DEPARTMENT

OF

ELECTRONICS & COMMUNICATION ENGINEERING

HDL LABORATORY MANUAL

V Semester (19EC5DLHDL)

Autonomous Course 2021-22

HOD	**:**	**Dr. Manjunath T C**
Lab In –Charges	**:**	**Dr. Rajagopal A.**
		Dr. Sapna P J
		Dr. Shashi Raj K
		Prof. Madhura R
		Prof. Manasa R
Lab Instructors	:	**Mr. Nuthesh Kumar**
		Mrs. Divya

Name of the Student	**:**	
Semester /Section	**:**	
USN	**:**	
Batch	**:**	

Dayananda Sagar College of Engineering Shavige Malleshwara Hills, Kumaraswamy Layout, Banashankari, Bangalore-560078, Karnataka

Tel : +91 80 26662226 26661104 Extn : 2731 Fax : +90 80 2666 0789 Web - http://www.dayanandasagar.edu Email : hod-ece@dayanandasagar.edu (An Autonomous Institute Affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified) (Accredited by NBA, National Assessment & Accreditation Council (NAAC) with 'A' grade)

About the College & the Department

The Dayananda Sagar College of Engineering was established in 1979, was founded by Sri R. Dayananda Sagar and is run by the Mahatma Gandhi Vidya Peetha Trust (MGVP). The college offers undergraduate, post-graduates and doctoral programmes under Visvesvaraya Technological University & is currently autonomous institution. MGVP Trust is an educational trust and was promoted by Late. Shri. R. Dayananda Sagar in 1960. The Trust manages 28 educational institutions in the name of "Dayananda Sagar Institutions" (DSI) and multi – Specialty hospitals in the name of Sagar Hospitals - Bangalore, India. Dayananda Sagar College of Engineering is approved by All India Council for Technical Education (AICTE), Govt. of India and affiliated to Visvesvaraya Technological University. It has widest choice of engineering branches having 16 Under Graduate courses & 17 Post Graduate courses. In addition, it has 21 Research Centres in different branches of Engineering catering to research scholars for obtaining Ph.D under VTU. Various courses are accredited by NBA & the college has a NAAC with ISO certification. One of the vibrant & oldest dept is the ECE dept. & is the biggest in the DSI group with 70 staffs & 1200+ students with 10 Ph.D.'s & 30+ staffs pursuing their research in various universities. At present, the department runs a UG course (BE) with an intake of 240 & 2 PG courses (M.Tech.), viz., VLSI Design Embedded Systems & Digital Electronics & Communications with an intake of 18 students each. The department has got an excellent infrastructure of 10 sophisticated labs & dozen class room, R & D centre, etc

Vision of the College

To impart quality technical education with a focus on Research and Innovation emphasising on Development of Sustainable and Inclusive Technology for the benefit of society.

Mission of the College

- To provide an environment that enhances creativity and Innovation in pursuit of Excellence.
- To nurture teamwork in order to transform individuals as responsible leaders and entrepreneurs.
- To train the students to the changing technical scenario and make them to understand the importance of Sustainable and Inclusive technologies.

Vision of the Department

To achieve continuous improvement in quality technical education for global competence with focus on industry, societal needs, research and professional success.

Mission of the Department

- Offering quality education in Electronics and Communication Engineering with effective teaching learning process in multidisciplinary environment.
- Training the students to take-up projects in emerging technologies and work with team spirit.
- To imbibe professional ethics, development of skills and research culture for better placement opportunities.

Programme Educational Objectives [PEOs]

The Graduate students must be able to:

PEO-1: Ready to apply the state-of-art technology in industry and meeting the societal needs with knowledge of Electronics and Communication Engineering due to strong academic culture.

PEO-2: Competent in technical and soft skills to be employed with capability of working in multidisciplinary domains.

PEO-3: Professionals, capable of pursuing higher studies in technical, research or management programs.

Programme Specific Outcomes [PSOs]

PSO-1: Design, develop and integrate electronic circuits and systems using current practices and Standards.

PSO-2: Apply knowledge of hardware and software in designing Embedded and Communication Systems.

HDL LAB Syllabus

V SEMESTER B. E (ECE)

Course Code : 19EC5DLHDL
L : T : P : S : 0 : 1 : 2 : 0
Exam Hours : 03
Total Hours : 26

Credits : 02
CIE Marks : 50
SEE Marks : 50
CIE + SEE : 100

COURSE OBJECTIVES:

1. To describe the functionality of combinational and sequential circuits using HDL.
2. Learn how to simulate, analyze the designed program.
3. To know how to debug the code.
4. To justify the design on FPGA kit through implementation.
5. To illustrate interfacing concepts using FPGA kit.
6. To give an in-depth exposure to the various tools in Intel Quartus Prime/De-Sim/Modelsim/iverilog/Xilinx ISE.

COURSE OUTCOMES:

At the end of the course, student will be able to

CO1	Gain the knowledge of using Intel Quartus Prime/De-Sim/Modelsim/iverilog/Xilinx ISE tool.
CO2	Design and simulate the HDL code for digital circuits.
CO3	Analyze and debug HDL code for digital circuits.
CO4	Synthesize digital circuits with FPGA kit.
CO5	Design HDL program to synthesize hardware interfacing experiments.
CO6	Develop digital system for real time problems and Implement on FPGA kit.

Mapping of Course Outcomes to Program Outcomes:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	2	2	2	3	-	-	-	-	-	-	1
CO2	3	2	2	2	3	-	-	-	-	-	-	1
CO3	3	2	2	2	3	-	-	-	-	-	-	1
CO4	3	2	2	2	3	-	-	-	-	-	-	1
CO5	2	2	1	1	2	-	-	-	-	-	-	1
CO6	2	2	2	2	2	-	-	-	-	-	-	1

Expt. No.	Content of the Lab Module with Expt. Nos.	Hours	COs
NOTE: 1. Design using Verilog ♦ RTL Development 2. Verification using Verilog ♦ Test bench and Test case Development 3. Simulation and Debug Simulation Tools: Intel Quartus Prime/De-Sim/Modelsim/iverilog/Xilinx ISE tool.			
1	Write a HDL code to realize all the logic gates	03	CO 1-4
2	Write a HDL code to describe the functions of a Full Adder using three modelling styles and full adder using Half adder	03	CO 1-4
3	Write HDL code for 4-bit ripple carry adder using full adders.	03	CO 1-4
4	Write HDL code to realize Adder/Subtractor circuit.	03	CO 1-4
5	Write a HDL program for the following combinational designs a. 2 to 4 Decoder. b. 8 to 3 Encoder (without priority & with priority). c. 8 to 1 Multiplexer. d. De-multiplexer. e. 4 bit Binary to Gray converter. f. 4 bit Gray to Binary converter g. Comparator	03	CO 1-4
6	Write HDL code to design BCD Adder.	03	CO 1-4
7	Develop the HDL code for the following flip-flops, a. D Flip-Flop b. T Flip-Flop c. JK Flip-Flop	03	CO 1-4
8	Write a model for 4bit ALU. ALU should use combinational logic to calculate an output based on the 3bit op-code input. ALU should decode the 3 bit op-code according to the given in example below. Opcode (2:0) OPCODE ALU OPERATION i) A + B ii) A – B iii) A Complement iv) A * B v) A AND B	03	CO 1-4

	vi) A OR B vii) A NAND B viii) A XOR B		
9	Design 4 bit Binary and BCD counters (Synchronous reset and Asynchronous reset) and "any sequence" counters	03	CO 1-4
10	Design Mealy and Moore state machine for a given sequence.	03	CO 1-4
11	Write HDL code to display any message using Seven Segment Display.	03	CO 1-6
12	Write HDL code to display the color images using VGA cable.	03	CO 1-6

ASSESSMENT PATTERN:

CIE – Continuous Internal Evaluation Lab (50 Marks)

10 Marks for the record, 10 Marks for the conduction & 5 marks for the viva-voce

SEE – Semester End Examination Lab (50 Marks)

Bloom's Category	**Performance (Day To Day)**	**Internal Test**
Marks (Out of 50)	**25**	**25**
Remember		
Understand		
Apply	05	05
Analyze	10	10
Evaluate	05	05
Create	05	05

Bloom's Category	**Marks Theory (50)**
Remember	-
Understand	5
Apply	15
Analyze	10
Evaluate	10
Create	10

Experiment No: 1 Date: _____

LOGIC GATES

Aim: Write a HDL code to realize all the LOGIC GATES.

