دانشگاه صنعتی شریف
دانشکده مهندسی برق

آزمایشگاه FPGA

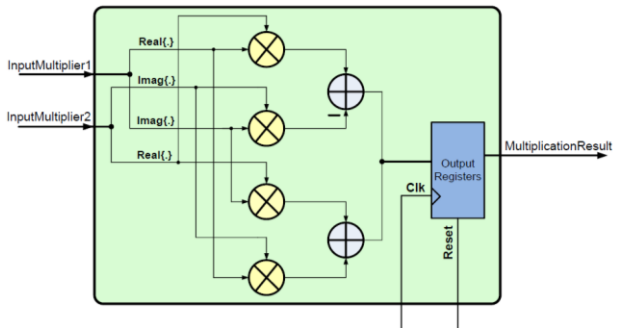
گزارش آز 4

گروه شنبه
محمدرضا حاجی بابایی
99101416
امیرحسین یاری
99102507

آزمایش 1)

طبق دستور کار پروژه را می سازیم و کد ضرب کننده را با توجه به بلوک دایاگرام مدار می نویسیم:

```
`timescale 1ns / 1ps
module ComplexMultiplier(
    input Clk,
    input Reset,
    input [15:0] InputMultiplier1,
    input [15:0] InputMultiplier2,
    output reg [33:0] MultiplicationResult
    );
   wire [15:0] ac;
   wire [15:0] bd;
   wire [15:0] ad;
   wire [15:0] bc;
   wire [15:0] ac2;
   wire [15:0] bd2;
   wire [15:0] ad2;
   wire [15:0] bc2;

   wire [15:0] ImagPart;
   wire [15:0] RealPart;
   wire [7:0]a;
   wire [7:0]b;
   wire [7:0]c;
   wire [7:0]d;

   assign a = InputMultiplier1[15] ? ~InputMultiplier1[15:8] + 1 : InputMultiplier1[15:8];
   assign b = InputMultiplier1[7] ? ~InputMultiplier1[7:0] + 1 : InputMultiplier1[7:0];
   assign c = InputMultiplier2[15] ? ~InputMultiplier2[15:8] + 1 : InputMultiplier2[15:8];
   assign d = InputMultiplier2[7] ? ~InputMultiplier2[7:0] + 1 : InputMultiplier2[7:0];

   assign ac = a * c;
   assign bd = b * d;
   assign ad = a * d;
   assign bc = b * c;

   assign ac2 = InputMultiplier1[15] ^ InputMultiplier2[15] ? ~ac + 1 : ac;
   assign bd2 = InputMultiplier1[7] ^ InputMultiplier2[7] ? ~bd + 1 : bd;
   assign ad2 = InputMultiplier1[15] ^ InputMultiplier2[7] ? ~ad + 1 : ad;
   assign bc2 = InputMultiplier1[7] ^ InputMultiplier2[15] ? ~bc + 1 : bc;

   assign ImagPart = ad2 + bc2;
   assign RealPart = ac2 - bd2;
   always @(posedge Clk or negedge Reset)
   begin
      if(!Reset)
      begin
         MultiplicationResult <= 34'b0;
      end
      else
      begin
         MultiplicationResult <= {RealPart[15],RealPart,ImagPart[15],ImagPart};
      end
   end
endmodule
```

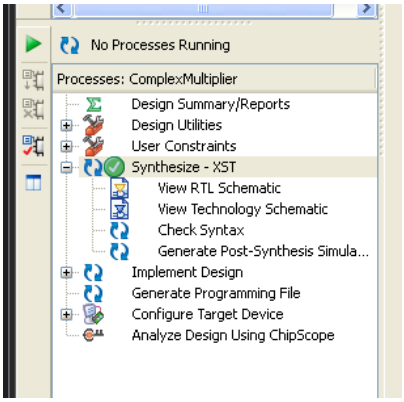


در این کد با توجه به فرمول زیر ضرب قسمت های حقیقی و موهومی اعداد ورودی انجام شده و در خروجی نوشته می‌شود.

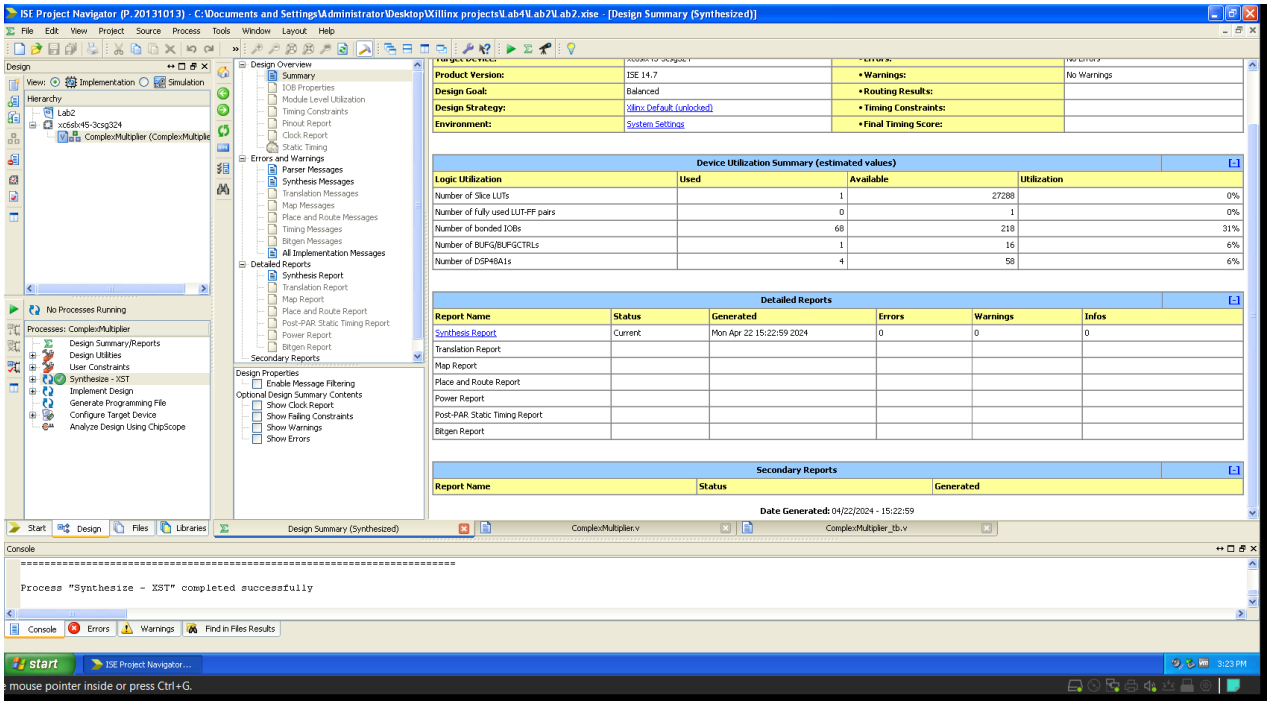
$$(a + bj) * (c + dj) = (ac - bd) + (ad + bc)j$$

در خروجی نیز 17 بیت اول برای قسمت موهومی و 17 بیت دوم برای قسمت حقیقی است. همچنین چون اعداد علامت‌دار هستند در هنگام ضرب اعداد ابتدا آن ها را مثبت می کنیم سپس آن ها را با توجه به علامت ورودی ها به فرمت 2's complement در می‌آوریم.

