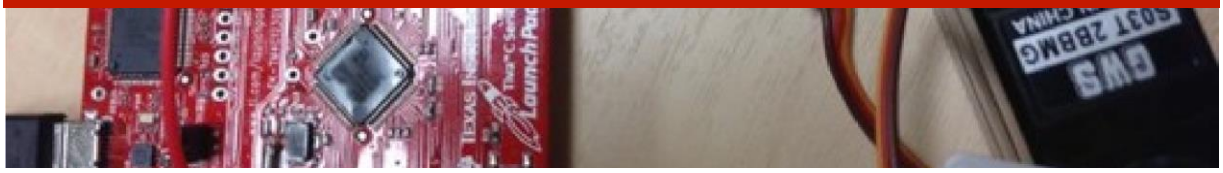
情報工学基礎実験 組み込みシステム基礎



報告者

学生番号:　　氏名:

メールアドレス:

実験実施日

実験週　　（　1　2　3　4　5　）

実施日　　＿＿＿年＿＿月＿＿日

報告書提出日　　＿＿＿年＿＿月＿＿日　受理 / 要再提出

（再提出）　　＿＿＿年＿＿月＿＿日　受理 / 要再提出

（再提出）　　＿＿＿年＿＿月＿＿日　受理 / 要再提出

自己チェック欄

☐ 実験結果は示されているか？　☐ 図表の書き方は正しいか？

☐ 考察は十分か？　☐ 演習問題はできたか？

☐ プログラムは見やすく示しているか？　☐ 課題番号等が記述されているか？

☐ 構成はテキストの指示通りになっているか？

教員所見 （redmine で指摘された内容を報告者がここにコピーして下さい）	報告者の回答

目次

1 実験機材

マイコンボードの通し番号: 72
使用したパソコン: Lenovo Ideapad slim 550（持ち込み）
OS: Windows11 Home

2 理論

2.1 PWM

PWM は，周期一定の矩形波を出力し続けながら，パルス幅を変化させる機能である．本実験においては，サーボモーターの制御に用いられる．LaunchPad には最大 16 個の PWM 波形を生成する機能があり，各波形を自由に調整することができる．PWM は 2 系統存在し，各系統別に 4 つの PWM 生成器を持っている．各生成器は 2 本の出力ピンを制御することができ，内蔵 RGB LED を制御するためにはピンの割り当てを適切に行う必要がある．

2.2 I^2C

I^2C は，低速な周辺機器をマザーボードへ接続するのに用いられる通信規格である．SCL（クロック）と SDA（データ）の 2 本の信号線によるシリアル通信であり，この信号線上に複数の機器を接続することができる．複数の周辺機器は異なるアドレスが付与され，通信の最初にアドレスを指定する．送信と受信は同時に行わず時分割する．通信速度は遅いため高速に大量のデータをやりとりするような用途には向かないが，信号線の本数を劇的に削減できるメリットがあるため，速度の要求されない表示パネルなどに利用される．

2.3 マイコンボード

本実験で扱うボードは，Tiva C Series LaunchPad の EK-TM4C123GXL というボードである．このボードには 80MHz で動作するプロセッサ，256KB のフラッシュメモリ，32KB の SRAM が搭載されており，各種 I/O を制御することができる．また，システムクロックには 16MHz の水晶発振器を利用している．主な機能はすべて中央のプロセッサに内蔵されており，プロセッサから利用できるピンを周囲のヘッダに引き出した形になっている．

2.4 LED

LaunchPad には 3 色 LED が内蔵されており，この LED は GPIO ポート F に接続されている．LED を点灯させるには割り当てられた GPIO ピンに H(1) を，消灯するには L(0) を出力する．また，PWM の制御によって LED の光量や色合いを変えられる．

