USB IR Toy v2 用のPythonプログラムが実行できるようにする。

拡張子がpyのファイルはPythonだということはわかる。

Pythonの環境をインストールしてそこから実行するんだということもわかる。

その程度の知識から、実際に実行できるまでの紆余曲折を書く。

1.何でも最新がいいんだろうと思って、3.5.1をインストールした。

2.pythonそのものが実行できない。

3.パスを通すのを忘れていたので通す。

4.簡単なコードで試す方がいいだろうと思って、Hello, World!を表示するだけのプログラムをどこかで聞きかじってきて実行するも、実行できない。

5.3と2とでは結構違いがあって、2to3とかで変換しなきゃだめらしいと知る。

6.変換するのも完全ではないだろうと思って、3をやめて2をインストールし直すことにする。

7.Python 2.7.12をインストールする。

8.実行すると、import wx でエラーが出る。

9.wxPythonという、PythonでGUIアプリケーションを作成するためのツールキットが必要らしい。

10.Pythonが2.7なのにwxPython 2.6をもらってきておかしいことになって、2.7用をもらい直してやり直す。

11.import serial でエラーが出て、調べるとpyserialが必要だとわかる。

12.python -m pip install pyserial とやれば勝手にインストールしてくれると知り、その通りにする。

pipは、外部ライブラリをインストールするPythonのコマンドだ。

13.やっと実行できた。

リモコンの信号を読み取って、表示し、グラフを描くプログラムのようだ。

PIC18F2550 - USB IR Toy v2 を作っ

てみた。 -

USB IR Toy v2 は、名前の通り、USBでPCと接続して、赤外線関連で遊ぶモジュールのようだ。

いろいろできるようだが、とりあえず、赤外線リモコンの信号を読み取って、再現することは出来た。

回路図が公開されているのでそのまま組んでみようとしたが、手持ちのパーツだけで試すつもりなので、以下の点が異なる。

クロックはクリスタルでなくセラロックを使った。

赤外線LED用の増幅はC1815を使った。

赤外線リモコン受信モジュールはPL-IRM2161-XD1を使った。

赤外線LEDはOSI5LA5113Aを使った。

QSE159は代替品も無かったので取り付けてない。

抵抗は手持ちの中から一番近い値の物で代用した。

もちろん、ブレッドボード上に組む。

コマンド・プロンプトからirtoy.exe用に書いたバッチ・ファイルを実行して、手持ちの扇風機のリモコンの信号を読み込んでみた。

この際に作成されたtest_000.binを読み込んで、ブレッド・ボード上の赤外線LEDから信号を出力させ、実際に扇風機が所定の動作をすることが確認できた。

最後に、若干エラーっぽい表示が出ているが、とりあえず動作確認できたと言っていいだろう。

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。

Real Time Clock Module (DS1307 + AT24C32) + PIC18F2550 プリント基板作成

「Real Time Clock Module (DS1307 + AT24C32) + PIC18F2550 + SC1602B」や「16×2 Character LCD with I2C Module の分離」でもちょっと触れている時計回路のプリント基板を作成して組み終わった。

相変わらずのなんちゃって表面実装なので基板の部品面には何も無く、大嫌いなドリル作業も電源用のピンヘッダの穴を２つあけただけだ。

表面実装用のパーツは半固定抵抗１個とチップ抵抗２個だけで、あとは普通のアキシャルリードなカーボン抵抗、足を開いて短くしたタクトスイッチなど、ユニバーサル基板だった頃のパーツの消費に努めている。

実は同時に作ったIchigoJam Clone(「255円(税込み、送料別)ぽっきりでIchigoJamを試す。(プリント基板に組んだ)」)では63個もの穴をあけて辟易したので、穴の２個しかないこの基板の作成は本当に楽だなあと実感した。

表面実装パーツでコンパクトにするというよりは、とにかく極力穴をあけたくないという方針だ。

DIP用のICソケットも足を逆方向に折り曲げて表面実装しているし、本来は穴をあけて立てて実装するピンソケットも寝かして取り付けている。

今回はまったのは、まず、中華製の安いピンヘッダのハンダの乗りが異常に悪くて導通に苦労したこと。

もうひとつは表面実装用の半固定抵抗だ。

これ。

まさかこんなに小さいとは思ってもみなかったので、実際にエッチングが済んで、取り付けるパーツを揃えてみてはじめて小ささを実感した始末。

基板上のランドはこんな。

案の定、うまくハンダ付けできていなくて、一通りの導通チェックでも見つからなくて苦労した。

バックライトは点灯する>表示はされない>コントラストを調整する>ドライバーを半固定抵抗に当ててまわす>時々ちらちら表示されるという状況なので、つまり半固定抵抗に上から圧を掛けると表示するので、ハンダつけの不良と判明した。

