گزارش تمرین ۶

فهرست

بخش ۱

	C_i	T_i
τ_1	2	10
τ_2	5	20
τ_3	8	30

تحلیل امکان زمان‌بندی از نظر بهره‌گیری

بهره‌گیری کلی سیستم U برابر است با:

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = \frac{2}{10} + \frac{5}{20} + \frac{8}{30} = 0.716$$

برای $n = 3$، کران بهره‌گیری RMS طبق فرمول زیر است:

$$U_{RMS} = \left(\sqrt{2}^{n} - 1\right).n = \left(\sqrt{2}^{3} - 1\right).3 = 0.779$$

از آنجا که $U = 0.716 < 0.779$ می‌باشد، این مجموعه با الگوریتم RMS قابل زمان‌بندی هستند.

الگوریتم EDF در صورتی که بهره‌گیری کلی کمتر از ۱ باشد، قادر به زمان‌بندی بدون از دست دادن ددلاین است.

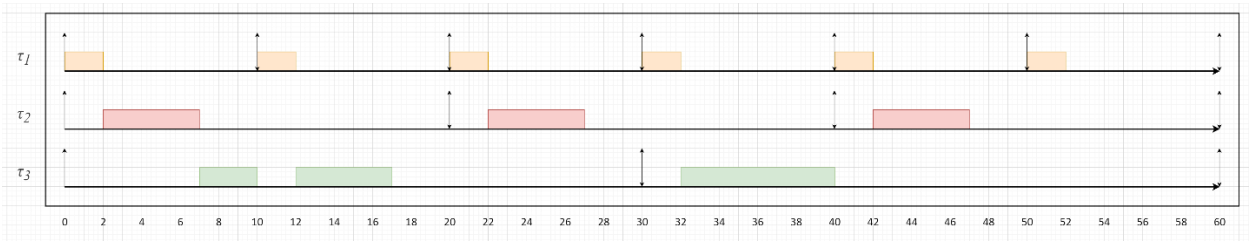
از آن‌جا که $U = 0.716 < 1$ می‌باشد، EDF نیز زمان‌بندی را بدون مشکل انجام می‌دهد.

زمان‌بندی با الگوریتم RMS

در RMS، اولویت‌ها بر اساس دوره تناوب تعیین می‌شوند، تسک با دوره کوتاه‌تر اولویت بالاتری دارد.

ترتیب اولویت‌ها: $\tau_1 > \tau_2 > \tau_3$ (زیرا $T_1 < T_2 < T_3$)

در $t = 0$، $\tau 1$، $\tau 2$، $\tau 3$ همزمان رها می‌شوند.

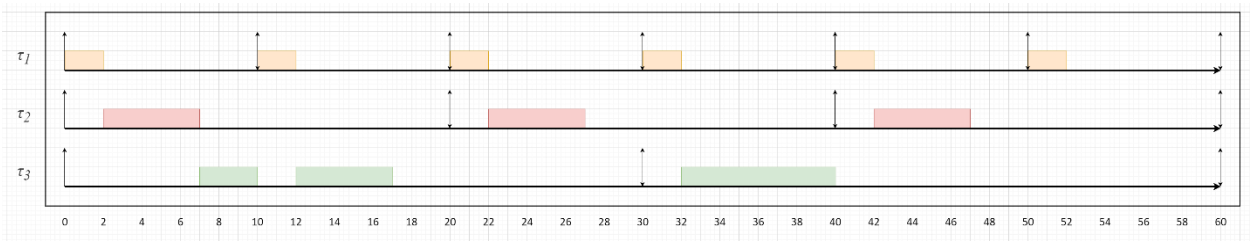


زمان‌بندی با الگوریتم EDF

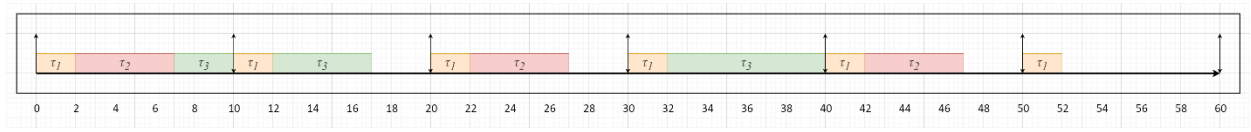
در EDF، اولویت‌ها بر اساس ددلاین تعیین می‌شوند، تسک با نزدیک‌ترین ددلاین اولویت بالاتری دارد.

