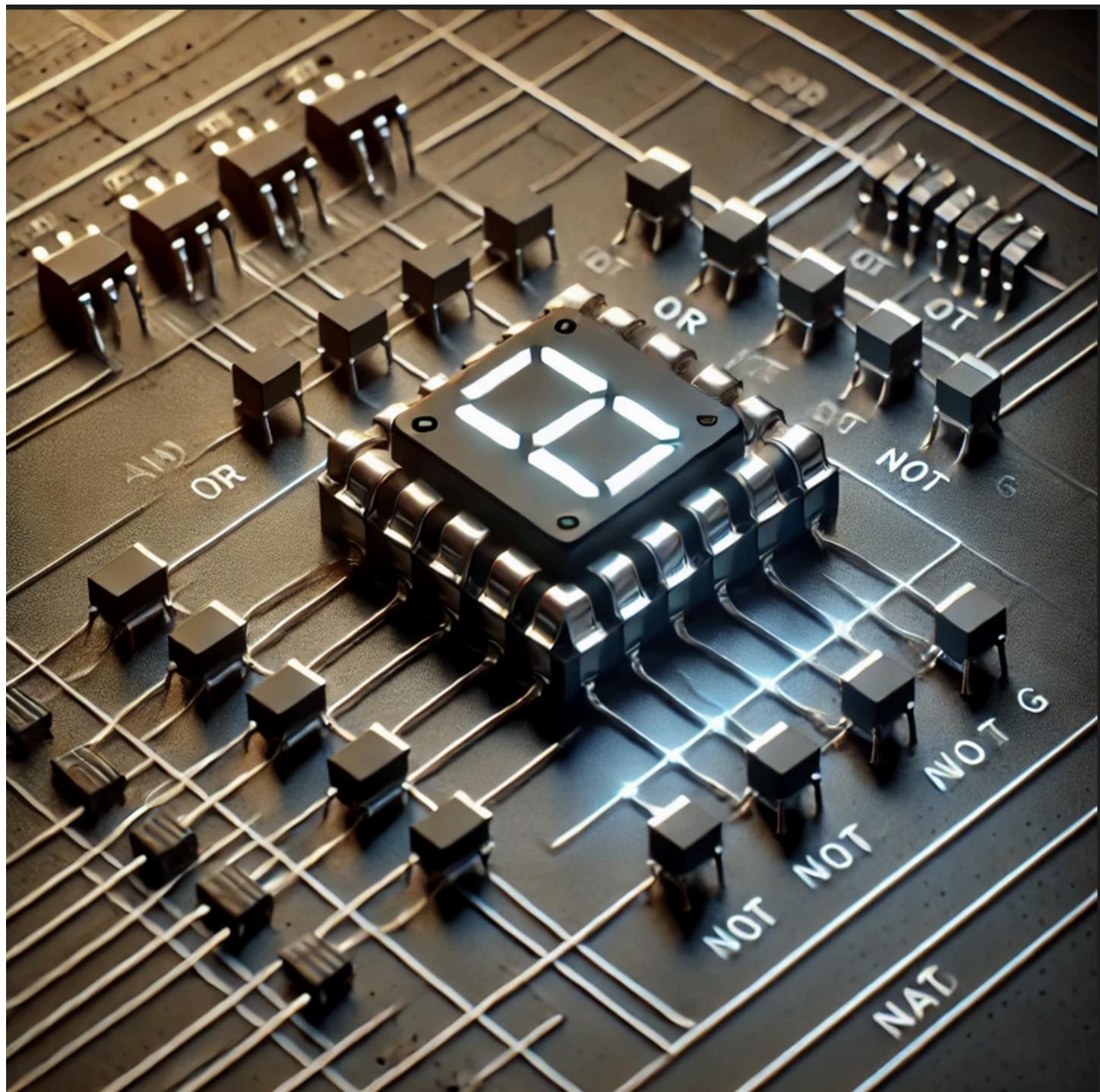


# پروژه کامپیوتري چهارم درس مدار منطقی (RTL)

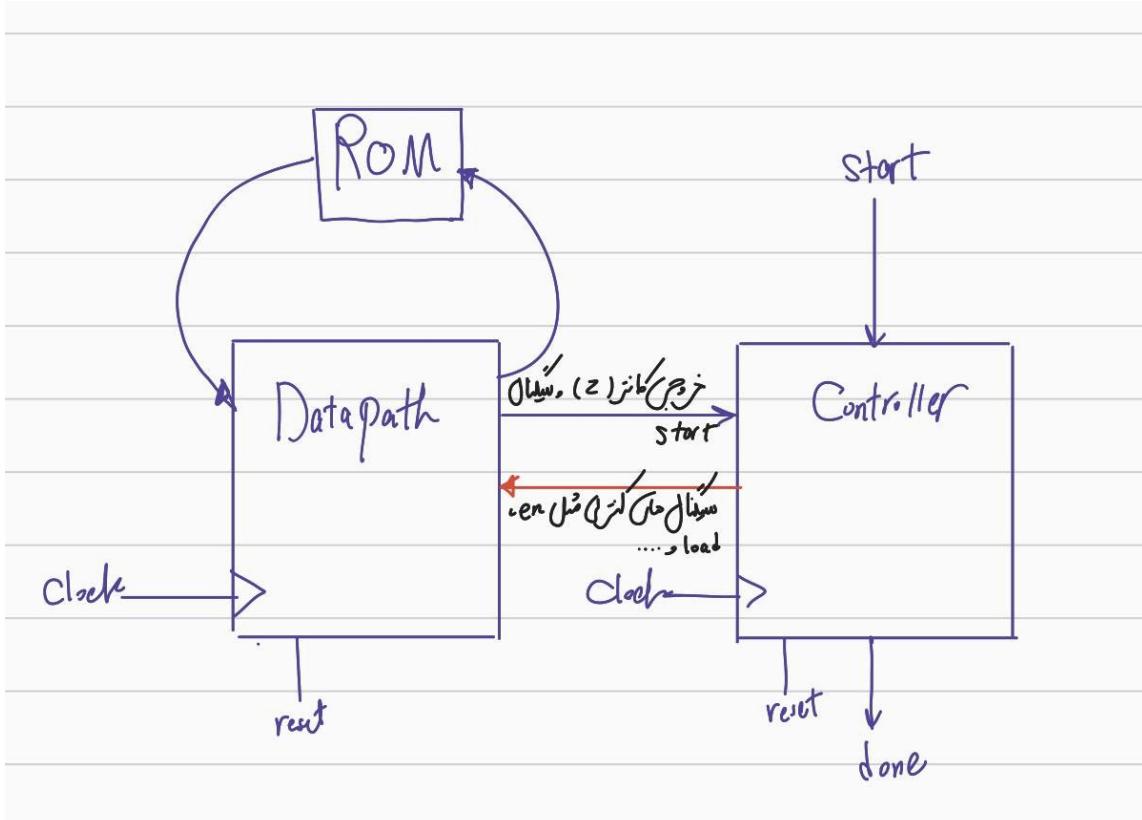
محمد امین رشید 810102454



---

سوال 1:

در این سوال باید دترمینان یک ماتریس 2 در 2 را محاسبه کنیم و نیاز داریم که در هر مرحله محاسباتی را انجام دهیم. ابتدا شماتیک کلی top module را میکشیم:



که در data path و کنترلر درون top module قرار دارند و هم چنین ROM بیرون از این مازول و در تست بنچ قرار میگیرد. data path هم وظیفه محاسبات اصلی و هم وظیفه دادن آدرس به ROM و دریافت فیدبک(data in) را در اختیار دارد.

