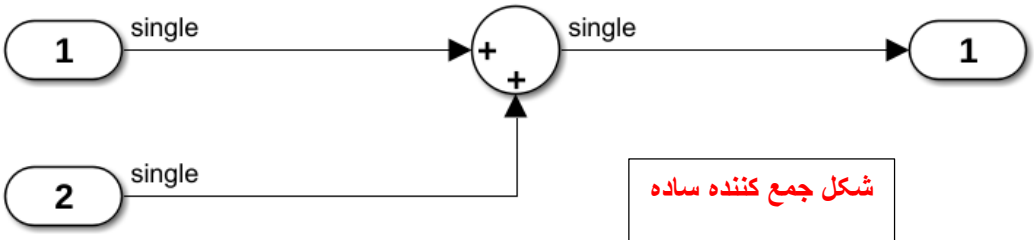
بسم الله الرحمن الرحیم

گزارش تمرین 4

درس مبانی سیستم نهفته و بیدرنگ

بخش اول)



شکل جمع کننده ساده

Code

untitled.c ▾ Search

```
18  #include "untitled.h"
19
20  /* External inputs (root inport signals with default storage) */
21  ExtU rtU;
22
23  /* External outputs (root outports fed by signals with default storage) */
24  ExtY rtY;
25
26  /* Model step function */
27  void untitled_step(void)
28  {
29    /* Outport: '<Root>/Out1' incorporates:
30     *  Inport: '<Root>/In1'
31     *  Inport: '<Root>/In2'
32     *  Sum: '<Root>/Sum'
33     */
34    rtY.Out1 = rtU.In1 + rtU.In2;
35  }
36
37  /* Model initialize function */
38  void untitled_initialize(void)
39  {
40    /* (no initialization code required) */
41  }
42
```

تصویر کد ایجاد شده
از مدار بالا

1) همان طوریکه در تصویر بالا می ببینید **کد** ایجاد شده شامل فراخوانی کتابخانه "#include "untitled.h و متغیر های ExtU rtU برای ورودی و ExtU rtY برای خروجی است و دو تابع داریم

- **void untitled_initialize(void)** برای مقددهی اولیه است
- **void untitled_step(void)** برای جمع کردن دو عدد است.

2) منطق اصلی مدل در تابع **untitled_step** پیاده‌سازی شده است. این تابع مسئولیت محاسبه‌ی خروجی (جمع دو ورودی In1 و In2) را بر عهده دارد و در هر فراخوانی، این محاسبه را انجام می‌دهد.

می توانید کد این فایل به اسم ActiveClass1.c دسترسی داشته باشید.

3) اجرای بی‌درنگ (Real-Time Execution) کد

برای اجرای بی‌درنگ این کد، مراحل زیر را می‌توان انجام داد:

زمان‌بندی دوره‌ای:

برای اجرای تابع untitled_step در فواصل منظم (بی‌درنگ)، نیاز به یک تایمر یا مکانیزم زمان‌بندی دوره‌ای داریم. در محیط‌های بی‌درنگ، تایمر سیستم عامل (مانند تایمرهای سخت‌افزاری در سیستم‌های امبدد یا استفاده از ابزارهای خاص در سیستم‌عامل‌های بی‌درنگ) می‌تواند این کار را انجام دهد.

سیستم عامل بی‌درنگ (RTOS):

اگر کد شما روی یک سیستم عامل بی‌درنگ مانند(FreeRTOS، VxWorks، یا QNX) اجرا می‌شود، می‌توانید یک تسک (task) ایجاد کنید که تابع untitled_step را در فواصل زمانی مشخص فراخوانی کند.

RTOS این امکان را فراهم می‌کند تا تسک‌ها با اولویت‌های مشخص و با حداقل تأخیر (تا حد امکان) اجرا شوند.

تایمر در سیستم‌های امبدد:

در سیستم‌های امبدد (مانند میکروکنترلرها)، از تایمرهای سخت‌افزاری برای تنظیم نرخ اجرای تابع استفاده کنید. به عنوان مثال، در کد تایمر، untitled_step را در بازه‌های مشخصی فراخوانی کنید.

اجرای زمان‌بندی‌شده در محیط‌های غیر بی‌درنگ:

در محیط‌های غیر بی‌درنگ (مانند سیستم عامل‌های معمولی)، می‌توانید از توابع زمان‌بندی مانند sleep در حلقه‌ها استفاده کنید، اما به خاطر تأخیرهای سیستمی، این روش کاملاً بی‌درنگ نیست و نمی‌توان دقت بالایی در اجرای زمان‌بندی آن داشت.

