# DEPARTMENT
# OF
# ELECTRONICS & COMMUNICATION ENGINEERING

## ABILITY ENHANCEMENT IV

## Digital Circuit Design with FPGA Laboratory Manual

## IV Semester (22ECL471) Autonomous Course

**HOD**

**Dr. SHOBHA K R**

Lab Manual

Prepared by

**Dr. MANASA R**

**Prof. LAVANYA K M**

Lab Instructors

**Mr. Pradhymna & 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)**


**2024-2025**

# Dayananda Sagar College of Engineering

## Dept. of E & C Engg

| | | |
|---|---|---|
| **Name of the Laboratory** | **:** | **Digital Circuit Design with FPGA Lab** |
| **Semester/Year** | **:** | IV/ 2025(Autonomous) |
| **No. of Students/Batch** | **:** | 22 |
| **No. of Equipment's** | **:** | 60 |
| **Major Equipment's** | **:** | DE-1 SoC P0159-OW-AUP-2, DE-10 Lite P0466-Oh-AUP-7, Qurtus prime 20.1 & Intel FPGA-10 Kits, Modelsim Version 17.1 PCs with peripherals |
| **Area in square meters** | **:** | 100.7 Sq Mts |
| **Location** | **:** | Level – 2 |
| **Total Cost of Lab** | **:** | Rs. 24,58,000/- |
| **Lab In charge/s** | **:** | Dr. Manasa.R |
| **Instructor** | **:** | Mr. Pradhymna & Mrs. Divya N |
| **HOD** | **:** | Dr. SHOBHA K R |

## About the College

The Dayananda Sagar College of Engineering was established in 1979 by Sri R. Dayananda Sagar. Dayananda Sagar College of Engineering operates under the aegis of the **Mahatma Gandhi Vidya Peetha Trust** is approved by All India Council for Technical Education (AICTE), Govt. of India and affiliated to Visvesvaraya Technological University. It has the widest choice of engineering branches having 20 Under Graduate courses & 6 Post Graduate courses. In addition, it has 20 Research Centres in different branches of Engineering catering to research scholars for obtaining Ph.D under VTU. Various courses are accredited by NBA.The Institute is spread over 23 acres of land with large infrastructure supported by laboratories with state-of-the-art, Equipment & Machines. The Central Library with modern facilities and the Digital Library provides the knowledge base for the students.The campus is WI-FI equipped with large bandwidth internet facility. The College has good faculty strength with highest professional integrity and committed to the academics with transparency in their actions. Each faculty takes the responsibility of mentoring a certain number of students' through personal attention paving the way for the students' professional growth. The faculty are research oriented having number of sponsored R & D projects from different agencies such as Department of Science & Technology, Defence R & D organizations, Indian Space Research Organization, AICTE etc..

## About the Department

Department of Electronics and Communication Engineering is one of the oldest and biggest department in the DSI group with 50 faculty members and 1200+ students. Most of our faculty have pursued Ph.D. and others are in the process. The ECE department provides high-quality, innovative technical education at the undergraduate, graduate, and research levels. At present, the department offers an UG course (BE) with an intake of 240, a PG course in VLSI Design and Embedded Systems with an intake of 18 students and Ph.D . The department has got an excellent infrastructure with sophisticated labs, class rooms and R & D center. The department faculty and support personnel are dedicated towards helping students achieve success in today's world by offering them the knowledge and skills required. The department places a strong emphasis on leading-edge research through its research centre and research forum. These factors together enable our graduating students to create a big impact on the workforce from day one.In addition to offering innovative courses, the department is more focused on improving students' educational experiences by offering value-added courses, skill-development programs, technical challenges and hackathons at state, national, and international arenas. Through its professional societies and technical clubs, the department takes great satisfaction in exposing its students to current industry developments and upcoming technologies. The dedicated staff at the ECE department help students procure jobs in prestigious companies both domestically and abroad.

## *Vision of the College:*

To impart quality technical education with a focus on Research and Innovation emphasizing 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 focuson 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.