Verilog

```
module logicgates(a,b,y);
input a,b;
output[6:0] y;
assign y[0] = ~a;
assign y[1] = a & b;
assign y[2] = a|b;
assign y[3] = ~(a&b);
assign y[4] = ~(a|b);
assign y[5] = a^b;
assign y[6] = ~(a^b);
endmodule
```

Test Bench:

```
module logicgates_tb;
integer i;
//Initialize inputs as registers and outputs as Wire
reg a,b;
wire [6:0] y;
//Connect the testbench with the file you want to run
logicgates uut (.a(a),.b(b),.y(y));
initial begin
        #100
        // Initialize Inputs that you want to pass
        a=0;
```

```
        b=0;
        for(i=0;i<4;i=i+1)
        #10 {a,b}=i;
        end
        initial begin
//This is equivalent to print statement
        $monitor("a=%b,b=%b,y=%b\n",a,b,y);
        end
endmodule
```

Results:

Experiment No: 2 Date: _____

FULL ADDER

Aim: a) Write a HDL code to describe the functions of a Full Adder using three modelling styles.

1. DATA FLOW
2. BEHAVIORAL
3. STRUCTURAL

1. DATA FLOW

Verilog

```
module fulladder_df (a, b, cin, sum, cout);
input a, b, cin;
output sum, cout;
assign sum = a^b^cin;
assign cout = (a&b)|(b&cin)|(a&cin);
endmodule
```

2. BEHAVIORAL

Verilog

```
 module fulladder_bh( a, b, cin, sum, cout);
 input a, b, cin;
 output sum,cout;
   reg sum,cout;
  always @(a,b,cin)
  {cout,sum}=a+b+cin;
endmodule
```

3. STRUCTURAL

Verilog

```
module fulladder_str (a, b, cin, sum, cout); input a, b, cin;
output sum, cout; wire s1,s2,s3;
xor x1(sum,a,b,cin);
and a1(s1,a,b);
and a2(s2,a,cin);
and a3(s3,b,cin);
 or o1(cout,s1,s2,s3);
endmodule
```

b) Full adder using Half adders

Aim: b) Write a HDL code to realize full adder using Half adders

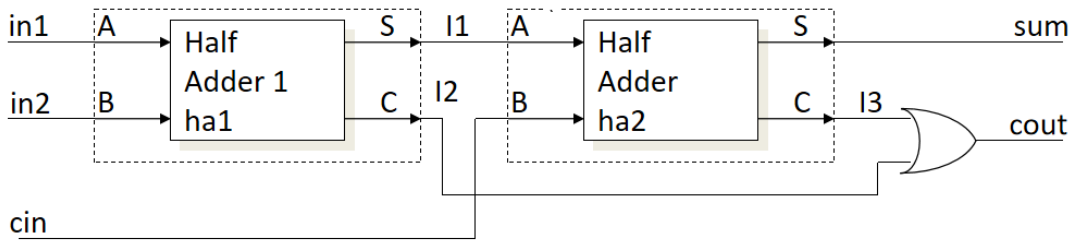


Fig 2b. Full adder using 2 half adders & OR gate

Verilog

```
module half_adder(S, C, A, B);
output S, C;
input A, B;
assign S = A ^ B;
assign C = A & B;
endmodule
module full_adder(sum, cout, in1, in2, cin);
```

```
output sum, cout;
input in1, in2, cin;
wire I1, I2, I3;
half_adder ha1(I1, I2, in1, in2);
half_adder ha2(sum, I3, I1, cin);
assign cout = I2 || I3;
endmodule
```

Test Bench:

```
module fulladder_tb;
reg  a,b,cin;
wire sum, cout;
fulladder adder(.a(a), .b(b), .cin(cin), .sum(sum), .cout(cout));
 initial begin
#10; a =1; b =0; cin =0;
#10; a =0; b =1; cin =0;
#10; a=1; b =1; cin=0;
#10; a =0; b =0; cin =1;
#10; a =1; b =0; cin =1;
#10; a =1; b =0; cin =1;
#10; a =0; b =1; cin =1;
#10; a=1; b =1; cin =1;
end
endmodule
```

Results:

Experiment No: 3 Date: _____

RIPPLE CARRY ADDER

Aim: Write a HDL code for 4-bit ripple carry adder using full adders.

Verilog

```
module fulladder(a, b, cin, sum, cout);
input a, b, cin;
output sum, cout;
assign sum = a^b^cin;
assign cout = (a&b)|(b&cin)|(a&cin);
endmodule

module ripple_add(a,b,cin,s,carry,);
output [3:0]s;
output carry;
input [3:0]a,b;
input cin;
wire c1,c2,c3;
fulladder f1(a[0],b[0],cin,s[0],c1);
fulladder f2(a[1],b[1],c1,s[1],c2);
fulladder f3(a[2],b[2],c2,s[2],c3);
fulladder f4(a[3],b[3],c3,s[3],carry);
endmodule
```

Test Bench:

Results:

Experiment No: 4 Date: _____

ADDER SUBTRACTOR

Aim: Write HDL code to realize Adder/Subtractor circuit.

Verilog

```
module adder_subtractor(S, C, V, A, B, Op);
  output [3:0] S;   // The 4-bit sum/difference.
  output C;   // The 1-bit carry/borrow status.
  output V;   // The 1-bit overflow status.
  input [3:0] A;   // The 4-bit augend/minuend.
  input [3:0] B;   // The 4-bit addend/subtrahend.
  input Op;  // The operation: 0 => Add, 1=>Subtract.

  wire  C0; // The carry out bit of fa0, the carry in bit of fa1.
  wire  C1; // The carry out bit of fa1, the carry in bit of fa2.
  wire  C2; // The carry out bit of fa2, the carry in bit of fa3.
  //wire C3; // The carry out bit of fa2, used to generate final carry/borrrow.

  wire  B0; // The xor'd result of B[0] and Op
  wire  B1; // The xor'd result of B[1] and Op
  wire  B2; // The xor'd result of B[2] and Op
  wire  B3; // The xor'd result of B[3] and Op
   // Looking at the truth table for xor we see that
  // B xor 0 = B, and
  // B xor 1 = not(B).
  // So, if Op==1 means we are subtracting, then
  // adding A and B xor Op alog with setting the first
  // carry bit to Op, will give us a result of
  // A+B when Op==0, and A+not(B)+1 when Op==1.
```

```
  // Note that not(B)+1 is the 2's complement of B, so
  // this gives us subtraction.
  xor(B0, B[0], Op);
  xor(B1, B[1], Op);
  xor(B2, B[2], Op);
  xor(B3, B[3], Op);
 // xor(C, C3, Op);     // Carry = C3 for addition, Carry = not(C3) for subtraction.
 // xor(V, C3, C2);     // If the two most significant carry output bits differ, then we have an overflow.

  full_adder fa0(S[0], C0, A[0], B0, Op);    // Least significant bit.
  full_adder fa1(S[1], C1, A[1], B1, C0);
  full_adder fa2(S[2], C2, A[2], B2, C1);
  full_adder fa3(S[3], C, A[3], B3, C2);    // Most significant bit.
endmodule // ripple_carry_adder_subtractor

module full_adder (sum, cout,a, b, cin, );
input a, b, cin;
output sum, cout; wire s1,s2,s3;
xor x1(sum,a,b,cin);
and a1(s1,a,b);
and a2(s2,a,cin);
and a3(s3,b,cin);
 or o1(cout,s1,s2,s3);
endmodule
```

Test Bench:

Results:

Experiment No:5 Date: _____

Combinational circuits design

Experiment No:5a

DECODER (2:4)

Aim: Write a HDL code to design 2:4 DECODER

Verilog

```
module decoder(I,D);
input [1:0] I;
output [3:0] D;
reg [3:0] D;
always @(I)
begin
if (I==2'B00) D=4'B0001;
else if (I==2'B01) D=4'B0010;
else if (I==2'B10) D=4'B0100;
else if (I==2'B11) D=4'B1000;
else D=4'BZZZZ;
end
endmodule
```

Test Bench:

Results:

Experiment No:5b Date: _____

ENCODER (8:3)

Aim: Write a HDL code to design 8:3 ENCODER with and without priority.

