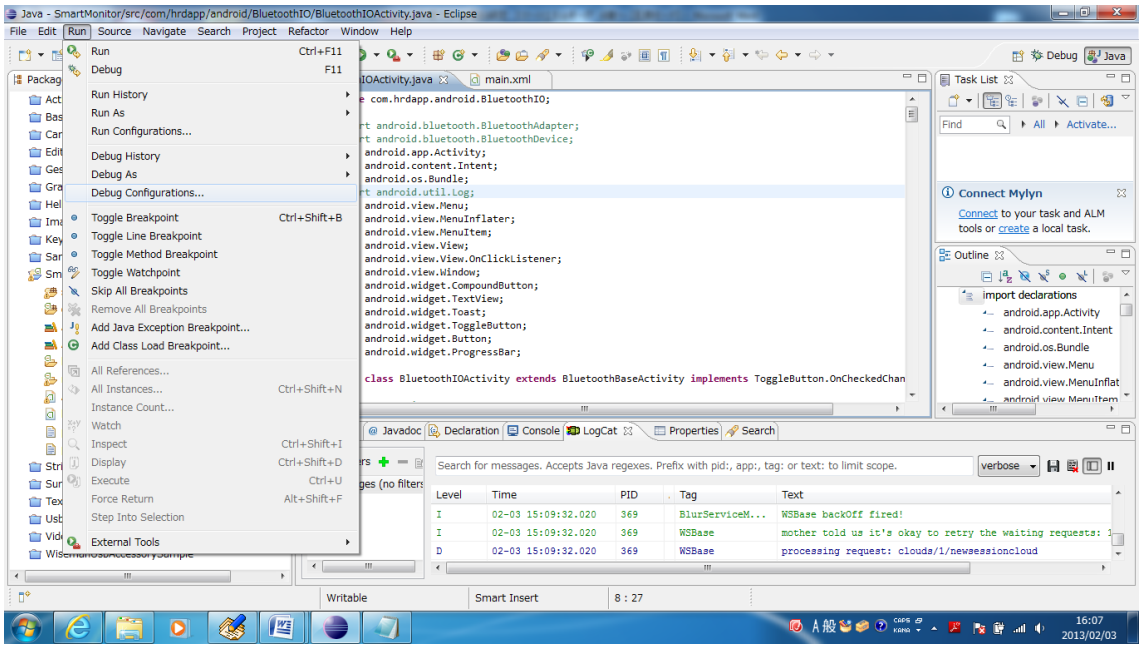
IDE の上部にある Project メニューから Build Automatically を選択しておくと、ソースードを入力すると同時に Build が実行されます。誤った記述をするとその時点で、エラー表示されるので、すぐに訂正することができます。

◆書き込み

Android タブレットのプログラム書き込みは、書き込み操作をおこなうというよりも、Debug 操作をおこなうと説明したほうがよく理解できます。

Android タブレットを USB ケーブルで PC に接続します。

Eclipse ウインドウで RUN メニューから Debug Configuration に進み Debug を開始すると、現在のタブレットプログラムがダウンロードされて、プログラムが開始されます。



５．テスト

テストは、一般的なソフトウエアの Debug 作業になります。詳細な記述はさけますが、基本的には、３つのユニットのなかで、簡易モニターを先にやや完成させ、続けて Main ボードを進めて、Bluetooth で接続してタブレットをテストしたあと、全体のテストを行うという手順です。

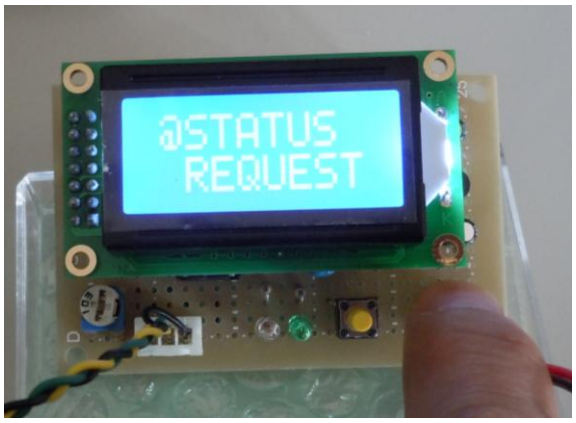
簡易モニター表示例

５．１　簡易モニター

完成した簡易モニターは、まず電池を接続し、5V の電圧が PIC、RS232C レベル変換 IC、LCD 電源ピンに正しく出ているかをテスタで確認します。その後、電池を一度外し LCD を接続した後、改めて電池を接続します。その際、LED が何度か点滅した後に、LCD にメッセージが表示されるかどうかを確認します。

次に、SW を操作して該当のメッセージが表示されることを確認します。

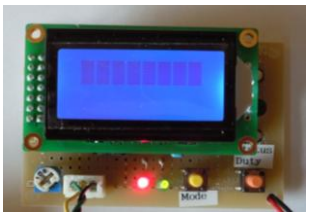
簡易モニターは、後の操作で逐次画面を確認しますので、この段階で完全に出来ていなくても、メッセージの表示と SW 操作に対する反応ができていれば良いでしょう。



状態要求画面

写真は、SW１を操作した際の STATUS 要求画面が正しく表示されている様子です。

以下、簡易モニターの画面表示例を示します。

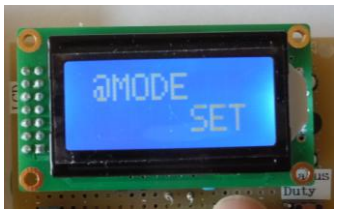


電源投入時

モニタースタート

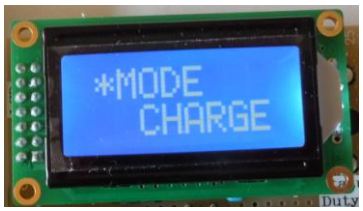
バッテリ上限電圧

バッテリ下限電圧



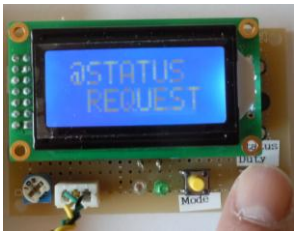
モードセット要求

駆動（放電）モード

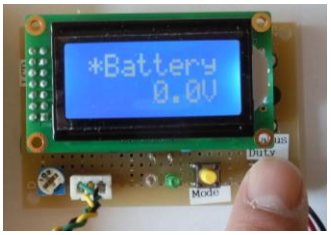


充電テスト

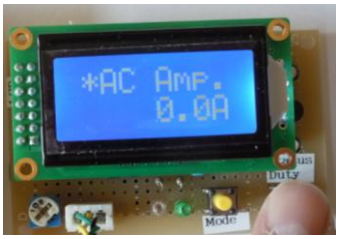
通常モード



状態要求



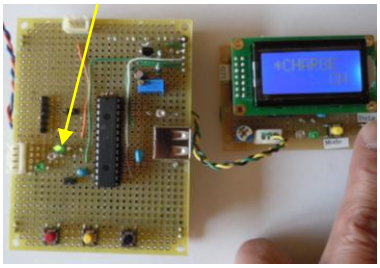
状態要求応答



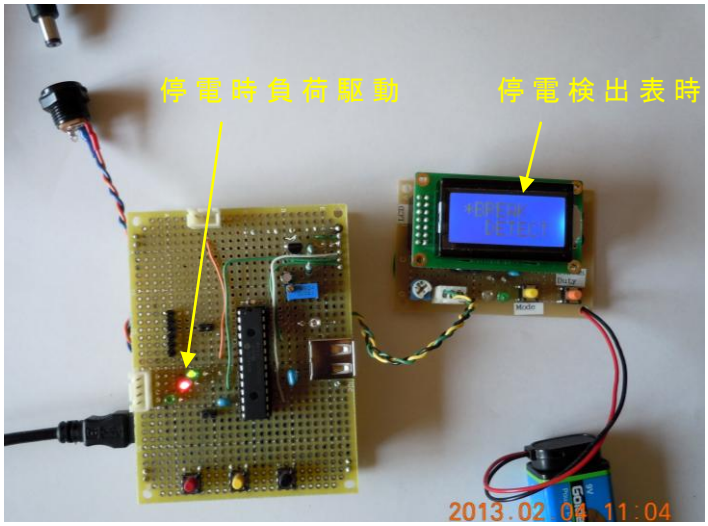
状態要求応答

状態要求応答予備

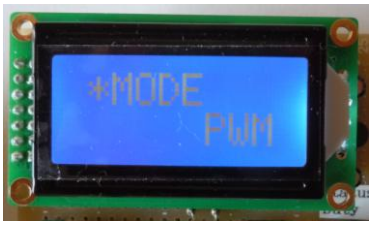
充電 LED 点灯



充電テスト



停電検出



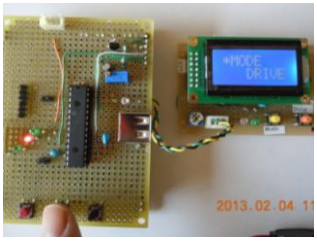
PWM モード

DUTY 変更要求

DUTY10%

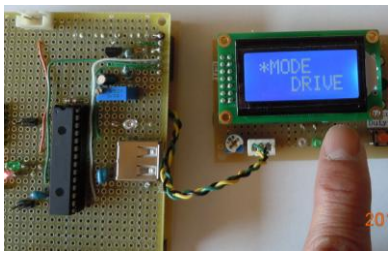
DUTY30%

DUTY40%



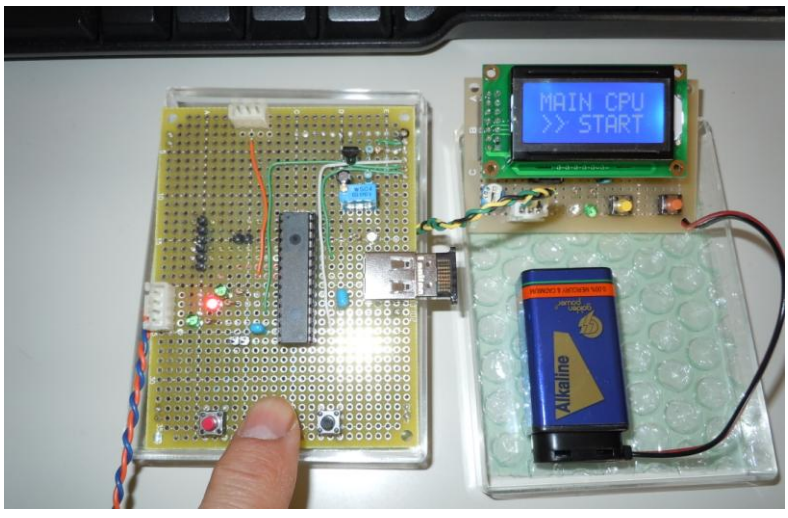
Main Drive

Main Charge



Monitor Drive

５．２　Main ボード



Main ボード

Main ボードは、Power ボードから電源が届いて要ることを確認したら、簡易モニターを接続して、電源を入れます。

通電開始すると、LED が何度か点滅を繰り返し点滅した後、簡易モニターに、Main ボードが処理開始した

メッセージが表示されれば、シリアル通信正常であることが確認できます。

開発時は、一気に全部のプログラムを作らずに通信部分だけ先に作り、簡易モニターの画面表示を利用しながらさらに作り込んで行きました。簡易モニターは単なるモニターだけでなく簡易デバッガーの役目も果たしてくれます。各々のユニットの SW 操作では、互いに通信を行い、モニター画面に表示が行われるので、１つずつプログラムを追加しながらテストします。

