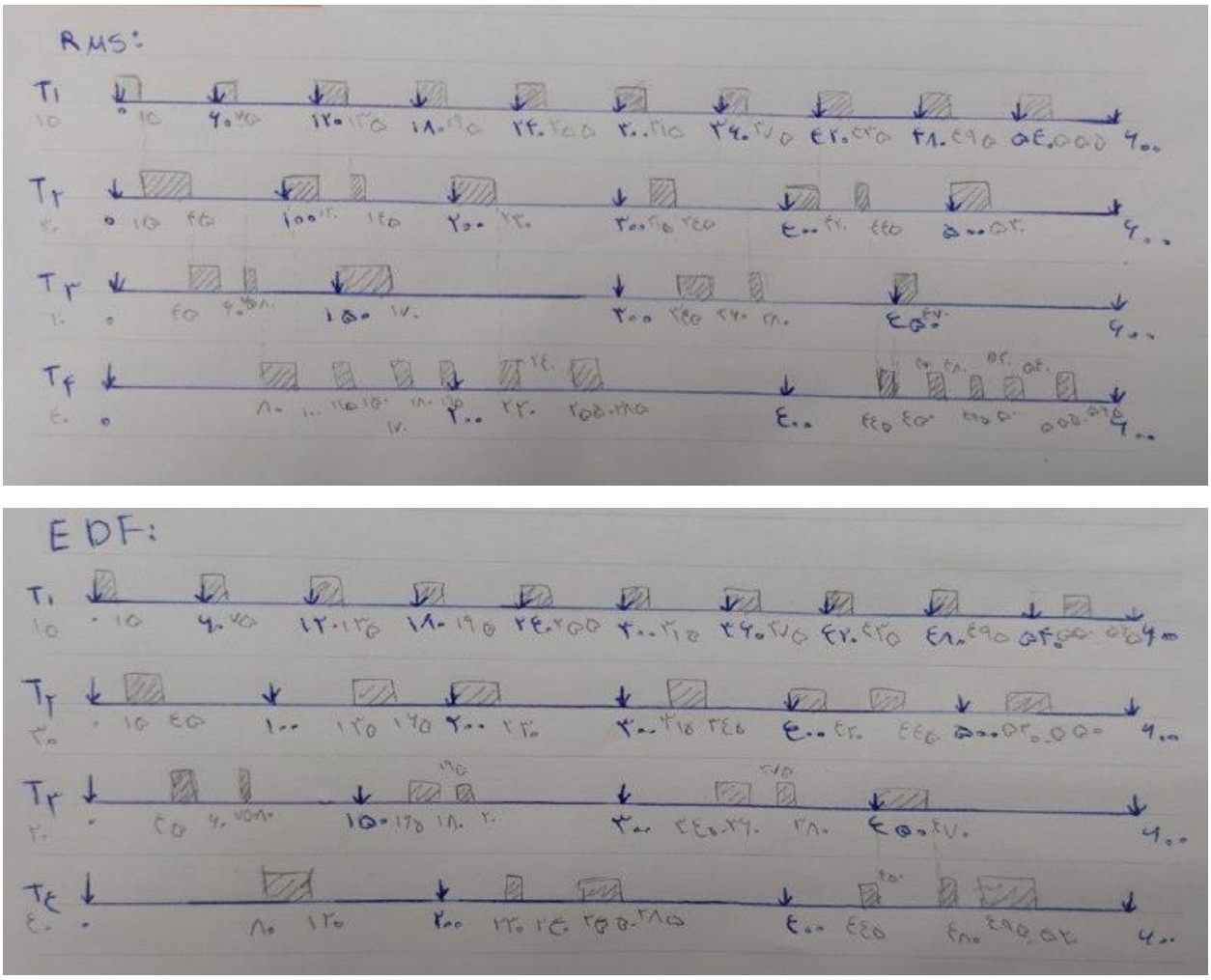
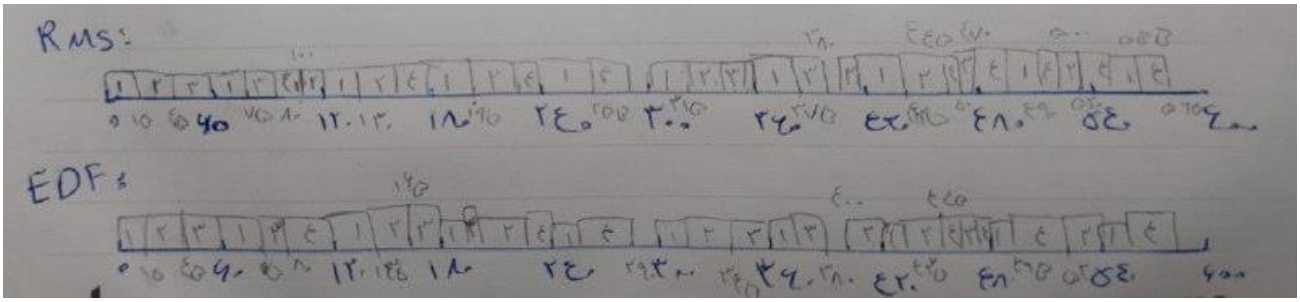
پاسخ تمرین سری ششم مبانی سیستم‌های نهفته و بیدرنگ