Without priority Verilog

```
module encoder_wop (I,D);
input [7:0] I;
output [2:0] D;
reg [2:0] D;
always @(I)
begin
case (I)
8'd1 : D=3'd0;
8'd2 : D=3'd1;
8'd4 : D=3'd2;
8'd8 : D=3'd3;
8'd16 : D=3'd4;
8'd32 : D=3'd5;
8'd64 : D=3'd6;
8'd128 : D=3'd7;
default: D=3'BZZZ;
endcase
end
endmodule
```

Test Bench:

With Priority Verilog

```
module encoder_wp (I,D);
input [7:0] I;
output [2:0] D;
reg [2:0] D;
always @ (I)
begin
if (I[7]==1'B1) D=3'B111;
else if (I[6]==1'B1) D=3'B110;
else if (I[5]==1'B1) D=3'B101;
else if (I[4]==1'B1) D=3'B100;
else if (I[3]==1'B1) D=3'B011;
else if (I[2]==1'B1) D=3'B010;
else if (I[1]==1'B1) D=3'B001;
else if (I[0]==1'B1) D=3'B000;
else D=3'BZZZ;
end
endmodule
```

Test Bench:

Results:

Experiment No: 5c Date: _____

MULTIPLEXER (8:1)

Aim: Write a HDL code to design 8:1 MULTEPLEXER

Verilog

```
module mux8_1 (I, S, D);
input [7:0] I;
input [2:0] S;
output D;
reg D;
always @ (I, S)
begin
 case(S)
 3’d0: D=I[0];
 3’d1: D=I[1];
 3’d2: D=I[2];
 3’d3: D=I[3];
 3’d4: D=I[4];
 3’d5: D=I[5];
 3’d6: D=I[6];
 3’d7: D=I[7];
default: D=1'BZ;
endcase
end
endmodule
```

Test Bench:

Results:

Experiment No: 5d Date: _____

DEMULTIPLEXER

Aim: Write a HDL code to design 1:8 DEMULTEPLEXER

Verilog

```
module demux1_8(I,S,D);
input I;
input [2:0] S;
output [7:0] D;
reg [7:0] D;
always @(I,S)
begin
D=8'd0;
 CASE (S)
 3'd0 : D[0]=I;
 3'd1 : D[1]=I;
 3'd2 : D[2]=I;
 3'd3 : D[3]=I;
 3'd4 : D[4]=I;
 3'd5 : D[5]=I;
 3'd6 : D[6]=I;
 3'd7 : D[7]=I;
default: D=8'B00000000;
endcase
end
endmodule
```

Test Bench:

Results:

Experiment No: 5e Date: _____

BINARY TO GRAY

Aim: Write a HDL program to convert 4-bit BINARY TO GRAY code.

Verilog

```
module bin_to_gray(B,G);
input [3:0] B;
output [3:0] G;
assign G[3] = B[3];
assign G[2] = B[3]^B[2];
assign G[1] = B[2]^B[1];
assign G[0] = B[1]^B[0];
endmodule
```

Test Bench:

Results:

Experiment No: 5f Date: _____

GRAY TO BINARY

Aim: Write a HDL program to convert 4-bit GRAY TO BINARY code.

Verilog

```
module gray_to_bin
      (input [3:0] G, //gray code output
       output [3:0] bin   //binary input );
 assign bin[3] = G[3];
assign bin[2] = G[3] ^ G[2];
assign bin[1] = G[3] ^ G[2] ^ G[1];
assign bin[0] = G[3] ^ G[2] ^ G[1] ^ G[0];
endmodule
```

Test Bench:

Results:

Experiment No: 5g Date: _____

COMPARATOR

Aim: Write a HDL code to design 4-bit comparator.

Verilog

```
module comparator (a, b, alb, aeb, agb)
input[3:0] a, b;
output alb, aeb, agb;
reg alb, aeb, agb;
always@(a,b)
begin
if(a<b)
begin
alb=1'b1; aeb=1'b0; agb=1'b0;
end
else if (a>b)
begin
alb=1'b0; aeb=1'b0; agb=1'b1;
end
else
begin
alb=1'b0; aeb=1'b1; agb=1'b0;
end
end
endmodule
```

Test Bench:

Results:

Experiment No: 6 Date: _____

BCD ADDER

Aim: Write a HDL code to design BCD Adder.

Verilog

```
//module declaration with inputs and outputs
module bcd_adder(a,b,carry_in,sum,carry);
input [3:0] a,b;
input carry_in;
output [3:0] sum;
output carry;
//Internal variables
reg [4:0] sum_temp;
reg [3:0] sum;
reg carry;
always @(a,b,carry_in)
begin
sum_temp = a+b+carry_in; //add all the inputs
if(sum_temp > 9)    begin
          sum_temp = sum_temp+6; //add 6, if result is more than 9.
          carry = 1;  //set the carry output
          sum = sum_temp[3:0];    end
       else
           begin
          carry = 0;
          sum = sum_temp[3:0];
       end
   end
endmodule
```

Test Bench:

Results:

Experiment No: 7 Date: _____

FLIP-FLOPS

a) D FLIP-FLOP

Aim: Write a HDL code for D-FF.

Verilog

```
module dff(d, rst, clk, q, qb);
input d, rst, clk;
output q, qb;
reg q, qb;
always @(posedge clk)
begin
if (rst == 1'B1)
begin
q = 1'B0; qb = 1'B1;
end
else
begin
q = d; qb = ~q;
end
end
endmodule
```

Test Bench:

Results:

b) T FLIP-FLOP

Aim: Write a HDL code for T-FF.

Verilog

For Simulation:

```
module tff(t, clk, rstn, q, qb);
input t, rstn, clk;
output q, qb;
reg q, qb;
always @(posedge clk)
begin
if (!rstn)
begin
q <= 0;
qb <= 1;
end
else
if (t)
begin
q = ~q; qb = ~qb;
end
else
begin
q = q; qb = qb;
end
end
endmodule
```

Test Bench:

Hardware Implementation code:

```
module tff1(t,clk,rstn,q,qb);
input t,rstn,clk;
output q,qb;
reg q,qb;
reg clkdiv;
reg[24:0] div;
always@(posedge clk)
begin
div=div+1'b1;
clkdiv=div[24];
end
always@(posedge clkdiv)
begin
if(!rstn)
begin
q<=0;
qb<=1;
```

```
end
else
if(t)
begin
q=~q;
qb=~qb;
end
else
begin
q=q;
qb=qb;
end
end
endmodule
```

Results:

c) JK FLIP-FLOP

Aim: Write a HDL code for JK-FF.

Verilog

```
module jkff(jk, clk, q, qb);
input clk;
input [1:0]jk;
output q,qb;
reg q,qb;
always @(posedge clk)
begin
case(jk)
2'B00:q=q;
2'B01:q=1'B0;
2'B10:q=1'B1;
2’B11:q= ~q;
default:q=1'BZ;
endcase
qb=~q;
end
endmodule
```

Test Bench:

Note: Hardware implementation code needs to be written.

Results:

Experiment No: 8 Date: _____

4-BIT ALU

Aim: Write a HDL code to design 4-bit ALU.

Verilog

```
module alu (a, b, code, aluout );
input [3:0] a, b;
input [2:0] code;
output [7:0] aluout;
reg [7:0] aluout ;
wire[7:0] x, y;
assign x = {4'B0000, a};
assign y = {4'B0000, b};
always @ (x, y, code, a,b)
begin
case (code)
3'd0: aluout = x + y;
3'd1: aluout = x - y;
3'd2: aluout = ~x;
3'd3: aluout = a * b;
3'd4: aluout = x & y;
3'd5: aluout = x | y;
3'd6: aluout = ~(x & y);
3'd7: aluout = ~(x | y);
default: aluout = 8'B00000000;
endcase
end
endmodule
```

Test Bench:

Results:

Experiment No: 9 Date: _____

COUNTERS

a) SYNCHRONOUS BINARY COUNTER

Aim: Write a HDL code to design 4-bit synchronous binary counter.