با این توضیحات شماتیک کلی مسیرداده و ROM را طراحی میکنیم.

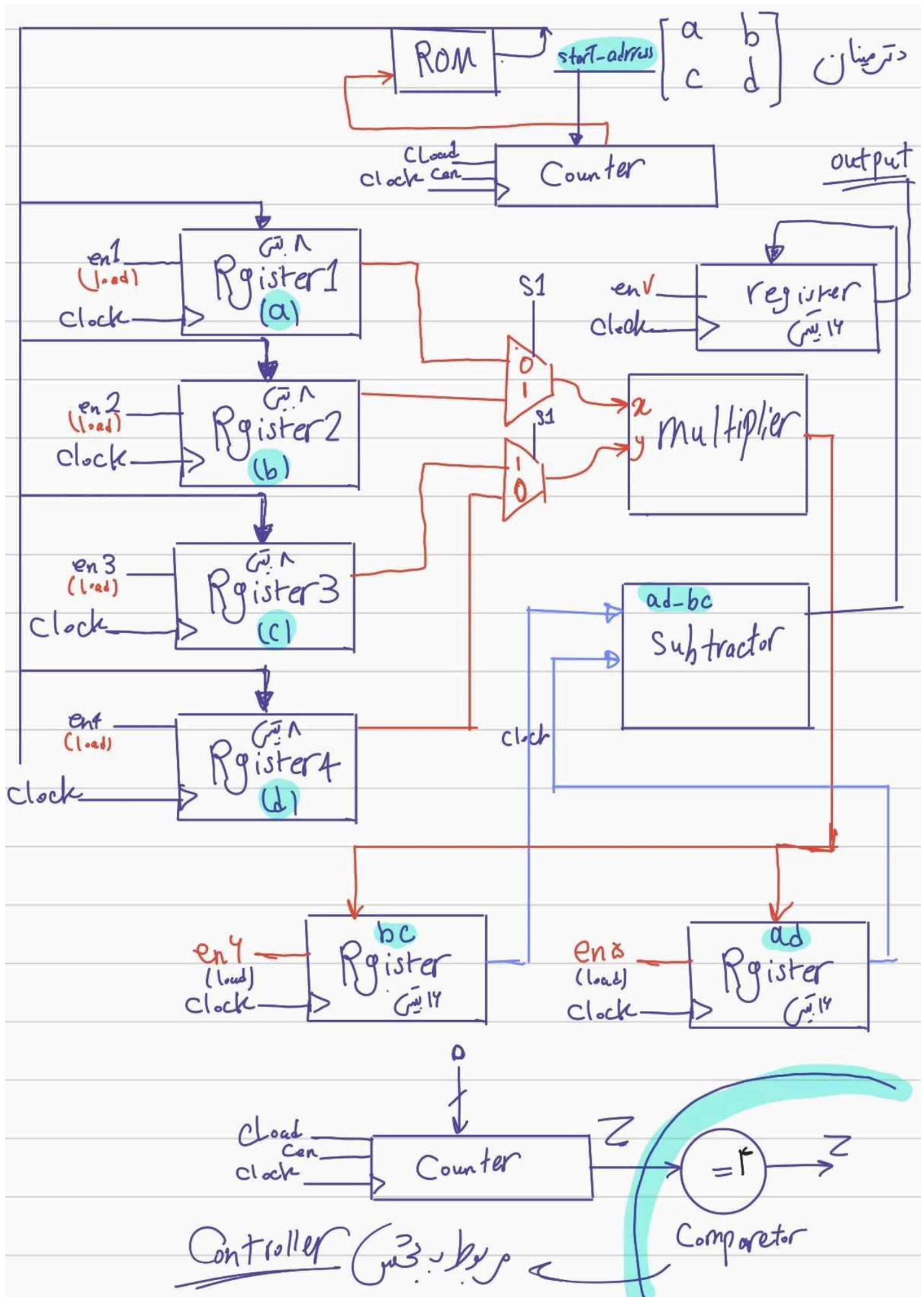
از 4 تا رجیستر 8 بیتی برای ذخیره درایه های ماتریس

از 3 تا رجیستر 16 بیتی برای ذخیره ضرب ترم ها و جواب نهایی

از 2 تا شمارنده: یکی برای شمردن آدرس و دیگری برای شمارش تا 4 (شرط خروج از استیت لودینگ)

از یک ضرب کننده و تفریق کننده برای محاسبات روی درایه های ماتریس

از 2 تا مولتی پلکسر برای سوییچ کردن ورودی ها به دلیل اینکه فقط از یک multiplier میتوانیم استفاده کنیم.



کنترلر وظیفه دارد تا سیگنال های کنترلی را خروجی دهد که ما 8 تا سیگنال کنترلی به نام en که هر رجیستر (7 تا) یک en دارند و ما این را دریک آرایه ذخیره میکنیم و یک سیگنال s1 برای مولتی پلکسر ها.

از 5 استیت استفاده میکنیم و دیاگرام زیر را رسم میکنیم:

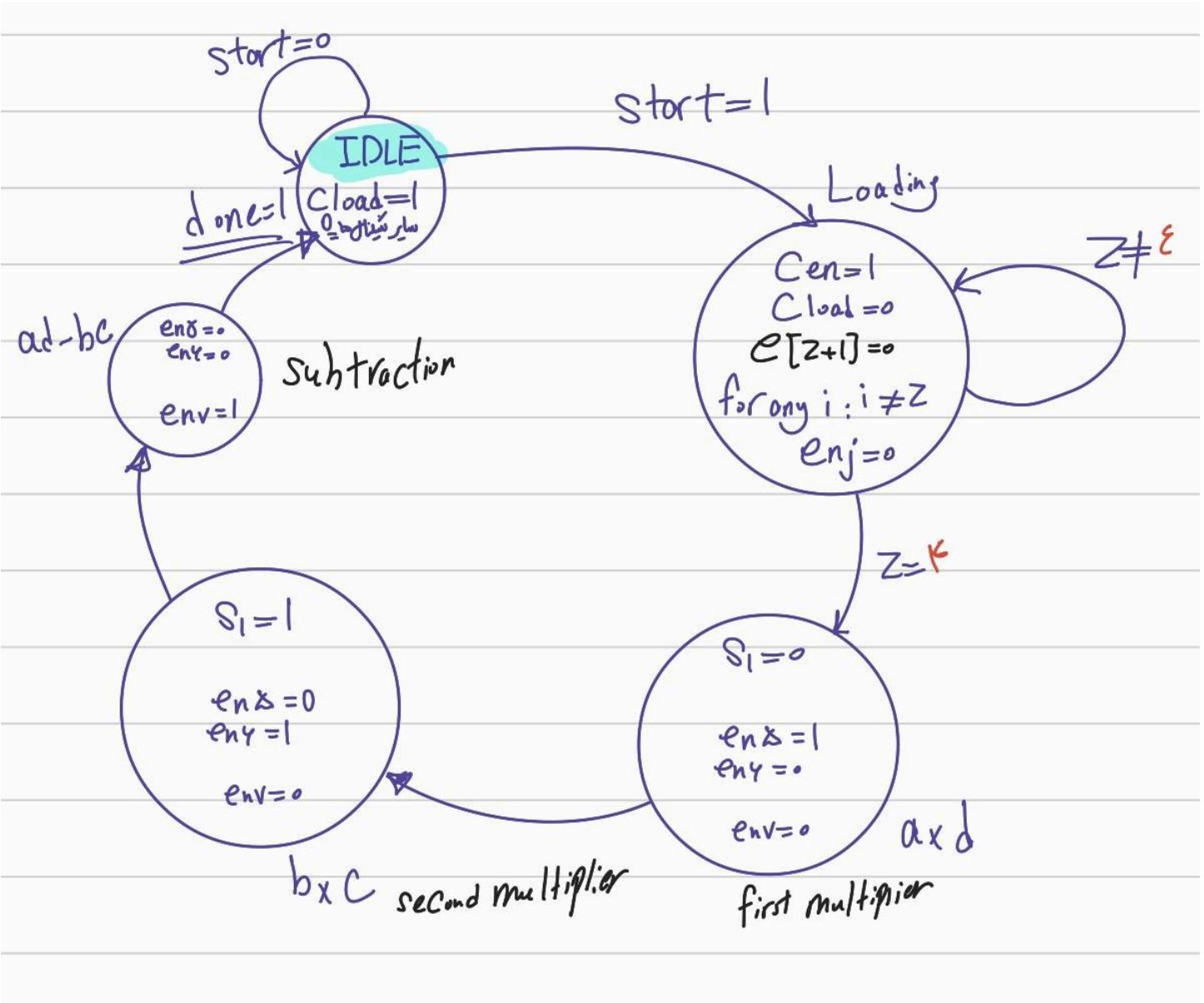
این کنترلر باید خروجی کانتر و start را ورودی بگیرد.