Main ボードのシリアル通信は（簡易モニターも）、PC に接続してターミナルソフトでのテストもできます。コネクタケーブルの断線等の時には、どちらが不具合か切り分けが出来ない場合がありますので、写真の様に、PC 接続で判断します。

PC との接続テスト

PC のシリアルポートにテストコネクタをつけて接続しています。シリアル通信の為のソフトウエアは『TeraTerm』を使用してテストしました。

Mian ボードの SW で DRIVE・CHARGE のテストが出来ます。この SW をクリックしたときの簡易モニターの画面表示と、点灯する LED が、放電回路と充電回路の LED であることが確認出来れば、よいと思います。

５．３　Android タブレット

タブレットは、Bluetooth での通信が主体なので、まず、Main ボードとのペアリングを行い、操作してみます。その操作状態で、簡易モニターにメッセージ表示が適切に行われていれば、動作は概ね間違い有りません。

今回、簡易モニターの SW と同じ機能をタブレットのボタンに持たせましたので、各々のボタンを操作して、Main ボードの動作と簡易モニターのメッセージが正しいことを確認します。

システムが完成すると、Main ボードが電圧計測値を常時送信してくるので、このメッセージがタブレットに正しく表示されて、定期的に更新されることが確認出来れば完成です。

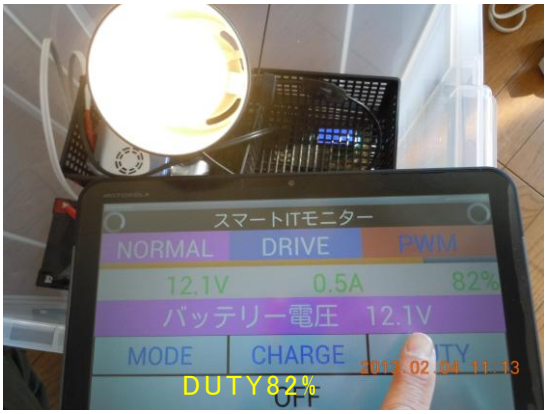
以下、タブレットでシステムを稼働させている様子示します。

DUTY60%

DUTY70%

DUTY80%で点灯

80%から1%刻み



DUTY82%

充電テスト指示

充電テスト駆動

負荷駆動モード

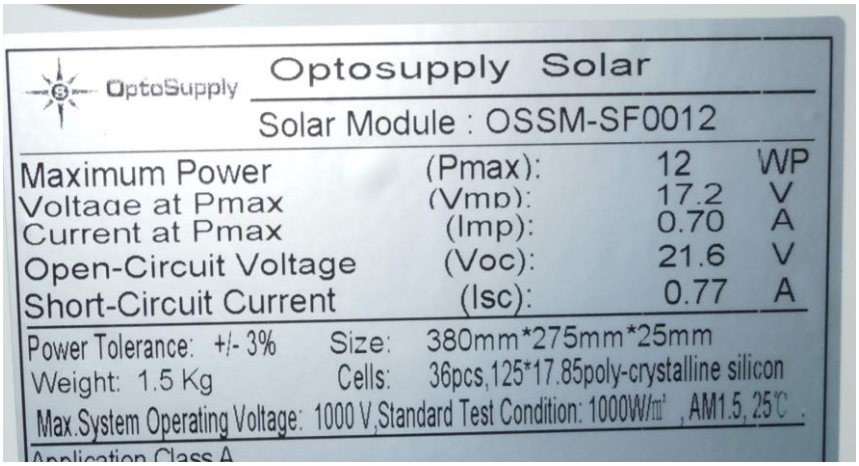

停電検出実験

自動切り替え負荷駆動

６．稼働実験

６．１　充電テスト

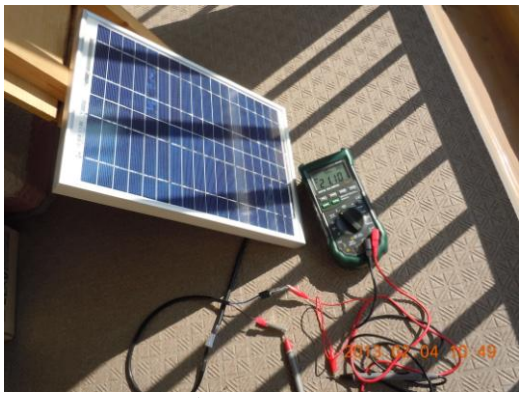
このシステムでは、システムが稼働（通電）している時に、太陽光パネルからの電流をバッテリに流して充電することとなっています。



太陽光パネルの裏面

太陽光パネルの規格は、パネルの裏面シールで確認できます。

システムを組み上げて、充電テスト SW を操作したとき、パネルからバッテリに充電電流が流れている事をテスタで確認します。



開放電圧 21.2V

電流 21.26mA

６．２　放電テスト

放電 SW を操作したとき、Main ボードの LED が点灯し、インバータ接続した AC 電源家電（テストでは、電球型蛍光ランプ）が点灯すれ

ば、問題有りません。　５章で解説の通りです。

６．３　PWM 制御実験

モードが NORMAL のとき、簡易モニターの SW を操作して、放電テストモード（DRIVE モード）にします。このとき放電している事を示す LED が点灯します。数秒後に NORMAL モードに復帰しますので、復帰する前にもう一度 SW をクリックすると、PWM モードに入ります。

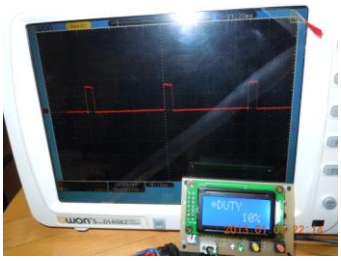
DRIME モード

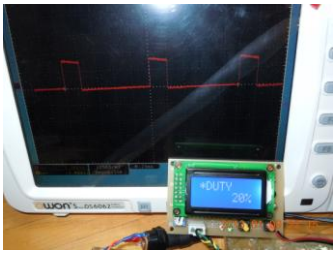
PWM モード

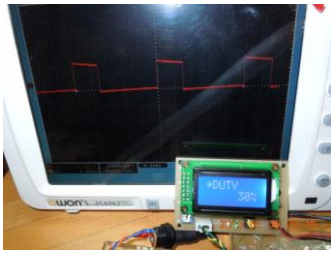
PWM モードになったとき、DUTY が０％で、放電 LED は消灯します。簡易モニターの DUTY 変更 SW を１度クリックすると DUTY が 10%ずつ上昇します。LED は点滅する様になります。

DUTY は 10～80％の間は、10％刻みで変化します。80～95％の間は、1％刻みで変化します。SW を操作して変化する DUTY に対応した信号が出ていることを、出力端子にオシロスコープを接続して確認します。

LED は、95％で連続点灯しているように見えます。







DUTY 変化をオシロスコープで観測

DUTY 変化をオシロスコープで観測

次に、インバータに AC100V のランプを接続して点灯テストを行います。実際に行って見ると、準備したランプは、70%程度で点滅し始めて、やはり 95%で連続点灯しているように見えます。

この状態で 5%の電流カットを行っていますので、その分が節電となる事が確認できます。

環境エネルギーを利用するうえでは、ＰＷＭ制御が重要性を持つことを理解出来る教材となります。

６．４　停電検出

システムを稼働させたうえで、乾電池電源の SW も ON して、放電テストを行います。その際、インバータに接続した電球が点灯することを確認します。

次に、システムに供給している AC アダプタ電源を抜きます。電源が電池電源に切り替わり、インバータに電源供給が始まり、電球が点灯することが確認出来れば、停電検出は問題無く稼働していることになります。

このシステムでは、太陽光パネルを発電源としていますが、考え方を変えて、通常 AC 電源からバッテリに通電して充電をしておき、停電時にバックアップ電源に切り替わることを利用すると、UPS などに代表される無停電電源装置を作ることもできます。

７．教材としての評価

システム全体は、研究を行う際に手作りしていますが、基板を量産し半田付けを省略して、情報系専門学校の教材として利用し易く出来るでしょう。

簡易モニターと Main ボードは、PIC のプログラムを作り替えることで、PWM 制御だけでなく、MPPT 制御（Maximum Power Point Tracking）などを組み込むことができます。この様にして組み込んだ新たな制御技術で、どれほどの効果があるかを、Bluetooth 通信の対向機である Android タブレットで、SD カードなどに保存、グラフ表示するなどのプログラムを開発が、さらに進んだ教材となり得ると考えています。

発展的には、この教材を組み込んだ、『太陽光で走り、AC 電源も供給できる EV（電気自動車）のモデル』を開発して、まさにスマートエネルギーの IT 実験教材にまとめ上げたいと考えます。

この研究は、まだ教材テーマとしては模索が続くことと思われますので、開発者としては次年度以後も継続研究できることを強く望む次第です。

８．反省点

この研究では環境エネルギーの利用に関わる IT 技術者の学習教材を開発して来ましたが、設計・製作・ソフト開発・実験などを通じて得られた反省点を列記します。

◆教材の仕様について

何を教えるべきか、何が必要とされるのかが分からないままに、始めた教材開発ですが、初年度としてはまとまりました。この研究で開発した仕様がベストとは思いませんので、これをたたき台とした、継続研究が必要です。

◆システムとして、停電検出や PWM 制御など、実用性のあるものが組み込めることは分かりましたので、電子・電気系学科と情報（ソフト）系学科でコラボレーションした学習など、多面的な展開を模索することも必要と思われます。

◆製作段階での半田付けなどは必須ですが、これをこなせる情報系学生は少ないと思いますので、キット化する際には、パターン基板の小ロット製造を行うか､または､個別パーツをブロックのように加工して、組合せながら学べる教材とする研究も需要があると思われます。

