

تمرین کامپیوتری امتیازی مدار منطقی

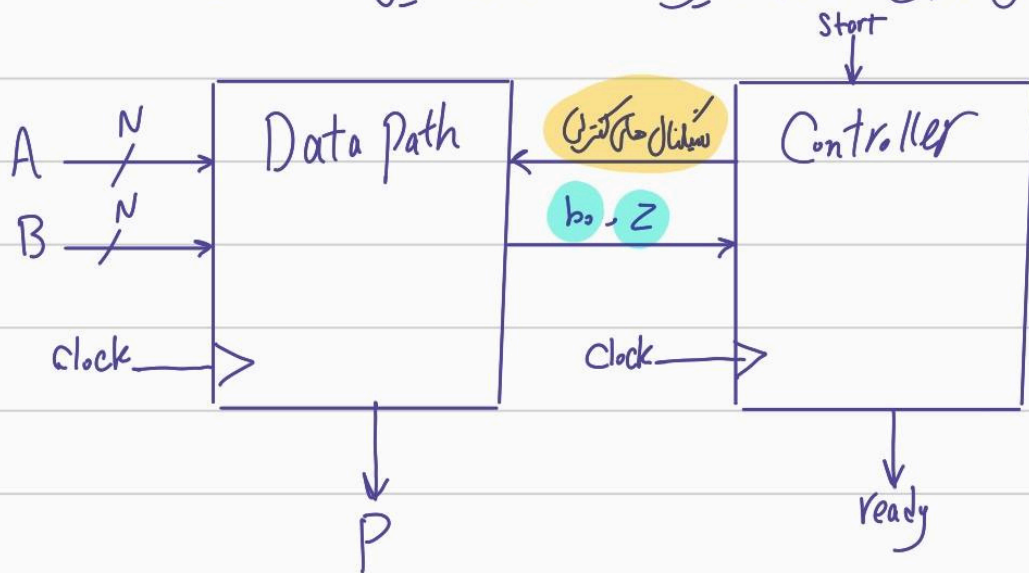
محمد امین رشید 810102454



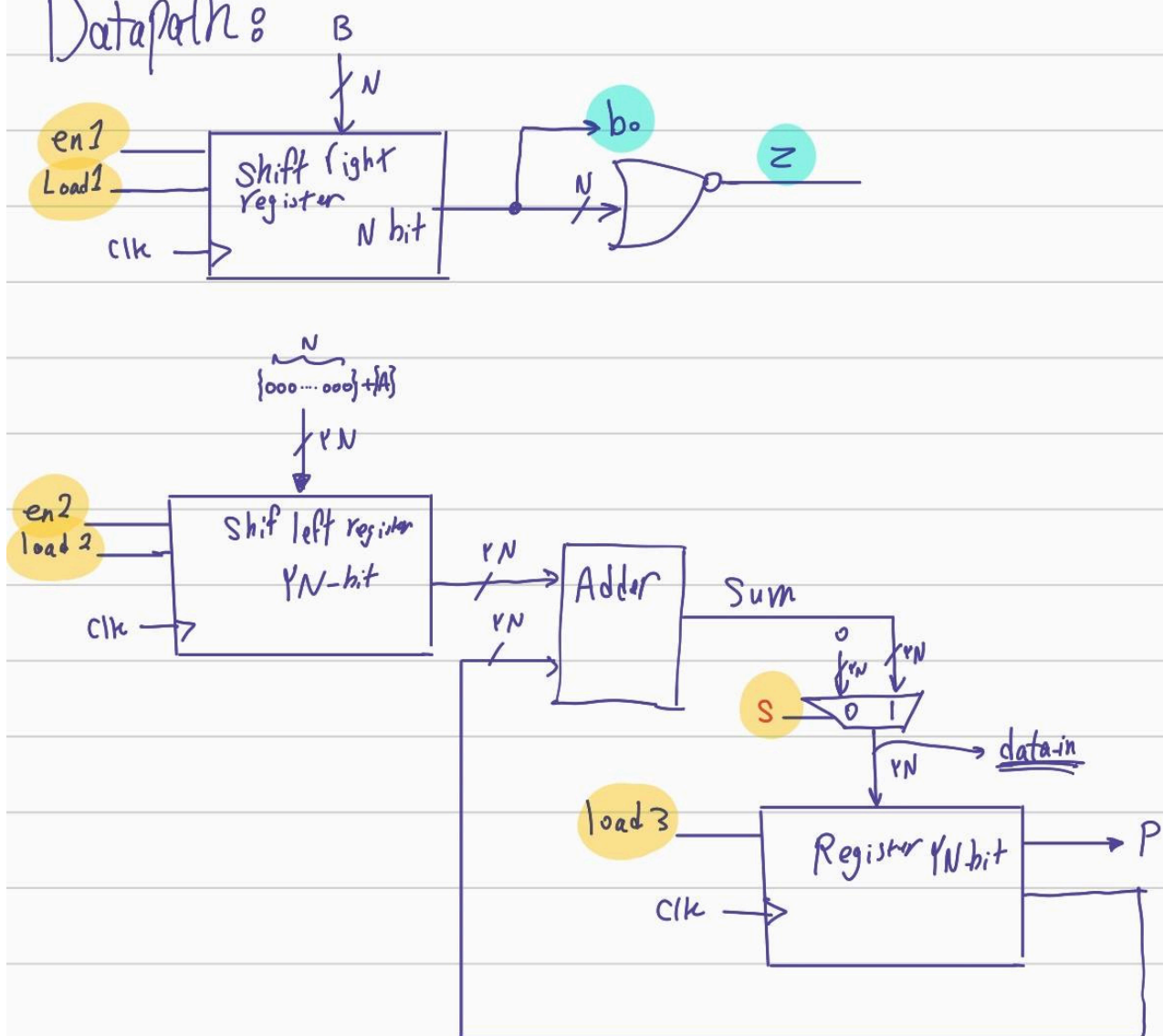
ابتدا یک ضرب کننده ترتیبی طراحی میکنیم که با عمل shift کردن و جمه کردن partial sum ها به دست می آید.

قرار است دو عدد N بیتی را با Shift-Add، با هم ضرب کنیم و یک خروجی $2N$ بیتی بدهیم.

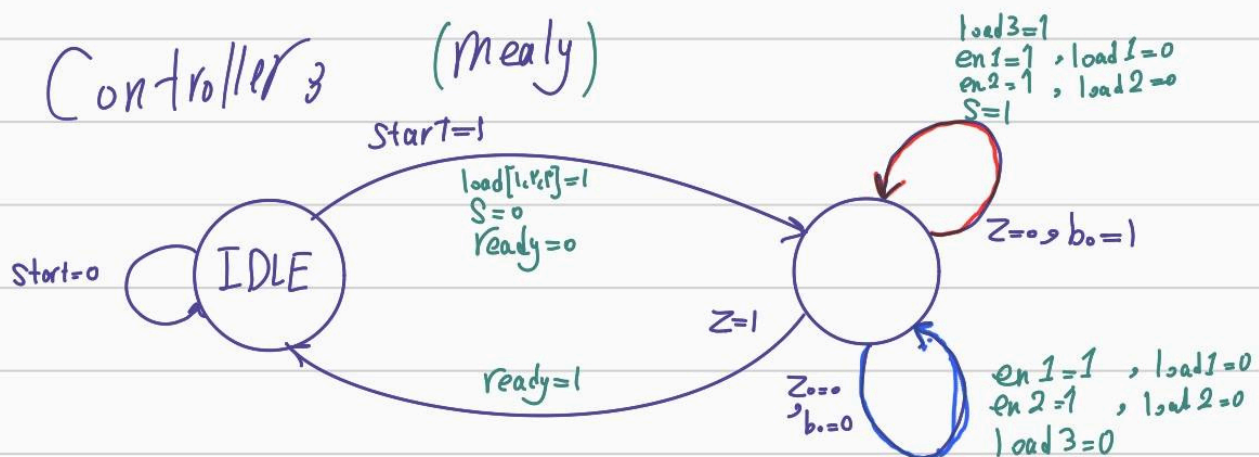
ساختار کلی ما در این مورد از زیر Data Path و Controller تشکیل خواهد شد.



Datapath:



Controller (mealy)



خروجی ها باید سبز نشان داده شده اند.

کد های سیستم وریلاگ:

```

1  module shift_add_multiplier #(parameter N=8)(input logic clock, start, input logic [N-1:0] A,B, output logic [2*N-1:0] P, output logic ready);
2
3      logic [1:2] en;
4      logic [1:3] load;
5      logic z,b0,s;
6
7      data_path_multiplier #(N) dp(clock,A,B,en,load,s,b0,z,P);
8      controller_multiplier cont(clock,start,z,b0,en,load,ready,s);
9
10 endmodule

```

```

1  module data_path_multiplier #(parameter N=8)(input logic clock, input logic [N-1:0] A,B,
2  input logic [1:2] en, input logic [1:3] load, input logic s, output logic b0,z, output logic [2*N-1:0] P);
3
4  logic [N-1:0] B_out;
5  logic [2*N-1:0] A_out,sum,data_in;
6
7
8  shift_register #(N) sr1(clock,1,load[1],en[1],B,B_out);
9  shift_register #(2*N) sr2(clock,0,load[2],en[2],{N{1'b0}},A,A_out);
10
11 assign b0=B_out[0];
12 assign z= ~|B_out;
13 assign sum=A_out + P;
14 assign data_in=s?sum:{2*N{1'b0}};
15
16 register #(2*N) r1(clock,load[3],data_in,P);
17
18
19 endmodule
20
21
22

```

controller_multiplier.sv

```
1 module controller_multiplier (input logic clock,start,z,b0,output logic[1:2] en,output logic [1:3] load ,output logic ready,s);
2 parameter idle=0,active=1;
3 logic p_state,n_state;
4 always @(start or z or p_state or b0)begin
5     case(p_state)
6     idle:begin
7         en ='{default:0};
8         load ='{default:0};
9         if(start)begin
10             load ='{default:1};
11             s=0;
12             ready=0;
13             n_state=active;
14         end
15     else begin
16         n_state=idle;
17     end
18 end
19 active:begin
20     if(z)begin
21         ready=1;
22         en ='{default: 0};
23         load='{default: 0};
24         n_state=idle;
25     end
26     else if ((z==0)&&(b0==0))begin
27         en ='{default:1};
28         load ='{default:0};
29
30         n_state=active;
31     end
32     else begin
33         s=1;
34         load ='{default:0};
35         load[3]=1;
36     end
37 end
```

controller_multiplier.sv

```
4 always @(start or z or p_state or b0)begin
5     case(p_state)
18     end
19     active:begin
27         else if ((z==0)&&(b0==0))begin
30
31         n_state=active;
32
33     end
34     else begin
35         s=1;
36         load ='{default:0};
37         load[3]=1;
38         en ='{default:1};
39         n_state=active;
40     end
41 end
42 default: begin
43     en ='{default: 0};
44     load='{default: 0};
45     ready=0;
46     s=0;
47     n_state = idle;
48 end
49 endcase
50 end
51
52
53 always @(posedge clock)begin
54     p_state<=n_state;
55 end
56 endmodule
57
```

```

shift_register.sv
1  module shift_register #(parameter N=8)(input logic clock,input logic right_shift,input logic load,en,input logic [N-1:0] data_in
2  ,output logic [N-1:0] data_out);
3
4  always @(posedge clock) begin
5      if (load) begin
6          data_out <= data_in;
7      end else begin
8          if(en)begin
9              if (right_shift) begin
10                 data_out <= {1'b0, data_out[N-1:1]};
11             end else begin
12                 data_out <= {data_out[N-2:0],1'b0};
13             end
14         end
15     end
16 end
17 endmodule
18