2.5 ブザー

圧電ブザーは GPIO ピンに接続されており，PWM 機能を利用して周期的に ON-OFF を繰り返すパルス波形を出力することによって音を鳴らす．一般的な音階の周波数 f は次の式で与えられる．

$$f = 440 \times 2^{\frac{n}{12}} \tag{1}$$

n の値を -9 から $+3$ と変化させることで音階を表せる．また，ブザーの ON と OFF の時間を同じにするためにはパルス幅を周期の半分として与えるなど，様々な工夫をすることで多種多様な音を表すことができる．

2.6 LCD

LCD とはキャラクタ液晶ディスプレイのことである．LCD を周辺機器として接続し，LCD のアドレスに適切な命令とデータを I^2C バスに流すことで LCD の設定や文字情報を制御することができる．

3 実験内容

3.1 【課題 1】仮想シリアルポートへの出力

プロジェクト blinky を CCS にインポートした．そして，仮想シリアルポートへの出力を Terminal Plugin によって受信するために，以下のように設定した．

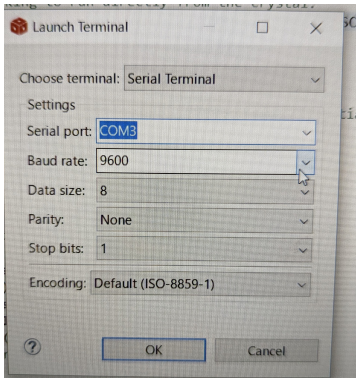


図 1: Terminal Plugin の設定

この設定の後，blinky_main.c の main() 関数内の UARTprintf() のコメントアウトを外してビルドを行い，仮想シリアルポートへの出力をターミナルを通して確認した．

3.2 【課題 2】blinky の改造

blinky_main.c の main() 関数内で GPIO 割り込みを有効化し，initInterruptPins() 関数と SW1PinIntHandler() 関数を実装し，SW1 の割り込みによって LED の点灯色を変化させるプログラムに改造した．また，このとき初期状態で有効化されていた Systick の割り込みを無効にした．

3.3 【課題 3】buzzer の改造

まず，プロジェクト buzzer をインポートし，未完成の buzzer.c において音階を表す変数を受け取ってその音を鳴らす toneBuzzer() 関数と，呼び出すことで音が鳴り止む restBuzzer() 関数を実装した．そして，blinky_main.c で initInterruptPins() 関数を実装し，SW1PinIntHandler() 関数にて SW1 を押すことで音を鳴らしたり消したりするのに加えて，押した回数に従って「ド」から音階を上げて規定回数以降にまた「ド」に戻るようにプログラムを改造した．

3.4 【課題 4】setAddressLCD() と writeTextLCD() の実装

プロジェクト lcd をインポートした後，lcd_SB1602.c に，LCD の文字位置を指定する setAddress() 関数と，LCD への文字表示を実行する writeTextLCD() 関数を実装した．そして，lcd_main.c で直書きされている文字位置の指定と文字表示の実行のコードを setAddress() と writeText() を使うように改造した．

3.5 【課題 5】SW1 を押した回数の LCD への表示

まず，initInterruptPins() 関数を実装し，SW1PinIntHandler() 関数を，Systick に割り込みを無効化した後に SW1 の押下によってカウントが実行されて LCD へカウント文字列を表示するように実装した．また，main() 関数内で GPIO 割り込みを有効化した．

4 実験結果

4.1 【課題 1】仮想シリアルポートへの出力

仮想シリアルポートへの出力を確認するためにターミナルの設定をした後 UARTprintf() のコメントアウトを外してビルドを行い実行すると，LED が青く点滅すると同時に以下のようなメッセージがターミナルで確認できた．

