PIC18F14K50 + MAX6675使用K型熱電対モジュール

前回Arduinoで動作テスト(「MAX6675使用K型熱電対モジュールをArduinoで動作テスト」)だけしたMAX6675使用K型熱電対モジュールを、今度はPIC18F14K50でテストしてみた。
参考にしたのはこの記事。
追記(2021/11/22)
□コードをもらっておいた(PIC18F13K50_USB_Temp_LCD-master)。

PIC18F13K50を使っているが、兄弟チップなのでピン・アサインは同じだ。
`main.c` の `#include <pic18f13k50.h>` を `#include <pic18f14k50.h>` に変更するだけで動く。

LCDと、さらにUSB経由でシリアル出力もしてくれるので、TeraTermでも温度を表示できる。
USBのドライバが認識されず、たぶんそこで止まっているためにLCDにも表示されずで、少々戸惑った。
何のことはない、USBのD+とD-が逆につながれていた。

トーチであぶって200℃オーバーまで測定できることは確認したが、それが正しいのかどうか、確認する術は無い。
あと、難を言えばレスポンスが悪いというか、加熱をやめてから室温に戻るのに結構時間がかかる。
もともとこういうものなのかもしれない。

> *K*型熱電対
> ＋極側：ニッケルおよびクロムを主とした合金*(*クロメル*)*
> −極側：ニッケルを主とした合金*(*アルメル*)*
> 温度範囲：*-200*℃~*1000*℃
> 特徴：温度と熱起電力との関係が直線的であり、工業用として最も多く使用されている。

「PIC16F1503 or PIC16F1827 + MAX6675使用K型熱電対モジュール」も参照のこと。

□□□□

PIC18F14K50 + USB + PC + HID Mouse (2) ちょっとだけコードを追ってみた

「PIC18F14K50 + USB + PC + HID Mouse (MLA v2015-08-10)」の続き。

あまりにも簡単に動いたので、少しだけコードを眺めてみた。
まあ、PICをやり始めてから見よう見まねでCをやりだした私には、「チンプンカンプン」という表現がぴったりの状態だ。

それでもmain()からはじまるくらいの事はわかるのでたどってみる。

すると、USBの接続待ちとか認識待ちのための処理なんだろうなと思われるコードを除けば、それ以外にずっと繰り返し実行しているのは

SYSTEM_Tasks();

なので、これを眺めてみる。

[crayon-6717472d5d6ab237541719/]

ざっくりと、ボタンが押されたかどうかをチェックして、トグルでマウスを動かすか止めるかを設定しているらしいことがわかる。

mouse.movementMode = !mouse.movementMode;

このmouse.——というのは何だろな。

[crayon-6717472d5d6b3889062622/]

「構造体」というものらしい。
いろいろ検索した結果、http://www9.plala.or.jp/sgwr-t/c/sec15.htmlやhttp://www9.plala.or.jp/sgwr-t/c/sec16.htmlで教わった限りでは、

[crayon-6717472d5d6b5201257632/]

のどちらも同じことらしく、型名が長くならないように下のように書くらしい。

いずれにしてもこのmouse.ではじまる構造体の要素の値をいろいろ変えればマウスの操作ができるということだろう。
この近辺に実際に実行する行が見つからないのは、たぶん割り込みをかけて定期的に割り込みルーチンから実行しているんだろう。
そちらは後回しにして、値をいろいろ変えて遊んでみることはできる。
例えば、元の行をコメントアウトして、xVector[]とmouseReport.buttons.button2の値を以下のように変更すると、

[crayon-6717472d5d6b7149855825/]

元々は正方形だった動く範囲が横長の長方形になり、マウスの右ボタンが押された状態になる。

マウスから手を離していてもマウスの任意のボタンを押しっぱなしに出来るというのは、アプリケーションによっては使い道がある。
例えば、動画の編集で使う「MPEGEnc MPEG Smart Render 4」などは、サムネイル上で右クリックしたままでいると早送り状態が継続し、左右のマウスの位置で送る速度が変えられる仕様だ。

これが、ものによっては結構長い間右ボタンを押し続けたままのこともあるので、代わりにPICにやらせたら楽ができて便利かもしれない。

上の動画、マウスには触れてないけどPICで右ボタンを押し続けてるので、モニター上では動画が早送りされている。

最近テレビ番組の質がすっかり落ちて、早送りで観れば十分なケースがほとんどなので、これはいいかも。

気になる場面を通過したときにタクトスイッチを押せば早送りが停止する。

app_device_mouse.c の172行目あたりを

```
//static const uint8_t xVector[]={ -1, 0, 1, 0};
static const uint8_t xVector[]={ 0, 0, 0, 0};
//static const uint8_t yVector[]={ 0, -1, 0, 1};
static const uint8_t yVector[]={ 0, 0, 0, 0
```