◆教材を使いこなせる人材育成も必要とおもいます。開発側の意図に沿った教材の利用と工夫で授業内容が高まると思われますので、モデル授業を開催し、教材がどのように使えるかを研究すべきと考えます。

◆ソフト開発については、2 種のマイコンに加え Android タブレットのソフト開発（Basic・C・JAVA）と、かなり工数の掛かる教材開発となりましたが、既存のソフトウエア資産を旨く利用する事で日程内の工程となりました。この様な研究を行う人材も貴重なので、是非継続的な研究を行い、Android のグラフィックスに関する開発に取り組み、EV

車のメータ周りや、現在の Status 表示のモバイル対応などを行えば、さらに魅力有る教材・教材開発研究となるでしょう。

プログラムリスト

Appendix 1. 簡易モニター　プログラムリスト

Appendix 2. Main ボード　プログラムリスト

Appendix 3. Android タブレットプログラムリスト

Appendix 1.　簡易モニター　プログラムリスト

```
program serial_monitor_1st
"2012.12.28 1st version
" by wiseman Co,.Inc. kenichi.harada
" CPU: PIC16F648A
" LCD
" USART with RxIRQ.
" LED
" SW
" CopyRight kenichi-harada


' Declarations section

' Define Constants

' LED Port connection
dim
   LED1 as sbit at RA6_bit
   LED2 as sbit at RA7_bit
   LED_R as sbit at RA6_bit
   LED_G as sbit at RA7_bit

dim
   LED1_Direction as sbit at TRISA6_bit
   LED2_Direction as sbit at TRISA7_bit
' End LED Port connection



' SW Port connection
dim
   SW1 as sbit at RB0_bit
   SW2 as sbit at RB3_bit

dim
   SW1_Direction as sbit at TRISB0_bit
   SW2_Direction as sbit at TRISB3_bit
' End SW Port connection

' Lcd module connections
dim
   LCD_RS as sbit at RA1_bit
```

```
    LCD_EN as sbit at RA0_bit
    LCD_D7 as sbit at RB7_bit
    LCD_D6 as sbit at RB6_bit
    LCD_D5 as sbit at RB5_bit
    LCD_D4 as sbit at RB4_bit

dim
    LCD_RS_Direction as sbit at TRISA1_bit
    LCD_EN_Direction as sbit at TRISA0_bit
    LCD_D7_Direction as sbit at TRISB7_bit
    LCD_D6_Direction as sbit at TRISB6_bit
    LCD_D5_Direction as sbit at TRISB5_bit
    LCD_D4_Direction as sbit at TRISB4_bit
' End Lcd module connections

dim msg as char[8]
        i     as byte                       ' Loop variable
        cbuff as char[9]
        sbuff as char[9]

dim mode as byte
        rxbuf as char[10]
        rxdata as char[10]
        rxreq as byte
        rxindex as integer

dim eep_temp as byte
        h_volt as word
        l_volt as word
        f_volt as byte
        f_volt_l as byte

        eep_check as word


" Selfcheck LED
sub Procedure LED_Blink()
    LED_R = 1
    Delay_ms(500)
    LED_G = 1
    Delay_ms(500)
    LED_R = 0
    Delay_ms(500)
    LED_G = 0
    Delay_ms(500)
```

```
end sub

" Blinking Green LED
sub procedure Blink_LED_G()
   LED_G = 1
   Delay_ms(30)
   LED_G = 0
   Delay_ms(30)
end sub


" Blinking Red LED
sub procedure Blink_LED_R()
   LED_R = 1
   Delay_ms(30)
   LED_R = 0
   Delay_ms(30)
end sub

" Wait for SW1 Off
sub procedure SW1_OFF()
   while SW1=0
      asm
         nop
      end asm
   wend
end sub

" Wait for SW2 Off
sub procedure SW2_OFF()
   while SW2=0
      asm
         nop
      end asm
   wend
end sub


"
sub Procedure SW1_Proc()
   if SW1 = 0 then
      Delay_ms(5)              " Chattering protection.
      if SW1 = 0 then
         rxindex = 0
         Blink_LED_G()
```

```
        SW1_OFF
        select case mode
          case "N"
            LED_R = 1
            Lcd_Out(1,1,"@STATUS ")            ' Write text in first row
            Lcd_Out(2,1," REQUEST")            ' Write text in second row
            UART1_Write_Text("S")              ' Send Command for Status request
            LED_R = 0
            Blink_LED_G()

          case "P"
            LED_R = 1
            Lcd_Out(1,1,"@DUTY     ")          ' Write text in first row
            Lcd_Out(2,1,"    CHANGE")          ' Write text in second row
            UART1_Write_Text("D")              ' Send Command for Duty change
            LED_R = 0
            Blink_LED_G()

          case "D"

        end select
      else
      end if
    end if
end sub


sub procedure SW2_Proc()
  if SW2 = 0 then
    Delay_ms(5)              " Chattering protection.
    if SW2 = 0 then
      rxindex = 0
      Blink_LED_G()
      SW2_OFF

      LED_R = 1
      Lcd_Out(1,1,"@MODE     ")              ' Write text in first row
      Lcd_Out(2,1,"        SET")             ' Write text in second row
      UART1_Write_Text("M")                  ' Send Command for Status request
      LED_R = 0
      Blink_LED_G()

    end if
  end if
end sub
```

```
sub procedure RxData_Proc()
  if rxreq = "*" then
    rxreq=" "
    select case rxdata[0]
      case "U","u"
        for i = 0 to 7
          msg[i] = rxdata[i+2]
        next i

        Lcd_Cmd(_LCD_CLEAR)                    ' Clear display
        Lcd_Out(1,1,msg)              ' Write text in first row

      case "L","l"
        for i = 0 to 7
          msg[i] = rxdata[i+2]
        next i
        Lcd_Out(2,1,msg)              ' Write text in second row

        if strncmp(msg,"   CHARGE",8)=0 then
          mode = "C"
        end if

        if strncmp(msg,"    DRIVE",8)=0 then
          mode = "D"
        end if

        if strncmp(msg,"       PWM",8)=0 then
          mode = "P"
        end if

        if strncmp(msg,"   NORMAL",8)=0 then
          mode = "N"
        end if

        " Threshold High
        if strncmp(msg,"H",1)=0 then
          EEPROM_Write($02, msg[4])            ' High Voltage
          Delay_ms(100)
          EEPROM_Write($03, msg[5])            '
          Delay_ms(100)
          EEPROM_Write($04, msg[6])            '
          Delay_ms(100)
          EEPROM_Write($05, msg[7])            '
          Delay_ms(100)
```

```
            if f_volt="I" then
              f_volt="L"
            end if

          end if

          " Threshold Low
          if strncmp(msg,"L",1)=0 then
            EEPROM_Write($06, msg[4])          ' Low Voltage
            Delay_ms(100)
            EEPROM_Write($07, msg[5])          '
            Delay_ms(100)
            EEPROM_Write($07, msg[6])          '
            Delay_ms(100)
            EEPROM_Write($08, msg[7])          '
            Delay_ms(100)
          end if

      end select
    end if
end sub

'    USART Rx Interrupt
sub procedure interrupt
    ' Check for Rx IRQ.
    if PIR1.5 = 1 then
      rxbuf[rxindex]=RCREG                  ' Reciev 1 character.
      rxindex = rxindex + 1
      if rxindex = 10 then                  ' Rx data length = 10 characters.
        rxindex=0                           ' Reset buffer index
        rxreq = "*"                         ' Set Rxdata opereation flug
        rxdata = rxbuf                      ' Copy Rx data
        rxbuf = "          "                ' Reset Rx data buffer
      end if
    end if
end sub


" EEPROM Initialization
sub procedure EEP_Init()
    EEPROM_Write($00, $AB)          ' Check Digit.
    Delay_ms(100)
    EEPROM_Write($01, $CD)          ' Check Digit.
    Delay_ms(100)
```

```
    EEPROM_Write($02, "0")              ' High Voltage 0615 (12V)
    Delay_ms(100)
    EEPROM_Write($03, "6")              '
    Delay_ms(100)
    EEPROM_Write($04, "1")              '
    Delay_ms(100)
    EEPROM_Write($05, "5")              '
    Delay_ms(100)

    EEPROM_Write($06, "0")              ' Low Voltage 0563 (11V)
    Delay_ms(100)
    EEPROM_Write($07, "5")              '
    Delay_ms(100)
    EEPROM_Write($08, "6")              '
    Delay_ms(100)
    EEPROM_Write($09, "3")              '
    Delay_ms(100)

end sub


sub procedure send_high_threshold()
    '---------------------------------
    " High threshold
    cbuff="High       "
    cbuff[4]=EEPROM_Read($02)
    Delay_ms(100)
    cbuff[5]=EEPROM_Read($03)
    Delay_ms(100)
    cbuff[6]=EEPROM_Read($04)
    Delay_ms(100)
    cbuff[7]=EEPROM_Read($05)
    Delay_ms(100)

    Lcd_Out(1,1,"@THRESH ")             ' Write text in first row
    Lcd_Out(2,1,cbuff)                  ' Write text in second row

    sbuff="H       "
    sbuff[1]=cbuff[4]
    sbuff[2]=cbuff[5]
    sbuff[3]=cbuff[6]
    sbuff[4]=cbuff[7]
    UART1_Write_Text(sbuff)
```

```
end sub

sub procedure send_low_threshold()

   if f_volt="L" then
      f_volt=""
      '---------------------------------
      " Low threshold
      cbuff="Low         "
      cbuff[4]=EEPROM_Read($06)
      Delay_ms(100)
      cbuff[5]=EEPROM_Read($07)
      Delay_ms(100)
      cbuff[6]=EEPROM_Read($08)
      Delay_ms(100)
      cbuff[7]=EEPROM_Read($09)
      Delay_ms(100)

      Lcd_Out(1,1,"@THRESH ")              ' Write text in first row
      Lcd_Out(2,1,cbuff)                   ' Write text in second row

      sbuff="L       "
      sbuff[1]=cbuff[4]
      sbuff[2]=cbuff[5]
      sbuff[3]=cbuff[6]
      sbuff[4]=cbuff[7]
      UART1_Write_Text(sbuff)
   end if

end sub


main:

' Main program
   mode = "N"
   rxdata = "                  "
   rxbuf = "                   "
   rxreq = " "
   rxindex = 0

" Clear Port A and B
   porta=0
   portb=0
```

```
' PortDirections
   CMCON = %00000111
   TRISA = %00111100
   TRISB = %00001111
   SPBRG = 25                          ' USART BAUD RATE Setting. 9600 baud
   RCSTA = %10010000                   ' USART Rx Status and Control
   TXSTA = %00100110                   ' USART Tx Status and Control
   OPTION_REG.7 = 0                    ' Pull up Port B for SW

   " Interrupt Enable
   PIE1 = %00100000
   INTCON = %11000000

   UART1_Init(9600)                    ' USART Initialize with 9600 baud


' Indicates LED Blinking
'----------------------------
   LED_Blink()
   LED_Blink()
   LED_Blink()
'----------------------------


   " LCD Initialize
   '----------------------------
   Lcd_Init()                          ' Initialize Lcd
   Lcd_Cmd(_LCD_CLEAR)                 ' Clear display
   Lcd_Cmd(_LCD_CURSOR_OFF)            ' Cursor off

   " Display Startting Message
   Lcd_Out(1,1,"@MONITOR")             ' Write text in first row
   Lcd_Out(2,1,"     START")           ' Write text in second row

   " Send Power On Command.
   UART1_Write_Text("P")

   " Wait for 2sec.
   Delay_ms(2000)

   " Mode Set Request
   'Lcd_Out(1,1,"@MODE     ")          ' Write text in first row
   'Lcd_Out(2,1,"        SET")         ' Write text in second row

   " Send Mode Set Command.
```

```
'UART1_Write_Text("M")

" EEPROM Check
Lcd_Out(1,1,"@EEPROM ")             ' Write text in first row
Lcd_Out(2,1,"     CHECK")           ' Write text in second row

eep_temp = EEPROM_Read($00)
Delay_ms(100)
eep_check = (eep_temp << 8) or EEPROM_Read($01)
Delay_ms(100)
if eep_check=$ABCD then
   'Lcd_Out(1,1,"@EEPROM ")          ' Write text in first row
   'Lcd_Out(2,1,"          OK")      ' Write text in second row
   Lcd_Out(1,1,"@MONITOR")           ' Write text in first row
   Lcd_Out(2,1,"     START")         ' Write text in second row
else
   Lcd_Out(1,1,"@EEPROM ")           ' Write text in first row
   Lcd_Out(2,1,"     Init.")         ' Write text in second row
   EEP_Init()                        ' EEPROM Initialize
   Lcd_Out(1,1,"@MONITOR")           ' Write text in first row
   Lcd_Out(2,1,"     START")         ' Write text in second row
end if

Delay_ms(1000)

Lcd_Out(1,1,"@MONITOR")             ' Write text in first row
Lcd_Out(2,1,"     START")           ' Write text in second row
Delay_ms(1000)

" Battery threshold: High
f_volt="I"
send_high_threshold()
Delay_ms(1000)

" Main Loop
while TRUE                          ' Endless loop
   " sw1
   SW1_Proc()

   " sw2
   SW2_Proc()

   " Rx Data
   RxData_Proc()
```

```
        " Battery threshold: Low
        send_low_threshold()

    wend

end.
```