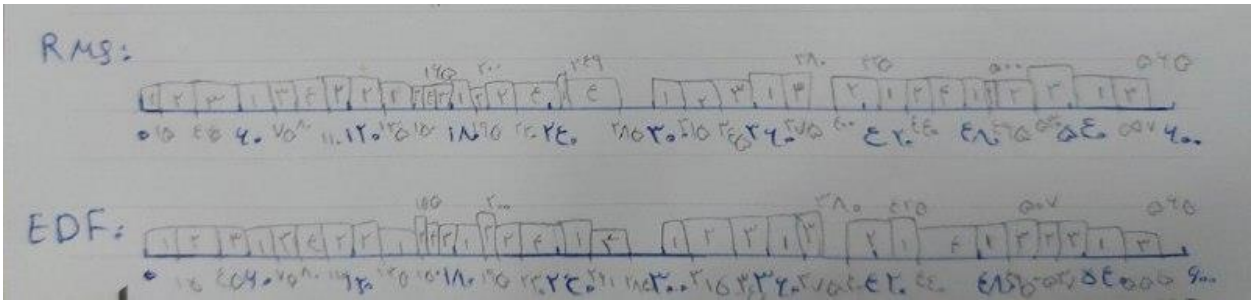
۱. الف) هر دو الگوریتم می‌توانند بدون دیرکرد کار کنند. به صورت زیر:



ب)



ج)



۲. الف) بعد از اضافه کردن کتابخانه FreeRTOS به پروژه، تعدادی مثال نیز درون این کتابخانه مشاهده می‌کنیم (در آدرس ./.pio/libdeps/uno/FreeRTOS/examples). در این بخش، مثال Blink_AnalogRead بررسی می‌شود.

در ابتدای کد، کتابخانه FreeRTOS، include شده است تا از توابع موجود در این کتابخانه استفاده کنیم.

سپس prototype دو تابع task با نام‌های TaskBlink و TaskAnalogRead مشاهده می‌کنیم. تابع TaskBlink برای چشمک زدن چراغ و تابع TaskAnalogRead برای خواندن مقدار آنالوگ استفاده می‌شود.

در تابع setup، ابتدا baud rate مقداردهی شده است و تا زمانی که اتصالی برقرار نشده باشد، درون یک حلقه خالی قرار می‌گیریم. پس از اتصال، دو task یکی از نوع blink و دیگری از نوع analogRead توسط تابع xTaskCreate() ساخته شده است. درون هر یک از توابع xTaskCreate، پارامترهای موردنیاز از جمله پوینتر به تابع، نام تابع، سایز استک و ... مقداردهی شده است. در آخر نیز بیان شده است که تابع scheduler به صورت خودکار فراخوانی خواهد شد.

```
// the setup function runs once when you press reset or power the board
void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB, on LEONARDO, MICRO, YUN, and other 32u4 based boards.
  }

  // Now set up two tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  "Blink"   // A name just for humans
    ,  128  // This stack size can be checked & adjusted by reading the Stack Highwater
    ,  NULL
    ,  2  // Priority, with 3 (configMAX_PRIORITIES - 1) being the highest, and 0 being the lowest.
    ,  NULL );

  xTaskCreate(
    TaskAnalogRead
    ,  "AnalogRead"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

  // Now the task scheduler, which takes over control of scheduling individual tasks, is automatically started.
}
```

