作業報告書（2022 年 1 月 16 日）

J20413 北野正樹

【作業内容】

気圧と温度のデータを 4 連 7 セグメントディスプレイに表示させるために、プログラムを変更する。

【作業項目】

① プログラムを作成する。作成したプログラムを以下に示す。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <unistd.h>
// BMP180 の I2C インタフェースのアドレス
// 確認方法：gpio i2cdetect
#define BMP180ADDR 0x77
// レジスタアドレスの定義
// キャリブレーションデータレジスタのアドレス
// AC1-AC6,B1,B2,MB,MC,MD 全て 2 バイト（16 ビット）長
// 先頭アドレス
#define CALSADR 0xaa
// 終了アドレス
#define CALEADR 0xbf
// キャリブレーションデータの個数
#define CALDATANUM 11
// キャリブレーション用のデータ配列参照用
#define AC1 0
#define AC2 1
#define AC3 2
#define AC4 3
#define AC5 4
#define AC6 5
#define B1 6
#define B2 7
#define MB 8
#define MC 9
#define MD 10
// データレジスタのアドレス
#define DATAMSB 0xF6
```

作業報告書（2022 年 1 月 16 日）

J20413 北野正樹

```
#define DATALSB 0xF7
#define DATAXLSB 0xF8
// コントロールレジスタのアドレス
// b7,b6:oss b5:sco b4-b0:measurement
// oss-> 0 :lowpower,1:standard,2:highres,3:ultrahighres
#define CTRLREG 0xf4
// 測定開始コマンドの定義
#define TEMP 0x2e
#define PRESS0 0x34
#define PRESS1 0x74
#define PRESS2 0xb4
#define PRESS3 0xf4
// リセットレジスタ（書き込み専用）のアドレス
// 0xb6 を書き込むとパワーオンリセット動作
#define RESETREG 0xe0
// ID レジスタ
// 機能チェック用、正常なら読み出すと常に 0x55
#define IDREG 0xd0
// コンパイル方法
//      gcc -Wall -o bmp180 bmp180.c -lwiringPi
// 測定モードの定義（oss）
#define LOWPOWER 0
#define STANDARD 1
#define HIGHRES 2
#define ULTRARES 3
// BMP180 の機能チェック用
#define CHECKOK 0
#define CHECKNG -1
// キャリブレーションレジスタ名のテーブル
const char* calregname[11] = {"AC1", "AC2", "AC3", "AC4", "AC5", "AC6",
                              "B1",  "B2",  "MB",  "MC",  "MD"};
// 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
// 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
int testcaldata[11] = {408,   -72, -14383, 32741, 32757, 23153,
                       6190, 4,     -32768, -8711, 2868};
//7seg
int seg_list[][7] = {
```

```
    {1,1,1,1,1,1,0},
    {0,1,1,0,0,0,0},
    {1,1,0,1,1,0,1},
    {1,1,1,1,0,0,1},
    {0,1,1,0,0,1,1},
    {1,0,1,1,0,1,1},
    {1,0,1,1,1,1,1},
    {1,1,1,0,0,0,0},
    {1,1,1,1,1,1,1},
    {1,1,1,0,0,1,1},
    {1,0,0,1,1,1,0},//C
};
//
int gpio_pin[12]={26,19,13,12,6,22,23,25,16,5,18,24};
int dig_pin[4]={26,19,13,12};//1,2,3,4
int seg_pin[7]={6,22,23,25,16,5,18};//a,b,c,d,e,f,g,dp

int testut = 27898;
int testup = 23843;
int testoss = 0;
// プロトタイプ宣言
int get_press(int ut, int up, int oss, int* caldata);
int get_temp(int ut, int* caldata);
int get_raw_press(int fd, int oss);
int get_raw_temp(int fd);
int check_bmp180_function(int fd);
int* read_caldata(int fd, int* caldata);
void clear_seg();

int main() {
    int fd;
    // 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
    int oss = 1;
    // 気圧標準測定（2 回サンプリング平均）
    int caldata[CALDATANUM];
    // 補正データを格納する配列
    int ut, up;
```

作業報告書（2022 年 1 月 16 日）

J20413 北野正樹

