作業報告書（2022 年 11 月 28 日）

J20413 北野正樹

【作業内容】

・I2C インターフェースについての調査

・プログラムの作成

【作業項目】

① I2C インターフェースとは

I2C インターフェースは、フィリップス社で開発されたシリアルバスで、さまざまなデバイスを複数接続して相互にデータのやり取りを行うような用途に用い折られている。I2C で使われているのは、抵抗でプルアップされた双方向のオープンコレクタ信号線であり。図 1 のようにシリアルデータ（SDA）とシリアルクロック（SCL）からなる。電圧は最高で＋5V までで、よく使われるのは＋３．3V である。接続される各デバイスはマスタまたはスレーブとして機能し、マスタが指定したアドレスのスレーブとデータのやり取りが行われる。アドレス空間は 7 ビットで、そのうちの 16 個の予約アドレスを除いた最大 112 個のノードが、同じバス上で通信できる。一般的な I2C のバスのモードは、100kbit/s の標準モードと 10kbit/s の低速モードがある。

通信方式は、必ず８ビット単位のデータ転送となっており。マスタがスタート状態の後第一バイト目として通信相手の 7 ビットスレーブアドレスと読み書きの方向を含めた８ビットのデータを送出する。図 2 に各信号線の動作の様子を示す。各デバイスは受信したアドレスを監視して、自分のアドレスと同一の時だけ ACK の送信と第２バイト以降のデータ転送を行う。第２バイト以降は読み出しではスレーブが送信側→マスタが受信側となり、書き込みでハマスタが送信側→スレーブが受信側となる。ACK は受信側が返す。通信の最後にストップ状態とすることで、終了する。

よって、一連の通信の長さは転送するバイト数によって変化し。最短でも２バイトとなる。

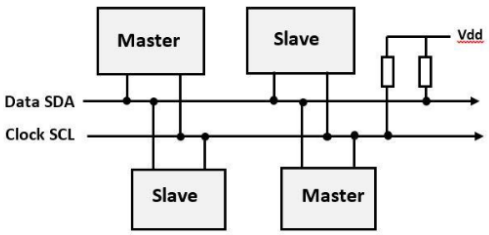

図 1　I2C の接続形態

図 1：I2C の接続形態

作業報告書（2022 年 11 月 28 日）

J20413 北野正樹

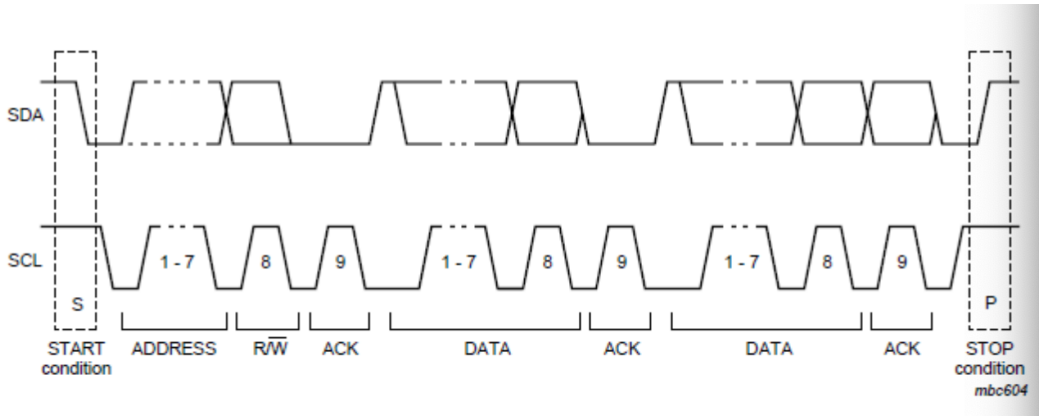

図 2 I2C のバス動作（出典: https://www.nxp.com/docs/jauser-guide/UM10204.pdf）

② 使用するセンサ（BMP180）について

使用するセンサは BOSCH 社の BMP180 である。I2C 接続することで、デジタルデータとして温度と気圧を計測することができる。

ただし、個体差によってばらつきが生じるため、読み出した値に対してキャリブレーション（補正）を行うことが必要となる。補正方法はマニュアルに記述されており、補正に必要なデータはメーカによってセンサ自体に記録されており。特定のレジスタから読み出すことができる。したがって、温度や気圧のデータを読み出した後に、各補正データを使ってマニュアルに指定された補正計算を行う必要がある。さらに、気圧の補正には温度も必要となるため、先に温度を読み出して補正を行った後で、気圧の補正を行うことになる。

