# 組込みシステム(HI-4)

# 後期期末レポート課題２

HI4 45 号 山口惺司

提出日: 2025/2/4

## 1. ストップウォッチに求められる要件について

ストップウォッチとして求められている要件は以下の通りである。

- ボタンの役割と動作
  - スタート及びストップボタン
    - スタート：カウントを開始する
    - ストップ：カウントを停止する
  - リセットボタン
    - リセット：カウントを初期化する
- 時間の表示形式
  - hh:mm:ss.d で表示し、初期状態は 00:00:00.0 である。
- 時間間隔
  - 0.1s 間隔でカウントを行うようにする。
- 状態遷移図及び状態遷移表

  作成した状態遷移図を図 1, 状態遷移表を表 1 に示す.

  SW_start, SW_stop は同じボタンを使用している.

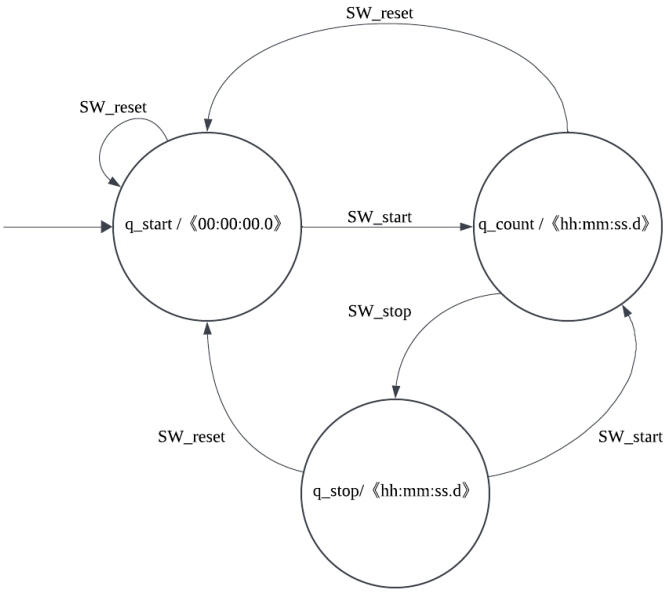


図 1 状態遷移図

表 1 状態遷移表

| | SW_start & SW_stop | SW_reset | 出力 |
|---|---|---|---|
| →q_start | q_count | q_start | 《00:00:00.0》 |
| q_count | q_stop | q_start | 《hh:mm:ss:d》（カウントアップ） |
| q_stop | q_count | q_start | 《hh:mm:ss:d》（カウント停止） |

## 2. 液晶ディスプレイについて

- 液晶ディスプレイの接続端子
  - Vss：GND

- Vdd：電源 5V
- V0(NC)：液晶コントラスト調整
- SDA：I2C 通信(Serial input data)
- SCL：I2C 通信(Serial clock input)
- BL+：バックライト用電源 5V
- BL-：バックライト用電源 GND

- LCD に表示するための使用方法
  - 初期設定方法

```
#include <Wire.h>
const uint8_t LCD_I2C_ADR = 0x3F;
void setup()
{
  Wire.begin();

  putCMD(0x38);
  delay(20);
  clear_display();
  cursor_home();
  cursor_disp_on();
}
```

  - カーソルの位置を設定する関数

```
void cursor(int line, int col) {
  uint8_t pos = 0;

  if (line==1)
    pos += 40;
  else if (line==2)
    pos += 20;
  else if (line==3)
    pos += 84;

  pos += col;

  putCMD((0x80) | pos); //set cursor position
}
```

  - 文字列を LCD に表示する関数

```
void disp_string(char str[]) {
  for (int i=0; str[i]!=NULL; i++)
    putDAT(str[i]);
}
```

- 画面をクリアする関数

```
void clear_display(void) {
  putCMD(0x01);
  delay(20);
}
```

- カーソルをホーム位置（画面の左上)に設定する関数

```
void cursor_home(void) {
  putCMD(0x02);
  delay(20);
}
```

- カーソルの表示をオンにする関数

```
void cursor_disp_on(void) {
  putCMD(0x0C);
  delay(20);
}
```

- コマンドを LCD に送信する関数

```
void putCMD(uint8_t dat) {
  Wire.beginTransmission(LCD_I2C_ADR);
  Wire.write(0x00);
  Wire.write(dat);
  Wire.endTransmission();
}
```

- 文字を LCD に送信する関数

```
void putDAT(uint8_t dat) {
  Wire.beginTransmission(LCD_I2C_ADR);
  Wire.write(0x40);
  Wire.write(dat);
  Wire.endTransmission();
}
```

## 3. 回路について

- 回路図

  作成した回路図を図 2 に示す.