در قسمت بعد مدار را سنتز می کنیم:

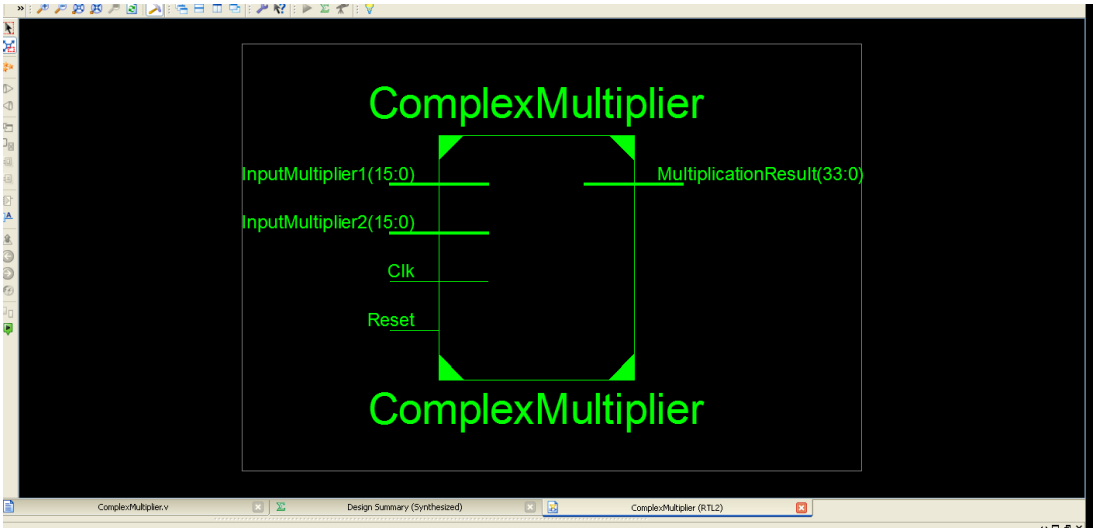


همانطور که مشخص است سنتیز مدار به درستی انجام شده است. در این قسمت تنظیمات سنتز را تغییری نمی دهیم.



هیچ گونه ارور و وارنینگی وجود ندارد.

در این قسمت RTL schematic را نمایش می دهیم:



آزمایش 2)

کد آزمایش 1 را با استفاده از تست بنچ زیر شبیه سازی می کنیم و نتیجه را در هر بخش نمایش می دهیم:

```
`timescale 1ns / 1ps
module ComplexMultiplier_tb;
   // Inputs
   reg Clk;
   reg Reset;
   reg [15:0] InputMultiplier1;
   reg [15:0] InputMultiplier2;
   reg [15:0] inputs[0:7];
   reg [33:0] outputs[0:3];
   reg [4:0]error;
   // Outputs
   wire [33:0] MultiplicationResult;

   // Instantiate the Unit Under Test (UUT)
   ComplexMultiplier uut (
      .Clk(Clk),
      .Reset(Reset),
      .InputMultiplier1(InputMultiplier1),
      .InputMultiplier2(InputMultiplier2),
      .MultiplicationResult(MultiplicationResult)
   );

   always #10 Clk = ~Clk;
   initial begin
      // Initialize Inputs
      Clk = 0;
      Reset = 0;
      error = 0;
      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab2/Data/inputs.txt",inputs);
      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab2/Data/outputs.txt",outputs);
      #20
      Reset = 1;
      InputMultiplier1 = inputs[0];
      InputMultiplier2 = inputs[1];
      #20
      if(MultiplicationResult != outputs[0])
         error = error + 1;
      #20
      InputMultiplier1 = inputs[2];
      InputMultiplier2 = inputs[3];
      #20
      if(MultiplicationResult != outputs[1])
         error = error + 1;
      InputMultiplier1 = inputs[4];
      InputMultiplier2 = inputs[5];
      #20
      if(MultiplicationResult != outputs[2])
         error = error + 1;
      InputMultiplier1 = inputs[6];
      InputMultiplier2 = inputs[7];
      #20
      if(MultiplicationResult != outputs[3])
         error = error + 1;
      if(error == 0)
         $display("results are true.");
      else
         $display("the number of error is %d",error);
   end

endmodule
```

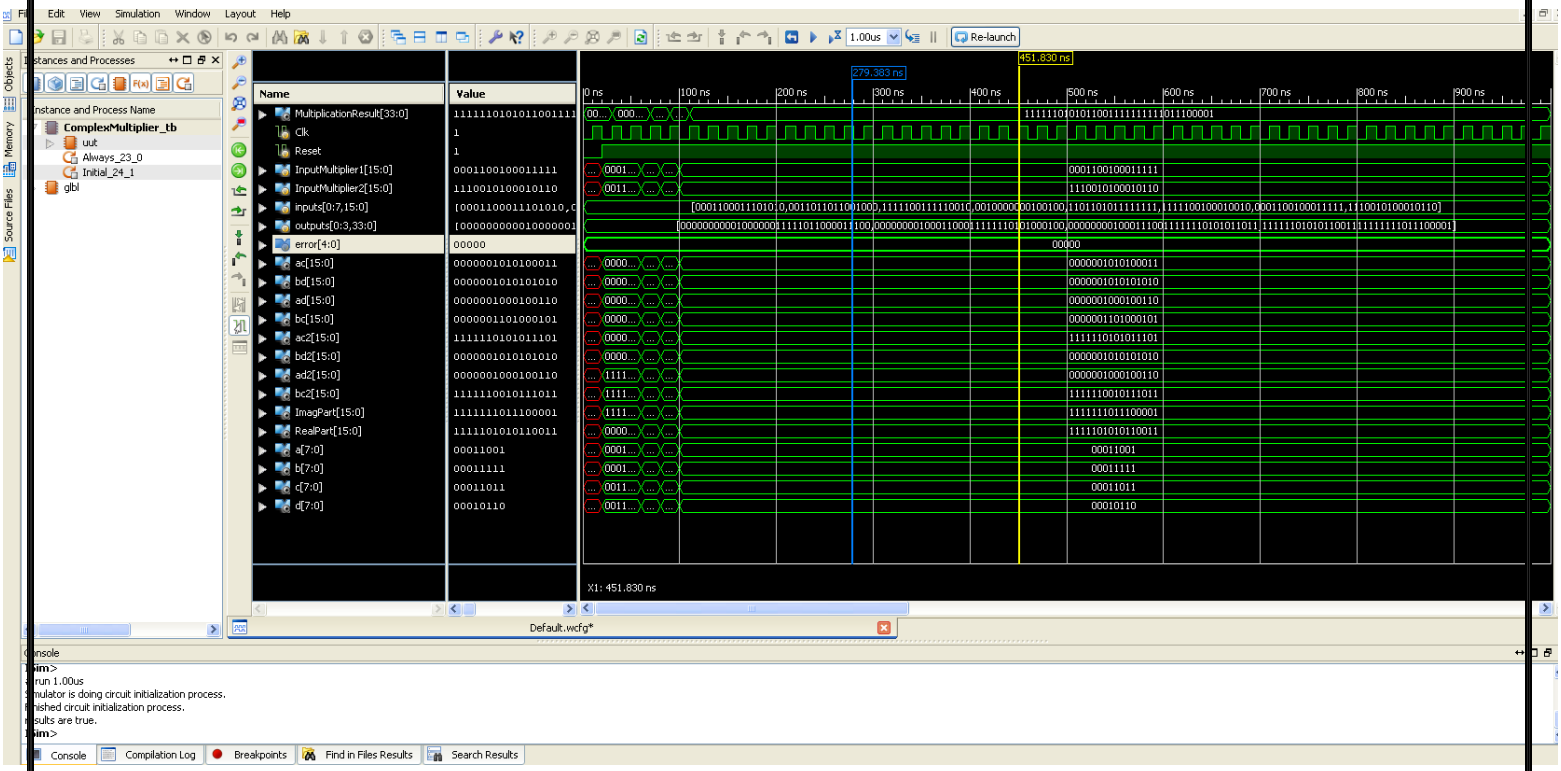
در این کد ورودی هایی که با استفاده از متلب ساخته شده اند را به تست بنچ می دهیم و خروجی ماژول را با نتایج به دست آمده از کد متلب مقایسه می کنیم و تعداد نابرابری های خروجی را در متغیر error ذخیره می کنیم.ورودی ها و نتایج آن ها به ترتیب در فایل های inputs و results ذخیره شده اند.

