به نام خدا

3/12/2021

گزارش آزمایش شماره 7 (ماشین لباس شویی ساده)

آزمایشگاه ریزپردازنده و زبان اسمبلی

محمد جواد زندیه , ابوالفضل بکیاسای کیوی
دانشگاه صنعتی امیرکبیر دانشکده مهندسی کامپیوتر

پرسش: در چه کاربرد هایی EEPROM به کار برده می شود؟ چرا در اینجا حافظه Flash یا RAM را به کار نمی بریم؟ تفاوت RAM با EEPROM در چیست؟

حافظه EEPROM یک حافظه non-volatile می باشد به این معنا که اطلاعات موجود در آن با قطع برق سیستم از بین نمی رود، در حالی که حافظه RAM حافظه ای volatile است یعنی با قطع برق سیستم اطلاعات موجود در آنها پاک می شود. در واقع RAM اطلاعات را به صورت موقت در خود نگهداری می کنند. علت استفاده از EEPROM در این آزمایش این است که قرار است تنظیمات مربوط به ماشین لباس شویی در حافظه ای دائمی نگهداری شود تا نیاز نباشد که هر دفعه مجدد تنظیمات را انجام دهیم و در واقع مد های کاری را ذخیره شده داشته باشیم. در گذشته اگر می خواستیم که اطلاعات یک بلاک از حافظه های Flash را پاک کنیم میبایست کل حافظه را پاک میکردیم و تمام مقادیر گذشته به علاوه تغییرات جدید را مجدد می نوشتیم و این برای مواقعی که بخواهیم تنها بخشی از تنظیمات را تغییر دهیم مناسب نمی باشد و به همین دلیل از آن استفاده نمی کردیم(البته الان حافظه های flash را می توان فقط بلاک های خاصی از آن را پاک کرد و rewrite کرد.)

تفاوت های RAM و EEPROM :

RAM مخفف Random Access Memory می باشد و یک حافظه فرار می باشد که دسترسی به اطلاعات آن با سرعت زیاد امکان پذیر است که امکان نوشتن و خواندن همزمان از آن وجود دارد.

EEPROM مخفف electrically erasable programmable read-only memory می باشد که حافظه ای دائم است و کاربران قابلیت این را دارند که خود این حافظه را erase کرده و دیتای جدید را در آن قرار دهند. برخلاف اسم آن که read-only مانده است در طول تغییرات و تکامل آن اما در حال حاظر می توان داده های آن را پاک کرد و مجددا در آن دیتا نوشت.

پرسش: اگر بخواهیم برای نگهداری مدهای کاری حافظه Flash را به کار ببریم، فرآیند نوشتن باید چگونه انجام شود که داده های دیگری که بر روی همان بلاک هستند از دست نروند؟