بخش دوم)

کد C تولید شده در این پروژه شامل توابع و ساختارهایی است که برای پیاده‌سازی منطق کنترل‌کننده و مدیریت ورودی/خروجی‌ها (I/O) استفاده می‌شوند. این کد احتمالاً با نرم‌افزار Simulink ایجاد شده و برای کنترل سخت‌افزار مشخصی مانند AVR تنظیم شده است. در ادامه بخش‌های اصلی این کد بررسی می‌شوند:

1. ساختار داده‌ها

در ابتدا، چندین ساختار داده برای ذخیره‌سازی وضعیت و مقادیر سیگنال‌های ورودی و خروجی تعریف شده‌اند:

B_classActivity0x2820x29_T و DW_classActivity0x2820x29_T
این دو ساختار به ترتیب برای ذخیره سیگنال‌های بین بلوک‌های مختلف و وضعیت‌های داخلی سیستم استفاده می‌شوند. اطلاعاتی که باید بین توابع به اشتراک گذاشته شوند، در اینجا ذخیره می‌شوند.

2. تابع classActivity0x2820x29_step

این تابع هسته اصلی منطق کنترل‌کننده را پیاده‌سازی می‌کند و در هر گام زمانی اجرا می‌شود. عملکردهای اصلی این تابع شامل موارد زیر است:

خواندن ورودی‌های آنالوگ:

این کد از ورودی‌های آنالوگ برای اندازه‌گیری سیگنال‌های مختلف استفاده می‌کند.

از توابع MW_AnalogInSingle_ReadResult استفاده می‌شود تا مقادیر ورودی آنالوگ از پین‌های مشخص دریافت و در متغیر b_varargout_1 ذخیره شوند.

مقایسه ورودی‌ها:

پس از خواندن ورودی‌ها، مقادیر ورودی با مقادیر ثابت مقایسه می‌شوند و نتایج به صورت بولی (true یا false) برای تصمیم‌گیری استفاده می‌شوند.

برای مثال، rtb_Compare نتیجه مقایسه یک ورودی با مقدار مرجع را ذخیره می‌کند و به کنترل وضعیت‌ها کمک می‌کند.

نمودار حالت (State Chart):

یک منطق حالت که رفتار کنترل‌کننده را مدیریت می‌کند.

این منطق حالت تعیین می‌کند که کنترل‌کننده در چه حالتی (مانند Shutdown، OP، ON، یا Off) قرار بگیرد و همچنین انتقال بین حالت‌ها را بر اساس ورودی‌های آنالوگ انجام می‌دهد.

حالت‌ها و انتقالات: هر حالت مشخص می‌کند که خروجی‌های دیجیتال به چه صورتی تنظیم شوند و هر انتقال مشخص می‌کند که تحت چه شرایطی سیستم باید به حالت جدید منتقل شود.

کنترل خروجی‌های دیجیتال:

پس از تعیین حالت‌ها و پردازش منطق کنترلی، پین‌های دیجیتال تنظیم می‌شوند.

با استفاده از توابعی مانند writeDigitalPin، مقادیر بولی که نشان‌دهنده وضعیت خروجی‌ها هستند، به پین‌های دیجیتال خاص ارسال می‌شوند.

3. تابع classActivity0x2820x29_initialize

این تابع برای مقداردهی اولیه سیستم طراحی شده است و در ابتدای اجرا، قبل از فراخوانی‌های دوره‌ای تابع step، اجرا می‌شود. کارهایی که این تابع انجام می‌دهد شامل موارد زیر است:

پیکربندی ورودی‌ها و خروجی‌ها:

تنظیم نمونه‌گیری ورودی‌های آنالوگ و اختصاص منابع سخت‌افزاری.

پیکربندی پین‌های دیجیتال برای استفاده به عنوان خروجی.

4. تابع classActivity0x2820x29_terminate

این تابع در پایان کار کنترل‌کننده اجرا می‌شود و منابع سخت‌افزاری سیستم را آزاد می‌کند:

آزادسازی منابع سخت‌افزاری:

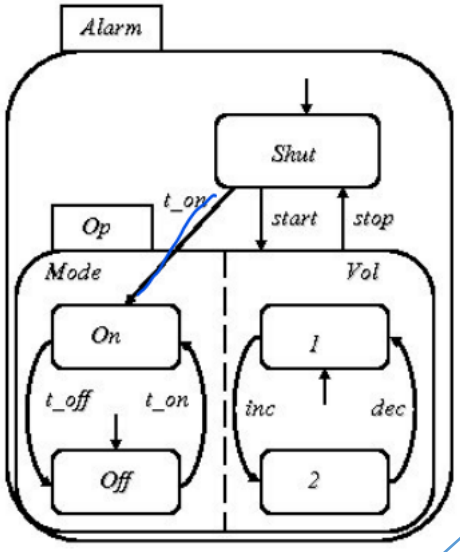
بستن دسترسی به پین‌های آنالوگ و دیجیتال.