Verilog

```
module syncbinary (clk, rst, q);
input clk, rst;
output [3:0]q;
reg [3:0]q;
initial q = 4'B0000;
always @ (posedge clk)
begin
if (rst == 1'B1)
q = 4'B0000;
else
q = q + 1;
end
endmodule
```

Test Bench:

Hardware Implementation code:

```
module syncbinary (clk, rst, q);
input clk, rst;
output [3:0]q;
reg [3:0]q;
reg clkdiv;
reg[24:0] div;
initial q = 4'B0000;
always @ (posedge clk)
begin
div=div+1'B1;
clkdiv=div[24];
end
always@(posedge clkdiv)
begin
if (rst == 1'B1)
q = 4'B0000;
else
q = q + 1;
end
endmodule
```

Results:

b) SYNCHRONOUS BCD COUNTER

Aim: Write a HDL code to design 4-bit synchronous BCD counter.

Verilog

```
module syncbcd (clk, rst, q);
input clk, rst;
output [3:0]q;
reg [3:0]q;
initial q = 4'b0000;
always @ (posedge clk)
begin
if (rst == 1'b1|q==4'b1001)
q = 4'b0000;
else
q=q+1;
end
endmodule
```

Test Bench:

Note: Hardware implementation code needs to be written.

Results:

c) ASYNCHRONOUS BINARY COUNTER

Aim: Write a HDL code to design 4-bit Asynchronous Binary counter.

Verilog

```
module asyncbinary (clk, rst, q);
input clk, rst;
output [3:0]q;
reg [3:0]q;
initial q = 4'b0000;
always @ (posedge clk or posedge rst)
begin
if (rst == 1'b1)
q = 4'b0000;
else
q=q+1;
end
endmodule
```

Test Bench:

Note: Hardware implementation code needs to be written.

Results:

d) ASYNCHRONOUS BCD COUNTER

Aim: Write a HDL code to design 4-bit Asynchronous BCD counter.

Verilog

```
module asyncbcd (clk, rst, q);
input clk, rst;
output [3:0]q;
reg [3:0]q;
initial q = 4'b0000;
always @ (posedge clk or posedge rst)
begin
if (rst == 1'b1)q = 4'b0000;
else if (q== 4'b1001) q = 4'b0000;
else
q=q+1;
end
endmodule
```

Test Bench:

Hardware Implementation code:

```
module asyncbcd (clk, rst, q);
input clk, rst;
output [3:0]q;
reg [3:0]q;
reg clkdiv;
reg[22:0] div;
initial q = 4'b0000;
always @ (posedge clk )
begin
div=div+1'B1;
clkdiv=div[22];
end
always@(posedge clkdiv or posedge rst)
begin
if (rst == 1'b1)
q = 4'b0000;
else if (q== 4'b1001)
q = 4'b0000;
else
q=q+1;
end
endmodule
```

Results:

Experiment No: 10 Date: _____

MEALY AND MOORE STATE MACHINES

a) MEALY STATE MACHINE

Aim: Write a HDL code to design Mealy state machine for the sequence 101.

Verilog

```
//Mealy Sequence -101
module seq_mealy(
input x, input clk , input rst , output reg y );
reg [1:0]cst;
reg [1:0]nst;
parameter s0 = 2'b00 , s1 = 2'b01 , s2 = 2'b10;
always @(cst or x)
begin
case(cst)
s0: begin
   if (x)
         begin
         nst = s1;
         y=0;
         end
         else
         begin
         nst = cst;
         y=0;
         end
         end
s1: begin
```

```
        if (x==0)
        begin
        nst = s2;
        y=0;
        end
        else
        begin
        nst = cst;
        y=0;
        end
        end
s2: begin
        if (x)
        begin
        nst = s1;
        y=1;
        end
        else
        begin
        nst = s0;
        y=0;
        end
        end
        default: nst = s0 ;
endcase
end
always@(posedge clk)
begin
```

```
if(rst)
cst <= s0;
else
cst <= nst;
end
endmodule
```

Test Bench:

```
module seq_mealy_tb;
// Inputs
        reg x; reg clk; reg rst;
// Outputs
        wire y;
// Instantiate the Unit Under Test (UUT)
        seq_mealy uut (
                .x(x),
                .clk(clk),
                .rst(rst),
                .y(y)
        );
reg [19:0]data;
        integer k;
        initial begin
                // Initialize Inputs
                data = 20'b10100101011101010101;
                clk = 0;
                k=0;
                rst = 1;
```

```
                #60 rst =0 ;
// Wait 100 ns for global reset to finish
#340; // Add stimulus here
        end
        always@(posedge clk)
        begin
        x = data>>k;
        k = k+1;
        end
    always #20 clk=~clk;
endmodule
```

Results:

b) MOORE STATE MACHINE

Aim: Write a HDL code to design Moore state machine to detect the Sequence -101

Verilog

```
module seq_moore(input x, input clk , input rst , output reg y );
reg [1:0]cst;
reg [1:0]nst;
parameter s0 = 2'b00 , s1 = 2'b01 , s2 = 2'b10 , s3 = 2'b11;
always @(cst or x)
begin
case(cst)
s0: begin
   y=0;
        if (x)
        nst = s1;
        else
        nst = cst;
        end
s1: begin
   y=0;
        if (x==0)
        nst = s2;
        else
        nst = cst;
        end
s2: begin
   y=0;
        if (x)
        nst = s3;
```

```
        else
        nst = s0;
        end
s3: begin
   y=1;
        if (x)
        nst = s1;
        else
        nst = s2;
        end
default: nst = s0 ;
endcase
end
always@(posedge clk)
begin
if(rst)
cst <= s0;
else
cst <= nst;
end
endmodule
```

Test Bench:

```
module seq_moore_tb;
        // Inputs
        reg x;
        reg clk;
        reg rst;
```

```
        // Outputs
        wire y;
        // Instantiate the Unit Under Test (UUT)
        seq_moore uut (
                .x(x),
                .clk(clk),
                .rst(rst),
                .y(y)
        );
reg [15:0]data;
        integer k;
        initial begin
                // Initialize Inputs
                data = 16'b1010010101110101;
                clk = 0;
                k=0;
                rst = 1;
                #60 rst =0 ;
                // Wait 100 ns for global reset to finish
                #500;
                // Add stimulus here
                $stop;
        end
        always@(posedge clk)
        begin
        x = data>>k;
        k = k+1;
        end
```

```
    always #20 clk=~clk;
endmodule
```

Results:

Experiment No: 11 Date: _____

SEVEN SEGMENT DISPLAY

a) Aim: Write a HDL code to display the word **"dsce"** using Seven Segment Display.

Verilog

```
module dsce (SW, HEX0);
   input [1:0] SW;
   output reg [6:0] HEX0;
always@(SW)
begin
case(SW)
        2'b00: HEX0 = 7'b0100001;
        2'b01: HEX0 = 7'b0010010;
        2'b10: HEX0 = 7'b1000110;
        2'b11: HEX0 = 7'b0000110;
endcase
end
endmodule
```

b) Aim: Write a HDL code to display **"Binary to decimal"** using Seven Segment Display.

Verilog

```
module b2d_ssd (SW, HEX0, HEX1);
  input [7:0] SW;
  output [6:0] HEX0,HEX1;

  b2d_7seg B0 (SW[3:0], HEX0);
  b2d_7seg B1 (SW[7:4], HEX1);
endmodule
```

```
module b2d_7seg (X, SSD);
    input [3:0] X;
    output reg [6:0] SSD;
            always@(X)
            begin
            case(X)
            4'b0000 : SSD=7'b1000000;
            4'b0001 : SSD=7'b1111001;
            4'b0010 : SSD=7'b0100100;
            4'b0011 : SSD=7'b0110000;
            4'b0100 : SSD=7'b0011001;
            4'b0101 : SSD=7'b0010010;
            4'b0110 : SSD=7'b0000010;
            4'b0111 : SSD=7'b1111000;
            4'b1000 : SSD=7'b0000000;
            4'b1001 : SSD=7'b0010000;
            endcase
            end
endmodule
```

Results:

Experiment No: 12 Date: _____

VGA DISPLAY

Aim: Write a HDL code to to display the color images using VGA cable.