## *Program 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.

## ***Programme Outcomes [POs]***

Engineering Graduates will be able to:

1. **Engineering Knowledge:** Apply the Knowledge of Mathematics, Science, Engineering Fundamentals, and an Engineering specialization to the solution of complex Engineering problems.
2. **Problem Analysis:** Identify, Formulate, Review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of Mathematics, natural sciences and engineering sciences.
3. **Design/Development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental conditions.
4. **Conduct investigations on complex problems:** Use research based knowledge and research methods including design of Experiments, analysis and interpretation of data, and synthesis of Information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate technique, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess society, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Lifelong learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**Ability Enhancement Course-II (ABNC Lab)**
**(Digital Circuit Design with FPGA) - HDL**

**Course Code :** 22ECL471 **Credits :** 01
**L : T : P :** 0 : 0 : 2 : (2 periods per week of lab) **CIE Marks :** 50
**Exam Hours :** 03 **SEE Marks :** 50
**Total Hours :** 24 (14 weeks) **CIE + SEE Marks** : 100

**Ability Enhancement Course Objectives :**

| | |
|---|---|
| 1. | To describe the functionality of combinational and sequential circuits using HDL. |
| 2. | To Learn how to simulate, analyze, verify the designed program |
| 3. | To realize Verilog code using Test bench and also evaluate the design on FPGA |
| 4. | To give an in-depth exposure to the various tools in Intel Quartus Prime/Modelsim |

**Ability Enhancement Course Outcomes :**

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

| | |
|---|---|
| CO1 | Understand the use of Intel Quartus Prime, ModelSim, and Verilog tools for digital circuit design. |
| CO2 | Apply knowledge of coding techniques to simulate and verify digital circuits using Verilog |
| CO3 | Analyze and synthesize digital circuits using FPGA kits |
| CO4 | Develop and implement digital systems to address real-time problems on FPGA kits. |

**Mapping of Course Outcomes to Program Outcomes:**

| | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CO1 | | | | | | | | | | | | | | |
| CO2 | 3 | | | | | | | | | | | | | |
| CO3 | | 3 | 2 | | 3 | | | | | | | | | |
| CO4 | | | | 3 | 3 | 2 | | 1 | 2 | 2 | | | 2 | 2 |

Weightage Values: 3 = Strongly matched, 2 = Moderately matched, 1 = Weakly matched, (--) = Not matched.

| Expt. No. | Ability Enhancement Course Details | Hours |
|---|---|---|
| | **Experiments** | |
| 1 | Write a HDL program for the following combinational designsusing FPGA implementation<br>a. 2 to 4 Decoder.<br>b. 8 to 3 Encoder (without priority & with priority).<br>c. 4-bit Binary to Gray converter.<br>d. 4-bit Gray to Binary converter | 02 |
| 2 | Write a HDL program using generate for 4 bit parallel adder using FPGA implementation | 02 |
| 3 | Write HDL code for Carry look ahead adder using FPGA implementation | 02 |
| 4 | 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.<br>Opcode (2:0)<br>OPCODE ALU OPERATION<br>i) A + B<br>ii) A – B<br>iii) A Complement<br>iv) A * B<br>v) A AND B<br>vi) A OR B<br>vii) A NAND B<br>viii) A XOR B | 02 |
| 5 | Develop the HDL code for the following flip-flops using FPGA implementation<br>a) D Flip-Flop<br>b) T Flip-Flop<br>c) JK Flip-Flop | 04 |
| 6 | Write HDL code and implement on FPGA for 4 x 4 Multiplier | 02 |
| 7 | Write a Verilog code using task and function that will check the parity of a word for even parity and implement the same on FPGA | 02 |
| 8 | Write a Verilog code for 4 bit Linear feedback shift register(LFSR) using FPGA implementation. | |
| 9 | Design and implement 4 bit Binary and BCD counters Synchronous reset counters using Intel FPFA | 02 |
| 10 | Design and implement 4 bit Binary and BCD counters Asynchronous reset counters using Intel FPFA | |
| 11 | Design Mealy and Moore state machine for a given sequence. | 04 |
| 12 | 2 Open ended experiments | 02 |

**Self – Study Components** : Different types of FPGA.

**Pre-Requisites / Pre-amble :** Digital fundamentals**,** Few tools & hardware kits have to be known before performing the simulation and verifying it with the hardware like Intel Quartus Prime/De-Sim/Modelsim/iverilog/Xilinx ISE tool.

**Teaching Practice:**

Group project discussion.

ICT – Power Point Presentations.

Practical classes in outdoor and indoor as per requirements.

Hands on skills using programming tools.

Development of project-based learning skill sets.

