RL78/G22-FPB(048)で遊んでみた(その 4)

1. はじめに

RL78/G22-FPB(048)で遊んでみた(その 2)の LCD の接続方法をポートから I2C バスに変更してみました。

FPB では、使用できる端子数に制限があり、7 本のポートを使用するのはもったいないので、I/F を I2C に変更することにしました。

このために、PCF8574(A/T)という I2C バスを使った 8 ビットのポート拡張用 IC を用いたモジュールを使用することにします。モジュールの外観の例を図 1. に示します。

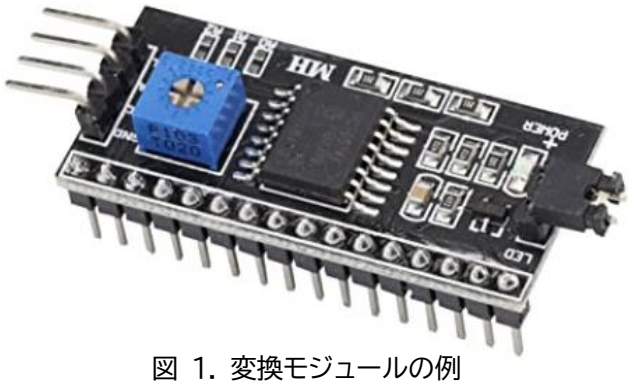

図 1. 変換モジュールの例

モジュール単体や LCD と組み合わせたものなどがネット通販で販売されています。LCD も 2 行のものだけでなく4行のものと組み合わされたものも販売されているので、簡単に入手できます。

PCF8574 はファーストモード(MAX400kbps)対応と書かれていることもあるようですが、スペックには標準モードの 100kbps までしか対応出来ていないとかかれているので、安全を考えてクロックは 100kHz で使用します。

このモジュールには LCD 表示モジュールのコントラスト調整用の半固定抵抗器や最大 8 個接続できるようなアドレス指定機能が準備されています。これは、複数の PCF8574 を使うことで、ポート数をふやすためのものです。さらに上位アドレスが2種類あるようです。実際、購入したものは 0x20 と 0x3F の両方が存在しました。このため、どのアドレスになっているかを確定する必要があります。そこで、実際に LCD の初期化を行う前に、0x20～0x27 と 0x38～0x3F の範囲で 0 バイトの書き込み(送信動作でスレーブドレスを送信)して、ACK 応答があるかどうかで判断します。最初に見つけた ACK 応答があったアドレスを採用することにします。ACK 応答がなければエラーを示すために LED を点滅させます。

2. ハードウェア構成

HDC1080(温湿度センサー)は Pmod のモジュールです。また、Arduino コネクタや Pmod コネクタではなく、Groveコネクタから信号を引き出すことにしました。

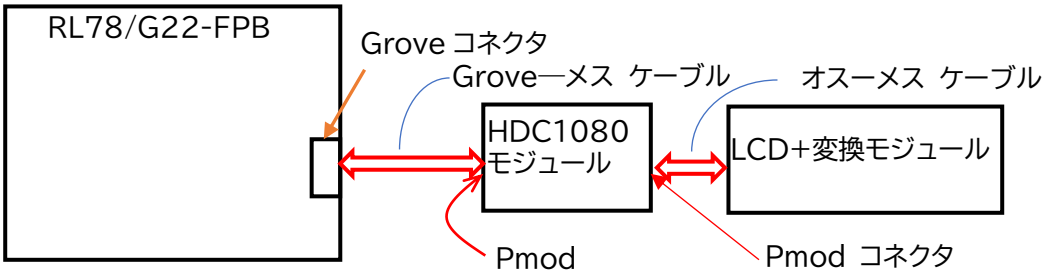

図 2. I2C バスの接続

HDC1080のモジュールには、I2C バスのプルアップ抵抗が実装されており、さらにカスケード接続できるような Pmod コネクタがあるので、そこに LCD の変換モジュールを接続することで、RL78/G22-FPB には、1 本のケーブルだけ接続することにします。これだけケーブルを接続するにはロック機能があるGroveコネクタから信号を引き出すのが安心です。そのように接続を変更した例を図 3.に示します。(ここでは、LCD は 4 行×20 文字のものを接続しています。)