Verilog

```
module vga_display(SW,CLOCK_50,green,red,blue,hsync,vsync);
input CLOCK_50;
input [2:0] SW;
output reg [2:0] green , red, blue;
output hsync,vsync;
VGADemo demo(CLOCK_50, pixel,hsync_out,vsync_out);
always @ (posedge CLOCK_50) begin
if (SW == 2'b00) begin
red = pixel;
end
else if (SW == 2'b01) begin
green = pixel;
end
else if (SW == 2'b10) begin
blue = pixel;
end
end
assign hsync = hsync_out;
assign vsync = vsync_out;
endmodule
module VGADemo(
   input CLOCK_50,
   output reg [2:0] pixel,
   output hsync_out,
```

```
  output vsync_out
);
  wire inDisplayArea;
  wire [9:0] CounterX;
reg clk_25;
always @ (posedge CLOCK_50) begin
clk_25 = ~clk_25;
end
   hvsync_generator hvsync(
    .clk(clk_25),
    .vga_h_sync(hsync_out),
    .vga_v_sync(vsync_out),
    .CounterX(CounterX),
    .inDisplayArea(inDisplayArea)
  );
always @(posedge clk_25)
  begin
    if (inDisplayArea)
      pixel <= CounterX[9:6];
    else // if it's not to display, go dark
      pixel <= 3'b000;
  end
endmodule
module hvsync_generator(
  input clk,
  output vga_h_sync,
  output vga_v_sync,
  output reg inDisplayArea,
```

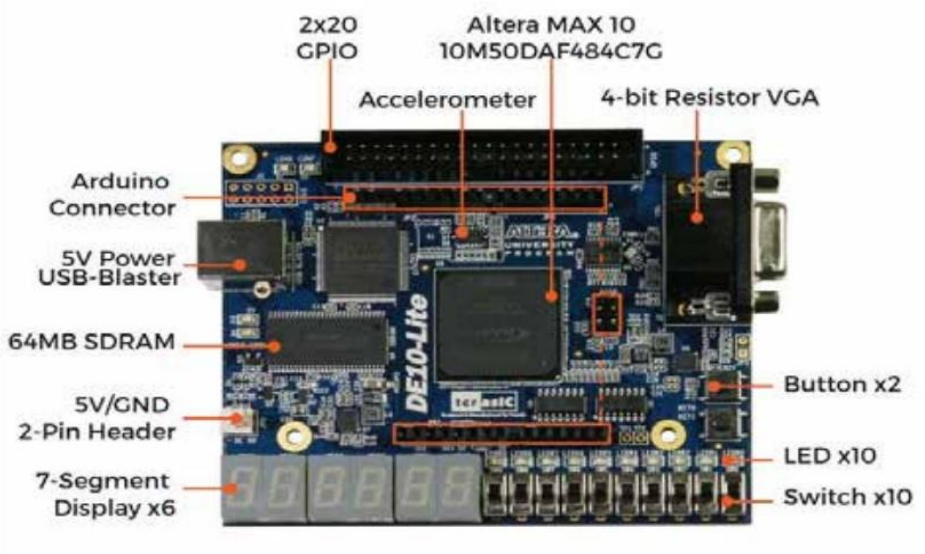
```
  output reg [9:0] CounterX,
  output reg [9:0] CounterY
);
  reg vga_HS, vga_VS;
  wire CounterXmaxed = (CounterX == 800); // 16 + 48 + 96 + 640
  wire CounterYmaxed = (CounterY == 525); // 10 + 2 + 33 + 480
  always @(posedge clk)
  if (CounterXmaxed)
    CounterX <= 0;
  else
    CounterX <= CounterX + 1;
  always @(posedge clk)
  begin
    if (CounterXmaxed)
    begin
      if(CounterYmaxed)
        CounterY <= 0;
      else
        CounterY <= CounterY + 1;
    end
  end
  always @(posedge clk)
  begin
  vga_HS <= (CounterX > (640 + 16) && (CounterX < (640 + 16 + 96)));   //active for 96 clocks
  vga_VS <= (CounterY > (480 + 10) && (CounterY < (480 + 10 + 2)));   // active for 2 clocks
  end
always @(posedge clk)
  begin
```

```
    inDisplayArea <= (CounterX < 640) && (CounterY < 480);
  end
   assign vga_h_sync = ~vga_HS;
  assign vga_v_sync = ~vga_VS;
endmodule
```

Results:

DE10 lite FPGA Board

The DE10-Lite presents a robust hardware design platform built around the Altera MAX 10 FPGA. The MAX 10 FPGA is well equipped to provide cost effective, single-chip solutions in control plane or data path applications and industry-leading programmable logic for ultimate design flexibility. With MAX 10 FPGA, you can get lower power consumption / cost and higher performance. When you need high-volume applications, including protocol bridging, motor control drive, analog to digital conversion, image processing, and handheld devices, the MAX 10 Lite FPGA is your best choice. The DE10-Lite development board includes hardware such as on-board USB Blaster, 3-axis accelerometer, video capabilities and much more. By leveraging all of these capabilities, the DE10-Lite is the perfect solution for showcasing, evaluating, and prototyping the true potential of the Altera MAX 10 FPGA. The DE10-Lite contains all components needed to use the board in conjunction with a computer that runs the Microsoft Windows XP or later.



Download the user manual here:
https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/Boards/DE10-Lite/DE10_Lite_User_Manual.pdf

DESim tool and steps:

1) Go to https://github.com/fpgacademy/DESim

2) Download and install Modelsim Intel FPGA Starter Edition (link in the README.md) and DESim (under "Releases" section)

3) Open the folder containing the DESim installation (C:\DESim by default)

4) Open the demos folder

5) You will find demo projects for compilation and simulation on DESim

6) Create a new folder, this will be your project folder

7) Open the Counter demo project folder (C:\ DESim\demos\Counter) and copy the "sim" and "tb" folders as well as the "top.v" file to your empty project folder, this is required for any new DESim project. "sim" and "tb" remain unchanged while "top.v" is modified slightly for different designs.

8) Copy your Verilog design file to your project folder.

9) Open "top.v" in your project folder and change the DUT instantiation name from "Counter" to your DUT name. Also make sure that the ports are set correctly.

10) If your verilog files use a clock, you must rename your input clk variable to "CLOCK_50" in both the design files. DESim uses a 50 MHz clock by default. You may use a clock divider if necessary.

11) All output variables should be assigned to "output [9:0] LEDR" in the design. This is because DESim has 10 LED output lights. For example, if you are synthesizing a counter with 4 outputs "q0", "q1", "q2" and "q3" then assign these variables to "[3:0] LEDR".

12) All input variables using a switch should be renamed to "input [9:0] SW" in the design. This is because DESim has 9 input switches controlled by the 10-bit reg "SW".

13) All input variables using a push button should be renamed to "input [3:0] KEY" in the design. This is because DESim has 4 push buttons controlled by the 4-bit reg "KEY".