Display of video programming skills

**On-Line Materials & Resources (NPTEL courses / Video lectures / You-tube Videos / Power points / On-line notes / Web-links / Case studies / E-Text Books / Digital Library / Virtual Labs, etc…**

| | |
|---|---|
| 1. | https://opencourses.gr/index.xhtml?ln=en |
| 2. | https://github.com/topics/verilog-project |
| 3. | https://edaplayground.com/ |

**ABNC Laboratory References :**


| | |
|---|---|
| 1. | Peter J. Ashenden, “Digital Design: An Embedded Sytems Approach UsingVERILOG”, Elesvier, 2010. |
| 2. | Ming-Bo Lin, “Digital System designs and Practices using Verilog Hdl and FPGAs”, John Wiley & Sons,2008 |
| 3. | Wayne Wolf, FPGA Based System Design, Prentices Hall Modern SemiconductorDesign Series, 2011. |
| 4. | Charles H Roth Jr, Lizy Kurian John and Byeong Kil Lee Digital Systems Designusing Verilog, Cengage Learning, First Edition, 2016. |


**Assessment Pattern :**

**CIE –** Continuous Internal Evaluation Lab (50 Marks)

**SEE –** Semester End Examination Lab (50 Marks)

**Note :**

For conduction, record book writing, viva, marks is kept, totalling to 30 marks & there will be 1CIE test in a semester conducted for 50 marks at the end of the semester & reduced to 20 marks,i.e., 30 record + 20 test = 50 marks.

| **Assessment Evaluation pattern for Standalone Laboratory** | | | | |
|---|---|---|---|---|
| **Conduction of Experiments** | | 30 (G) | Total CIE Lab Internal Marks = G + H = 30 + 20 = 50 | **Total out of 50**<br><br>**Marks** |
| Performance of the Experiment (On completion of every experiment /program in the laboratory, the students shall be evaluated and marks shall be awarded on the same day.<br>20 marks are for conducting the experiment and calculations / observations / output) | 20 | | | |
| Record | 05 | | | |
| Evaluation of outcome / Viva | 05 | | | |
| Final test / Case Study / Open Ended Experiment (if it is not test then a five-page report stapled has to be submitted) | 50 | Reducedto 20 (H) | | |
| CIE Marks (Lab) = 50 Marks = Final CIE Marks of the student | | | | |
| SEE Lab will be conducted for 50 marks | | | | |
| Total Lab Marks = CIE + SEE = 50 + 50 = 100 Marks | | | | |

| **Bloom's Category** | **Performance (Day To Day)** | **Internal Test conducted for 50 M & reduced to 20 M** | **SEE** |
|---|---|---|---|
| **Marks (Out of 50)** | **30** | **20** | **50** |
| Remember | 05 | | 05 |
| Understand | 05 | 04 | 05 |
| Apply | 05 | 04 | 10 |
| Analyze | 05 | 04 | 10 |
| Evaluate | 05 | 04 | 10 |
| Create | 05 | 04 | 10 |

| **Bloom's Category** | **Marks Practicals (100)** |
|---|---|
| Remember | 10 |
| Understand | 14 |
| Apply | 19 |
| Analyze | 19 |
| Evaluate | 19 |
| Create | 19 |

Experiment No:1 Date: _____

## Combinational circuits design

Experiment No:1a

## 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:**

```
module decoder_tb;
reg [1:0] I;
wire [3:0]D;
decoder uut (.D(D), .I(I));
initial begin
I=2'b00; #20;
```

```
I=2'b01; #20;
I=2'b10; #20;
I=2'b11; #20;
end
endmodule
```

**Results:**

Experiment No:1b 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: 1c 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: 1d 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: 2 Date:

# 4 BIT PARALLEL ADDER

**Aim:** Write a HDL program using generate for 4 bit parallel adder using FPGA implementation.

**Verilog**

```
module padd (a, b,cin,s,cout); input [3:0]a,b;
input cin; output [3:0]s; output cout; wire [3:1]c;
FAA FA0 (a[0],b[0], cin,s[0],c[1]);
FAA FA1 (a[1],b[1],c[1],s[1],c[2]);
FAA FA2 (a[2],b[2],c[2],s[2],c[3]);
FAA FA3 (a[3],b[3],c[3],s[3],cout);
endmodule
module FAA (a,b,ci,s,co); input a,b,ci;
output s,co;
assign s = a^b^ci;
assign co = (a&b)|(b&ci)|(ci&a); endmodule
```

## Test Bench:

padd_tb.v:

```
module padd_tb;
reg [3:0]A,B;
reg Cin; wire [3:0]S; wire Cout;
padd PA1 (A,B,Cin,S,Cout); initial begin
A=4'b0001; B=4'b0010; Cin=1'b1; #2 A=4'b1111; B=4'b1111; Cin=1'b1; #2 A=4'b1001; B=4'b1010;
Cin=1'b1; #2 A=4'b0001; B=4'b0010; Cin=1'b0; #2
$finish; end
Endmodule
```