図 2: 【課題 1】における仮想シリアルポートへの出力内容

4.2 【課題 2】blinky の改造

以下に，blinky_main.c における initInterruptPins() のソースコードを示す．

```
1  void initInterruptPins(void) {
2      // Clear Interrupt
3      GPIOIntClear(GPIO_PORTF_BASE,INT_ALL_BUTTONS);
4
5      // Register a handler function
6      GPIOIntRegister(GPIO_PORTF_BASE,SW1PinIntHandler);
7
8      // Set type of interrupt to falling edge
9      GPIOIntTypeSet(GPIO_PORTF_BASE,INT_ALL_BUTTONS,GPIO_FALLING_EDGE);
10 }
```

ソースコード 1: 【課題 2】における initInterruptPins 関数

この関数内では，3 行目にポート F に接続された全てのスイッチによる割り込みの処理を完了させ，6 行目でポート F に SW1PinIntHandler 関数を割り当てて割り込みハンドラを指定し，9 行目で SW1 が押されたとき，すなわち波形が L(0) のときに割り込みが発生するように設定した．

次に，blinky_main.c における SW1PinIntHandler() のソースコードを示す．

```
1  void SW1PinIntHandler(void) {
2      disableSW1PinInt();
3      // Clear Interrupt
4      clearSW1PinInt();
5      SysTickIntDisable();
6
7      GPIOPinWrite(GPIO_PORTF_BASE,base_led_color,0);
8
9      int light[]={LED_BLUE,LED_RED,LED_GREEN,LED_WHITE};
10     static int cnt=0;
11     base_led_color=light[(++cnt)%4];
12     GPIOPinWrite(GPIO_PORTF_BASE,base_led_color,led_color);
13
14     enableSW1PinInt();
15 }
```

ソースコード 2: 【課題 2】における SW1PinIntHandler 関数

この関数内では，2 行目に新たな SW1 の押下による割り込みを防ぎ，4 行目で SW1 による割り込みの処理をクリアし，5 行目では Systick の割り込みを禁止した．そして，7 行目で LED を一旦消灯させて，9 行目から 12 行目にかけて LED を点灯させるコードを加えた．初期状態が青でそこから赤，緑，白，青となるように，LED の色を表すマクロ定数を配列に格納し，static 宣言されたカウント用変数 cnt の値に応じて色を選択するようにした．これらの処理を終えたあと，14 行目で SW1 の押下による新たな割り込みを許可した．

4.3 【課題 3】buzzer の改造

以下に，toneBuzzer() のソースコードを示す．

```
1  void toneBuzzer(int tone) {
2      PWMGenPeriodSet(PWM0_BASE,PWM_GEN_1,tone);
3      PWMPulseWidthSet(PWM0_BASE,PWM_OUT_3,tone>>1);
4      PWMOutputState(PWM0_BASE,PWM_OUT_3_BIT,true);
5  }
```

ソースコード 3: 【課題 3】における toneBuzzer 関数

この関数の 2 行目では出力する波形の周期を与えられた音階 tone の周期に設定し，3 行目ではパルス幅の設定を行い，4 行目でパルス波形の出力を行った．

次に，restBuzzer() のソースコードを示す．

```
1  void restBuzzer() {
2      PWMOutputState(PWM0_BASE,PWM_OUT_3_BIT,false);
3  }
```

ソースコード 4: 【課題 3】における restBuzzer 関数

この関数で, PWMOutputState 関数の 3 つ目を bool 型の false にすることで波形の出力を止めた．

また，buzzer_main.c における SW1PinIntHandler() を以下に示す．

```
1  void SW1PinIntHandler(void) {
2      GPIOIntDisable(GPIO_PORTF_BASE,INT_ALL_BUTTONS);
3      // Clear Interrupt
4      GPIOIntClear(GPIO_PORTF_BASE,INT_ALL_BUTTONS);
5
6      static int index = 0;
7      static bool flag = true;
8      int tone_set[] = {O4C, O4D, O4E, O4F, O4G};
9
10     if(flag) {
11       toneBuzzer(tone_set[index]);
12       index = (index + 1) % 5;
13     } else {
14       restBuzzer();
15     }
16
17     flag = !flag;
18
19     GPIOIntEnable(GPIO_PORTF_BASE,INT_ALL_BUTTONS);
20 }
```

