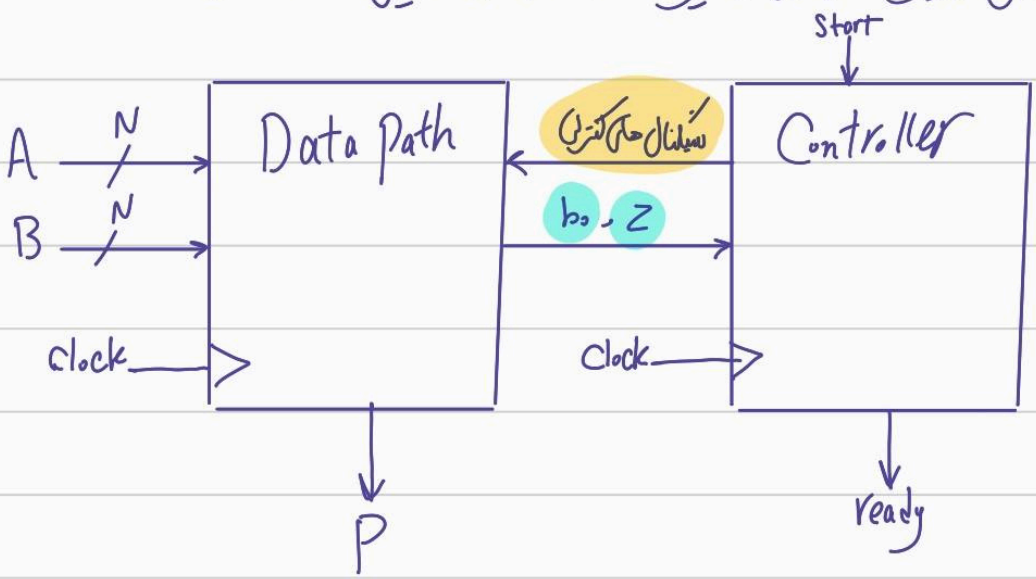
تمرین کامپیوتری امتیازی مدار منطقی

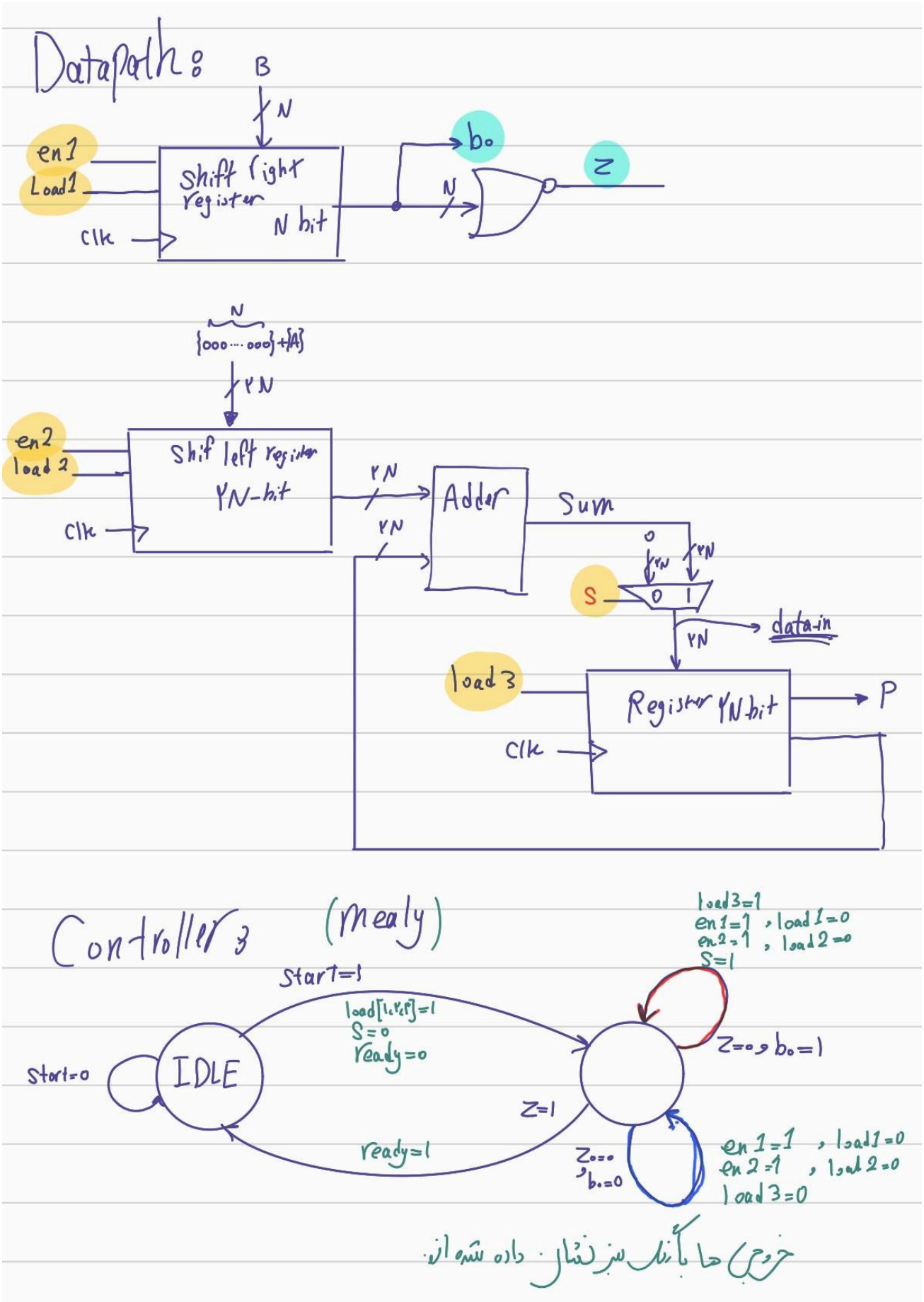
محمدامین رشید 810102454

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-and-Add با هم ضرب کنیم و یک خروجی ۲N بیتی بدهیم.

ساختار کلی مدار حل از یک Datapath و یک Controller تشکیل خواهد شد.





کد های سیستم وریلاگ:

shift_add_multiplier.sv

```
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);

    logic [1:2] en;
    logic [1:3]load;
    logic z,b0,s;

    data_path_multiplier #(N) dp(clock,A,B,en,load,s,b0,z,P);
    controller_multiplier cont(clock,start,z,b0,en,load,ready,s);

endmodule
```

data_path_multiplier.sv

```
module data_path_multiplier #(parameter N=8)(input logic clock,input logic [N-1:0] A,B,
input logic [1:2] en,input logic[1:3] load,input logic s,output logic b0,z,output logic [2*N-1:0] P);

logic [N-1:0] B_out;
logic [2*N-1:0] A_out,sum,data_in;


shift_register #(N) sr1(clock,1,load[1],en[1],B,B_out);
shift_register #(2*N) sr2(clock,0,load[2],en[2],{{N{1'b0}},A},A_out);

assign b0=B_out[0];
assign z= ~|B_out;
assign sum=A_out + P;
assign data_in=s?sum:{2*N{1'b0}};

register #(2*N) r1(clock,load[3],data_in,P);


endmodule
```

controller_multiplier.sv

```
module controller_multiplier (input logic clock,start,z,b0,output logic[1:2] en,output logic [1:3] load ,output logic ready,s);
parameter  idle=0,active=1;
logic p_state,n_state;
always @(start or z or p_state or b0)begin
    case(p_state)
     idle:begin
        en ='{default:0};
        load ='{default:0};
        if(start)begin
           load ='{default:1};
           s=0;
           ready=0;
           n_state=active;
        end
        else begin
            n_state=idle;
        end
     end
      active:begin
           if(z)begin
               ready=1;
                en ='{default: 0};
                load='{default: 0};
               n_state=idle;

           end
           else if ((z==0)&&(b0==0))begin
            en ='{default:1};
            load ='{default:0};

            n_state=active;

           end
           else begin
            s=1;
            load ='{default:0};
            load[3]=1;
```

controller_multiplier.sv

```
always @(start or z or p_state or b0)begin
    case(p_state)
     end
      active:begin
           else if ((z==0)&&(b0==0))begin

            n_state=active;

           end
           else begin
            s=1;
            load ='{default:0};
            load[3]=1;
            en ='{default:1};
            n_state=active;
           end
     end
         default: begin

                en ='{default: 0};
                load='{default: 0};
                ready=0;
                s=0;
                n_state = idle;
            end
    endcase
end

always @(posedge clock)begin

     p_state<=n_state;
end
endmodule
```

shift_register.sv

```sv
module shift_register #(parameter N=8)(input logic clock,input logic right_shift,input logic load,en,input logic [N-1:0] data_in
,output logic [N-1:0] data_out);

    always @(posedge clock) begin
        if (load) begin
            data_out <= data_in;
        end else begin
            if(en)begin
                if (right_shift) begin
                    data_out <= {1'b0, data_out[N-1:1]};
                end else begin
                    data_out <= {data_out[N-2:0],1'b0};
                end
            end
        end
    end
endmodule

```

register.sv

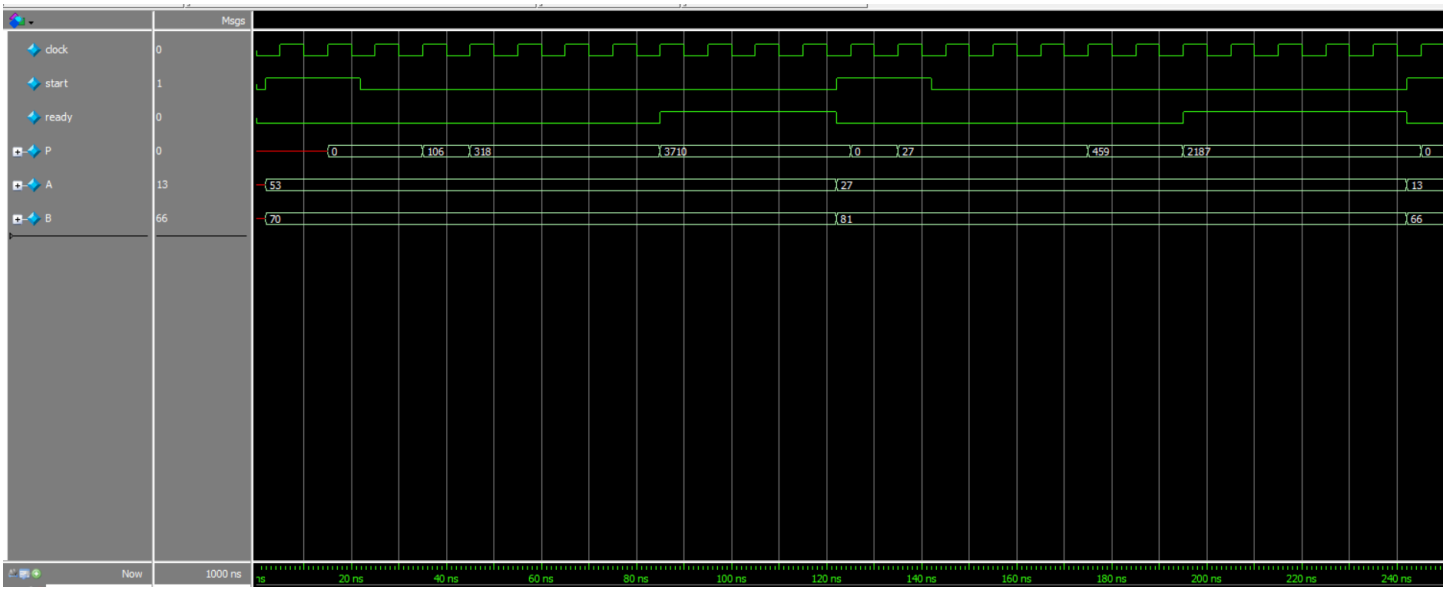
```sv
module register #(parameter N=8)(input logic clock,input logic load,input logic [N-1:0] data_in,output logic [N-1:0] data_out );

    always @(posedge clock ) begin

        if(load)begin
            data_out<= data_in;

        end

    end
endmodule