```
// 生の温度・圧力値（ADC の変換結果そのもの）
int t, p;
// 補正した後の温度・圧力値
// I2C インタフェースの初期化
fd = wiringPiI2CSetup(BMP180ADDR);
wiringPiSetupGpio();
for(int i=0;i<12;i++){
	pinMode(gpio_pin[i],OUTPUT);
}
if (fd < 0) {
	printf("I2C 初期化エラー！¥n");
	exit(EXIT_FAILURE);
}
// 動作チェック機能を使った動作確認
if (check_bmp180_function(fd) != CHECKOK) {
	printf("BMP180 の動作確認が不良です。¥n");
	exit(EXIT_FAILURE);
}
//
printf("BMP180 の動作確認 OK¥n¥n");
read_caldata(fd, caldata);
// 補正データの読み出し
ut = get_raw_temp(fd);
// 温度測定値の読み出し
up = get_raw_press(fd, oss);
// 気圧測定値の読み出し
t = get_temp(ut, caldata);
// 補正計算を行って補正した温度を求める
// t = get_temp(testut,testcaldata);のようにすると補正の動作確認ができる
printf("気温（補正済み）= %4.1f ° C, ", (float)t / 10.0);
p = get_press(ut, up, oss, caldata);
// 補正計算を行って補正した気圧を求める
//p = get_press(testut, testup,
// testoss,testcaldata);で補正の動作確認ができる
printf("気圧（補正済み）= %6.2f hPa¥n", (float)p / 100.0);
char temp[100];
sprintf(temp,"%4.1f",(float)t / 10.0);
```

```
    char press[100];
    sprintf(press,"%6.2f",(float)p / 100.0);
    while(1){
        for(int x=0;x<1000;x++){
            for(int i=0,k=0;i<4;i++){
                digitalWrite(dig_pin[i],HIGH);
                for(int j=0;j<7;j++){
                    if(seg_list[temp[i+k]-'0'][j]==1){
                        digitalWrite(seg_pin[j],LOW);
                    }
                }
                if(i==1){
                    digitalWrite(gpio_pin[11],LOW);
                    k=1;
                }
                delay(1);
                clear_seg();
            }
        }
        for(int x=0;x<1000;x++){
            for(int i=0,k=0;i<4;i++){
                digitalWrite(dig_pin[i],HIGH);
                for(int j=0;j<7;j++){
                    if(seg_list[press[i+k]-'0'][j]==1){
                        digitalWrite(seg_pin[j],LOW);
                    }
                }
                delay(1);
                clear_seg();
            }
        }
    }

    return 0;
}

void clear_seg(){
```

```
	digitalWrite(gpio_pin[0],LOW);
	digitalWrite(gpio_pin[1],LOW);
	digitalWrite(gpio_pin[2],LOW);
	digitalWrite(gpio_pin[3],LOW);
	digitalWrite(gpio_pin[4],HIGH);
	digitalWrite(gpio_pin[5],HIGH);
	digitalWrite(gpio_pin[6],HIGH);
	digitalWrite(gpio_pin[7],HIGH);
	digitalWrite(gpio_pin[8],HIGH);
	digitalWrite(gpio_pin[9],HIGH);
	digitalWrite(gpio_pin[10],HIGH);
	digitalWrite(gpio_pin[11],HIGH);
}

int get_press(int ut, int up, int oss, int* caldata)
// 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
// 戻り値は補正した気圧の値
{
	long X1 = (ut-caldata[AC6])*caldata[AC5]/32768;
	long X2 = caldata[MC]*2048/(X1+caldata[MD]);
	long B5 = X1+X2;
	long B6 = B5-4000;
	X1 = (caldata[B2]*(B6*B6/4096))/2048;
	X2 = caldata[AC2]*B6/2048;
	long X3 = X1+X2;
	long B3 = (((caldata[AC1]*4+X3)<<oss)+2)/4;
	X1 = caldata[AC3]*B6/8192;
	X2 = (caldata[B1]*(B6*B6/4096))/65536;
	X3 = ((X1+X2)+2)/4;
	unsigned long B4 = caldata[AC4]*(unsigned long)(X3+32768)/32768;
	unsigned long B7 = ((unsigned long)up-B3)*(50000>>oss);
	long p;
	if(B7<0x80000000){
		p=(B7*2)/B4;
	}else{
		p=(B7/B4)*2;
	}
```

