بسمه تعالی

گزارش تمرین 4

درس امبدد

گروه 13

1) این مدل خوش ساخت نیست . در استیت 1 اگر ورودی absent باشد s1 ، present میشود و با توجه به اینکه در استیت 3 هستیم و ورودی پرزنت است پس خروجی نیز پرزنت میشود پس به تضاد میخوریم .

در حالتی که در s1 باشیم و ورودی پرزنت باشد s1 پرزنت شده و مطابق روش بالا خروجی پرزنت میشود . پس در استیت 1 فقط با حالتی که ورودی پرزنت باشد خروجی پرزنت میشود و میتوانیم به استیت بعد برویم .

اگر در s2 باشیم و ورودی ابسنت باشد در اینصورت s1 پرزنت شده و خروجی نهایی پرزنت میشود که به تضاد میخوریم

در حالت دیگر اگر ورودی پرزنت باشد خروجی ابسنت است که سبب میشود s1 ابسنت باشد و در حالتی که در ماشین B در استیت s3 باشیم و ورودی ابسنت باشد خروجی B ابسنت است که سبب میشود باز به تضاد برسیم . پس به این دلیل که در s2 هیچ fixpoint وجود ندارد این ماشین خوش ساخت نیست .

2) الف)

-3	0	2	c->a
0	3	-1	b->c
1	-2	0	a->b
-1	2	0	b->a
0	0	0	b->b

چون دو سطر آخر بر حسب ضرایبی از دوسطر اول نوشته شده اند پس مرتبه ماتریس 2 میباشد .

ب) با استفاده از روش گاوس جردن داریم :

Q1=2

Q2=1

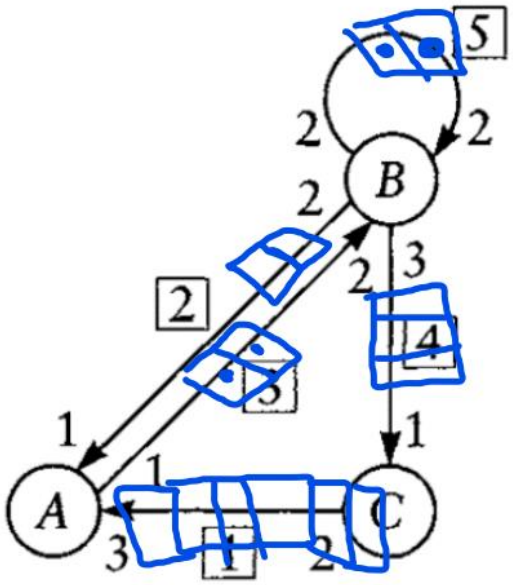
Q3=3

ج)B->C->C->C->A->A

برای اینکه B در ابتدا فایر کند باید در یال دو توکن در یال 5 و دو توکن در یال 3 مصرف کند پس یک بافر 2 دوتایی در یال 5 و یک بافر دوتایی در یال 3 قرار میدهیم . با مصرف توکن های موجود در این لاین اکتور b فایر میکند . یک بافر 3 تایی در یال 1 و یک بافر 2 تایی در یال 2 قرار میدهیم که این توکن ها را در آن ها ذخیره کنیم . اکتور C پس از مصرف سه توکن موجود در یال 4 سه بار فایر میکند که یعنی 6 توکن را در یال 1 قرار میدهد . یک بافر 6 تایی در این یال قرار میدهیم . اکتور A 3 توکن موجود در بافر یال 1 و یک توکن در بافر یال 2 را مصرف کرده و یک توکن در بافر یال 3 قرارمیدهد . دوباره همین کار تکرار شده و بافر یال 1 خالی میشود . حال که در بافر یال 3 دو توکن در آن قرار دارد مشخص است که دوباره همین روند گفته شده تکرار میشود .

پس به حداقل 15 بافر برای عملکرد صحیح نیازمندیم .

در هردو بافر یال 5 و یال 3 که ظرفیت آن ها برابر 2 است فرض شده که در ابتدا دو توکن در آنها قرار دارد .