```

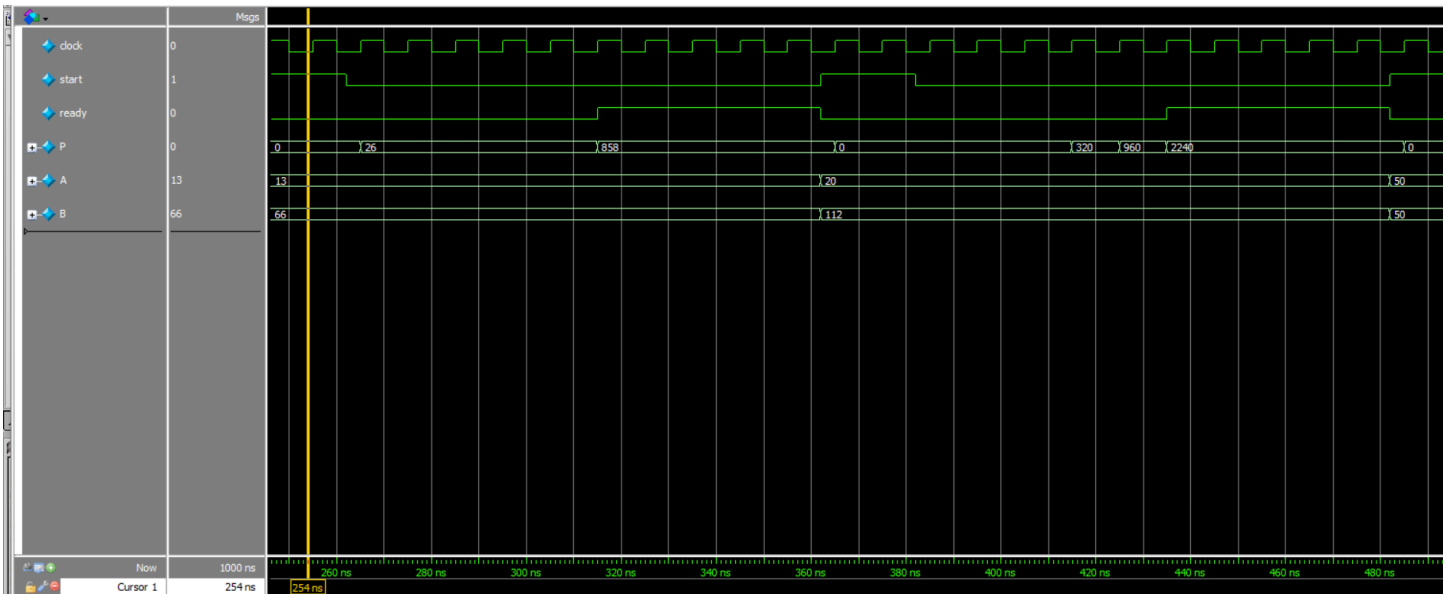
تست بنچ برای عرض 8 بیت:

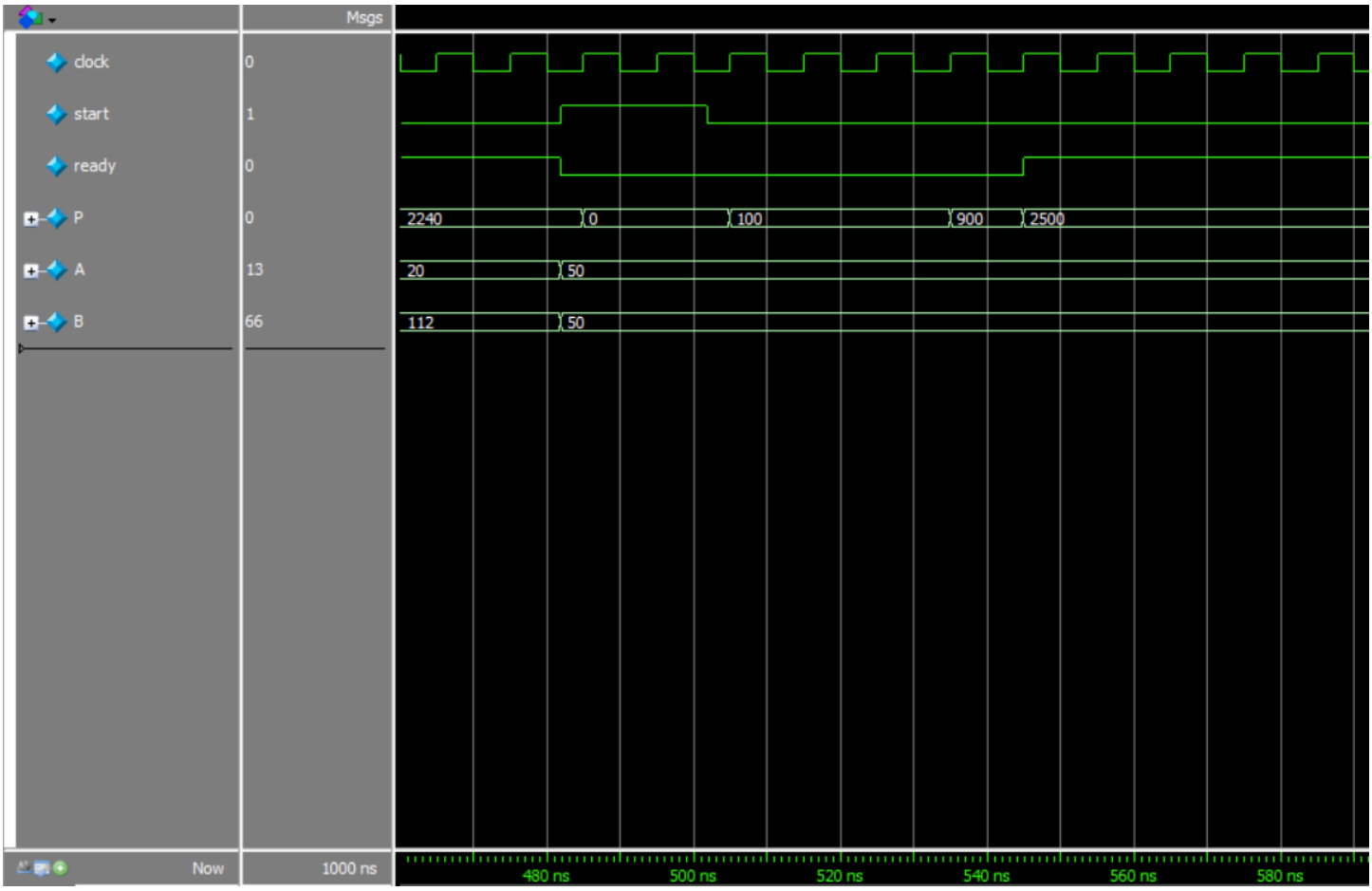
multiplier_tb.sv

```
module multiplier_tb();
    logic[7:0] A,B;
    shift_add_multiplier #(8) dc(clock,start,A,B,P,ready);
    initial begin
        clock=0;
        start=0;
    end
    initial begin
    repeat(200) #5 clock=~clock;
    end
    initial begin
        #2
        start=1;
        A=8'b00110101;
        B=8'b01000110;
        #20
        start=0;
        #100
        start=1;
        A=27;
        B=81;
        #20
        start=0;
        #100
        start=1;
        A=13;
        B=66;
        #20
        start=0;
        #100
        start=1;
        A=20;
        B=112;
        #20
        start=0;
        #100
        start=1;
        A=50;
        B=50;
        #20
        start=0;

    end
endmodule
```

نتایج شبیه سازی:





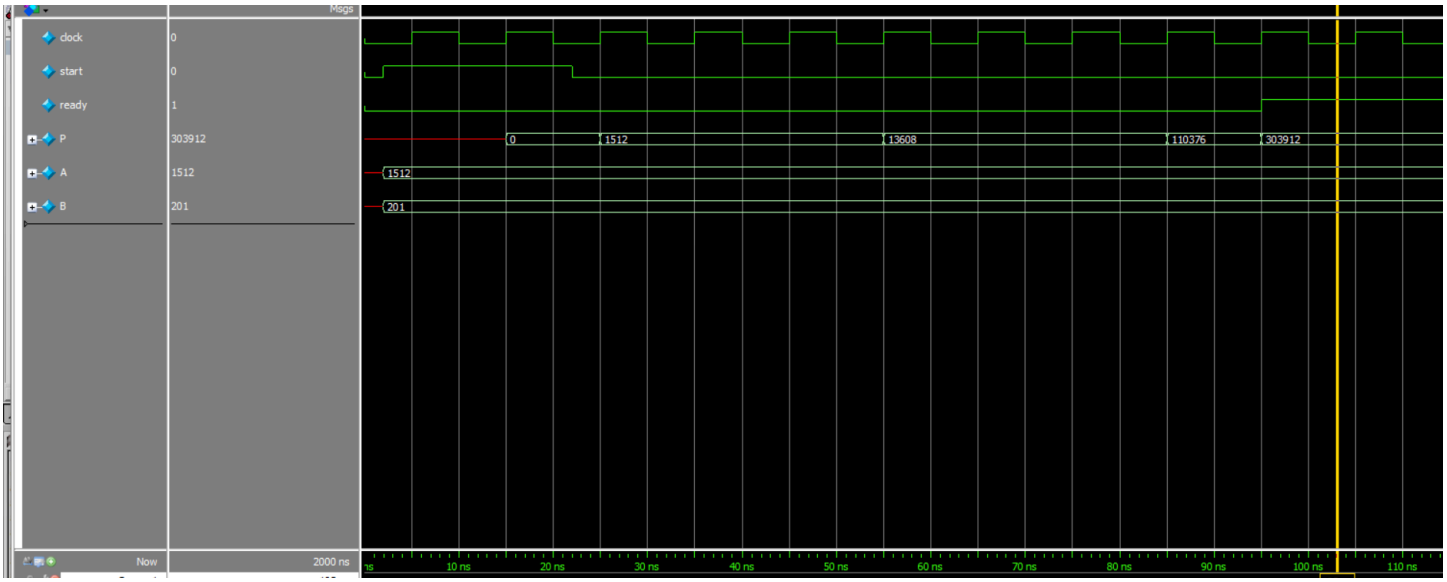


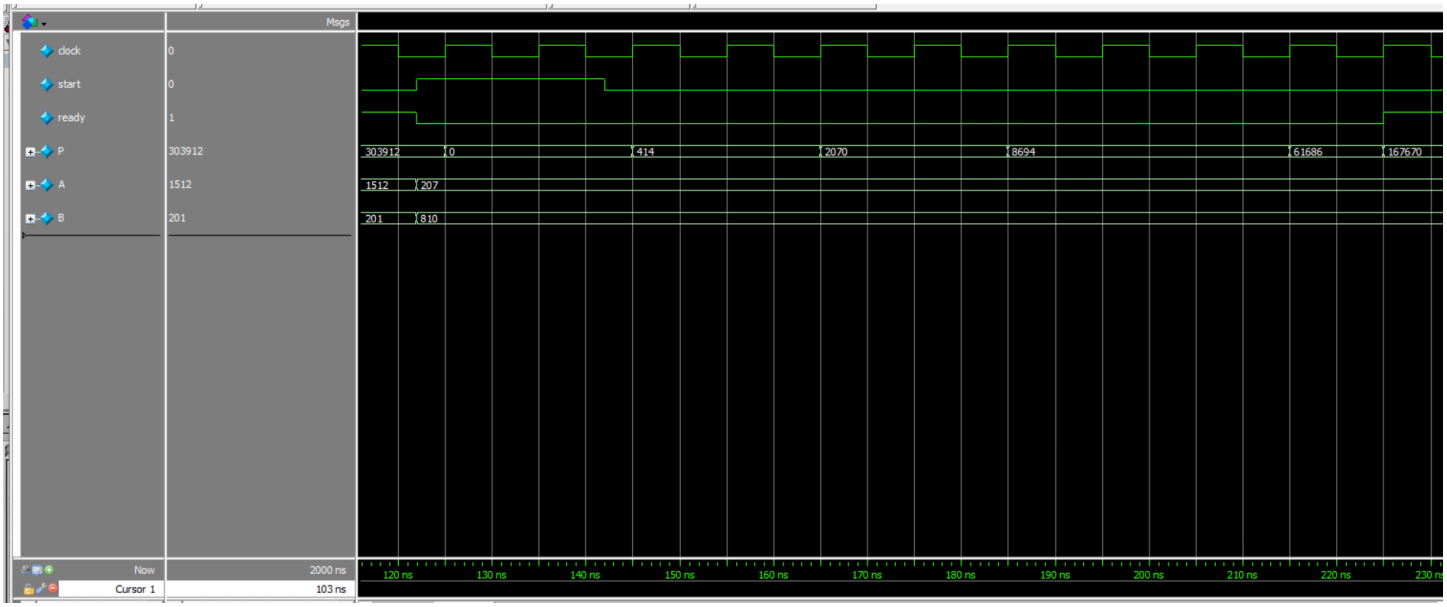
تست بنچ برای عرض 16 بیت:

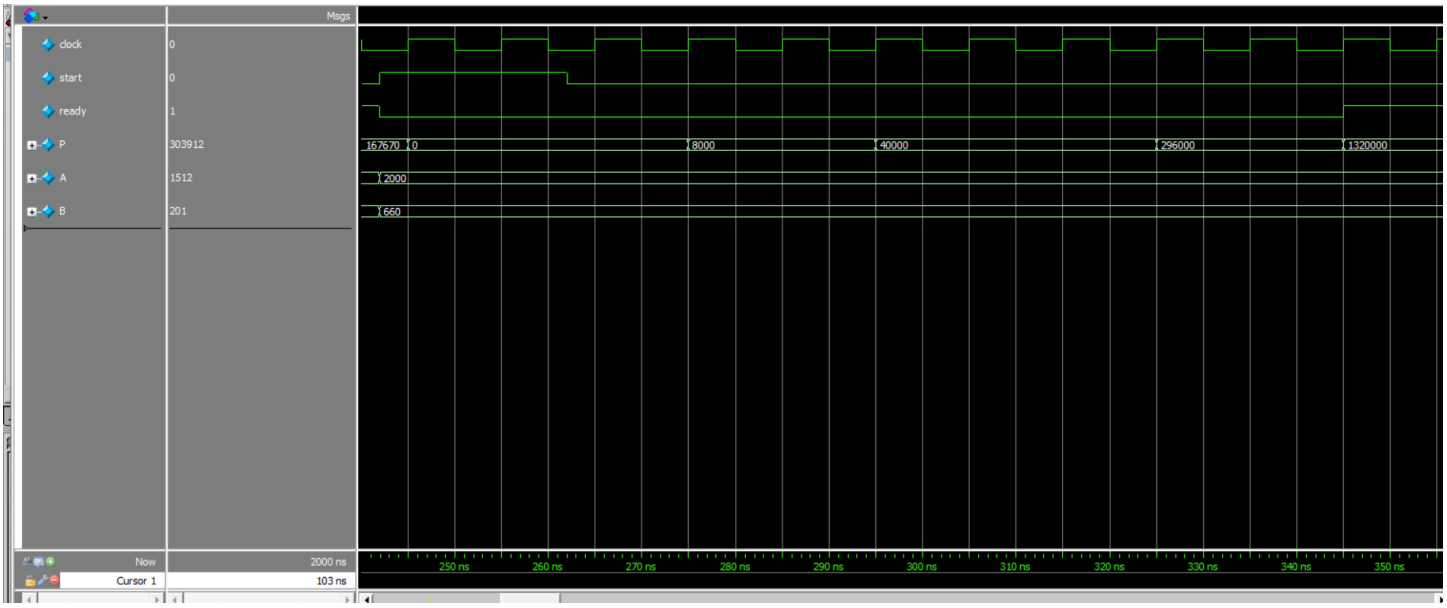
multiplier_tb2.sv

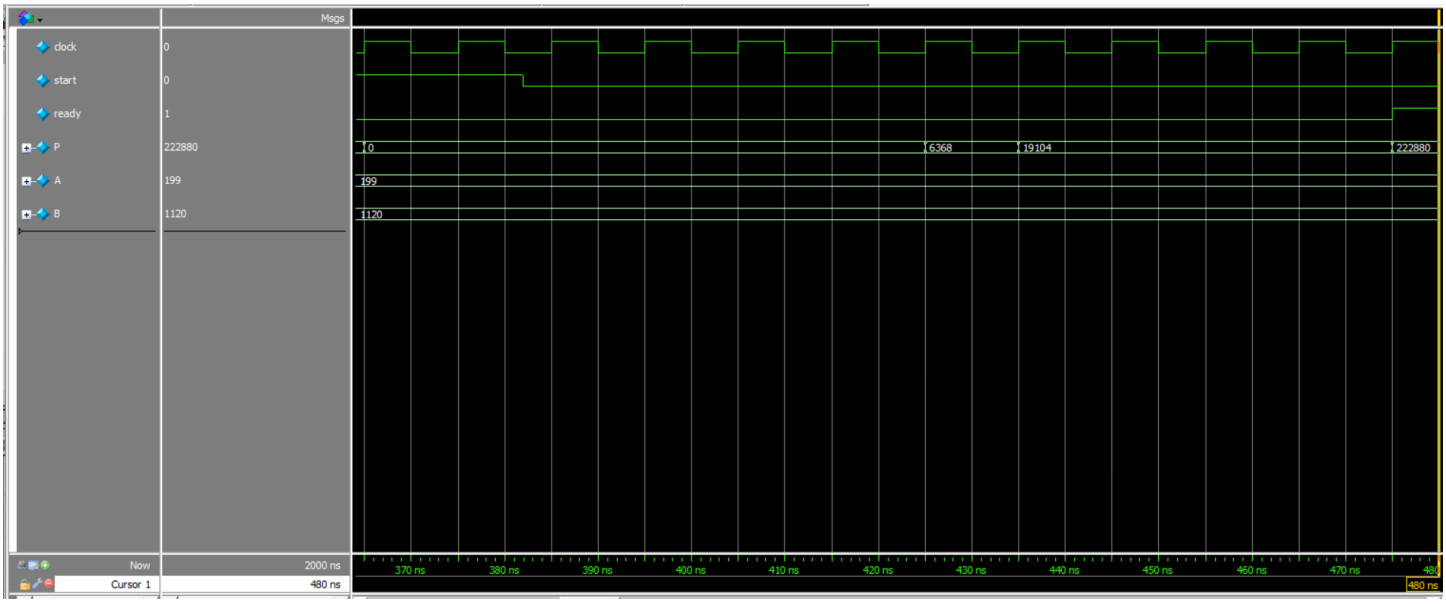
```
module multiplier_tb2();
    logic [31:0] P;
    logic[15:0] A,B;
    shift_add_multiplier #(16) dc(clock,start,A,B,P,ready);
    initial begin
        clock=0;
        start=0;
    end
    initial begin
    repeat(400) #5 clock=~clock;
    end
    initial begin
        #2
        start=1;
        A=1512;
        B=201;
        #20
        start=0;
        #100
        start=1;
        A=207;
        B=810;
        #20
        start=0;
        #100
        start=1;
        A=2000;
        B=660;
        #20
        start=0;
        #100
        start=1;
        A=199;
        B=1120;
        #20
        start=0;
        #100
        start=1;
        A=5000;
        B=500;
        #20
        start=0;

    end
endmodule
```

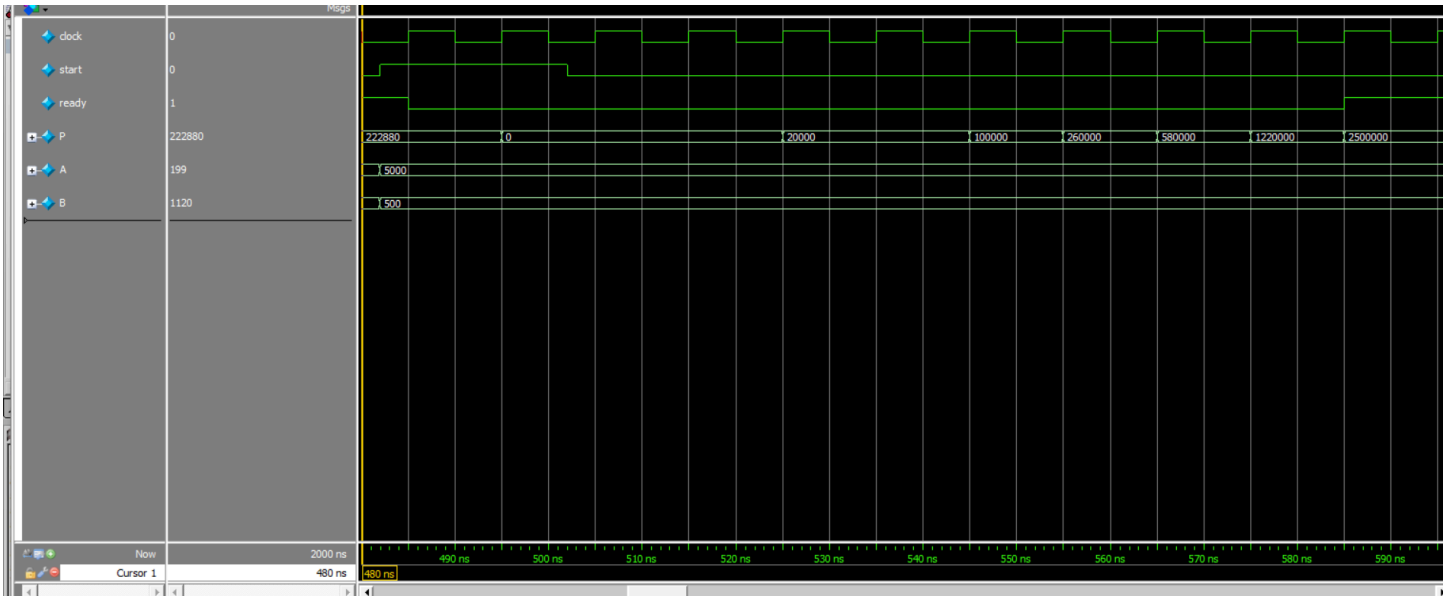
نتایج شبیه سازی:


Msgs
clock
start
ready
P
A
B
303912
1512
201
0
1512
13608
110376
303912
Now
2000 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns


Msgs
clock
start
ready
P
A
B
303912
1512
201
303912
0
414
2070
8694
61686
167670
1512
207
201
810
Now
2000 ns
Cursor 1
103 ns
120 ns
130 ns
140 ns
150 ns
160 ns
170 ns
180 ns
190 ns
200 ns
210 ns
220 ns


Msgs
clock
start
ready
P
A
B
303912
1512
201
167670
0
8000
40000
296000
1320000
2000
660
Now
2000 ns
Cursor 1
103 ns
250 ns
260 ns
270 ns
280 ns
290 ns
300 ns
310 ns
320 ns
330 ns
340 ns
350 ns





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 → 3.25

b=100.1 → 4.5

ab=1101 * 1001=01110101 → تا اعشار 3

01110.101=14.62

مثال دوم:

a=1101.0101 → 13.3125

b=0010.1011 → 2.6875

ab=11010101 * 00101011= **010001111000111** → تا اعشار 8

0100011.11000111=35.77734375

Reciprocal:

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

Initialize y_0 with an initial guess

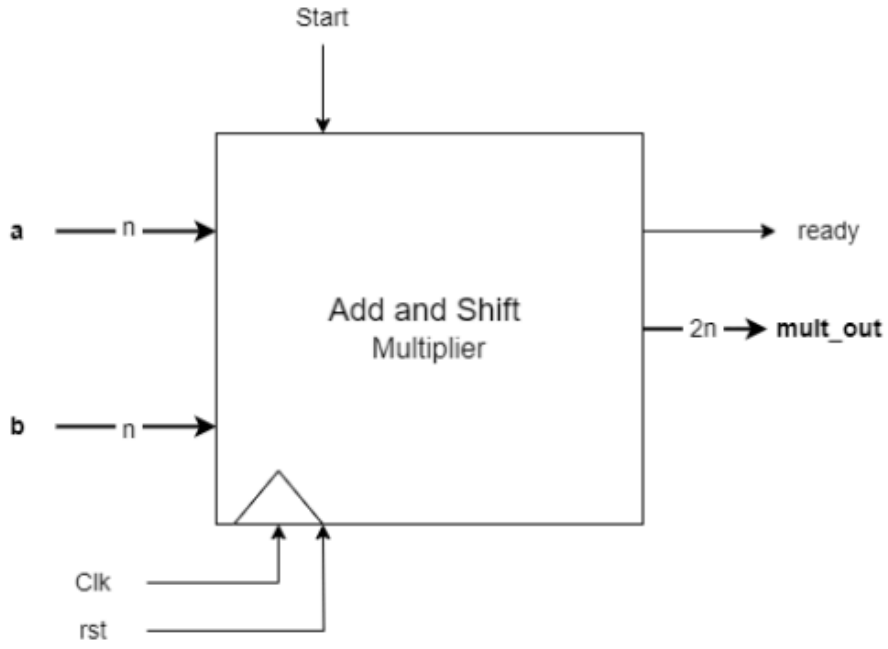
for i = [0:n-1] do {

$$y_{i+1} <= y_i * (2.0 - x_i * y_i)$$

}

xi همان عدد ورودی است.

y1=1

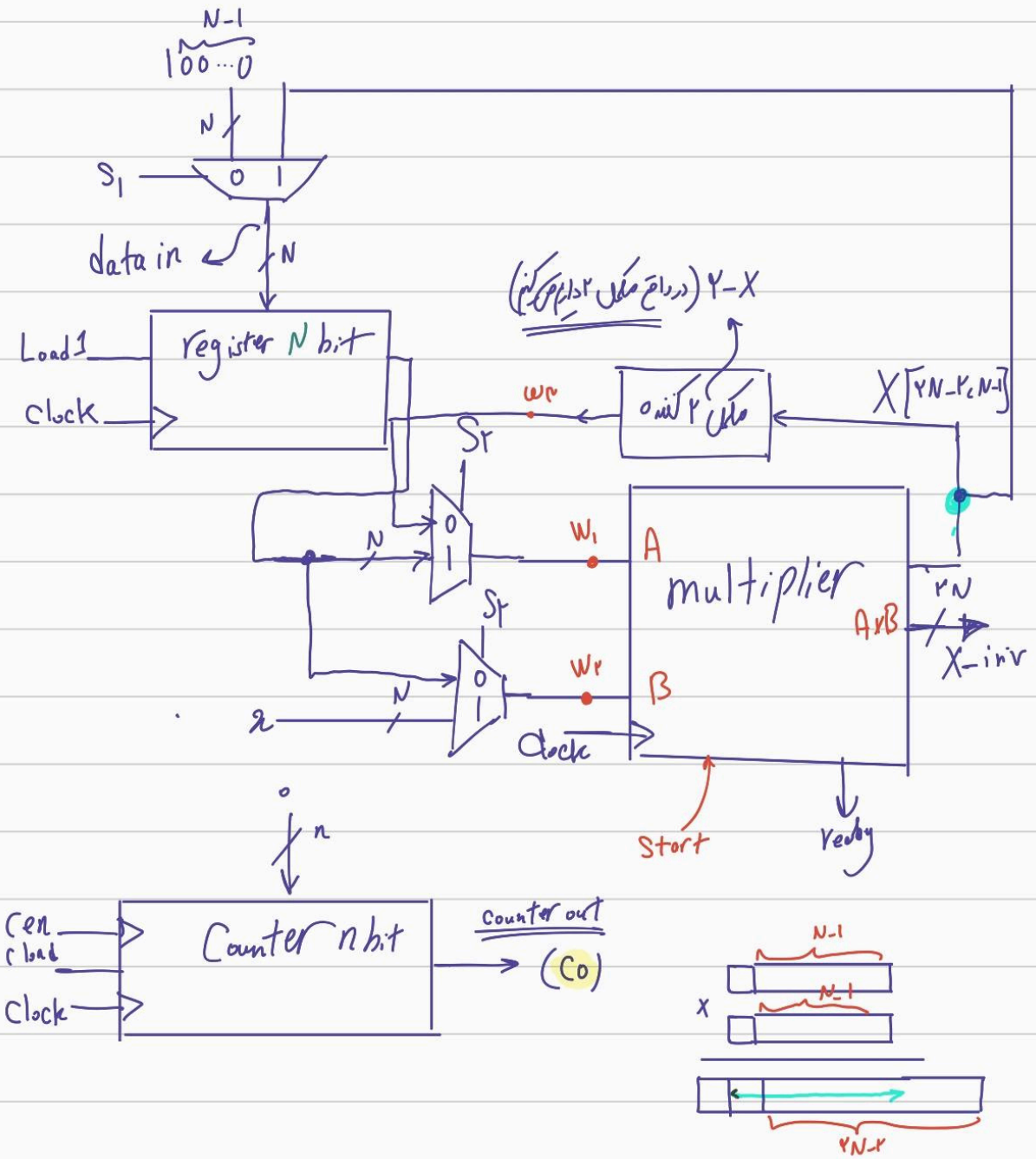


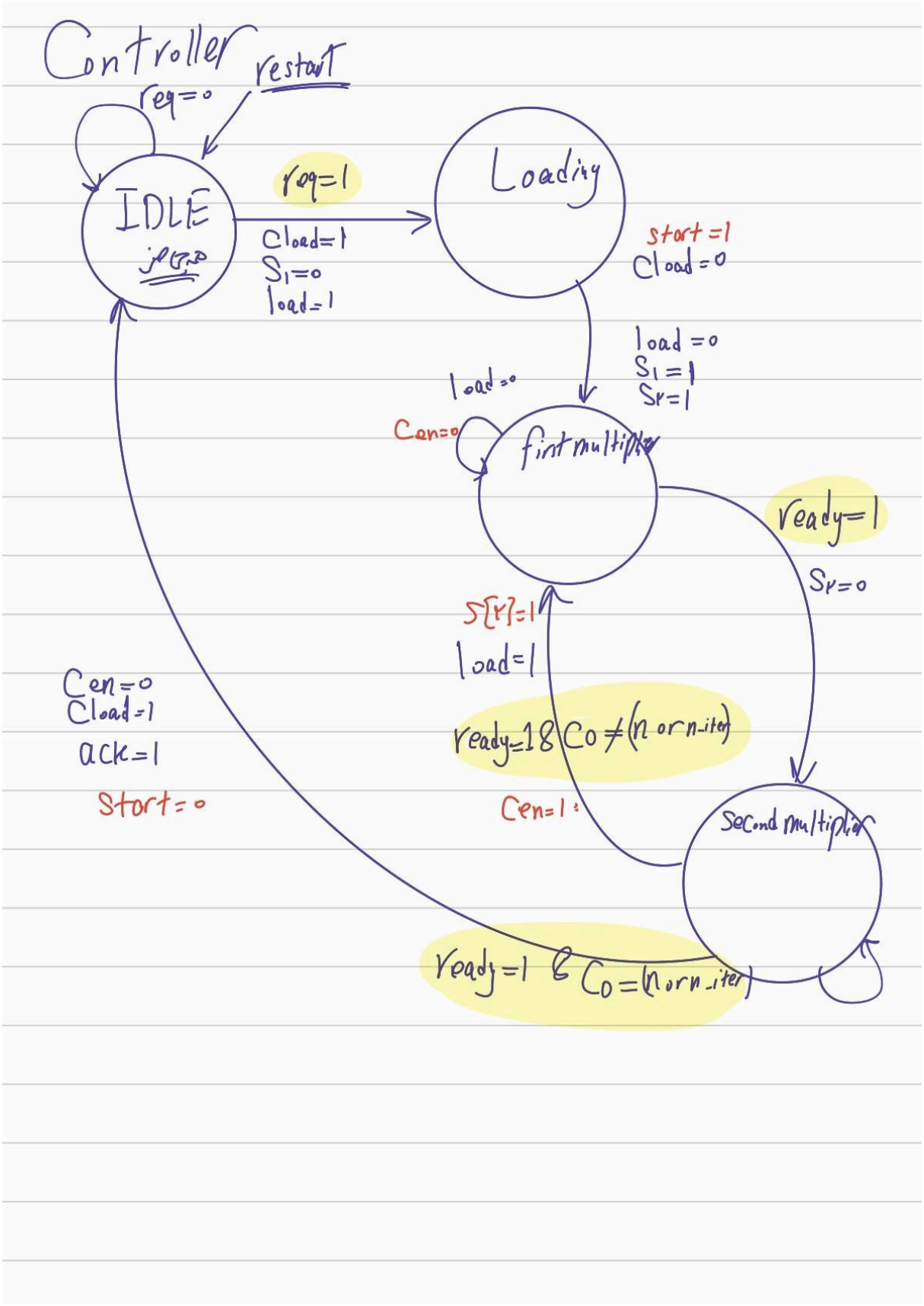
شکل ۱- دیاگرام ماژول Add and Shift

$\frac{10}{12} \not\frac{\alpha}{\varepsilon} \longrightarrow y_{i+1} = y_i\left(2 - \not\frac{10}{\alpha} y_i\right) \longrightarrow \simeq \frac{\varepsilon}{\alpha}$

Data path:

$y_1 = 1$





کد های سیستم وریلاگ:

reciprocal.sv

```
module reciprocal #(parameter N=8)(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