```
	X1 = (p/256)*(p/256);
	X1 = (X1*3038)/65536;
	X2 = (-7357*p)/65536;
	p += (X1+X2+3791)/16;

	return p;
}
int get_temp(int ut, int* caldata)
// 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
// 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）
{
	long X1 = (ut-caldata[AC6])*caldata[AC5]/32768;
	long X2 = caldata[MC]*2048/(X1+caldata[MD]);
	long B5 = X1+X2;	long t = (B5+8)/16;
		return t;
}
int get_raw_press(int fd, int oss)
	// 気圧の測定値を求める関数
	// oss で測定時の変換回数を指定する
{
	int m,l, x;
	// MSB, LSB, XLSB を入れる変数
	int up;
	// 計算して求めた値を入れる変数
	// oss の範囲は 0 から 3 まで
	if (oss < 0) oss = 0;
	if (oss > 3) oss = 3;
	wiringPiI2CWriteReg8(fd, CTRLREG, PRESS0 + (oss << 6));
	// 変換開始
	// 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
	switch (oss) {
		case 0:
			delay(5);
			break;
		case 1:
			delay(8);
```

```
                break;
            case 2:
                delay(14);
                break;
            default:
                delay(26);
        }
        // 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
        // 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
        m = wiringPiI2CReadReg8(fd, DATAMSB);
        l = wiringPiI2CReadReg8(fd, DATALSB);
        x = wiringPiI2CReadReg8(fd, DATAXLSB);
        // 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
        up = ((m<<16) + (l<<8) + x) >> (8-oss);
        return up;
}

int get_raw_temp(int fd){
        int m, l;
        // 読み出した MSB, LSB を入れる変数
        int ut;
        // 計算で求めた測定温度を入れる変数
        wiringPiI2CWriteReg8(fd, CTRLREG, TEMP);
        // 温度の測定開始
        delay(5);
        // 変換時間待ち、最大変換時間は 4.5ms
        // 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
                // 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
        m = wiringPiI2CReadReg8(fd, DATAMSB);
        l = wiringPiI2CReadReg8(fd, DATALSB);
        //ここに読み出した m, l から値を計算するコードを書く
        ut = (m<<8) + l;
        return ut;
}

int check_bmp180_function(int fd)
// BMP180 の機能チェックを行う関数
```

作業報告書（2022 年 1 月 16 日）

J20413 北野正樹

```
{
	int r;
	if (wiringPiI2CReadReg8(fd, IDREG) == 0x55){
		r = CHECKOK;
	}else{
		r = CHECKNG;
	}
	return r;
}

int* read_caldata(int fd,int* caldata)
// キャリブレーションデータの読み出し関数
{
	int i; int l,m;
	printf("キャリブレーション値を読み込みます。¥n");
	for (i = 0; i < CALDATANUM; i++){
		m = wiringPiI2CReadReg8(fd,CALSADR+i*2);
		l = wiringPiI2CReadReg8(fd,CALSADR+i*2+1);
		caldata[i] = (m<<8) + l;
		// 符号なし 16 ビットに変換
		if ((i != AC4) && (i != AC5) && (i != AC6)){
			// AC4, AC5, AC6 は符号付き 16 ビットデータなので対処が必要
			// その対処をここに書く
			caldata[i] = (signed short)caldata[i];
		}
	}
	return caldata;
}
```

【作業時間】

・作業時間：90 分

・報告書作成時間：30 分