غیرفعال کردن پین‌هایی که در طول اجرای برنامه به عنوان ورودی یا خروجی استفاده شده بودند.

نحوه عملکرد کلی کد

این کد برای مدیریت یک کنترل‌کننده ساده طراحی شده است که از ورودی‌های آنالوگ برای تصمیم‌گیری درباره وضعیت خروجی‌های دیجیتال استفاده می‌کند. منطق کنترل‌کننده از یک نمودار حالت ساده بهره می‌برد که به سیستم اجازه می‌دهد بر اساس مقادیر ورودی و حالت فعلی، وضعیت خروجی‌ها را تغییر دهد.

می توانید کد این فایل به اسم ActiveClass2.c دسترسی داشته باشید.



فرض کنید رویدادهای زیر را به‌ترتیب از چپ به راست در ورودی ببینیم:

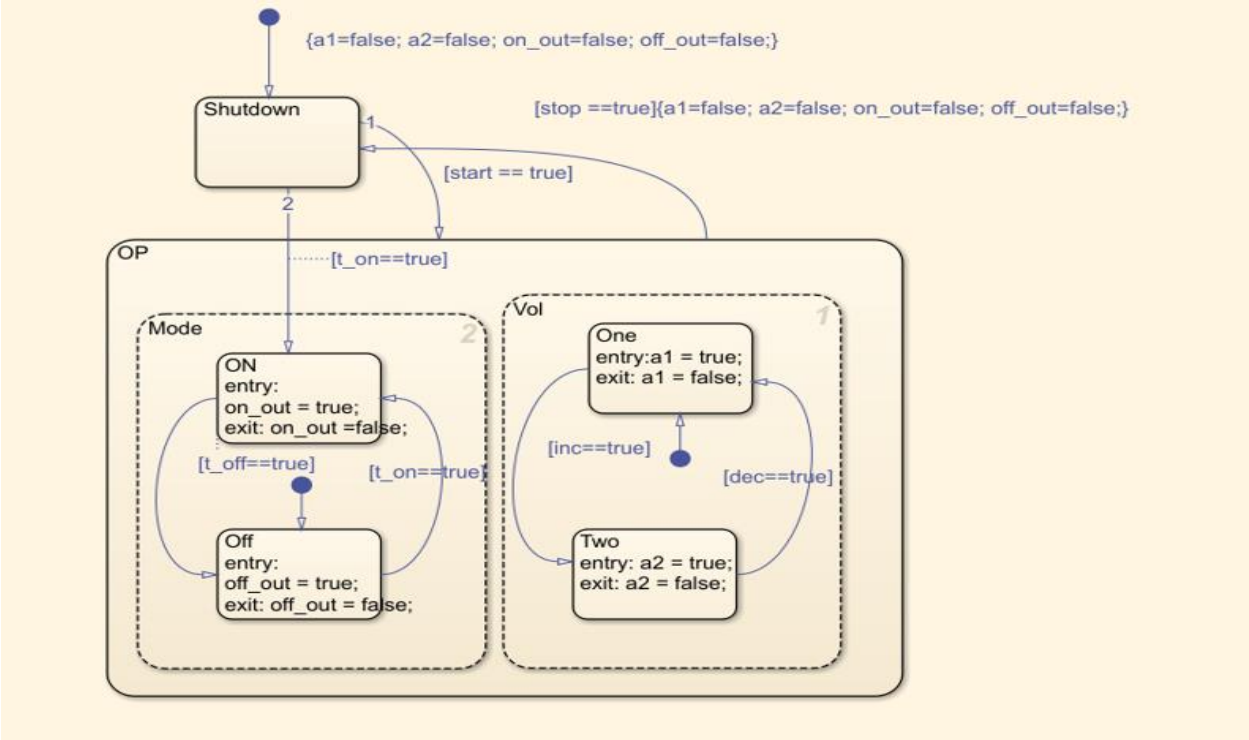
start, t_on, inc, dec, stop

passive

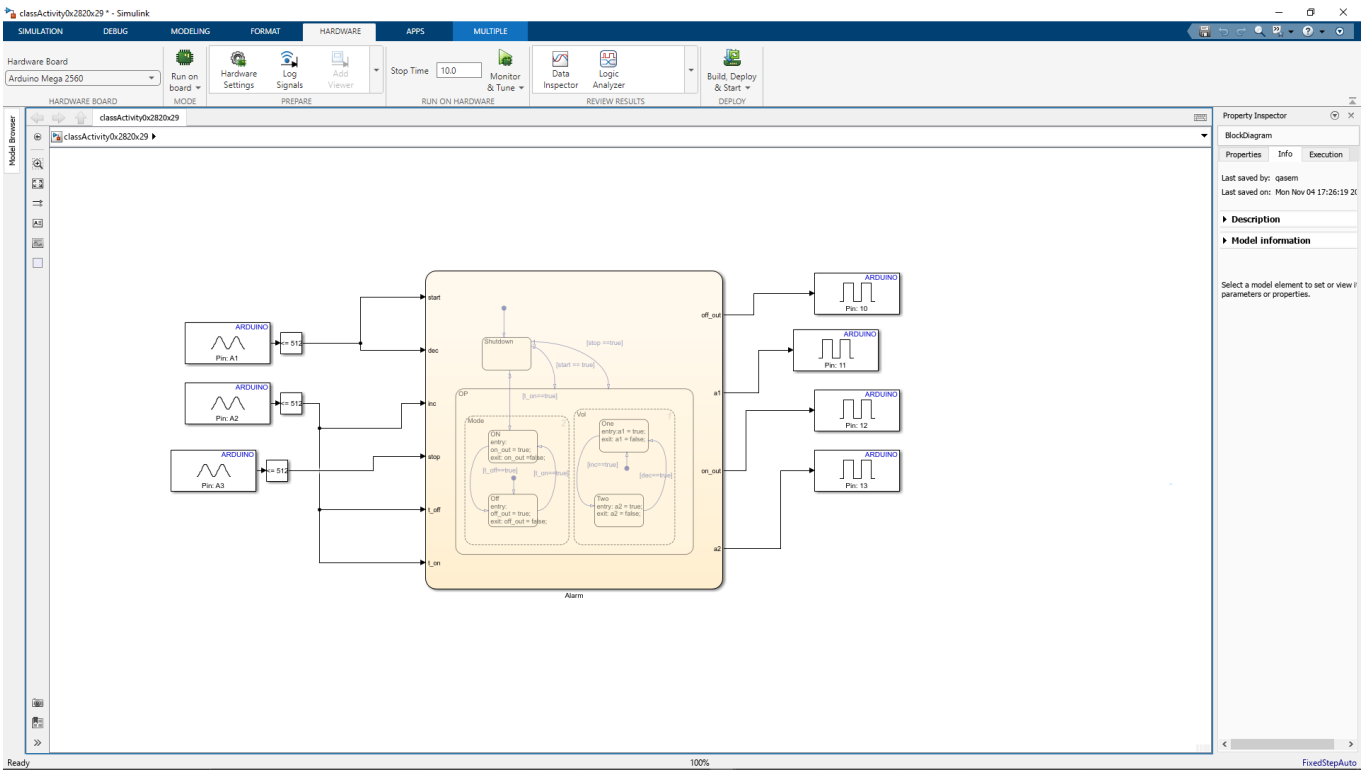
active

	Shut	Op	Mode	Vol	On	Off	1	2
reset	x				F	F	F	F
start		X	X	X	F	T	T	F
t_on		X	X		T	F	T	F
inc		X		X	T	F	F	T
dec		X		X	T	F	T	F
stop	X				F	F	F	F

F=false; T= true;