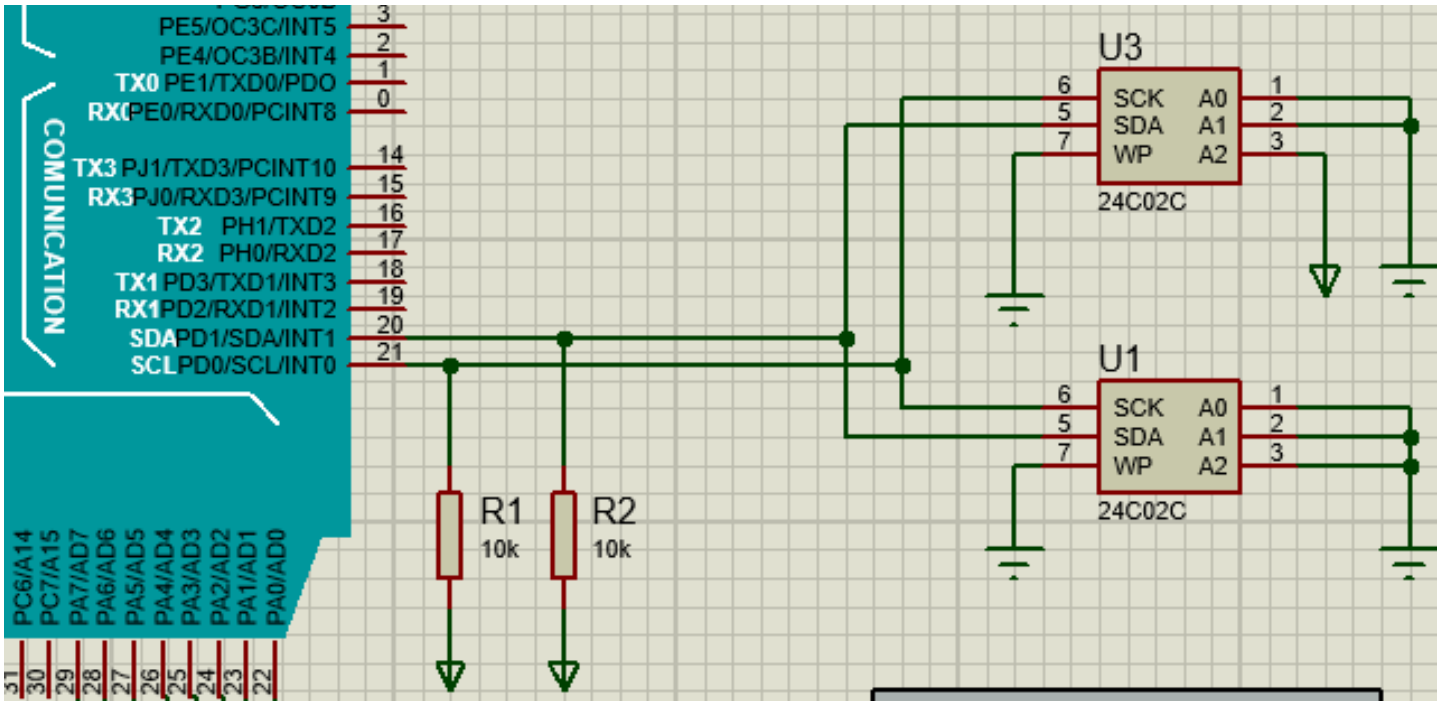
در حافظه های Flash قدیم چون عمل پاک کردن موجب پاک شدن کل یک بلاک می شد پس برای تغییر یک ردیف از یک بلاک باید مقادیری که نمیخواستیم تغییر کند در هنگام نوشتن مقادیر جدید مجدد باز نویسی می کردیم. البته الان فلش های جدیدی آمده است.

today's Flash memories usually require less complex programming algorithms and they are now divided into several sectors. The benefit of having sectors is that the Flash memory is sector-erasable, meaning you can erase one sector at a time. In the past, erase commands erased the entire memory chip

پرسش: اگر یک حافظه EEPROM بیرونی دارای 4KB حافظه و 2 پایه آدرس باشد در این صورت می توان حداکثر چند KB حافظه EEPROM بیرونی بر روی یک باس مشترک داشت؟

با توجه به اینکه 2 پایه آدرس داریم می توان در یک باس مشترک 4 عدد Slave داشته باشیم و در نتیجه با داشتن 4 حافظه بیرونی میتوان 16KB حافظه بیرونی روی یک باس مشترک داشت.

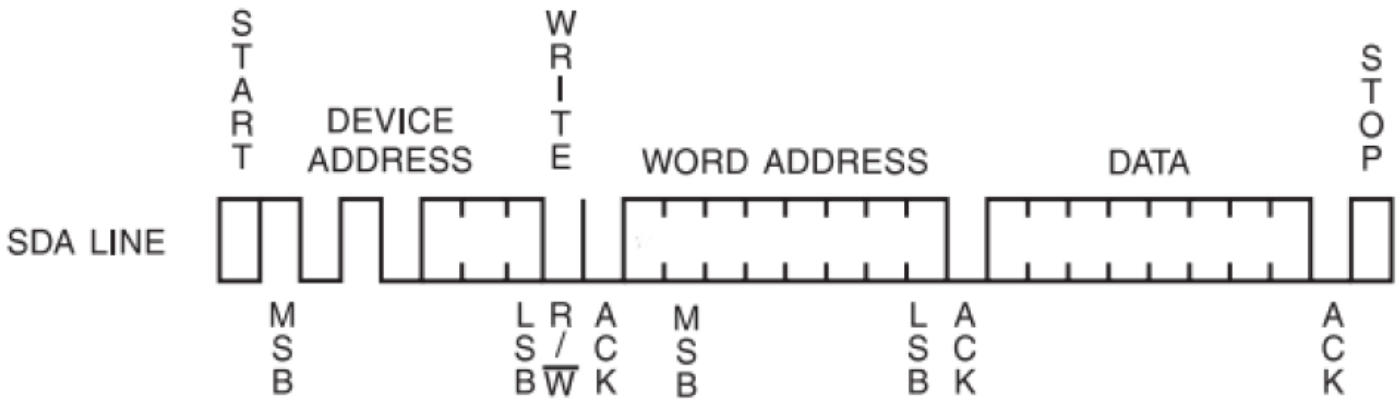
پرسش: نمودار شماتیک برای این که دو AT24C02 را به یک باس مشترک وصل کنیم و حفاظت نوشتن غیر فعال باشد را رسم کنید. (آدرس دهی سخت افزاری دل خواه - باس را هم به پایه های میکروکنترلر متصل کنید)