د) در این قسمت با توجه به بخش ج ابتدا بافرهایمان را ایجاد و تنظیم میکنیم:

```
//buffers and init theme
int buffer1[6];
int buffer1i = 0;
int buffer2[2];
int buffer2i = 0;
int buffer3[2];
int buffer3i = 2;
int buffer4[3];
int buffer4i = 0;
int buffer5[2];
int buffer5i = 2;
```

سپس دو تابع اضافه کردن به بافر و کم کردن از بافر را ایجاد کردیم:

```
//add token to buffer
void buffer_add(int buffer[], int *index, int tokens[], int token_number){
    for (int i = 0; i < token_number; ++i){
        buffer[*index+i] = tokens[i];
    }
    *index += token_number;
}

//get token from buffer
int* buffer_minus(int buffer[], int *index, int token_number){
    //4 is size of int
    int mySize = token_number * 4;
    //add memory to our buffer
    int tokens[] = (int*) malloc(mySize);
    for (int i = 0; i < token_number; ++i){
        tokens[i] = buffer[i];
    }
    for (int i = 0; i < token_number; ++i){
        buffer[i] = buffer[i+1];
    }
    *index -= number;
    return tokens;
}
```

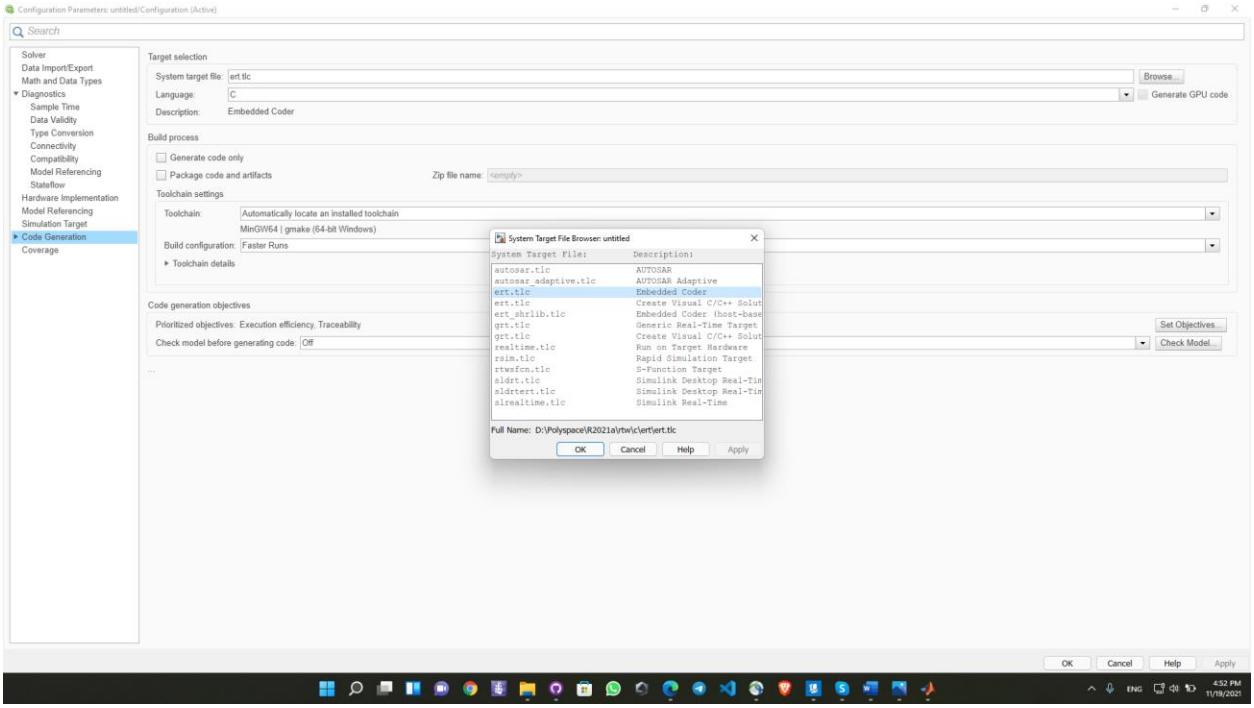
با توجه به پیاده سازی شدن تابع های a , b, c، ما تابع های اولیه آن را ایجاد کردیم، سپس از بخش های اضافه کردن به بافر و کم کردن از آن استفاده کردیم.