ابتدا در idle منتظر 1 شدن start می ماند.

در loading درایه های ماتریس طی 4 سیکل از رام خوانده و در رجیستر ذخیره میشوند.

در 3 تا استیت بعدی به ترتیب ضرب ها و سپس تفریق انجام میشود و دوباره به حالت idle برمیگردد.

- استیت ماشین از نوع moore است.



اکنون کد نویسی این سوال را شروع میکینم:

top module:

```

1 module top_module1(input [3:0] start_adress, input [7:0] data_in , input clock , input start, input reset,
2   output logic [15:0] out_put, output done, output logic [3:0] adress);
3
4   logic [1:7] en;
5   logic s1;
6   logic [2:0]z;
7   logic cload,cen;
8
9   determinant_calculator2_2_datapath dp(clock,reset,en,data_in,s1,start_adress,cload,cen,adress,z,out_put);
10  determinant_calculator2_2_controller cont(clock,reset,start,z,en,s1,cload,cen,done);
11
12 endmodule

```

## data path:

```
☰ determinant_calculator2_2_datapath.sv
 1 module determinant_calculator2_2_datapath(input clock,input reset,input [1:7] en,input[7:0] data_in,input s1,
 2   input [3:0] start_adress,input cload,input cen ,output logic[3:0] adress,output logic [2:0]z, output logic[15:0] out_put );
 3   logic [7:0] a;
 4   logic [7:0] b;
 5   logic [7:0] c;
 6   logic [7:0] d;
 7
 8   logic [15:0] ad;
 9   logic [15:0] bc;
10   logic [15:0] ad_bc;
11   logic [7:0] x;
12   logic [7:0] y;
13   logic [15:0] multiplier_result;
14
15 counter #(4) countaddresses (clock,cload,cen,start_adress,adress);
16 counter #(3) count0to3(clock,cload,cen,3'b000,z);
17
18 n_register #(8) ra(clock,reset,en[1],data_in,a);
19 n_register #(8) rb(clock,reset,en[2],data_in,b);
20 n_register #(8) rc(clock,reset,en[3],data_in,c);
21 n_register #(8) rd(clock,reset,en[4],data_in,d);
22
23
24 n_register #(16) rad(clock,reset,en[5],multiplier_result,ad);
25 n_register #(16) rbc(clock,reset,en[6],multiplier_result,bc);
26 n_register #(16) rad_bc(clock,reset,en[7],ad_bc,out_put);
27
28 assign x= s1? b:a;
29 assign y= s1? c:d;
30
31
32 multiplier mult(x,y,multiplier_result);
33 subtractor subt(ad,bc,ad_bc);
34
35 endmodule
```

## multiplier & subtractor & counter:

پیاده سازی به صورت behavioral و استفاده از signed برای اینکه فرض کردیم داده ها مکمل 2 هستند.(علامت دار هستند).

```
☰ multiplier.sv
 1 module multiplier(input signed [7:0] x,input signed [7:0] y,output signed [15:0] multiplier_result);
 2   .....
 3   assign multiplier_result =x*y;
 4   endmodule
```

```
☰ subtractor.sv
1 module subtractor(input signed [15:0] a, input signed [15:0] b, output signed [15:0] result);
2
3     assign result = a - b;
4
5 endmodule
6
```

```
☰ counter.sv
1 module counter #(parameter N=4)(input clock,input load, input en,input[N-1:0] par_load,output logic[N-1:0] count);
2
3 always @(posedge clock) begin
4     if(load)begin
5         count<=par_load;
6     end
7     else if(en)begin
8         count<=count+1;
9     end
10
11 end
12 endmodule
13
```

controller:

cen=counter enable & cload=counter load

```
☰ determinant_calculator2_2_controller.sv ☰ determinant_calculator3_3_controller.sv ☰ test_bench22.sv ☰ test_bench3
☰ determinant_calculator2_2_controller.sv
1 module determinant_calculator2_2_controller(input clock,input reset,input start,input [2:0]z,
2 output logic [1:7] en,output logic s1, output logic cload,output logic cen,output logic done );
3 parameter [2:0] idle=0 ,loading=1,first_multiplier=2,second_multiplier=3,subtraction=4;
4 logic [2:0] p_state,n_state;
5 always @(start or z or p_state)begin
6     case(p_state)
7         idle:begin
8             done=1;
9             cload=1;
10            cen=0;
11            en = '{default: 0};
12            s1=0;
13            n_state=(start)?loading: idle;
14        end
15        loading:begin
16            done=0;
17            cen=1;
18            cload=0;
19            en = '{default: 0};
20            en[z+1]=1;
21            n_state=(z==4)?first_multiplier: loading;
22        end |
23         first_multiplier:begin
24             en = '{default: 0};
25             s1=0;
26             en[5]=1;
27
28             n_state=second_multiplier;
29         end
30         second_multiplier:begin
31             en = '{default: 0};
32             s1=1;
33             en[6]=1;
34             n_state=subtraction;
35         end
36         subtraction:begin
37             en = '{default: 0};
```

```
34         n_state=subtraction;
35     end
36     subtraction:begin
37         en = '{default: 0};
38         en[7]=1;
39         n_state=idle;
40     end
41     endcase
42 end
43 always @(`posedge clock or posedge reset)begin
44     if(reset)begin
45         p_state<=idle;
46     end
47     else begin
48
49         p_state<=n_state;
50     end
51 end
52 endmodule
```

test bench:

```
test_bench22.sv
1 `timescale 1ns/1ns
2 module determinant_calculator2_2_tb();
3
4 logic clock,start,done,reset;
5 logic[3:0] start_adress;
6 logic [15:0] out_put;
7 logic [3:0] adress;
8 logic[7:0] data_in;
9
10 top_module1 dc(start_adress,data_in,clock,start,reset,out_put,done,adress);
11 ROM #(8,16) MMRY(adress,data_in);
12
13 initial begin
14     clock=0;
15     start=0;
16     reset=1;
17 end
18 initial begin
19     repeat(60) #5 clock=~clock;
20 end
21 initial begin
22     #2
23     start_adress=1;
24     start=1;
25     reset=0;
26     #10
27     start=0;
28     #99
29     reset=1;
30     #1
31     reset=0;
32     start=1;
33     start_adress=8;
34 end
35 endmodule
```