کد متلب:

```
random_numbers = randi([-60, 60], 1, 16);
state01_input = [];
for j = 1:2:16
    state01_input = [state01_input;[dec2bin(random_numbers(j),8),dec2bin(random_numbers(j + 1),8)]];
end
results = [];
for j = 1:4:16
    results = [results ; (random_numbers(j) + random_numbers(j + 1) * 1i)*(random_numbers(j + 2) + random_numbers(j + 3) * 1i)];
end
results_bin = [];
for i = 1:16/4
   results_bin = [results_bin ; [dec2bin(real(results(i)),17),dec2bin(imag(results(i)),17)]];
end
fileID_input = fopen('inputs.txt', 'w');
for i = 1:size(state01_input,1)
    for j = 1:size(state01_input,2)
        fprintf(fileID_input , '%s', state01_input(i,j));
    end
    fprintf(fileID_input , '\n');
end
fclose(fileID_input);
fileID_output = fopen('outputs.txt', 'w');
for i = 1:size(results_bin,1)
    for j = 1:size(results_bin,2)
        fprintf(fileID_output , '%s', results_bin(i,j));
    end
    fprintf(fileID_output , '\n');
end
fclose(fileID_output);
```

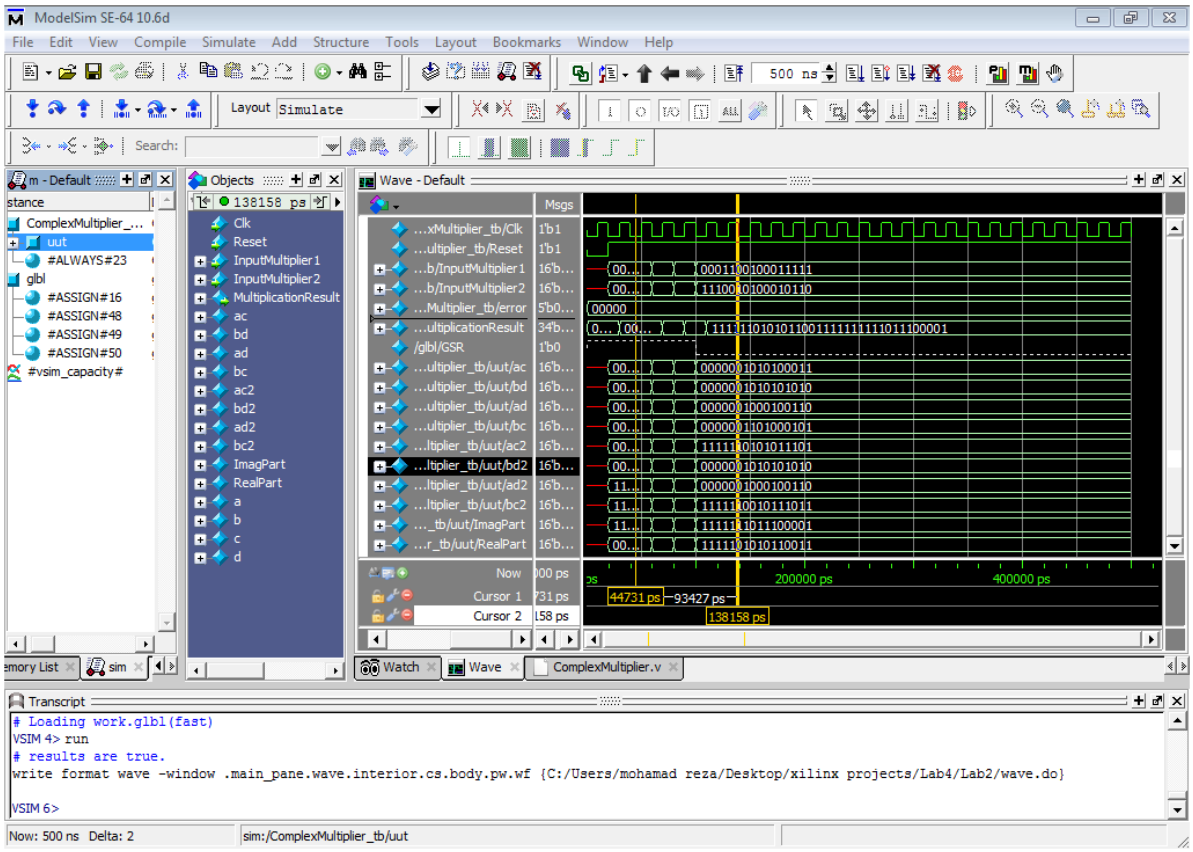
در این کد ورودی های رندوم ساخته می شوند و به صورت دسیمال داده می شوند و بعد به ورودی مورد نظر ماژول تبدیل شده و از طرفی خروجی متناظر ورودی ها نیز محاسبه می شود (حاصل ضرب آن ها) و نتیجه حاصل ضرب و ورودی ها به ترتیب در فایل های outputs و inputs ذخیره می شوند.

شبیه سازی با Isim:



همانطور که مشخص است شبیه سازی به درستی انجام شده است و تمام سیگنال های ضرب کننده به صفحه موج اضافه شدند. و تعداد ارور صفر است و در کنسول نیز عبارت results are true نمایش داده شده است و کد به درستی کار می‌کند. همچنین مارکر نیز اضافه شده است.

شبیه سازی با Modelsim:



همانطور که مشخص است تمام سیگنال های ماژول اضافه شده اند و در کنسول عبارت results are true چاپ شده است. همچنین cursor نیز اضافه شده است.

آزمایش3)

در این آزمایش مدار خواسته شده را پیاده سازی می کنیم و هر استیت را به طور جداگانه توضیح مختصر می دهیم و بعد با استفاده از تست بنچ و داده های ساخته شده با استفاده از متلب کد را صحت سنجی می کنیم.