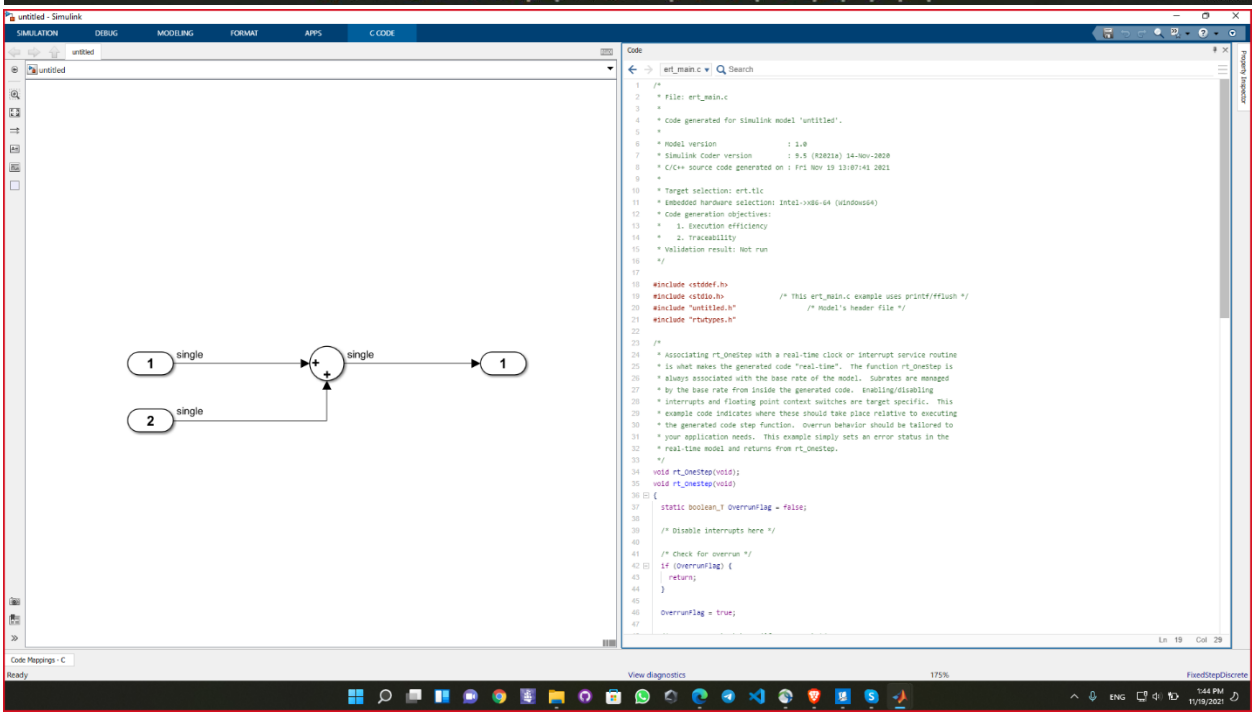
```
void a(){
    int *TokensFromBuffer2 = buffer_minus(buffer2, &buffer2i, 1);
    int *TokensFromBuffer1 = buffer_minus(buffer1, &buffer1i, 3);
    int *token = addtoA(TokensFromBuffer2 , TokensFromBuffer1);
    buffer_add(buffer3, &buffer3i, token, 1);
}

void b(){ int *TokensFromBuffer2
    int *TokensFromBuffer2 = buffer_minus(buffer2, &buffer2i, 2);
    int *TokensFromBuffer5 = buffer_minus(buffer5, &buffer5i, 2)
    int *tokens = addtoB(TokensFromBuffer2, TokensFromBuffer5);
    buffer_add(buffer2, &buffer2i, {tokens[0][0], tokens[0][1]}, 2);
    buffer_add(buffer4, &buffer4i, {tokens[2][0], tokens[2][1], tokens[2][2]}, 3);
    buffer_add(buffer5, &buffer5i, {tokens[1][0], tokens[1][1]},2);
}

void c(){
    int *TokensFromBuffer4 = buffer_minus(buffer4, &buffer4i, 1)
    int *tokens = addtoC(TokensFromBuffer4);
    buffer_add(buffer1, &buffer1i, tokens, 2);
}
```