```

data_path_reciprocal.sv

```
module  data_path_reciprocal #(parameter N=8)(input logic clock,cload,cen,load,start,input logic [1:2]s
,input logic [N-1:0] x,output logic[2*N-1:0] x_inv,output logic ready,output logic [3:0] co);

    logic [N-1:0] r_out,data_in,w1,w2,w3;
    logic [2*N-1:0] mult_result;


    assign data_in=s[1]?mult_result[2*N-2:N-1]:{1'b1,{(N-1){1'b0}}};
    assign w1=s[2]?r_out:w3;
    assign w2=s[2]?x:r_out;
    assign w3=~mult_result[2*N-2:N-1]+1;
    assign x_inv=mult_result;
    register #(N) r(clock,load,data_in,r_out);
    shift_add_multiplier sam(clock,start,w1,w2,mult_result,ready);
    counter#(4) count(clock,cload,cen,4'b0000,co);
endmodule


```

controller_reciprocal.sv

```
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,
input logic[3:0] n_iter);
parameter [2:0] idle=0 ,loading=1,fisrt_multiplier=2, second_multiplier=3 ;
logic [2:0] p_state,n_state;
logic first_time_mult1, first_time_mult2;
always @(req or co or p_state or ready)begin
    case(p_state)
     idle:begin
       if(req)begin
          cload=1;
          cen=0;
          s[1]=0;
          load=1;
          n_state=loading;
       end
       else begin
           n_state=idle;
       end
     end
      loading:begin

          start=1;
          ack=0;
          cload=0;

          first_time_mult1=1;
          first_time_mult2=1;
          n_state=fisrt_multiplier;

      end


       fisrt_multiplier:begin
```

```
        end
          fisrt_multiplier:begin
                if(first_time_mult1)begin
                      first_time_mult1=0;
                        s[1]=1;////////
                         s[2]=1;
                      load=0;
                      cen=0;

                end
                else begin

                        if(ready)begin
                             s[2]=0;
                             first_time_mult2=1;
                             n_state=second_multiplier;

                        end


                end


          end
          second_multiplier:begin

                if(first_time_mult2)begin
                      first_time_mult2=0;
                end
                else begin

                     if(ready==1&&(co==n_iter))begin

                             cen=0;
```

```
always @(req or co or p_state or ready)begin
    case(p_state)
     end
      end
        second_multiplier:begin
                    if(ready==1&&(co==n_iter))begin

                            cen=0;
                            cload=1;
                            ack=1;
                            start=0;

                            n_state=idle;

                    end
                    else if(ready)begin

                     cen=1;
                     load=1;
                     s[2]=1;

                       first_time_mult1=1;
                       n_state=fisrt_multiplier;

                    end


                end


          end
            default: begin
                  n_state = idle;
              end
    endcase
end
```

تست بنچ برای ورودی های مختلف:

X_inv=14 رقم اعشار

X=7 رقم اعشار

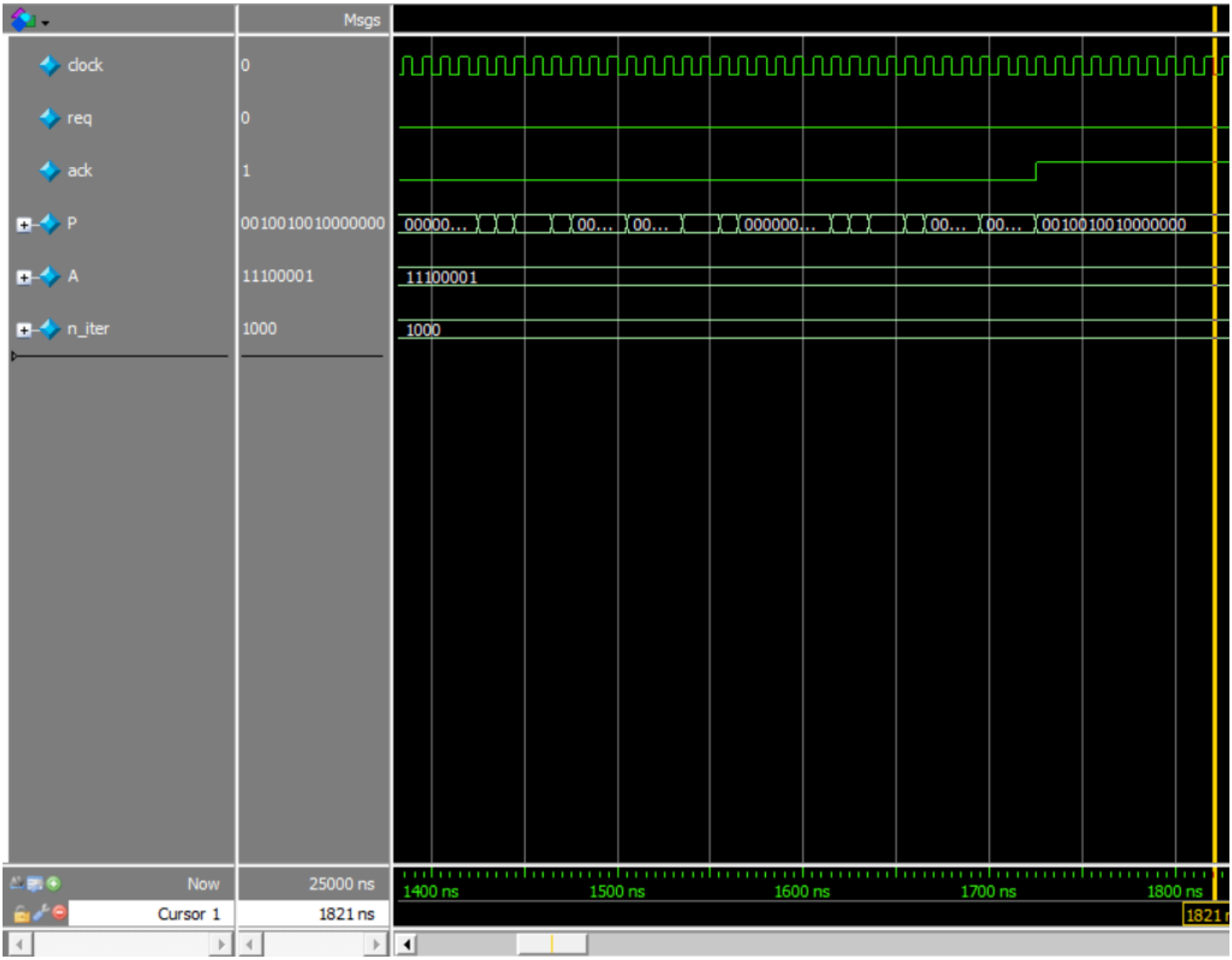
reciprocal_tb.sv

```
module reciprocal_tb();

    logic clock,req,ack;
    logic [15:0] P;
    logic[7:0] A=8'b11100001;
    logic [3:0] n_iter=8;
    reciprocal #(8) rep(clock,req,A,n_iter,P,ack);
    initial begin
        clock=0;
        req=0;
        #30
        req=1;
        #200
        req=0;
        #2000
        A=8'b11110111;
        req=1;
        #200
        req=0;
        #2000
        A=8'b10010100;
        req=1;
        #200
        req=0;
        #2000
        A=8'b10100000;
        req=1;
          #200
        req=0;


    end
    initial begin
    repeat(5000) #5 clock=~clock;
    end
endmodule
```