```

```

register.sv
1  module register #(parameter N=8)(input logic clock,input logic load,input logic [N-1:0] data_in,output logic [N-1:0] data_out );
2
3  always @(posedge clock ) begin
4
5      if(load)begin
6          data_out<= data_in;
7      end
8
9  end
10 endmodule
11
12

```

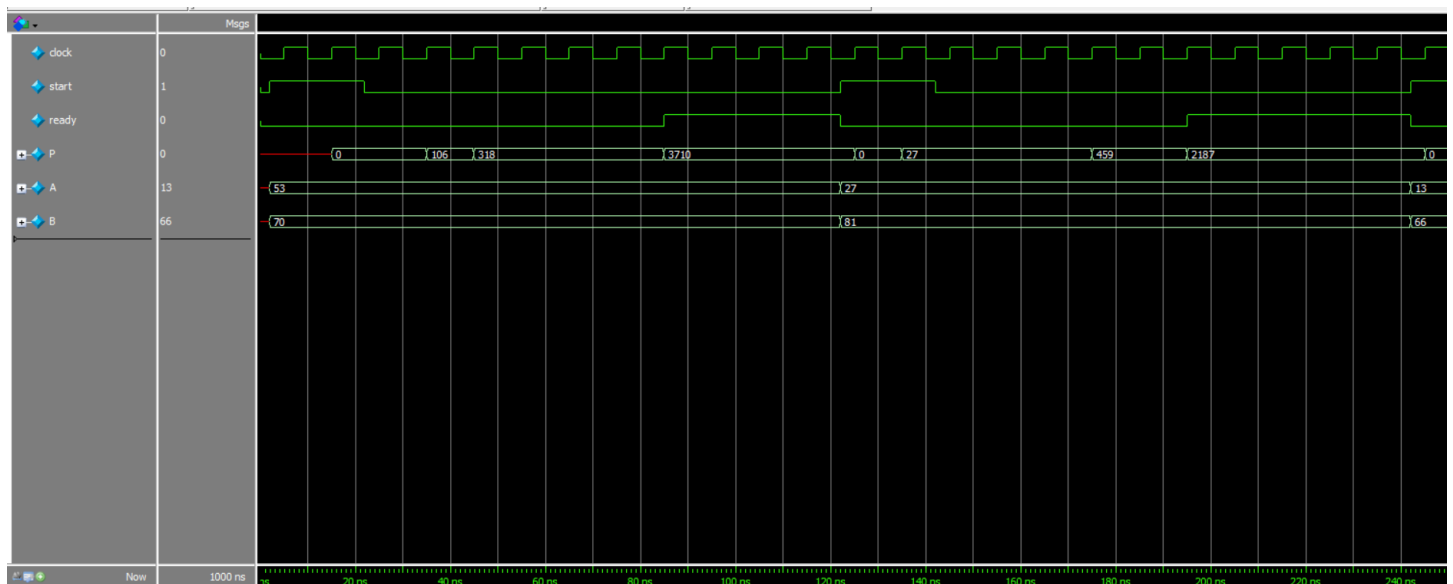
تست بنچ برای عرض 8 بیت:

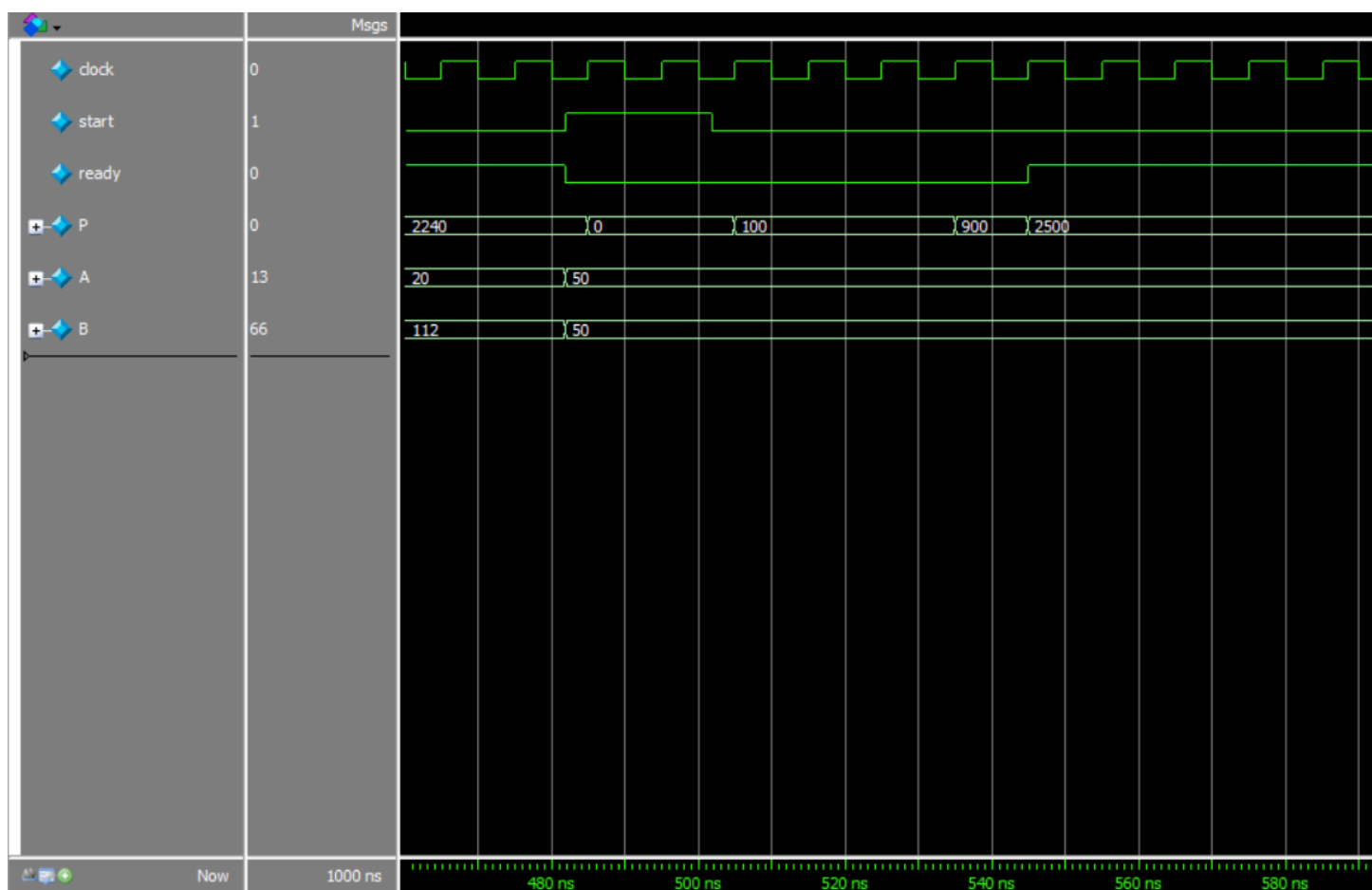
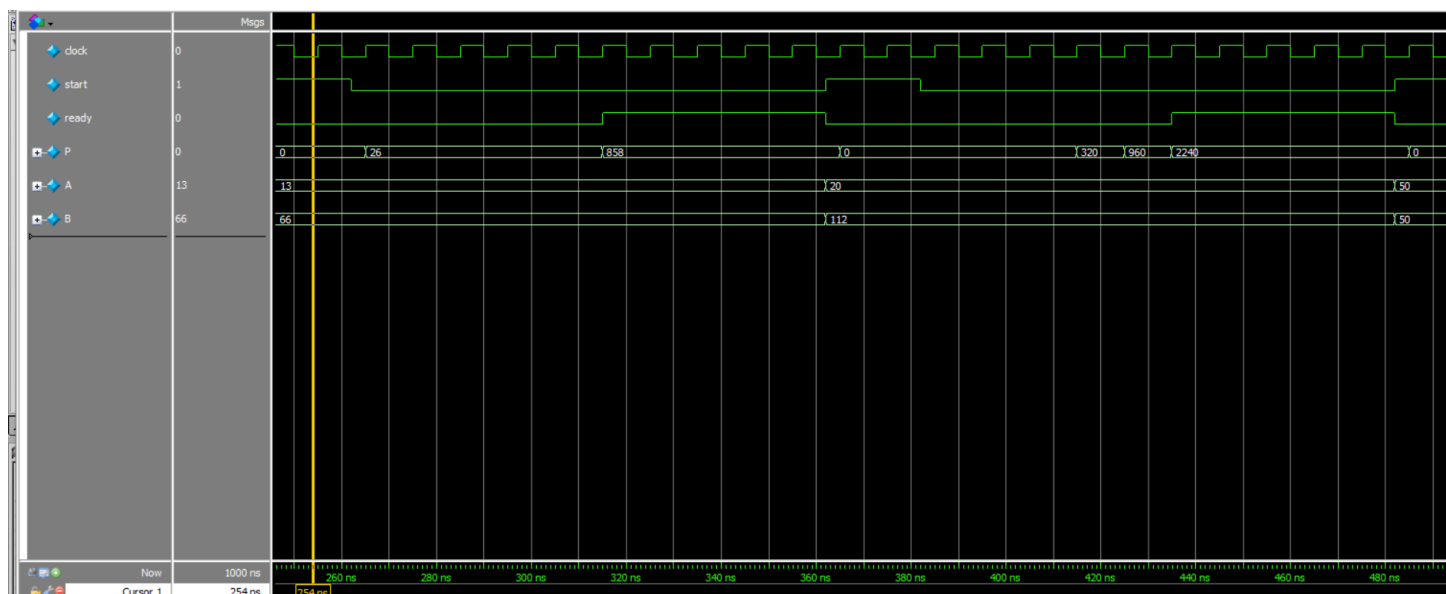
```

1 multiplier_tb.sv
2 module multiplier_tb();
3     logic[7:0] A,B;
4     shift_add_multiplier #(8) dc(clock,start,A,B,P,ready);
5     initial begin
6         clock=0;
7         start=0;
8     end
9     initial begin
10        repeat(200) #5 clock=~clock;
11    end
12    initial begin
13        #2
14        start=1;
15        A=8'b00110101;
16        B=8'b01000110;
17        #20
18        start=0;
19        #100
20        start=1;
21        A=27;
22        B=81;
23        #20
24        start=0;
25        #100
26        start=1;
27        A=13;
28        B=66;
29        #20
30        start=0;
31        #100
32        start=1;
33        A=20;
34        B=112;
35        #20
36        start=0;
37        #100
38        start=1;
39        A=50;
40        B=50;
41        #20
42        start=0;
43    end
44 endmodule