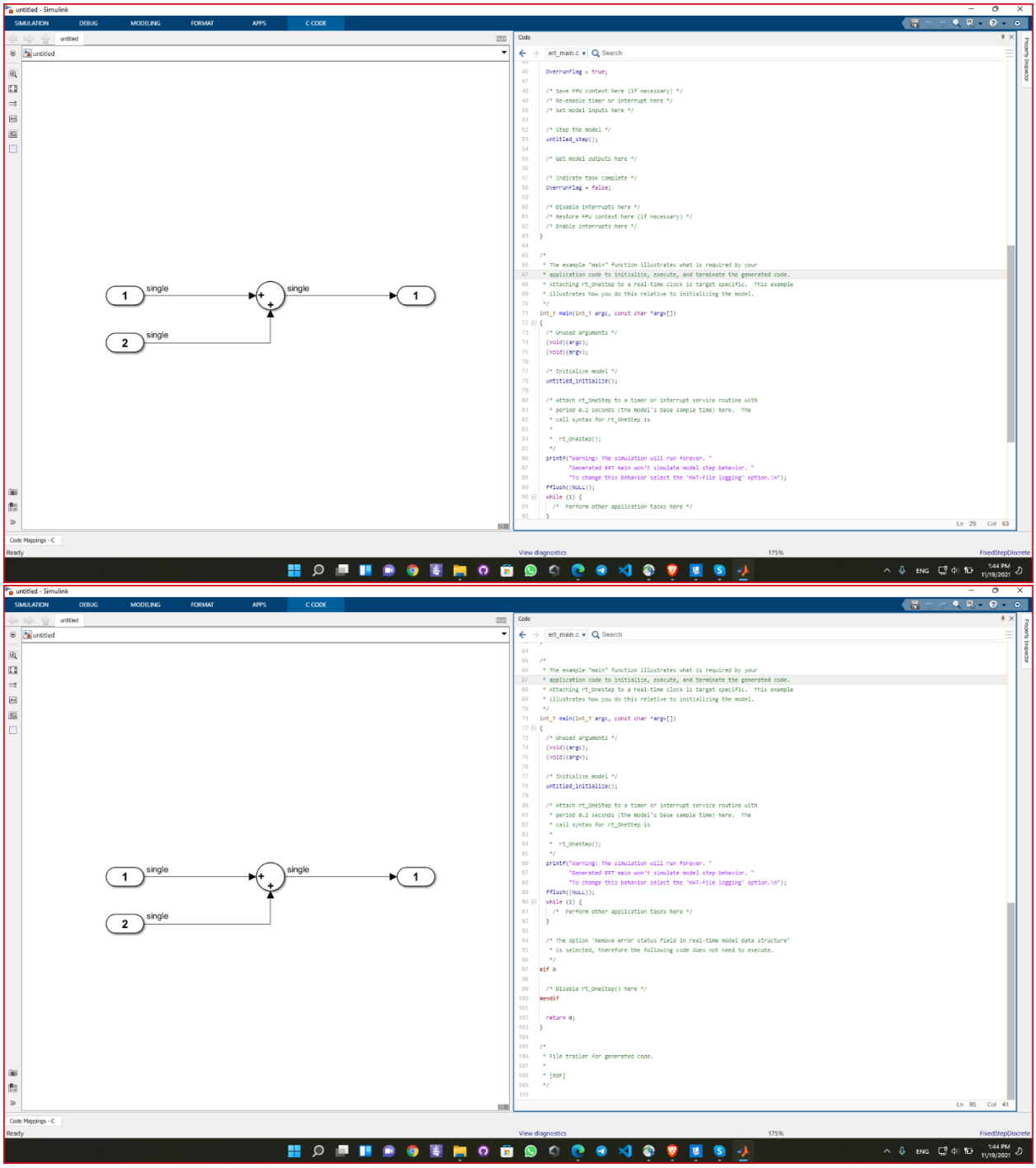
در نهایت هم از آن در ()main استفاده کردیم:

```
int main(){
    while(true){
        b();
        c();
        c();
        c();
        a();
        a();
    }
    return 0;
}
```

3)الف)


Configuration Parameters: untitled/Configuration (Active)
Search
Solver
Data Import/Export
Math and Data Types
Diagnostics
Sample Time
Data Validity
Type Conversion
Connectivity
Compatibility
Model Referencing
Stateflow
Hardware Implementation
Model Referencing
Simulation Target
Code Generation
Coverage
Target selection
System target file: ert.tlc
Browse...
Language: C
Generate GPU code
Description: Embedded Coder
Build process
Generate code only
Package code and artifacts
Zip file name: <empty>
Toolchain settings
Toolchain: Automatically locate an installed toolchain
MinGW64 | gmake (64-bit Windows)
Build configuration: Faster Runs
Toolchain details
Code generation objectives
Prioritized objectives: Execution efficiency, Traceability
Set Objectives...
Check model before generating code: Off
Check Model...
System Target File Browser: untitled
System Target File: Description:
autosar.tlc AUTOSAR
autosar_adaptive.tlc AUTOSAR Adaptive
ert.tlc Embedded Coder
ert.tlc Create Visual C/C++ Solut
ert_shrlib.tlc Embedded Coder (host-base
grt.tlc Generic Real-Time Target
grt.tlc Create Visual C/C++ Solut
realtime.tlc Run on Target Hardware
rsim.tlc Rapid Simulation Target
rtwsfcn.tlc S-Function Target
sldrt.tlc Simulink Desktop Real-Tim
sldrtert.tlc Simulink Desktop Real-Tim
slrealtime.tlc Simulink Real-Time
Full Name: D:\Polyspace\R2021a\rtw\c\ert\ert.tlc
OK
Cancel
Help
Apply
OK
Cancel
Help
Apply
ENG
4:52 PM
11/19/2021


