بسم الله الرحمن الرحیم

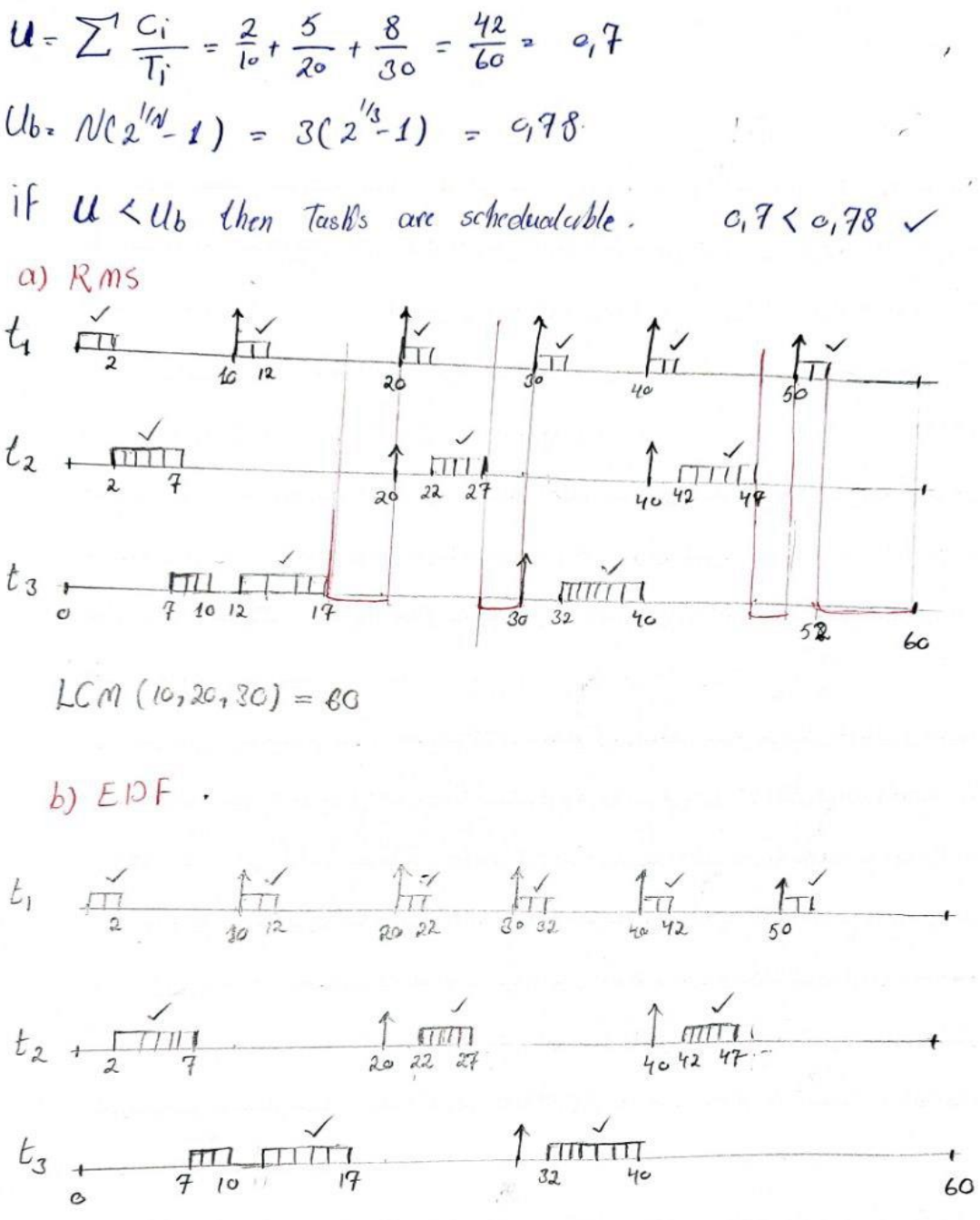
گزارش تمرین 6

درس مبانی سیستم نهفته و بیدرنگ

جواب سوال 1)

1- امکان زمانبندی مجموعه وظایف مقابل را هم بصورت تحلیلی با به کارگیری کران های بهرهگیری مرتبط و هم با رسم زمانبندی برای یک فراتناوب بررسی کنید.