کد ماژول:

```
module DUT(
    input Clk,
    input Reset,
    input [1:0] Control,
    input [15:0] Input1,
    input [15:0] Input2,
    output reg [37:0] Output1
    );


   wire [31:0]input1_square;
   wire [31:0]input2_square;
   wire [32:0]x_in;
   wire [16:0] out_00;
   wire [15:0]input1;
   wire [15:0]input2;

   assign input1 = Input1[15]?(~Input1 + 1):Input1;
   assign input2 = Input2[15]?(~Input2 + 1):Input2;;
   assign input1_square = input1 * input1;
   assign input2_square = input2 * input2;
   assign x_in = input1_square + input2_square;

   CORDIC suare_root (
  .x_in(x_in), // input [32 : 0] x_in
  .x_out(out_00), // output [16 : 0] x_out
  .clk(Clk) // input clk
   );

   wire [33:0]MultiplicationResult;
   ComplexMultiplier complex_multiplier (
    .Clk(Clk),
    .Reset(Reset),
    .InputMultiplier1(Input1),
    .InputMultiplier2(Input2),
    .MultiplicationResult(MultiplicationResult)
    );


   reg [15:0] a1;
   reg [15:0] b1;
   reg [15:0] a2;
   reg [15:0] b2;
   reg [15:0] a3;
   reg [15:0] b3;
   wire [32:0]p1;
   wire [33:0]p2;
   wire [34:0]p3;
   wire [47:0]pcout1;
   wire [47:0]pcout2;
   reg [7:0] counter;
   reg [1:0]state10;
   MultAdd1 MultAdd01 (
  .clk(Clk), // input clk
  .ce(1), // input ce
  .sclr(Reset), // input sclr
  .a(a1), // input [15 : 0] a
  .b(b1), // input [15 : 0] b
  .c(0), // input [1 : 0] c
  .subtract(0), // input subtract
  .p(p1), // output [32 : 0] p
  .pcout(pcout1) // output [47 : 0] pcout
   );

   MultAdd2 MultAdd02 (
  .clk(Clk), // input clk
  .ce(1), // input ce
  .sclr(Reset), // input sclr
  .a(a2), // input [15 : 0] a
  .b(b2), // input [15 : 0] b
  .c(p1), // input [47 : 0] c
  .pcin(pcout1), // input [47 : 0] pcin
  .subtract(0), // input subtract
  .p(p2), // output [33 : 0] p
  .pcout(pcout2) // output [47 : 0] pcout
   );

   MultAdd3 MultAdd03 (
  .clk(Clk), // input clk
  .ce(1), // input ce
  .sclr(Reset), // input sclr
  .a(a3), // input [15 : 0] a
  .b(b3), // input [15 : 0] b
  .c(p2), // input [47 : 0] c
  .pcin(pcout2), // input [47 : 0] pcin
  .subtract(0), // input subtract
  .p(p3), // output [34 : 0] p
  .pcout() // output [47 : 0] pcout
```

```
//state 11
reg [15:0]Mem1[0:8];
reg [15:0]Mem2[0:8];
reg [4:0]state11;
reg [15:0]Input1Mult1;
reg [15:0]Input2Mult1;
reg [15:0]Input1Mult2;
reg [15:0]Input2Mult2;
reg [15:0]Input1Mult3;
reg [15:0]Input2Mult3;
wire [33:0] MultiplicationResult1;
wire [33:0] MultiplicationResult2;
wire [33:0] MultiplicationResult3;
ComplexMultiplier1 CM1 (
 .Clk(Clk),
 .Reset(Reset),
 .InputMultiplier1(Input1Mult1),
 .InputMultiplier2(Input2Mult1),
 .MultiplicationResult(MultiplicationResult1)
 );

 ComplexMultiplier2 CM2 (
 .Clk(Clk),
 .Reset(Reset),
 .InputMultiplier1(Input1Mult2),
 .InputMultiplier2(Input2Mult2),
 .MultiplicationResult(MultiplicationResult2)
 );

 ComplexMultiplier3 CM3 (
 .Clk(Clk),
 .Reset(Reset),
 .InputMultiplier1(Input1Mult3),
 .InputMultiplier2(Input2Mult3),
 .MultiplicationResult(MultiplicationResult3)
 );
wire [18:0]out_state11Imag;
wire [18:0]out_state11Real;
assign out_state11Imag = {MultiplicationResult1[16],MultiplicationResult1[16],MultiplicationResult1[16:0]} + {MultiplicationResult2[16],MultiplicationResult2[16],
assign out_state11Real = {MultiplicationResult1[33],MultiplicationResult1[33],MultiplicationResult1[33:17]} + {MultiplicationResult2[33],MultiplicationResult2[33]
//
reg control_enable;
reg [1:0]old_control;
always @(posedge Clk)
```

```
135   begin
136      if(Reset)
137      begin
138         state10 <= 2'b00;
139         counter <= 8'b0;
140         state11 <= 5'b0;
141         control_enable <= 1;
142      end
143      else
144      begin
145         if(control_enable)
146            begin
147               old_control <= Control;
148               control_enable <= 0;
149            end
150         case(old_control)
151            2'b00:
152                begin
153                    Output1 <= {21'b0,out_00};
154                    control_enable <= 1;
155                end
156                2'b01:
157                begin
158                    Output1 <= {4'b0,MultiplicationResult};
159                    control_enable <= 1;
160                end
161                2'b10:
162                  begin
163                    case(state10)
164                  2'b00:
165                  begin
166                     a1 <= Input1;
167                     b1 <= Input2;
168                     state10 <= 2'b01;
169                  end
170                  2'b01:
171                  begin
172                     a2 <= Input1;
173                     b2 <= Input2;
174                     state10 <= 2'b10;
175                  end
176                  2'b10:
177                  begin
178                     a3 <= Input1;
```

```
179                     b3 <= Input2;
180                     state10 <= 2'b11;
181                  end
182                  2'b11:
183                  begin
184                     counter <= counter + 1;
185                     if(counter < 6)
186                        state10 <= 2'b11;
187                     else
188                     begin
189                        state10 <= 2'b00;
190                        counter <= 8'b0;
191                        Output1 <= {3'b0,p3};
192                        control_enable <= 1;
193                     end
194                  end
195               endcase
196            end
197            2'b11:
198            begin
199               case(state11)
200                  5'b00000:
201                  begin
202                     Mem1[0] <= Input1;
203                     Mem2[0] <= Input2;
204                     state11 <= 5'b00001;
205                  end
206                  5'b00001:
207                  begin
208                     Mem1[1] <= Input1;
209                     Mem2[1] <= Input2;
210                     state11 <= 5'b00010;
211                  end
212                  5'b00010:
213                  begin
214                     Mem1[2] <= Input1;
215                     Mem2[2] <= Input2;
216                     state11 <= 5'b00011;
217                  end
218                  5'b00011:
219                  begin
220                     Mem1[3] <= Input1;
221                     Mem2[3] <= Input2;
222                     state11 <= 5'b00100;
```

```
223                  end
224                  5'b00100:
225                  begin
226
227                     Mem1[4] <= Input1;
228                     Mem2[4] <= Input2;
229                     state11 <= 5'b00101;
230                  end
231                  5'b00101:
232                  begin
233                     Mem1[5] <= Input1;
234                     Mem2[5] <= Input2;
235                     state11 <= 5'b00110;
236                  end
237                  5'b00110:
238                  begin
239                     Mem1[6] <= Input1;
240                     Mem2[6] <= Input2;
241                     state11 <= 5'b00111;
242                  end
243                  5'b00111:
244                  begin
245                     Mem1[7] <= Input1;
246                     Mem2[7] <= Input2;
247                     state11 <= 5'b01000;
248                     Input1Mult1 <= Mem1[0];
249                     Input2Mult1 <= Mem2[0];//1
250                     Input1Mult2 <= Mem1[1];
251                     Input2Mult2 <= Mem2[3];
252                     Input1Mult3 <= Mem1[2];
253                     Input2Mult3 <= Mem2[6];
254                  end
255                  5'b01000://1,1
256                  begin
257                     Mem1[8] <= Input1;
258                     Mem2[8] <= Input2;
259                     state11 <= 5'b01001;
260                     Input1Mult1 <= Mem1[0];
261                     Input2Mult1 <= Mem2[1];
262                     Input1Mult2 <= Mem1[1];
263                     Input2Mult2 <= Mem2[4];
264                     Input1Mult3 <= Mem1[2];//2
265                     Input2Mult3 <= Mem2[7];
266
```

```
267                  end
268                  5'b01001://1,2
269                  begin
270                     Output1 <= {out_state11Real,out_state11Imag};//1
271                     Input1Mult1 <= Mem1[0];
272                     Input2Mult1 <= Mem2[2];
273                     Input1Mult2 <= Mem1[1];
274                     Input2Mult2 <= Mem2[5];//3
275                     Input1Mult3 <= Mem1[2];
276                     Input2Mult3 <= Mem2[8];
277                     state11 <= 5'b01010;
278                  end
279                  5'b01010://1,3
280                  begin
281                     Output1 <= {out_state11Real,out_state11Imag};//2
282                     Input1Mult1 <= Mem1[3];
283                     Input2Mult1 <= Mem2[0];
284                     Input1Mult2 <= Mem1[4];
285                     Input2Mult2 <= Mem2[3];
286                     Input1Mult3 <= Mem1[5];//4
287                     Input2Mult3 <= Mem2[6];
288                     state11 <= 5'b01011;
289                  end
290                  5'b01011://2,1
291                  begin
292                     Output1 <= {out_state11Real,out_state11Imag};//3
293                     Input1Mult1 <= Mem1[3];
294                     Input2Mult1 <= Mem2[1];
295                     Input1Mult2 <= Mem1[4];
296                     Input2Mult2 <= Mem2[4];
297                     Input1Mult3 <= Mem1[5];
298                     Input2Mult3 <= Mem2[7];//5
299                     state11 <= 5'b01100;
300                  end
301                  5'b01100://2,2
302                  begin
303                     Output1 <= {out_state11Real,out_state11Imag};//4
304                     Input1Mult1 <= Mem1[3];
305                     Input2Mult1 <= Mem2[2];
306                     Input1Mult2 <= Mem1[4];
307                     Input2Mult2 <= Mem2[5];
308                     Input1Mult3 <= Mem1[5];//6
309                     Input2Mult3 <= Mem2[8];
310                     state11 <= 5'b01101;
```

```
311                 end
312                 5'b01101://2,3
313                 begin
314                    Output1 <= {out_state11Real,out_state11Imag};//5
315                    Input1Mult1 <= Mem1[6];
316                    Input2Mult1 <= Mem2[0];
317                    Input1Mult2 <= Mem1[7];
318                    Input2Mult2 <= Mem2[3];//7
319                    Input1Mult3 <= Mem1[8];
320                    Input2Mult3 <= Mem2[6];
321                    state11 <= 5'b01110;
322                 end
323                 5'b01110://3,1
324                 begin
325                    Output1 <= {out_state11Real,out_state11Imag};//6
326                    Input1Mult1 <= Mem1[6];
327                    Input2Mult1 <= Mem2[1];
328                    Input1Mult2 <= Mem1[7];//8
329                    Input2Mult2 <= Mem2[4];
330                    Input1Mult3 <= Mem1[8];
331                    Input2Mult3 <= Mem2[7];
332                    state11 <= 5'b01111;
333                 end
334                 5'b01111://3,2
335                 begin
336                    Output1 <= {out_state11Real,out_state11Imag};//7
337                    Input1Mult1 <= Mem1[6];
338                    Input2Mult1 <= Mem2[2];
339                    Input1Mult2 <= Mem1[7];//9
340                    Input2Mult2 <= Mem2[5];
341                    Input1Mult3 <= Mem1[8];
342                    Input2Mult3 <= Mem2[8];
343                    state11 <= 5'b10000;
344                 end
345                 5'b10000://3,3
346                 begin
347                    Output1 <= {out_state11Real,out_state11Imag};//8
348                    state11 <= 5'b10001;
349                 end
350                 5'b10001:
351                 begin
352                    state11 <= 5'b00000;
353                    Output1 <= {out_state11Real,out_state11Imag};//9
354                   control_enable <= 1;
```

```
353                    Output1 <= {out_state11Real,out_state11Imag};//9
354                   control_enable <= 1;
355                end
356             endcase
357          end
358          default: Output1 <= 38'bx;
359       endcase
360    end
361 end
362
363 endmodule
364
```