Appendix 2.　Main ボード　プログラムリスト

```
/*
Copyright (c) 2012-2013, Kenichi Harada wiseman All rights reserved.
*/

/*
 * main.c
 */

#include <stdlib.h>
#include <stdint.h>

#include "GenericTypeDefs.h"
#include "HardwareProfile.h"
#include "USB/usb.h"
#include "usb_bt_driver.h"

#include <btstack/run_loop.h>
#include <btstack/sdp_util.h>
#include "btstack/src/hci.h"
#include "btstack/src/l2cap.h"
#include "btstack/src/btstack_memory.h"
#include    "btstack/src/rfcomm.h"
#include "btstack/src/sdp.h"
#include "btstack/include/config.h"

#include "bt_spp.h"

#include<ports.h>       // C:¥Program Files¥Microchip¥MPLAB C30¥support¥peripheral_24F¥ports.h

#include<timer.h>

/* Configuration Bits */

#if defined(__PIC24FJ256GB106__)
    _CONFIG1( JTAGEN_OFF & GCP_OFF & GWRP_OFF & COE_OFF & FWDTEN_OFF & ICS_PGx1)
    _CONFIG2( PLL_96MHZ_ON & IESO_OFF & FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMOD_HS
& FNOSC_PRIPLL & PLLDIV_DIV4 & IOL1WAY_ON)
#elif defined(__PIC24FJ64GB002__)
        _CONFIG1(WDTPS_PS1 & FWPSA_PR32 & WINDIS_OFF & FWDTEN_OFF & ICS_PGx1 &
GWRP_OFF & GCP_OFF & JTAGEN_OFF)
        _CONFIG2(POSCMOD_NONE & I2C1SEL_PRI & IOL1WAY_OFF & OSCIOFNC_ON &
FCKSM_CSDCMD & FNOSC_FRCPLL & PLL96MHZ_ON & PLLDIV_DIV2 & IESO_OFF)
```

```
        _CONFIG3(WPFP_WPFP0 & SOSCSEL_IO & WUTSEL_LEG & WPDIS_WPDIS &
WPCFG_WPCFGDIS & WPEND_WPENDMEM)
        _CONFIG4(DSWDTPS_DSWDTPS3 & DSWDTOSC_LPRC & RTCOSC_SOSC & DSBOREN_OFF
& DSWDTEN_OFF)
#endif


// Global Variables

BYTE deviceAddress = 0;

static uint8_t    rfcomm_channel_nr = 1;
static uint8_t    spp_service_buffer[120];


// -------------------------
#define true            1
#define false           0


//typedef unsigned char BOOL;
//typedef unsigned int WORD;

WORD BHighVolt;
WORD BLowVolt;
WORD BVolt;

BOOL BreakNow = false;
BOOL ChargeNow = false;
BOOL Test1 = false;
BOOL Test2 = false;

BOOL TestTime;
BOOL SaveChargeSw;

char mode = 'N';
float A_amp = 0.0;
float B_volt = 0.0;
float AD_val = 0.0;
char rx = ' ';
int ix = 0;
char Vol[6] = "01234";

double Volt0,Volt1;                    // A/D 変換用データバッファ
```

```
int tm_5ms = 0;                         // 5ms timer
int tm_50ms = 0;                        // 50ms timer
int tm_500ms = 0;                       // 500ms timer
int tm_500msbase = 2;                   // 500ms base
int tm_1sec = 0;                        // 1sec timer
int tm_10sec = 10;                      // 10sec timer
int tm_BT = 10;                         // 10sec timer
int tm_AC = 20;                         // 10sec timer
int tm_charge;                          // Charge mode timer
int tm_drive;                           // Drive mode timer
int tm_queu_drive;
int tm_mode;
int tm_status;
int tm_duty;
int tm_volt_h;
int tm_volt_l;


int status = 0;                         //  簡易モニター用ステータスインデックス
int duty = 0;
char duty_f = 'U';
char buf[11] = "0123456789";

char control_on = 'N';



// USB Event
BOOL USB_ApplicationEventHandler ( BYTE address, USB_EVENT event, void *data, DWORD size )
{
    // Handle specific events.
    switch ( (INT)event )
    {
        case EVENT_VBUS_REQUEST_POWER:
            // We'll let anything attach.
            return TRUE;

        case EVENT_VBUS_RELEASE_POWER:
            // We aren't keeping track of power.
            return TRUE;

        case EVENT_HUB_ATTACH:
            return TRUE;
            break;
```

```
            case EVENT_UNSUPPORTED_DEVICE:
                return TRUE;
                break;

            case EVENT_CANNOT_ENUMERATE:
                return TRUE;
                break;

            case EVENT_CLIENT_INIT_ERROR:
                return TRUE;
                break;

            case EVENT_OUT_OF_MEMORY:
                return TRUE;
                break;

            case EVENT_UNSPECIFIED_ERROR:   // This should never be generated.
                return TRUE;
                break;

            case EVENT_SUSPEND:
            case EVENT_DETACH:
            case EVENT_RESUME:
            case EVENT_BUS_ERROR:
                return TRUE;
                break;

            default:
                break;
        }

        return FALSE;

} // USB_ApplicationEventHandler

// USBTask
static int usbhost_process(struct data_source *ds)
{
            // Maintain USB Host State
            USBHostTasks();

            USBHostUSBBTTask();
            return 0;
}
```

```
// SW[1]Task
static int sw1_process(struct data_source *ds)
{
	static short swcount;
	static short lastsw;
	if(swcount == 0)
	{
		short sw = mPORTBReadBit(0x4000);
		if(sw != lastsw)
		{
			if(sw == 0x4000)
			{
				bt_spp_send("S0",2);
			}else{
				bt_spp_send("S1",2);

				control_on = 'N';                                    // 通常制御
停止
				mPORTAClearBits(0x0008);          // 停電回復・放電停止
				BreakNow = false;
				mPORTAClearBits(0x0004);          //  充電 OFF
				ChargeNow=false;
				printf( "U *CONTROL" );
				printf( "L          OFF" );
			}
			swcount = 1500;
			lastsw = sw;
		}
	}
	if(swcount != 0)
	{
		swcount--;
	}
	return 0;
}


// SW[2]Task  強制停電テスト
static int sw2_process(struct data_source *ds)
{
	static short swcount;
	static short lastsw;
	if(swcount == 0)
	{
		short sw = mPORTBReadBit(0x0200);
```

```
		if(sw != lastsw)
		{
			if(sw == 0x0200)
			{
				//mPORTAClearBits(0x0008);		//<---このマクロも <p
ｏｒｔｓ．ｈ> にあります。
				//bt_spp_send("M0",2);
			}else{
				short jp1 = mPORTAReadBit(0x0002);	//JP1 接続か？：low
				short jp2 = mPORTBReadBit(0x0020);	//JP2 接続か？：low


				if(mode=='N' && jp1==0x0002 && jp2==0x0020)	//jp1 jp2
open
				{
					mPORTASetBits(0x0008);
	//<---このマクロも <ｐｏｒｔｓ．ｈ> にあります。
					bt_spp_send("MD",2);
					tm_drive = 5*2+1;				//
5Sec

					mode = 'D';

					printf( "U *MODE     " );
					printf( "L       DRIVE" );
				}else if(jp1==0x0000 && jp2==0x0020){
	//jp1 ショート:バッテリ電圧上限を EEPROM 書き込み
					printf( "U *Battery" );
					printf( "L High%04u",BVolt );
				}else if(jp1==0x0002 && jp2==0x0000){
	//jp2 ショート:バッテリ電圧下限を EEPROM 書き込み
					printf( "U *Battery" );
					printf( "L Low %04u",BVolt );
				}
			}
			swcount = 500;
			lastsw = sw;
		}
	}
	if(swcount != 0)
	{
		swcount--;
	}
	return 0;
}
```

```
// SW[3]Task 強制充電テスト
static int sw3_process(struct data_source *ds)
{
	static short swcount;
	static short lastsw;
	if(swcount == 0)
	{
		short sw = mPORTBReadBit(0x0100);
		if(sw != lastsw)
		{
			if(sw == 0x0100)
			{
				//mPORTAClearBits(0x0004);		//<---このマクロも <p
ｏｒｔｓ．ｈ> にあります。
			}else{
				if(mode=='N')
				{
					mPORTASetBits(0x0004);
	//<---このマクロも <ｐｏｒｔｓ．ｈ> にあります。
					bt_spp_send("MC",2);
					tm_charge = 5*2+1;			//
5Sec

					mode = 'C';

					printf( "U *MODE     " );
					printf( "L     CHARGE" );
				}
			}
			swcount = 500;
			lastsw = sw;
		}
	}
	if(swcount != 0)
	{
		swcount--;
	}
	return 0;
}


// HB LED task ---> 忙しいとき、ＬＥＤが点いている時間が長くなる
```

```
static int led_process(struct data_source *ds)
{
	mPORTAClearBits(0x0010);		//<---HB LED OFF
	return 0;
}



// USART Rx task ---> シリアルポート１から１文字のデータを受信する
static int rx_process(struct data_source *ds)
{
	if(U1STAbits.URXDA == 1)
	{
		// 受信エラーチェック
		if(U1STAbits.OERR || U1STAbits.FERR){
			// エラーのときは、エラーフラグをキャンセルして、
			// ＵＡＲＴを一度停止して、再度有効にします。
			U1STA &= 0xFFF0;		// Error cancel
			U1MODEbits.UARTEN = 0;	// Stop UART1
			U1MODEbits.UARTEN = 1;	// Restart UART1
		}else{
			rx = getchar();				// Recive 1 chara.

			switch(rx)
			{
				case 'S':	// Status Request
					tm_status = 2;
//					control_on = 'G';			// 最初の STATUS 要求
で、システム制御開始 GO とします。
					break;

				case 'M':	// Mode Change Request
					tm_mode = 2;
					break;

				case 'D':	// Duty Change Request
					tm_duty = 2;
					break;

				case 'H':	// High Threshold
					Vol[0]='H';
					ix=1;
					break;
```

```
                                        case 'L':   // Low Threshold
                                                    Vol[0]='L';
                                                    ix=1;
                                                    break;

                                        default:
                                                    Vol[ix]=rx;
                                                    if(ix>=4){
                                                                ix=0;
                                                                if(Vol[0]=='H'){
                                                                            tm_volt_h=2;
                                                                            mPORTBSetBits(0x8000);          //
LED ON
                                                                }else{
                                                                            tm_volt_l=2;
                                                                            mPORTBClearBits(0x8000);        //
LED OFF
                                                                }
                                                    }else{
                                                                ix++;
                                                    }

                                    }// end switch
                        }
            }
            return 0;
}


// メインコントロール　タスク---->充電・放電制御をおこないます。
static int main_control_process(struct data_source *ds)
{
            // main control flug が GO でかつ、mode が Normal のときだけ制御する
            if(control_on == 'G' && mode == 'N'){
                        short ibreak = mPORTBReadBit(0x0080);
                        if(ibreak == 0x0080) // RB7 が 1 のとき停電検出　通常は 0
                        {
                                    mPORTASetBits(0x0008);                  // 放電開始
                                    if(BreakNow==false){
                                                // 停電中！
                                                BreakNow = true;
                                                bt_spp_send("BD",2);            // 停電検出メッセージ　--->
Android
                                                printf( "U *BREAK    " );
```

```
                    printf( "L     DETECT" );
                }
            }else{
                mPORTAClearBits(0x0008);      // 停電回復・放電停止
                if(BreakNow==true)
                {
                    BreakNow = false;
                    bt_spp_send("BR",2);              // 停電回復メッセージ  --->
Android
                    printf( "U *BREAK    " );
                    printf( "L RECOVERE" );
                }
            }

            // バッテリ電圧を上限値・下限値と比較して、充電制御をおこないます。
            if(BVolt < BLowVolt){
                //   VL よりバッテリ電圧が低いとき
                mPORTASetBits(0x0004);                          //   充電 ON
                if(ChargeNow==false){
                    // 充電中になったときだけ、メッセージを送ります
                    ChargeNow=true;
                    bt_spp_send("CD",2);              // 充電開始メッセージ  --->
Android
                    printf( "U *CHARGE " );
                    printf( "L          ON" );
                }
            }else if(BVolt > BHighVolt){
                //   VH よりバッテリ電圧が高いとき
                mPORTAClearBits(0x0004);              //   充電 OFF
                if(ChargeNow==true){
                    // 充電完了になったときだけ、メッセージを送ります
                    ChargeNow=false;
                    bt_spp_send("CR",2);              // 充電終了メッセージ  --->
Android
                    printf( "U *CHARGE " );
                    printf( "L          OFF" );
                }
            }

        }

        return 0;
}
```

```
// main
int main ( void )
{
        // Initialize the processor and peripherals.

        // Init Clock
        // CPU 32MHz
        // Peripheral 8MHz
    CLKDIV = 0x0000;                    // クロック　1:1　分周


        unsigned int pll_startup_counter = 600;
        CLKDIVbits.PLLEN = 1;
        while(pll_startup_counter--);