## Results:

Experiment No: 3 Date: _

# 4 BIT CARRY LOOK AHEAD ADDER

**Aim:** Write HDL code for Carry look ahead adder using FPGA implementation

**Verilog**

```
module CLD(
input [3:0] A,
input [3:0] B,
input Cin,
output [3:0] Sum,
output Cout);
wire [3:0] P, G;
wire [3:0] Ci;
assign P[0] = A[0] ^ B[0];
assign G[0] = A[0] & B[0];
 assign P[1] = A[1] ^ B[1];
assign G[1] = A[1] & B[1];
assign P[2] = A[2] ^ B[2] ;
assign G[2] = A[2] & B[2] ;
assign P[3] = A[3] ^ B[3] ;
assign G[3] = A[3] & B[3] ;

assign Ci[0] = Cin;
assign Ci[1] = G[0] | (P[0] & Cin);
assign Ci[2] = G[1] | (P[1] & G[0])|(P[1] & P[0]& Cin);
assign Ci[3] = G[2] | (P[2] & G[1])|(P[2] & P[1]& G[0])|(P[2]& P[1]& P[0]& Cin);

assign Sum = A^B^Ci;
assign Cout = G[3] | (P[3] & G[2])|(P[3] & P[2]& G[1])|(P[3]& P[2]& P[1]& G[0])|(P[3] & P[2]&
P[1]& P[0]& Cin);
endmodule
```

**Test Bench:**

## Results:

Experiment No: 4 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: 5 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: 6 Date:

# 4x4 MULTIPLIER

**Aim:** Write HDL code and implement on FPGA for 4 x 4 Multiplier

## Verilog

```
module half_adder(input a, b, output s0, c0);
assign s0 = a ^ b;
assign c0 = a & b;
endmodule

module full_adder(input a, b, cin, output s0, c0);
assign s0 = a ^ b ^ cin;
assign c0 = (a & b) | (b & cin) | (a & cin);
endmodule

module array_multiplier(input [3:0] A, B, output [7:0] z);
wire signed p[3:0][3:0];
wire [10:0] c; // c represents carry of HA/FA
wire [5:0] s;  // s represents sum of HA/FA
// For ease and readability, two different name s and c are used instead of single wire name.
genvar g;

generate
  for(g = 0; g<4; g=g+1) begin
    and a0(p[g][0], A[g], B[0]);
    and a1(p[g][1], A[g], B[1]);
    and a2(p[g][2], A[g], B[2]);
    and a3(p[g][3], A[g], B[3]);
  end
  endgenerate
   assign z[0] = p[0][0];
```

```
  //row 0
  half_adder h0(p[0][1], p[1][0], z[1], c[0]);
  half_adder h1(p[1][1], p[2][0], s[0], c[1]);
  half_adder h2(p[2][1], p[3][0], s[1], c[2]);

  //row1
  full_adder f0(p[0][2], c[0], s[0], z[2], c[3]);
  full_adder f1(p[1][2], c[1], s[1], s[2], c[4]);
  full_adder f2(p[2][2], c[2], p[3][1], s[3], c[5]);

  //row2
  full_adder f3(p[0][3], c[3], s[2], z[3], c[6]);
  full_adder f4(p[1][3], c[4], s[3], s[4], c[7]);
  full_adder f5(p[2][3], c[5], p[3][2], s[5], c[8]);

  //row3
  half_adder h3(c[6], s[4], z[4], c[9]);
  full_adder f6(c[9], c[7], s[5], z[5], c[10]);
  full_adder f7(c[10], c[8], p[3][3], z[6], z[7]);
endmodule

module array_multiplier_tb;
  reg [3:0] A, B;
  wire [7:0] P;

  array_multiplier am(A,B,P);

  initial begin
    $monitor("A = %b: B = %b --> P = %b, P(dec) = %0d", A, B, P, P);
    A = 1; B = 0; #3;
    A = 7; B = 5; #3;
    A = 8; B = 9; #3;
```

```
    A = 4'hf; B = 4'hf;
     end
endmodule
```

## Test Bench:

## Results:

Experiment No: 7 Date: _

# EVEN PARITY

**Aim:** Write a Verilog code using task and function that will check the parity of a word for even parity and implement the same on FPGA