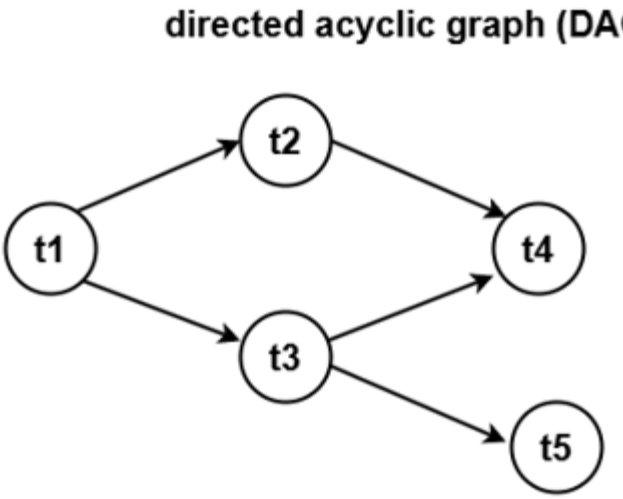
ا.RMS
ب.EDF



جواب سوال 2)

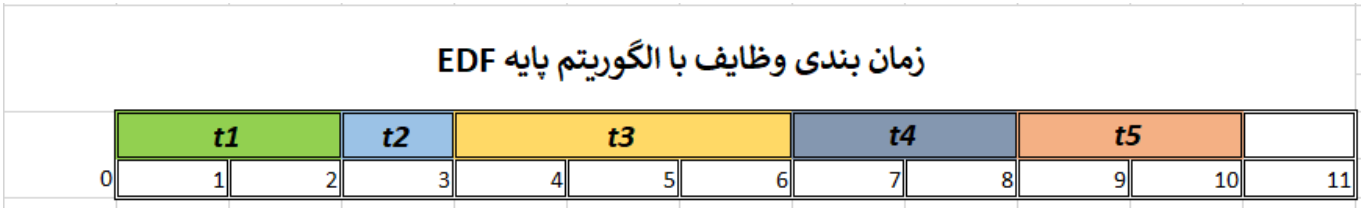
بخش الف:

ترسیم گراف وابستگی)



بخش ب:

زمان بندی با الگوریتم EDL) **بله** همان طوریکه در شکل زیر می ببینید زمان بندی ممکن است.



بخش ج:

برای اینکه وظایف را با استفاده از الگوریتم EDF* زمان بندی کنیم در قدم اول باید $d`_i$ , $r`_j$ حساب کنیم

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

$$d'_i = \min(d_i, d'_j - C_j)$$

$r_j`$ ها را حساب کنیم پس داریم

$r_1` = 0$
$r_2` = \max(1, 0 + 2) = \max(1, 2) = 2$
$r_3` = \max(3, 0 + 2) = \max(3,2) = 3$
$r_4` = \max(4, 2 + 1, 3 + 3) = \max(4,3,6) = 6$
$r_5` = \max(5, 3 + 3) = \max(5,6) = 6$

$d_i`$ ها را حساب کنیم پس داریم

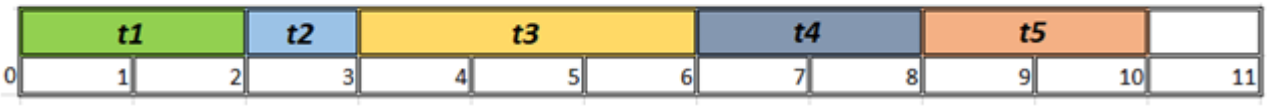
$d_5` = d_5 = 11$
$d_4` = d_4 = 10$
$d_3` = \min(8, 10 - 2, 11 - 2) = \min(8,8,9) = 8$
$d_2` = \min(7, 10 - 2) = \min(7,8) = 7$
$d_1` = \min(6, 7 - 1, 8 - 3) = \min(6,6,5) = 5$