と書き変えれば矩形を描く動作をしなくなり、353行目あたりを

```
mouseReport.buttons.button1 = 0;
//mouseReport.buttons.button2 = 0;
mouseReport.buttons.button2 = 1;
mouseReport.buttons.button3 = 0;
```

と書き変えればマウスの右ボタンが押された状態になる。

変更点はたったこれだけだ。

PIC18F14K50 + USB + PC + HID Mouse (MLA v2015-08-10)

「PIC18F14K50 + USB + PC + Tera Term (3) MLA v2015-08-10 でふたたび」の続きで、HID Mouseもやってみる。

D:¥直下にMLA v2015-08-10をインストールすると

D:¥microchip¥mla¥v2015_08_10¥apps¥usb¥device¥hid_mouse¥firmware¥MPLAB.X

にあるプロジェクトだ。

MPLAB Xに読み込んでBuildしてPIC18F14K50のボードに書き込んで、何も問題なくそのまま動く。

具体的に何に使うかというアイデアがわかないので、まだソースをちゃんと追ってないが、動きとしては下の動画のようになる。

電源投入時(USBとの接続時)にマウスカーソルのあったところを基点として正方形を描くように動き続ける。

ボード上のスイッチを押すと停止、再びスイッチを押すとまたそこを基点として正方形を描き続ける。

PIC18F14K50 + USB + PC + Tera Term (4) Projectを整頓する

「PIC18F14K50 + USB + PC + Tera Term (3) MLA v2015-08-10 でふたたび」の続き。

Buildも出来たし、PICへの書き込みも出来たし、意図したとおりに動作しているが、何せ、HDD上でのProjectの階層が深すぎる。

D:¥microchip¥mla¥v2014_07_22¥apps¥usb¥device¥cdc_basic¥firmware¥MPLAB.X

さらに言えば、Project内に余分な(と思われる)フォルダとかがたくさんある。

ので、少し整頓したいなと思う。

ヘッダ・ファイルとかの階層関係や位置が良く分からないので、とりあえずPackage化する。

すると、

D:¥microchip¥mla¥v2014_07_22¥apps¥usb¥device¥cdc_basic¥firmware¥MPLAB.X

以下に、

USB Device – CDC – Basic.zip

が作成されるので、これを最上層(ここではD:¥)に移動して解凍すれば、

D:¥USB Device – CDC – Basic¥src¥apps¥usb¥device¥cdc_basic¥firmware¥MPLAB.X

と、少しだけ階層が浅くなる。

これをMPLAB X IDEで開いて、今度は余分なフォルダをRemoveしていく。

例えばこんな感じだ。

基本的に、low_pin_count_usb_development_kitやPIC18F14K50を残して他はRemove From Projectを行う。

もう一度Package化すると、さっきは463KBだったファイルが149KBとコンパクトになる。

新しいPackage内には、いまRemoveされたファイルやフォルダが含まれなくなるからだ。

ヘッダ・ファイルとかの参照関係は、

D:¥USB Device – CDC – Basic¥src¥apps¥usb¥device¥cdc_basic¥firmware¥MPLAB.X¥nbproject¥configurations.xml

に書かれているので、フォルダを整理してこれを書き換えれば良さそうだが、今日のところはここまでにしておく。

追記

結局、configurations.xmlも編集して、こんなすっきりした階層構造にした。

プロジェクトのフォルダ内にすべてのソースが入っている方が何かと管理しやすいので。

PIC18F14K50 + USB + PC + Tera Term (3) MLA v2015-08-10 でふたたび

「PIC18F14K50 + USB + PC + Tera Term (2)」が去年の1月のことなので、当然、もう忘れている。

v2013-06-15での記事だったが、最新はv2015-08-10となっているので、もらってきてインストールする。

どうせすぐに新しいバージョンが出てリンク切れになるので、http://www.microchip.com/pagehandler/en-us/devtools/mla/home.html を載せておく。