Verilog

```
module function_test(Z);
 output reg [4:0] Z;
 reg [3:0] INP;
 initial begin
 INP = 4'b0101;
 Z = parity(INP);
#100;
$finish;
 End

 function [4:0] parity;
 input [3:0] A;
 reg temp_parity;
 begin
 temp_parity = A[0] ^ A[1] ^ A[2] ^ A[3];
 parity = {A, temp_parity};
 end
endfunction

endmodule
```

## Test Bench:

## Results:

Experiment No: 8 Date: _

# LINEAR FEEDBACK SHIFT REGISTER(LFSR)

**Aim:** Write a Verilog code for 4 bit Linear feedback shift register(LFSR) using FPGA implementation.

Verilog:

```
module LFSR8_11D(
  input clk,
  output reg [7:0] LFSR = 255   // put here the initial value
);

wire feedback = LFSR[7];
always @(posedge clk)
begin
  LFSR[0] <= feedback;
  LFSR[1] <= LFSR[0];
  LFSR[2] <= LFSR[1] ^ feedback;
  LFSR[3] <= LFSR[2] ^ feedback;
  LFSR[4] <= LFSR[3] ^ feedback;
  LFSR[5] <= LFSR[4];
  LFSR[6] <= LFSR[5];
  LFSR[7] <= LFSR[6];
end
endmodule
```