ケーブルは合計 50cm 程度で、少し長すぎる気がしますが、I2C のクロックを 100kHz にしているので、大丈夫かなと思っています。

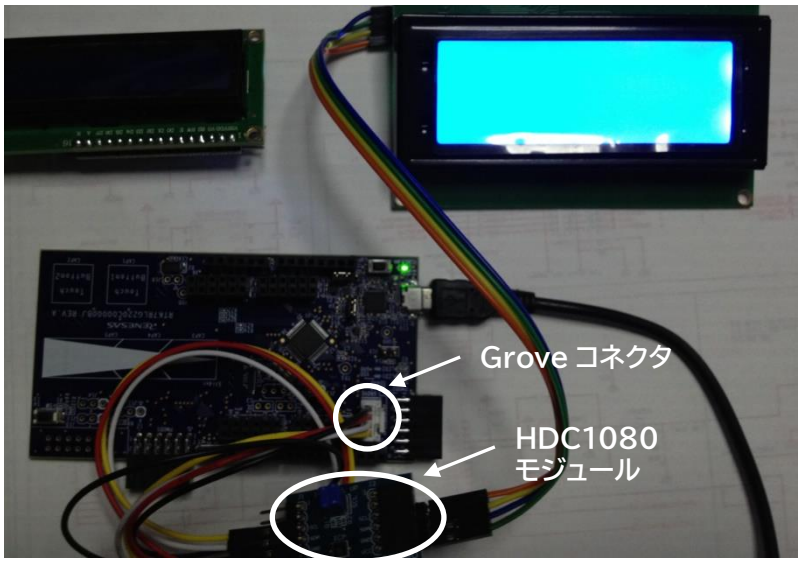

図 3. カスケード接続の例

実際にスケッチを動かして、表示させたところを図 4. に示します。LCD の表示部を抜き出したものです。

2 行×16 文字で表示するものを 4 行×20 文字の LCD に表示しているので、
表示は 4 行中の上 2 行に表示されています。

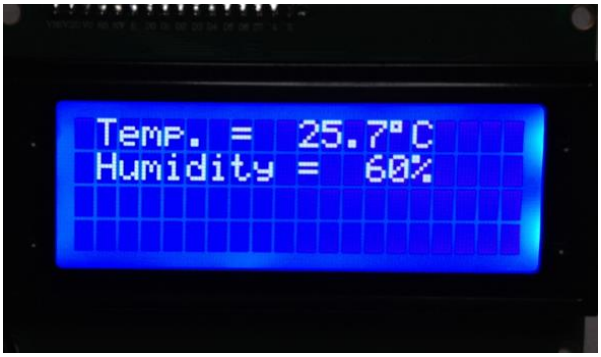

図 4. LCD 表示例

ちなみに 2 行×16 文字の LCD を接続したときの表示イメージを図 5. に示します。

図 5. 2 行×16 文字の LCD での表示例

Groveコネクタ対応のケーブルは、アマゾンで、「GROVE -4 ピン-ジャンパメスケーブル(5 本セット)」を購入して使用しました。また、RL78/G23-64PFPB も同様にするために、GROVE – ユニバーサル 4 ピンコネクタも同時に購入してあります。

3. ソフトウェア構成

最初は、Arduino の PCF8574 用ライブラリを使用するつもりでしたが、RL78/G22-FPB 用の LiquidCrystal API がサポートされていないので、この方法はあきらめました。その代わりに、HD44780 用のライブラリを改造して PCF8574 対応するライブラリを作成することにします。基本的には、関数はほとんど同じで、実際に LCD への信号を制御する関数だけを PCF8574 経由での制御に変えるだけにしてみます。

ただし、それだけでは無駄な待ち時間が出てきて、速度低下の原因となることは明らかです。まずは、動作確認して、速度が気になる程度かをチェックしていくことにします。

気にしているのが、センサから読み出した結果を変化しない無駄なデータを含めて 1 行単位で 16 バイト分送信していることです。しかも、この部分は Wire のデータバッファが 32 バイト分しかないので、図 6. に示すような処理にしてあります。

```
377     do
378     {
379         LCD_write(*string++);               // 表示データ転送
380         delayMicroseconds(60);              // 60マイクロ秒待つ
381 
382     }
383     while (*string);
```

図 6. 文字送信処理

これは、1 文字ごとに、 Wire.beginTransmission から Wire.endTransmission の処理を行い、スレーブ選択を含めて 6 バイトの転送が必要になっていました。複数文字を送信するようにすると、1 文字に対して 4 バイトが増えるだけになります。

それよりも、変化するデータ部分だけを送信するように変更すると、32 文字から 8 文字になります。ここらは、表示が気になったなら手を付けることにします。

LCD 用のライブラリでは、初期化処理だけは引数が異なるので関数名を変更しましたがその他の関数は同じ関数名のままなので、loop の中はいじらなくてもスケッチは動作します。

基本的に、変更したのは setup の部分です。

図 7. に示すのが初期化の最初の部分です。

```
59      pinMode(swPin, INPUT);              // set D18pin to input mode
60      pinMode(USERLED1, OUTPUT);          // set D16pin to output mode
61      digitalWrite(USERLED1, 1);          // turn off LED
62 
63      Wire.setClock(100000);              // set normal mode (100kHz clock)
64 
65      Wire.begin();                       // set IICA0 for I2C bus master
66 
```