در این کد یک ورودی control وجود دارد که مشخص می کند می خواهیم از کدام یک از استیت های مدار استفاده کنیم. از آنجایی که هر کدام از استیت ها برای خروجی دادن به چند کلاک احتیاج دارند اگر ورودی control در بین خروجی دادن ماژول تغییر کند نباید اتفاقی بیافتد و مدار باید ابتدا خروجی های استیت قبلی را به صورت کامل خروجی دهد. برای این منظور از 2 رجیستر control_enable 1بیتی و old_conrol استفاده می کنیم که تعداد بیت دومی برابر تعداد بیت ورودی control است. در ابتدا control_enable 1 است و اجازه می دهد که ورودی control داخل old_control ریخته شود که این رجیستر حالت های استیت ماشین اصلی را کنترل می کند. سپس با ورودی به استیت ماشین(case) control_enable صفر می شود و اجازه نمی دهد old_conrol تغییر کند تا زمانی که کار استیت قبلی به پایان برسد و همه ی خروجی ها داده شود سپس conrol_enabel 1 می شود و اجازه تغییر old_control داده می شود.

استیت 00:

در این استیت طبق دستور کار ip core مناسب برای رادیکال گرفتن را می سازیم و به ماژول اضاف می کنیم. این ای پی کور به گونه ای کار می کند که خروجی آن روند می شود و یک عدد صحیح است. همچنین اگر در این قسمت ورودی‌ها اعداد منفی باشند در فرمت 2's complement ابتدا آن ها را مثبت می کنیم و سپس به ای پی کور می دهیم. در نهایت خروجی ماژول انتگرال گیر را در استیت ماشین و در استیت 00 داخل خروجی می ریزیم که کل عملیات 1 کلاک طول می کشد.همچنین با استفاده از کد متلب زیر داده های مناسب برای تست بنچ را تولید می کنیم و همچنین حاصل عملیات را نیز محاسبه می کنیم. سپس در دو فایل state00_outputs,state00_inputs آن ها را می نویسیم.

```
4 -   random_numbers = randi([-16000, 16000], 1, 10);
5 -   state00_input = dec2bin(random_numbers,16);
6 -   squares = random_numbers.^2;
7 -   output = zeros([1,5]);
8 -   output_bin = [];
9 -   for j = 1:2:10
10 -      output((j+1)/2) = round(sqrt(squares(j) + squares(j + 1)));
11 -      output_bin = [output_bin ; dec2bin(output((j+1)/2),17)];
12 -  end
13 -  fileID_input = fopen('state00_inputs.txt', 'w');
14 -  for i = 1:size(state00_input,1)
15 -      for j = 1:size(state00_input,2)
16 -          fprintf(fileID_input , '%s', state00_input(i,j));
17 -      end
18 -      fprintf(fileID_input , '\n');
19 -  end
20 -  fclose(fileID_input);
21
22 -  fileID_output = fopen('state00_outputs.txt', 'w');
23 -  for i = 1:size(output_bin,1)
24 -      for j = 1:size(output_bin,2)
25 -          fprintf(fileID_output , '%s', output_bin(i,j));
26 -      end
27 -      fprintf(fileID_output , '\n');
28 -  end
29 -  fclose(fileID_output);
```

استیت 01:

در این استیت که دقیقا ماژول استفاده شده در آزمایش 2 است را در کد این ماژول وارد می کنیم و در استیت مورد نظر ورودی ها را به آن می دهیم و در استیت ماشین خروجی این ماژول در خروجی ماژول اصلی می ریزیم. کد متلب آن نیز دقیقا مانند آزمایش 2 می باشد و داده ها را در دو فایل state01_inputs , state_01_outputs ذخیره می کند.

استیت 10:

ابتدا طبق دستور کار ip core 3 مورد نظر را می سازیم و داخل ماژول میاوریم. سپس در استیت مورد نظر یک استیت ماشین دیگر می سازیم و ورودی ها را به ترتیب در 3 کلاک داخل ورودی های این 3 ای پی کور می‌ریزیم. سپس از یک counter استفاده می کنیم و بعد از 6 کلاک خروجی می دهیم. که کل عملیات این استیت 10 کلاک طول می کشد.
در ادامه کد متلب برای تولید تست این استیت اورده شده است که ورودی ها و خروجی های را در 2 فایل state10_outputs,state10_inputs ذخیره می کنیم.

```
clc
random_numbers = randi([-16000, 16000], 1, 18);
state10_input = dec2bin(random_numbers,16);
output = zeros([1,3]);
output_bin = [];
for j = 1:6:18
    a1 = [random_numbers(j),random_numbers(j + 1),random_numbers(j + 2)];
    a2 = [random_numbers(j + 3);random_numbers(j + 4);random_numbers(j + 5)];
    output((j+5)/6) = a1 * a2;
    output_bin = [output_bin ; dec2bin(output((j+5)/6),35)];
end
fileID_input = fopen('state10_inputs.txt', 'w');
for i = 1:size(state10_input,1)
    for j = 1:size(state10_input,2)
        fprintf(fileID_input , '%s', state10_input(i,j));
    end
    fprintf(fileID_input , '\n');
end
fclose(fileID_input);

fileID_output = fopen('state10_outputs.txt', 'w');
for i = 1:size(output_bin,1)
    for j = 1:size(output_bin,2)
        fprintf(fileID_output , '%s', output_bin(i,j));
    end
    fprintf(fileID_output , '\n');
end
fclose(fileID_output);
```