## Test Bench:

## Results:

Experiment No: 9 Date: _

# SYNCHRONOUS 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:

## a) 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:**

# ASYNCHRONOUS COUNTERS

## 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:

## b) 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: 11 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:

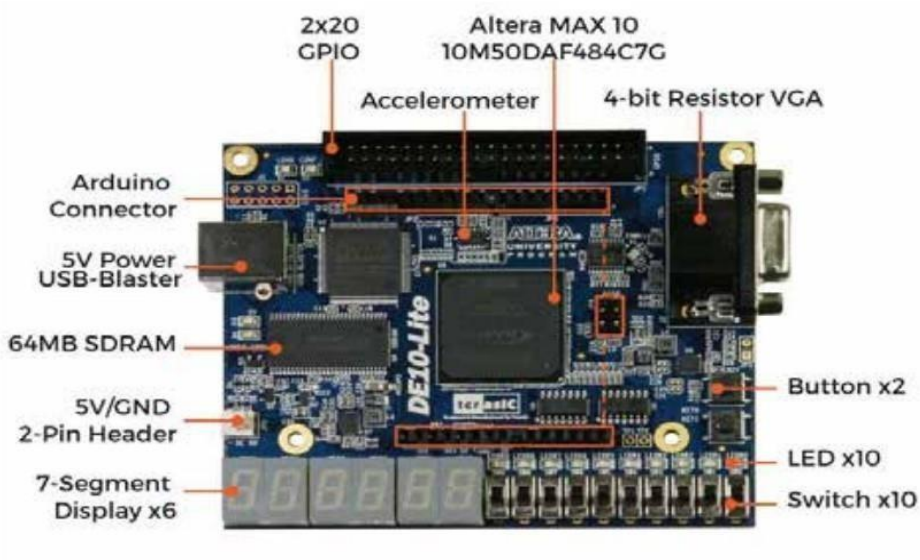
```
module seq_moore_tb;
        // Inputs
         reg x;
         reg clk;
         reg rst;
        // Outputs
         wire y;
        // Instantiate the Unit Under Test (UUT)