```

نتایج شبیه سازی:





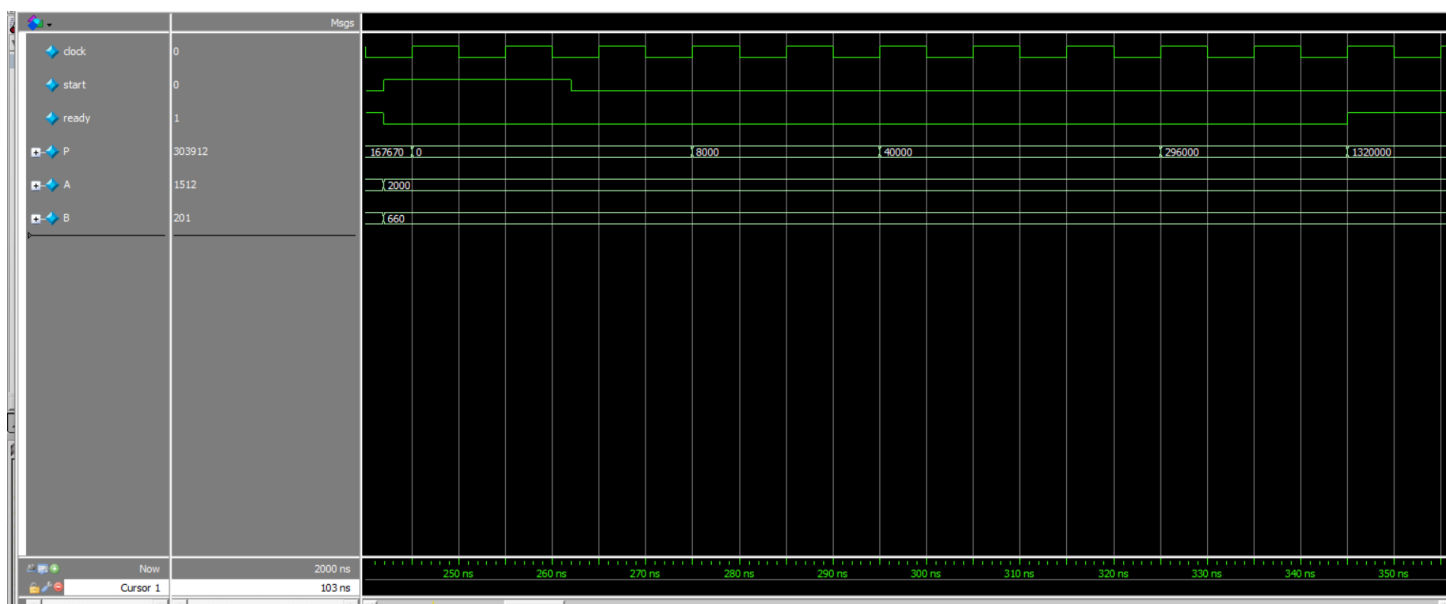
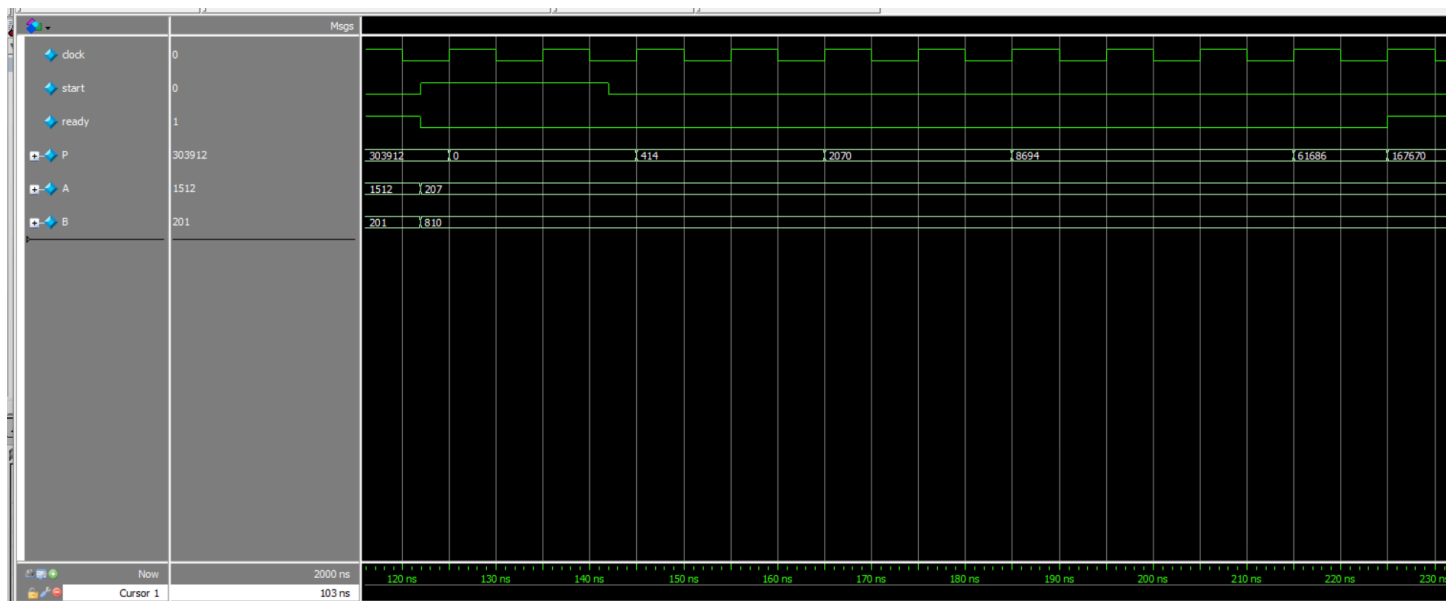
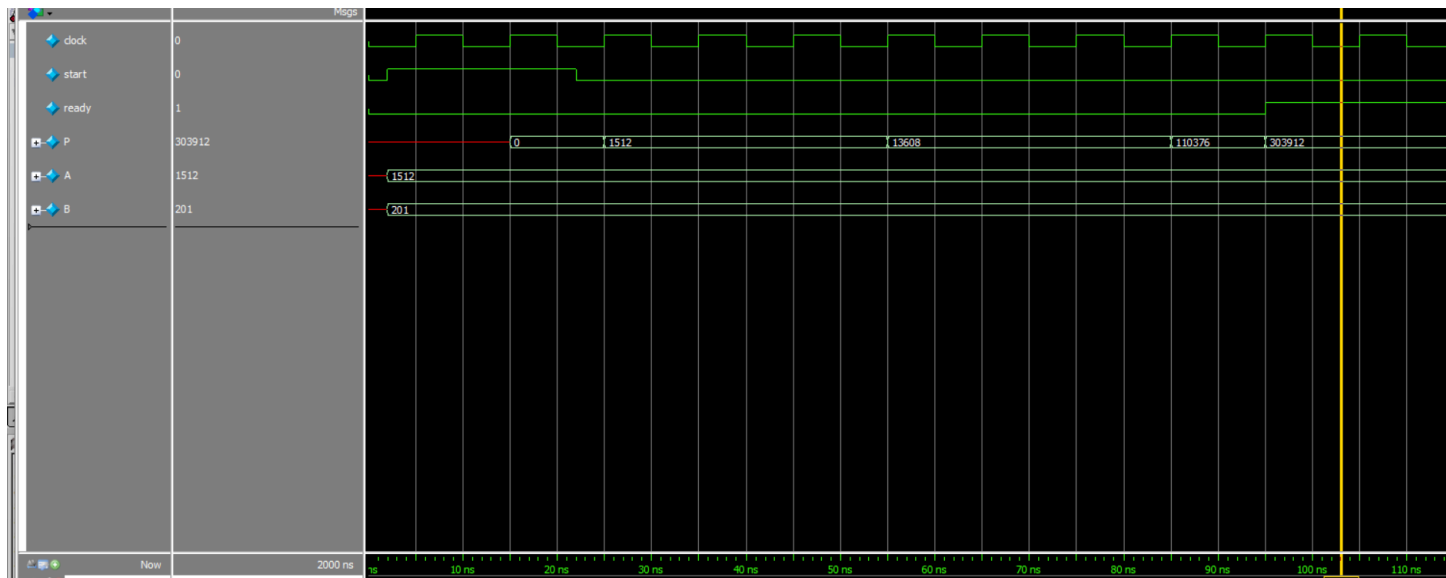
تست بنچ برای عرض 16 بیت:

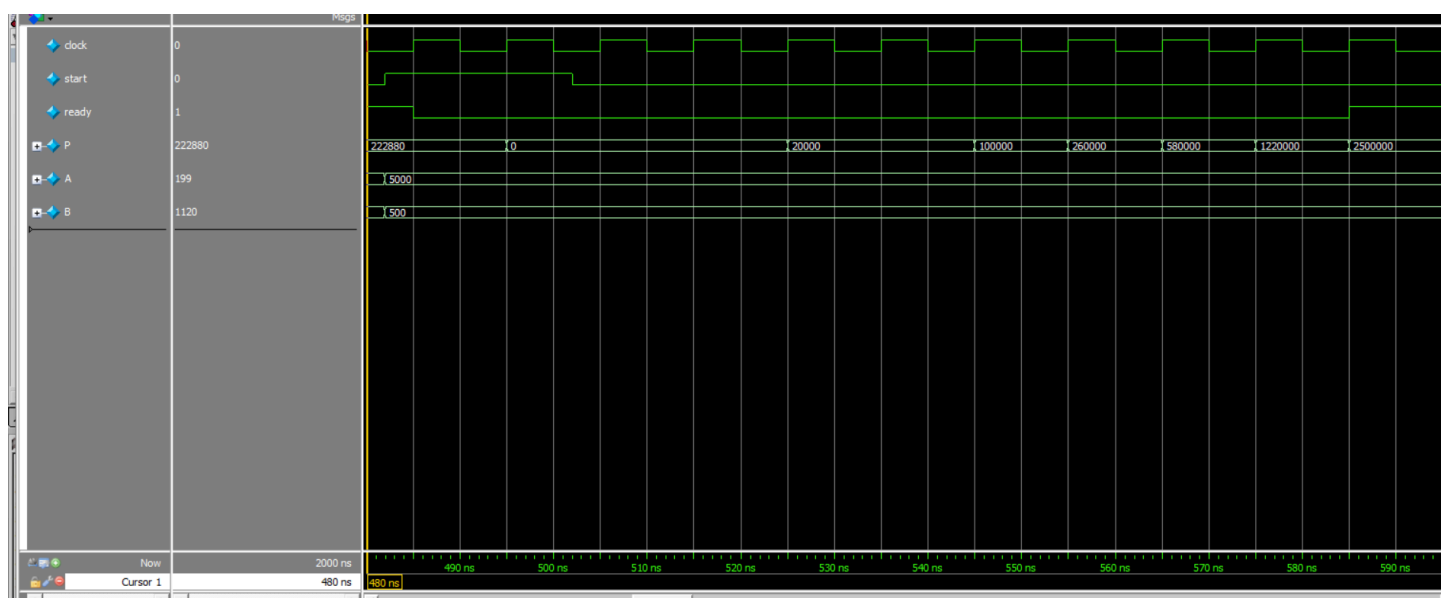
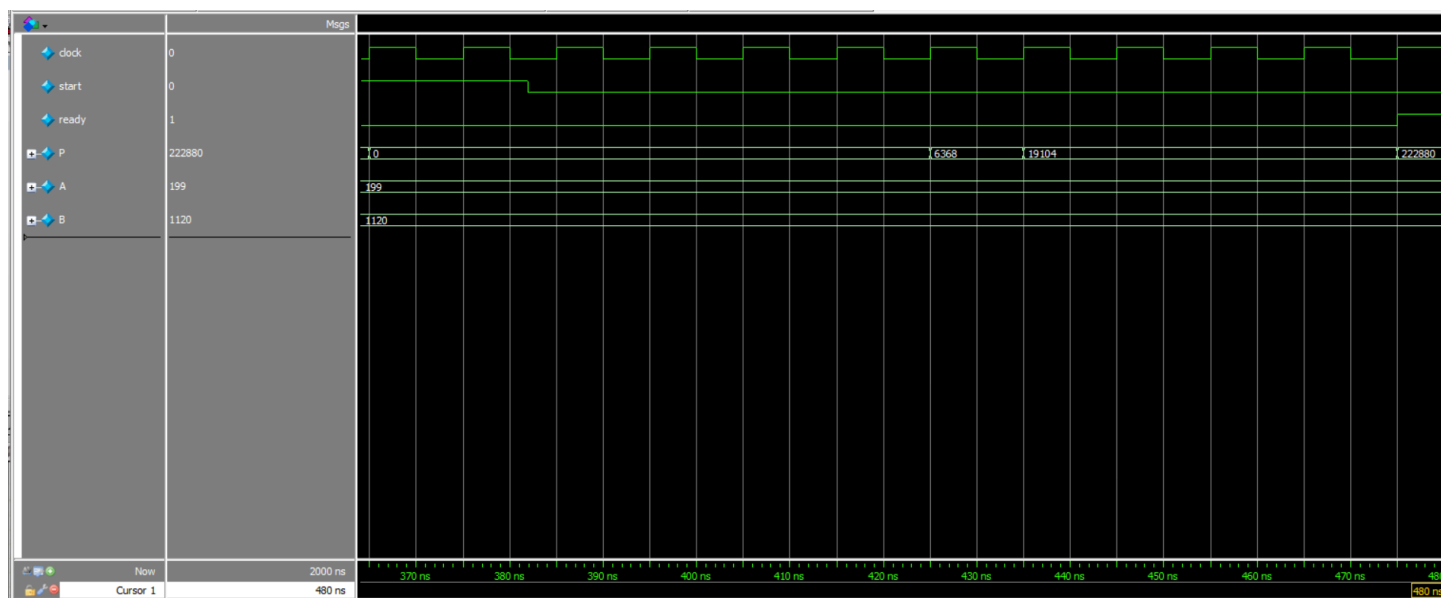

```

multiplier_tb2.sv
2  module multiplier_tb2();
5      logic [31:0] P;
6      logic[15:0] A,B;
7      shift_add_multiplier #(16) dc(clock,start,A,B,P,ready);
8      initial begin
9          clock=0;
10         start=0;
11     end
12     initial begin
13         repeat(400) #5 clock=~clock;
14     end
15     initial begin
16         #2
17         start=1;
18         A=1512;
19         B=201;
20         #20
21         start=0;
22         #100
23         start=1;
24         A=207;
25         B=810;
26         #20
27         start=0;
28         #100
29         start=1;
30         A=2000;
31         B=660;
32         #20
33         start=0;
34         #100
35         start=1;
36         A=199;
37         B=1120;
38         #20
39         start=0;
40         #100
41         start=1;
42         A=5000;
43         B=500;
44         #20
45         start=0;
46
47     end
48 endmodule

```

نتایج شبیه سازی:





برای محاسبه اعداد اعشاری باید اعشار دو تا عدد رو باهم جمع کنیم تا تعداد اعشار حاصل ضرب آن دو عدد معلوم شود. و از اینجا fix point مشخص میشود.

اثبات این موضوع بسیار راحت است و از طریق این است که میدانیم هر عدد بدون اعشار را برای تبدیل به عدد اعشاری لازم است تقسیم بر 2 به توان تعداد اعشارش کنیم. با ضرب دو عدد در هم توان های 2 با هم جمع میشود.

دو تا مثال:

مثال اول:

$$a=11.01 \rightarrow 3.25$$

$$b=100.1 \rightarrow 4.5$$

$$ab=1101 * 1001=01110101 \rightarrow \text{تا اعشار 3}$$

$$01110.101=14.62$$

مثال دوم:

$$a=1101.0101 \rightarrow 13.3125$$

$$b=0010.1011 \rightarrow 2.6875$$

$$ab=11010101 * 00101011= \mathbf{010001111000111} \rightarrow \text{تا اعشار 8}$$

$$\mathbf{0100011.11000111}=35.77734375$$

Reciprocal:

در این بخش میخواهیم با فرمول نیوتون رافسون معکوس یک عدد را حساب کنیم که آن عدد بین 1 و 2 است. یعنی n بیتی است و $n-1$ بیت اعشار دارد.

Initialize y_0 with an initial guess

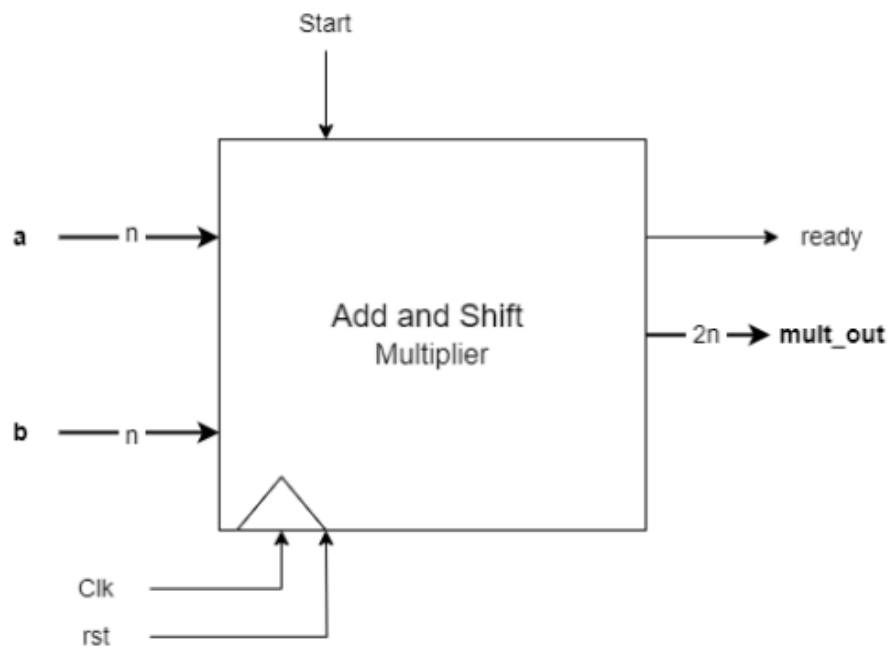
for $i = [0:n-1]$ do {

$$y_{i+1} \leftarrow y_i * (2.0 - x_i * y_i)$$

}

همان عدد ورودی است x_i .