    /// ポート割り当て


        AD1PCFG = 0xFFCF;                                   // ポート A4-5 をアナログに設定
        // Analog IN Disable            // 元の Bluetooth プログラムでは Disable だった。
        //AD1PCFG = 0xffff;


        TRISA = 0xFFE3;                     // RA2-4 を出力に指定。LED 用
        TRISB = 0x7FEF;                     // RB15,4 を出力に指定。
                                                                    // RB15:LED
                                                                    // RB4:Tx として使う。
<---RP4 で使用するので、この指定は要らないかも知れない。
        //mPORTAOutputConfig(0xFFE3);       //<---このマクロも <ｐｏｒｔｓ．ｈ> にあります。
        //mPORTBOutputConfig(0x7FEF);       //<---このマクロも <ｐｏｒｔｓ．ｈ> にあります。

        // プルアップは、<ports.h> にマクロがあり、ピンごとにプルアップ、プルダウンの指定が可能。
        // 詳細は、main.c の先頭部分で #include している部分に、ｐｏｒｔｓ．ｈの在処があるので
        // そちらを参照してください。
        // SW 用 PIN プルアップ
        EnablePullUpCN12;  // SW プルアップ
        EnablePullUpCN21;
        EnablePullUpCN22;
        EnablePullUpCN23;  // 停電検出プルアップ

        // JP 用 PIN プルアップ
        EnablePullUpCN3;
```

```
    EnablePullUpCN27;



    /// シリアルポートの指定
// U1RX を RP13 (pin 23) に指定
    RPINR18bits.U1RXR = 13;

    // U1TX を RP4 (pin 11) に指定
    RPOR2bits.RP4R = 3;


    // Init UART
    U1BRG = 103;                    // 9600bps @ 8MHz
    U1MODE = 0b1000100000000000;        // USART1 初期設定
    U1STA =  0b0000010000000000;


    // Timer 2
    T2CON = 0b1000000000110000;
    PR2 = 32149;        // 500ms
    IPC1bits.T2IP = 5;              // IQR Level 5.
    IEC0bits.T2IE = 1;              // 割り込み許可



    // Timer 3
    T3CON = 0b1000000000110000;
    PR3 = 3125;         // 50ms
    //IPC2bits.T3IP = 6;              // IQR Level 6.
    //IEC0bits.T3IE = 1;              // 割り込み許可


    // Timer 4
    T4CON = 0b1000000000110000;
    PR4 = 312;// 5ms
    IPC6bits.T4IP = 6;              // IQR Level 6.
    IEC1bits.T4IE = 1;              // 割り込み許可



    // ADC
    AD1CON1 = 0x8044;           // AD オン、整数、タイマー３トリガー、自動サンプリング
    //AD1CON2 = 0x2414;                 // 外部 Vref+, AVss, 自動スキャン、
```

```
AD1CON2 = 0x0414;               // AVdd, AVss, 自動スキャン、
AD1CON3 = 0x1F05;               // 31Tad, ad ,5*Tcy
AD1CHS = 0x0000;                // 自動スキャン
AD1PCFG = 0xFFCF;                       // RA4,5 Analog
AD1CSSL = 0x0030;               // AN4,5 自動スキャン
IEC0bits.AD1IE = 1;             // AD 割り込み許可


BHighVolt=615;                          // デフォルト上限値 12V
BLowVolt=563;                           // 下限値 11V



//for test Brink LED
//for test Brink LED
//for test Brink LED
mPORTBSetBits(0x8000);                  //<---このマクロも <ｐｏｒｔｓ．ｈ> にあります。
DelayMs(250);
mPORTBClearBits(0x8000);                //<---このマクロも <ｐｏｒｔｓ．ｈ> にあります。
DelayMs(250);
mPORTBSetBits(0x8000);
DelayMs(250);
mPORTBClearBits(0x8000);
DelayMs(250);
mPORTBSetBits(0x8000);
DelayMs(250);
mPORTBClearBits(0x8000);
DelayMs(250);
mPORTBSetBits(0x8000);
DelayMs(250);
mPORTBClearBits(0x8000);
DelayMs(250);
mPORTBSetBits(0x8000);
DelayMs(250);
mPORTBClearBits(0x8000);
DelayMs(250);
mPORTBSetBits(0x8000);
DelayMs(250);
mPORTBClearBits(0x8000);
DelayMs(250);
mPORTBSetBits(0x8000);
DelayMs(250);
mPORTBClearBits(0x8000);
DelayMs(250);
mPORTBSetBits(0x8000);
```

```
        DelayMs(250);
        mPORTBClearBits(0x8000);
        DelayMs(250);
        mPORTBSetBits(0x8000);
        DelayMs(250);
        mPORTBClearBits(0x8000);
        //for test Brink LED
        //for test Brink LED
        //for test Brink LED



        printf( "U MAIN CPU" );
        printf( "L >> START" );


        mPORTASetBits(0x001C);
        DelayMs(250);
        mPORTAClearBits(0x001C);
        DelayMs(250);
        mPORTASetBits(0x001C);
        DelayMs(250);
        mPORTAClearBits(0x001C);
        DelayMs(250);


/*
        while(1)
{
        if( mPORTBReadBit(0x4000) == 0x0000 )
        {
                mPORTASetBits(0x001C);
      printf( "U 12345678" );
                while(mPORTBReadBit(0x4000) == 0x0000);
        }else{
                mPORTAClearBits(0x001C);
        }
}
*/

        // Init USB
        if ( USBHostInit(0) != TRUE )
    {
                mPORTASetBits(0x0010);
                DelayMs(250);
```

```
                    mPORTAClearBits(0x0010);
                    DelayMs(250);
                    mPORTASetBits(0x0010);
                    DelayMs(250);
                    mPORTAClearBits(0x0010);
                    DelayMs(250);
                    mPORTASetBits(0x0010);
                    DelayMs(250);
                    mPORTAClearBits(0x0010);
                    DelayMs(250);

                    printf( "U USB HOST" );
                    printf( "L      ERROR" );
                    //while (1);
    }

        /// GET STARTED with BTstack ///
        btstack_memory_init();
    run_loop_init(RUN_LOOP_EMBEDDED);

    // init HCI
        hci_transport_t      * transport = hci_transport_usb_instance();
        bt_control_t         * control   = NULL;
    hci_uart_config_t   * config      = NULL;
    remote_device_db_t * remote_db = NULL;//(remote_device_db_t *) &remote_device_db_memory;
        hci_init(transport, config, control, remote_db);

        // ここは、Bluetooth を PIC SPP で使うときの、データパケットの扱いを初期化しています。
    // init L2CAP
    l2cap_init();
    l2cap_register_packet_handler(bt_packet_handler);

    // init RFCOMM
    rfcomm_init();
    rfcomm_register_packet_handler(bt_packet_handler);
    rfcomm_register_service_internal(NULL, rfcomm_channel_nr, 100);   // reserved channel, mtu=100

    // init SDP, create record for SPP and register with SDP
    sdp_init();
        memset(spp_service_buffer, 0, sizeof(spp_service_buffer));
    service_record_item_t * service_record_item = (service_record_item_t *) spp_service_buffer;
    sdp_create_spp_service( (uint8_t*) &service_record_item->service_record, 1, "SPP");
//      printf("SDP service buffer size: %u¥n¥r", (uint16_t) (sizeof(service_record_item_t) + de_get_len((uint8_t*)
&service_record_item->service_record)));
    sdp_register_service_internal(NULL, service_record_item);
```

```
        // usbhost Bluetooth 受信タスクを登録
        data_source_t usbhost;
        usbhost.process = &usbhost_process;
        run_loop_add_data_source(&usbhost);


        // Bluetooth 通信テストタスク（SW1 でタブレット側 LED 表示制御）を登録（送信）
        data_source_t sw1task;
        sw1task.process = &sw1_process;
        run_loop_add_data_source(&sw1task);


        // 強制駆動テストタスクを登録
        data_source_t sw2task;
        sw2task.process = &sw2_process;
        run_loop_add_data_source(&sw2task);


        // 強制充電テストタスクを登録
        data_source_t sw3task;
        sw3task.process = &sw3_process;
        run_loop_add_data_source(&sw3task);


        // ハートビート LED タスクを登録
        data_source_t ledtask;
        ledtask.process = &led_process;
        run_loop_add_data_source(&ledtask);


        // シリアル受信タスクを登録
        data_source_t rxtask;
        rxtask.process = &rx_process;
        run_loop_add_data_source(&rxtask);


        // メイン制御タスクを登録
        data_source_t controltask;
        controltask.process = &main_control_process;
        run_loop_add_data_source(&controltask);


// go!
run_loop_execute();
```

```
    return 0;
} // ------------------------------------------------------- main


/*
  AD 割り込み処理 2ch 3 回　変換後平均を計算
*/
void __attribute__((interrupt, auto_psv)) _ADC1Interrupt(void)
{
        IFS0bits.AD1IF = 0;             // AD 割り込みフラグクリア
        Volt0 = ((ADC1BUF0 + ADC1BUF2 + ADC1BUF4)*20.)/3072.0;      // AD4:バッテリ電圧
        Volt1 = (ADC1BUF1 + ADC1BUF3 + ADC1BUF5)*51.2/3072;         // AD5:ＡＣ電流(1bit=0.05A)

        BVolt = (ADC1BUF0 + ADC1BUF2 + ADC1BUF4)/3;                 // EEPROM 書き込み用のバ
イナリ値をとっておく

                                                                    // この値で、バッテリの充電制御もおこなう
        if(BVolt >= 1024){BVolt=1024;}
        // AD 値リミット閾値を設けることができる（任意の値で比較する）
}

//
//  タイマー１割り込み処理
//    ---> システムで使用している
//
//void __attribute__((interrupt, no_auto_psv)) _T1Interrupt(void)
//{
//}

//  タイマー２割り込み処理
//    --->タイマー２割り込みでＨＢ　ＬＥＤを点灯し、タスクで消灯する。
//        main loop 処理が忙しいとき、点いている時間が長くなる
//
void __attribute__((interrupt, no_auto_psv)) _T2Interrupt(void)
{
        IFS0bits.T2IF = 0;          // 割り込みフラグ クリア
        mPORTASetBits(0x0010);      // HB LED 点灯


        if(tm_500ms >0){tm_500ms--;}

        if(tm_volt_h>0)
        {
                tm_volt_h--;
```

```
		if(tm_volt_h==0)
		{
			// Recieve high threshold
			strcpy(buf,"4321");
			buf[0] = Vol[1];			// Recive 1 chara.
			buf[1] = Vol[2];			// Recive 1 chara.
			buf[2] = Vol[3];			// Recive 1 chara.
			buf[3] = Vol[4];			// Recive 1 chara.
			buf[4] = '¥n';
			BHighVolt = atoi(buf);		// String to value


// デフォルトで制御する（異常な値にされると、充電制御がおかしくなり
// 見た目では PG エラーか、壊れたのか、設定値が正しくないのかの、判断がつかない）
BHighVolt=615;				// デフォルト上限値 12V
BLowVolt=563;				// 下限値 11V



			printf( "U *Battery" );
			printf( "L High%4u",BHighVolt );
		}
}


if(tm_volt_l>0)
{
		tm_volt_l--;
		if(tm_volt_l==0)
		{
			// Recieve high threshold
			strcpy(buf,"4321");
			buf[0] = Vol[1];			// Recive 1 chara.
			buf[1] = Vol[2];			// Recive 1 chara.
			buf[2] = Vol[3];			// Recive 1 chara.
			buf[3] = Vol[4];			// Recive 1 chara.
			buf[4] = '¥n';
			BLowVolt = atoi(buf);		// String to value

// デフォルトで制御する（異常な値にされると、充電制御がおかしくなり
// 見た目では PG エラーか、壊れたのか、設定値が正しくないのかの、判断がつかない）
BHighVolt=615;				// デフォルト上限値 12V
BLowVolt=563;				// 下限値 11V
```

```
                printf( "U *Battery" );
                printf( "L Low %4u",BLowVolt );
        }
}




if(tm_charge>0)
{
        tm_charge--;
        if(tm_charge==0)
        {
                // Stop Charge
                mPORTAClearBits(0x0004);
                bt_spp_send("MN",2); // Send to Android by Bluetooth

                mode = 'N';

                printf( "U *MODE     " );
                printf( "L     NORMAL" );
        }
}


if(tm_drive>0)
{
        tm_drive--;
        if(tm_drive==0)
        {
                // Stop Drive
                mPORTAClearBits(0x0008);
                bt_spp_send("MN",2); // Send to Android by Bluetooth

                mode = 'N';

                printf( "U *MODE     " );
                printf( "L     NORMAL" );
        }
}


if(tm_status > 0)
{
```

```
tm_status--;
if(tm_status==0)
{
        //
        status++;
        switch(status)
        {
                case 1:
                        printf( "U *Battery" );
                        printf( "L %7.1fV",Volt0 );
                        break;
                case 2:
                        printf( "U *AC Amp." );
                        printf( "L %7.1fA",Volt1 );
                        break;
                case 3:
                        printf( "U *Battery" );
                        printf( "L %7.1fV",Volt0 );
                        break;
                case 4:
                        printf( "U *AC Amp." );
                        printf( "L %7.1fA",Volt1 );
                        break;
                case 5:
                        printf( "U *Battery" );
                        printf( "L %7.1fV",Volt0 );
                        break;
                case 6:
                        printf( "U *AC Amp." );
                        printf( "L %7.1fA",Volt1 );
                        break;
                case 7:
                        printf( "U *STATUS " );
                        printf( "L       007" );
                        break;
                case 8:
                        printf( "U *STATUS " );
                        printf( "L       008" );
                        break;
                case 9:
                        printf( "U *STATUS " );
                        printf( "L       009" );
                        break;
                case 10:
                        control_on = 'G';                       // 最初の 10 回目
```