        seq_moore uut (
        .x(x),
        .clk(clk),
        .rst(rst),
        .y(y)
        );
```

```
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:

## 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, motorcontrol drive, analog to digital conversion, image processing, and handheld devices, the MAX 10Lite 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 ofthese 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"

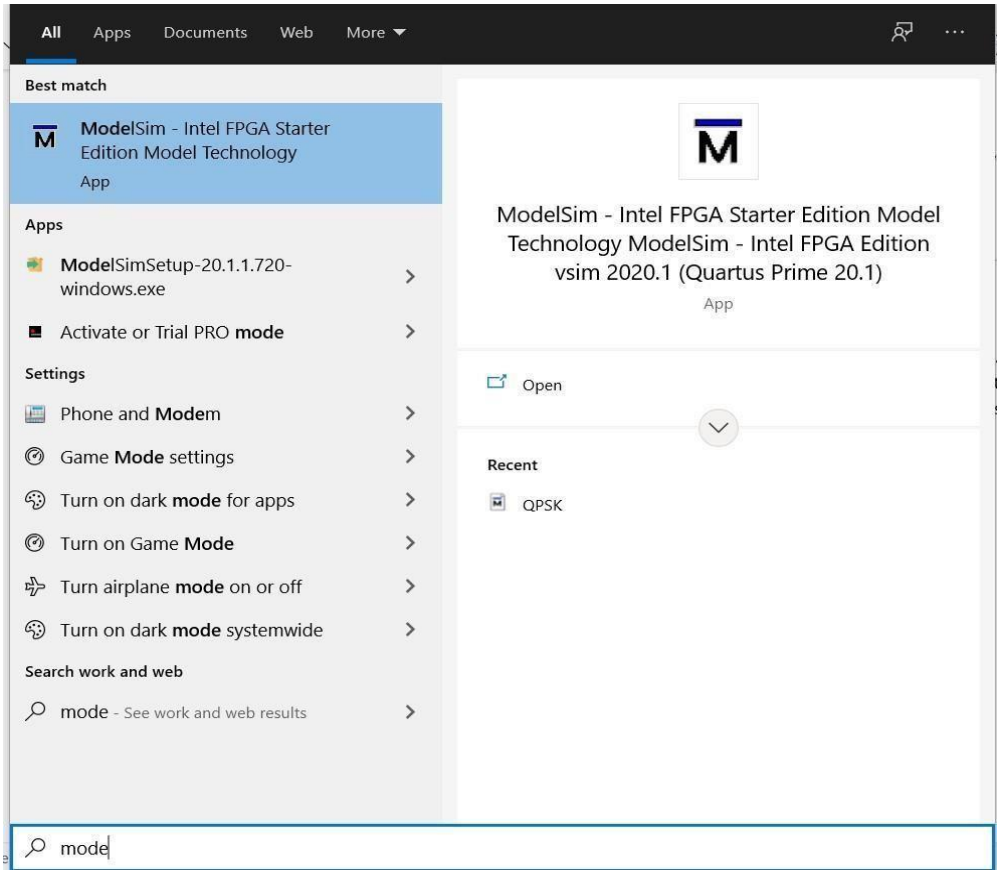
17) Click on "Compile Testbench"

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