untitled - Simulink
SIMULATION
DEBUG
MODELING
FORMAT
APPS
C CODE
untitled
1
single
2
single
single
1
Code
ert_main.c
Search
/*
* File: ert_main.c
*
* Code generated for Simulink model 'untitled'.
*
* Model version : 1.0
* Simulink Coder version : 9.5 (R2021a) 14-Nov-2020
* C/C++ source code generated on : Fri Nov 19 13:07:41 2021
*
* Target selection: ert.tlc
* Embedded hardware selection: Intel->x86-64 (Windows64)
* Code generation objectives:
* 1. Execution efficiency
* 2. Traceability
* Validation result: Not run
*/
#include <stddef.h>
#include <stdio.h> /* This ert_main.c example uses printf/fflush */
#include "untitled.h" /* Model's header file */
#include "rtwtypes.h"
/*
* Associating rt_OneStep with a real-time clock or interrupt service routine
* is what makes the generated code "real-time". The function rt_OneStep is
* always associated with the base rate of the model. Subrates are managed
* by the base rate from inside the generated code. Enabling/disabling
* interrupts and floating point context switches are target specific. This
* example code indicates where these should take place relative to executing
* the generated code step function. Overrun behavior should be tailored to
* your application needs. This example simply sets an error status in the
* real-time model and returns from rt_OneStep.
*/
void rt_OneStep(void);
void rt_OneStep(void)
{
static boolean_T OverrunFlag = false;
/* Disable interrupts here */
/* Check for overrun */
if (OverrunFlag) {
return;
}
OverrunFlag = true;
Ln 19 Col 29
Code Mappings - C
Ready
View diagnostics
175%
FixedStepDiscrete
ENG
1:44 PM
11/19/2021



کد c تولید شده شامل تعدادی فایل سورس و هدر کمکی و یک فایل و هدر اصلی است و سورس ها و هدرهای کمکی برای راه اندازی اولیه مورد استفاده قرار می گیرد و بعنوان بستر اجرای کد اصلی لازم و ضروری است . سورس و هدر اصلی که untitled.c و untitled.h ساخته میشوند شامل توصیف های اصلی مربوط به مدل طراحی شده میباشد. منطق اصلی در تابع untitled_step و untitled_initialize و untitled_terminate پیاده سازی شده است . البته به علت سادگی مدل جمع کننده فقط یک پیاده سازی را شامل میشود. برای اجرای بلادرنگ این کد باید تابع rt_OneStep به یک ساعت بلادرنگ وصل شود یا توسط یک ISR فراخوانی شود .

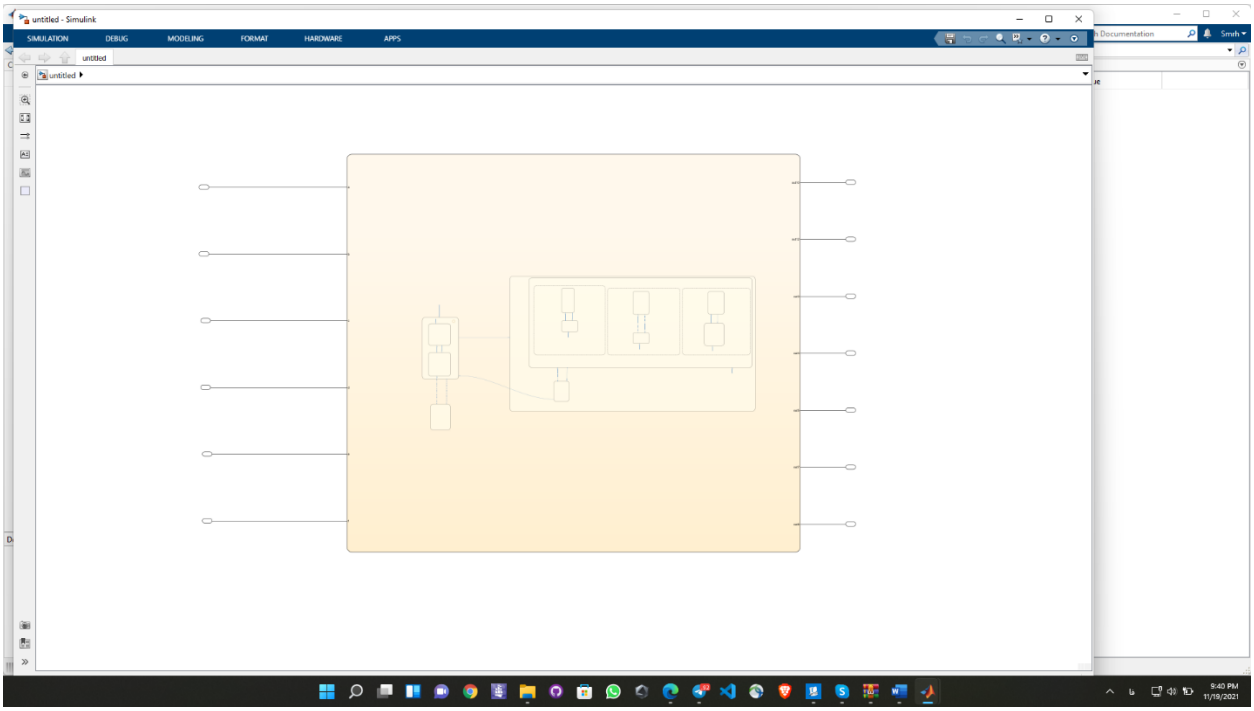
ب) در این سوال مطابق صورت سوال طراحی را طوری تغییر دادیم که 6 حالت ورودی به ازای a تا f داشته باشد و 7 استیت خروجی خواسته شده را در نظر گرفتیم . سپس در تنظیمات مدل system target file را برابر ert.tlc قراردادیم . سپس در بخش set objectives دو اولویت execution efficiency و traceability را انتخاب کردیم . در گام آخر بستر اجرا در قسمت hardware implementation برابر Arduino UNO گذاشتیم و خروجی کد c را بدست آوردیم . دو فایل ert_main.c و untitled.c حاوی پیاده سازی های اصلی هستند . فایل ert_main.c شامل متود main و rt_OneStep برای همگام سازی با کلاک میباشد . فایل untitled.c شامل کد های متناظر با مدل طراحی شده در سیمولینک است .