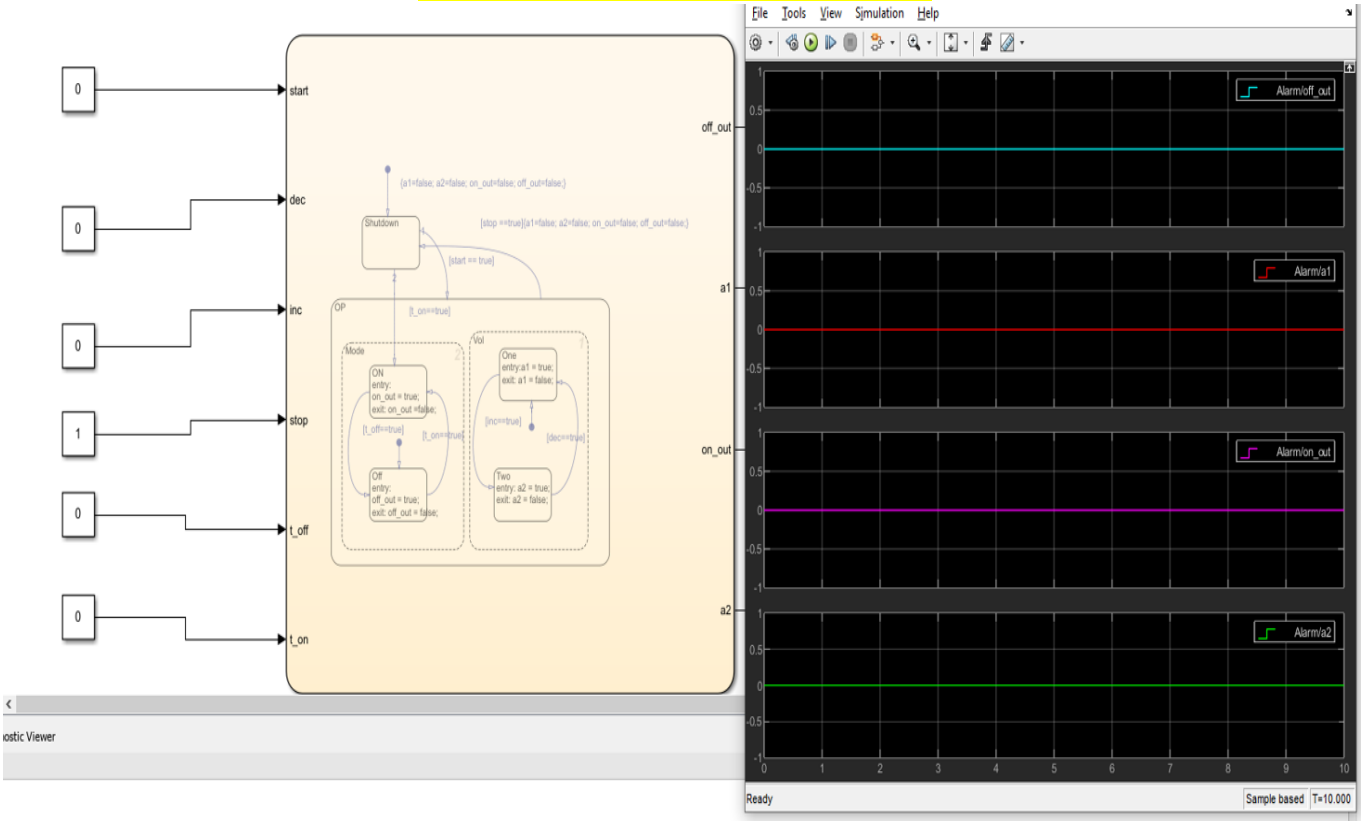


شکل مدار اصلی:

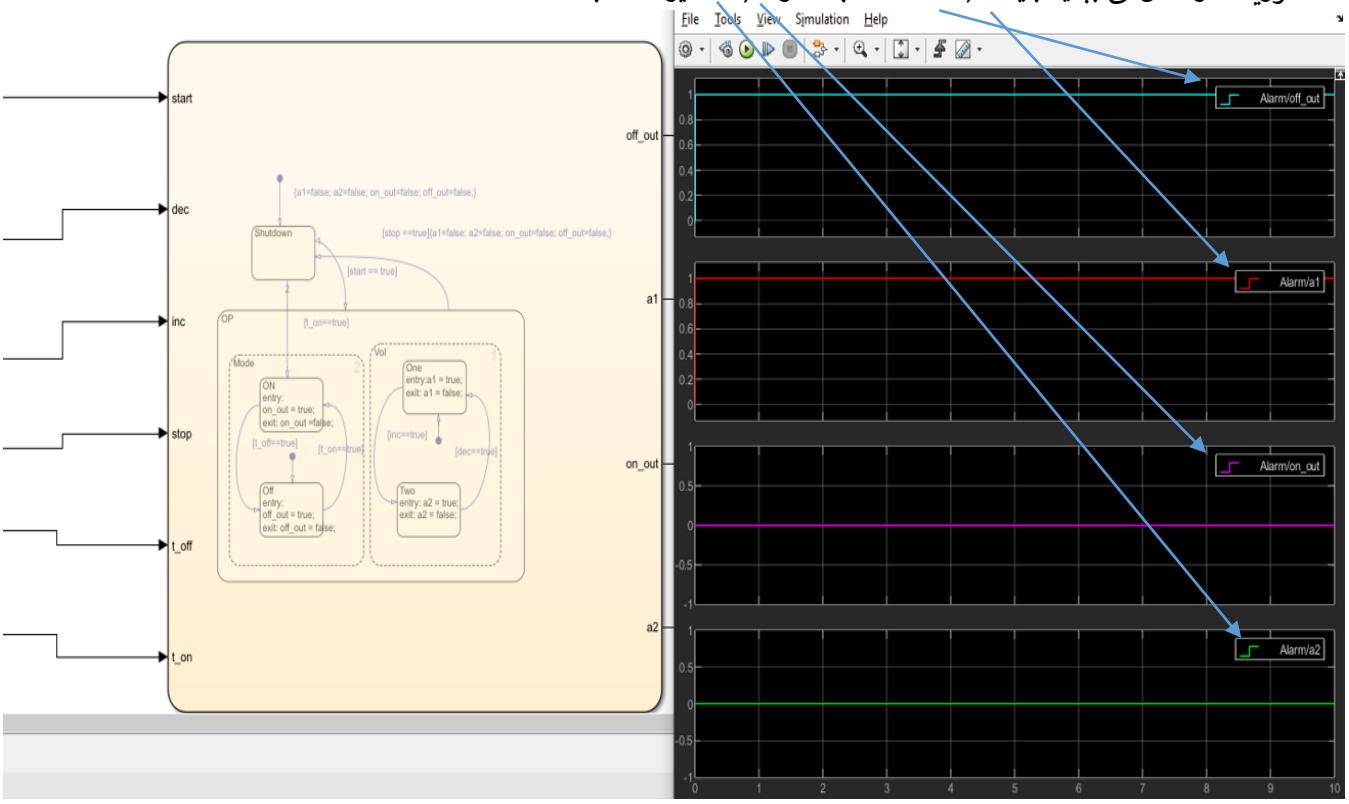


نمونه خروجی ها:

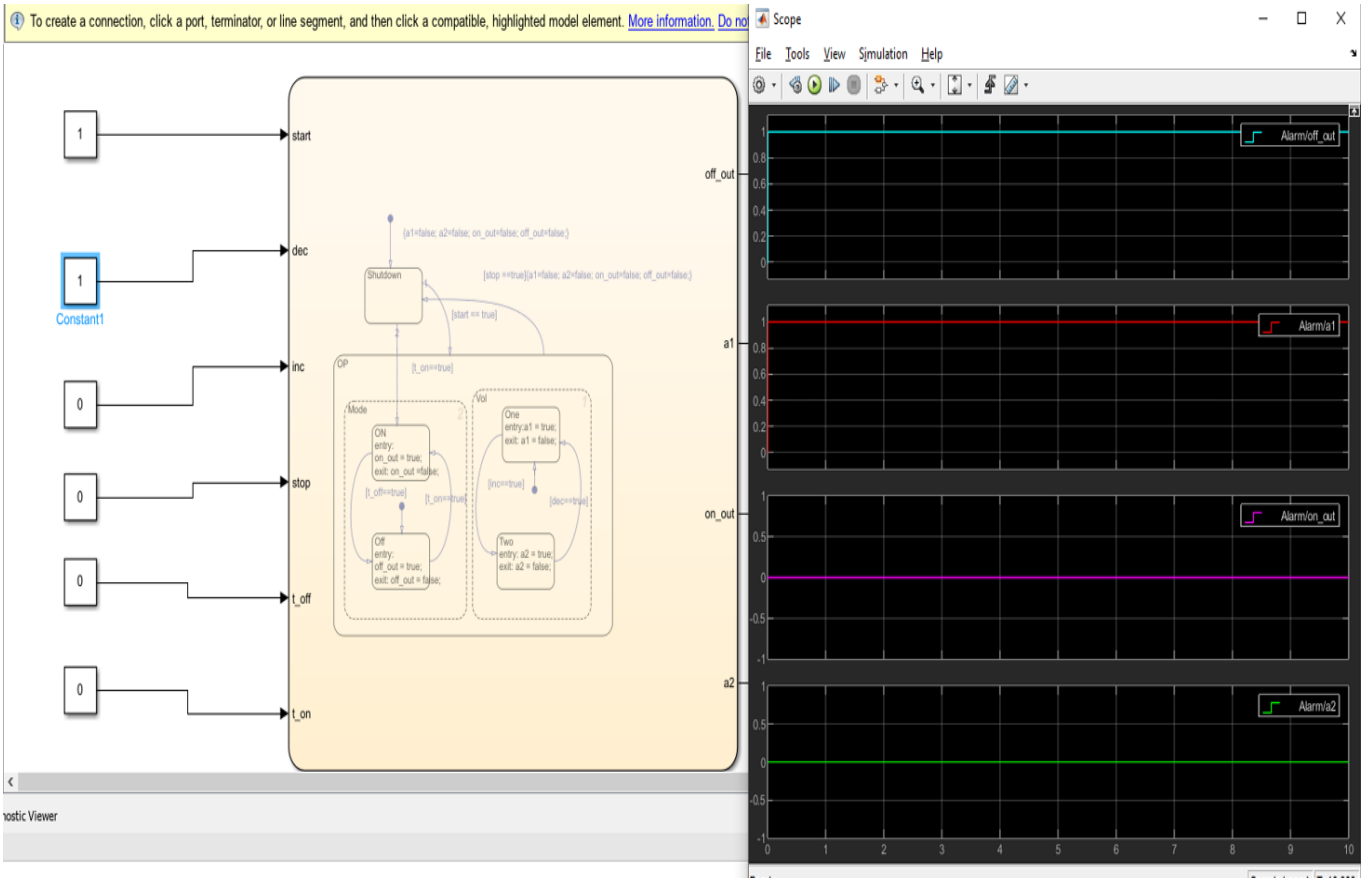
اگر ریست فعال باشه بقیه غیرفعال است(**برای تست ورودی ها بصورت دستی وارد شده !**)

