作業報告書（2022 年 12 月 12 日）

J20413 北野正樹

【作業内容】

プログラムを完成させる

I2C 温度・圧力センサの回路を組む

実際に数値を計測する

【作業項目】

① プログラムを作成する。今回作成したプログラムを以下に示す。

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

作業報告書（2022 年 12 月 12 日）

J20413 北野正樹

```
32 #define DATAMSB 0xF6
33 #define DATALSB 0xF7
34 #define DATAXLSB 0xF8
35 // コントロールレジスタのアドレス
36 // b7,b6:oss b5:sco b4-b0:measurement
37 // oss-> 0 :lowpower,1:standard,2:highres,3:ultrahighres
38 #define CTRLREG 0xf4
39 // 測定開始コマンドの定義
40 #define TEMP 0x2e
41 #define PRESS0 0x34
42 #define PRESS1 0x74
43 #define PRESS2 0xb4
44 #define PRESS3 0xf4
45 // リセットレジスタ（書き込み専用）のアドレス
46 // 0xb6 を書き込むとパワーオンリセット動作
47 #define RESETREG 0xe0
48 // ID レジスタ
49 // 機能チェック用、正常なら読み出すと常に 0x55
50 #define IDREG 0xd0
51 // コンパイル方法
52 //       gcc -Wall -o bmp180 bmp180.c -lwiringPi
53 // 測定モードの定義（oss）
54 #define LOWPOWER 0
55 #define STANDARD 1
56 #define HIGHRES 2
57 #define ULTRARES 3
58 // BMP180 の機能チェック用
59 #define CHECKOK 0
60 #define CHECKNG -1
61 // キャリブレーションレジスタ名のテーブル
62 const char* calregname[11] = {"AC1", "AC2", "AC3", "AC4", "AC5", "AC6",
63                                 "B1",   "B2",   "MB",   "MC",   "MD"};
64 // 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
65 // 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
66 int testcaldata[11] = {408,   -72, -14383, 32741, 32757, 23153,
67                                 6190, 4,     -32768, -8711, 2868};
68 int testut = 27898;
```

作業報告書（2022 年 12 月 12 日）

J20413 北野正樹

```
69  int testup = 23843;
70  int testoss = 0;
71  // プロトタイプ宣言
72  int get_press(int ut, int up, int oss, int* caldata);
73  int get_temp(int ut, int* caldata);
74  int get_raw_press(int fd, int oss);
75  int get_raw_temp(int fd);
76  int check_bmp180_function(int fd);
77  int* read_caldata(int fd, int* caldata);
78  int main() {
79      int fd;
80      // 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
81      int oss = 1;
82      // 気圧標準測定（2 回サンプリング平均）
83      int caldata[CALDATANUM];
84      // 補正データを格納する配列
85      int ut, up;
86      // 生の温度・圧力値（ADC の変換結果そのもの）
87      int t, p;
88      // 補正した後の温度・圧力値
89      // I2C インタフェースの初期化
90      fd = wiringPiI2CSetup(BMP180ADDR);
91      if (fd < 0) {
92          printf("I2C 初期化エラー！¥n");
93          exit(EXIT_FAILURE);
94      }
95      // 動作チェック機能を使った動作確認
96      if (check_bmp180_function(fd) != CHECKOK) {
97          printf("BMP180 の動作確認が不良です。¥n");
98          exit(EXIT_FAILURE);
99      }
100     //
101     printf("BMP180 の動作確認 OK¥n¥n");
102     read_caldata(fd, caldata);
103     // 補正データの読み出し
104     ut = get_raw_temp(fd);
105     // 温度測定値の読み出し
```