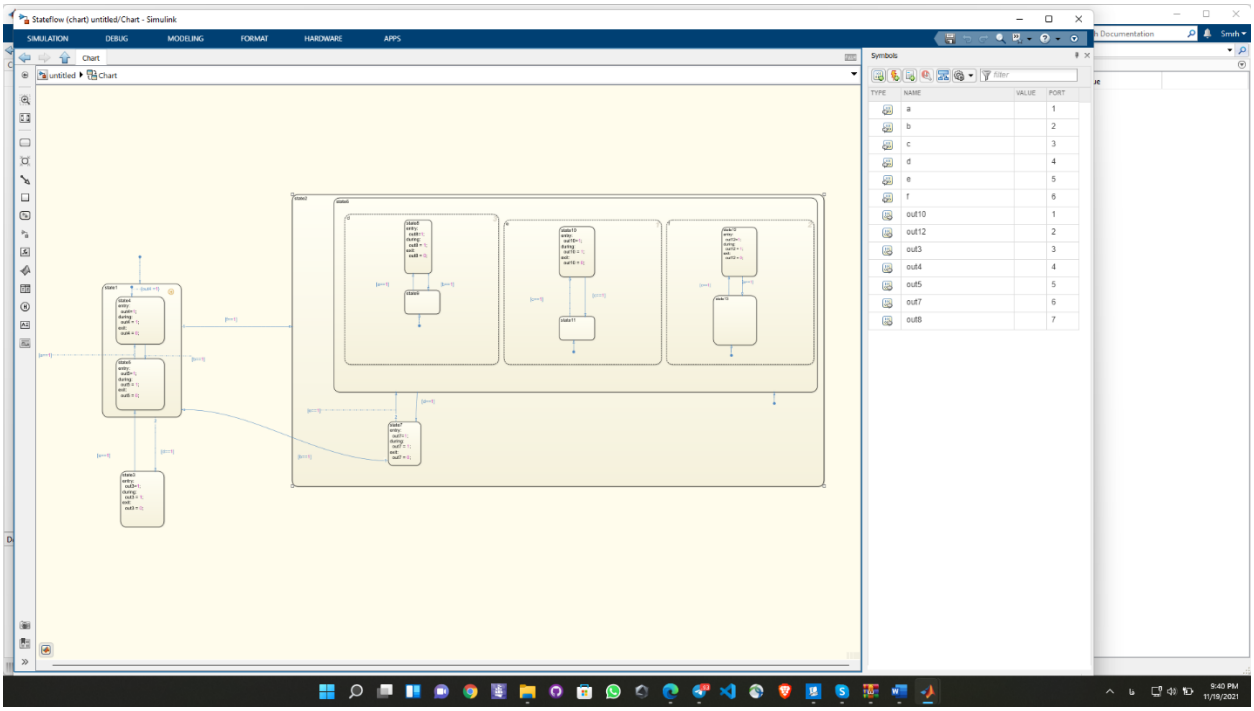
ج) با توجه به مستندات موجود در سایت ، محیط فراخواننده و توابع تولید شده ورودی و خروجی ها را از طریق متغیر های جهانی یا از طریق پارامتر های رسمی (ارگیومنت) ها مبادله میکنند . متلب به این مکانیزم تبادل اطلاعات interfaces میگوید که خود میتواند بر دو نوع کلی reentrant و nonreentrant مدل و پیاده سازی شود . تغییرات متنوعی میتواند در interface های ورودی و خروجی اعمال شود که این تغییرات در یک یا چند دسته از دسته های زیر قرار میگیرد :