$$y_1=1$$

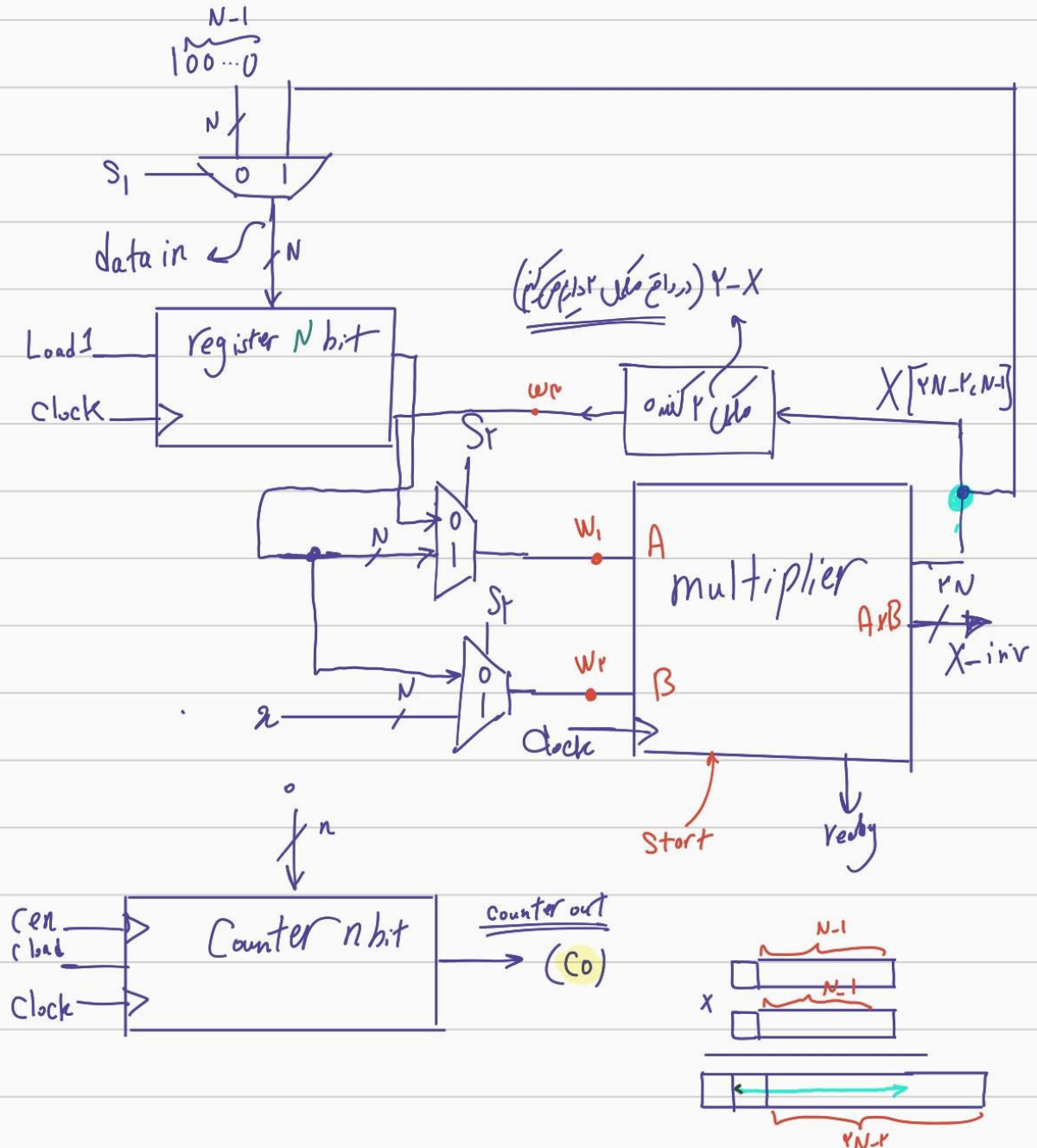


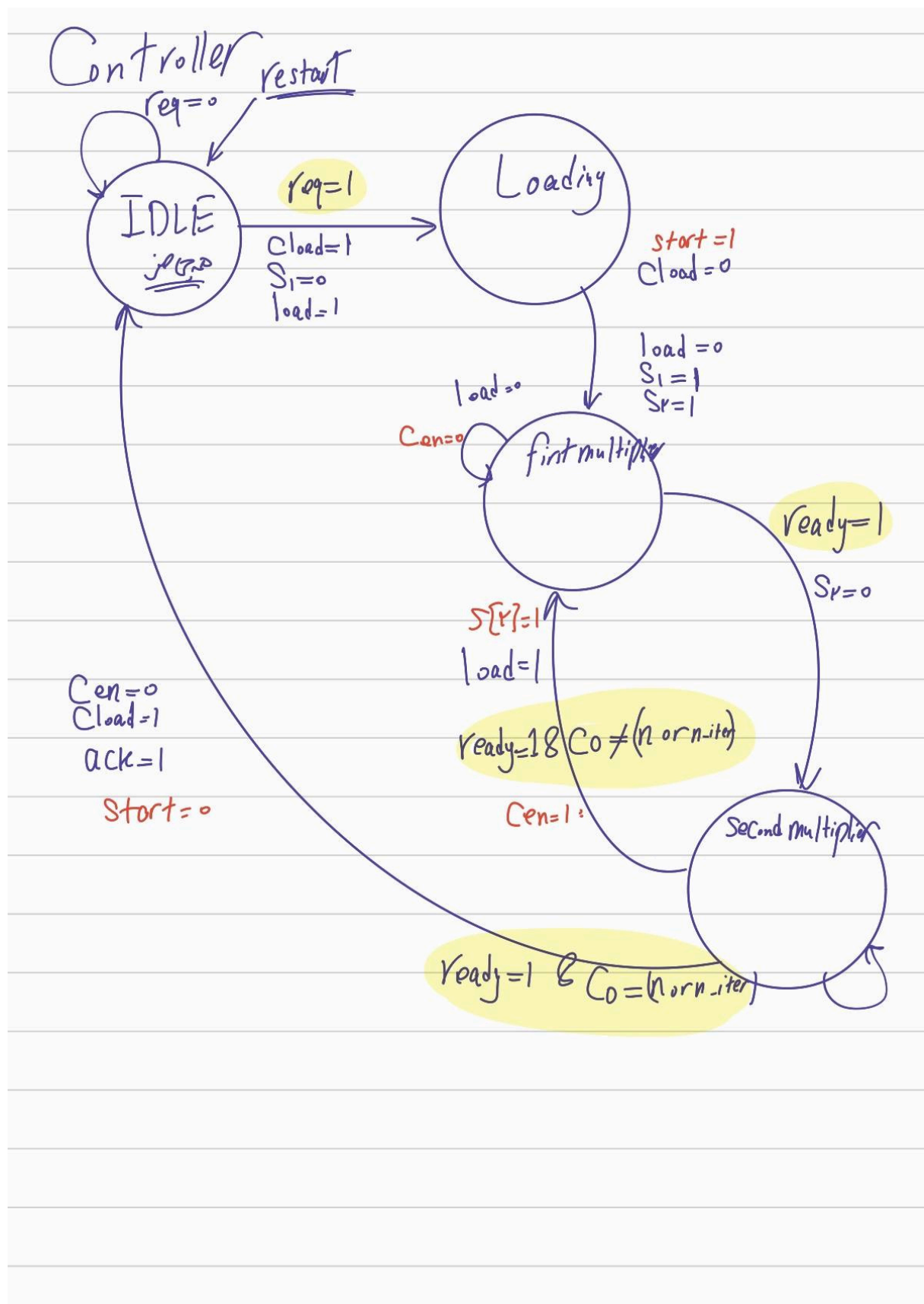
شکل ۱- دیاگرام ماژول Add and Shift

$$\frac{1V_0}{1V_N} \times \frac{\delta}{\epsilon} \rightarrow y_{i+1} = y_i \left(1 - \frac{1V_0}{\epsilon} y_i \right) \rightarrow \approx \frac{\epsilon}{\delta}$$

Data path:

$$y_1 = 1$$





کد های سیستم وریلاگ:

reciprocal.sv

```
1 module reciprocal #(parameter N=8)(input logic clock, req, input logic [N-1:0] x, input logic [3:0] n_iter,  
2   output logic [2*N-1:0] x_inv, output logic ack);  
3  
4   logic [1:2] s;  
5   logic cload, cen, load, start, ready;  
6   logic [3:0] co;  
7  
8  
9   data_path_reciprocal #(N) dp2(clock, cload, cen, load, start, s, x, x_inv, ready, co);  
10  controller_reciprocal cont2(clock, ready, req, co, s, start, cload, cen, load, ack, n_iter);  
11  
12  
13 endmodule  
14
```

data_path_reciprocal.sv

```
1 module data_path_reciprocal #(parameter N=8)(input logic clock, cload, cen, load, start, input logic [1:2] s  
2   , input logic [N-1:0] x, output logic [2*N-1:0] x_inv, output logic ready, output logic [3:0] co);  
3  
4   logic [N-1:0] r_out, data_in, w1, w2, w3;  
5   logic [2*N-1:0] mult_result;  
6  
7  
8   assign data_in = s[1] ? mult_result[2*N-2:N-1] : {1'b1, {(N-1){1'b0}}};  
9   assign w1 = s[2] ? r_out : w3;  
10  assign w2 = s[2] ? x : r_out;  
11  assign w3 = ~mult_result[2*N-2:N-1] + 1;  
12  assign x_inv = mult_result;  
13  register #(N) r(clock, load, data_in, r_out);  
14  shift_add_multiplier sam(clock, start, w1, w2, mult_result, ready);  
15  counter#(4) count(clock, cload, cen, 4'b0000, co);  
16 endmodule  
17  
18
```



```

1 module controller_reciprocal (input logic clock,ready,req,input logic[3:0] co,output logic [1:2]s,output logic start,cload,cen,load,ack,
2 input logic[3:0] n_iter);
3 parameter [2:0] idle=0,loading=1,fisrt_multiplier=2, second_multiplier=3 ;
4 logic [2:0] p_state,n_state;
5 logic first_time_mult1, first_time_mult2;
6 always @(req or co or p_state or ready)begin
7     case(p_state)
8     idle:begin
9         if(req)begin
10             cload=1;
11             cen=0;
12             s[1]=0;
13             load=1;
14             n_state=loading;
15         end
16     else begin
17         n_state=idle;
18     end
19 end
20 loading:begin
21     start=1;
22     ack=0;
23     cload=0;
24
25     first_time_mult1=1;
26     first_time_mult2=1;
27     n_state=fisrt_multiplier;
28
29 end
30
31
32
33     fisrt_multiplier:begin

```

```

30     end
33     fisrt_multiplier:begin
35         if(first_time_mult1)begin
36             first_time_mult1=0;
37             s[1]=1;////////
38             s[2]=1;
39             load=0;
40             cen=0;
41
42         end
43     else begin
44
45         if(ready)begin
46             s[2]=0;
47             first_time_mult2=1;
48             n_state=second_multiplier;
49
50         end
51
52     end
53
54
55
56     end
57     second_multiplier:begin
58
59         if(first_time_mult2)begin
60             first_time_mult2=0;
61         end
62     else begin
63
64         if(ready==1&&(co==n_iter))begin
65
66             cen=0;

```

```

6  always @(req or co or p_state or ready)begin
7      case(p_state)
8      end
9      end
10     end
11     second_multiplier:begin
12         if(ready==1&&(co==n_iter))begin
13             cen=0;
14             cload=1;
15             ack=1;
16             start=0;
17
18             n_state=idle;
19
20         end
21         else if(ready)begin
22             cen=1;
23             load=1;
24             s[2]=1;
25
26             first_time_mult1=1;
27             n_state=fisrt_multiplier;
28
29         end
30     end
31 end
32 default: begin
33     n_state = idle;
34 end
35 endcase
36 end

```

تست بنچ برای ورودی های مختلف:

رقم اعشار 14 X_inv=

رقم اعشار 7 X=

```

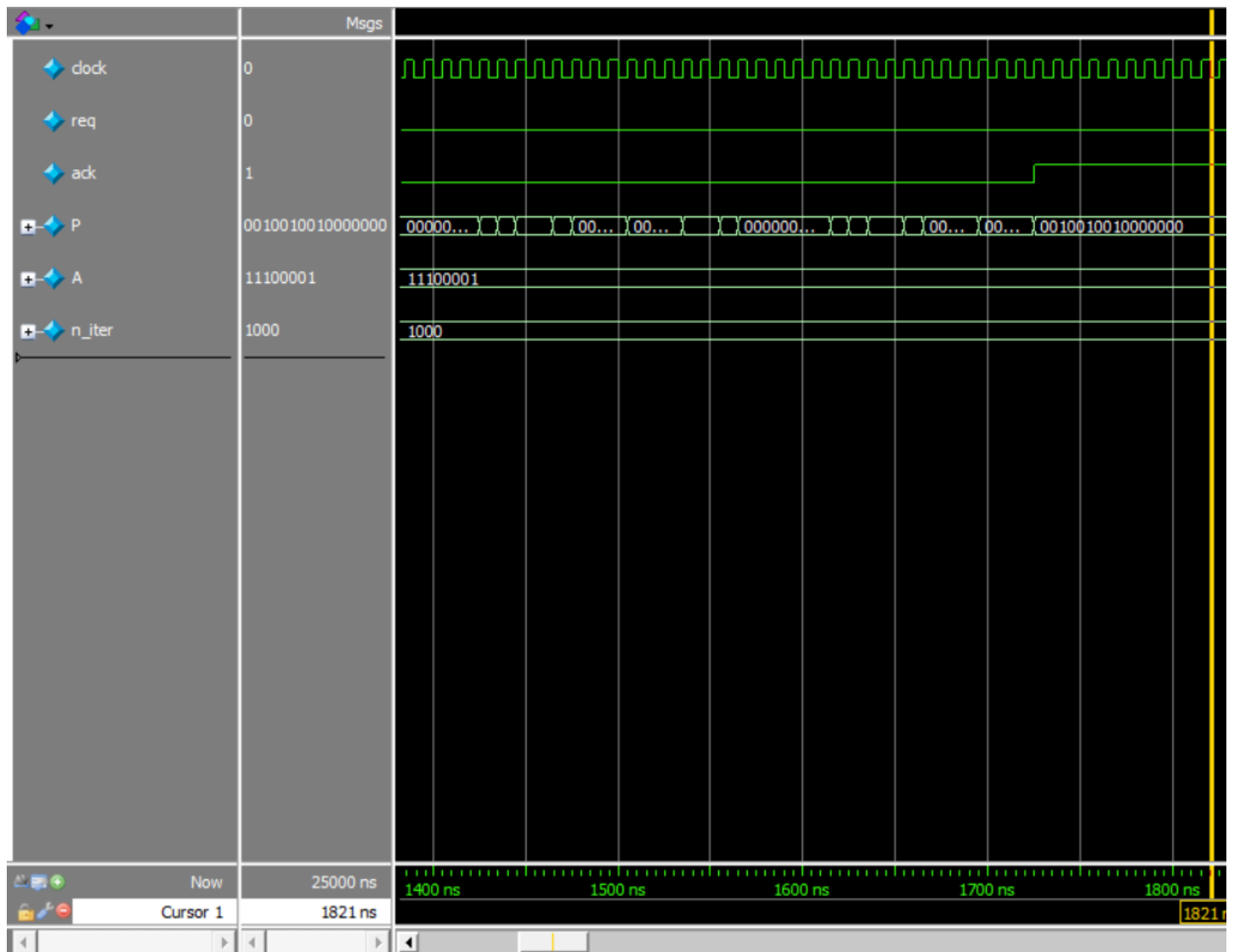
reciprocal_tb.sv
2  module reciprocal_tb();
3
4      logic clock,req,ack;
5      logic [15:0] P;
6      logic[7:0] A=8'b11100001;
7      logic [3:0] n_iter=8;
8      reciprocal #(8) rep(clock,req,A,n_iter,P,ack);
9      initial begin
10         clock=0;
11         req=0;
12         #30
13         req=1;
14         #200
15         req=0;
16         #2000
17         A=8'b11110111;
18         req=1;
19         #200
20         req=0;
21         #2000
22         A=8'b10010100;
23         req=1;
24         #200
25         req=0;
26         #2000
27         A=8'b10100000;
28         req=1;
29         #200
30         req=0;
31
32
33     end
34     initial begin
35         repeat(5000) #5 clock=~clock;
36     end
37 endmodule

```

$X=11100001 \rightarrow 1.7578125$

مقدار واقعی معکوس: 0.5688888888888888

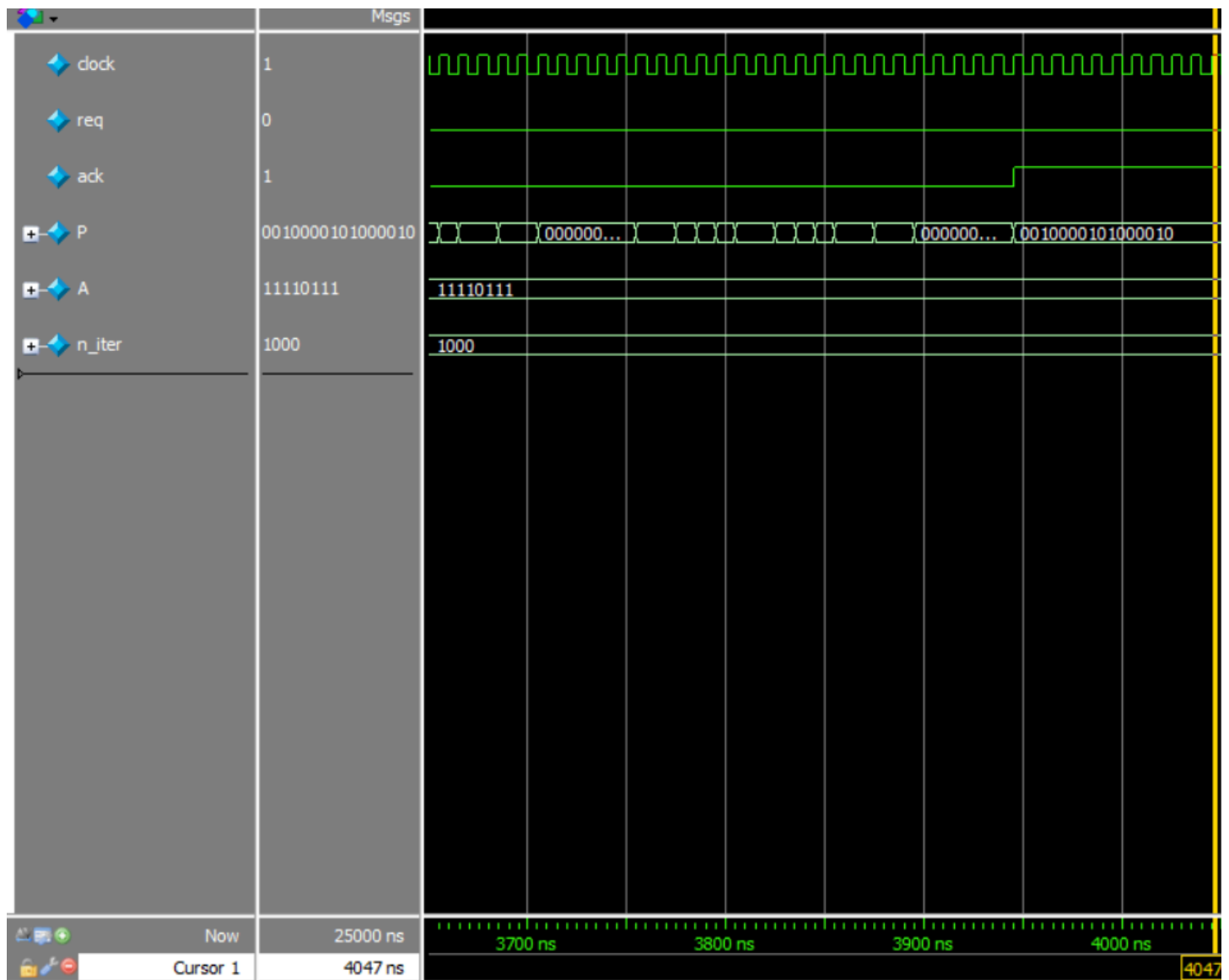
$X_{inv}=0010010010000000 \rightarrow 0.5703125$



$X=11110111 \rightarrow 1.9296875$

مقدار واقعی معکوس: 0.5182186234817

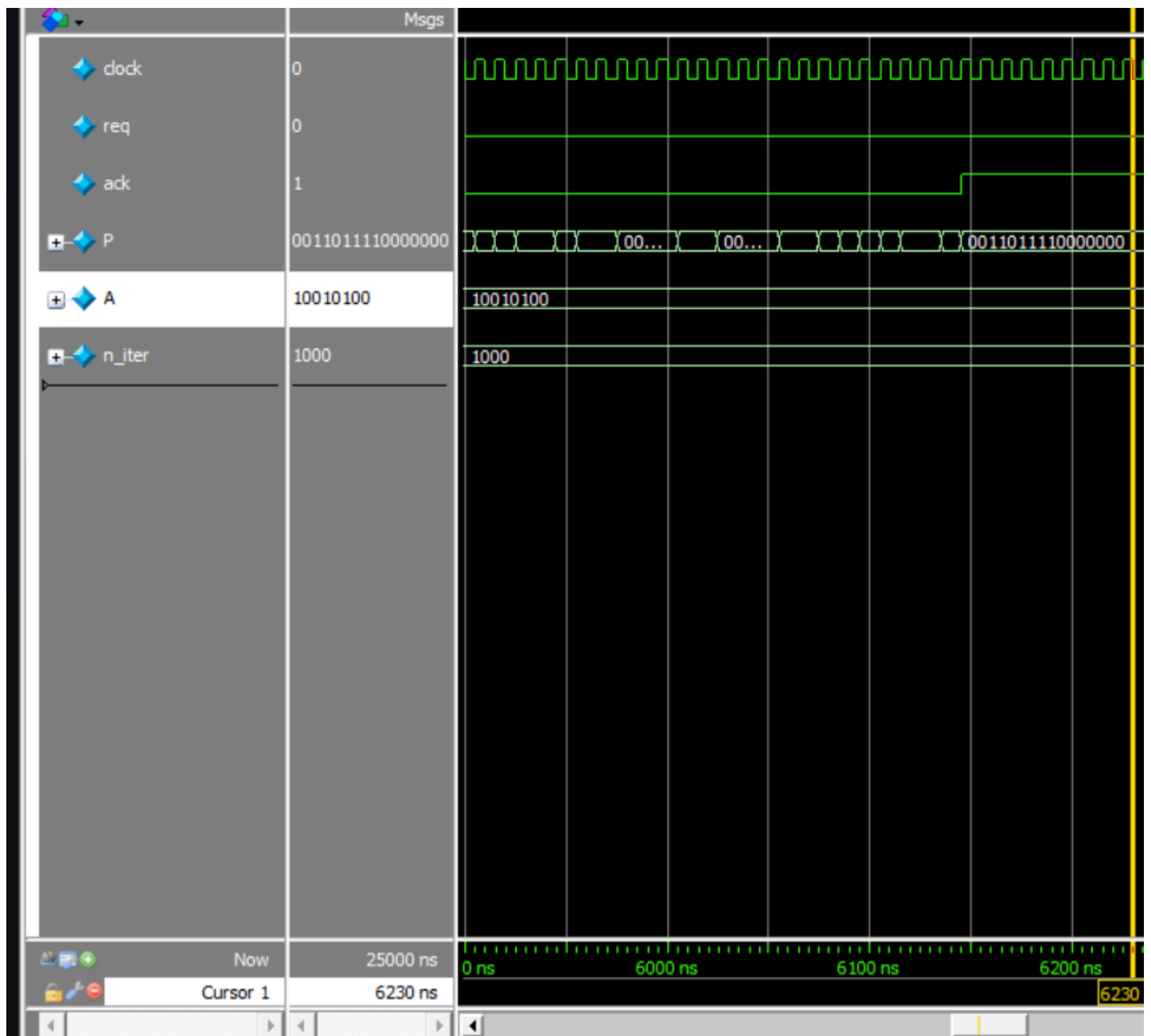
$X_{inv}=0010000101000010 \rightarrow 0.5196533203125$