در $t = 0$، $\tau1$، $\tau2$، $\tau3$ همزمان رها می‌شوند.

پس ترتیب اولویت‌ها: $\tau_1 > \tau_2 > \tau_3$ (زیرا $T_1 < T_2 < T_3$)



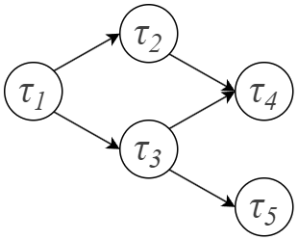
زمان‌بندی در یک خط



بخش ۲

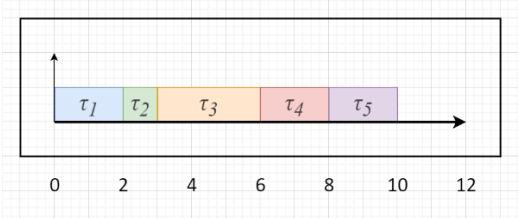
	C_i	r_i	d_i
τ_1	2	0	6
τ_2	1	1	7
τ_3	3	3	8
τ_4	2	4	10
τ_5	2	5	11

گراف وابستگی



زمان‌بندی با الگوریتم EDF

EDF در هر لحظه تسکی را انتخاب می‌کند که ددلاین آن زودتر از بقیه است (مشروط بر اینکه تسک رسیده باشد و محدودیت‌های وابستگی رعایت شود).

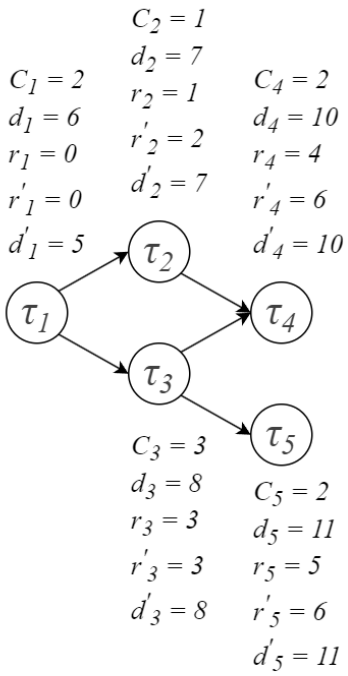


زمان‌بندی با الگوریتم EDF*

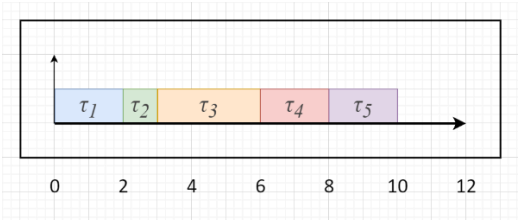
قبل از اجرای الگوریتم EDF*، نیاز است زمان‌های رسیدن و ددلاین‌ها را اصلاح کنیم.

$$r_j' = \max(r_j, r_i' + C_i)$$

$$d_i' = \min(d_i, d_j' - C_j)$$



حال، مانند الگوریتم EDF در هر لحظه تسکی که d' آن نزدیک‌تر است را انتخاب می‌کنیم.



بخش ۳

تسک‌های تعریف شده

برنامه پنکه چرخان شامل بخش‌های خواندن ورودی‌ها از سنسور دما و دکمه سوییچ، اعمال خروجی به موتور DC و Servo است. دو بخش منطق کنترل‌کننده‌ی موتور DC و Servo قابل جداسازی هستند. هر یک از این بخش‌ها شامل یک ورودی و یک خروجی هستند که هر بخش هم قابل جداسازی به دو بخش جدا از هم است. در نتیجه برای این قسمت ۴ تسک در نظر گرفته شده است.