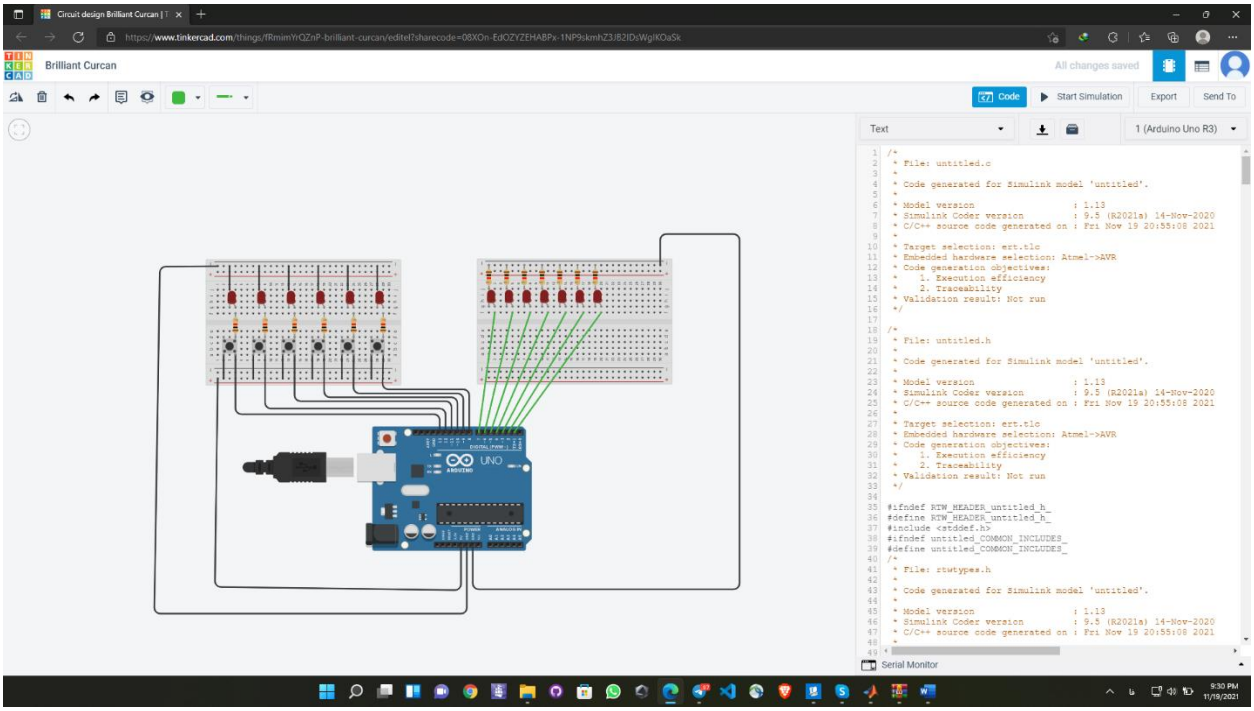
- Control Type Names, Field Names, and Variable Names of Standard I/O Structures (Embedded Coder)
- Control Names of Generated Entry-Point Functions (Embedded Coder)
- Control Data Interface for Nonreentrant Code
  - **Configure Inport or Outport Block as Separate Global Variable**
  - **Configure Generated Code to Read or Write to Global Variables Defined by External Code**
  - **Package Multiple Inputs or Outputs into Custom Structure**
  - **Configure Inport or Outport Block as Function Call (Embedded Coder)**
  - **Pass Inputs and Outputs Through Function Arguments (Embedded Coder)**
  - **Configure Referenced Model Inputs and Outputs as Global Variables (void-void)**
- Control Data Interface for Reentrant Code
- Prevent Unintended Changes to the Interface
- Reduce Number of Arguments by Using Structures
- Control Data Types of Arguments
- Promote Data Item to the Interface

د) تمامی کد های c را در یک کد قراردادیم و در نهایت کد نهایی را در tinkercad استفاده کردیم.