استیت 11:

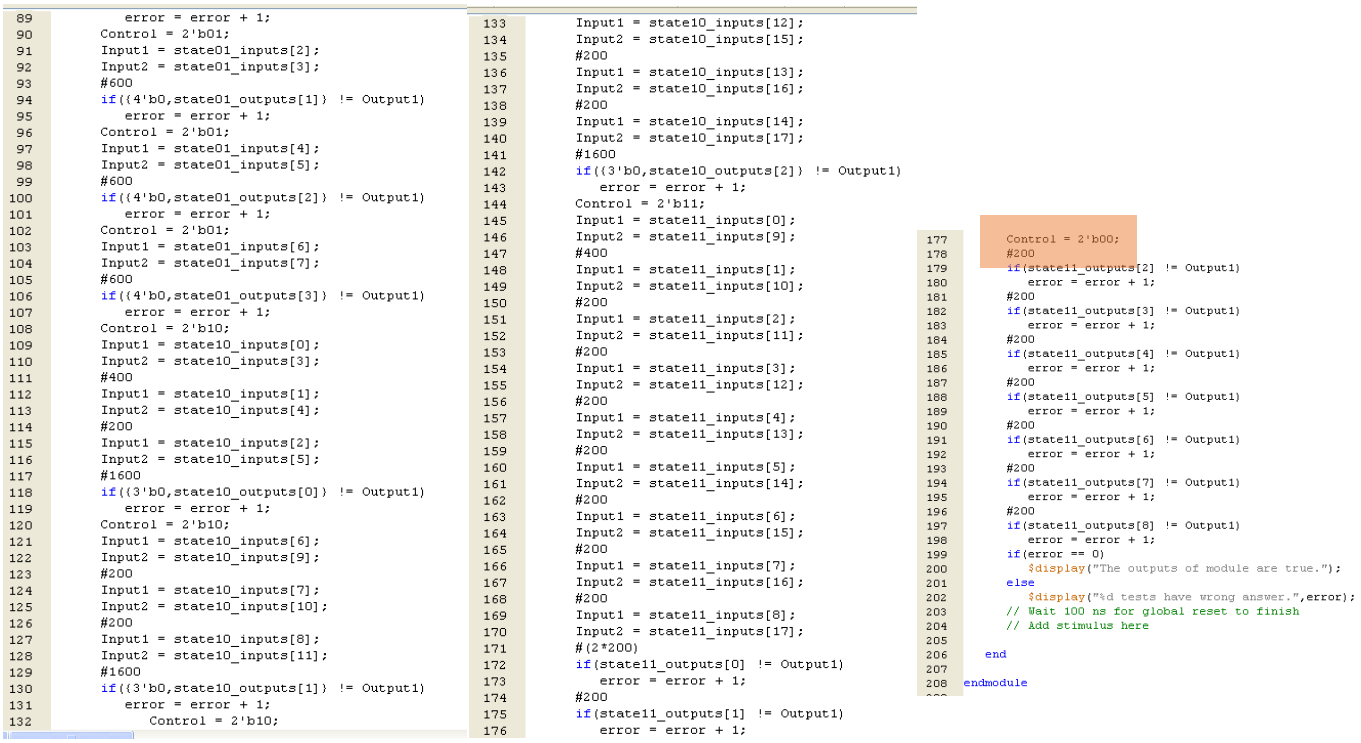
در این قسمت نیز از ماژول استفاده شده در آزمایش 2 استفاده می کنیم و ورودی ها را در 9 کلاک به ترتیب در دو رم می ریزیم که ترتیب ورودی سطر به سطر ماتریس است. بعد از 7 کلاک که ستون اول ماتریس دوم کامل شد ورودی‌های 3 ضرب کننده را تعیین می کنیم و جمع خروجی این 3 ضرب کننده که 1 درایه از ماتریس خروجی را مشخص می کند در دو کلاک بعدی در خروجی قرار می دهیم دلیل آن این است که ضرب کننده 1 کلاک برای ضرب کردن احتیاج دارد و در کل 2 کلاک برای خروجی دادن نیاز دارد. بنابراین بعد از 9 کلاک خروجی دادن شروع می شود و کل عملیات ورودی گرفتن و خروجی دادن در 20 کلاک انجام می شود. کد متلب زیر یک مثال برای این استیت می سازد و به همراه خروجی در دو فایل state11_outputs,state11_inputs ذخیره می کند.

```
random_numbers = randi([-50, 50], 1, 36);
state11_input = [];
for j = 1:2:36
    state11_input = [state11_input;[dec2bin(random_numbers(j),8),dec2bin(random_numbers(j + 1),8)]];
end
A = zeros(3);
B = zeros(3);
j = 1;
for i = 1:2:size(random_numbers,2)
    if j < 10
        A(j) = random_numbers(i) + random_numbers(i + 1) * 1i;
    else
        B(j-9) = random_numbers(i) + random_numbers(i + 1) * 1i;
    end
    j = j + 1;
end
A = A.';
B = B.';
C = A * B;
C = C.';
results_bin = [];
for i = 1:9
    results_bin = [results_bin;dec2bin(real(C(i)),19),dec2bin(imag(C(i)),19)];
end
fileID_input = fopen('state11_inputs.txt', 'w');
for i = 1:size(state11_input,1)
    for j = 1:size(state11_input,2)
        fprintf(fileID_input , '%s', state11_input(i,j));
    end
    fprintf(fileID_input , '\n');
end
fclose(fileID_input);

fileID_output = fopen('state11_outputs.txt', 'w');
for i = 1:size(results_bin,1)
    for j = 1:size(results_bin,2)
        fprintf(fileID_output , '%s', results_bin(i,j));
    end
    fprintf(fileID_output , '\n');
end
fclose(fileID_output);
```

کد تست بنچ:

```
`timescale 1ns / 1ps

module DUT_tb;

   // Inputs
   reg Clk;
   reg Reset;
   reg [1:0] Control;
   reg [15:0] Input1;
   reg [15:0] Input2;
   reg [15:0] state00_inputs[0:9];
   reg [15:0] state01_inputs[0:7];
   reg [15:0] state10_inputs[0:17];
   reg [15:0] state11_inputs[0:17];

   reg [16:0] state00_outputs[0:4];
   reg [33:0] state01_outputs[0:3];
   reg [34:0] state10_outputs[0:2];
   reg [37:0] state11_outputs[0:8];
   // Outputs
   wire [37:0] Output1;
   wire [34:0]out1;
   reg [5:0]error;
   // Instantiate the Unit Under Test (UUT)
   DUT uut (
      .Clk(Clk),
      .Reset(Reset),
      .Control(Control),
      .Input1(Input1),
      .Input2(Input2),
      .Output1(Output1)
   );
   assign out1 = Output1[34:0];
   always #100 Clk = ~Clk;
   initial begin
      // Initialize Inputs
      Clk = 0;
      Reset = 1;
      error = 0;
      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab3/data/state00_inputs.txt",state00_inputs);
      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab3/data/state00_outputs.txt",state00_outputs);

      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab3/data/state01_inputs.txt",state01_inputs);
      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab3/data/state01_outputs.txt",state01_outputs);
```

```

      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab3/data/state10_inputs.txt",state10_inputs);
      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab3/data/state10_outputs.txt",state10_outputs);

      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab3/data/state11_inputs.txt",state11_inputs);
      $readmemb("C:/Documents and Settings/Administrator/Desktop/Xillinx projects/Lab4/Lab3/data/state11_outputs.txt",state11_outputs);

      #200
      Reset = 0;
      Control = 2'b00;
      Input1 = state00_inputs[0];
      Input2 = state00_inputs[1];
      #600
      if((21'b0,state00_outputs[0]) != Output1)
         error = error + 1;
      Control = 2'b00;
      Input1 = state00_inputs[2];
      Input2 = state00_inputs[3];
      #600
      if((21'b0,state00_outputs[1]) != Output1)
         error = error + 1;
      Control = 2'b00;
      Input1 = state00_inputs[4];
      Input2 = state00_inputs[5];
      #600
      if((21'b0,state00_outputs[2]) != Output1)
         error = error + 1;
      Control = 2'b00;
      Input1 = state00_inputs[6];
      Input2 = state00_inputs[7];
      #600
      if((21'b0,state00_outputs[3]) != Output1)
         error = error + 1;
      Control = 2'b00;
      Input1 = state00_inputs[8];
      Input2 = state00_inputs[9];
      #600
      if((21'b0,state00_outputs[4]) != Output1)
         error = error + 1;
      Control = 2'b01;
      Input1 = state01_inputs[0];
      Input2 = state01_inputs[1];
      #600
      if((4'b0,state01_outputs[0]) != Output1)
```