آپدیت جدول تسک ها با استفاده از الگوریتم *EDF

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

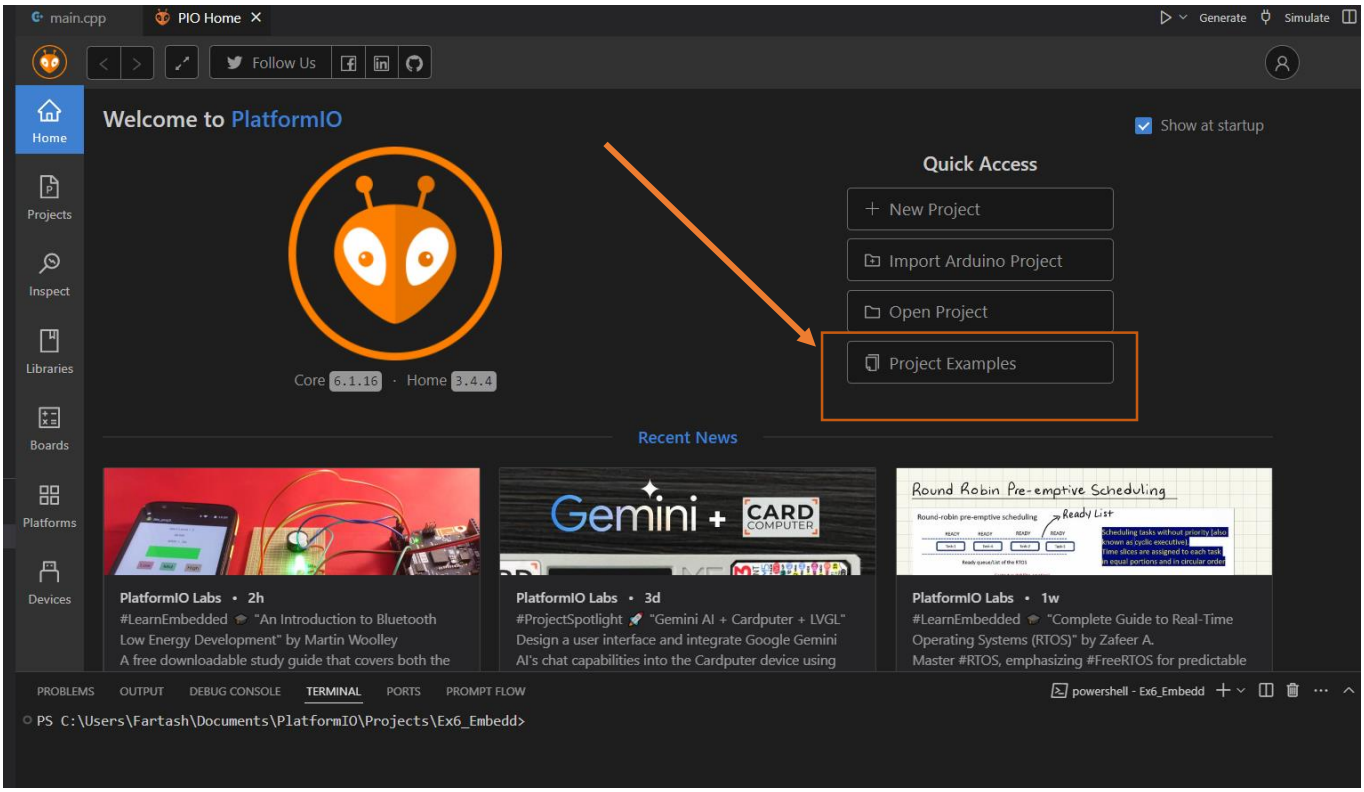
بله همان طوریکه در شکل زیر می ببینید با استفاده از دلاین ها و زمان منتشر جدید که بدست آوردیم امکان زمان بندی ممکن است.