D:¥にインストールすると、目的のプロジェクトは、

D:¥microchip¥mla¥v2014_07_22¥apps¥usb¥device¥cdc_basic¥firmware¥MPLABX

に存在する。

安全のために

D:¥microchip¥mla¥v2014_07_22¥apps¥usb¥device¥cdc_basic

を丸ごとコピーしてフォルダ名を変更したものをキープしておく。

MPLAB Xに読み込んで、そのまま難なくビルドできて、PICにも書き込めて、TeraTermとのやり取りも問題なく、すんなり完了してしまった。

前回の印象よりももっと簡単だった気がする。

上は、「abcde」+「スペース」とキー入力したあと「sahara」と打って、基板上のタクトスイッチを５回押してから、再び「sahara」と打ったところ。

ちゃんとやり取りできている。

基板は「PIC18F14K50 + AT24C256B on ブレッドボード配線パターンタイプ基板」で作成したものだ。

「PIC18F14K50 + USB + PC + Tera Term (4) Projectを整頓する」へ続く

PIC18のdelayで「in-line delay argument too large」って叱られる件

__delay大好きな私にとって、表題の件は重要なので調べてみた。

MicrochipのForumで、リミットがあるんだというような書き込みがあったので、XC8のUsr’s Guideを開いて見てみると、

Chapter 3. How To’s
3.5 GETTING MY APPLICATION TO DO WHAT I WANT
- How Can I Implement a Delay in My Code?

という、そのものずばりの内容が書いてあった。

> *If an accurate delay is required, or if there are other tasks that can be performed during the delay, then using a timer to generate an interrupt is the best way to proceed.*
> *If these are not issues in your code, then you can use the compiler’s in-built delay pseudo-functions: _delay, __delay_ms or __delay_us; see Appendix A. Library Functions. These all expand into in-line assembly instructions or a (nested) loop of instructions which will consume the specified number of cycles or time. The delay argument must*

> *be a constant and less than approximately 179,200 for PIC18 devices and approximately 50,659,000 for other devices.*
> *Note that these code sequences will only use the NOP instruction and/or instructions which form a loop. The alternate PIC18-only versions of these pseudo-functions, e.g., _delaywdt, can use the CLRWDT instruction as well. See also, Appendix A. Library Functions.*

正確なdelayが必要なときや、delay中に他の事をやらせたいときにはタイマー割り込みを使ってね。

_delay, __delay_ms そして __delay_usってのがあって、インライン・アセンブラに展開されて所定のサイクルを消費することで時間を稼ぐよ。

delayの引数は、**PIC18**では**179200**、その他では50659000より少ない定数が指定できるよ。

NOPとループで時間稼ぎしているだけなんだって覚えておいてね。

PIC18では_delaywdtという関数も使えるよ。

> *__delaywdt_ms(x) // request a delay in milliseconds*
> *__delaywdt_us(x) // request a delay in microseconds*
> *On PIC18 devices only, you can use the alternate WDT-form of these functions, which uses the CLRWDT instruction as part of the delay code. See the _delaywdt function.*

とは言っても、別に179200のリミットが外れるわけでは無いようで、

For very large delays, call this function multiple times.

と書いてある。

まあ、そうだろうね。

さて、では、エラーにならないようにするには、179200ってどう計算すればいいのかということになる。

__delay_msや__delay_usは、結局内部で__delay();に換算していて、その際の値が179200を越えたらダメっていう意味だと思う。

仮に8MHzで__delay_ms(100);とやって、NOP();だけで消化しようとすると、100/1000秒間に8MHzのPICは何サイクル分仕事するかと考えればいい。

8MHzは1秒間に8000000/4サイクル仕事をするので、これを1/10すると200000となって179200を超えてしまう。

__delay_ms(89);だと、(8000000/4)×(89/1000)=178000となって超えずに、計算上も実際もエラーにならない。

ただ、現実にはNOPでやったりloopでやったりだと思うので、User’s Guideにも「approximately」となっていて、試してみたら__delay_ms(98);はコンパイルできた、計算上は196000なのに。

せっかく高機能なんだから、Timer割り込み使えよ。

割り込みでやるほどのことも無い単純作業だけならもっと低機能のPIC使えよ。

Microchipさん的に言えば、こういうことでしょう。

PIC18F14K50 ADC --> PWM --> LED

PIC18F14K50 で PWM の覚書き

データ・シートに、

Period = 4 * TOSC * (PR2 + 1) * (TMR2 Prescale Value)

Pulse Width = TOSC * (CCPR1L<7:0>:CCP1CON<5:4>) * (TMR2 Prescale Value)

とあるので、

Duty Cycle = Pulse Width / Period = [CCPR1L<7:0>:CCP1CON<5:4>] / [4 * (PR2 + 1)]