STATUS 要求で、システム制御開始 GO とします。

```
                                        status = 0;
                                        printf( "U *STATUS " );
                                        printf( "L          010" );
                                        break;
                        }

                }
        }



        if(tm_mode> 0)
        {
                tm_mode--;
                if(tm_mode==0)
                {
                        if(mode=='N'){
                                mode = 'D';

                                mPORTASetBits(0x0008);                  //<---このマ
クロも <ｐｏｒｔｓ．ｈ> にあります。
                                bt_spp_send("MD",2);
                                tm_drive = 5*2+1;                       // 5Sec
                                printf( "U *MODE     " );
                                printf( "L       DRIVE" );

                        }else if(mode=='D'){
                                mode = 'P';
                                duty = 0;
                                duty_f = 'U';
                                tm_drive=0;
                                mPORTAClearBits(0x0008);                //<---このマ
クロも <ｐｏｒｔｓ．ｈ> にあります。
                                bt_spp_send("MP",2);

                                printf( "U *MODE     " );
                                printf( "L          PWM" );
                        }else if(mode=='P'){
                                mode = 'N';

                                mPORTAClearBits(0x0008);                //<---このマ
クロも <ｐｏｒｔｓ．ｈ> にあります。
                                bt_spp_send("MN",2);
```

```
                                        printf( "U *MODE     " );
                                        printf( "L     NORMAL" );
                                }
                        }
                }


if(tm_duty > 0)
{
                tm_duty--;
                if(tm_duty==0)
                {
                        if(mode=='P'){

                                        if(duty<80 && duty_f=='U'){
                                                duty+=10;
                                        }else if(duty>=80 && duty<95 && duty_f=='U'){
                                                duty+=1;
                                        }else if(duty==95 && duty_f=='U'){
                                                duty=94;
                                                duty_f='D';
                                        }else if(duty<=94 && duty>80 && duty_f=='D'){
                                                duty--;
                                        }else if(duty<=80 && duty>0 && duty_f=='D'){
                                                duty-=10;
                                        }else if(duty==0 && duty_f=='D'){
                                                duty=10;
                                                duty_f='U';
                                        }

                                        sprintf(buf,"DU%2u%%",duty);
                                        bt_spp_send(buf,5);

                                        printf( "U *DUTY     " );
                                        printf( "L        %3u%%",duty );
                                }
                        }
                }


// 500ms base
if(tm_500msbase > 0){tm_500msbase--;}
if(tm_500msbase==0)
{
                tm_500msbase = 2;     // 1sec
```

```
                if(tm_1sec >0){
                        tm_1sec--;
                        if(tm_1sec==0){tm_1sec=1;}
                }
                if(tm_10sec >0){
                        tm_10sec--;
                        if(tm_10sec==0){tm_10sec=0;}
                }
                if(tm_BT > 0){
                        tm_BT--;
                        if(tm_BT==0){
                                tm_BT=20;
                                sprintf(buf,"BA%4.1fV",B_volt);
                                bt_spp_send(buf,7);
                                B_volt = Volt0;
        // 送った直後に最新の値にしておく
                                }
                }
                if(tm_AC > 0){
                        tm_AC--;
                        if(tm_AC==0){
                                tm_AC=20;
                                sprintf(buf,"AC%4.1fA",A_amp);
                                bt_spp_send(buf,7);
                                A_amp = Volt1;
        // 送った直後に最新の値にしておく
                                }
                }
        }
}


//
//  タイマー３割り込み処理
//      ---> ＰWMで使用している
//
void __attribute__((interrupt, no_auto_psv)) _T3Interrupt(void)
{
        IFS0bits.T3IF = 0;                      // 割り込みフラグ クリア
        tm_50ms++;
        if(tm_50ms>=20){
                tm_50ms=0;
        }

//      if(mode=='P')
```

```
//              {
//                      if(tm_50ms==0){mPORTBSetBits(0x8000);}
//                      if(tm_50ms ==(duty/5)){
//                              mPORTBClearBits(0x8000);
//                      }
//              }
}

//
//   タイマー４割り込み処理
//     ---> ＰＷＭで使用している
//
void __attribute__((interrupt, no_auto_psv)) _T4Interrupt(void)
{
        IFS1bits.T4IF = 0;                      // 割り込みフラグ クリア
        tm_5ms++;
        if(tm_5ms>=20){
                tm_5ms=0;
        }

        if(mode=='P')
        {
                if(tm_5ms==0){mPORTASetBits(0x0008);}
                if(tm_5ms ==(duty/5)){
                        mPORTAClearBits(0x0008);
                }
        }
}
```



```
/*
 * bt_spp.c
 */

#include <string.h>
```

```
#include <stdio.h>

#include <btstack/hci_cmds.h>
#include <btstack/run_loop.h>
#include <btstack/sdp_util.h>

#include "btstack/src/hci.h"
#include "btstack/src/l2cap.h"
#include "btstack/src/btstack_memory.h"
#include "btstack/src/remote_device_db.h"
#include "btstack/src/rfcomm.h"
#include "btstack/src/sdp.h"
#include "btstack/include/config.h"

#include "bt_spp.h"

#define USE_AND_OR /* To enable AND_OR mask setting */
#include<ports.h>


// Global Variables in main.c
// -------------------------
#define true          1
#define false         0
extern char mode;
extern int tm_mode;
extern int tm_status;
extern int tm_duty;
extern int tm_charge;
extern BOOL BreakNow;
extern BOOL ChargeNow;
extern char control_on;
// Global Variables in main.c



static uint16_t    rfcomm_channel_id;

uint16_t bt_localid;
char bt_localname[10] = {'P','I','C','B','T','0','0','0','0',0x00};
char send_buf[17];

/* SPP RECIVE DATA    */
int bt_spp_recive_callback(uint16_t channel, uint8_t *packet, uint16_t size)
{
```

```
//          static char a = 0;
            packet[size] = 0;
            switch(*(packet+0))
            {
                        case 'l':
                        case 'L':
                                    if(*(packet+1) == '1')
                                    {
                                                mPORTBSetBits(0x8000);
                                    }else{
                                                mPORTBClearBits(0x8000);
                                    }
                                    control_on = 'N';                                   // 通常制御停止
                                    mPORTAClearBits(0x0008);                            // 停電回復・放電停止
                                    BreakNow = false;
                                    mPORTAClearBits(0x0004);                             //   充電 OFF
                                    ChargeNow=false;
                                    break;

                        case 'S':  // Status Request
                                    tm_status = 2;
                                    break;

                        case 'M':  // Mode Change Request
                                    tm_mode = 2;
                                    break;

                        case 'D':  // Duty Change Request
                                    tm_duty = 2;
                                    break;


                        case 'C':  // Duty Change Request

                                    if(mode=='N')
                                    {
                                                mPORTASetBits(0x0004);                          //<---この
マクロも <ｐｏｒｔｓ．ｈ> にあります。
                                                bt_spp_send("MC",2);
                                                tm_charge = 5*2+1;                              // 5Sec

                                                mode = 'C';

                                                printf( "U *MODE     " );
                                                printf( "L     CHARGE" );
```

```
                    }
                    break;


        }
        return 0;
}
```