یک بخش از منطق کد وابسته به کدهای ماشین حالت تولید شده توسط embedded coder متلب است. این بخش در فاصله زمانی‌های مشخص باید یک step اجرا شود. در نتیجه این بخش هم یک تسک در نظر گرفته شده است.

ماشین حالت

ماشین حالت طراحی شده در متلب باید به صورت متناوب اجرا شود. مقدار Step زمانی در متغیری به اسم `ex5_StateChart1_STEP_SIZE` توسط embedded coder ذخیره شده است. با استفاده از این متغیر در انتهای حلقه یک تاخیر اعمال می‌شود.

در هر بار اجرای این حلقه، ماشین حالت با دریافت ورودی‌ها یک بار اجرا می‌شود و خروجی‌های جدید را تولید می‌کند. ماشین حالت دو خروجی جهت چرخش موتور سروو و سرعت چرخش موتور DC را تعیین می‌کند. برای آن که این مقادیر در دسترس یک تسک دیگر قرار گیرد، از صف استفاده شده است. به کمک دو صف می‌توان زاویه سروو و سرعت چرخش موتور DC را به دو تسکی که می‌خواهند این مقادیر را روی پین خروجی بنویسند، منتقل کرد.

```
void taskStep(void *p) {
  int angle = 0;

  for (;;)
  {
    ex5_StateChart1_step();

    int servo_rotate = ex5_StateChart1_Y.servo_rotate;
    if (servo_rotate != 0)
    {
      angle += servo_rotate * servo_angle_change;
    }

    xQueueSend(dcSpeedQueue, &ex5_StateChart1_Y.dc_speed, portMAX_DELAY);
    xQueueSend(servoAngleQueue, &angle, portMAX_DELAY);
```

```
    vTaskDelay(ex5_StateChart1_STEP_SIZE / portTICK_PERIOD_MS);
  }
}
```

موتور DC

در قسمت موتور DC دو تسک taskTemperature و taskDcMotor تعریف شده است. تسک اول وظیفه خواندن مقادیر آنالوگ از سنسور دماسنج را دارد و تسک دوم وظیفه تعیین سرعت چرخش موتور را به عهده دارد.

تسک دماسنج یک مقدار آنالوگ را از پین متصل به سنسور می‌خواند و یکی از متغیرهای ورودی ماشین حالت را مقداردهی می‌کند. این تسک یک تسک متناوب با فاصله زمانی ۱ ثانیه است در نتیجه در پایان حلقه یک تاخیر زمانی ۱ ثانیه قرار داده شده است.

```
void taskTemperature(void *p) {
  for (;;)
  {
    ex5_StateChart1_U.temp_adc = analogRead(temp_pin);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}
```

در تسک موتور، سرعت جدید چرخش به موتور اعمال می‌شود. سرعت جدید چرخش در صف dcSpeedQueue قرار می‌گیرد و در صورت موجود بودن یک مقدار در این صف، شرط if اجرا و سرعت جدید روی پین نوشته می‌شود.

```
void taskDcMotor(void *p) {
  int dc_speed = 0;
  for (;;)
  {
    if (xQueueReceive(dcSpeedQueue, &dc_speed, portMAX_DELAY) == pdPASS) {
      analogWrite(dc_ENA_pin, dc_speed);
    }
  }
}
```

موتور Servo

در قسمت موتور Servo دو تسک taskSwitchButton و taskServo تعریف شده است. تسک اول وظیفه خواندن مقدار دیجیتال صفر یا یک از سوییچ را دارد و تسک دوم وظیفه تعیین زاویه قرارگیری موتور را به عهده دارد.