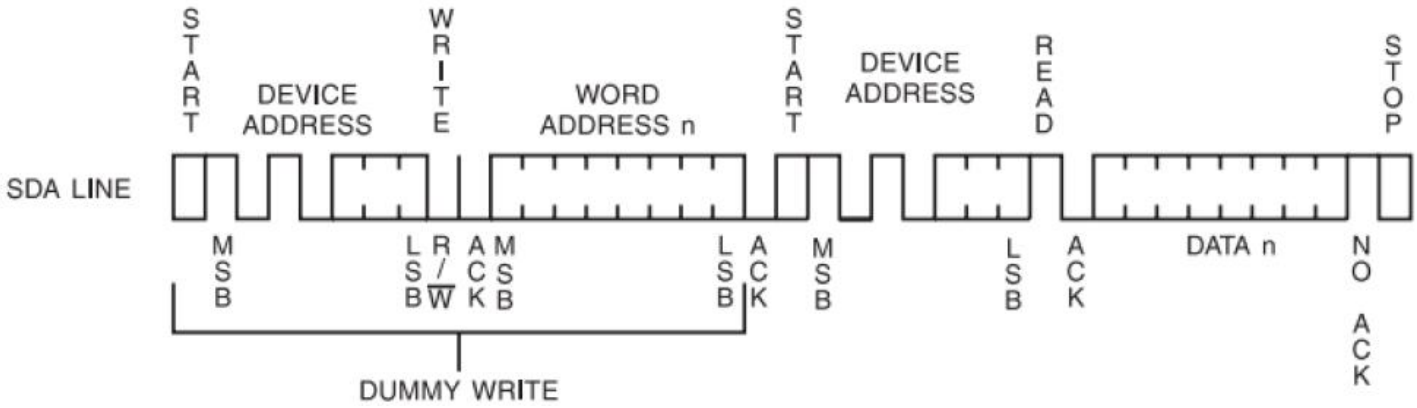
پرسش: هم خوانی این دنباله فریم ها را با پروتکل TWI بررسی کنید. (فریم های آدرس و داده را مشخص کنید، دستور خواندن یا نوشتن چگونه مشخص می شوند؟)

برای برقراری یک ارتباط از طریق پروتکل I2C از یک رابط به نام TWI استفاده می شود. در این پروتکل یک خط داده به نام SDA یا serial data line و یک خط کلاک به نام SCL یا serial clock line استفاده می شود. برای آنکه master بتواند مشخص کند که ارتباط آغاز شده است ابتدا یک start bit می فرستد که با تغییر SDA به صورت high-to-low اتفاق می افتد. سپس master باید مشخص کند که با کدام یک از slave ها قرار است در ارتباط باشد و در نتیجه آدرس slave مورد نظر را در خط SDA قرار می دهد که در میکرو ما آدرس های مربوط به device های خارجی 7 بیتی هستند که در TWI فقط میتوان به 8 دستگاه متصل بود به صورت حد اکثر و 4 بیت پرارزش این 7 بیت آدرس به صورت پیش فرض با مقادیر باینری 1010 پر شده است در نتیجه در یک باس مشترک دیتا، حداکثر میتوان به 8 slave متصل بود.
بعد از ارسال آدرس دستگاه خارجی باید مشخص کنیم که masterکه در اینجا همان میکروکنترلر ما میباشد قرار است عملیات نوشتن در slave (که اینجا همان حافظه EEPROM هست) را انجام دهد یا قرار است از آن داده ای را بخواند. با قرار دادن مقدار بیت 0 میگوییم که قرار است میکرو دیتا را بفرستد و با قرار دادن مقدار بیت 1 قرار است میکرو دیتا را بگیرد(R/W_bar)
سپس از طرف slave یک بیت ack به نام اینکه آماده ارتباط است فرستاده می شود و داده ها با توجه به اینکه کدام یک قرار است فرستنده و یا گیرنده باشد در خط SDA فرستاده می شوند و در انتها هم برای پایان دادن به ارتباط، master یک stop bit ارسال می کند که به صورت low-to-high در خط SDA ظاهر می شود.