فایل (ROM): txt

```
ROM_data.txt
1 // memory data file (do not edit the following line - required for mem load use)
2 // instance=/TB1/MMRY/memData
3 // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4 00100001
5 11010110
6 00111010
7 11110011
8 10110011
9 01010011
10 00110011
11 11110011
12 11010011
13 00011011
14 11110011
15 11110011
16 11010011
17 11110011
18 11100100
19 11100100
20
```

در تست بنچ start adress را به ترتیب 1 و 8 امتحان کردم که در حالت

start adress=1:

$$a = 11010110 \rightarrow -42$$

$$b = 00111010 \rightarrow 58$$

$$c = 11110011 \rightarrow -13$$

$$d = 10110011 \rightarrow -77$$

$$\det = ad - bc = \underline{3988}$$

start adress=8:

a=11010011 → -45

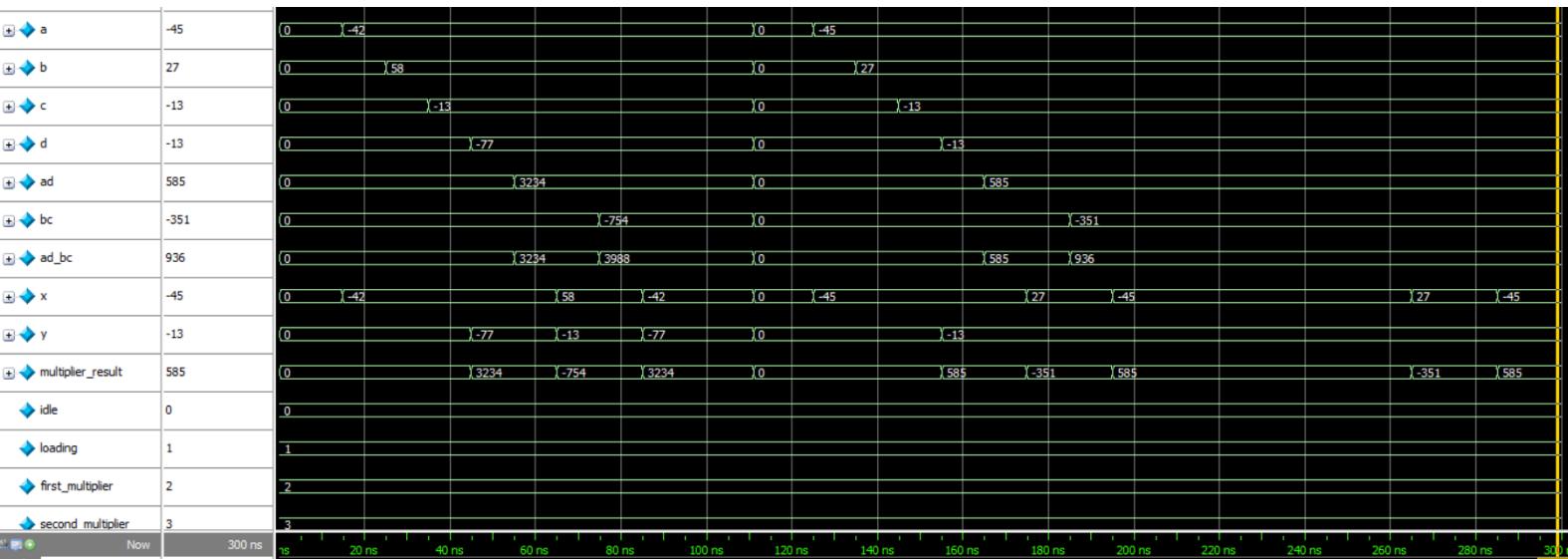
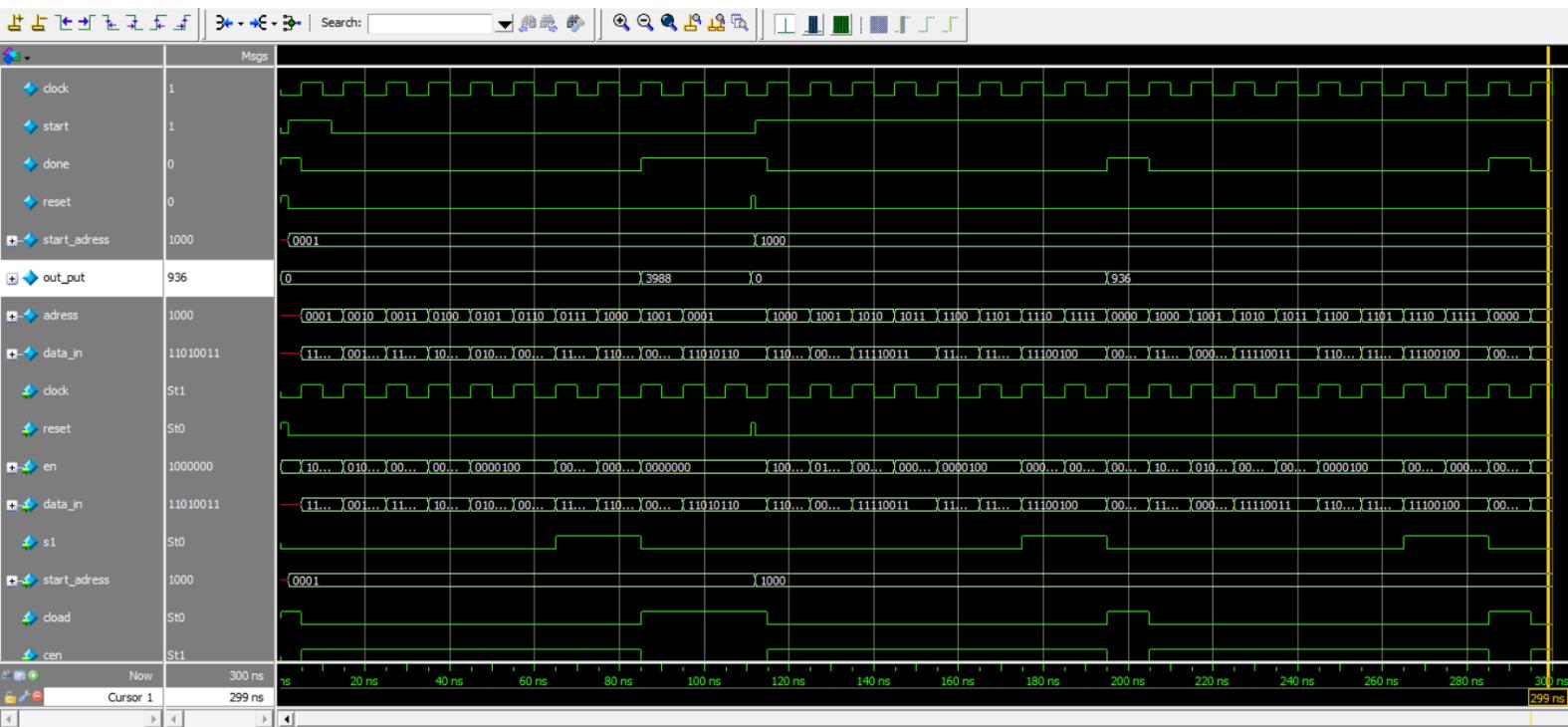
b=00011011 → 27

c=11110011 → -13

d=11110011 → -13

det=ad - bc= 936

:simulate نتیجه



## سوال 2:

در این سوال میخواهیم تنها با یکبار اینستنس گیری از top module سوال قبل دترمینان یک ماتریس 3 در 3 را حساب کنیم.