Appendix 3.　Android タブレットプログラムリスト

```
package com.hrdapp.android.BluetoothIO;

//import android.bluetooth.BluetoothAdapter;
//import android.bluetooth.BluetoothDevice;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
//import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.widget.CompoundButton;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ToggleButton;
import android.widget.Button;
import android.widget.ProgressBar;

public class BluetoothIOActivity extends BluetoothBaseActivity implements
ToggleButton.OnCheckedChangeListener{

    // Layout Views
    private TextView mTitle;
    private TextView cTitle;
    private TextView mSWStatus;
    private ToggleButton mToggleButton;
    private Button mButton1;            // モード変更ボタン
    private Button mButton2;                    // 停電テストボタン
    private Button mButton3;                    //　充電テストボタン
    private ProgressBar PBar;           // プログレスバー
    private ProgressBar PBar2;          // プログレスバー
    private ProgressBar PBar3;          // プログレスバー
    private TextView ModeNormal;        // ノーマルモード表示
    private TextView ModeDrive;                 // ドライブモード表示
    private TextView ModePWM;                   // PWM モード表示
    private TextView Voltage;           // バッテリ電圧
    private TextView Ampere;            // ＡＣ電流
    private TextView Duty;                      // Duty
```

```
//private final static int WC=LinearLayout.LayoutParams.WRAP_CONTENT;



@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set up the window layout
    requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
    setContentView(R.layout.main);
    getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.custom_title);

    // Set up the custom title
    mTitle = (TextView) findViewById(R.id.title_left_text);
    mTitle.setText(R.string.app_name);
    mTitle = (TextView) findViewById(R.id.title_right_text);


    cTitle = (TextView) findViewById(R.id.cTitle);
    cTitle.setHeight(80);
    cTitle.setTextSize(60);


    mSWStatus = (TextView) findViewById(R.id.SWStatus);
    mSWStatus.setHeight(100);
    mSWStatus.setTextSize(80);
    mSWStatus.setBackgroundColor(getResources().getColor(R.color.magenta));

    mToggleButton = (ToggleButton)findViewById(R.id.LEDtoggleButton);
    mToggleButton.setChecked(false);
    mToggleButton.setOnCheckedChangeListener(this);
    mToggleButton.setHeight(120);
    mToggleButton.setTextSize(80);

    mButton1 = (Button)findViewById(R.id.button1);
    mButton1.setTag("button1");
    mButton1.setOnClickListener(new BtnClickListener());
    mButton1.setHeight(120);
    mButton1.setWidth(430);
    mButton1.setTextSize(75);
    mButton1.setTextColor(getResources().getColor(R.color.blue));

    mButton2 = (Button)findViewById(R.id.button2);
```

```
mButton2.setTag("button2");
mButton2.setOnClickListener(new BtnClickListener());
mButton2.setHeight(120);
mButton2.setWidth(430);
mButton2.setTextSize(75);
mButton2.setTextColor(getResources().getColor(R.color.blue));

mButton3 = (Button)findViewById(R.id.button3);
mButton3.setTag("button3");
mButton3.setOnClickListener(new BtnClickListener());
mButton3.setHeight(120);
mButton3.setWidth(430);
mButton3.setTextSize(75);
mButton3.setTextColor(getResources().getColor(R.color.blue));

PBar    = (ProgressBar) findViewById(R.id.progressBar1);
PBar.setMax(100);
PBar.setProgress(0);
PBar.setSecondaryProgress(80);
//PBar.setBackgroundColor(getResources().getColor(R.color.lime));


PBar2    = (ProgressBar) findViewById(R.id.progressBar2);
PBar2.setMax(100);
PBar2.setProgress(0);
PBar2.setSecondaryProgress(80);

PBar3    = (ProgressBar) findViewById(R.id.progressBar3);
PBar3.setMax(100);
PBar3.setProgress(0);
PBar3.setSecondaryProgress(80);



ModeNormal = (TextView) findViewById(R.id.textView1);
ModeNormal.setHeight(100);
ModeNormal.setWidth(430);
ModeNormal.setTextSize(80);
ModeNormal.setBackgroundColor(getResources().getColor(R.color.magenta));
ModeNormal.setTextColor(getResources().getColor(R.color.blue));

ModeDrive = (TextView) findViewById(R.id.textView2);
ModeDrive.setHeight(100);
ModeDrive.setWidth(430);
ModeDrive.setTextSize(80);
```

```
        ModeDrive.setBackgroundColor(getResources().getColor(R.color.amethyst));
        ModeDrive.setTextColor(getResources().getColor(R.color.white));

        ModePWM = (TextView) findViewById(R.id.textView3);
        ModePWM.setHeight(100);
        ModePWM.setWidth(430);
        ModePWM.setTextSize(80);
        ModePWM.setBackgroundColor(getResources().getColor(R.color.coral));
        ModePWM.setTextColor(getResources().getColor(R.color.white));

        Voltage = (TextView) findViewById(R.id.textView4);
        Voltage.setHeight(100);
        Voltage.setWidth(430);
        Voltage.setTextSize(80);
        Voltage.setBackgroundColor(getResources().getColor(R.color.almond));
        Voltage.setTextColor(getResources().getColor(R.color.lime));

        Ampere = (TextView) findViewById(R.id.textView5);
        Ampere.setHeight(100);
        Ampere.setWidth(430);
        Ampere.setTextSize(80);
        Ampere.setBackgroundColor(getResources().getColor(R.color.almond));
        Ampere.setTextColor(getResources().getColor(R.color.lime));

        Duty = (TextView) findViewById(R.id.textView6);
        Duty.setHeight(100);
        Duty.setWidth(430);
        Duty.setTextSize(80);
        Duty.setBackgroundColor(getResources().getColor(R.color.almond));
        Duty.setTextColor(getResources().getColor(R.color.lime));




        if(!initBT())
        {
                Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).show();
                finish();
        }
    }


    class BtnClickListener implements OnClickListener
    {
```

```
    //@Override
    public void onClick(View v){

      String tag=(String)v.getTag();

      if (tag=="button1"){
            sendMessage("M");
            //Toast.makeText(v.getContext(),tag,Toast.LENGTH_SHORT).show();
            mSWStatus.setText("モード変更ボタンがおされました");

      }else if (tag=="button2"){
            sendMessage("C");
            //Toast.makeText(v.getContext(),tag,Toast.LENGTH_SHORT).show();
            mSWStatus.setText("充電テストぼたんがおされました");

      }else if (tag=="button3"){
            sendMessage("D");
            //Toast.makeText(v.getContext(),tag,Toast.LENGTH_SHORT).show();
            mSWStatus.setText("DUTY 変更ボタンがおされました");

      }

    }

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.option_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.connect:
        // Launch the DeviceListActivity to see devices and do scan
        Intent serverIntent = new Intent(this, DeviceListActivity.class);
        startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
        return true;
    case R.id.disconnect:
      stopBTIO();
        return true;
    }
```

```
        return false;
    }

    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        switch (requestCode) {
        case REQUEST_CONNECT_DEVICE:
            // When DeviceListActivity returns with a device to connect
            if (resultCode == Activity.RESULT_OK) {
                // Get the device MAC address
                String address = data.getExtras()
                                        .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
                connectionBT(address);
            }
            break;
        case REQUEST_ENABLE_BT:
            // When the request to enable Bluetooth returns
            if (resultCode == Activity.RESULT_OK) {
                // Bluetooth is now enabled, so set up a io session
                setupBTIO();
            } else {
                // User did not enable Bluetooth or an error occured
                Toast.makeText(this, R.string.bt_not_enabled_leaving, Toast.LENGTH_SHORT).show();
                finish();
            }
        }
    }

    @Override
    public void onChangeConnectionStatus(int status) {
                switch (status) {
                case BluetoothBaseService.STATE_CONNECTED:
                    mTitle.setText(R.string.title_connected_to);
                    mTitle.append(getmConnectedDeviceName());
            Toast.makeText(getApplicationContext(), "Connected to "
                      + getmConnectedDeviceName(), Toast.LENGTH_SHORT).show();
                    break;
                case BluetoothBaseService.STATE_CONNECTING:
                    mTitle.setText(R.string.title_connecting);
                    break;
                case BluetoothBaseService.STATE_LISTEN:
                case BluetoothBaseService.STATE_NONE:
                    mTitle.setText(R.string.title_not_connected);
                    break;
                }
    }
```

```
        // Tauch Button and chenge status
    public void onCheckedChanged(CompoundButton _tbutton,boolean _checked){
                // ＬＥＤ　ＯＮ／ＯＦＦボタンが押されました。
                if(_checked){
                        // Send message LED ON Command
        sendMessage("L1");
        Toast.makeText(this,"ＬＥＤ　ＯＮです",Toast.LENGTH_SHORT).show();
                }else{
                        //Send message LED OFF command
        sendMessage("L0");
        Toast.makeText(this,"ＬＥＤ　ＯＦＦです",Toast.LENGTH_SHORT).show();
                }
        }

        // with Bluetooth rechive message from PIC
    @Override
    public void onReviceMessage(String message)
    {
        if(message.equals("S1")){
                // SW ON Command
                mSWStatus.setText(R.string.SW_ON);
                Toast.makeText(this,"ＳＷが押されました",Toast.LENGTH_SHORT).show();

        }else if(message.equals("S0")){
                // SW OFF command
                mSWStatus.setText(R.string.SW_OFF);
                Toast.makeText(this,"ＳＷが離されました",Toast.LENGTH_SHORT).show();

        }else if(message.substring(0,2).equals("MD")){
                // Mode change to Drive
          ModeNormal.setTextColor(getResources().getColor(R.color.white));
          ModeDrive.setTextColor(getResources().getColor(R.color.blue));
          ModePWM.setTextColor(getResources().getColor(R.color.white));
                //Toast.makeText(this,"モードが　ＤＲＩＶＥ　になりました
",Toast.LENGTH_SHORT).show();
          mSWStatus.setText("モード　ＤＲＩＶＥ　です");

        }else if(message.substring(0,2).equals("MN")){
                // Mode change to Normal
          ModeNormal.setTextColor(getResources().getColor(R.color.blue));
          ModeDrive.setTextColor(getResources().getColor(R.color.white));
          ModePWM.setTextColor(getResources().getColor(R.color.white));
                ModePWM.setText("PWM");
                Duty.setText("0%");
```

```
                    PBar.setProgress(0);
                    //Toast.makeText(this,"モードが　ＮＯＲＭＡＬ　になりました
",Toast.LENGTH_SHORT).show();
                mSWStatus.setText("モード　ＮＯＲＭＡＬ　です");

            }else if(message.substring(0,2).equals("MP")){
                    // Mode change to PWD
                ModeNormal.setTextColor(getResources().getColor(R.color.white));
                ModeDrive.setTextColor(getResources().getColor(R.color.white));
                ModePWM.setTextColor(getResources().getColor(R.color.blue));
                    //Toast.makeText(this,"モードが　ＰＷＭ　になりました",Toast.LENGTH_SHORT).show();
                mSWStatus.setText("モード　ＰＷＭ　です");

            }else if(message.substring(0,2).equals("MC")){
                    // Charge On
                    ModePWM.setText("CHARG");
                ModeNormal.setTextColor(getResources().getColor(R.color.white));
                ModeDrive.setTextColor(getResources().getColor(R.color.white));
                ModePWM.setTextColor(getResources().getColor(R.color.blue));
                    //Toast.makeText(this,"充電テスト中です",Toast.LENGTH_SHORT).show();
                mSWStatus.setText("充電テスト中です");

            }else if(message.substring(0,2).equals("DU")){
                // Duty value
                    Duty.setText(message.substring(2));
                    PBar.setProgress(Integer.parseInt(message.substring(2,4).replace(" ", "0"))); //プログレスバ
ーの値変更
                    PBar2.setProgress(Integer.parseInt(message.substring(2,4).replace(" ", "0"))); //プログレス
バーの値変更
                    PBar3.setProgress(Integer.parseInt(message.substring(2,4).replace(" ", "0"))); //プログレス
バーの値変更
                    mSWStatus.setText(message);
                    //Toast.makeText(this,"ＤＵＴＹが変更されました　" +
message.substring(2),Toast.LENGTH_SHORT).show();
                mSWStatus.setText("ＤＵＴＹが変更されました　" + message.substring(2));

            }else if(message.substring(0,2).equals("BA")){
                // Battery voltage
                    Voltage.setText(message.substring(2));
                    //Toast.makeText(this,"バッテリー電圧　" +
message.substring(2),Toast.LENGTH_SHORT).show();
                mSWStatus.setText("バッテリー電圧　" + message.substring(2));

            }else if(message.substring(0,2).equals("AC")){
                // AC current
```

```
                    Ampere.setText(message.substring(2));
                    //Toast.makeText(this,"ＡＣ電流　" +
message.substring(2),Toast.LENGTH_SHORT).show();
            mSWStatus.setText("ＡＣ電流　" + message.substring(2));

        }else if(message.substring(0,2).equals("BD")){
            // 停電検出　放電開始
                    Toast.makeText(this,"停電です！！",Toast.LENGTH_SHORT).show();
            mSWStatus.setText("停電検出しました。　放電しています。　" + message.substring(2));

        }else if(message.substring(0,2).equals("BR")){
            // 停電回復　放電停止
                    Toast.makeText(this,"停電回復！！",Toast.LENGTH_SHORT).show();
            mSWStatus.setText("停電回復しました。　通常制御しています。　" + message.substring(2));

        }else if(message.substring(0,2).equals("CD")){
            // 停電検出　放電開始
                    Toast.makeText(this,"バッテリ電圧低下！！",Toast.LENGTH_SHORT).show();
            mSWStatus.setText("バッテリの充電を開始しました。　" + message.substring(2));

        }else if(message.substring(0,2).equals("CR")){
            // 停電回復　放電停止
                    Toast.makeText(this,"電圧回復！！",Toast.LENGTH_SHORT).show();
            mSWStatus.setText("充電終了しました。　通常制御しています。　" + message.substring(2));

        }
    }
}
```