図 7. 初期設定部 1

59 行目はスイッチ用の設定で、60 と 61 行目はエラー表示用の LED の設定です。LED は LCD が見つからなかったときに、点滅させてエラー状態を示すためのものです。

63 行目は I2C の最大通信速度を 100kbps に設定し、65 行目でマスタとして使うことを指定しています。

その後に、図 8. に示す LCD の確認処理です。

```
71  for( slave7 = 0x20 ; slave7 < 0x28 ; slave7++ )
72  {
73      Wire.beginTransmission(slave7); // マスタ送信で起動
74      work = Wire.endTransmission(1); // 送信データ数0で終了
75      delay(10);
76      if ( 0x00 == work )             // スレーブからACK応答か?
77      {
78          break;
79      }
80
81  }
82
83  if( 0x00 != work )
84  {
85      for( slave7 = 0x38 ; slave7 < 0x40 ; slave7++ )
86    {
87      Wire.beginTransmission(slave7); // マスタ送信で起動
88      work = Wire.endTransmission(1); // 送信データ数0で終了
89      delay(10);
90      if ( 0x00 == work )             // スレーブからACK応答か?
91      {
92          break;
93      }
94
95    }
96
97  }
```

図 8. LCD の確認処理

71 行目～81 行目では、スレーブアドレス(7 ビット)が 0x20～0x27 に LCD(PCF8574)が存在するかの確認を行っています。具体的には、送信モードでスレーブを選択し、ACK 応答を確認後すぐにストップコンディションを発行することで、0 バイトのデータ書き込み(つまり、具体的なデータ送信はなし)を行っています。ACK 応答が確認されたら、その時点で処理を完了します。結果は変数 work で得られます。

ここで、ACK 応答が確認されなかったら、83 行目～97 行目で 0x38～0x3F について同様に確認を行います。もし、LCD が見つからないと、図 9. に示すように LED を点滅させます。

```
 99  while ( 0x00 != work )
100  {
101      digitalWrite(USERLED1, temp);   // LEDを点滅
102      temp ^= 1;
103      delay(500);
104  }
```

図 9. LCD が見つからなかった場合の処理

LCD が見つかったなら、図 10. に示すように 107 行目で LCD(HD44780)を 4 ビット I/F、2 行表示等々に設定しています。

```
107     init_LCD_S();                       // initialize LCD display
108
109     disp_line1[14] = 0xDF;              // set degreeC character of LCD
110     print_LCD( (int8_t *)disp_line1, (int8_t *)disp_line2);   // display start data
111
112 }
```

図 10. LCD の初期設定

109 行目で、温度表示用に"°"を設定し、110 行目で初期画面を表示して setup を完了しています。loop に入ると、すぐにセンサを起動して、全て書き換えられるので、無駄になるかもしれませんが、loop ではデータ部分だけの書き換えもできるように、ダミーデータを含めて画面初期化を行い、表示しています。

4. ライブラリの変更点

PCF8574 を使った I2C バス接続では、I/F は固定なので、接続設定（LiquidCrystal 関数や自作ライブラリの init_LCD 関数）のようにパラメータは不要なので、初期化関数（init_LCD_S）では引数はありません。それ以外は init_LCD 関数と同じです。

HD44780 を直接制御する場合には、RS 信号を直接設定しておけばよかったのですが、I2C バス接続では、それだけを制御するのは無駄なので、PCF8574 に設定する制御信号を保持しておく変数（write_mode）の bit0 に設定しておきます。HD44780 に設定するために PCF8574 に書き込む際に write_mode の値の上位 4 ビットに設定した値をマージして I2C バスに送信します。

実際に I2C バスをアクセスしているのは、LCD_write 関数と LCD_write_nibble 関数だけで、他の関数はこの 2 つの関数を介して LCD にアクセスしています。

図 11. に LCD_write_nibble 関数を示します。この関数は、HD44780 の初期設定で使用しています。

```
489  byte LCD_nibble_write(uint8_t value)
490  {
491      uint8_t work;                          // データ書き込み作業用
492
493      Wire.beginTransmission( slave7 );      // PCF8574のアドレス指定
494
495      work    = (write_mode & 0b00001001);   // RS信号をコピー,R/Wを0に
496      Wire.write(work);                      // 書き込み信号セット
497
498      work    |= 0x04;                       // E信号セット
499      work    |= (value << 4);               // 書き込みデータを上位4bitに設定
500      Wire.write(work);                      // 書き込みデータとE信号アソート
501
502      work    ^= 0x04;                       // E信号をクリア
503      Wire.write(work);                      // E信号をネゲート
504
505      return(Wire.endTransmission(1));       // データ送信とストップコンディション完了
506
507  }
```

図 11. LCD_write_nibble 関数

この関数は、あらかじめ変数 write_mode に RS 信号の状態、変数 slave7 に LCD のスレーブアドレス（7bit）が設定されているものとして動作します。