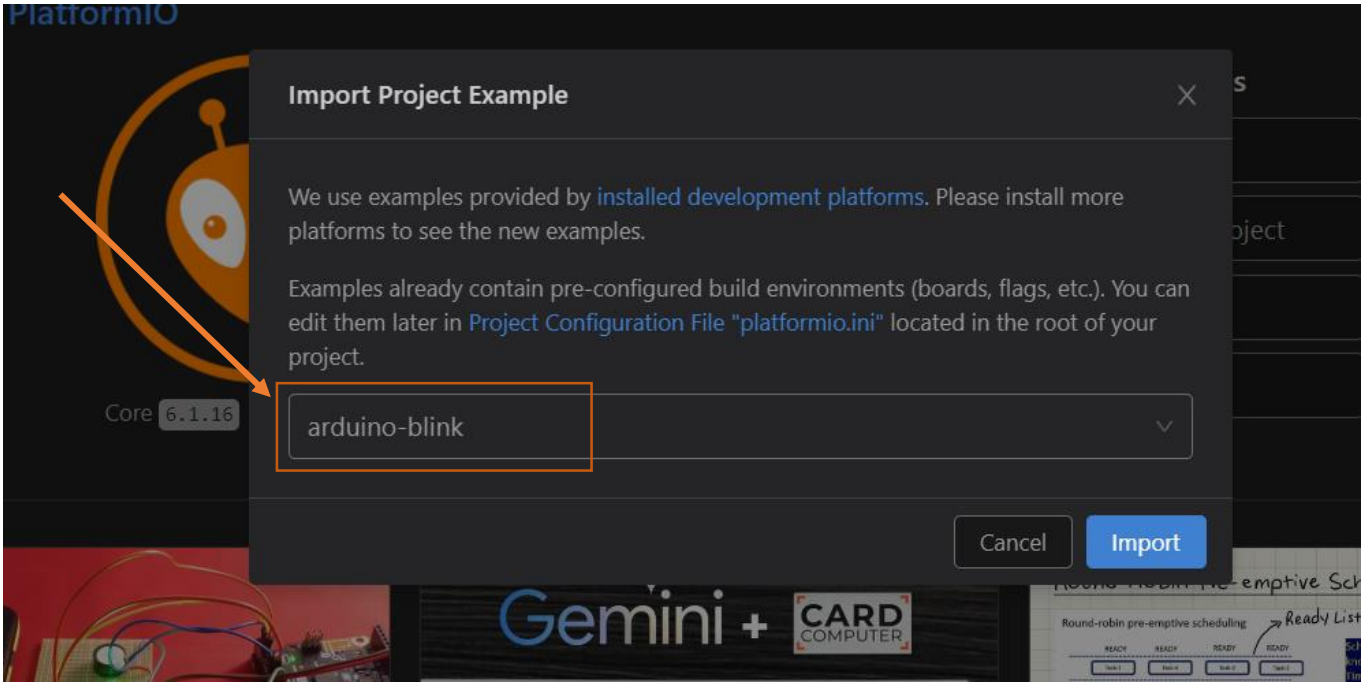
زمان بندی وظایف با الگوریتم پایه EDF*

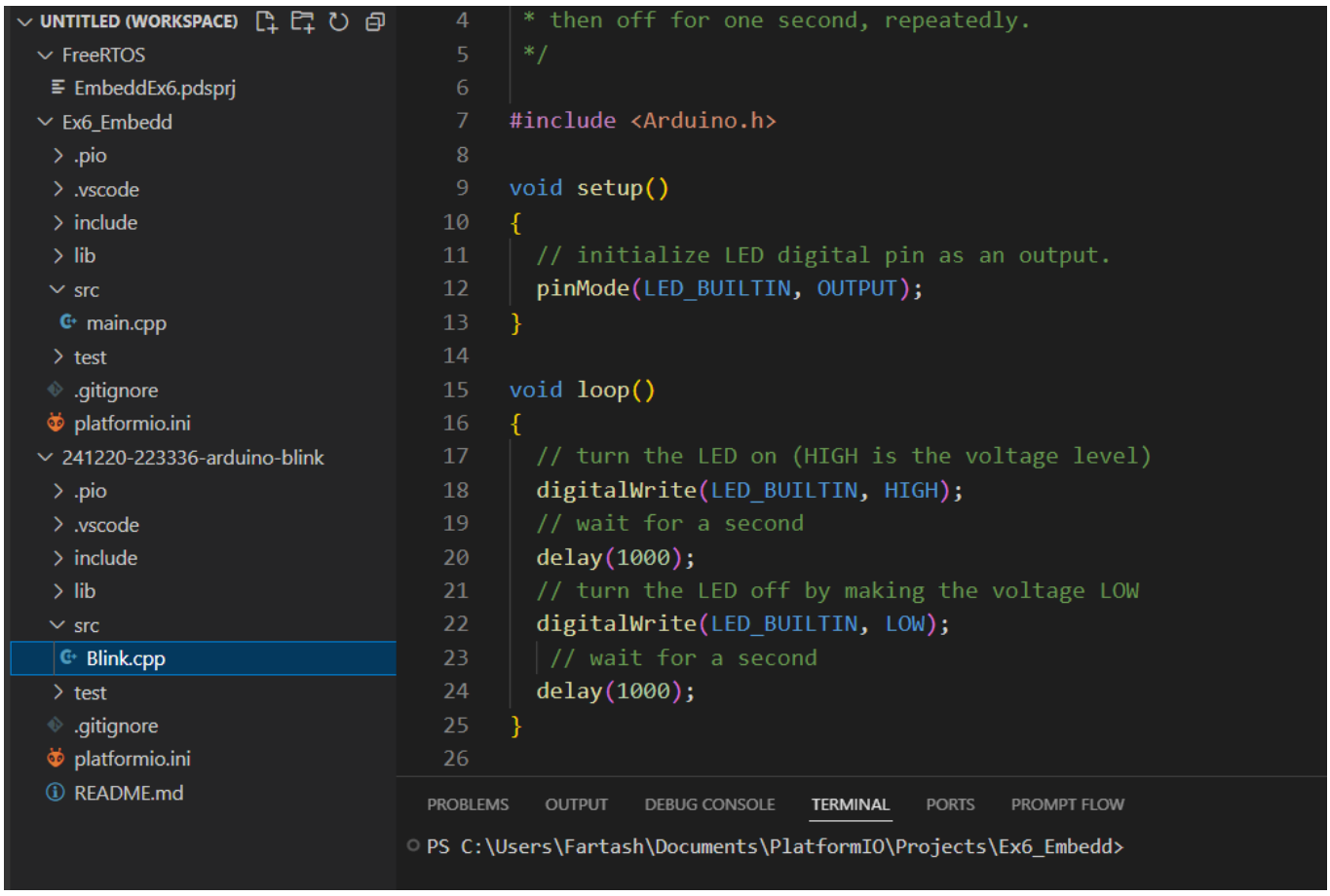


جواب سوال 3)

قسمت الف)




PlatformIO
Import Project Example
We use examples provided by installed development platforms. Please install more platforms to see the new examples.
Examples already contain pre-configured build environments (boards, flags, etc.). You can edit them later in Project Configuration File "platformio.ini" located in the root of your project.
arduino-blink
Cancel
Import
Core 6.1.16
Gemini + CARD COMPUTER


UNTITLED (WORKSPACE)
FreeRTOS
EmbeddEx6.pdsprj
Ex6_Embedd
.pio
.vscode
include
lib
src
main.cpp
test
.gitignore
platformio.ini
241220-223336-arduino-blink
.pio
.vscode
include
lib
src
Blink.cpp
test
.gitignore
platformio.ini
README.md
* then off for one second, repeatedly.
*/
#include <Arduino.h>
void setup()
{
// initialize LED digital pin as an output.
pinMode(LED_BUILTIN, OUTPUT);
}
void loop()
{
// turn the LED on (HIGH is the voltage level)
digitalWrite(LED_BUILTIN, HIGH);
// wait for a second
delay(1000);
// turn the LED off by making the voltage LOW
digitalWrite(LED_BUILTIN, LOW);
// wait for a second
delay(1000);
}
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PROMPT FLOW
PS C:\Users\Fartash\Documents\PlatformIO\Projects\Ex6_Embedd>