ソースコード 5: 【課題 3】における SW1PinIntHandler 関数

この割り込みハンドラでは，2 行目と 4 行目で新たな割り込みを無効化し，割り込みをクリアした．また，6 行目から 17 行目にかけて SW1 を押した回数によって音を鳴らしたり止めたりすると同時に音階を周期的に変えることを行った．そして，19 行目で新たな割り込みの許可をした．

4.4 【課題 4】setAddressLCD() と writeTextLCD() の実装

以下に，setAddressLCD() のソースコードを示す．

```
inline void setAddressLCD(uint8_t x, uint8_t y) {
  uint8_t cursor = x | (y * 0x40);
  uint8_t command[] = {SB1602_COMMAND_SINGLE, 0x80 | cursor};
  writeDataI2C(I2C3_BASE, SB1602_SLAVE_ADDRESS, command, 2);
}
```

ソースコード 6: 【課題 4】における setAddressLCD 関数

この関数では，2 行目に引数で受け取った LCD への表示位置の情報を現在のカーソルとして変数 cursor に格納し，3 行目では指定したカーソル位置をコマンドとして送るのに必要な配列 command[] に適切な初期化を行い，4 行目で I^2C バスを通して LCD にカーソル位置を指定した．

次に，writeTextLCD() のソースコードを示す．

```
inline void writeTextLCD(uint8_t *text, uint8_t length) {
  uint8_t command[17];
  command[0] = SB1602_DATA_BURST;
  int i;
  for (i = 0; i < length; i++){
    command[i + 1] = text[i];
  }
  writeDataI2C(I2C3_BASE, SB1602_SLAVE_ADDRESS, command, length + 1);
}
```

ソースコード 7: 【課題 4】における writeTextLCD 関数

まず 2 行目に，送るデータの形式を表すマクロ定数と，引数で与えられた文字列の情報を格納するために要素数を 17 とした配列 command[] を定義した．3 行目から 7 行目では，先頭にデータを連続的に送ることを意味するマクロ定数を格納し，以降は文字列の長さに応じて文字を格納した．そして 8 行目では I^2C バスを通して LCD に指定したカーソル位置から文字列を表示するコマンドを送った．

4.5 【課題 5】SW1 を押した回数の LCD への表示

以下に，lcd_main.c における SW1PinIntHandler() のソースコードを示す．

```
void SW1PinIntHandler(void) {
  SysTickIntDisable();
  disableSW1PinInt();
  clearSW1PinInt();

  static uint8_t spaces[] = {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '
      ', ' ', ' ', ' '};

  setAddressLCD(0, 0);
  writeTextLCD(spaces,16);
  setAddressLCD(0, 1);
  writeTextLCD(spaces,16);

  static uint8_t counter[] = {'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '
      0', '0', '0', '0'};
```

```
14    counter[15]++;
15
16    uint8_t ptr;
17    for(ptr = 15; ptr > 0; ptr--){
18      if(counter[ptr] > '9') {
19        counter[ptr] = '0';
20        counter[ptr - 1]++;
21      } else break;
22    }
23
24    if(counter[ptr] > '9') {
25      for(ptr = 0; ptr < 15; ptr++) counter[ptr] = '0';
26    }
27
28    uint8_t head = 0;
29    while(head < 15 && counter[head] == '0') head++;
30    uint8_t counter_length = 16 - head;
31
32    setAddressLCD(head, 0);
33    writeTextLCD(&counter[head], counter_length);
34
35    enableSW1PinInt();
36  }
```