شکل ۱: تابع setup کد Blink_AnalogRead

درون تابع loop چیزی نوشته نمی‌شود و تمامی کارها درون توابع task انجام می‌شود.

حال به سراغ تابع ()TaskBlink می‌رویم:

در ابتدا، مقداری توضیح در مورد کار این تابع داده شده است (که قرار است یک on-board LED را روشن و خاموش کند). سپس پین متناظر را ست کرده و درون حلقه for، این پین را high می‌کند؛ سپس به مدت یک ثانیه تسک را block می‌کند، سپس بعد از اینکه scheduler زمان اجرا را به آن داد، پین را low می‌کند تا LED خاموش گردد؛ سپس دوباره تسک را block می‌کند. این روند همینطور ادامه خواهد داشت.

```
void TaskBlink(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

/*...

  // initialize digital LED_BUILTIN on pin 13 as an output.
  pinMode(LED_BUILTIN, OUTPUT);

  for (;;) // A Task shall never return or exit.
  {
    digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // wait for one second
  }
}
```

شکل ۲: تابع TaskBlink

در ادامه، تابع TaskAnalogRead را داریم. در این تابع نیز ابتدا توضیحی در مورد کارکرد این تابع داده شده است (که مقدار پین آنالوگ را خوانده، سپس این مقدار را چاپ می‌کند). درون حلقه for مقدار پین A0 خوانده و پرینت می‌شود. سپس به وسیله تابع vTaskDelay، به مدت ۱۵ میلی ثانیه برنامه block می‌شود.

```
void TaskAnalogRead(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

/*...

  for (;;)
  {
    // read the input on analog pin 0:
    int sensorValue = analogRead(A0);
    // print out the value you read:
    Serial.println(sensorValue);
    vTaskDelay(1);  // one tick delay (15ms) in between reads for stability
  }
}
```

شکل ۳: تابع TaskAnalogRead

ب) بخش اول: طبق گفته سوال، سه وظیفه تعریف کرده‌ایم: یکی برای خواندن مقدار سنسور photoresistor؛ یکی برای خواندن مقدار flex sensor و دیگری برای مشخص کردن جهت حرکت موتور. دو تسک دیگر هم برای دیباگ کردن مقادیر خوانده شده توسط photoresistor و دیگری برای دیباگ مقدار خوانده شده flex sensor ساخته شده است.

```
void ReadPhotoresistor( void *pvParameters );
void ReadFlexSensor( void *pvParameters );
void MotorDirection( void *pvParameters );
void DebugPhotoresistor( void *pvParameters );
void DebugFlexSensor( void *pvParameters );
```

شکل ٤: prototype تسک‌های تعریف شده

برنامه ما به دو روش مختلف نوشته شده است:

روش اول:

حال در تابع setup()، ابتدا پین‌های ورودی و خروجی را ست می‌کنیم و سپس تسک‌ها را می‌سازیم.

```
void setup() {

  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);

  Serial.begin(9600);
  while (!Serial) {
    ;
  }

  xTaskCreate(ReadPhotoresistor, "ReadPhotoresistor", 128, NULL, 1, NULL);
  xTaskCreate(ReadFlexSensor, "ReadFlexSensor", 128, NULL, 1, NULL);
  xTaskCreate(MotorDirection, "MotorDirection", 128, NULL, 1, NULL);
  xTaskCreate(DebugPhotoresistor, "DebugPhotoresistor", 128, NULL, 1, NULL);
  xTaskCreate(DebugFlexSensor, "DebugFlexSensor", 128, NULL, 1, NULL);

}
```

شکل ۵: تابع setup

همه تسک‌ها اولویت یکسان دارند و از طریق taskYIELD() اجرا را به سایر تسک‌ها نیز می‌دهیم.