幾度か経験しているが、導通テストって、テスト棒で上から押さえたときだけ導通するというケースは導通OKにしてしまうという問題がある。

今回も典型的なこのケースだった。

Real Time Clock Module (DS1307 + AT24C32) + PIC18F2550 + SC1602B

Real Time Clock Module は「Real Time Clock Module (DS1307 + AT24C32) + Arduino UNO + Arduino IDE 1.0.5-r2」でArduinoを使って動作確認したっきり放置してあったもの。

これをPIC18F2550で動かしてみた。

I2CはBit Bangingで実装したいところだが、まずは物真似なので、I2Cを装備しているPICを使う。

それに加えてLCD用にI/Oが６本用意できれば他のPICでも動くとは思う。

参考にしたのはこのサイト。

上のサイトで使っているPICはPIC18F2620というものだが、手持ちになかったので同じF18で28ピンの物ってことで、PIC18F2550にしてみた。

あと、LCDへの表示に MPLAB C Compiler Libraries for PIC18 MCUs に含まれている LCD Function を使っているのを「PIC16F84A + SC1602BS + XC8」で使ったものに変更した。

PIC18F2620を使っている元の回路ではRB0~RB3をLCDのデータ用に割り当てているが、PIC18F2550ではRB0とRB1にI2C用のSDAとSCLが割り当てられていて、I2Cを優先するためにLCD用には使えない。

ところが LCD Function でピンの割り当てを変更する方法がよく分からなかったので、使い慣れたコードを流用することにした。

表示部分を該当するものに機械的に置き換えるだけなのでそれほど面倒ではない。

電源関係を除く配線は以下のとおり。

RB4~RB7 D4~D7(SC1602B)
RB2 Enable(SC1602B)
RB3 RS(SC1602B)
SCL SCL(RTC Module)
SDA SDA(RTC Module)

※改良

実際に動かしてみたら、時刻などの設定をする時に現在どの値を設定している最中なのかが分りにくかったので、その部分だけ少し改良した。

改良前と改良後

[crayon-67174722dbd4c502344735/]

RC0に接続してあるタクト・スイッチの押下で設定モードに入るが、その際にLCDの右下隅に文字を表示するようにした。

下の写真は「月」を設定している最中であることをあらわす「M」が表示されているところ。

※修正

時間の設定に不具合があった。

きっちり確認したわけではないが、12時間制を前提にプログラムされていて、うちのモジュールが24時間制に設定されているせいでおかしくなるような気がした。

そのため、起動時に強制的に12時間制に設定するコードを追加した。

アドレス0x02のBIT6をHighにすると12時間制になる。

```
while( SSPCON2bits.SEN );
```

の後に次の３行を追加する。

```
hour = read_ds1307(0x02);//0x02を読み込む
hour = hour | 0b01000000;//BIT6だけを変更して１にする
write_ds1307(2, hour); //0x02に書き戻す
```

※追加

曜日を表示するコードを追加した。

ddayはもともと宣言もされていて値も読み込まれているのに使われていない。

つまり、下の関数を追加してどこか適当なところで呼び出すだけでいい。

[crayon-67174722dbd54042090434/]

ちなみに、曜日は時間が午前０時をまたいだ時にだけ変更される。

日付だけを変更してもすぐには反映されない。

午前０時をまたいで変更されても正しい曜日に自動的に変わるわけではなく、前の曜日から次の曜日に変わるだけ。

※追加

曜日を変更できるようにした。

変更モード時の表示用文字列を以下のように変更する。

```
char disp[] = “hmsDMYd”;
```

「hms」は「時分秒」、「DMY」は「日月年」で、「d」は「曜日」を表す。

項目が１つ増えたのでswitch文用のカウントを１つ増やす。

```
if(set_count >= 8)
```

switch文に以下のcaseを追加する。

[crayon-67174722dbd58942371230/]

RTC内部では数値をBCD(Binary-coded decimal)で扱っているので、値の増減には普通のBinaryに直す必要がある。

ところが、曜日はどうせ７種類しかないので１桁で納まるから、BinaryもBCDも同じことになる。

つまり、BinaryとBCDの相互の変換処理が不要で、ちょっと楽した気分だ。

プリント基板を作成した。

「Real Time Clock Module (DS1307 + AT24C32) + PIC18F2550 プリント

□□□□□