x=11100001 → 1.7578125

مقدار واقعی معکوس: 0.56888888888888

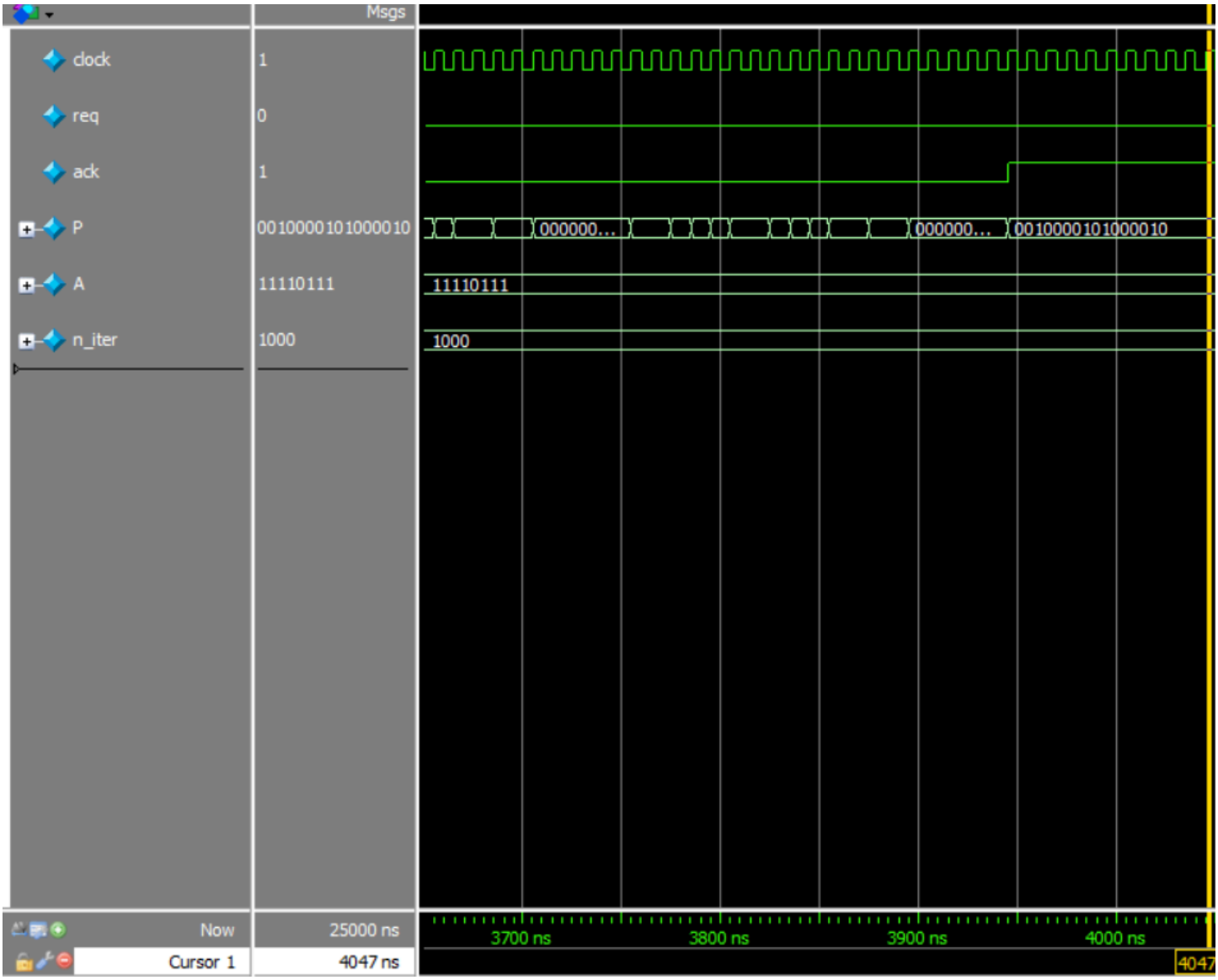
X_inv=0010010010000000 → 0.5703125



x=11110111 → 1.9296875

مقدار واقعی معکوس: 0.5182186234817

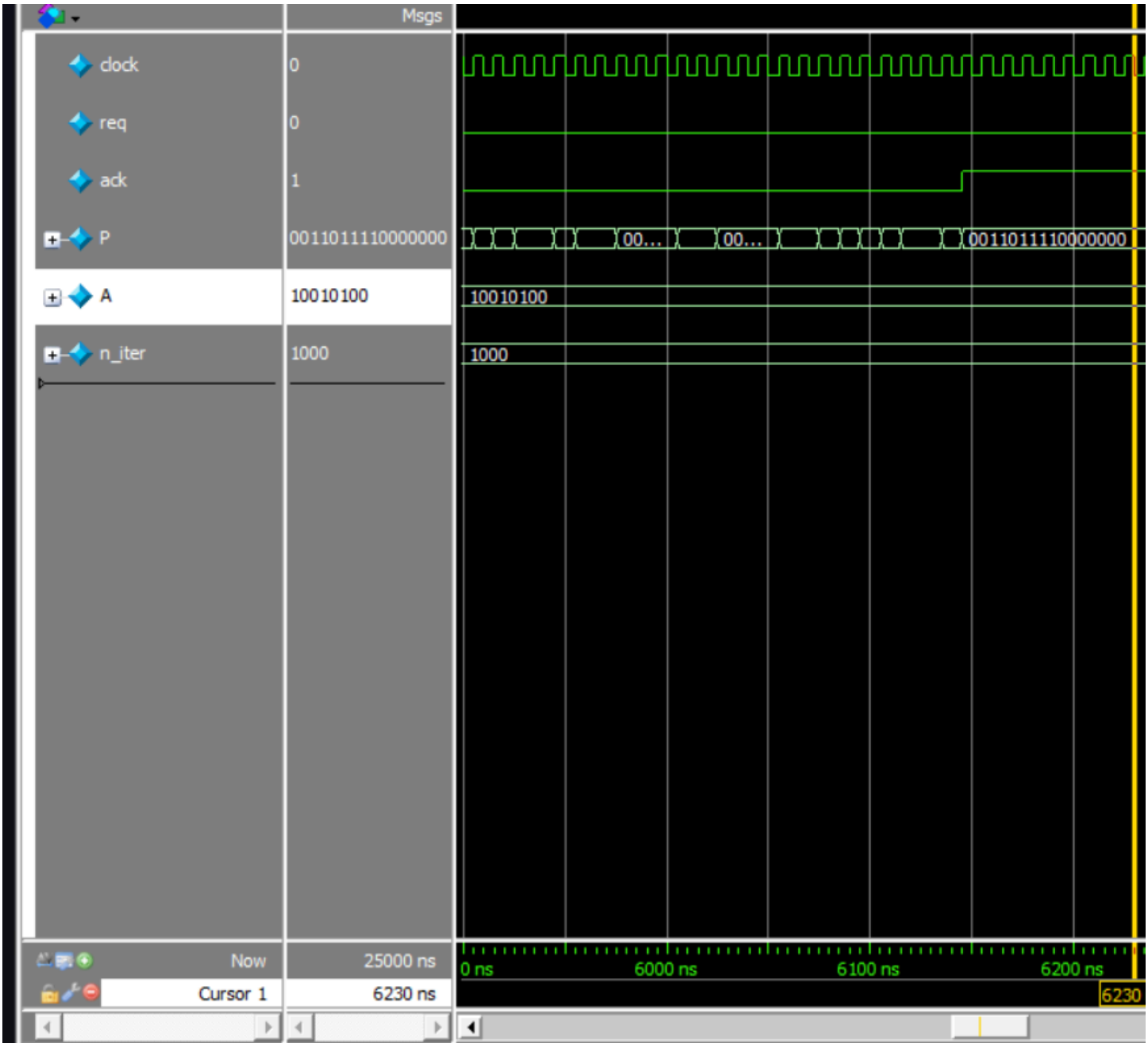
X_inv=00100001010000010 → 0.5196533203125



x=10010100 → 1.15625

مقدار واقعی معکوس: 0.8648648648

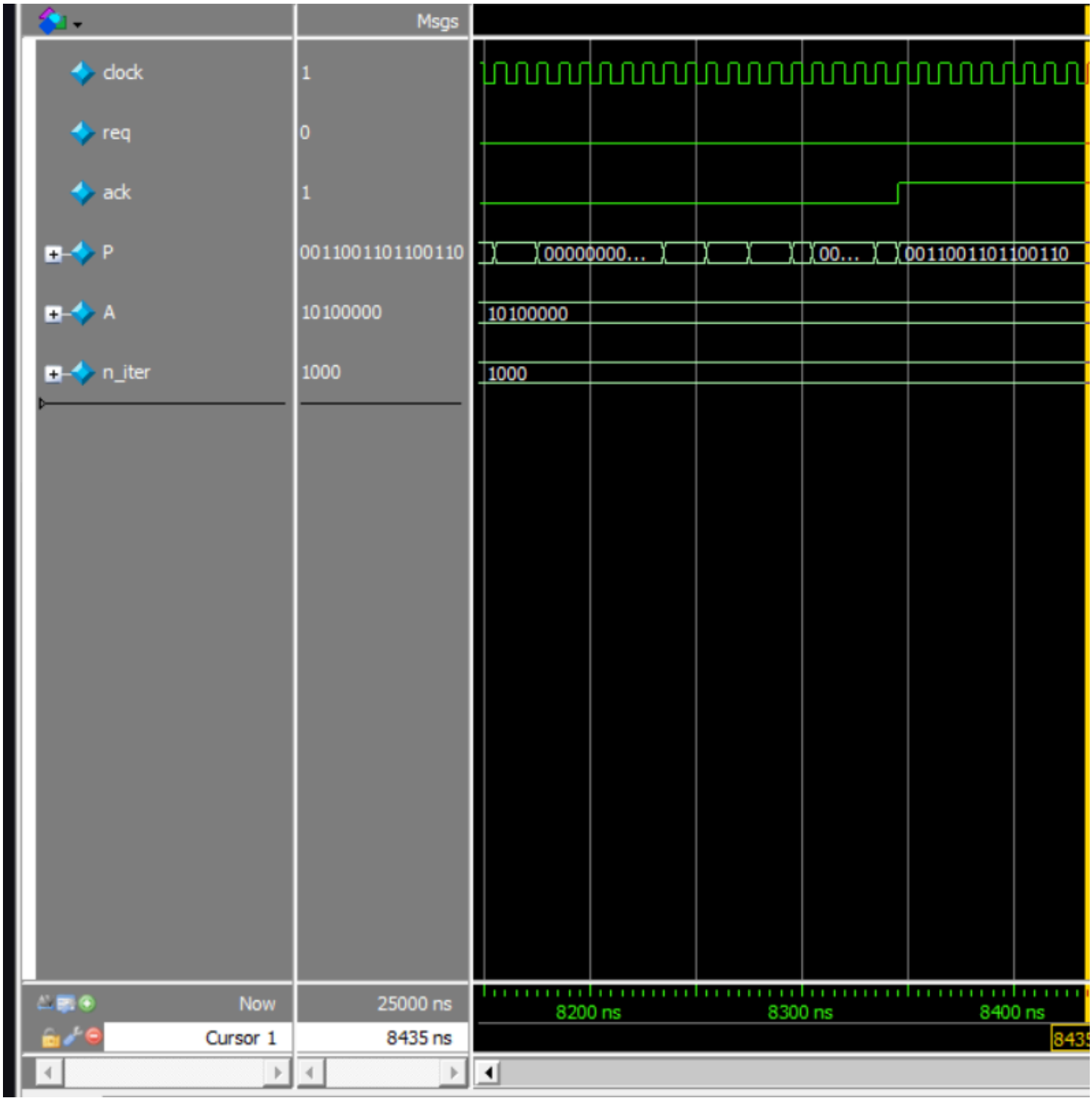
X_inv=0011011110000000 → 0.8671875



x=10100000 → 1.25

مقدار واقعی معکوس: 0.8

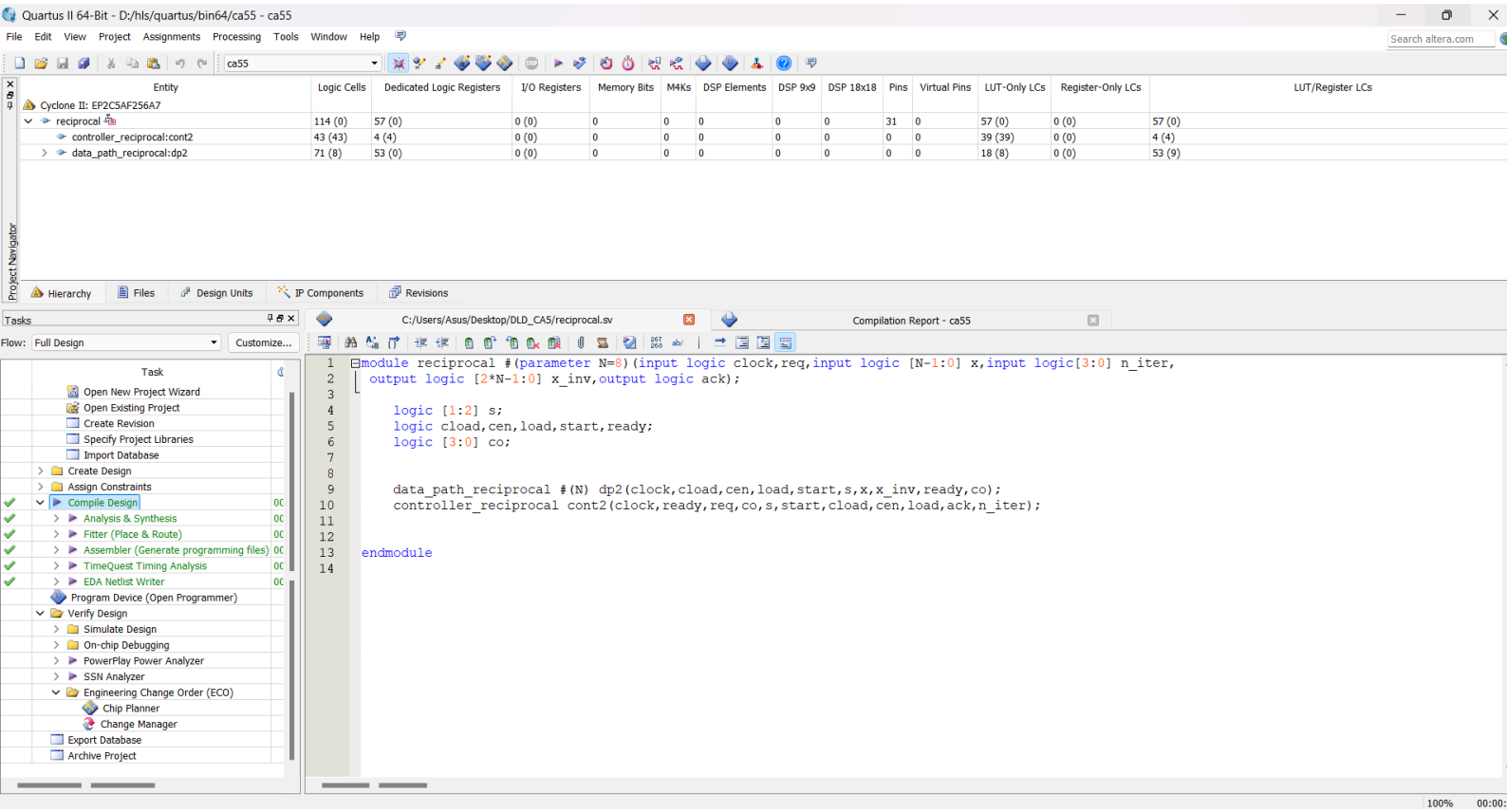
X_inv=0011001101100110 → 0.803100585937 5