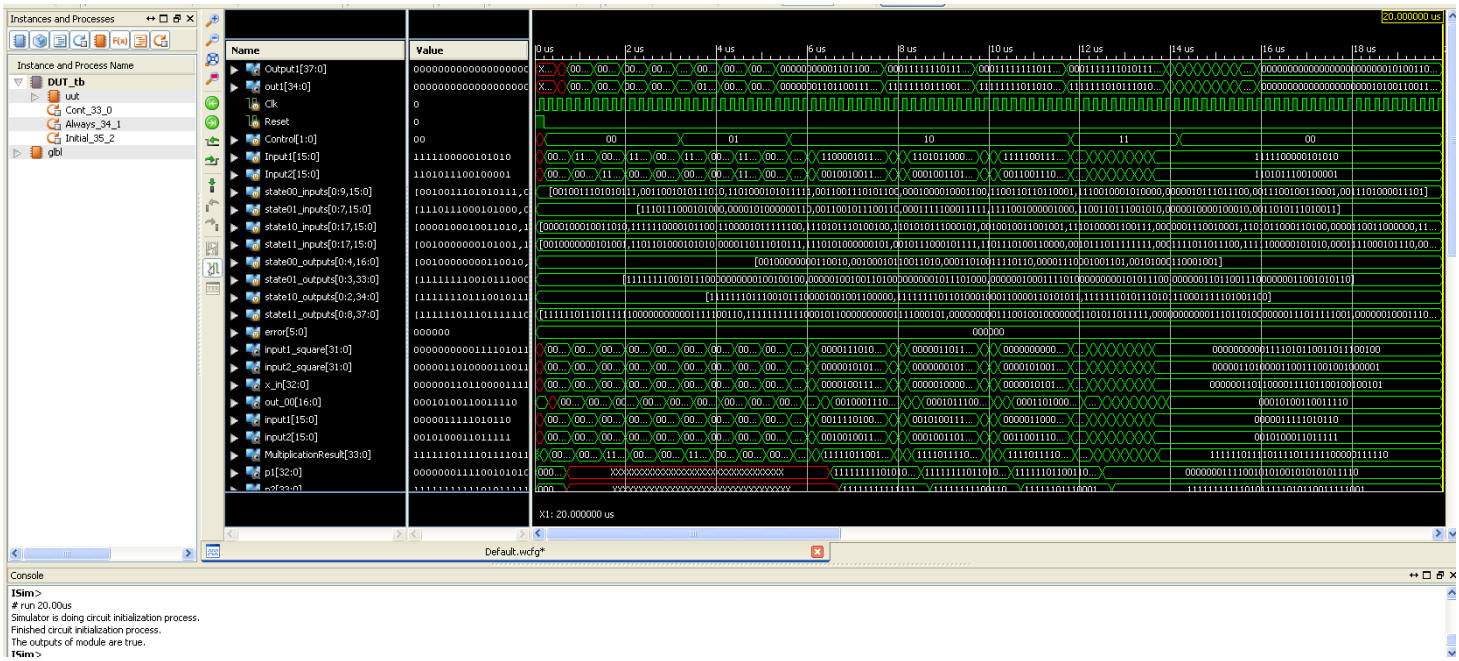
```
89          error = error + 1;
90      Control = 2'b01;
91      Input1 = state01_inputs[2];
92      Input2 = state01_inputs[3];
93      #600
94      if({4'b0,state01_outputs[1]} != Output1)
95         error = error + 1;
96      Control = 2'b01;
97      Input1 = state01_inputs[4];
98      Input2 = state01_inputs[5];
99      #600
100     if({4'b0,state01_outputs[2]} != Output1)
101        error = error + 1;
102     Control = 2'b01;
103     Input1 = state01_inputs[6];
104     Input2 = state01_inputs[7];
105     #600
106     if({4'b0,state01_outputs[3]} != Output1)
107        error = error + 1;
108     Control = 2'b10;
109     Input1 = state10_inputs[0];
110     Input2 = state10_inputs[3];
111     #400
112     Input1 = state10_inputs[1];
113     Input2 = state10_inputs[4];
114     #200
115     Input1 = state10_inputs[2];
116     Input2 = state10_inputs[5];
117     #1600
118     if({3'b0,state10_outputs[0]} != Output1)
119        error = error + 1;
120     Control = 2'b10;
121     Input1 = state10_inputs[6];
122     Input2 = state10_inputs[9];
123     #200
124     Input1 = state10_inputs[7];
125     Input2 = state10_inputs[10];
126     #200
127     Input1 = state10_inputs[8];
128     Input2 = state10_inputs[11];
129     #1600
130     if({3'b0,state10_outputs[1]} != Output1)
131        error = error + 1;
132           Control = 2'b10;
```

```
133     Input1 = state10_inputs[12];
134     Input2 = state10_inputs[15];
135     #200
136     Input1 = state10_inputs[13];
137     Input2 = state10_inputs[16];
138     #200
139     Input1 = state10_inputs[14];
140     Input2 = state10_inputs[17];
141     #1600
142     if({3'b0,state10_outputs[2]} != Output1)
143        error = error + 1;
144     Control = 2'b11;
145     Input1 = state11_inputs[0];
146     Input2 = state11_inputs[9];
147     #400
148     Input1 = state11_inputs[1];
149     Input2 = state11_inputs[10];
150     #200
151     Input1 = state11_inputs[2];
152     Input2 = state11_inputs[11];
153     #200
154     Input1 = state11_inputs[3];
155     Input2 = state11_inputs[12];
156     #200
157     Input1 = state11_inputs[4];
158     Input2 = state11_inputs[13];
159     #200
160     Input1 = state11_inputs[5];
161     Input2 = state11_inputs[14];
162     #200
163     Input1 = state11_inputs[6];
164     Input2 = state11_inputs[15];
165     #200
166     Input1 = state11_inputs[7];
167     Input2 = state11_inputs[16];
168     #200
169     Input1 = state11_inputs[8];
170     Input2 = state11_inputs[17];
171     #(2*200)
172     if(state11_outputs[0] != Output1)
173        error = error + 1;
174     #200
175     if(state11_outputs[1] != Output1)
176        error = error + 1;
```

```
177     Control = 2'b00;
178     #200
179     if(state11_outputs[2] != Output1)
180        error = error + 1;
181     #200
182     if(state11_outputs[3] != Output1)
183        error = error + 1;
184     #200
185     if(state11_outputs[4] != Output1)
186        error = error + 1;
187     #200
188     if(state11_outputs[5] != Output1)
189        error = error + 1;
190     #200
191     if(state11_outputs[6] != Output1)
192        error = error + 1;
193     #200
194     if(state11_outputs[7] != Output1)
195        error = error + 1;
196     #200
197     if(state11_outputs[8] != Output1)
198        error = error + 1;
199     if(error == 0)
200        $display("The outputs of module are true.");
201     else
202        $display("%d tests have wrong answer.",error);
203     // Wait 100 ns for global reset to finish
204     // Add stimulus here
205
206  end
207
208 endmodule
```

در این تست بنچ ابتدا تمام فایل های ساخته شده در قسمت قبل را می خوانیم و داخل 8 رم می ریزیم که 4 تا برای ورودی ها و 4 تا برای خروجی های متناظر هستند. سپس با رعایت زمانبندی گفته شده در قسمت قبل ورودی ها را به ترتیب و در استیت های 00و01و10و11 وارد می کنیم و خروجی های هر تست را با پاسخ درست مقایسه می کنیم و اگر در جایی خروجی ماژول استباه بود به متغیر error یکی اضافه می کنیم. در نهایت اگر error صفر باشد عبارت درست بودن خروجی ها نوشته می شود در غیر این صورت تعداد ارور ها نمایش داده می شود. همچنین در استیت 11 که مثال آخر است وقتی هنوز خروجی ها به طور کامل داده نشده اند ورودی control را تغییر می دهیم (که با مستطیل قرمز نمایش داده شده است) تا مطمئن شویم کد به درستی کار می کند.