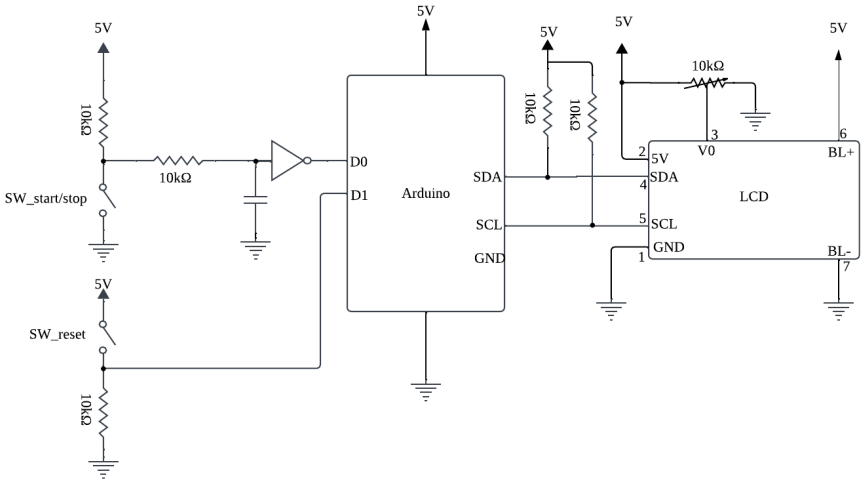


図 2 回路図

- チャタリング対策
  - チャタリングについて
    チャタリングとはスイッチをオンにする瞬間に微細な振動が発生して短時間にオンとオフを繰り返す現象である。
  - チャタリング対策とその効果
    チャタリング対策は図 2 回路図の左側(図 3)で行われている.

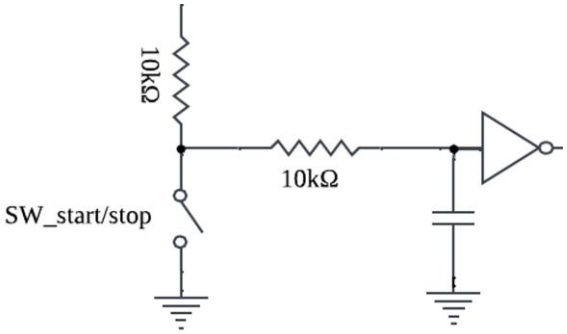


図 3 チャタリング対策回路

今回はスタート及びストップボタンにチャタリング対策をしている。
それによってスタートとストップが短時間に切り替わるような誤作動を防ぐことができる。
また、今回リセットボタンについてはチャタリング対策をしていない。
理由としては、チャタリングしても動作に支障がないためである。