نوشتن بایتی



خواندن بایت تصادفی



پرسش: فرکانس کلاک در کدام دستگاه پیکربندی می شود؟ کلاک را کدام دستگاه فراهم می کند؟ با توجه به زمان مورد نیاز برای انجام عملیات نوشتن، با فرض این که کلاک را 10KHz تنظیم کرده باشیم، در این صورت حداکثر با چه نرخی می توان عملیات نوشتن را انجام داد؟

به طور کلی master به عنوان دستگاه آغاز گر و پایان دهنده به ارتباط و همچنین فراهم کننده کلاک می باشد در واقع کنترل کننده اصلی ارتباط همان master می باشد که در این آزمایش میکرو همان master می باشد (میکرو ها می توانند در ارتباط با یکدیگر نقش slave هم داشته باشند که در این جا بررسی نمی کنیم)
پس پیکربندی و تامین کلاک بر عهده master است و خط SCL به عنوان خروجی از master است و به عنوان ورودی در slave ها می باشد.
حدود 30 بیت در هر عمل نوشتن یک بایت داده در یک آدرس مشخص از AT24C02 نیاز است. پس یعنی برای هر بایت نوشتن نیاز به 30 کلاک داریم:

$$T = \frac{1}{f} = 10^{-4}s \rightarrow 1\ byte\ in\ 30\ clock \rightarrow rate = \frac{1}{30 * 10^{-4}} = 333\ byte/s$$

پرسش: هر یک از تابع های نوشته شده را از راه لینک کتابخانه Wire، در مستندات آردوینو بررسی کنید و کد لازم برای تولید دنباله فریم ها برای عمل نوشتن و خواندن گفته شده با این تابع ها را بنویسید.

- begin()
- setClock()
- beginTransmission()
- write()
- endTransmission()
- requestFrom()
- available()
- read()

تابع begin : فعال کردن واحد I2C در میکرو

```
wire.begin(address_of_slave)
```

در slave تابع begin را با آدرس آن begin می کنیم

```
wire.begin(); //for master
```

در master تابع begin را بدون پارامتر ورودی صدا می زنیم.

تابع setClock : تغییر فرکانس کلاک در ارتباط I2C

تابع هایی که برای ارسال داده از master به slave استفاده می شود:

تابع beginTransmission : آغاز ارتباط برای ارسال دیتا به slave با آدرس مشخص شده از slave به عنوان پارامتر ورودی

```
wire.beginTransmission(slave_address)
```

تابع write : پس از آغاز ارتباط توسط تابع قبل باید بایت ها را توسط این تابع در صف ارسال قرار دهیم