تسک دکمه سوییچ یک مقدار دیجیتال را از پین متصل به سوییچ می‌خواند و یکی از متغیرهای ورودی ماشین حالت را مقداردهی می‌کند. این تسک یک تسک متناوب با فاصله زمانی ۰.۱ ثانیه است در نتیجه در پایان حلقه یک تاخیر زمانی ۰.۱ ثانیه قرار داده شده است.

```
void taskSwitchButton(void *p) {
  for (;;)
  {
    ex5_StateChart1_U.switch_on = digitalRead(switch_pin);
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}
```

در تسک موتور، زاویه جدید به موتور اعمال می‌شود. زاویه جدید چرخش در صف servoAngleQueue قرار می‌گیرد و در صورت موجود بودن یک مقدار در این صف، شرط if اجرا و زاویه جدید به کمک کتابخانه Servo به موتور اعمال می‌شود.

```
void taskServo(void *p) {
  int angle = 0;
  for (;;)
  {
    if (xQueueReceive(servoAngleQueue, &angle, portMAX_DELAY) == pdPASS) {
      servo.write(angle);
    }
  }
}
```

اولویت تسک‌ها

در مجموع ۵ تسک تعریف شده است. ۲ تسک برای خواندن ورودی، ۲ تسک برای نوشتن روی خروجی و ۱ تسک برای اجرای ماشین حالت.

دو تسکی که از ورودی یک مقدار را می‌خوانند با بالاترین اولویت با اولویت ۲ تعیین شده‌اند و دو تسکی که روی خروجی یک مقدار را می‌نویسند با کمترین اولویت با اولویت صفر تعیین شده‌اند. این به این خاطر است که نوشتن یک مقدار جدید روی خروجی، مثلا سرعت چرخش موتور DC، وابسته به مقدار دما باشد. پس ابتدا باید یک ورودی خوانده شود و پس از آن یک خروجی. تسک اجرا شدن ماشین حالت با اولویت میانی یعنی اولویت ۱ تعیین شده است.

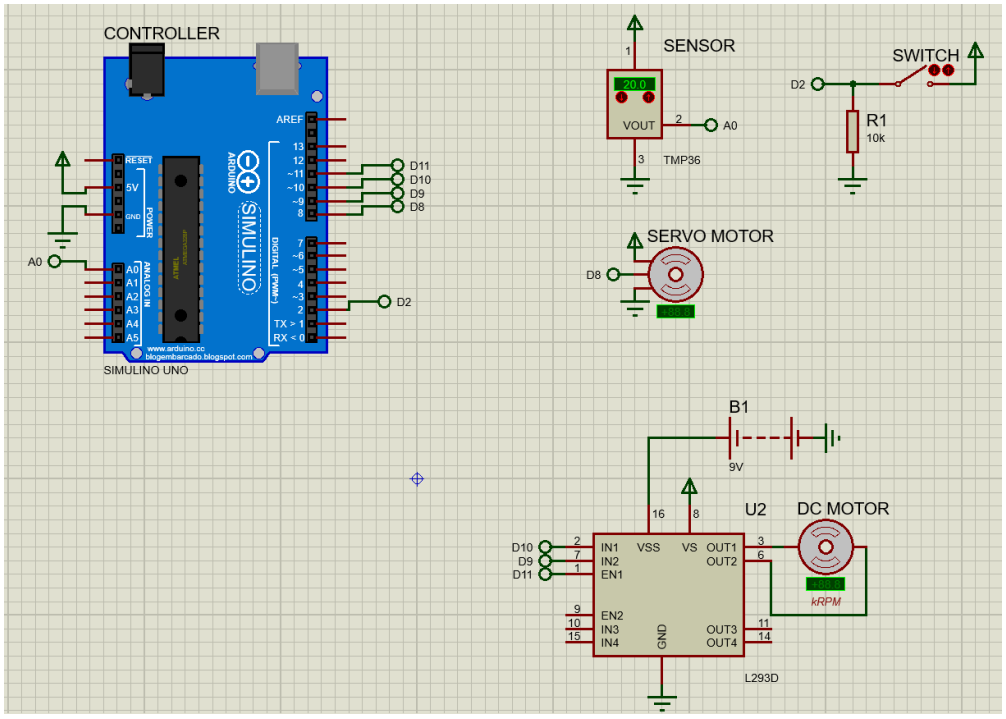
آماده‌سازی شبیه‌سازی

پس از نصب کتابخانه‌ها به کمک PlatformIO و توسعه کدها، نوبت به کامپایل می‌رسد. به کمک امکاناتی که PlatformIO در اختیار گذاشته است و تعیین نوع و مدل بورد مقصد، کدهای توسعه یافته را Build می‌کنیم. با Build کردن کدها، یک فایل با