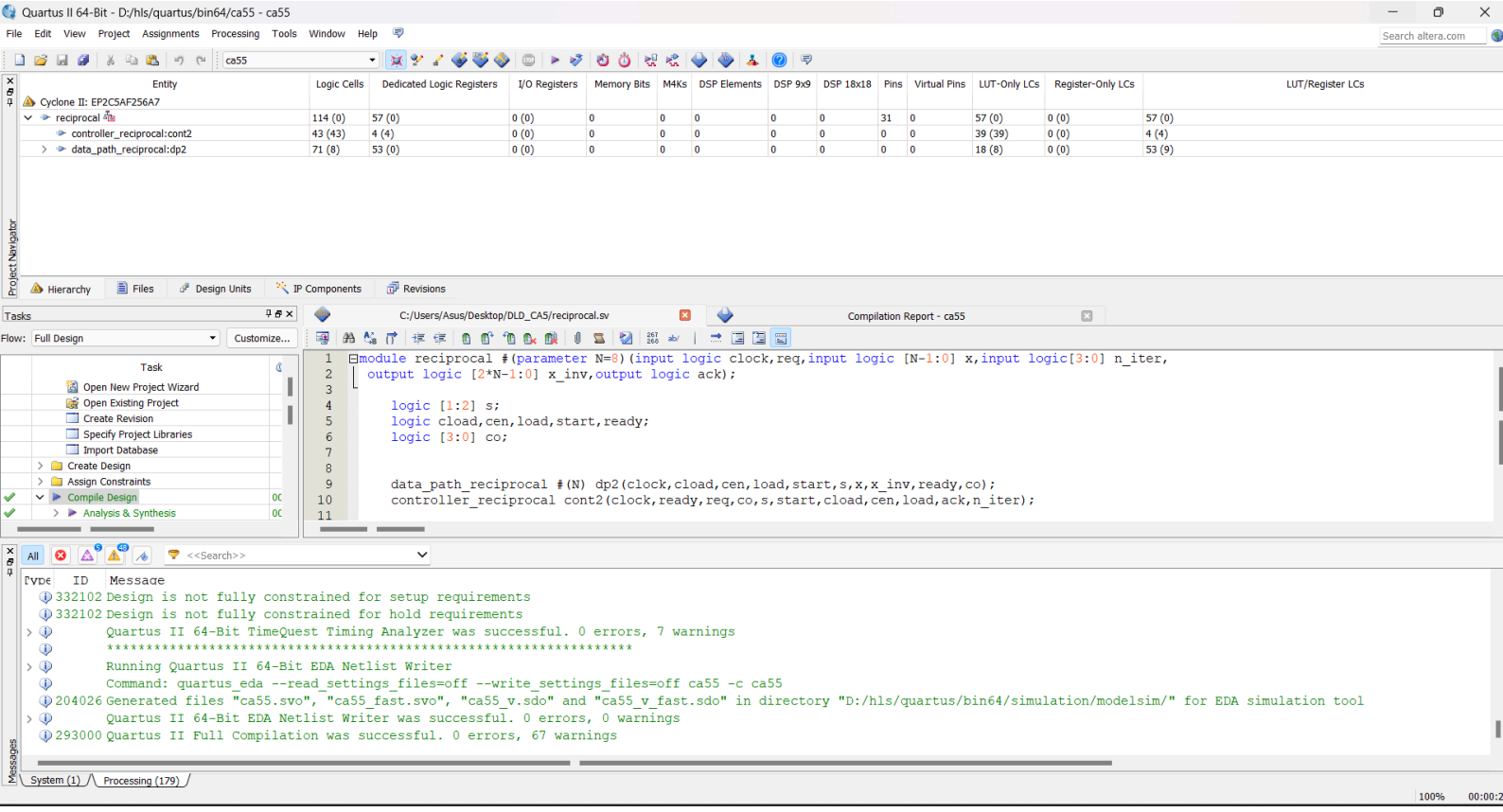


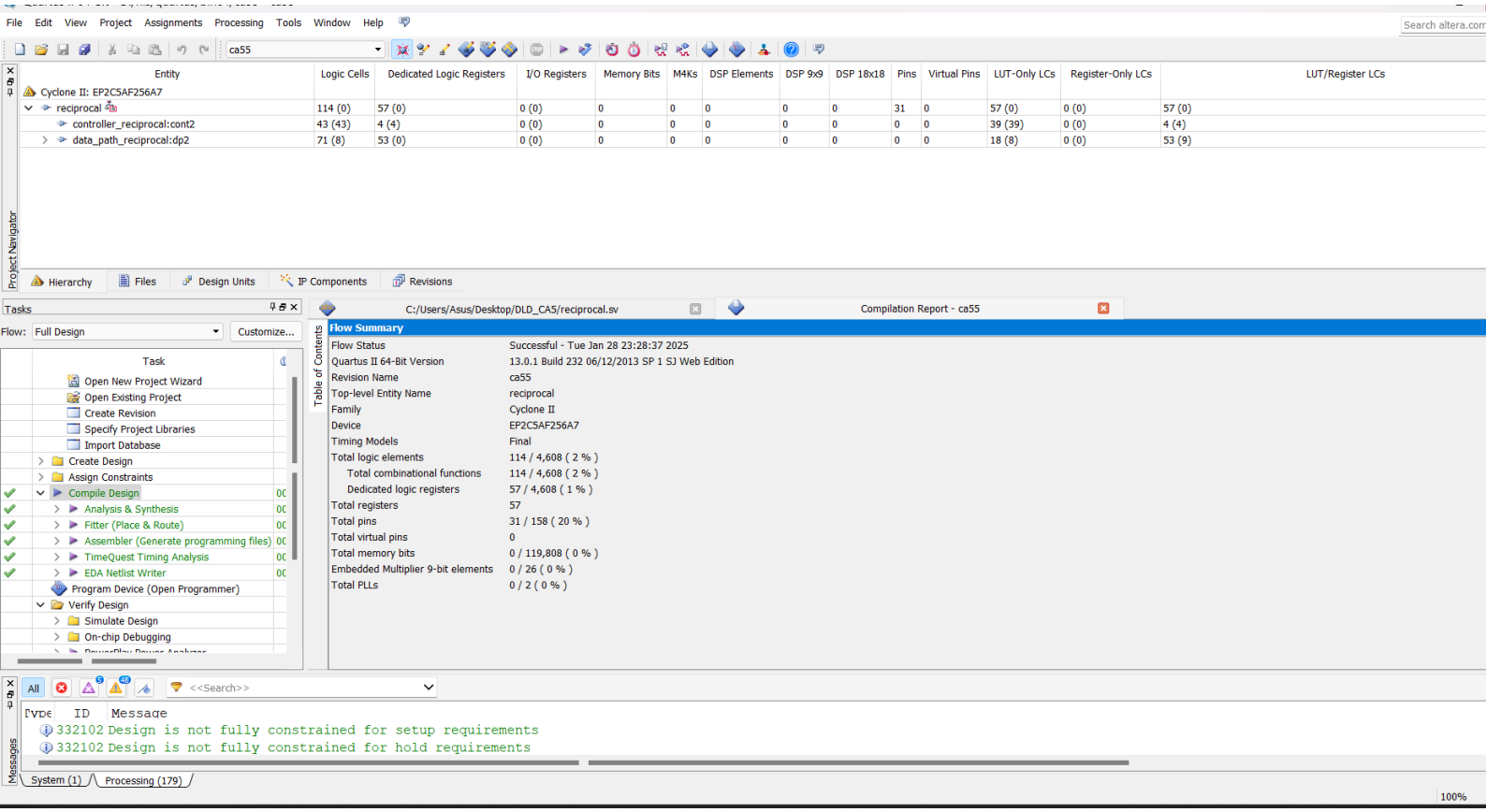
در بخش آخر با نرم افزار Quartus II یک پروژه جدید میسازیم.

و با دیواس سایکلون 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:

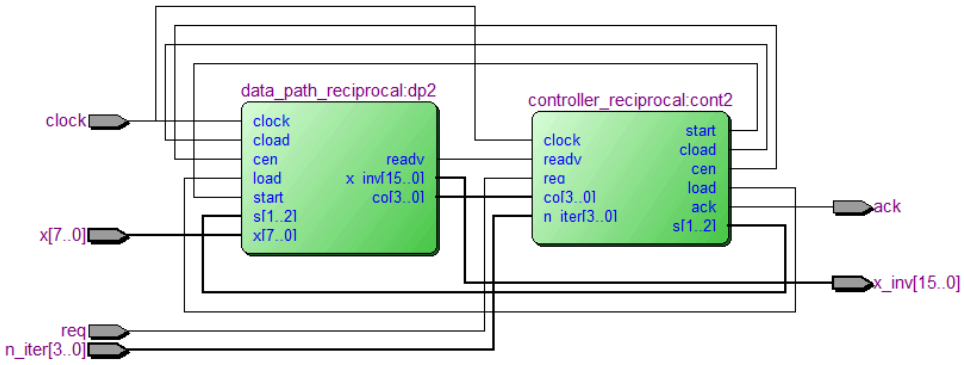






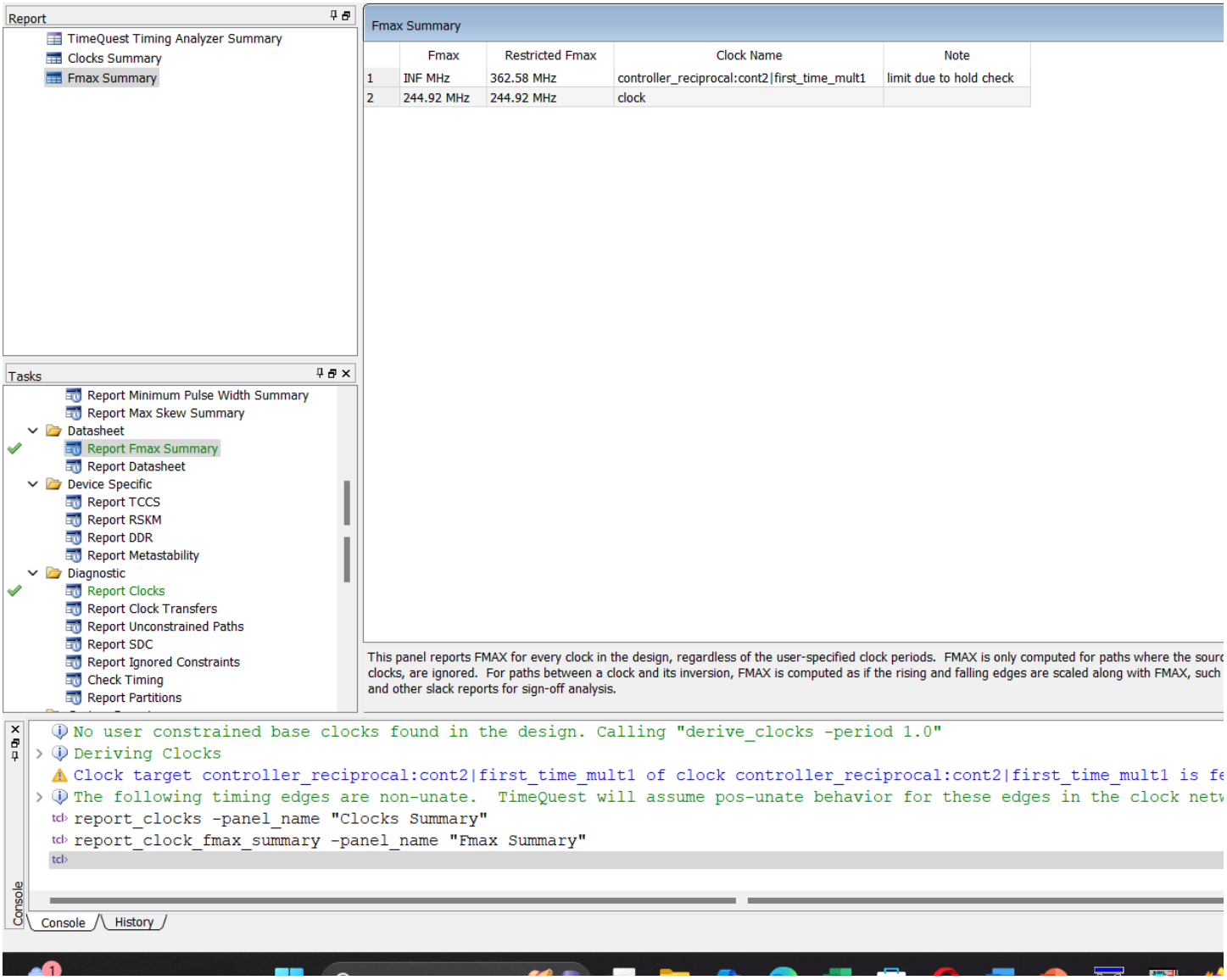
در بخش summary با فرض 8 بیتی بودن ماژول، از 57 رجیستر استفاده شده است که در اینجا هر فلیپ فلاپ را یک رجیستر در نظر گرفته است در واقع.