$X=10010100 \rightarrow 1.15625$

مقدار واقعی معکوس: 0.8648648648

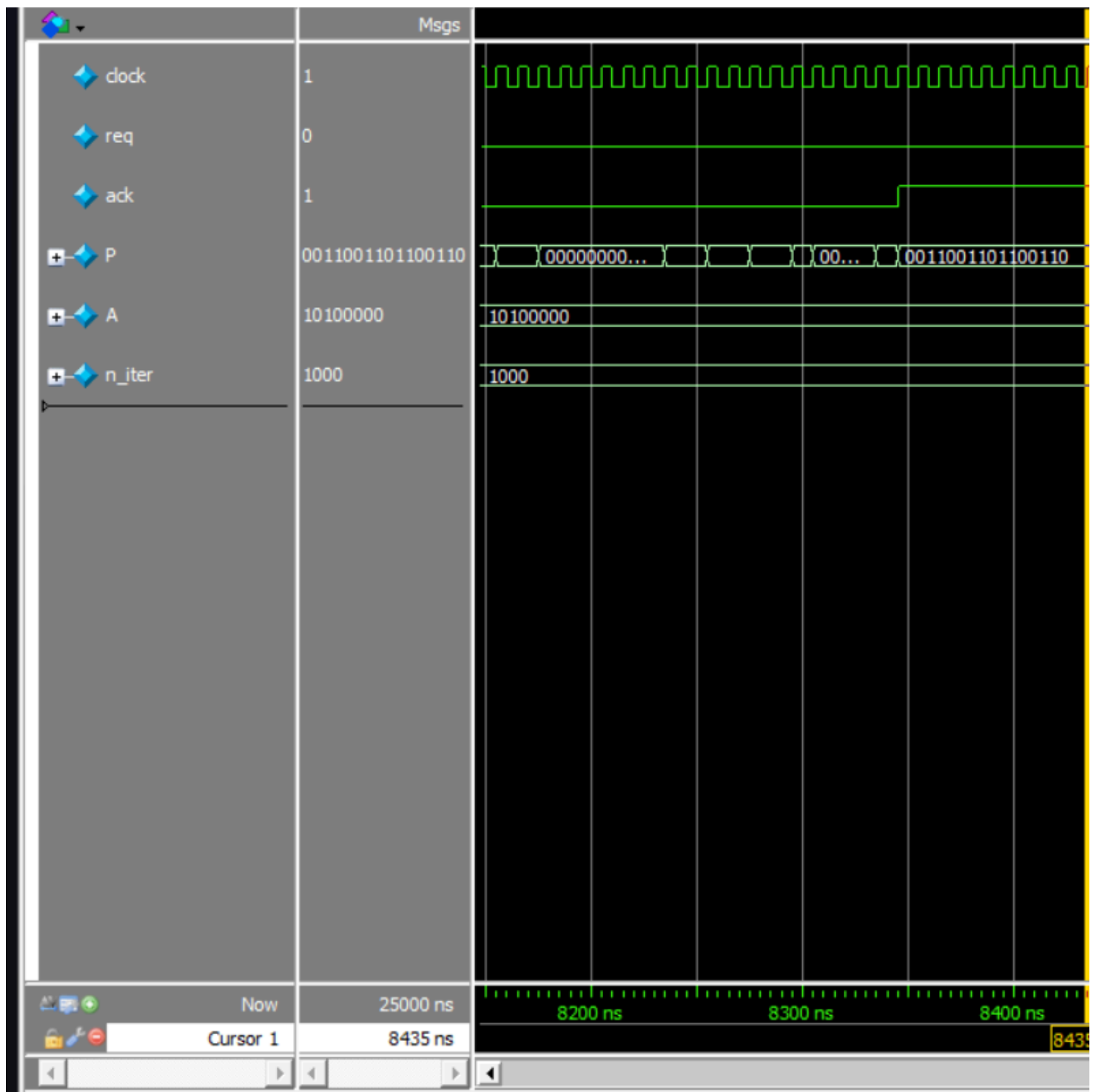
$X_{inv}=0011011110000000 \rightarrow 0.8671875$



$$X=10100000 \rightarrow 1.25$$

مقدار واقعی معکوس: 0.8

$$X_{inv}=0011001101100110 \rightarrow 0.8031005859375$$



در بخش آخر با نرم افزار Quartus II یک پروژه جدید میسازیم.

و با دیواس ساینکلون 2 کد های بخش قبل را سنتز میکنیم. توجه داریم که در این بخش تاپ ماژولمان باید reciprocal باشد.

این تصویری از محیط نرم افزار پس از کامپایل است:

The image displays two screenshots of the Quartus II 64-bit software interface, showing the compilation process and results for a project named 'ca55'.

Top Screenshot: The 'Entity' tab is selected, showing the compilation results for the 'ca55' project. The results are summarized in the following table:

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C5AF256A7	114 (0)	57 (0)	0 (0)	0	0	0	0	0	31	0	57 (0)	0 (0)	57 (0)
reciprocal	43 (43)	4 (4)	0 (0)	0	0	0	0	0	0	0	39 (39)	0 (0)	4 (4)
data_path_reciprocal:dp2	71 (8)	53 (0)	0 (0)	0	0	0	0	0	0	0	18 (8)	0 (0)	53 (9)

The 'Tasks' pane on the left shows the compilation process, with 'Compile Design' selected. The 'Compilation Report - ca55' is open, showing the Verilog code for the 'reciprocal' module.

```
1 module reciprocal #(parameter N=8) (input logic clock, req, input logic [N-1:0] x, input logic [3:0] n_iter,
2   output logic [2*N-1:0] x_inv, output logic ack);
3
4   logic [1:2] s;
5   logic cload, cen, load, start, ready;
6   logic [3:0] co;
7
8
9   data_path_reciprocal #(N) dp2(clock, cload, cen, load, start, s, x, x_inv, ready, co);
10  controller_reciprocal cont2(clock, ready, req, co, s, start, cload, cen, load, ack, n_iter);
11
12 endmodule
```

Bottom Screenshot: The 'Messages' pane is selected, showing the compilation results. The messages indicate that the design is not fully constrained for setup requirements, but the compilation was successful with 0 errors and 7 warnings.

System (1) / Processing (179)

Entity

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C5AF256A7	114 (0)	57 (0)	0 (0)	0	0	0	0	0	31	0	57 (0)	0 (0)	57 (0)
reciprocal	43 (43)	4 (4)	0 (0)	0	0	0	0	0	0	0	39 (39)	0 (0)	4 (4)
controller_reciprocal:cont2													
data_path_reciprocal:dp2	71 (8)	53 (0)	0 (0)	0	0	0	0	0	0	0	18 (8)	0 (0)	53 (9)

Flow Summary

Flow Status: Successful - Tue Jan 28 23:28:37 2025

Quartus II 64-Bit Version: 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition

Revision Name: ca55

Top-level Entity Name: reciprocal

Family: Cyclone II

Device: EP2C5AF256A7

Timing Models: Final

Total logic elements: 114 / 4,608 (2 %)

Total combinational functions: 114 / 4,608 (2 %)

Dedicated logic registers: 57 / 4,608 (1 %)

Total registers: 57

Total pins: 31 / 158 (20 %)

Total virtual pins: 0

Total memory bits: 0 / 119,808 (0 %)

Embedded Multiplier 9-bit elements: 0 / 26 (0 %)

Total PLLs: 0 / 2 (0 %)

Task

- Open New Project Wizard
- Open Existing Project
- Create Revision
- Specify Project Libraries
- Import Database
- Create Design
- Assign Constraints
- Compile Design
- Analysis & Synthesis
- Fitter (Place & Route)
- Assembler (Generate programming files)
- TimeQuest Timing Analysis
- EDA Netlist Writer
- Program Device (Open Programmer)
- Verify Design
- Simulate Design
- On-chip Debugging
- Device Programmer

Messages

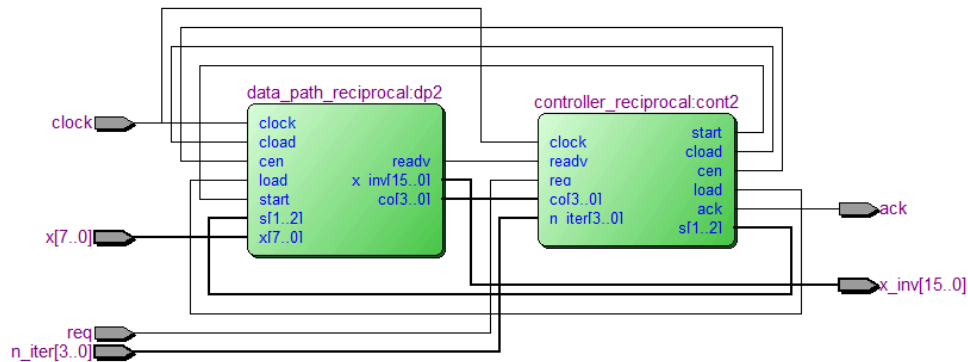
332102 Design is not fully constrained for setup requirements

332102 Design is not fully constrained for hold requirements

System (1) / Processing (179) / 100%

در بخش summary با فرض 8 بیتی بودن ماژول، از 57 رجیستر استفاده شده است که در اینجا هر فلیپ فلاپ را یک رجیستر در نظر گرفته است در واقع.

این هم شکل RTL کلی ماژول:



برای پیدا کردن FMAX که از فرمول زیر به دست میاد به بخش ANALYSIS
 TIME رفته و آن را بررسی میکنیم که برابر 244.92 مگاهرتز است.

$$\frac{1}{CriticalPathDelay} = FMAX$$

Report

TimeQuest Timing Analyzer Summary

Clocks Summary

Fmax Summary

Tasks

Report Minimum Pulse Width Summary

Report Max Skew Summary

Datasheet

Report Fmax Summary

Report Datasheet

Device Specific

Report TCCS

Report RSKM

Report DDR

Report Metastability

Dagnostic

Report Clocks

Report Clock Transfers

Report Unconstrained Paths

Report SDC

Report Ignored Constraints

Check Timing

Report Partitions

Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	INF MHz	362.58 MHz	controller_reciprocal:cont2 first_time_mult1	limit due to hold check
2	244.92 MHz	244.92 MHz	clock	

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such and other slack reports for sign-off analysis.