تابع loop() هم که خالی خواهد ماند.

در تسک ReadPhotoresistor، درون حلقه for، مقدار روی پین A0 که به فتورزیستور وصل است را می‌خوانیم و بیشترین مقدار و کمترین مقدار ورودی را به بازه 10 تا 100 مپ می‌کنیم. این مقدار را روی متغیر گلوبال light می‌ریزیم. سپس با تابع taskYIELD()، اجرا را به دست scheduler می‌دهیم تا برنامه بعدی درون scheduler را اجرا کند.

```
void ReadPhotoresistor(void *pvParameters){
  (void) pvParameters;

  for(;;){
    light = map(analogRead(A0), 10, 975, 10, 100);
    taskYIELD();
  }
}
```

شکل ۶: تابع ReadPhotoresistor

در تسک ReadFlexSensor نیز مشابه ReadPhotoresistor، پین A1 را می‌خوانیم و بازه مقدار خوانده شده را به 0 تا 180 مپ می‌کنیم. سپس taskYIELD() را فراخوانی می‌کنیم.

```
void ReadFlexSensor(void *pvParameters){
  (void) pvParameters;

  for(;;){
    degree = map(analogRead(A1), 0, 1019, 180, 0);
    taskYIELD();
  }
}
```

شکل ۷: تابع ReadFlexSensor

```
void MotorDirection(void *pvParameters){
  (void) pvParameters;

  for(;;){

    if(degree > 10 && degree < 80){

      if(light < 45){

        digitalWrite(2, LOW);
        digitalWrite(3, HIGH);

      }else if(light > 56){

        digitalWrite(2, HIGH);
        digitalWrite(3, LOW);

      }else{
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
      }


    }else{

      digitalWrite(2, LOW);
      digitalWrite(3, LOW);
    }

  taskYIELD();

  }
}
```

در تسک MotorDirection، درون حلقه for شروط لازم برای مشخص کردن جهت چرخش موتور را تعیین می‌کنیم. سپس با توجه به شروط، پین‌های 2 و 3 را High یا Low می‌کنیم. در آخر نیز taskYIELD() را فراخوانی می‌کنیم.

شکل ۸: تابع MotorDirection

همانطور که گفته شد، دو تسک DebugPhotoresistor و DebugFlexSensor داریم که زمانی که مقدار متغیر debug آن‌ها برابر یک باشد، مقدار پین‌ها را روی نمایشگر نشان می‌دهد.

```
void DebugPhotoresistor(void *pvParameters){
  (void) pvParameters;

  for(;;){
    if (f_debug == 1)
      Serial.println(analogRead(A0));
    taskYIELD();
  }
}


void DebugFlexSensor(void *pvParameters){
  (void) pvParameters;


  for(;;){
    if (f_debug == 1)
      Serial.println(analogRead(A1));
    taskYIELD();
  }

}
```

شکل ۹: توابع DebugPhotoresistor و DebugFlexSensor

نکته حائز اهمیت در این کد این است که همه تسک‌ها اولیت 1 دارند و از طریق taskYIELD() دائما بین تسک‌ها سوییچ می-کنیم.

روش دوم:

در این روش، هدف این است که مقادیر خوانده شده توسط سنسورها را درون یک آرایه قرار بدهیم و زمانی که مقداری روی این آرایه قرار گرفت، تسک MotorDirection که اولویت بالاتری دارد، اجرا می‌شود.