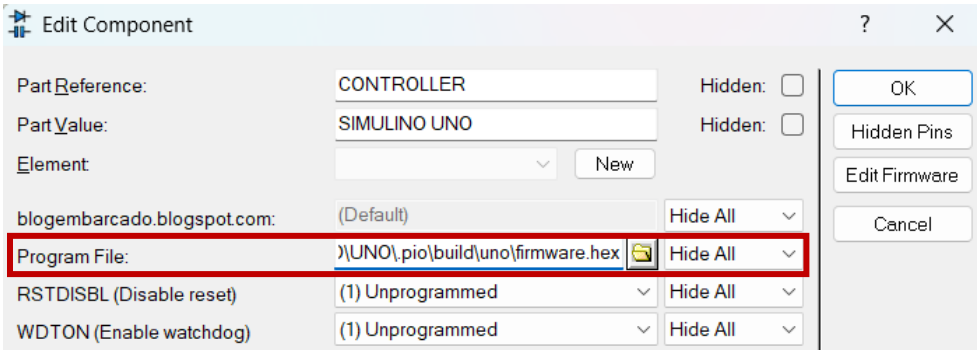
فرمت hex تولید می‌شود. این فایل شامل تمام کدها و کتابخانه‌ها به صورت تجمیع شده در یک فایل است. سپس این فایل hex را برای اجرا به پروتئوس منتقل می‌کنیم.

شبیه سازی در پروتئوس

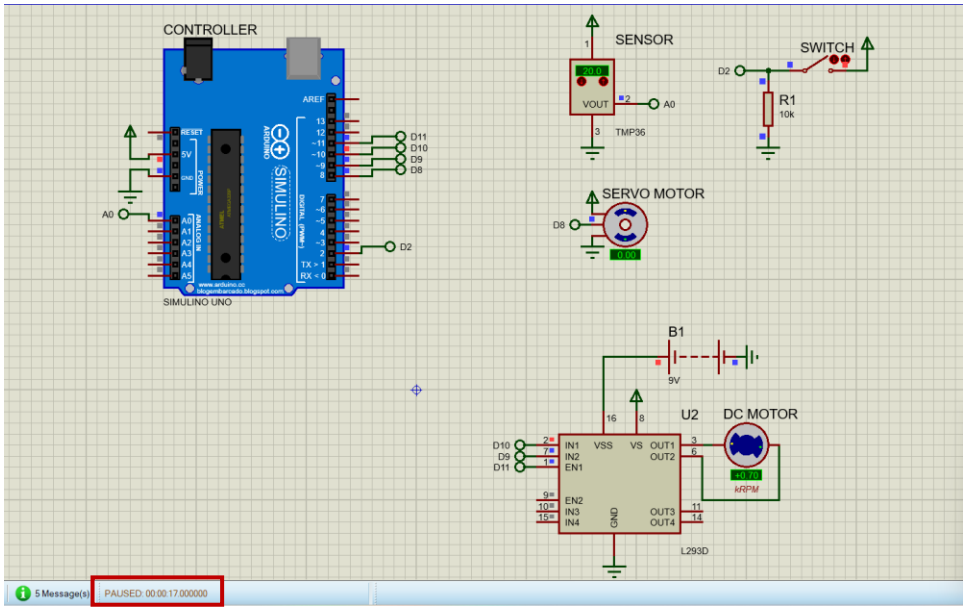
پس از اضافه کردن کتابخانه کنترلرهای Arduino به نرم‌افزار Proteus، کافی‌ست قطعات را در کنار یکدیگر قرار دهیم و متصل کنیم.



حال مسیر فایل Hex تولید شده از پروژه PlatformIO را به منظور اجرای برنامه در Arduino قرار می‌دهیم.



پس از اجرای شبیه‌سازی، در ثانیه ۱۷ سروو موتور یک دور چرخیده و برگشته است.



با توجه به وضعیت خاموش سوییچ، سروو پس از ۱۰ ثانیه در ثانیه ۲۷ دوباره شروع به گردش می‌کند.