قسمت ب)

بخش الف:

وظایف تولید خروجی‌ها:

1. **وظیفه vFanSpeedTask:**
   - مسئول تنظیم سرعت فن است.
   - سرعت فن از طریق صفی به نام xSpeedQueue دریافت می‌شود.
   - با استفاده از تابع analogWrite مقدار سرعت (Duty Cycle) به پایه فن اعمال می‌شود.
   - در صورت دریافت سرعت جدید، مقدار به‌روز می‌شود و پیام مناسبی برای نمایش سرعت فن ثبت می‌شود.
2. **وظیفه vFanDirectionTask:**
   - مسئول تنظیم جهت چرخش فن است.
   - مقدار جهت از طریق صف xDirectionQueue دریافت می‌شود.
   - از یک سروو موتور برای تغییر جهت استفاده می‌کند و موقعیت جدید سروو را بر اساس مقدار دریافت‌شده تنظیم می‌کند.
   - تغییر جهت با حرکت نرم (Smooth) انجام می‌شود و موقعیت نهایی ثبت می‌گردد.

وظایف پردازش ورودی‌ها:

1. **وظیفه vInputProcessingTask:**
   - ورودی‌های مربوط به دما و وضعیت کلید را پردازش می‌کند.
   - دمای سنسور از طریق پایه TEMP_SENSOR_PIN خوانده می‌شود و به مقدار سلسیوس تبدیل می‌شود.
   - وضعیت کلید از پایه SWITCH_PIN خوانده می‌شود.
   - بر اساس دما، مقدار سرعت و بر اساس وضعیت کلید، مقدار جهت چرخش تعیین شده و در صف‌های مربوطه قرار می‌گیرد.
2. **تابع Callback تایمر vTimerCallback:**
   - به صورت دوره‌ای (هر ۱۰۰۰ میلی‌ثانیه) یک مقدار سرعت پیش‌فرض را به صف xSpeedQueue ارسال می‌کند.
   - این تابع برای تولید رویدادهای دوره‌ای استفاده می‌شود و به‌نوعی مکمل وظایف پردازش ورودی‌ها است.