در ابتدای برنامه، یک آرایه دو بعدی داریم که درون آن مقادیر خوانده شده توسط سنسورها درون آن قرار می‌گیرد. دو متغیر نیز برای دیباگ مقادیر هرکدام از پین‌ها قرار داده‌ایم. در ادامه، سه handle به صف‌هایی که قرار است ایجاد کنیم (یکی برای مقادیر سنسورها و دوتای دیگر برای دیباگ این مقادیر)، تعریف می‌کنیم.

```
int sensorsArray[2] = {0, 0};

int l_debug_flag = 0;
int f_debug_flag = 0;


void ReadPhotoresistor( void *pvParameters );
void ReadFlexSensor( void *pvParameters );
void MotorDirection( void *pvParameters );
void DebugPhotoresistor( void *pvParameters );
void DebugFlexSensor( void *pvParameters );

QueueHandle_t sensors_q;

QueueHandle_t debug_degree_qh;
QueueHandle_t debug_light_qh;
```

شکل ۱۰: قطعه کد توضیح داده شده

حال در تابع setup()، سه صف مجزا (یکی مقادیر و دوتای دیگر برای دیباگ) می‌سازیم. در صورتی که ساخت این صف‌ها موفقیت‌آمیز بود، همانند روش قبل تسک‌های خود را می‌سازیم.

```
void setup() {

  pinMode(A0, INPUT);
  pinMode(A1, INPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);

  sensors_q = xQueueCreate(10, //Queue length
                           sizeof(int)); //Queue item size

  debug_degree_qh = xQueueCreate(10, //Queue length
                             sizeof(int)); //Queue item size

  debug_light_qh = xQueueCreate(10, //Queue length
                             sizeof(int)); //Queue item size


  if (sensors_q != NULL && debug_degree_qh != NULL && debug_light_qh != NULL){

    Serial.begin(9600);
    while (!Serial) {
      ;
    }

    xTaskCreate(ReadPhotoresistor, "ReadPhotoresistor", 128, NULL, 1, NULL);
    xTaskCreate(ReadFlexSensor, "ReadFlexSensor", 128, NULL, 1, NULL);
    xTaskCreate(MotorDirection, "MotorDirection", 128, NULL, 2, NULL);
    xTaskCreate(DebugPhotoresistor, "DebugPhotoresistor", 128, NULL, 2, NULL);
    xTaskCreate(DebugFlexSensor, "DebugFlexSensor", 128, NULL, 2, NULL);

  }
}
```

شکل ۱۱: تابع setup() آپدیت شده

درون تابع ReadPhotoresistor() مشابه روش قبل پین ورودی مربوط به فتورزیستور را می‌خوانیم. سپس این مقدار را درون آرایه می‌ریزیم و با استفاده از تابع xQueueSend()، آن را روی صف قرار می‌دهیم. در صورتی که حالت دیباگ روشن باشد، این مقدار را روی صف مربوط به دیباگ شدت نور نیز قرار می‌دهیم. در آخر نیز taskYIELD() را فراخوانی می‌کنیم تا نوبت به اجرای تسک ReadFlexSensor() نیز برسد.

```
void ReadPhotoresistor(void *pvParameters){
  (void) pvParameters;

  for(;;){
    int readA0 = map(analogRead(A0), 10, 975, 10, 100);
    sensorsArray[0] = readA0;

    if (l_debug_flag == 1)
      xQueueSend(debug_light_qh, &readA0, portMAX_DELAY);

    xQueueSend(sensors_q, &sensorsArray, portMAX_DELAY);

    taskYIELD();
  }
}
```

شکل ۱۲: تابع ReadPhotoresistor آپدیت شده

تابع ReadFlexSensor() نیز مشابه تابع قبلی، مقدار پین متصل به flex sensor را می‌خواند و پس از مپ کردن مقدار آن، درون آرایه نوشته و سپس آن را روی صف مربوطه قرار می‌دهد. مشابه تسک قبلی، اینجا نیز حالتی برای دیباگ داریم. در آخر نیز taskYIELD() فراخوانی می‌شود.

```
void ReadFlexSensor(void *pvParameters){
  (void) pvParameters;

  for(;;){
    int readA1 = map(analogRead(A1), 0, 1019, 180, 0);
    sensorsArray[1] = readA1;

    if (f_debug_flag == 1)
      xQueueSend(debug_degree_qh, &readA1, portMAX_DELAY);

    xQueueSend(sensors_q, &sensorsArray, portMAX_DELAY);

    taskYIELD();
  }
}
```