ソースコード 8: 【課題 5】における SW1PinIntHandler 関数

この関数では，2 行目に Systick による割り込みを禁止し，3，4 行目で SW1 による割り込みの禁止とクリアをした．また，6 行目から 11 行目にかけて LCD への表示を一旦消し，13 行目から 33 行目にかけて SW1 を押した回数の更新とその数の LCD への表示を行った．配列 counter[] はカウント数を文字列として格納する配列で，表示位置は変数 head で指定し setAddressLCD() と writeTextLCD() に適切な引数を与えた．そして，35 行目で SW1 による割り込みを許可した．

5 考察

【考察 1】 GPIO 割り込みは GPIO ポートの電圧の変化を読んで割り込みを認識し，これにより外部からの入力を検知して割り込みを発生させる．今回の実験では，GPIO 割り込みは SW1 スイッチの押下による割り込みを発生させる役割を担うと考えられる（外部割り込み）．また，Systick は 24 ビットのダウンカウンタであり，カウンタが 0 になると割り込みが発生する．そのため．Systick 割り込みはタイマーとしての役割を担うと考えられる（内部割り込み）．[1]

【考察 2】 周辺機器の接続に I^2C を使用すると，機器の master と slave を自由に決めることができる，2 本の信号線のみを用いて機器を接続できる，などの利点がある．[1]

【考察 3】 割り込みが発生したときだけセンサ読み込みを可能にする実装により，プロセッサのリソースの効率化，割り込みベースによるシステムの単純化，エネルギー効率化などが考えられる．センサが常に監視状態として働いているとそれだけで電力の消費が大きくなるが，割り込みの採用によりセンサが検知していないときはアイドル状態となり電力の消費を減らすことができる．

また，監視に割いていたリソースをその他の処理に割けるという点でリソースの効率化が見込める．割り込みを用いない場合，センサの処理はポーリング方式などで行う必要があるが，多数のセンサに対応させられるか，検知した瞬間に反応できるか，といった問題が生じる．その点，割り込みの採用によりこれらの問題の解消が見込める．

6 問題回答

6.1 問題 1

LED の各色について On,Off の 2 通りを行い，全部消灯している状態は含まないので，求める総数は

$$2^3 - 1 = 7 \text{ 通り}$$

6.2 問題 2

問題 1 における On,Off の 2 通りから 2^8 通りの制御に変わっただけなので，求める総数は

$$(2^8)^3 - 1 = 16777215 \text{ 通り}$$

6.3 問題 3

ソースコード 3 を見ると，周期の長さの設定は PWMGenPeriodSet() で行っている．また，パルス幅は PWMPulseWidthSet() で行っており，3 つ目の引数は tone»1 となっているため tone はパルス波の周期を表していると考えられる．よって 1 オクターブ音を高くするには音階の周期を表すマクロ定数（O4C 等）を $\frac{1}{2}$ 倍して toneBuzzer() に渡すと良い．

6.4 問題 4

まず，何も抵抗が繋がれていない状態では外部からの静電気などで壊れやすいため抵抗に繋いでおくと不安定を防ぐことができる．また，プルアップ方式とプルダウン方式のどちらも割り込みによって HIGH と LOW が切り替わるが，プルアップ方式は何もしていないときに HIGH を保ち，かつ外乱によるノイズなどにも影響されず HIGH を保つ．以上のことから割り込みを受けるピンにおいてプルアップするほうが良い．

6.5 問題 5

アドレスが n ビットで表されるとすると，接続可能な機器は理論上で 2^n 個である．

6.6 問題 6

delay_ms() 引数に与えた uint32_t 型の数（マイクロ秒単位）の分だけプログラムを遅らせる

itoh() uint32_t 型の数値を 16 進数表記の文字列に変換し，それを格納した配列を返す
itoa() int32_t 型の数値を 10 進数表記の文字列に変換し，それを格納した配列を返す

参考文献

[1] Texas Instruments Incorporated. *TM4C123GH6PM Microcontroller DATA SHEET*. TEXAS INSTRUMENTS. 2014. https://www.ti.com/lit/ds/spms376e/spms376e.pdf, (accsessed 2023-10-31)