14) Open the DESim program from your start menu/desktop shortcut/DESim installation folder (DESim_run.bat)

15) Click on "Open Project"

16) Navigate to your project folder. single-click it and click "Select Folder"

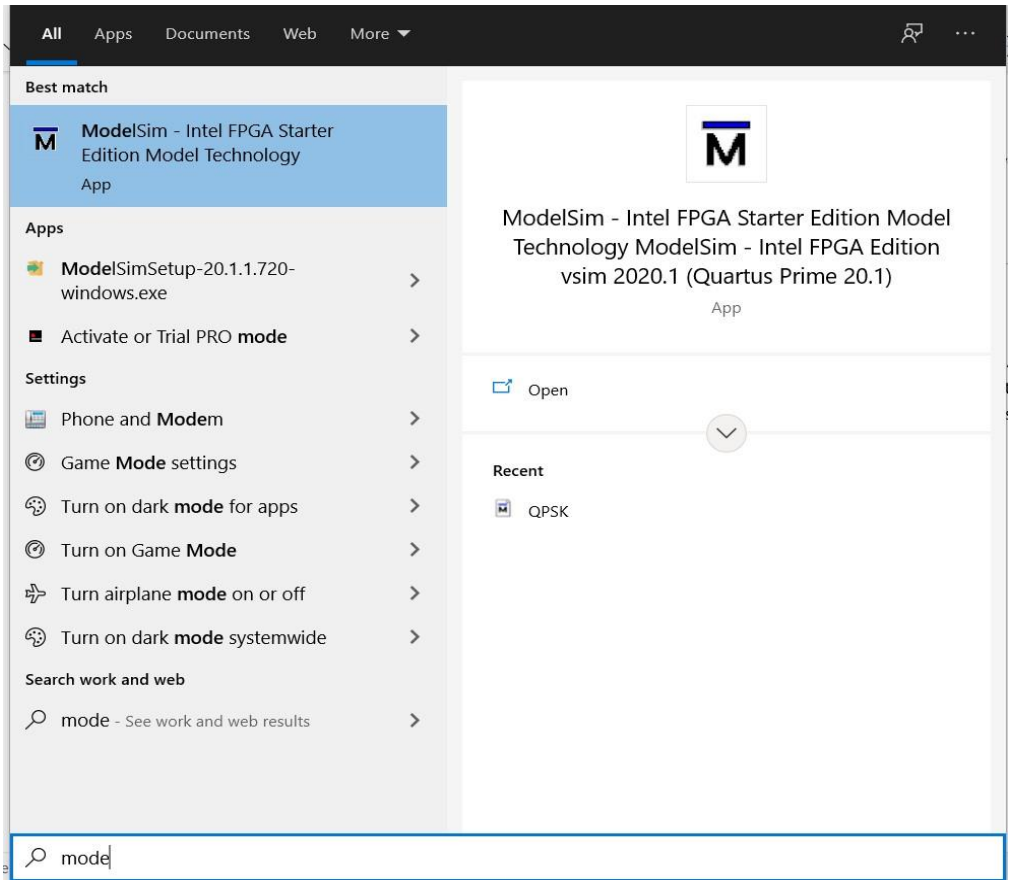
17) Click on "Compile Testbench"

18) If compilation is successful, click on "Start Simulation"

19) Provide inputs as per your code and Verify outputs by looking at the LEDs.20)

ModelSim Tool and Simulation Steps:

Go to start and type modelsim and run as adminstrator

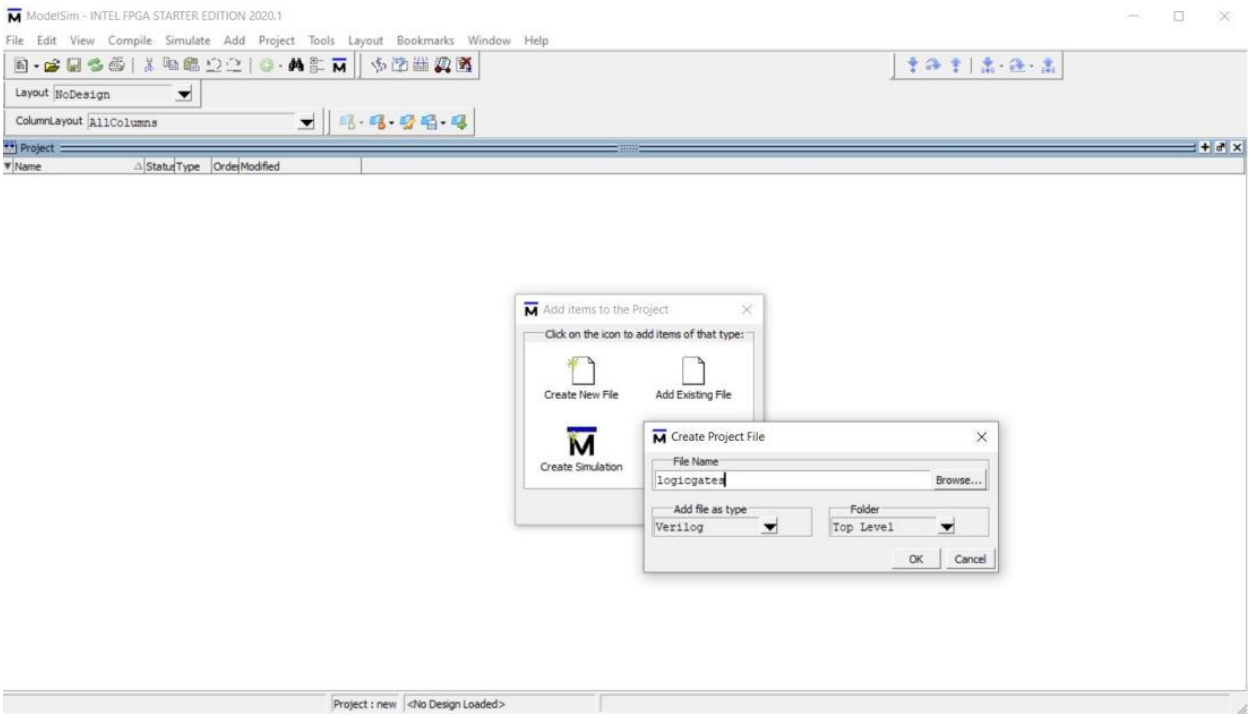


File>>new>> project

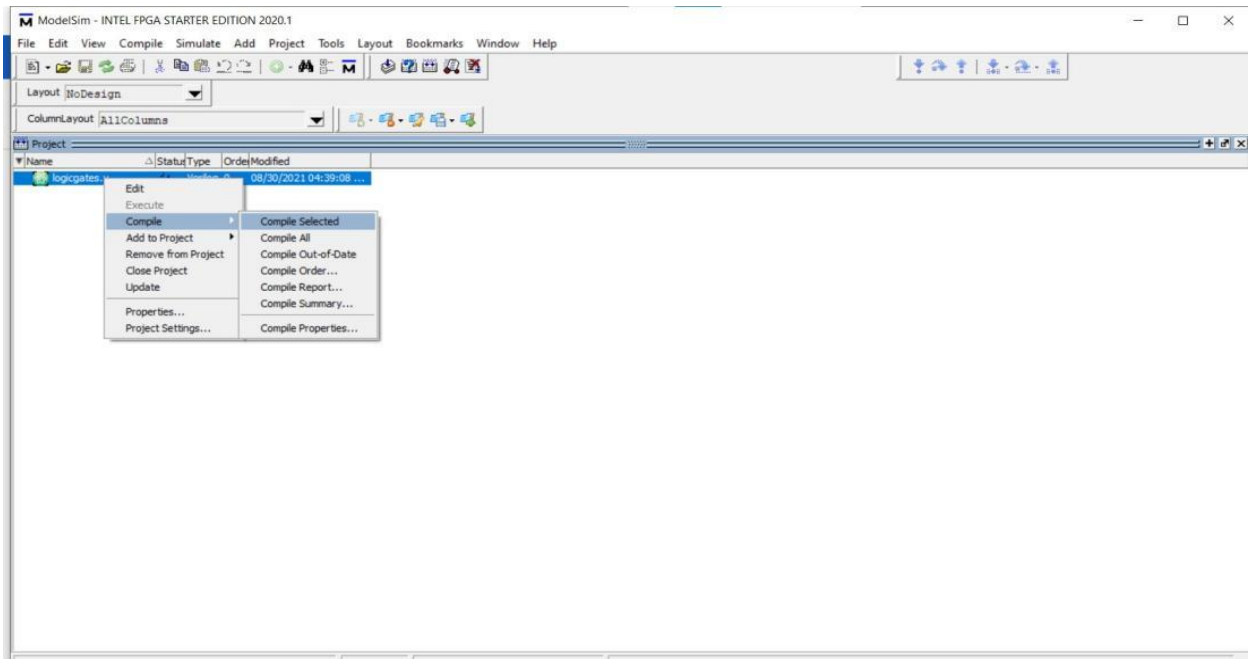
Give some project name and click ok.