Console

No user constrained base clocks found in the design. Calling "derive_clocks -period 1.0"

Deriving Clocks

Clock target controller_reciprocal:cont2|first_time_mult1 of clock controller_reciprocal:cont2|first_time_mult1 is fe

The following timing edges are non-unate. TimeQuest will assume pos-unate behavior for these edges in the clock netv

report_clocks -panel_name "Clocks Summary"

report_clock_fmax_summary -panel_name "Fmax Summary"

برای حالت 16 بیتی

Compilation Report - reciprocal

reciprocal.sv

Assignment Editor

How Summary

Flow Status

Successful - Fri Jan 31 04:03:43 2025

Quartus II 64-Bit Version

13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition

Revision Name

reciprocal

Top-level Entity Name

reciprocal

Family

Cyclone II

Total logic elements

100 / 4,608 (2 %)

Total combinational functions

99 / 4,608 (2 %)

Dedicated logic registers

50 / 4,608 (1 %)

Total registers

50

Total pins

55 / 89 (62 %)

Total virtual pins

0

Total memory bits

0 / 119,808 (0 %)

Embedded Multiplier 9-bit elements

0 / 26 (0 %)

Total PLLs

0 / 2 (0 %)

Device

EP2CST144C6

Timing Models

Final

Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	INF MHz	365.76 MHz	controller_reciprocal:cont2 first_time_mult1	limit due to hold check
2	306.47 MHz	306.47 MHz	clock	

برای حالت 32 بیتی:

```

module reciprocal #(parameter N=32)(input logic clock, req, input logic [N-1:0] x, input logic [3:0] n_iter,
  output logic [2*N-1:0] x_inv, output logic ack);

  logic [1:2] s;
  logic cload, cen, load, start, ready;
  logic [3:0] co;

  data_path_reciprocal #(N) dp2(clock, cload, cen, load, start, s, x, x_inv, ready, co);
  controller_reciprocal cont2(clock, ready, req, co, s, start, cload, cen, load, ack, n_iter);

endmodule

```

Flow Summary	
Flow Status	Successful - Fri Jan 31 04:06:55 2025
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	reciprocal
Top-level Entity Name	reciprocal
Family	Cyclone II
Total logic elements	93 / 4,608 (2 %)
Total combinational functions	93 / 4,608 (2 %)
Dedicated logic registers	49 / 4,608 (1 %)
Total registers	49
Total pins	103 / 158 (65 %)
Total virtual pins	0
Total memory bits	0 / 119,808 (0 %)
Embedded Multiplier 9-bit elements	0 / 26 (0 %)
Total PLLs	0 / 2 (0 %)
Device	EP2C5F256C6
Timing Models	Final

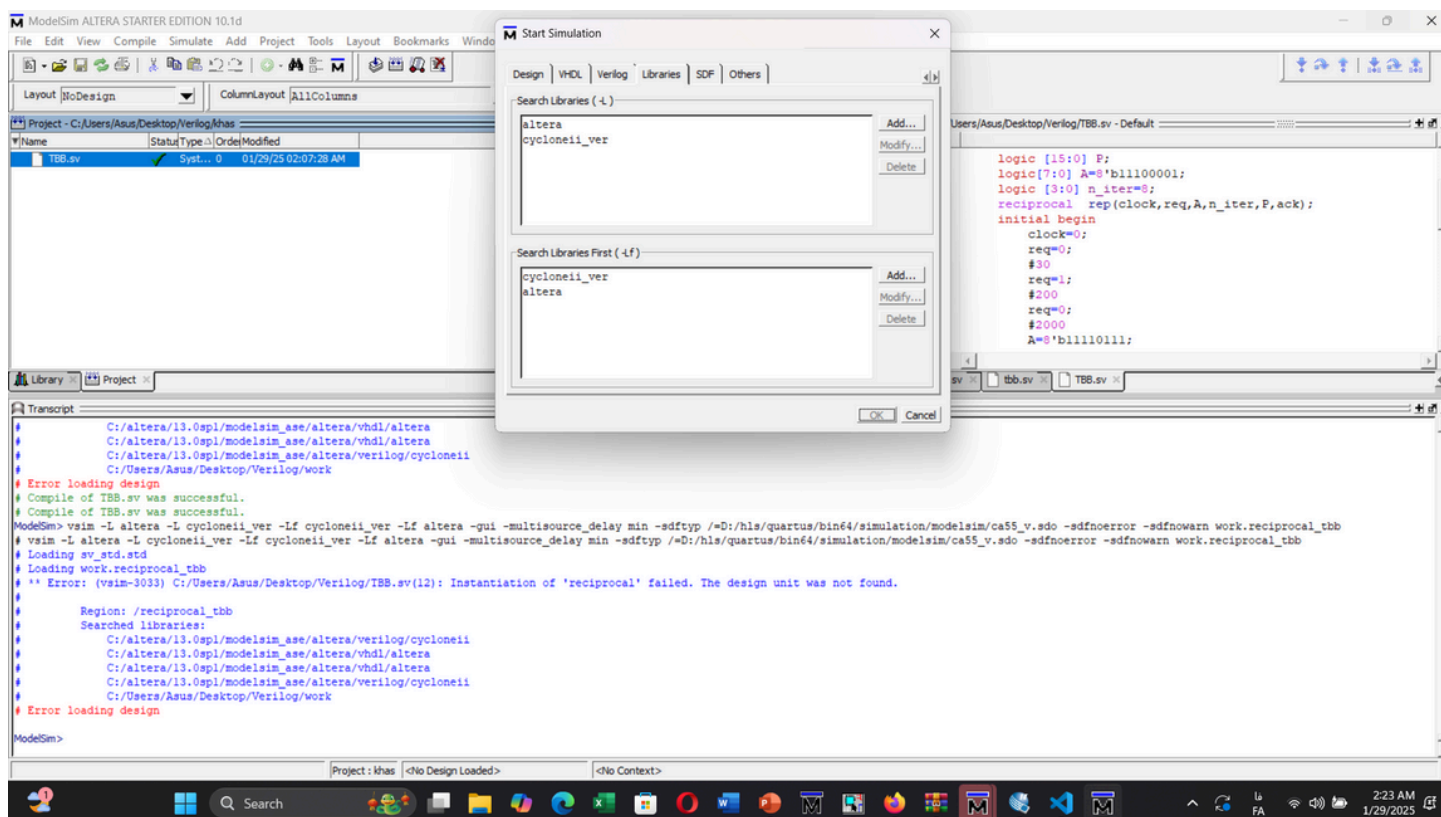
Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
	297.27 MHz	297.27 MHz	clock	
	1312.34 MHz	877.19 MHz	controller_reciprocal:cont2 first_time_mult1	limit due to hold check

در آخر با فایل sdo تولید شده توسط کوارتس می توانیم به Model sim altera edition رفته و پس از اضافه کردن فایل و لایبرری های مورد نیاز فایل تست بنچ جدید را simulate کرده و نتایج قسمت قبل تکرار میشود.

که به دلیل اروری نامعلوم این بخش تکمیل نشد.

Name	Date modified	Type	Size
ca55.sft	1/29/2025 12:05 AM	SFT File	1 KB
ca55.svo	1/29/2025 12:05 AM	SVO File	139 KB
ca55_fast.svo	1/29/2025 12:05 AM	SVO File	139 KB
ca55_modelsim.xrf	1/29/2025 12:05 AM	txtfile	17 KB
ca55_v.sdo	1/29/2025 12:05 AM	SDO File	97 KB
ca55_v_fast.sdo	1/29/2025 12:05 AM	SDO File	94 KB



دوباره تلاش میکنیم. این بار یک پروژه جدید میسازیم دوباره و دوباره سنتز میکنیم.

این بار فایل های tb و sdo و sv را همگی داخل یک فولدر می بریم و به یک پروژه جدید در مدل سیم اضافه میکنیم.

و کار های قسمت قبل را تکرار میکنیم

این بار به درستی simulate میشود ولی خروجی x_inv صفر میماند. pin های assign را هم بررسی میکنیم و مشکلی ندارند.

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
ack	Output				PIN_75	3.3-V L...efault)		24mA (default)	
clock	Input				PIN_17	3.3-V L...efault)		24mA (default)	
n_iter[3]	Input				PIN_79	3.3-V L...efault)		24mA (default)	
n_iter[2]	Input				PIN_80	3.3-V L...efault)		24mA (default)	
n_iter[1]	Input				PIN_81	3.3-V L...efault)		24mA (default)	
n_iter[0]	Input				PIN_94	3.3-V L...efault)		24mA (default)	
req	Input				PIN_21	3.3-V L...efault)		24mA (default)	
x[7]	Input				PIN_22	3.3-V L...efault)		24mA (default)	
x[6]	Input				PIN_115	3.3-V L...efault)		24mA (default)	
x[5]	Input				PIN_114	3.3-V L...efault)		24mA (default)	
x[4]	Input				PIN_97	3.3-V L...efault)		24mA (default)	
x[3]	Input				PIN_112	3.3-V L...efault)		24mA (default)	
x[2]	Input				PIN_70	3.3-V L...efault)		24mA (default)	
x[1]	Input				PIN_113	3.3-V L...efault)		24mA (default)	
x[0]	Input				PIN_18	3.3-V L...efault)		24mA (default)	
x_inv[15]	Output				PIN_86	3.3-V L...efault)		24mA (default)	
x_inv[14]	Output				PIN_125	3.3-V L...efault)		24mA (default)	
x_inv[13]	Output				PIN_122	3.3-V L...efault)		24mA (default)	
x_inv[12]	Output				PIN_101	3.3-V L...efault)		24mA (default)	
x_inv[11]	Output				PIN_121	3.3-V L...efault)		24mA (default)	
x_inv[10]	Output				PIN_118	3.3-V L...efault)		24mA (default)	
x_inv[9]	Output				PIN_119	3.3-V L...efault)		24mA (default)	
x_inv[8]	Output				PIN_99	3.3-V L...efault)		24mA (default)	
x_inv[7]	Output				PIN_120	3.3-V L...efault)		24mA (default)	
x_inv[6]	Output				PIN_96	3.3-V L...efault)		24mA (default)	
x_inv[5]	Output				PIN_87	3.3-V L...efault)		24mA (default)	
x_inv[4]	Output				PIN_93	3.3-V L...efault)		24mA (default)	
x_inv[3]	Output				PIN_103	3.3-V L...efault)		24mA (default)	
x_inv[2]	Output				PIN_92	3.3-V L...efault)		24mA (default)	
x_inv[1]	Output				PIN_100	3.3-V L...efault)		24mA (default)	
x_inv[0]	Output				PIN_104	3.3-V L...efault)		24mA (default)	