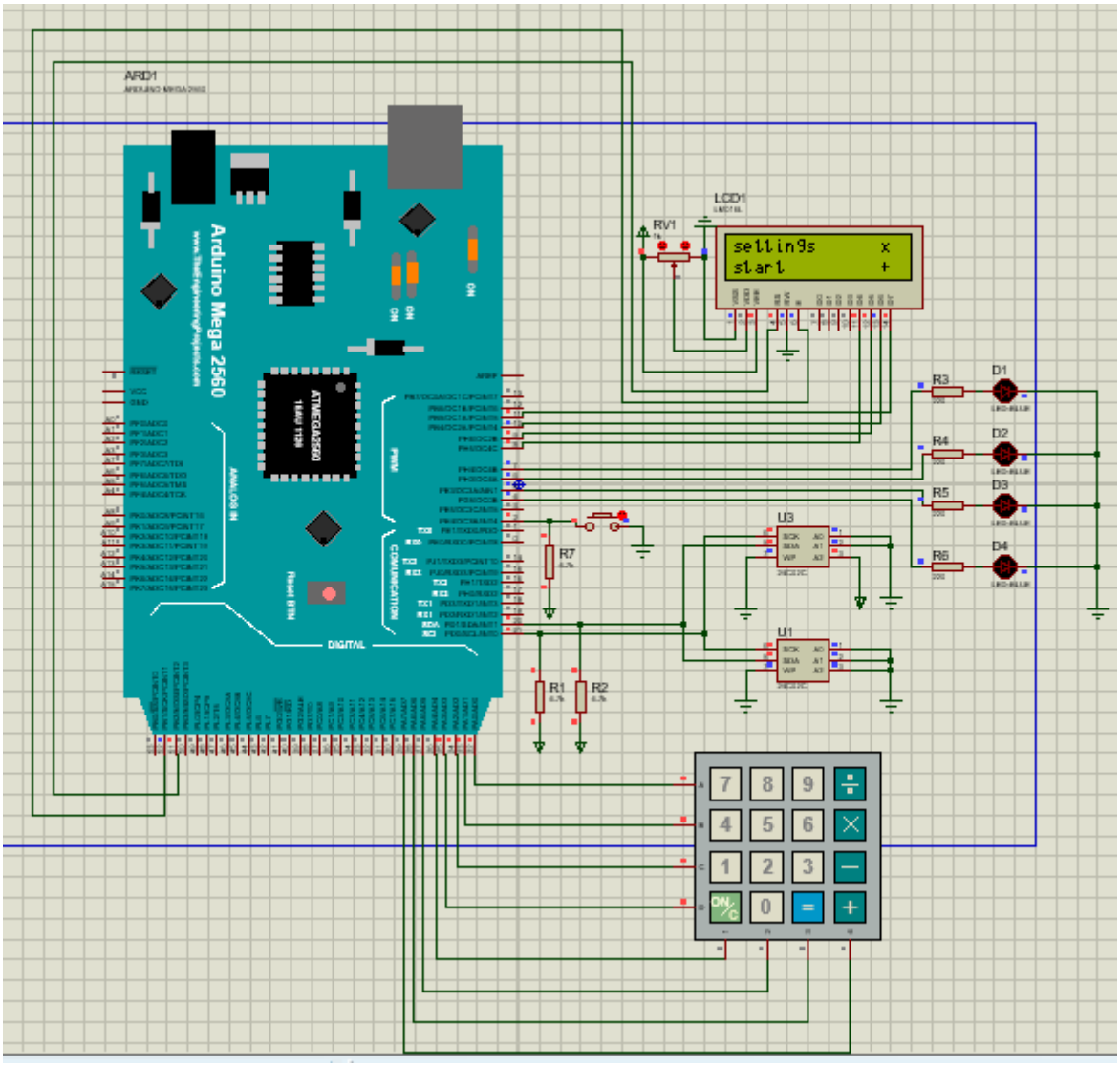
تابع endTransmission : ارسال داده هایی که توسط write در صف ارسال قرار گرفته بودند.

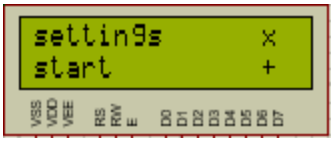
تابع هایی که برای ارسال داده از slave به master استفاده می شود:

تابع requestFrom : مستر توسط این تابع از slave درخواست می کند که داده ها را بفرستد.

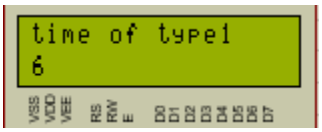
تابع available : داده هایی که توسط slave فرستاده شده بود توسط این تابع بازیابی می شوند(تعداد بایت هایی که موجود است تا توسط تابع read بازیابی شود را بر می گرداند)

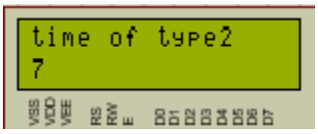
تابع read : بایت هایی که توسط slave فرستاده شده بود را بعد از فراخوانی requestFrom بازیابی می کند.

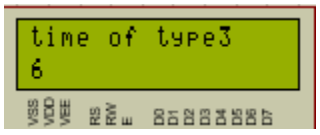


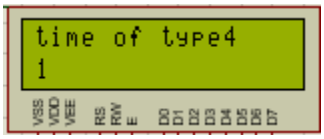


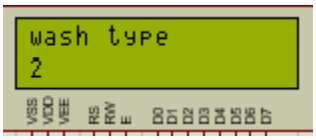
اگر کلید * را در کیبورد فشار دهیم به منوی زیر می رویم که میتوان زمان هر یک از 4 گام شست و شو و اینکه میخواهیم از کدام گام شروع کنیم را مشخص کنیم:



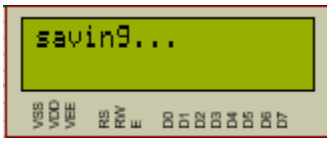






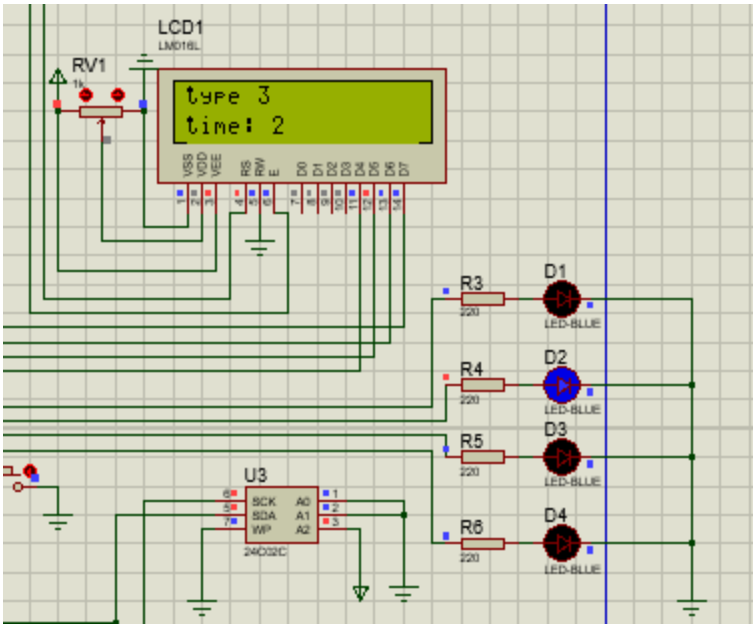


در پایان هم با نمایش ...saving نشان داده می شود که تنظیمات در حال ذخیره شدن هستند

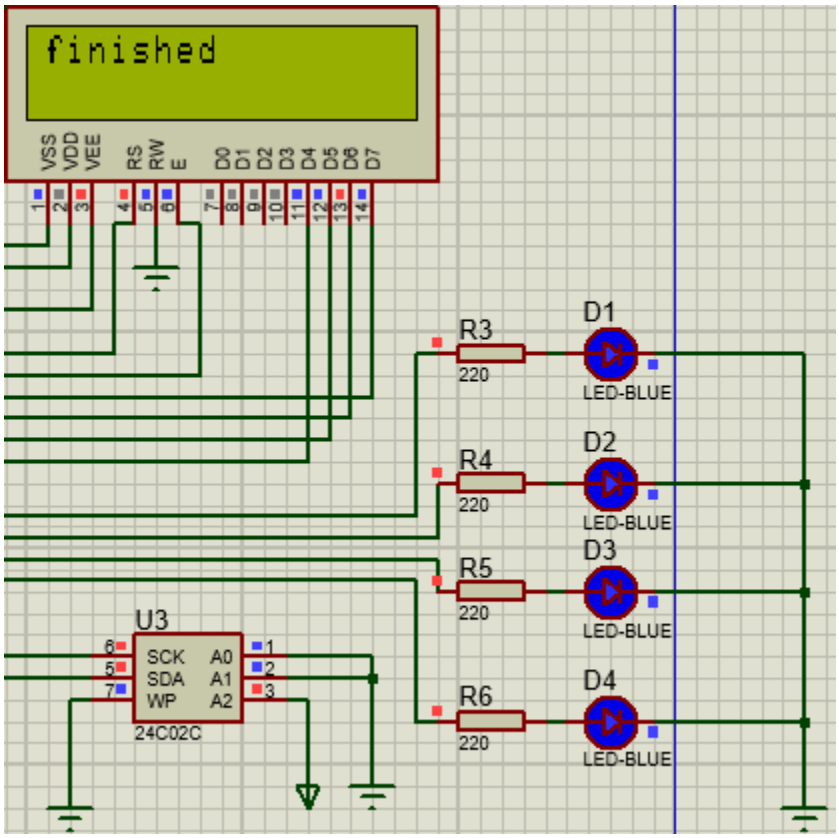


باز زدن کلید + در کیبورد اگر در منوی اولیه باشم آنگاه عملیات بازیابی اطلاعات انجام می شود و از جایی که پیش از این متوقف شده بود یا آنکه در تنظیمات اولیه انجام شده بود شروع می کنیم.

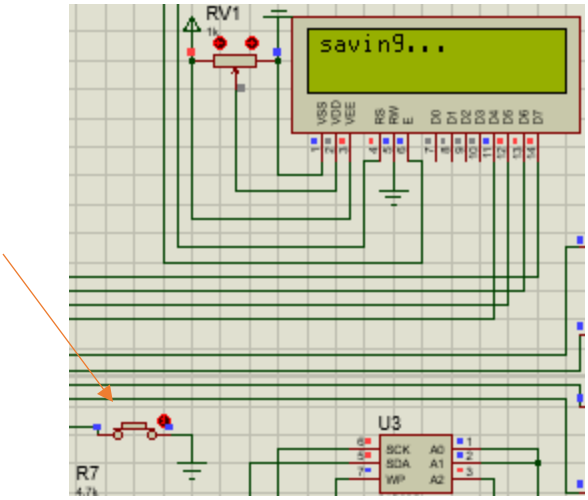
سپس طبق تنظیمات به ترتیب از جایی که معلوم کرده بودیم شست و شو انجام شود شروع به شمارش معکوس می کند. همزمان مرحله ای که در حال شست و شو هستیم را در led ها نشان می دهیم.