در thinkercad برد Arduino Uno را انتخاب کردیم و باتوجه به آموزش های موجود در یوتیوب مدار مورد نظر را بستیم .

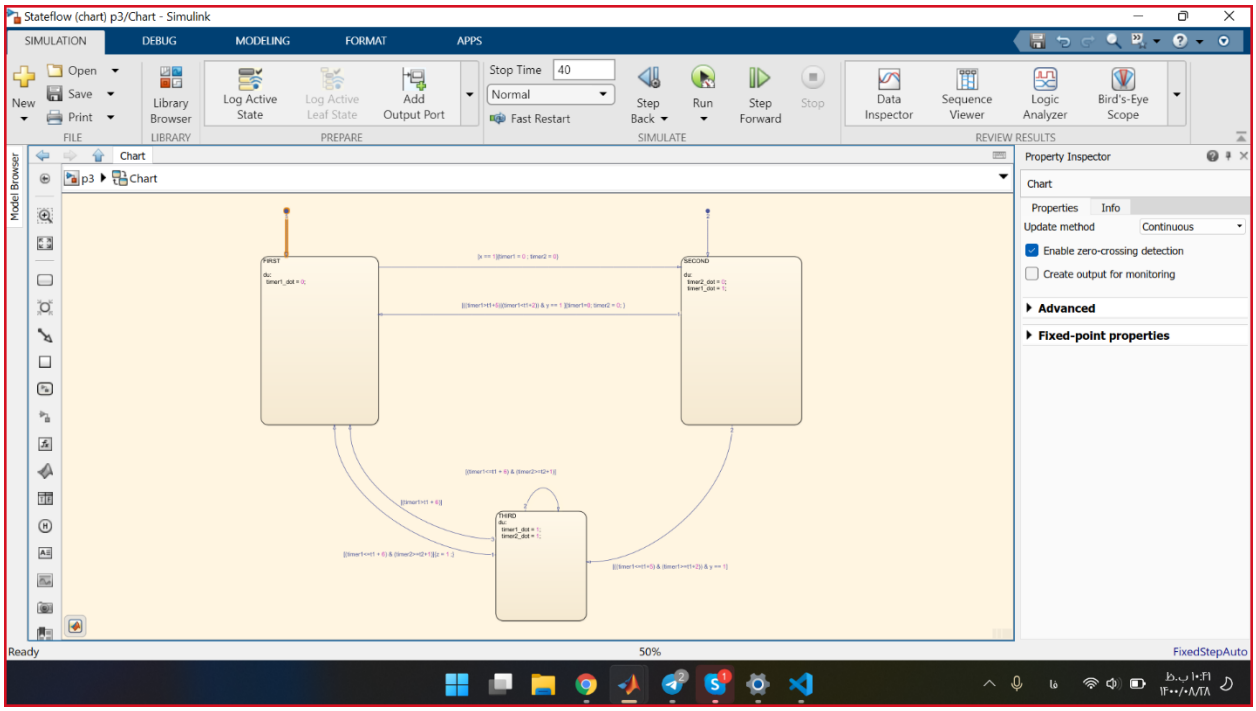






[Circuit design Brilliant Curcan | Tinkercad](#) لینک مدار بسته شده در سایت تینکرکد که در آن مدار به درستی بسته شده و مطابق مدل خواسته شده کار میکند .

تمرین 3 سوال 1



در ابتدا 3 استیت در نظر گرفته شده است . در استیت اول قرار است ورود x کنترل شود .

لحظه ورود x به عنوان لحظه t در نظر گرفته میشود سپس هر دو تایمر ریست شده و به استیت دو میرویم . در استیت دو تایمر 1 شروع به شمارش میکند . اگر در لحظه ای کوچکتر از 2 یا لحظه ای بزرگتر از 5 ، y بیاید به استیت 1 برمیگردیم .

اگر بین زمان 2 تا 5 از لحظه ورود به استیت 2 ، y اتفاق افتد به استیت سه میرویم .

در استیت 3 ، تایمر 2 شروع به کار میکند . از زمانی که تایمر 2 بزرگتر از 1 و تایمر 1 کوچکتر از 6 باشد با توجه به اینکه ماشین non deterministic است یا در بین لحظه های خواسته شده z یک میشود و به استیت 1 برمیگردیم یا اینکه در همان استیت می مانیم در نهایت پس از گذشت زمان خواسته شده به استیت 1 برمیگردیم .