493 行目で、PCF8574（LCD）をスレーブとして送信することを指定しています。495 行目で、変数 write_mode の RS 信号とバックライトを抽出して、それを 496 行目で送信します。ここは、図 12. に示すタイミング図で①に該当します。（実際には単にバッファに入れているだけで実際のデータ送信は 505 行目になります。

498 行目で E 信号をアクティブに設定し、499 行目で送信したデータ(引数の下位 4bit)をマージし、500 行目でバッファに入れます。図 12. では②に該当する部分ですが、E 信号だけでなく、データも設定しておきます。

502 行目で、E 信号をインアクティブに設定し、603 行目で、バッファに書き込んでいます。図 12. では③に該当します。

図 12.の④は単に E 信号に対しての保持時間を示しており、次の変化は次の書き込み時になるので、何もしなくても保持時間は確保できます。

505 行目で、ここまでバッファにためられたデータが実際に I2C バスに送信されていきます。

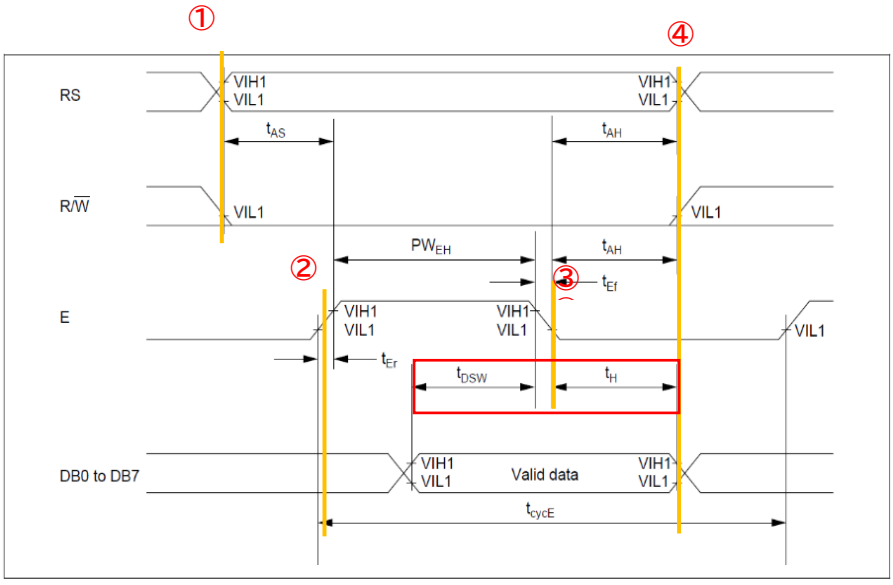

図 12. タイミング

LCD_write 関数は、基本的に 4bit の送信を 2 回実行することで実現できます。しかし、それでは通信が効率的ではありません。具体的には、1 回目の 4bit の③は 2 回目の①と同じと考えることができます。

具体的な LCD_write 関数を図 13. に示します。図 13. には、タイミングの①～④に関連する部分を明示しています。

図 12. のタイミングは HD44780 のタイミングなので、I2C 関係のスタート・コンディションやスレーブ選択タイミングは出てきませんが、実際にはその分の時間は必要です。

```
441     Wire.beginTransmission( slave7 );       // PCF8574のアドレス指定
442
443   /*-----------------------------------------------
444     上位ニブルを書き込み
445   -----------------------------------------------*/
446
447     work    = (write_mode & 0x09);          // RS信号をコピー,R/WをW(0)に,BLは1
448 ①   Wire.write(work);                       // 書き込み信号セット
449     work    |= 0x04;                        // E信号セット
450     work    |= (value & 0xF0);              // 書き込みデータ上位ビットをマージ
451 ②   Wire.write(work);                       // 書き込みデータとE信号アソート
452
453     work    ^= 0x04;                        // E信号をクリア
454 ③   Wire.write(work);                       // E信号をネゲート
455
456   /*-----------------------------------------------
457     下位ニブルを書き込み
458   -----------------------------------------------*/
459
460     work    &= 0x0F;                        // データ領域をクリア
461     work    |= 0x04;                        // E信号セット
462     work    |= (value << 4);                // 書き込みデータを上位4bitに設定
463 ②   Wire.write(work);                       // 書き込みデータとE信号アソート
464
465     work    ^= 0x04;                        // E信号をクリア
466 ③   Wire.write(work);                       // E信号をネゲート
467
468 ④   return(Wire.endTransmission(1));        // データ送信とストップコンディション完了
```

図 13. LCD_write 関数

その他の関数は、HD44780 をポートで制御しているものと同じです。このスケッチでは、いくつかの API 関数を準備していますが、実際に使っているのは DisplayString 関数だけです。

以上