となる。

一方で、PR2を求める計算式は、

PR2 = (8MHz / (PWM Freq. * 4 * TMR2 Prescale Value)) – 1

となって、PWM Freq. = 1kHz という条件では、PR2 = 124 とするのが適当となる。

以上から、

Duty Cycle = [CCPR1L<7:0>:CCP1CON<5:4>] / 500

なので、PR2が決まるとCCPR1L<7:0>:CCP1CON<5:4>の値は0~500までの値を取るように設定することになる。

CCP1CON<5:4>は設定が面倒なので0を入れておくとするとCCPR1L<7:0>には0~125までの値を入れることになる。

つまり、00000000|00から1111101|00ということ。

このCCPR1Lの値に10ビットのAD変換値を用いるとすると、AD変換値は0~1023の値の範囲なので、8で割って0~128の範囲にしてからCCPR1Lに入れる必要がある。

ADCのリファレンス電圧が3.3Vの場合に、約0.25VごとにADCによる値でPWMのデューティー比を決めてLEDを光らせてみた連続写真。

PIC18F14K50 + AT24C256B on ブレッドボード配線パターンタイプ基板 不調?

「PIC16F1823 + LM61BIZ + NJL7502L + I2C EEPROM プリント基板バージョン」が完成したので、久しぶりに温度・照度の測定を始めようかと思い、データ回収に使う「PIC18F14K50 + AT24C256B on ブレッドボード配線パターンタイプ基板」を持ち出してきて、EEPROMのデータをPCに送ってみた。

ドライバーのことも忘れていたし(「PIC18F14K50 + USB + PC + Tera Term (2)」参照)、TeraTermのデバッグ・モードのことも忘れていた(「PIC18F14K50 + AT24C256B -> USB -> Tera Term」)。

そんなこんなで、やっと接続してデータをPC上のTeraTermに送ってみると、データが全部FFになってしまう。
とにかく、TeraTermの窓にFFがダーっと並ぶだけということになる。
３つの測定ボードで実際にしばらく測定したあとの３つのEEPROMのデータが全部FFのみになり、「PIC12F675 外付けEEPROM消去器」を使って、手持ちの４つのEEPROMすべてを0で埋めてみても、４つとも全部FFしか返さない。

つまり、「PIC18F14K50 + AT24C256B on ブレッドボード配線パターンタイプ基板」がおかしい可能性が濃厚ということになった。

ダメ押しに、Arduinoを持ち出してRead/Write Serial EEPROM via I2CだったかAdding External I2C EEPROM to Arduino (24LC256)だったかのスケッチの例でEEPROMの動作を確認し、ArduinoのシリアルモニタのかわりにTeraTermを使っても全く問題なかった。

「PIC18F14K50 + AT24C256B on ブレッドボード配線パターンタイプ基板」が原因だとほぼ断定できたので、しぶしぶ作成時の回路図を持ち出してきて、DMMで導通チェックをやってみたが、特に問題は見つからなかった。
記録によると9月16日に使ったのが最後なので、３ヶ月も放置されてスネているのかもしれないし、ハンダ付けが劣化した可能性もある。
スネているかどうかはさておいて、こんなときの最後のアガキといえば、ハンダの溶かし直しと補充だ。
事実、これで直ることとって結構あるんだよね。
で、すべてのハンダ付け部分にこてを当ててもう一度溶かしつつ、ハンダが少なめかなと思われるところにはハンダを補充するという作業を行った。
再度EEPROMを接続してデータを読んでみた。

直った。

良く考えたら、DMMでの導通テストって、テスター棒で接点にある程度の力を加えた状態での導通なんだなと思う。

テスター棒を離してフリーになっても導通が保たれるかはわからないというわけだ。

目視で問題の場所を見つけ出して対処したというわけでなく、全部を総なめでハンダし直した結果なので、何か今後に生かせる新しい発見があったわけではないが、まあ、直ってよかった。

AVRにちょっかいを出してみる　-下調べ編-

PIC18F14K50で検索していたら、どこをどうたどったかAVRのライターをPIC18F14K50で作成できるというサイトに巡り会った。

http://hp.vector.co.jp/authors/VA000177/html/FrontPage.html

http://www-ice.yamagata-cit.ac.jp/ken/senshu/sitedev/index.php?AVR%2Fpic18spx01

実際にたどり着いた上記URLのリンク元はここ。

http://www.geocities.jp/kuman2600/o19pic18spx.html

そもそもPIC18F14K50で検索していたくらいなのでPIC18F14K50はもう持ってるし、回路図を見たところではいっさいの出費無しで手持ちのパーツだけでAVRのライターが作れそうだった(　もちろん書き込む対象のAVRが１つは必要だが　)。