نکات اضافی:

- **استفاده از وقفه :**در برنامه از یک وقفه شبیه‌سازی‌شده (vISRHandler) استفاده شده است که وظیفه فعال‌سازی یک Semaphore را برای هماهنگی وظیفه‌ها برعهده دارد.
- **ارتباط وظیفه‌ها :**وظایف از طریق صف‌ها و Semaphore با یکدیگر ارتباط دارند و هرکدام بخشی از کار را مستقل و همزمان انجام می‌دهند.

این ساختار به خوبی اصول طراحی برنامه‌های چندوظیفه‌ای را نشان می‌دهد و ارتباط مؤثر بین وظیفه‌ها و استفاده از منابع بهینه را تضمین می‌کند.

بخش دوم:

وظایف متناوب:(Time-Triggered)
این وظایف به صورت دوره‌ای اجرا می‌شوند و زمان‌بندی آن‌ها مشخص است.

1. **وظیفه vInputProcessingTask:**
   - **نوع تحریک :**متناوب.
   - **روش اجرا :**این وظیفه هر 500 میلی‌ثانیه با استفاده از تأخیر (vTaskDelay) اجرا می‌شود.
   - **وظیفه :**خواندن ورودی‌های دما و کلید، پردازش آن‌ها و ارسال مقادیر به صف‌های xSpeedQueue و xDirectionQueue.
2. **تابع vTimerCallback:**
   - **نوع تحریک :**متناوب.
   - **روش اجرا :**این تابع توسط یک تایمر نرم‌افزاری FreeRTOS به نام xTimer هر 1000 میلی‌ثانیه اجرا می‌شود.
   - **وظیفه :**ارسال مقدار سرعت پیش‌فرض به صف xSpeedQueue برای به‌روزرسانی متناوب سرعت فن.

وظایف نامتناوب:(Event-Triggered)
این وظایف فقط در پاسخ به یک رویداد (مانند داده در صف یا فعال شدن Semaphore) اجرا می‌شوند.

1. **وظیفه :vFanSpeedTask**
   - **نوع تحریک :**نامتناوب.
   - **روش اجرا :**زمانی که داده‌ای در صف xSpeedQueueقرار گیرد، وظیفه با استفاده از xQueueReceiveفعال می‌شود.
   - **وظیفه :**تنظیم سرعت فن با مقدار دریافتی.
2. **وظیفه :vFanDirectionTask**
   - **نوع تحریک :**نامتناوب.
   - **روش اجرا :**زمانی که داده‌ای در صف xDirectionQueueقرار گیرد، وظیفه با استفاده از xQueueReceiveفعال می‌شود.
   - **وظیفه :**تغییر جهت چرخش فن با مقدار دریافتی.
3. **Semaphoreمربوط به وقفه:(ISR)**
   - **نوع تحریک :**نامتناوب.
   - **روش اجرا :**وقفه شبیه‌سازی‌شده (vISRHandler) یک Semaphore باینری به نام xISRTriggerرا آزاد می‌کند. وظیفه vInputProcessingTaskمی‌تواند در صورت دریافت این Semaphore سریعاً به ورودی‌های جدید پاسخ دهد.
   - **وظیفه :**همگام‌سازی یک رویداد خاص (مانند تغییر حالت کلید یا وضعیت خاص) با وظایف.

نحوه اتصال وظایف به:ISR

1. **وقتی که ورودی جدیدی در صف‌ها قرار می‌گیرد:**
   - وظایف vFanSpeedTaskو vFanDirectionTaskاز این مکانیزم استفاده می‌کنند. به محض اینکه داده‌ای وارد صف شود، وظیفه از حالت بلوک خارج شده و اجرا می‌شود.
2. **وقتی که یک وقفه خارجی شبیه‌سازی شده اجرا می‌شود:**
   - در (vISRHandler) ISR از xSemaphoreGiveFromISRبرای فعال کردن وظایف مرتبط (مانند vInputProcessingTaskاستفاده شده است. این روش وظیفه را فوراً در پاسخ به رویداد فعال می‌کند.
3. **با استفاده از تایمر:**
   - وظیفه متناوب vTimerCallbackبه تایمر نرم‌افزاری FreeRTOS متصل است. این تایمر هر 1000 میلی‌ثانیه یک بار رویداد ایجاد می‌کند.