これらの補正計算の方法は BMP180 のデータシートに記載されているので、そのアルゴリズムに従ってプログラムを作成すれば良い。この資料の最後にアルゴリズムを掲載するので、それに従ってプログラムを作成することになる。

この時注意すべき点として、記載されているアルゴリズムには整数型として、short(符号付き１６ビット整数)、unsigned short（符号付き１６ビット整数)、long（符号付き３２ビット整数)、unsigned long(符号なし３２ビット整数)の４種類で使用されている。使用する言語処理系の変数のビット幅に注意して、アルゴリズムを実装する必要がある。

例えば、書くキャリブレーションデータは１６ビットで格納されており、８ビットずつ二回に分けて読み込むことになる。符号なしの場合は上位バイトを８ビット分上位側にシフトして、下位バイトを加算すれば良いが、符号付きの場合は最上位ビットが符号を示しているのでそれによって正しい計算を行う必要がある。

読み出しデータはノイズの影響を受けるため、ハードウェアの機能として、気圧に関してのみ、複数回のサンプリングを自動で行う機能がある。補正アルゴリズムには oss として変数の形で登場しているので、注意すること。サンプリングの回数分だけ出力値も積算されるがこれについては補正アルゴリズムに組み込まれているので、別途考慮する必要はない。また、サンプリングを繰り返す分だけ変換時間が増加するため、変換を開始してから値を読み出すまでの待ち時間が oss の値によって変化するので注意すること。補正アルゴリズムでは、気温の読み出しは wait 4.5ms と明示されているが、気圧の読み出しでは単に wait と記載されているのみなので、oss の

作業報告書（2022 年 11 月 28 日）

J20413 北野正樹

値に応じて変換時間以上の待ち時間を確保するようにしなければならない。oss の値と変換時間については、デバイスのデータシートに記載されており。それをまとめた表を撥水して以下に示す。

表 1　測定モード（測定精度）

モード	oss の値	サンプリング回数	気圧の変換時間 [ms]
超省電力	0	1	4.5
標準	1	2	7.5
高解像度	2	4	13.5
超高解像度	3	8	25.5

※データシート p.12 Table3 より抜粋

③ 作成したプログラムを以下に示す。

```
④ #include <stdio.h>
⑤ #include <stdlib.h>
⑥ #include <string.h>
⑦ #include <wiringPi.h>
⑧ #include <wiringPiI2C.h>
⑨ // BMP180 の I2C インタフェースのアドレス
⑩ // 確認方法：gpio i2cdetect
⑪ #define BMP180ADDR 0x77
⑫ // レジスタアドレスの定義
⑬ // キャリブレーションデータレジスタのアドレス
⑭ // AC1-AC6,B1,B2,MB,MC,MD 全て 2 バイト（16 ビット）長
⑮ // 先頭アドレス
⑯ #define CALSADR 0xaa
⑰ // 終了アドレス
⑱ #define CALEADR 0xbf
⑲ // キャリブレーションデータの個数
⑳ #define CALDATANUM 11
21 // キャリブレーション用のデータ配列参照用
22 #define AC1 0
23 #define AC2 1
24 #define AC3 2
25 #define AC4 3
26 #define AC5 4
27 #define AC6 5
```

```
28 #define B1 6
29 #define B2 7
30 #define MB 8
31 #define MC 9
32 #define MD 10
33 // データレジスタのアドレス
34 #define DATAMSB 0xF6
35 #define DATALSB 0xF7
36 #define DATAXLSB 0xF8
37 // コントロールレジスタのアドレス
38 // b7,b6:oss b5:sco b4-b0:measurement
39 // oss-> 0 :lowpower,1:standard,2:highres,3:ultrahighres
40 #define CTRLREG 0xf4
41 // 測定開始コマンドの定義
42 #define TEMP 0x2e
43 #define PRESS0 0x34
44 #define PRESS1 0x74
45 #define PRESS2 0xb4
46 #define PRESS3 0xf4
47 // リセットレジスタ（書き込み専用）のアドレス
48 // 0xb6 を書き込むとパワーオンリセット動作
49 #define RESETREG 0xe0
50 // ID レジスタ
51 // 機能チェック用、正常なら読み出すと常に 0x55
52 #define IDREG 0xd0
53 // コンパイル方法
54 //      gcc -Wall -o bmp180 bmp180.c -lwiringPi
55 // 測定モードの定義（oss）
56 #define LOWPOWER 0
57 #define STANDARD 1
58 #define HIGHRES 2
59 #define ULTRARES 3
60 // BMP180 の機能チェック用
61 #define CHECKOK 0
62 #define CHECKNG -1
63 // キャリブレーションレジスタ名のテーブル
64 const char* calregname[11] = {"AC1", "AC2", "AC3", "AC4", "AC5", "AC6",
```