平成 24 年度文部科学省委託　「成長分野等における中核的専門人材養成の戦略的推進事業」
環境・エネルギー分野のスマートグリッドエンジニア育成の調査研究プロジェクト

■実施委員会

◎	鳥居　高之	船橋情報ビジネス専門学校　校長
	古賀　稔邦	日本電子専門学校　校長
	岡田　靖志	浜松情報専門学校　教務課長
	竹原　伸	近畿大学　工学部　知能機械工学科　教授
	大串 卓矢	株式会社 スマートエナジー　代表取締役
	池澤　寿弘	コスモライフ株式会社　代表取締役社長
	原田　賢一	有限会社ワイズマン　代表取締役
	鳥海　豊彦	株式会社コラボレート研究所　代表取締役
	渡辺　登	株式会社アフレル　エディケーション・プランナー／事業企画室　室長
	飯塚　正成	一般社団法人全国専門学校情報教育協会　事務局長

■調査

◎	鳥居　高之	船橋情報ビジネス専門学校　校長
	原田　賢一	有限会社ワイズマン　代表取締役
	松川　恵美	株式会社グリッド&ファイナンス・アドバイザーズ　取締役副社長
	吉岡　正勝	有限会社ザ・ライスマウンド　マーケティングマネージャー

■開発

◎	鳥居　高之	船橋情報ビジネス専門学校　校長
	原田　賢一	有限会社ワイズマン　代表取締役
	松川　恵美	株式会社グリッド&ファイナンス・アドバイザーズ　取締役副社長
	吉岡　正勝	有限会社ザ・ライスマウンド　マーケティングマネージャー

平成 24 年度文部科学省委託
「成長分野等における中核的専門人材養成の戦略的推進事業」
環境・エネルギー分野のスマートグリッドエンジニア育成の調査研究プロジェクト

スマートグリッドエンジニア育成教材

平成 25 年 3 月

代表校
学校法人三橋学園　船橋情報ビジネス専門学校
〒273-0005　千葉県船橋市本町 7-12-16

問合せ先　有限会社ザ・ライスマウンド
〒164-0003　東京都中野区東中野 1-57-8　辻沢ビル 3F
電話：03-5332-5080　FAX　03-5332-5083