پس از پایان کار و در انتها با نمایش finish و روشن کردن کل led ها اعلام پایان کرده و مجدد با منوی اولیه باز می گردیم و led ها هم به طبع همگی خاموش می شوند.



همچنین در حین مراحل می توان با استفاده از push button موجود یک وقفه به عنوان وقفه از برق کشیدن و متوقف کردن کار بدهیم و در این زمان تمامی اطلاعات کنونی زمان ها ذخیره شده و مرحله شست و شوی کنونی هم ذخیره می شود و دوباره به منوی اصلی باز می گردیم و با شروع مجدد از جایی که توقف انجام شده بود کار ادامه می یابد.



دقت شود که push buttonباید حدود یک ثانیه نگه داشته شود چون از سیاست polling برای هندل کردن وقفه استفاده شده است البته میتوانستیم از interrupt-driven هم استفاده کنیم.

(دقت شود با اتمام simulation و شروع مجدد یک simulation دیگر همچنان تنظیمات ذخیره شده در eeprom موجود هستند زیرا eeprom یک حافظه خارجی متصل به برد ما هست و به non-volatile هست یعنی با قطع برق اطلاعات آن از بین نمی رود)

کد برنامه با استفاده از Arduino :

```
#include <Keypad.h>
#include <LiquidCrystal.h>
#include <Wire.h>

// ------------------------------------------------------
----
// keypad object initializing
const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
{'7','8','9','/'},
{'4','5','6','*'},
{'1','2','3','-'},
{'C','0','=','+'}
};
byte rowPins[ROWS] = {22, 23, 24, 25};
```

```
byte colPins[COLS] = {26, 27, 28, 29};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins,
ROWS, COLS);
#define INT_PIN 2
// ------------------------------------------------------------
----
// define lcd pins
#define RS_PIN 51
#define EN_PIN 52
#define D4_PIN 8
#define D5_PIN 9
#define D6_PIN 10
#define D7_PIN 11
// lcd object initializing
LiquidCrystal lcd(RS_PIN, EN_PIN, D4_PIN, D5_PIN, D6_PIN,
D7_PIN);
// states of lcd
int state = 0;
// ------------------------------------------------------------
----
// define led pins
#define led1 4
#define led2 5
#define led3 6
#define led4 7
// ------------------------------------------------------------
----
// eeprom settings
#define device_addr 0b1010000
byte type;
byte time1, time2, time3, time4;
byte default_time = 10; // 10 seconds
```

```
int delay_time = 1000; // 1 second
// ----------------------------------------------------------
----
// functions
void update_led();
void update_screen();
void update_screen_time(byte t);
void keypadEvent(KeypadEvent key);
void eeprom_write(byte memory_addr, byte data);
byte eeprom_read(byte memory_addr);
void retrieve_var();
void save_var();

// ----------------------------------------------------------
----
void setup() {
  // setup lcd
  lcd.begin(16, 2);
  state = 0;
  update_screen();

  // setup keypad event listener
  keypad.addEventListener(keypadEvent);

  // setup led pins
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);

  // begin TWI with eeprom
  Wire.begin();
```

```
  // interrupt pin
  pinMode(INT_PIN, INPUT);
}

void loop() {
  // keypad pressed button
  char c = keypad.getKey();

  if(digitalRead(INT_PIN) == LOW){
    lcd.clear();
    lcd.print("saving...");
    save_var();
    state = 0;
    update_screen();
  }else if(state == 1){
    if(time1 <= 0)
      state = 2;
    else{
      time1 = time1 - 1;
      update_screen_time(time1);
    }
  }else if(state == 2){
    if(time2 <= 0)
      state = 3;
    else{
      time2 = time2 - 1;
      update_screen_time(time2);
    }
  }else if(state == 3){
    if(time3 <= 0)
      state = 4;
```

```
    else{
      time3 = time3 - 1;
      update_screen_time(time3);
    }
  }else if(state == 4){
    if(time4 <= 0){
      lcd.clear();
      lcd.print("finished");
      state = 7;
      update_led();
      delay(2*delay_time);
      state = 0;
      update_screen();
    }else{
      time4 = time4 - 1;
      update_screen_time(time4);
    }
  }
}

// ------------------------------------------------------------
----
void update_led(){
  if(state == 1){
    digitalWrite(led1, HIGH);
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
    digitalWrite(led4, LOW);
  }else if(state == 2){