ابتدا سراغ طراحی مسیرداده میرویم.

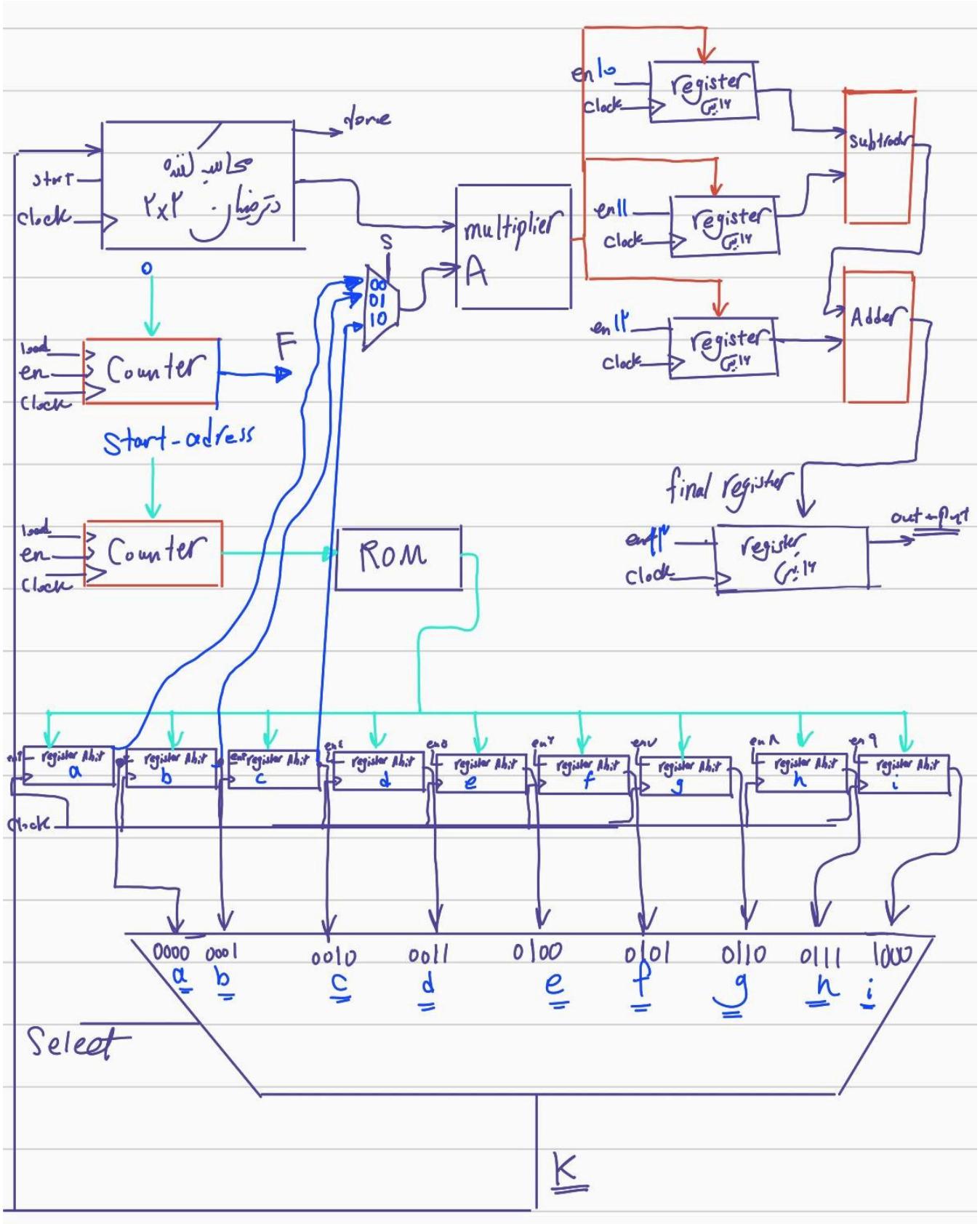
استفاده از 9 تا رجیستر 8 بیتی برای ذخیره h

استفاده از 4 تا رجیستر 16 بیتی برای ذخیره سازی ترم ها و جواب نهاي

استفاده از 1 به 9 بیتی multiplexer برای انتخاب هر رجیستر که به عنوان ورودی به محاسبه کننده دترمینان 2 در 2 بدهیم.(البته می شد اینکار را با مولتی پلکسر 6 به 1 هم انجام داد زیرا هیچ وقت درایه های سطر اول ماتریس 3 در 3 را به دترمینان 2 در 2 نمیدهیم ولی درحالت کلی نیاز است زیرا شاید بخواهیم دترمینان 3 در 3 را به نحو دیگری حساب کنیم و بسط متفاوتی بزنیم.

استفاده از 2 تا شمارنده همانند قسمت قبل برای شمردن آدرس و f که شرط پایان لودینگ را بدهد. در کد به جای f از flag استفاده کردم تا با رجیستر f اشتباه نگیرد.

اضافه کردن adder برای جمع کردن 3 تا ترم.



برای طراحی استیت ماشین در این سوال از روش orthogonal state استفاده کردم یعنی از 2 تا استیت ماشین که در استیت های machine

محاسبه دترمینان 2 در 2 ، استیت ماشین دوم فعال میشود تا select مولتی پلکسر 9 به 1 را تعیین کند و کنترل کند کدام داده روی ماثول برود.