Click Create New File

Select verilog and give file name



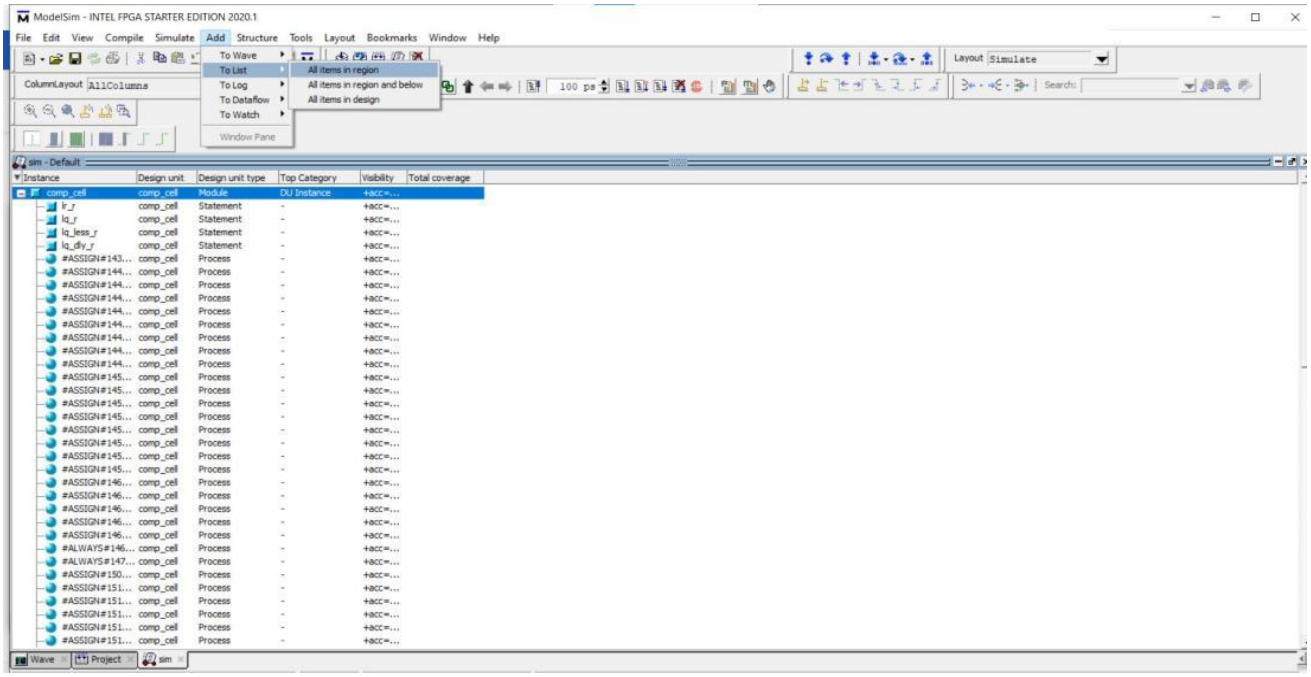
After writing code, compile the selected verilog file. You can also type a testbench file in the same file.



Go to simulate, start simulation and click on + mark near work folder.

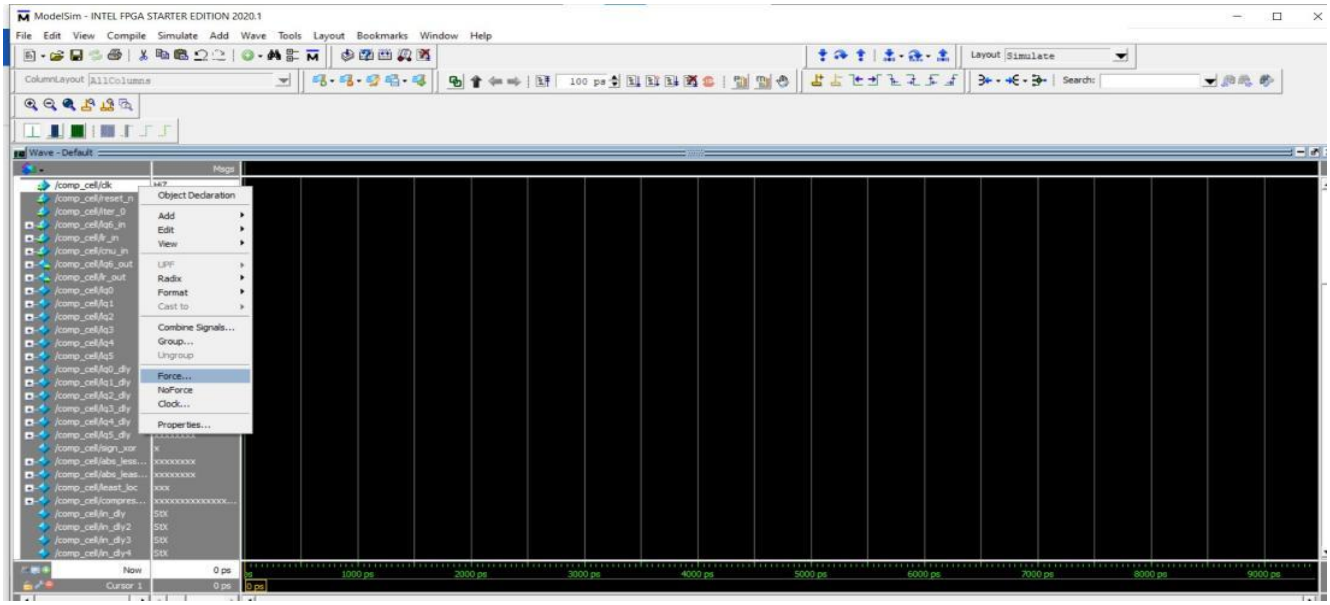
If you have a testbench select that, or else select your top module

Go to Add>> to wave >> All items in the region



Type run or run –all in transcript or use icon.

If you don't have testbench you will get force option near the waveform window.

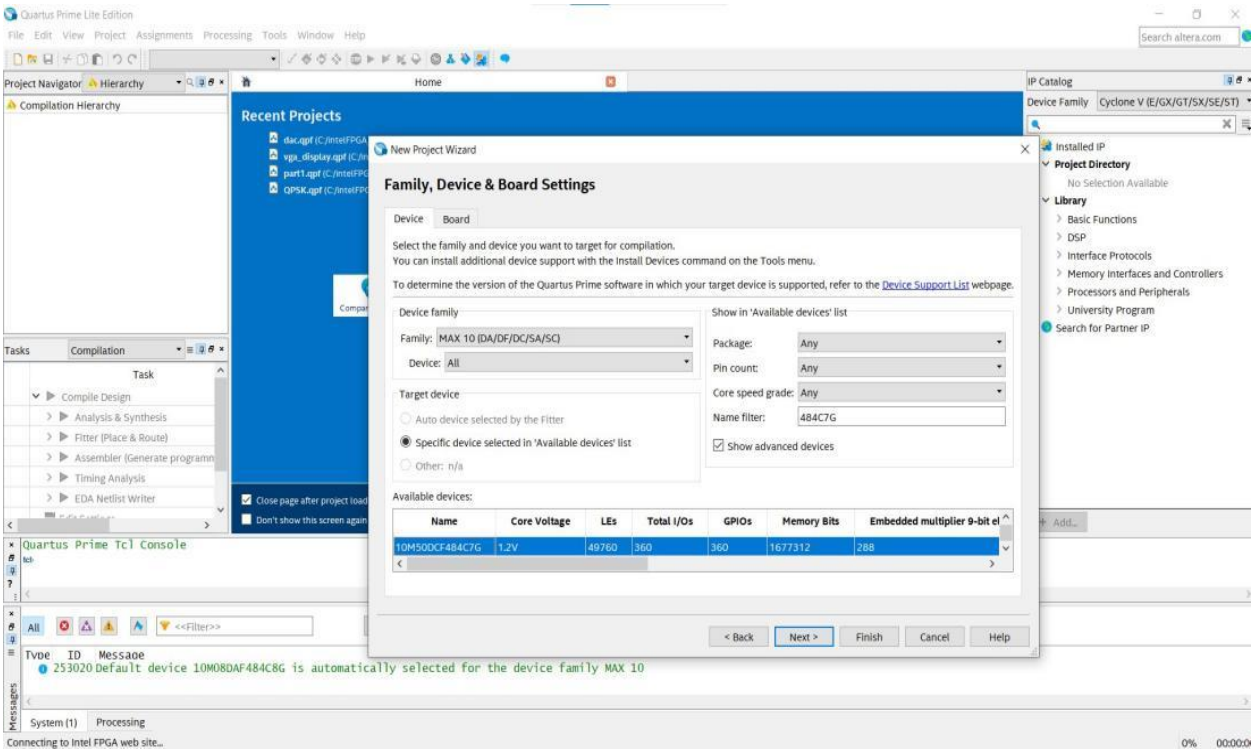


Run and view the waveforms and messages in the transcript

Quartus Prime Tool and Steps:

Run the Quartus Prime as administrator

New project wizard >> next>> Give your top module name to the project.>>no>>empty project>>next>>select family as max10 and 10M50DCF484C7G>>next and finish



Go to file>> new>>verilog hdl file>> give same name as your top module

Write the verilog code

Now view>>utility windows>>tcl console and type the set_pin_assignment tcl file there.

Pin assignments:

set_location_assignment PIN_P11 -to CLOCK_50

set_location_assignment PIN_A8 -to LEDR[0]

set_location_assignment PIN_A9 -to LEDR[1]

set_location_assignment PIN_A10 -to LEDR[2]

set_location_assignment PIN_B10 -to LEDR[3]

set_location_assignment PIN_D13 -to LEDR[4]

set_location_assignment PIN_C13 -to LEDR[5]

set_location_assignment PIN_E14 -to LEDR[6]

set_location_assignment PIN_D14 -to LEDR[7]

set_location_assignment PIN_A11 -to LEDR[8]

set_location_assignment PIN_B11 -to LEDR[9]

set_location_assignment PIN_C10 -to SW[0]

set_location_assignment PIN_C11 -to SW[1]

set_location_assignment PIN_D12 -to SW[2]

set_location_assignment PIN_C12 -to SW[3]

set_location_assignment PIN_A12 -to SW[4]

set_location_assignment PIN_B12 -to SW[5]

set_location_assignment PIN_A13 -to SW[6]

set_location_assignment PIN_A14 -to SW[7]

set_location_assignment PIN_B14 -to SW[8]

set_location_assignment PIN_F15 -to SW[9]

set_location_assignment PIN_B8 -to KEY[0]

set_location_assignment PIN_A7 -to KEY[1]

set_location_assignment PIN_C14 -to HEX0[0]

set_location_assignment PIN_E15 -to HEX0[1]

set_location_assignment PIN_C15 -to HEX0[2]

set_location_assignment PIN_C16 -to HEX0[3]

set_location_assignment PIN_E16 -to HEX0[4]

set_location_assignment PIN_D17 -to HEX0[5]

set_location_assignment PIN_C17 -to HEX0[6]

set_location_assignment PIN_C18 -to HEX1[0]

set_location_assignment PIN_D18 -to HEX1[1]

set_location_assignment PIN_E18 -to HEX1[2]

set_location_assignment PIN_B16 -to HEX1[3]

set_location_assignment PIN_A17 -to HEX1[4]

set_location_assignment PIN_A18 -to HEX1[5]

set_location_assignment PIN_B17 -to HEX1[6]

set_location_assignment PIN_B20 -to HEX2[0]

set_location_assignment PIN_A20 -to HEX2[1]

set_location_assignment PIN_B19 -to HEX2[2]

set_location_assignment PIN_A21 -to HEX2[3]

set_location_assignment PIN_B21 -to HEX2[4]

set_location_assignment PIN_C22 -to HEX2[5]

set_location_assignment PIN_B22 -to HEX2[6]

set_location_assignment PIN_F21 -to HEX3[0]

set_location_assignment PIN_E22 -to HEX3[1]

set_location_assignment PIN_E21 -to HEX3[2]

set_location_assignment PIN_C19 -to HEX3[3]

set_location_assignment PIN_C20 -to HEX3[4]

set_location_assignment PIN_D19 -to HEX3[5]

set_location_assignment PIN_E17 -to HEX3[6]

set_location_assignment PIN_F18 -to HEX4[0]

set_location_assignment PIN_E20 -to HEX4[1]

set_location_assignment PIN_E19 -to HEX4[2]

set_location_assignment PIN_J18 -to HEX4[3]

set_location_assignment PIN_H19 -to HEX4[4]

set_location_assignment PIN_F19 -to HEX4[5]

set_location_assignment PIN_F20 -to HEX4[6]

set_location_assignment PIN_J20 -to HEX5[0]

set_location_assignment PIN_K20 -to HEX5[1]

set_location_assignment PIN_L18 -to HEX5[2]

set_location_assignment PIN_N18 -to HEX5[3]

set_location_assignment PIN_M20 -to HEX5[4]

set_location_assignment PIN_N19 -to HEX5[5]

set_location_assignment PIN_N20 -to HEX5[6]

set_location_assignment PIN_AA1 -to red[0]

set_location_assignment PIN_V1 -to red[1]

set_location_assignment PIN_Y2 -to red[2]

set_location_assignment PIN_Y1 -to red[3]

set_location_assignment PIN_W1 -to green[0]

set_location_assignment PIN_T2 -to green[1]

set_location_assignment PIN_R2 -to green[2]

set_location_assignment PIN_R1 -to green[3]

set_location_assignment PIN_P1 -to blue[0]

set_location_assignment PIN_T1 -to blue[1]

set_location_assignment PIN_P4 -to blue[2]

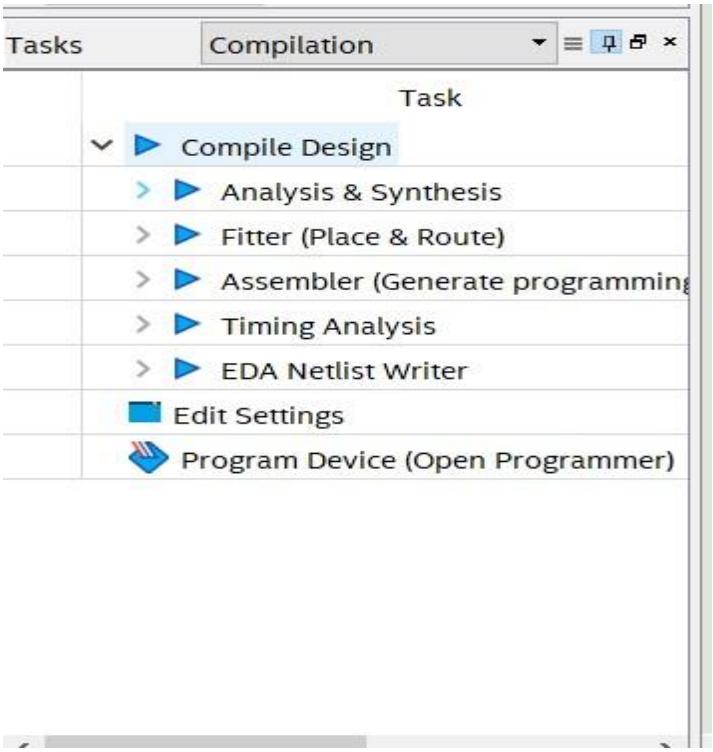
set_location_assignment PIN_N2 -to blue[3]

set_location_assignment PIN_N3 -to hsync

set_location_assignment PIN_N1 -to vsync

Alternatively, you can go to assignments and select manually using pin planner and visualize them as well.

Compile the design



After successful compilation click program device

Select hardware setup and make sure that board is connected and usb blaster is detected.

Click start, now the FPGA device is ready for simulation.