    digitalWrite(led1, LOW);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, LOW);
```

```
    digitalWrite(led4, LOW);
  }else if(state == 3){
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, LOW);
  }else if(state == 4){
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
    digitalWrite(led4, HIGH);
  }else if(state == 7){ //finish
    digitalWrite(led1, HIGH);
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    digitalWrite(led4, HIGH);
  }else{
    digitalWrite(led1, LOW);
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
    digitalWrite(led4, LOW);
  }
}

void update_screen(){
  update_led();
  lcd.clear();

  // state=0: time_v_type & wash
  // state=9: time1 / state=10: time2 / state=11: time3 /
state=12: time4 / state=13: wash_type
```

```
  if(state == 0){
    lcd.print("settings");
    lcd.setCursor(14, 0);
    lcd.print("x");
    lcd.setCursor(0, 1);
    lcd.print("start");
    lcd.setCursor(14, 1);
    lcd.print("+");
  }else if(state == 9){
    lcd.print("time of type1");
    lcd.setCursor(0, 1);
  }else if(state == 10){
    lcd.print("time of type2");
    lcd.setCursor(0, 1);
  }else if(state == 11){
    lcd.print("time of type3");
    lcd.setCursor(0, 1);
  }else if(state == 12){
    lcd.print("time of type4");
    lcd.setCursor(0, 1);
  }else if(state == 13){
    lcd.print("wash type");
    lcd.setCursor(0, 1);
  }
}

void update_screen_time(byte t){
  update_led();
  lcd.clear();
  lcd.print("type "+String(state));
  lcd.setCursor(0, 1);
  lcd.print("time: "+String(t));
```

```
  delay(delay_time);
}

void keypadEvent(KeypadEvent key){
  if(keypad.getState() == PRESSED){
    if(state == 0){
      if(key == '*')
        state = 9;
      else if(key == '+'){
        lcd.clear();
        lcd.print("retrieving...");
        retrieve_var();
        lcd.clear();
      }
      update_screen();
    }else if(state == 9){
      time1 = (byte)(key - '0');
      lcd.print(String(time1));
      delay(delay_time);
      state = 10;
      update_screen();
    }else if(state == 10){
      time2 = (byte)(key - '0');
      lcd.print(String(time2));
      delay(delay_time);
      state = 11;
      update_screen();
    }else if(state == 11){
      time3 = (byte)(key - '0');
      lcd.print(String(time3));
      delay(delay_time);
      state = 12;
```

```
      update_screen();
    }else if(state == 12){
      time4 = (byte)(key - '0');
      lcd.print(String(time4));
      delay(delay_time);
      state = 13;
      update_screen();
    }else if(state == 13){
      if(((key >= '1') && (key <= '4'))){
        state = (key - '0');
        lcd.print(String(state));
        delay(delay_time);
        lcd.clear();
        lcd.print("saving...");
        save_var();
        state = 0;
        update_screen();
      }
    }
  }
}

// ------------------------------------------------------------
----
void eeprom_write(byte memory_addr, byte data){
  Wire.beginTransmission(device_addr);
  Wire.write(memory_addr);
  Wire.write(data);
  Wire.endTransmission();
}

byte eeprom_read(byte memory_addr){
```

```
  byte data = NULL;
  Wire.beginTransmission(device_addr);
  Wire.write(memory_addr);
  Wire.endTransmission();

  Wire.requestFrom(device_addr, 1);
  delay(5);
  if(Wire.available())
    data = Wire.read();
  return data;
}

// ------------------------------------------------------------
----
void retrieve_var(){
  type = eeprom_read(0);
  delay(delay_time);
  time1 = eeprom_read(10);
  delay(delay_time);
  time2 = eeprom_read(20);
  delay(delay_time);
  time3 = eeprom_read(30);
  delay(delay_time);
  time4 = eeprom_read(40);

  // check if times or type if zero
  if(type == 0)
    type = 1;

  // set state
  state = (int)type;
}
```

```
void save_var(){
  type = (byte)state;
  eeprom_write(0, type);
  delay(delay_time);
  eeprom_write(10, time1);
  delay(delay_time);
  eeprom_write(20, time2);
  delay(delay_time);
  eeprom_write(30, time3);
  delay(delay_time);
  eeprom_write(40, time4);
}
```