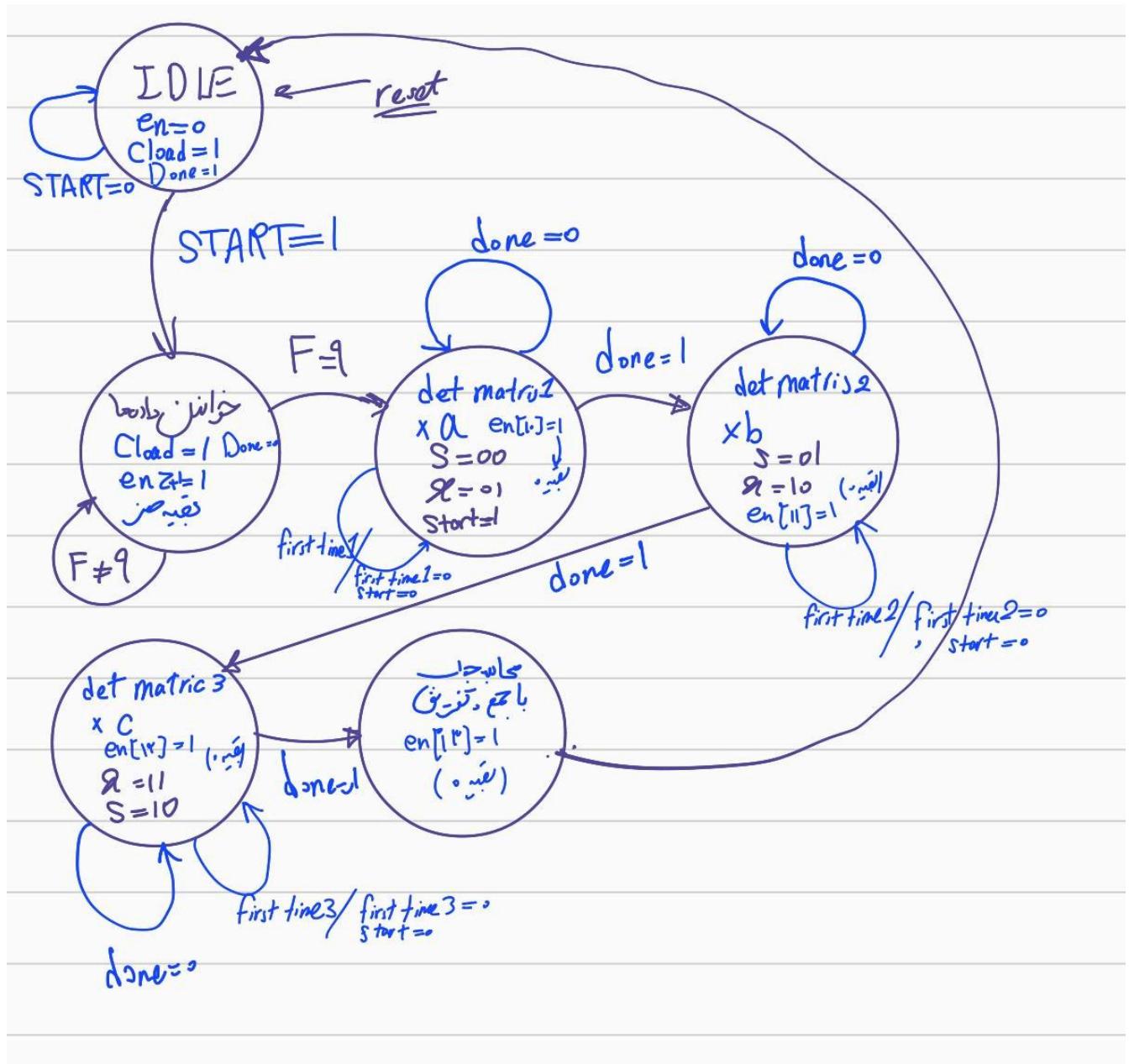
برای اینکار از کانتر به صورت مستقیم نمیشد استفاده کرد زیرا داده هایی که باید بخونیم به ترتیب در رام نیستند!

ابتدا در ساتیت IDLE هستیم و با 1 شدن START به استیت بعد میرویم.

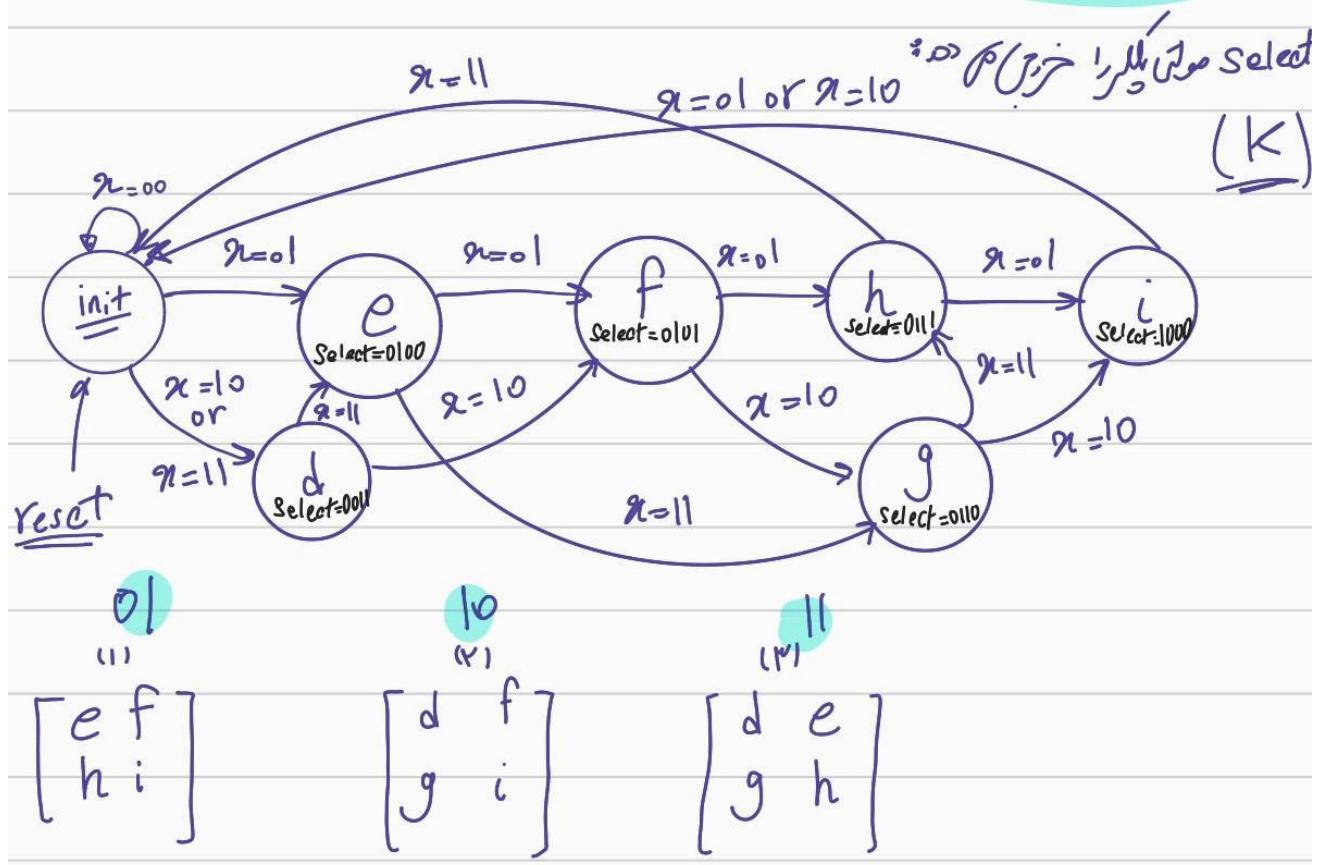
در لودینگ 9 تا دیتا را به ترتیب از رام میخوانم و در 9 تا رجیستر ذخیره میکنیم.

در 3 استیت بعدی هر بار یک دترمینان 2 در 2 را محاسبه میکنیم و در یک عدد ضرب میکنیم و در یک رجیستر 16 بیتی ذخیره میکنیم.

در استیت آخر جمع و تفریق های لازم برای محاسبه نهایی را انجام میدهیم و در یک رجیستر ذخیره میکنیم و سپس به حالت IDLE برمیگردیم.