شکل ۱۳: تابع ReadFlexSensor آپدیت شده

منطق تسک MotorDirection() مشابه روش قبل است؛ با این تفاوت که در ابتدای حلقه بررسی می‌کنیم که اگر درون صف sensors_q مقداری قرار گرفت، وارد این بخش می‌شویم. در نتیجه، تا زمانی که دو تسک قبلی مقداری را نخوانند، این تسک block خواهد شد.

```
void MotorDirection(void *pvParameters){
  (void) pvParameters;

  for(;;){

    if (xQueueReceive(sensors_q, &sensorsArray, portMAX_DELAY) == pdPASS){

      int light  = sensorsArray[0];
      int degree = sensorsArray[1];

      if(degree > 10 && degree < 80){

        if(light < 45){

          digitalWrite(2, LOW);
          digitalWrite(3, HIGH);

        }else if(light > 56){

          digitalWrite(2, HIGH);
          digitalWrite(3, LOW);

        }else{
          digitalWrite(2, LOW);
          digitalWrite(3, LOW);
        }


      }else{

        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
      }
    }

  }
}
```

شکل ۱۴: تابع MotorDirection آپدیت شده

در دو تسک debug نیز زمانی که مقداری روی صف مربوط به هرکدام نوشته شود، تسک از حالت blocked به حالت ready برمی‌گردد و مقدار آن پین روی نمایشگر نشان داده خواهد شد.

```
void DebugPhotoresistor(void *pvParameters){
  (void) pvParameters;

  int A0Read = 0;

  for(;;){
    if (xQueueReceive(debug_light_qh, &A0Read, portMAX_DELAY) == pdPASS){
      Serial.println(A0Read);
    }
  }
}


void DebugFlexSensor(void *pvParameters){
  (void) pvParameters;

  int A1Read = 0;

  for(;;){

    if (xQueueReceive(debug_degree_qh, &A1Read, portMAX_DELAY) == pdPASS){
      Serial.println(A1Read);
    }

  }

}
```

شکل ۱۵: توابع debug آپدیت شده

بخش دوم: در هر دو روش این موضوع را بررسی می‌کنیم:

روش اول:

در این حالت، هر 5 تسک موجود (سه‌تا تسک اصلی و دو تسک دیباگ) تحریک شده با زمان هستند و پس از یک مقداری زمان، تسک‌ها مجددا شروع به اجرا می‌کنند. هر وظیفه با استفاده از تابع taskYIELD() به صف ready برمی‌گردد و منتظر اجازه scheduler برای اجرای مجدد می‌گردد. پس در این روش از ISR به طور کل استفاده‌ای نشده است.

روش دوم:

در این حالت، تسک‌های ReadPhotoresistor و ReadFlexSensor تحریک شده با زمان هستند. اما تسک‌های MotorDirection و DebugPhotoresistor و DebugFlexSensor تحریک شده با رویداد هستند و زمانی اجرا می‌شوند که درون صف مربوط به هرکدام از این تسک‌ها، مقداری قرار بگیرد. در این روش، دو تسک ReadPhotoresistor و ReadFlexSensor پس از فراخوانی تابع taskYIELD() به صف ready برمی‌گردند و منتظر اجازه scheduler برای

اجرای مجدد می‌شوند. اما دو تسک دیباگ و تسک MotorDirection از طریق صف‌ها (در صورتی که خالی باشند)، بلاک می-شوند. پس در نتیجه جنس ISR آن‌ها از نوع interrupt source است.

بخش سوم: در هر دو روش این موضوع را بررسی می‌کنیم:

