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.

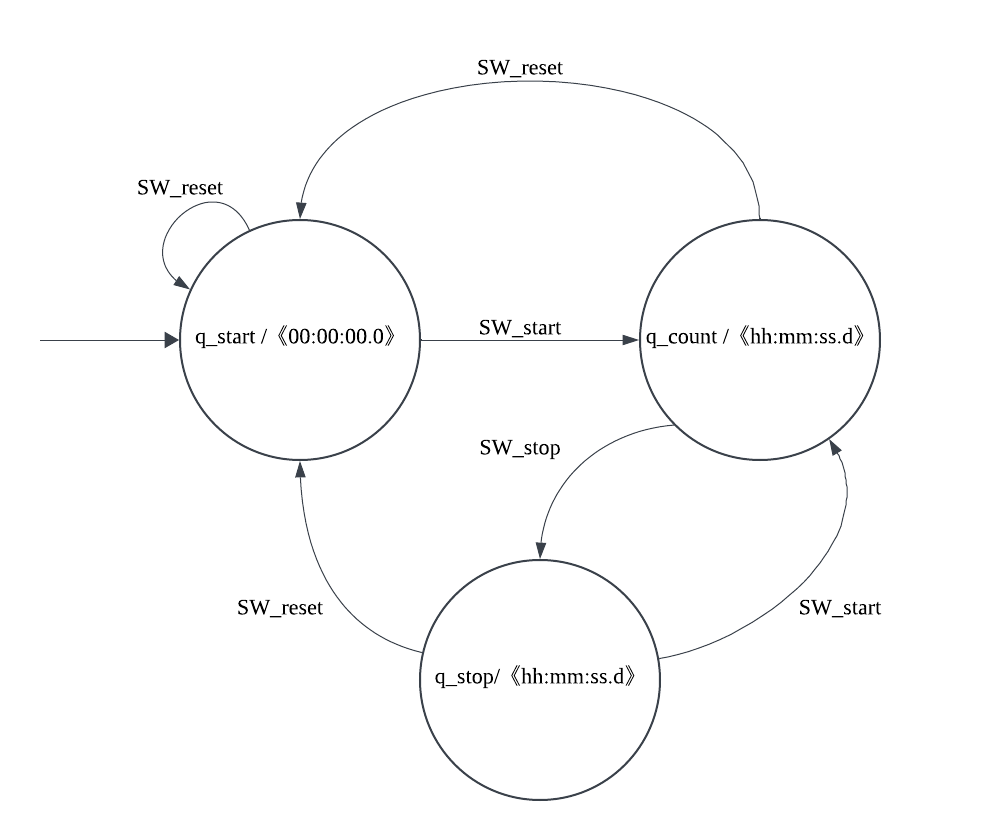


図1 状態遷移図

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) |

1. 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();

}

1. 回路について

* 回路図

作成した回路図を図2に示す.

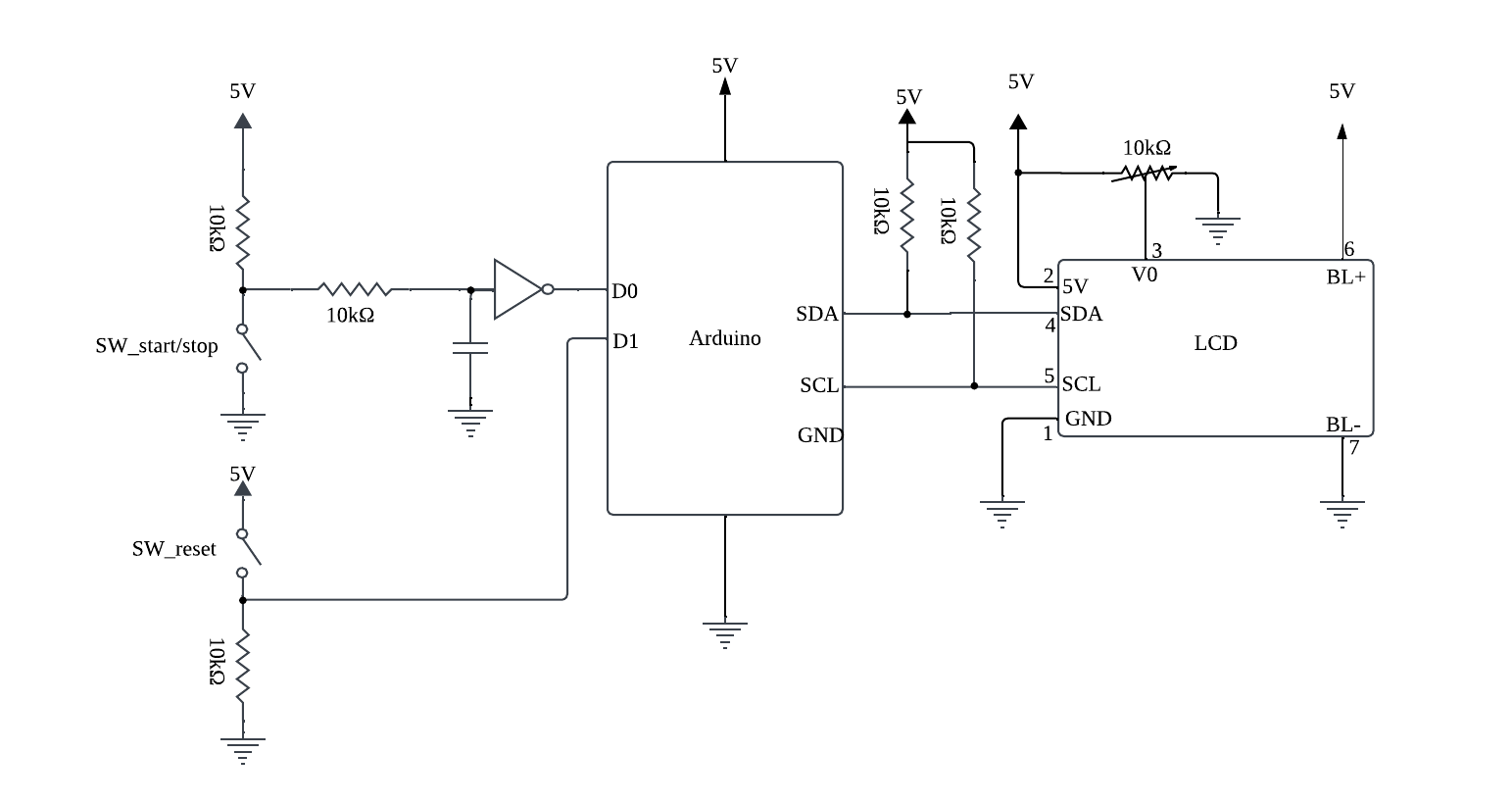


図2 回路図

* 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.

ダイアグラム, 概略図

自動的に生成された説明

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。

1. 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。

コンピュータ, テーブル, 座る, グリーン が含まれている画像

自動的に生成された説明 コンピュータ, テーブル が含まれている画像

自動的に生成された説明 回路 が含まれている画像

自動的に生成された説明

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。