این هم شکل RTL کلی ماژول:

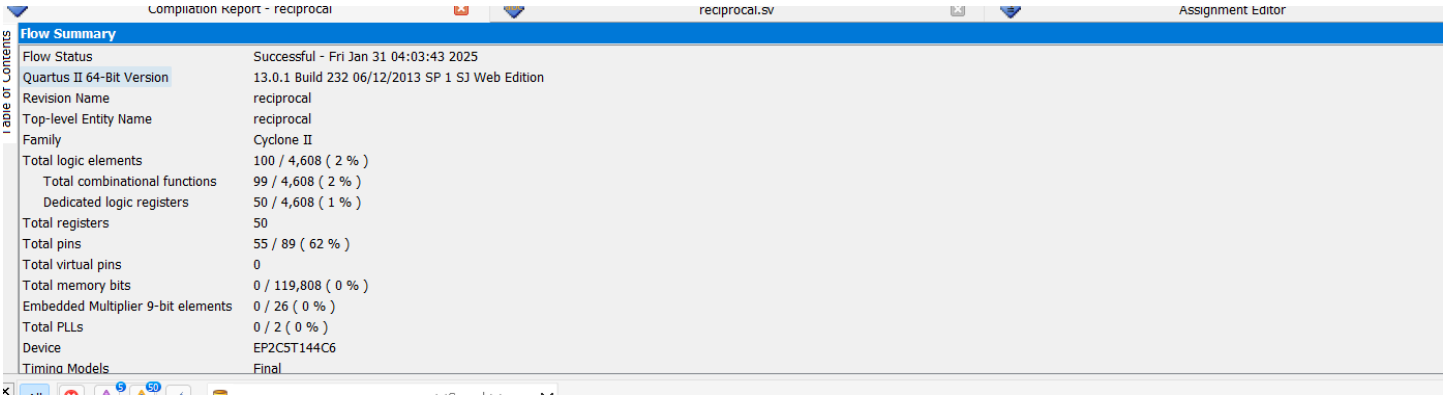


برای پیدا کردن FMAX که از فرمول زیر به دست میاد به بخش ANALYSIS TIME رفته و آن را بررسی میکنیم که برابر 244.92 مگاهرتز است.

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



برای حالت 16 بیتی:



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

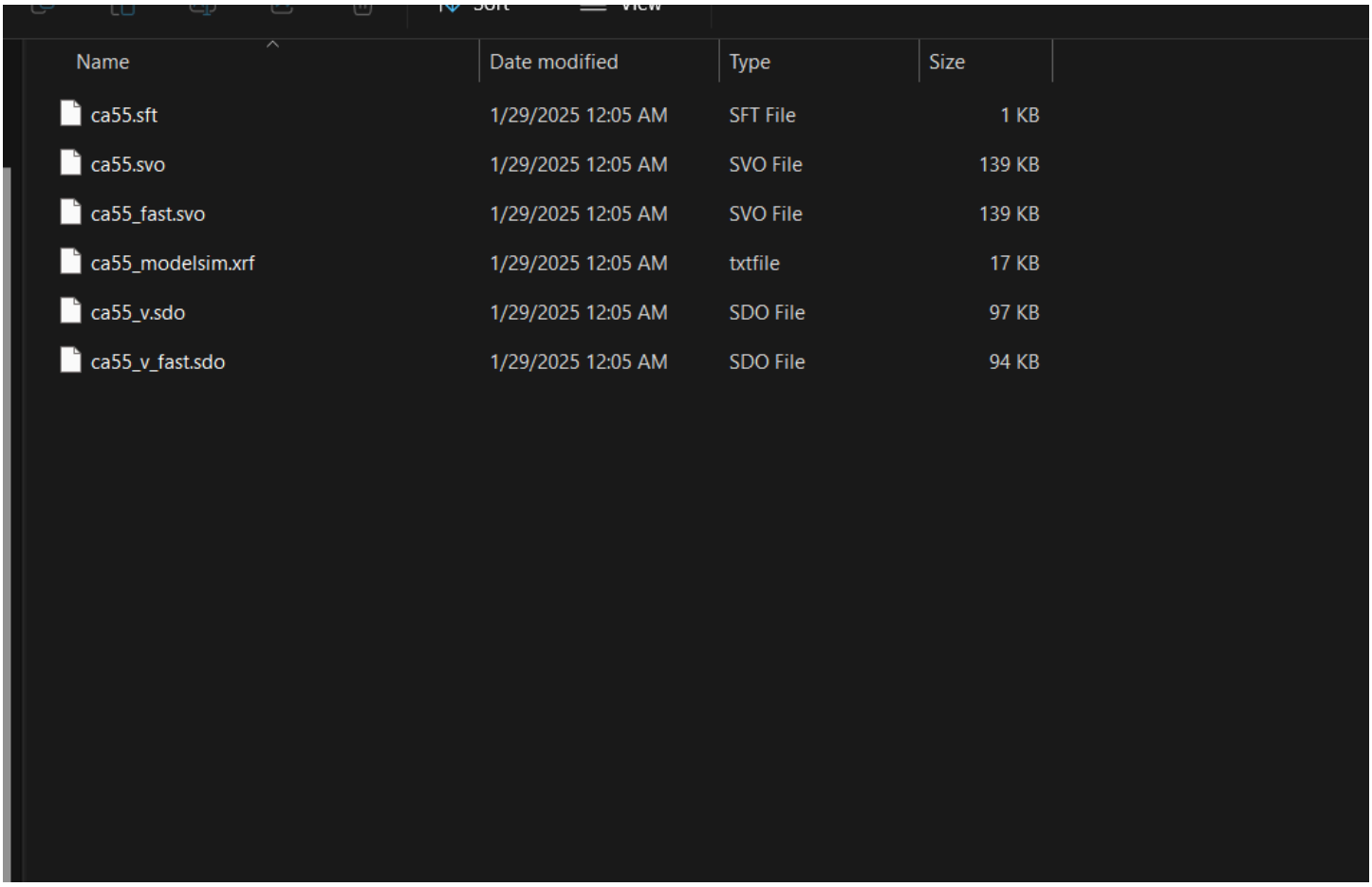
Flow Status	Successful - Fri Jan 31 04:06:55 2025
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	reciprocal
Top-level Entity Name	reciprocal
Family	Cyclone II
Total logic elements	93 / 4,608 (2 %)
Total combinational functions	93 / 4,608 (2 %)
Dedicated logic 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

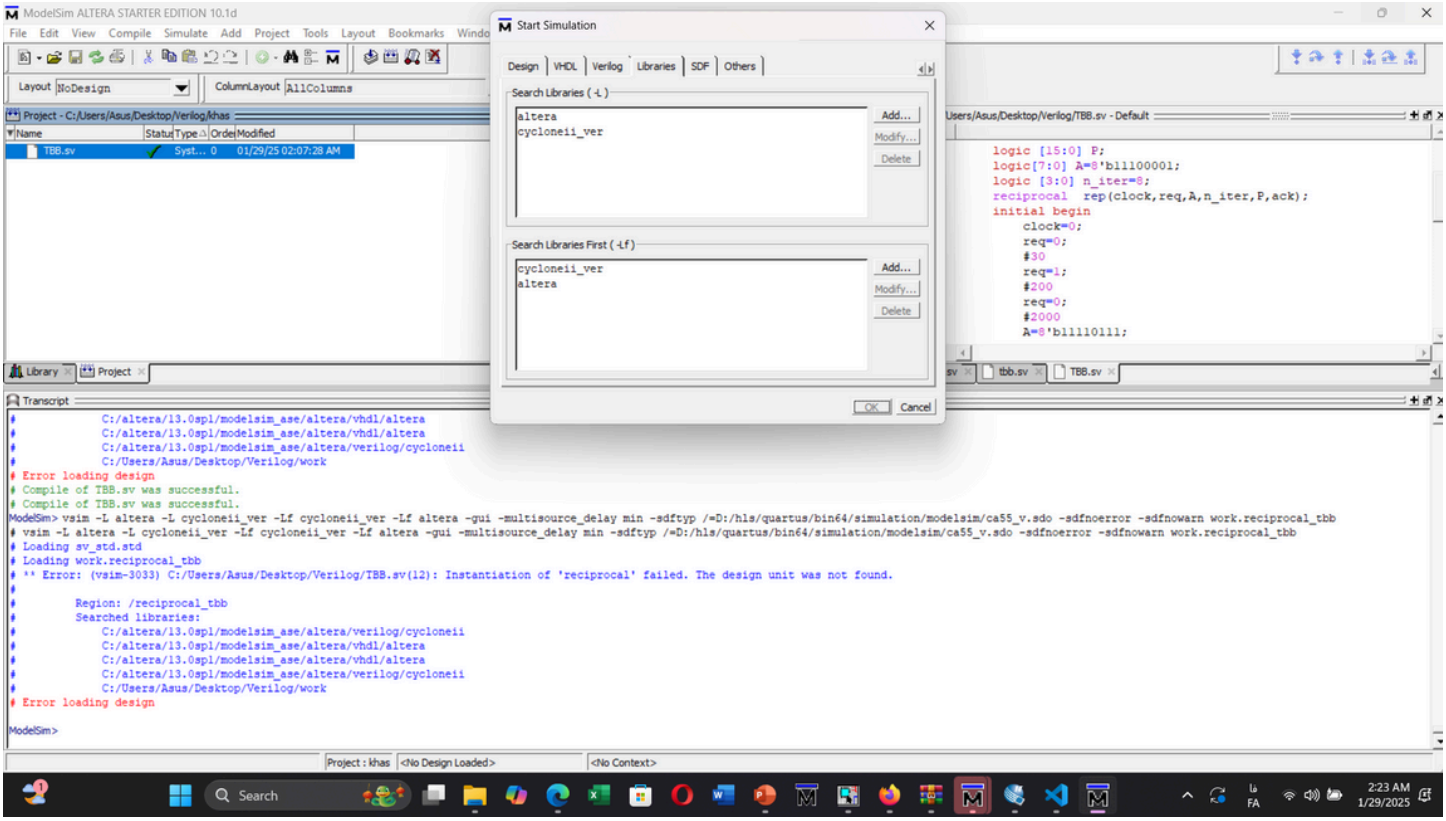
Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	297.27 MHz	297.27 MHz	clock	
2	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.





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)