## 4. 実行結果について

- プログラム
  作成したプログラムを以下に示す.

```
#include <Wire.h>
#include <AGTimerR4.h>
```

```
#define SW_reset 2 //D0
#define SW_stop 3 //D1

#define q_start 0
#define q_count 1
#define q_stop 2

const uint8_t LCD_I2C_ADR = 0x3F;   // 液晶 LCD の I2C アドレス

volatile int state;

volatile int count = 0;
volatile int hour = 0;
volatile int minute = 0;
volatile int sec = 0;
volatile int dec = 0;

void setup()
{
  Serial.begin(115200);
  Wire.begin();
  AGTimer.init(100000, timerCallback);

  pinMode(SW_reset, INPUT);
  pinMode(SW_stop, INPUT);

  state = q_start;
  attachInterrupt(0, resetTimer, RISING);
  attachInterrupt(1, stopTimer, RISING);

  // LCD 初期設定
  putCMD(0x38); //Function Set
  delay(20);
  clear_display();
  cursor_home();
  cursor_disp_on();
}

void timerCallback(void) {
  char timer[30];

  hour = count / 36000;
```

```
  minute = (count / 600) % 60;
  sec = (count / 10) % 60;
  dec = count % 10;

  interrupts();

  clear_display();

  cursor(0,0);
  disp_string("KUMAMOTO KOSEN");

  sprintf(timer, "%02d:%02d%:%02d.%d", hour, minute, sec, dec);

  cursor(1,0);
  disp_string(timer);

  if(state == q_count){
    count++;
  } else if(state == q_start){
    count = 0;
  }

}

void cursor(int line, int col) {
  uint8_t pos = 0;

  if (line==1)
    pos += 40;
  else if (line==2)
    pos += 20;
  else if (line==3)
    pos += 84;

  pos += col;

  putCMD((0x80) | pos); //set cursor position
}

void disp_string(char str[]) {
  for (int i=0; str[i]!=NULL; i++)
    putDAT(str[i]);
```

```
}

void clear_display(void) {
  putCMD(0x01);
  delay(20);
}

void cursor_home(void) {
  putCMD(0x02);
  delay(20);
}

void cursor_disp_on(void) {
  putCMD(0x0C);
  delay(20);
}

void putCMD(uint8_t dat) {
  Wire.beginTransmission(LCD_I2C_ADR);
  Wire.write(0x00);
  Wire.write(dat);
  Wire.endTransmission();
}

void putDAT(uint8_t dat) {
  Wire.beginTransmission(LCD_I2C_ADR);
  Wire.write(0x40);
  Wire.write(dat);
  Wire.endTransmission();
}

void resetTimer(){
  state = q_start;
}

void stopTimer(){
  if(state == q_start || state == q_stop){
    state = q_count;
  } else if(state == q_count){
    state = q_stop;
  }
}
```

```
void loop() {

}
```

- プログラムの説明
  - 割り込み処理について

    今回は割り込みを使った.

    ```
    attachInterrupt(0, resetTimer, RISING);
    attachInterrupt(1, stopTimer, RISING);
    ```

    によって SW_stop(D0), SW_reset(D1)が押されたときそれぞれ resetTimer(), stopTimer()関数を実行されるようになっている。

    resetTimer(), stopTimer()は状態遷移図と同じ動きをするように処理している。

  - タイマーについて

    ```
    AGTimer.init(100000, timerCallback);
    ```

    によって 0.1s 間隔で timerCallback()が実行される。

    timerCallback()では int 型の変数 count を用意し、q_count 状態だと 1 周期毎に値を 1 ずつ増やしていて、q_stop 状態だと値を増やさなくなる。

    q_start 状態の時は count を 0 にしている。

    hour, minute, sec, dec は count を用いてそれぞれの時間に変換している。

- 実行結果

  初期状態を図 4 にカウントしているときの実行結果を図 5, 6 に示す。

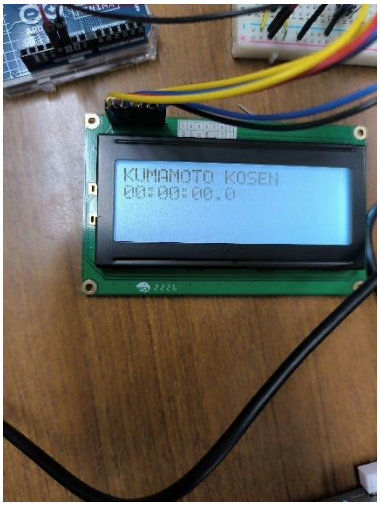


図 4 初期状態


図 5 カウント状態


図 6 カウント状態 2

- 考察

  図 5, 6 を見るとカウントが進んでいるため、タイマーは正しく動作していると言える。

  また、カウント状態の時にスタート及びストップボタンを押すと、カウントが止まり、再び押すとカウントがスタートすることが確認できた。

  カウント状態の時にリセットボタンを押すと図 4 のように初期状態に遷移することも確認できた。

以上より、正しいストップウォッチを作成することができたと言える。