作業報告書（2022 年 11 月 28 日）

J20413 北野正樹

```
65                                          "B1",  "B2",  "MB",  "MC",  "MD"};
66 // 以下の定数は補正アルゴリズム中にある数値をそのまま入れたもの
67 // 実際の測定値や補正データの代わりに使えば、期待した動作かどうかの検証ができる
68 int testcaldata[11] = {408,   -72, -14383, 32741, 32757, 23153,
69                                          6190, 4,     -32768, -8711, 2868};
70 int testut = 27898;
71 int testup = 23843;
72 int testoss = 0;
73 // プロトタイプ宣言
74 int get_press(int ut, int up, int oss, int* caldata);
75 int get_temp(int ut, int* caldata);
76 int get_raw_press(int fd, int oss);
77 int get_raw_temp(int fd);
78 int check_bmp180_function(int fd);
79 int* read_caldata(int fd, int* caldata);
80 int main() {
81       int fd;
82       // デバイスディスクプリタ、デバイスごとにつく番号のようなもの
83       int oss = 1;
84       // 気圧標準測定（2 回サンプリング平均）
85       int caldata[CALDATANUM];
86       // 補正データを格納する配列
87       int ut, up;
88       // 生の温度・圧力値（ADC の変換結果そのもの）
89       int t, p;
90       // 補正した後の温度・圧力値
91       // I2C インタフェースの初期化
92       fd = wiringPiI2CSetup(BMP180ADDR);
93       if (fd < 0) {
94             printf("I2C 初期化エラー！¥n");
95             exit(EXIT_FAILURE);
96       }
97       // 動作チェック機能を使った動作確認
98       if (check_bmp180_function(fd) != CHECKOK) {
99             printf("BMP180 の動作確認が不良です。¥n");
100            exit(EXIT_FAILURE);
101      }
```