## orthogonal FSM



:SYSTEM VERILOG کدهای

top module:

```

1 module top_module2 (input [3:0] start_adress,input [7:0]data_in , input clock , input Start,input reset,
2   output logic [15:0] out_put_det3,output logic Done,output logic [3:0] adress);
3
4   logic [1:13] en;
5   logic [1:0]s;
6   logic [3:0] select,flag;
7   logic cload,cen;
8   logic start;
9   logic [15:0] out_put_det2;
10  logic [7:0] k;
11  logic done;
12  determinant_calculator3_3_datapath dp2(clock,reset,out_put_det2 ,en,data_in,s,select,start_adress,cload,cen,out_put_det3,adress,flag,k);
13  determinant_calculator3_3_controller cont2(clock,reset,done,start,start,flag,en,s,select,cload,cen,Done);
14  top_module1_updated dc(k,clock,start,reset,out_put_det2,done);
15
16 endmodule
17
18
19

```

## data path:

```
1 module determinant_calculator3_3_datapath(input clock,input reset,input [15:0] out_put_det2,input [1:13] en,input[7:0] data_in,
2 input[1:0] s,input[3:0] select,input [3:0] start_adress,input cload,input cen,output logic [15:0] out_put_det3,output logic[3:0] adress,
3 output logic [3:0]flag,output logic[7:0] k);
4 logic [7:0] a,b,c,d,e,f,g,h,i;
5 logic[15:0] multiplier_result,term1,term2,term3,sub_result,final_result;
6 logic [7:0] A;
7 counter #(4) countaddresses2 (clock,cload,cen,start_adress,adress);
8 counter #(4) countf(clock,cload,cen,4'b0000,flag);
9 n_register #(8) ra(clock,reset,en[1],data_in,a);
10 n_register #(8) rb(clock,reset,en[2],data_in,b);
11 n_register #(8) rc(clock,reset,en[3],data_in,c);
12 n_register #(8) rd(clock,reset,en[4],data_in,d);
13 n_register #(8) re(clock,reset,en[5],data_in,e);
14 n_register #(8) rf(clock,reset,en[6],data_in,f);
15 n_register #(8) rg(clock,reset,en[7],data_in,g);
16 n_register #(8) rh(clock,reset,en[8],data_in,h);
17 n_register #(8) ri(clock,reset,en[9],data_in,i);
18 assign A= (s== 2'b00) ? a:
19     (s== 2'b01) ?b:
20     (s== 2'b10) ?c:
21     1'b0;
22
23 multiplier2 mult2(out_put_det2,A,multiplier_result);
24 n_register #(16) t1(clock,reset,en[10],multiplier_result,term1);
25 n_register #(16) t2(clock,reset,en[11],multiplier_result,term2);
26 n_register #(16) t3(clock,reset,en[12],multiplier_result,term3);
27 n_register #(16) t4(clock,reset,en[13],final_result,out_put_det3);
28 subtractor sub2(term1,term2,sub_result);
29 adder add(sub_result,term3,final_result);
30
31 always @(*) begin
32 case (select)
33     0: k =a;
34     1: k =b;
35     2: k =c;
36     3: k =d;
37     4: k =e;
38     5: k =f;
39     6: k =g;
40     7: k =h;
41     8: k =i;
42     default: k = 8'b00000000;
43 endcase
44 end
45 endmodule
```

## adder & new multiplier:

ضرب کننده جدید برای ضرب عدد 16 بیتی در 8 بیتی

```
1 module adder(input signed [15:0] a, input signed [15:0] b, output signed [15:0] result);
2
3     assign result = a + b;
4
5 endmodule
```

```
1 module multiplier2 (input signed [15:0] x,input signed [7:0] y,output signed [15:0] multiplier_result);
2     assign multiplier_result =x*y;
3 endmodule
```

ماژول اصلی سوال اول را لازم است تغییر کوچکی دهیم آن هم این است که دیگر این ماژول وظیفه ارتباط با رام را ندارد پس `input` و `output` که از رام به عنوان ورودی میگرفت و آدرسی که به رام میداد را حذف میکنیم.

این ماژول را `updated` نام گذاری کردیم.

top\_module1\_updated.sv

```

1 module top_module1_updated (input [7:0]data_in , input clock , input start,input reset,
2   output logic [15:0] out_put,output logic done);
3
4   logic [1:7] en;
5   logic s1;
6   logic [2:0]z;
7   logic cload,cen;
8
9   determinant_calculator2_2_datapath dp(clock,reset,en,data_in,s1,start_adress,cload,cen,adress,z,out_put);
10  determinant_calculator2_2_controller cont(clock,reset,start,z,en,s1,cload,cen,done);
11
12 endmodule

```

## controller:

determinant\_calculator3\_3\_controller.sv

```

1 module determinant_calculator3_3_controller(input clock,input reset,input done,output logic start,input Start,input [3:0]flag,
2   output logic [1:13] en,output logic[1:0] s,output logic [3:0] select,output logic cload,output logic cen,output logic Done );
3
4   logic first_time_det1, first_time_det2, first_time_det3;
5   parameter [2:0] idle=0 ,loading=1,det1=2, det2=3, det3=4 ,last=5;
6   parameter [2:0] init=0 ,E=1,F=2, H=3, I=4 ,G=5, D=6;
7   logic [2:0] p_state,n_state, p_state_prim , n_state_prim;
8   logic [1:0]x;
9   always @(Start or flag or p_state or done)begin
10    case(p_state)
11      idle:begin
12        Done=1;
13        start=0;
14        cload=1;
15        cen=0;
16        en = '{default: 0};
17        x=2'b00;
18        first_time_det1=1;
19        first_time_det2=1;
20        first_time_det3=1;
21        n_state=(Start)?loading: idle;
22
23    end
24    loading:begin
25      Done=0;
26      cen=1;
27      cload=0;
28      en = '{default: 0};
29      en[flag+1]=1;
30      n_state=(flag==9)? det1: loading;
31    end
32    det1:begin
33      s=2'b00;
34      en = '{default: 0};
35      en[10]=1;
36      if(first_time_det1)begin
37        start=1;

```

```
35
36     en[10]=1;
37     if(first_time_det1)begin
38         start=1;
39         x=2'b01;
40         first_time_det1=0;
41         n_state=det1;
42     end
43     else begin
44         start=0;
45         n_state=done?det2:det1;
46     end
47 end
48
49 det2:begin
50
51     s=2'b01;
52     en = '{default: 0};
53     en[11]=1;
54     if(first_time_det2)begin
55         start=1;
56         first_time_det2=0;
57         n_state=det2;
58         x=2'b10;
59     end
60     else begin
61         start=0;
62         n_state=done?det3:det2;
63     end
64 end
65
66 det3:begin
```

```
67
68     s=2'b10;
69     en = '{default: 0};
70     en[12]=1;
71     if(first_time_det3)begin
72         start=1;
73         first_time_det3=0;
74         n_state=det3;
75         x=2'b11;
76     end
77     else begin
78         start=0;
79         n_state=done?last:det3;
80     end
81
82
83     end
84     last:begin
85         en = '{default: 0};
86         en[13]=1;
87         start=0;
88
89         n_state=idle;
90     end
91 endcase
92 end
93 always @(x or p_state_prim)begin
94     case(p_state_prim)
95
96     init:begin
97         n_state_prim=(x==2'b01)?E:
98             (x==2'b10 )?D:
```

```
99      |     (x==2'b11 )?D:
100     |     init;
101    end
102    E:begin
103      |     select=4'b0100;
104      |     n_state_prim=(x==2'b01)?F:
105      |     (x==2'b11 )?G:
106      |     init;
107    end
108    F:begin
109      |     select=4'b0101;
110      |     n_state_prim=(x==2'b01)?H:
111      |     (x==2'b10 )?G:
112      |     init;
113
114  end
115  H:begin
116    |     select=4'b0111;
117    n_state_prim=(x==2'b01)?I:
118    |     (x==2'b11 )?init:
119    |     init;
120  end
121  I:begin
122    |     select=4'b1000;
123    n_state_prim=(x==2'b10)?init:
124    |     (x==2'b01 )?init:
125    |     init;
126  end
127  G:begin
128    |     select=4'b0110;
129    n_state_prim=(x==2'b10)?I:
130    |     (x==2'b11 )?H:
131    |     init;
```

```

131         init;
132     end
133     D:begin
134         select=4'b0011;
135         n_state_prim=(x==2'b10)?F:
136             (x==2'b11 )?E:
137                 init;
138             end
139         endcase
140     end
141     always @(* posedge clock or posedge reset)begin
142         if(reset)begin
143             p_state<=idle;
144             p_state_prim<=init;
145         end
146         else begin
147             p_state<=n_state;
148             p_state_prim<=n_state_prim;
149             x <= (n_state_prim == init) ? 2'b00 : x;
150         end
151     end
152 endmodule

```

در پیاده سازی این کنترلر ، از 3 تا متغیر بولین طور first time استفاده شده است زیرا شرط خروج از استیت های دوم و سوم و چهارم 1 شدن سیگنال done ماذول سوال 1 است و این باعث میشود بلافاصله به استیت بعد برویم. برای هندل کردن این موضوع از این 3 تا متغیر استفاده میکنیم.