جدول خلاصه:

وظیفه	نوع تحریک	روش اتصال به ISR یا تایمر
`vFanSpeedTask`	نامتناوب	صف `xSpeedQueue` .
`vFanDirectionTask`	نامتناوب	صف `xDirectionQueue` .
`vInputProcessingTask`	متناوب/نامتناوب	تأخیر زمانی / Semaphore.
`vTimerCallback`	متناوب	تایمر نرم‌افزاری `xTimer` .
وقفه شبیه‌سازی‌شده	نامتناوب	Semaphore `xISRTrigger`.

بخش سوم:

ارتباط بین وظایف:

ارتباط بین وظایف با **صف‌ها** انجام شده است. وظایف با ارسال و دریافت داده از صف‌ها، با یکدیگر ارتباط برقرار می‌کنند.

1. ارتباط وظایف از طریق `xSpeedQueue` :

- **وظایف فرستنده:**
- `vInputProcessingTask` : سرعت محاسبه‌شده بر اساس دمای سنسور را در این صف قرار می‌دهد.
- `vTimerCallback` : به صورت دوره‌ای مقادیر سرعت پیش‌فرض را در این صف قرار می‌دهد.
- **وظیفه گیرنده:**
- `vFanSpeedTask` : مقادیر موجود در صف را دریافت کرده و سرعت فن را تنظیم می‌کند.
- **نحوه اجرا:** وظیفه گیرنده (مصرف‌کننده) از طریق تابع `xQueueReceive` منتظر داده در صف باقی می‌ماند و بلافاصله پس از دریافت داده، اجرا می‌شود.

2. ارتباط وظایف از طریق `xDirectionQueue` :

- **وظیفه فرستنده:**
- `vInputProcessingTask` : مقدار جهت چرخش فن را بر اساس وضعیت کلید، محاسبه کرده و در این صف قرار می‌دهد.
- **وظیفه گیرنده:**
- `vFanDirectionTask` : مقادیر موجود در صف را دریافت کرده و جهت چرخش فن را از طریق سروو موتور تنظیم می‌کند.
- **نحوه اجرا:** مشابه `xSpeedQueue` ، وظیفه مصرف‌کننده به محض دریافت داده از صف، اجرا می‌شود.

ارتباط بین وظایف و ISRها:

ارتباط بین وظایف و ISRها با استفاده از **سمافور باینری (Binary Semaphore)** پیاده‌سازی شده است.

1. سمافور `xISRTrigger` :

- **فرستنده:**
- وقفه شبیه‌سازی‌شده `vISRHandler` : به هنگام وقوع یک رویداد (مثلاً تغییر وضعیت کلید)، با استفاده از تابع `xSemaphoreGiveFromISR` این سمافور را آزاد می‌کند.
- **وظیفه گیرنده:**
- `vInputProcessingTask` : منتظر فعال شدن سمافور باقی می‌ماند (`xSemaphoreTake`) و به محض دریافت سیگنال، عملیات پردازش ورودی را انجام می‌دهد.
- **نحوه اجرا:** وقفه باعث می‌شود وظیفه وابسته به رویداد، سریعاً و بدون تأخیر به وقفه پاسخ دهد.

ارتباط وظایف با تایمر:

از تایمر نرم‌افزاری FreeRTOS برای ایجاد ارتباط زمان‌بندی‌شده بین وظایف استفاده شده است.