نتیجه شبیه سازی :



همانطور که مشخص است error در انتها دارای مقدار صفر است و همچنین در کنسول عبارت درست بودن خروجی ها نمایش داده شده است. همچنین وقتی در استیت 11 هستیم و در حال خروجی گرفتن هستیم control را 00 می کنیم اما خروجی ها تغییر نمی کنند و استیت 11 به درستی و تا انتها خروجی می دهد و error تا انتها صفر می ماند.

آزمایش 4)

طبق دستور کار تنظیمات خواسته شده را پیاده سازی می کنیم و فایل های constrain را می سازیم:

فایل ucf:

```
INST "Output1<8>" TNM = Output1_Group;
INST "Output1<9>" TNM = Output1_Group;
INST "Output1<10>" TNM = Output1_Group;
INST "Output1<11>" TNM = Output1_Group;
INST "Output1<12>" TNM = Output1_Group;
INST "Output1<13>" TNM = Output1_Group;
INST "Output1<14>" TNM = Output1_Group;
INST "Output1<15>" TNM = Output1_Group;
INST "Output1<16>" TNM = Output1_Group;
INST "Output1<17>" TNM = Output1_Group;
INST "Output1<18>" TNM = Output1_Group;
INST "Output1<19>" TNM = Output1_Group;
INST "Output1<20>" TNM = Output1_Group;
INST "Output1<21>" TNM = Output1_Group;
INST "Output1<22>" TNM = Output1_Group;
INST "Output1<23>" TNM = Output1_Group;
INST "Output1<24>" TNM = Output1_Group;
INST "Output1<25>" TNM = Output1_Group;
INST "Output1<26>" TNM = Output1_Group;
INST "Output1<27>" TNM = Output1_Group;
INST "Output1<28>" TNM = Output1_Group;
INST "Output1<29>" TNM = Output1_Group;
INST "Output1<30>" TNM = Output1_Group;
INST "Output1<31>" TNM = Output1_Group;
INST "Output1<32>" TNM = Output1_Group;
INST "Output1<33>" TNM = Output1_Group;
INST "Output1<34>" TNM = Output1_Group;
INST "Output1<35>" TNM = Output1_Group;
INST "Output1<36>" TNM = Output1_Group;
INST "Output1<37>" TNM = Output1_Group;
TIMEGRP "Output1_Group" OFFSET = OUT 20 ns AFTER "Clk";

NET "Clk" LOC = L15;
NET "Reset" LOC = A10;
```

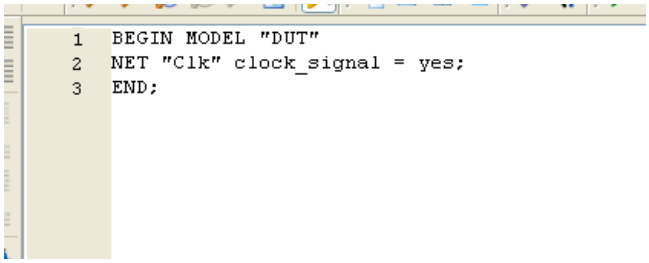
Design Summary (Synthesized) | DUT.ucf

همچنین constrain هایی که می توانیم در کد ماژول قرار دهیم را نیز قرار می دهیم:

```
    .InputMultiplier2(Input2),
    .MultiplicationResult(MultiplicationResult)
    );

    (* KEEP = "{TRUE}" *)
    reg [15:0] a1;
    reg [15:0] b1;
    reg [15:0] a2;
    reg [15:0] b2;
    reg [15:0] a3;
    reg [15:0] b3;
    wire [32:0]p1;
    wire [33:0]p2;
    wire [34:0]p3;
    wire [47:0]pcout1;
    wire [47:0]pcout2;
    reg [7:0] counter;
    reg [1:0]state10;
    MultAdd1 MultAdd01 (
.clk(Clk), // input clk
```

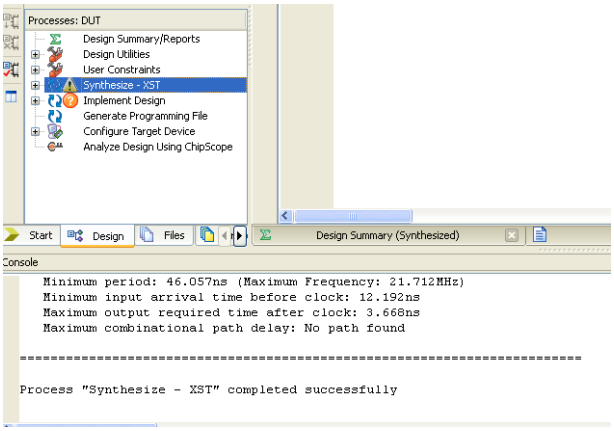
```
    reg [15:0]Input2Mult3;
    wire [33:0] MultiplicationResult1;
    wire [33:0] MultiplicationResult2;
    wire [33:0] MultiplicationResult3;
    (* KEEP_HIERARCHY = "{TRUE}" *)
    ComplexMultiplier1 CM1 (
      .Clk(Clk),
      .Reset(Reset),
      .InputMultiplier1(Input1Mult1),
      .InputMultiplier2(Input2Mult1),
      .MultiplicationResult(MultiplicationResult1)
      );

      ComplexMultiplier2 CM2 (
      .Clk(Clk),
      .Reset(Reset),
      .InputMultiplier1(Input1Mult2),
      .InputMultiplier2(Input2Mult2),
      .MultiplicationResult(MultiplicationResult2)
      );
```

همچنین فایل xcf را نیز طبق مراحل می سازیم و ذخیره می کنیم:

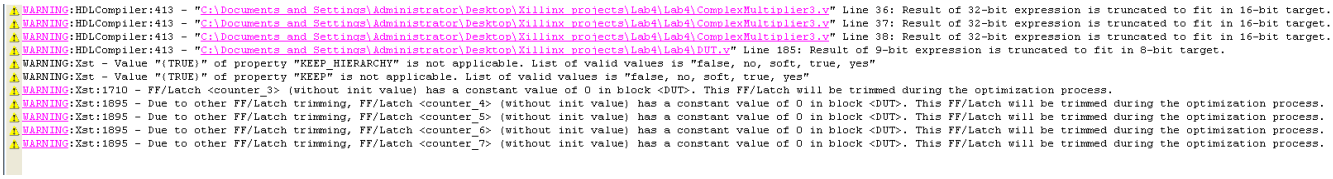


```
1  BEGIN MODEL "DUT"
2  NET "Clk" clock_signal = yes;
3  END;
```

بعد از نوشتن تمام constrain ها مدار را سنتز می کنیم:



که سنتز با موفقیت انجام شده است. اما چند وارنینگ وجود دارد:



که عموما مورد خاصی نیستند.