作業報告書（2022 年 11 月 28 日）

J20413 北野正樹

```
102     //
103     printf("BMP180 の動作確認 OK¥n¥n");
104     read_caldata(fd, caldata);
105     // 補正データの読み出し
106     ut = get_raw_temp(fd);
107     // 温度測定値の読み出し
108     up = get_raw_press(fd, oss);
109     // 気圧測定値の読み出し
110     t = get_temp(ut, caldata);
111     // 補正計算を行って補正した温度を求める
112     // t = get_temp(testut,testcaldata);のようにすると補正の動作確認ができる
113     printf("気温（補正済み）= %4.1f ° C, ", (float)t / 10.0);
114     p = get_press(ut, up, oss, caldata);
115     // 補正計算を行って補正した気圧を求める
116     //p = get_press(testut, testup,
117     // testoss,testcaldata);で補正の動作確認ができる
118     printf("気圧（補正済み）= %6.2f hPa¥n", (float)p / 100.0);
119     return 0;
120}
121int get_press(int ut, int up, int oss, int* caldata)
122// 測定した温度と気圧データから、気圧の補正計算を行う関数
123// 戻り値は補正した気圧の値
124{
125     long X1 = (ut-caldata[AC6])*caldata[AC5]/32768;
126     long X2 = caldata[MC]*2048/(X1+caldata[MD]);
127     long B5 = X1+X2;
128     long B6 = B5-4000;
129     X1 = (caldata[B2]*(B6*B6/4096))/2048;
130     X2 = caldata[AC2]*B6/2048;
131     long X3 = X1+X2;
132     long B3 = (((caldata[AC1]*4+X3)<<oss)+2)/4;
133     X1 = caldata[AC3]*B6/8192;
134     X2 = (caldata[B1]*(B6*B6/4096))/65536;
135     X3 = ((X1+X2)+2)/4;
136     unsigned long B4 = caldata[AC4]*(unsigned long)(X3+32768)/32768;
137     unsigned long B7 = ((unsigned long)up-B3)*(50000>>oss);
138     long p;
```

```
139     if(B7<0x80000000){
140         p=(B7*2)/B4;
141     }else{
142         p=(B7/B4)*2;
143     }
144     X1 = (p/256)*(p/256);
145     X1 = (X1*3038)/65536;
146     X2 = (-7357*p)/65536;
147     p += (X1+X2+3791)/16;
148     return p;
149}
150int get_temp(int ut, int* caldata)
151// 温度測定値と補正データを引数にとって補正した温度を求める関数
152// 戻り値は補正した温度（整数計算のため、真値の 10 倍になっているはず）
153{
154     long X1 = (ut-caldata[AC6])*caldata[AC5]/32768;
155     long X2 = caldata[MC]*2048/(X1+caldata[MD]);
156     long B5 = X1+X2;
157     long t = (B5+8)/16;
158     return t;
159}
160int get_raw_press(int fd, int oss)
161     // 気圧の測定値を求める関数
162     // oss で測定時の変換回数を指定する
163{
164     int m,l, x;
165     // MSB, LSB, XLSB を入れる変数
166     int up;
167     // 計算して求めた値を入れる変数
168     // oss の範囲は 0 から 3 まで
169     if (oss < 0) oss = 0;
170     if (oss > 3) oss = 3;
171     wiringPiI2CWriteReg8(fd, CTRLREG, PRESS0 + (oss << 6));
172     // 変換開始
173     // 変換時間待ち、oss の値（変換回数=2^oss）によって待ち時間が異なる
174     switch (oss) {
175         case 0:
```

```
176                 delay(5);
177                 break;
178             case 1:
179                 delay(8);
180                 break;
181             case 2:
182                 delay(14);
183                 break;
184             default:
185                 delay(26);
186     }
187     // ここにデータレジスタから MSB, LSB, XLSB を読み出すコードを書く
188     // 気圧は 3 バイトのデータ値から計算することになるので注意
189     m = wiringPiI2CReadReg8(fd, DATAMSB);
190     l = wiringPiI2CReadReg8(fd, DATALSB);
191     x = wiringPiI2CReadReg8(fd, DATAXLSB);
192     // ここに読み出した m, l, x から値を計算するコードを書く
193     up = ((m<<16) + (l<<8) + x) >> (8-oss);
194     return up;
195}
196int get_raw_temp(int fd){
197     int m, l;
198     // 読み出した MSB, LSB を入れる変数
199     int ut;
200     // 計算で求めた測定温度を入れる変数
201     wiringPiI2CWriteReg8(fd, CTRLREG, TEMP);
202     // 温度の測定開始
203     delay(5);
204     // 変換時間待ち、最大変換時間は 4.5ms
205     // ここにデータレジスタから MSB, LSB, XLSB を読み出すコードを書く
206     // 気圧は 2 バイトのデータ値から計算することになるので注意
207     m = wiringPiI2CReadReg8(fd, DATAMSB);
208     l = wiringPiI2CReadReg8(fd, DATALSB);
209     //ここに読み出した m, l から値を計算するコードを書く
210     ut = (m<<8) + l;
211     return ut;
212}
```

作業報告書（2022 年 11 月 28 日）

J20413 北野正樹

```
213int check_bmp180_function(int fd)
214// BMP180 の機能チェックを行う関数
215{
216     int r;
217     if (wiringPiI2CReadReg8(fd, IDREG) == 0x55){
218          r = CHECKOK;
219     }else{
220          r = CHECKNG;
221     }
222     return r;
223}
224int* read_caldata(int fd,int* caldata)
225// キャリブレーションデータの読み出し関数
226{
227     int i; int l,m;
228     printf("キャリブレーション値を読み込みます。¥n");
229     for (i = 0; i < CALDATANUM; i++){
230          m = wiringPiI2CReadReg8(fd,CALSADR+i*2);
231          l = wiringPiI2CReadReg8(fd,CALSADR+i*2+1);
232          caldata[i] = (m<<8) + l;
233          // 符号なし 16 ビットに変換
234          if ((i != AC4) && (i != AC5) && (i != AC6)){
235               // AC4, AC5, AC6 は符号付き 16 ビットデータなので対処が必要
236               // その対処をここに書く
237               caldata[i] = (signed short)caldata[i];
238          }
239     }
240     return caldata;
241}
```

【作業時間】

・作業時間：90 分

・報告書作成時間：60 分