そりゃあ話のタネにやってみるかという気にもなる。

世間ですばらしいと評判のAVRとやらを体験してみるのも後学のためには良いかも知れない。

とりあえず必要なのはAVRなので、どれにしようか秋月詣でをしてみるわけだが、上のサイトでは　ATmega88　というのに書き込んでるし、そもそもAVRが流行り出したきっかけであるArduinoに載ってるのは　ATmega328　というのらしい(※１)。

というか、AVRをやる気なんて無かったんで、いまのところ成り行きで目にしたこの２つくらいしか知らない。

秋月で探すと下の３つが該当する。

ATMEGA88-20PU 170円

ATMEGA88V-10PU 150円

ATMEGA328P-PU 250円

328は無くて328Pだけど、まあ同じなんだろう。

88-20PUと88V-10PUはどう違うかというと、秋月には低電圧駆動がVだと書いてある。

一応、データシートで確認してみるか。

どれも28pinなのでAtmelのProduct Finderへ行って28pinという条件で絞り込むと、

３つとも出てこない。

で、32pinの20MHzという、秋月に書いてあるのと全然違う絞り込みでやっと出てきた。

おい、大丈夫かAtmel、それともおいらが何か勘違いしてるか?

クロック数は分からないにしても、どう数えても秋月の写真では28pinなんだけど。

データ・シートを見ることで88の謎は解けた。

10、20はクロック数、PUは28pin DIPのことのようだ。

で、ATmega88にするならそんなに低電圧でなくてもいいので速い方のATmega88-20PUでいいなという結論に。

そして、pin数の謎は、たぶんTQFPだと32-leadで、28-pinはPDIPだけだからということだろう。

ま、データ・シートにはちゃんと28-pin、32-padという風に区別して書いてあるんだけどね。

ATmega328P-PNのPの意味はデータ・シートを斜めに読んだ限りでは見つけられなかったが(※2)、ま、秋月にはこれしかないんで。

結論

ATmega88-20PU か ATmega328P-PN を買う。

きっと1回こっきりしか使わないから安い88の方でいいとも思う。

一方で、PICで未知のセンサーを使うときの参考にArduinoでの接続例が役に立ったこともあるんだよな。

そうなるとArduinoがらみで328の環境は持っててもいいかなとは思う。

これがPICだったらこんなに悩まずもうとっくに両方とも2個ずつくらいはポチってるに違いないのだが。

※1

よく知らなかったがいろんなArduinoがあるらしく、現在秋月で扱われているものに限ると以下の3つのAVRが搭載されている。

ATMEGA8

ATMEGA168

ATMEGA328

※2

ATmega328P-PNのPはどうやらpicoPowerのPではないかと思う。

picoPowerとは省電力機能のことようだが、まだ斜めにしか読んでいないので自信は無い。

PIC18F14K50 + AT24C256B on ブレッ

ドボード配線パターンタイプ基板

秋月にブレッドボード配線パターンタイプの基板が売っている。

要はブレッドボーで仮組みした回路をそのまま基板にハンダ付けすることができるもので、一度ブレッドボードで動作確認したらそれを見ながら機械的に作っちゃおうということだ。

何が良いのかというと、部品の配置を考える手間や配線経路を考える手間や配線図を描く手間が一切必要無い点だ。

ちなみに、普通のユニバーサル基板に組むと、秋月のCタイプの基板の半分くらいを使えば済むところなので、ずいぶんゆったりとした配置ということになる。

表面が白いので見た目は若干おしゃれな仕上がりではある。

この回路は以前にも書いたが、MicrochipのLow Pin Count USB Development Kitのユーザー・ガイドの回路図から必要だと思われる部分だけを抜粋したものに、I2C接続のEEPROM用の回路を付け加えたもの(写真左の８ピンのICソケットの周辺)だ。

パーツ一覧

ICソケット２個	20円
PICマイコン PIC18F14K50	170円
USBコネクタ(Ｂタイプ、メス)	50円
カーボン抵抗６本	6円
セラミックコンデンサー	10円
セラミック発振子 コンデンサ内蔵タイプ １２ＭＨｚ	20円
タクトスイッチ	10円
ピンヘッダ (オスＬ型) ４本分	5円
赤色LED 2個	20円
分割ロングピンソケット ４ピン分	8円
片面ガラス・ユニバーサル基板(ブレッドボード配線パターンタイプ)	80円
合計	399円

ちなみにLow Pin Count USB Development Kitは39.99ドルだ。