作業報告書（2022 年 12 月 12 日）

J20413 北野正樹

```
106     up = get_raw_press(fd, oss);
107     // 気圧測定値の読み出し
108     t = get_temp(ut, caldata);
109     // 補正計算を行って補正した温度を求める
110     // t = get_temp(testut,testcaldata);のようにすると補正の動作確認ができる
111     printf("気温（補正済み）= %4.1f ° C, ", (float)t / 10.0);
112     p = get_press(ut, up, oss, caldata);
113     // 補正計算を行って補正した気圧を求める
114     //p = get_press(testut, testup,
115     // testoss,testcaldata);で補正の動作確認ができる
116     printf("気圧（補正済み）= %6.2f hPa¥n", (float)p / 100.0);
117     return 0;
118}
119int get_press(int ut, int up, int oss, int* caldata)
120// 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
121// 戻り値は補正した気圧の値
122{
123     long X1 = (ut-caldata[AC6])*caldata[AC5]/32768;
124     long X2 = caldata[MC]*2048/(X1+caldata[MD]);
125     long B5 = X1+X2;
126     long B6 = B5-4000;
127     X1 = (caldata[B2]*(B6*B6/4096))/2048;
128     X2 = caldata[AC2]*B6/2048;
129     long X3 = X1+X2;
130     long B3 = (((caldata[AC1]*4+X3)<<oss)+2)/4;
131     X1 = caldata[AC3]*B6/8192;
132     X2 = (caldata[B1]*(B6*B6/4096))/65536;
133     X3 = ((X1+X2)+2)/4;
134     unsigned long B4 = caldata[AC4]*(unsigned long)(X3+32768)/32768;
135     unsigned long B7 = ((unsigned long)up-B3)*(50000>>oss);
136     long p;
137     if(B7<0x80000000){
138          p=(B7*2)/B4;
139     }else{
140          p=(B7/B4)*2;
141     }
142     X1 = (p/256)*(p/256);
```

```
143     X1 = (X1*3038)/65536;
144     X2 = (-7357*p)/65536;
145     p += (X1+X2+3791)/16;
146     return p;
147}
148int get_temp(int ut, int* caldata)
149// 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
150// 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）
151{
152     long X1 = (ut-caldata[AC6])*caldata[AC5]/32768;
153     long X2 = caldata[MC]*2048/(X1+caldata[MD]);
154     long B5 = X1+X2;
155     long t = (B5+8)/16;
156     return t;
157}
158int get_raw_press(int fd, int oss)
159     // 気圧の測定値を求める関数
160     // oss で測定時の変換回数を指定する
161{
162     int m,l, x;
163     // MSB, LSB, XLSB を入れる変数
164     int up;
165     // 計算して求めた値を入れる変数
166     // oss の範囲は 0 から 3 まで
167     if (oss < 0) oss = 0;
168     if (oss > 3) oss = 3;
169     wiringPiI2CWriteReg8(fd, CTRLREG, PRESS0 + (oss << 6));
170     // 変換開始
171     // 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
172     switch (oss) {
173          case 0:
174               delay(5);
175               break;
176          case 1:
177               delay(8);
178               break;
179          case 2:
```

```
180                 delay(14);
181                 break;
182             default:
183                 delay(26);
184     }
185     // 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
186     // 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
187     m = wiringPiI2CReadReg8(fd, DATAMSB);
188     l = wiringPiI2CReadReg8(fd, DATALSB);
189     x = wiringPiI2CReadReg8(fd, DATAXLSB);
190     // 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
191     up = ((m<<16) + (l<<8) + x) >> (8-oss);
192     return up;
193}
194int get_raw_temp(int fd){
195     int m, l;
196     // 読み出した MSB, LSB を入れる変数
197     int ut;
198     // 計算で求めた測定温度を入れる変数
199     wiringPiI2CWriteReg8(fd, CTRLREG, TEMP);
200     // 温度の測定開始
201     delay(5);
202     // 変換時間待ち、最大変換時間は 4.5ms
203     // 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
204     // 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
205     m = wiringPiI2CReadReg8(fd, DATAMSB);
206     l = wiringPiI2CReadReg8(fd, DATALSB);
207     //ここに読み出した m, l から値を計算するコードを書く
208     ut = (m<<8) + l;
209     return ut;
210}
211int check_bmp180_function(int fd)
212// BMP180 の機能チェックを行う関数
213{
214     int r;
215     if (wiringPiI2CReadReg8(fd, IDREG) == 0x55){
216         r = CHECKOK;
```

作業報告書（2022 年 12 月 12 日）

J20413 北野正樹

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

② I2C 温度・圧力センサの回路を組む。回路図を図 1 に示す。

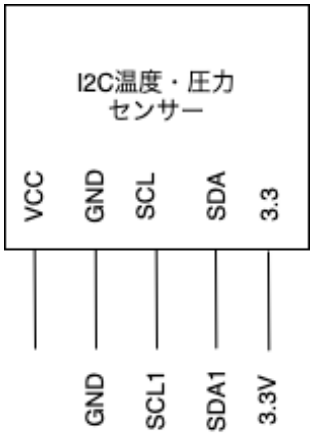

図 1：I2C 温度・圧力センサーの回路図

作業報告書（2022 年 12 月 12 日）

J20413 北野正樹

③ gcc –Wall –o I2C I2C.c –lwiringPi を実行し、プログラムをコンパイルする。

④ sudo で実行する。実行した結果を下記に示す。

```
[raspi4-12@raspi4-12:~/MyHomeDirectory/R4J3/RaspberryPi/j3ex1121 $ sudo ./I2C
BMP180の動作確認OK

キャリブレーション値を読み込みます。
気温（補正済み）= 20.4 °C, 気圧（補正済み）= 1015.65 hPa
[raspi4-12@raspi4-12:~/MyHomeDirectory/R4J3/RaspberryPi/j3ex1121 $ sudo ./I2C
BMP180の動作確認OK

キャリブレーション値を読み込みます。
気温（補正済み）= 20.3 °C, 気圧（補正済み）= 1015.69 hPa
```

温度と気圧は補正された値が出力されている。

【作業時間】

・作業時間：90 分

・報告書作成時間：30 分