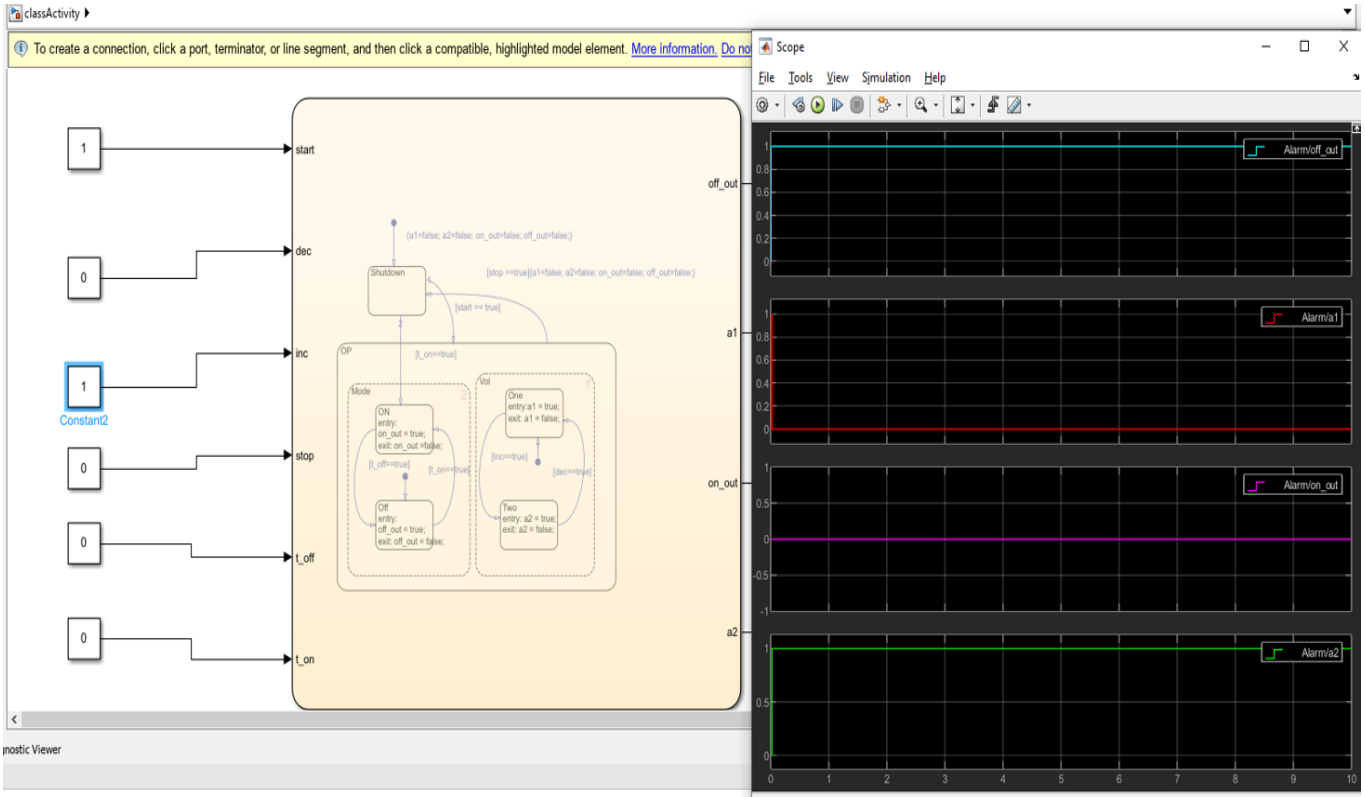


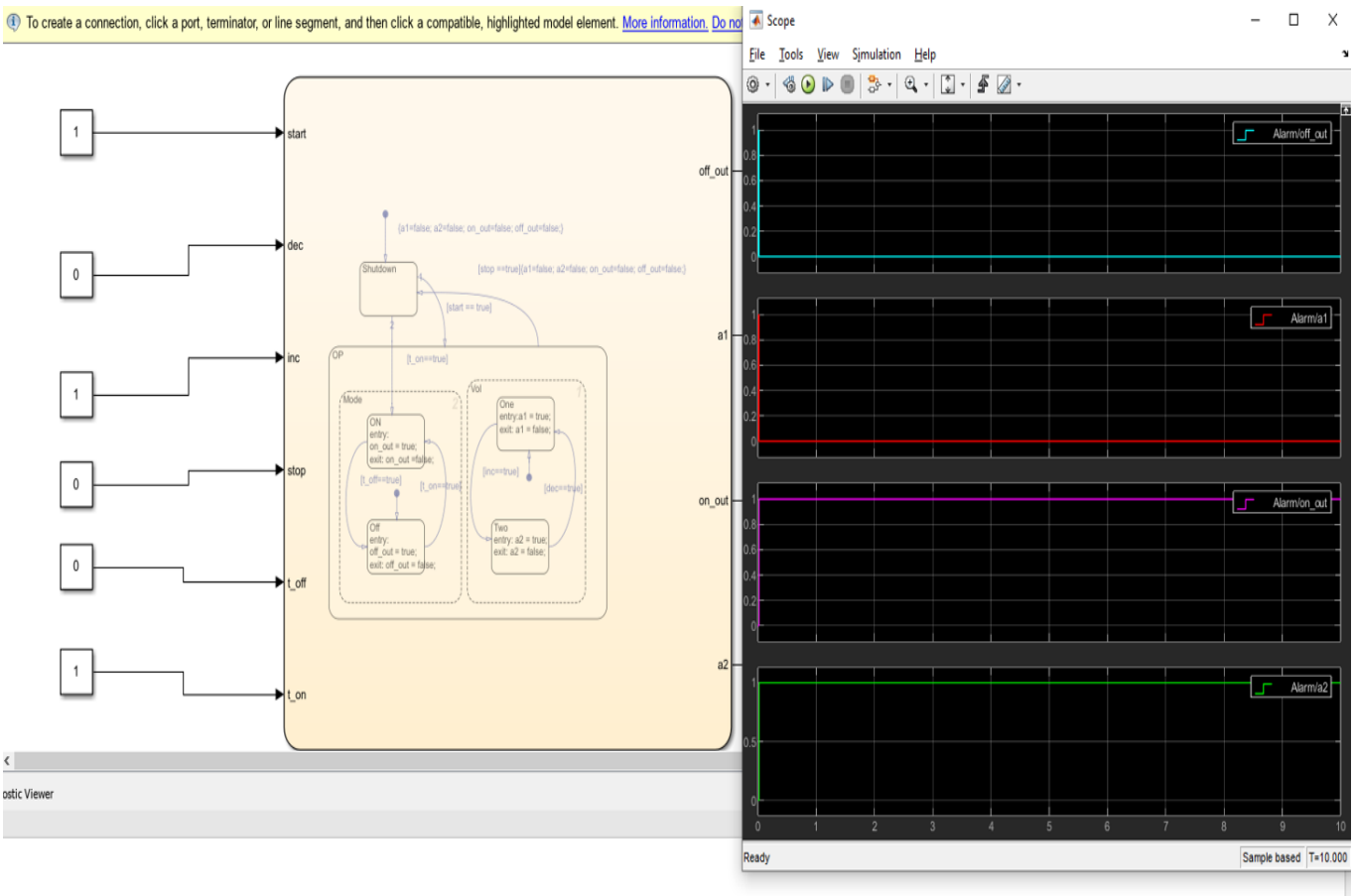
اگر ریست ما غیرفعال باشه و start فعال باشه
هماطوریکه در شکل می ببنید باید 1 , off فعال باشه و 2 ,on غیر فعال باشد.



و حالت های دیگر:




classActivity
To create a connection, click a port, terminator, or line segment, and then click a compatible, highlighted model element. More information. Do not
Scope
File Tools View Simulation Help
start
dec
inc
stop
t_off
t_on
off_out
a1
on_out
a2
Constant2
Shutdown
OP
Mode
ON
Off
Vol
One
Two
Alarm/off_out
Alarm/a1
Alarm/on_out
Alarm/a2
Ready
Sample based
T=10.000


To create a connection, click a port, terminator, or line segment, and then click a compatible, highlighted model element. More information. Do not
Scope
File Tools View Simulation Help
start
dec
inc
stop
t_off
t_on
off_out
a1
on_out
a2
Shutdown
OP
Mode
ON
Off
Vol
One
Two
Alarm/off_out
Alarm/a1
Alarm/on_out
Alarm/a2
Ready
Sample based
T=10.000