هم چنین موضوع دیگری که هست این است که وقتی از استیت ماشین دوم استفاده میکنیم، پس از خواندن 4 تا درایه ماتریس 2 در 2 باید به init برگردد و x=00 بشود تا در همان init تا زمان نرفتن به استیت بعد در استیت ماشین اصلی بماند. و به همین دلیل x را در استیت های محاسبه دترمینان برای بار اول مقدار دهی میکنیم و در always بلاکی که در انتهای داریم و با کلاک سینک است، x را با توجه به اینکه اسیتت بعد آیا init است ، مقدار دهی

میکنیم تا با خیال راحت بدانیم اگر استیت ماشین دوم در init است  $x$  هم 00 است.

test bench:

```
test_bench33.sv
1 `timescale 1ns/1ns
2 module determinant_calculator3_3_tb();
3
4 logic clock,Start,Done,reset;
5 logic[3:0] start_adress;
6 logic [15:0] out_put_det3;
7 logic [3:0] adress;
8 logic[7:0] data_in;
9
10 top_module2 dc(start_adress,data_in,clock,start,reset,out_put_det3,Done,adress);
11 ROM #(8,16) MMRY(adress,data_in);
12
13 initial begin
14     clock=0;
15     Start=0;
16     reset=1;
17 end
18 initial begin
19     repeat(200) #5 clock=~clock;
20 end
21
22 initial begin
23 #2
24     start_adress=1;
25     Start=1;
26     reset=0;
27     #10
28     Start=0;
29     #500
30     reset=1;
31     #1
32     reset=0;
33     start_adress=0;
34     Start=1;
35
36 end
```

به دلیل جلوگیری از رخ دادن overflow فایل ROM را اندکی تغییر میدهیم:

2.txt

```
1 // memory data file (do not edit the following line - required for mem load use)
2 // instance=/TB1/MMRY/memData
3 // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4 00000001
5 00000001
6 00000011
7 00000010
8 10110011
9 01010011
10 00110011
11 11110011
12 11010011
13 00011011
14 11110011
15 11110011
16 11010011
17 11110011
18 11100100
19 11100100
20
21
22
```

در تست بنچ ابتدا start adress را 1 میدهیم و سپس پس از 500 نانو ثانیه آن را به 0 تغییر میدهیم:

if start adress=1:

a=00000001 → 1

b=00000011 → 3

c=00000010 → 2

d=10110011 → -77

e=01010011 → 83

f=00110011 → 51

g=11110011 → -13

h=11010011 → -45

i=00011011 → 27

$$\det \begin{pmatrix} 1 & 3 & 2 \\ -77 & 83 & 51 \\ -13 & -45 & 27 \end{pmatrix}$$

Solution

17872

if start address=0:

a=00000001 → 1

b=00000001 → 1

c=00000011 → 3

d=00000010 → 2

e=10110011 → -77

f=01010011 → 83

g=00110011 → 51

h=11110011 → -13

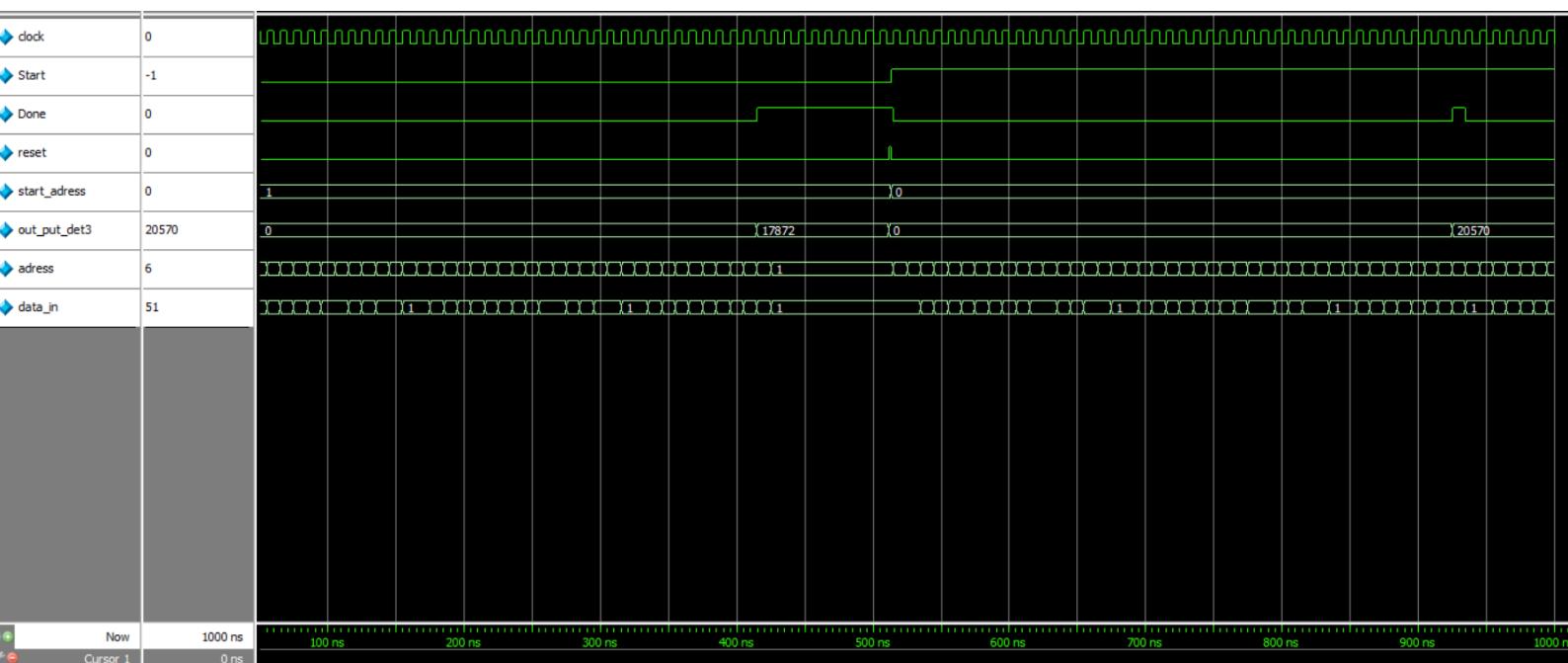
i=11010011 → -45

$$\det \begin{pmatrix} 1 & 1 & 3 \\ 2 & -77 & 83 \\ 51 & -13 & -45 \end{pmatrix}$$

Solution

20570

simulate:



The end.