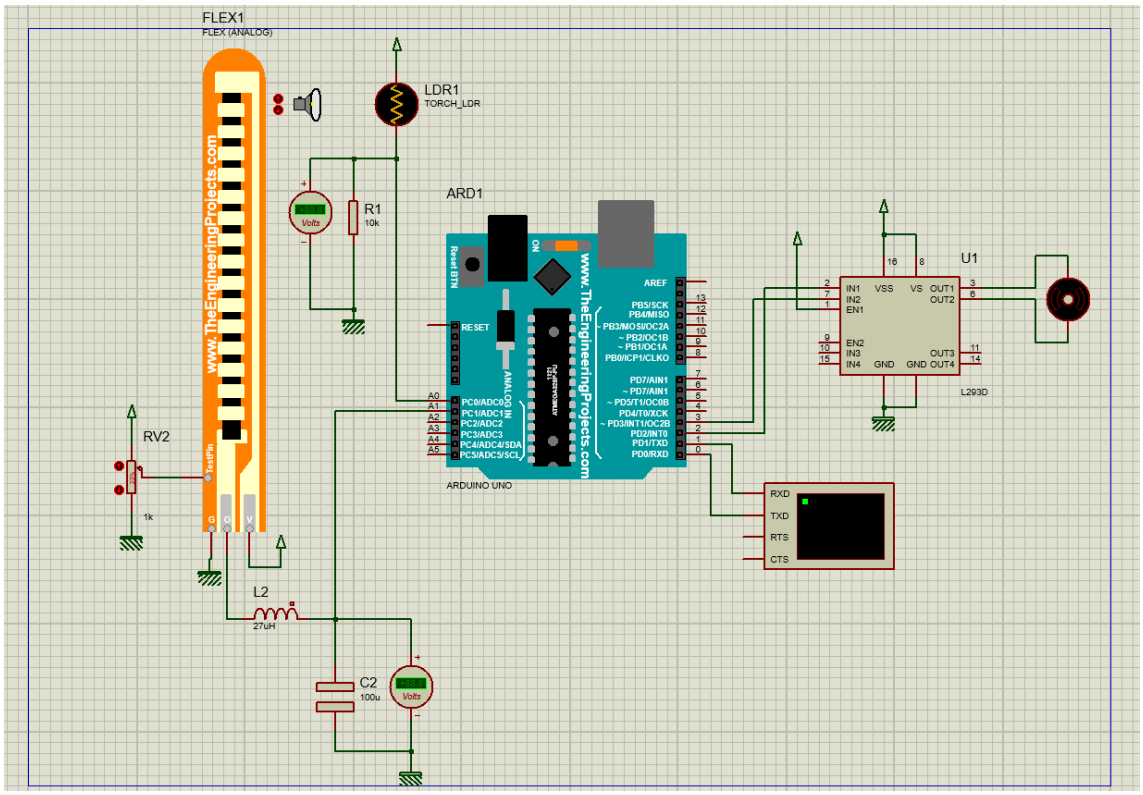
روش اول:

بین تسک‌ها ارتباطی وجود ندارد و از طریق دو متغیر گلوبال light و degree با یکدیگر ارتباط برقرار می‌کنند.

روش دوم:

ارتباط بین تسک‌ها از طریق یک سری صف خواهد بود؛ بدین گونه که هر کدام از توابع خواندن مقدار سنسورها، درون این صف‌ها مقادیری قرار می‌دهند و سپس تسک‌های دیباگ و MotorDirection این مقادیر را از روی صف‌های متناظرشان می‌خوانند. در صورت عدم وجود مقداری، این تسک‌ها وارد حالت blocked می‌شوند. همچنین ISR ما از جنس نرم‌افزاری بوده و در صورت عدم وجود مقدار روی صف، خود سیستم‌عامل subroutineای را اجرا می‌کند که باعث بلاک شدن تسک می‌شوند تا یک event جدید (مقدار جدید روی صف) رخ بدهد.

ج) با استفاده از مثال موجود در لینک‌های داده شده، flex sensor را روی محیط طراحی می‌کنیم و پین خروجی آن را به پین A1 متصل می‌کنیم. همچنین فتورزیستور را به پین A0 برد متصل می‌کنیم. از طرف دیگر پین‌های خروجی را به h-bridge متصل می‌کنیم و خروجی h-bridge را به dc motor وصل می‌کنیم. برای دیباگ مقادیر نیز یک virtual terminal قرار داده شده است. سپس فایل hex. جنریت شده از کدمان را درون برد آپلود می‌کنیم و شبیه‌سازی را انجام می‌دهیم.



شکل ۱۶: مدار ترسیم شده در محیط proteus