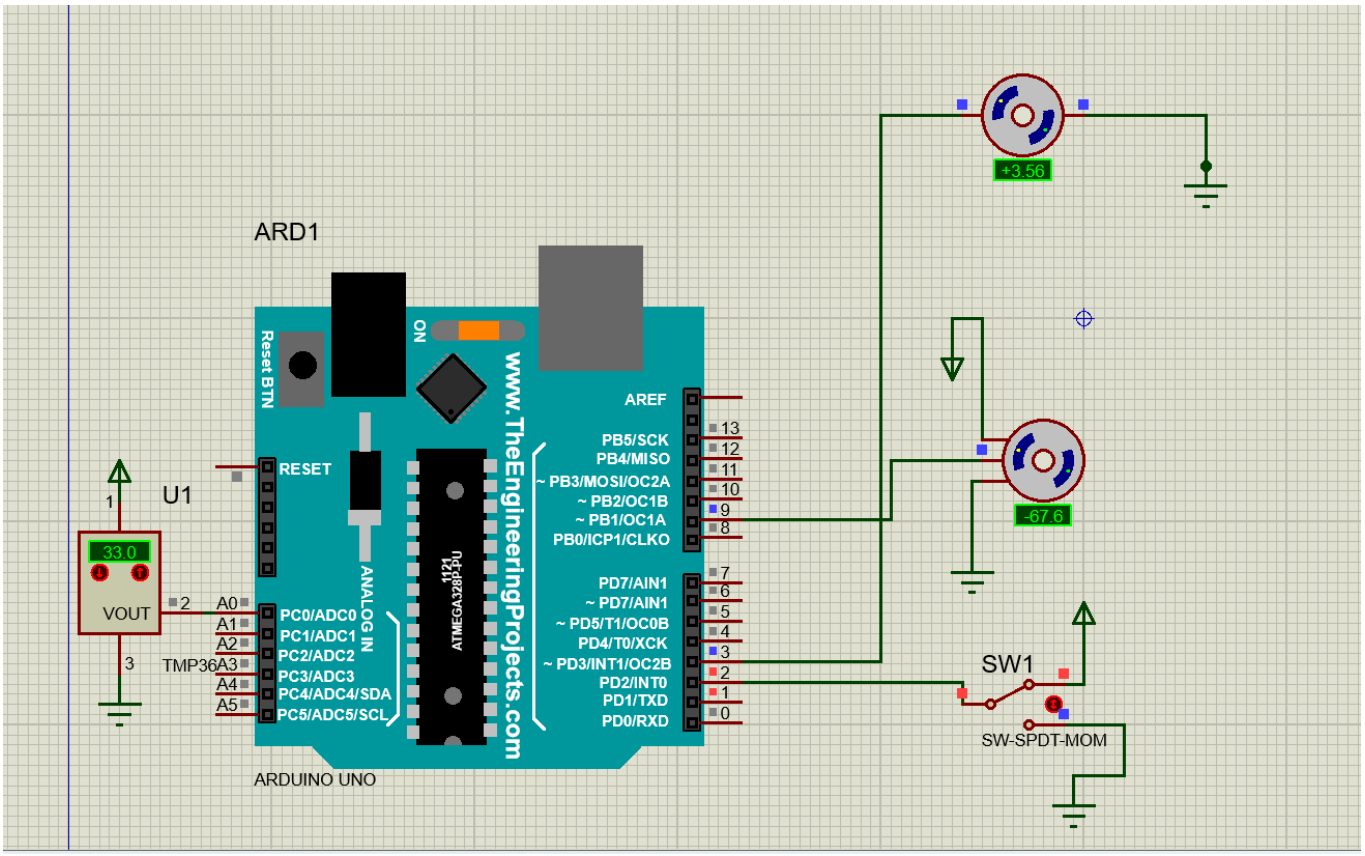
1. تایمر `xTimer` :

- **وظیفه تولیدکننده:**
- تایمر نرم‌افزاری `xTimerCreate` هر 1000 میلی‌ثانیه یک بار تابع `vTimerCallback` را اجرا می‌کند.
- **وظیفه گیرنده:**
- `vFanSpeedTask` : با دریافت داده از صف `xSpeedQueue` ، مقادیر جدید سرعت را اعمال می‌کند.
- **نحوه اجرا:** تایمر به صورت دوره‌ای یک مقدار سرعت پیش‌فرض را به صف ارسال می‌کند و وظیفه مربوطه را تحریک می‌کند.

جمع‌بندی ارتباطات:

ابزار ارتباطی	فرستنده	گیرنده	توضیح
`xSpeedQueue`	`vInputProcessingTask` و `vTimerCallback`	`vFanSpeedTask`	انتقال مقادیر سرعت فن.
`xDirectionQueue`	`vInputProcessingTask`	`vFanDirectionTask`	انتقال جهت چرخش فن.
`xISRTrigger`	`vISRHandler`	`vInputProcessingTask`	هماهنگی با وقفه‌ها.
تایمر `xTimer`	سیستم زمان‌بندی FreeRTOS	`vFanSpeedTask`	تولید رویدادهای دوره‌ای برای سرعت.

قسمت ج) بخش 1)



بخش 3و2) تست شبه سازی حسگرها و فعالگرهای