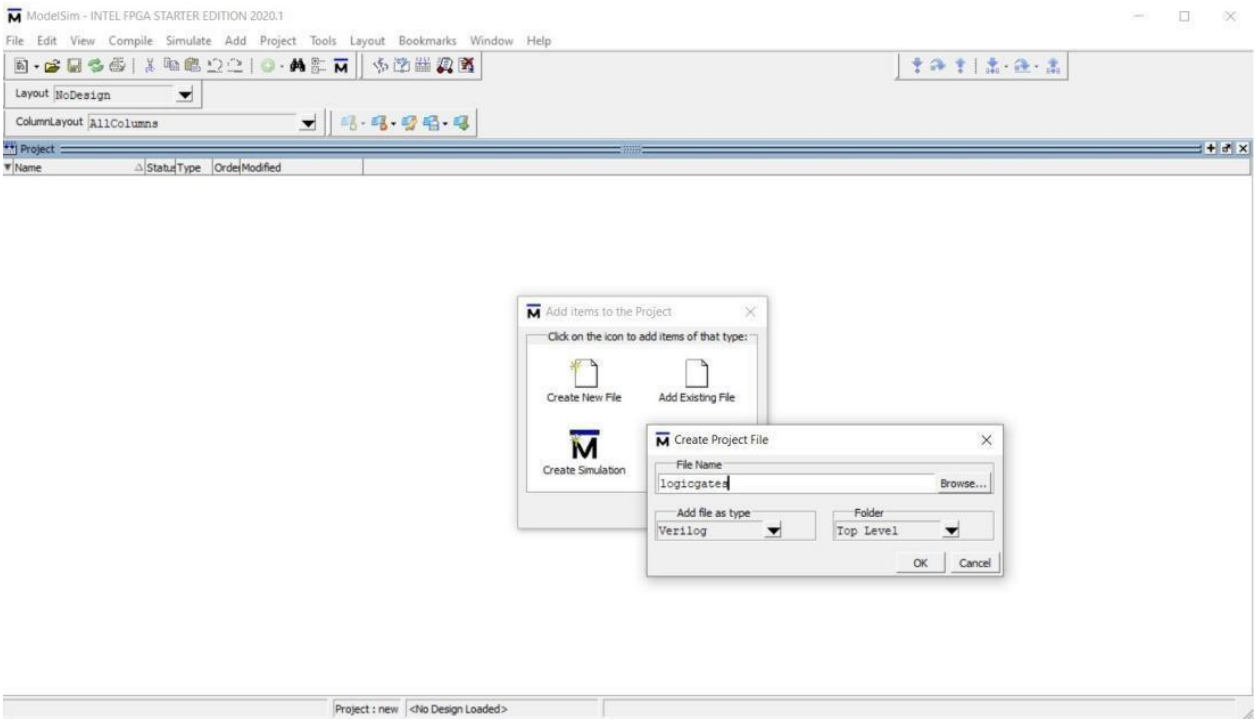



File>>new>> project

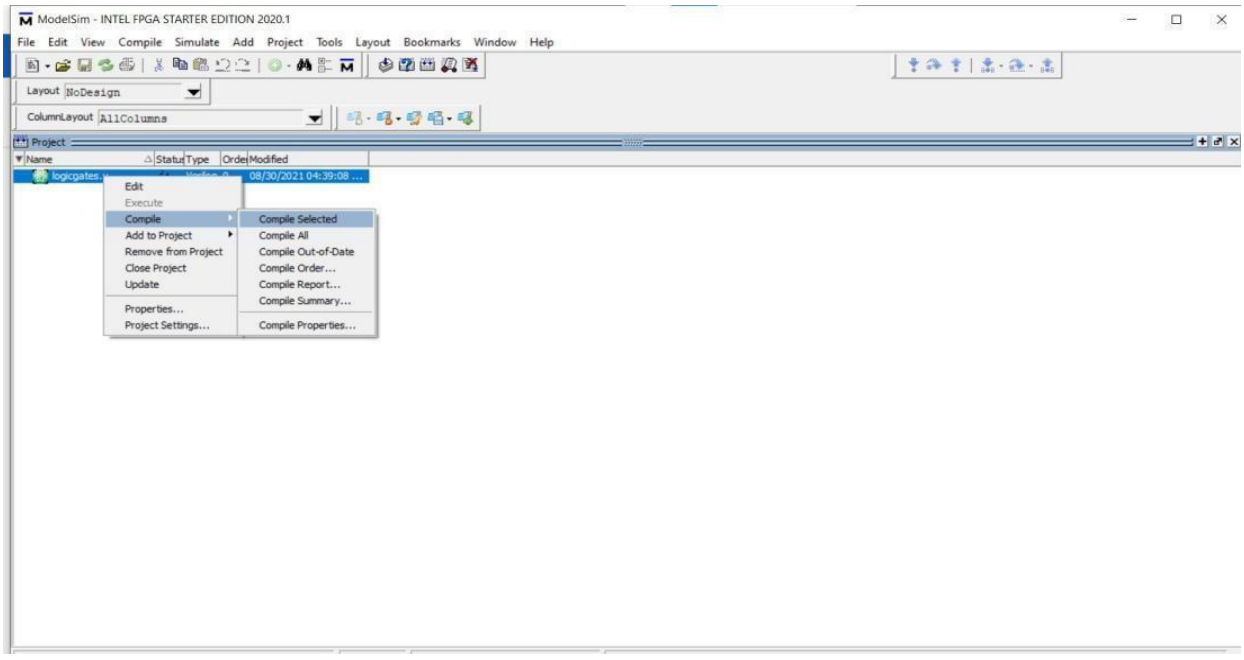
Give some project name and click ok.

Click Create New File

Select verilog and give file name



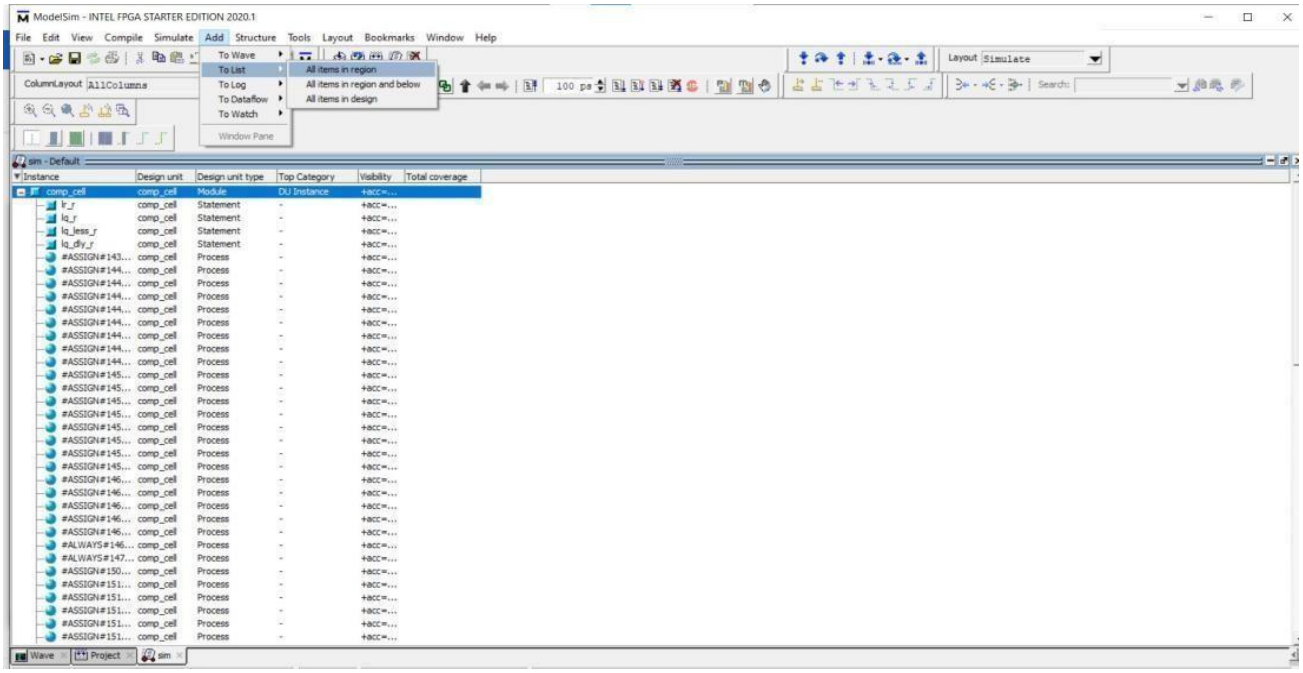

After writing code, compile the selected verilog file. You can also type a testbench file in the same file.




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

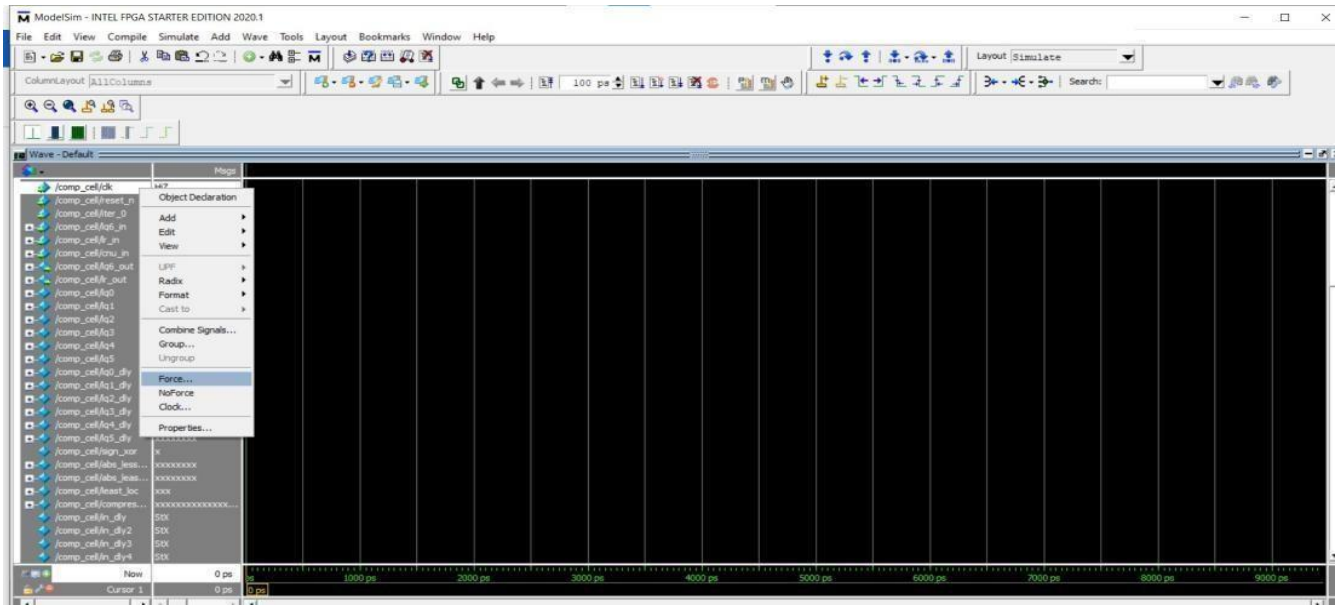
If you have a testbench select that, or else select your top module

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.

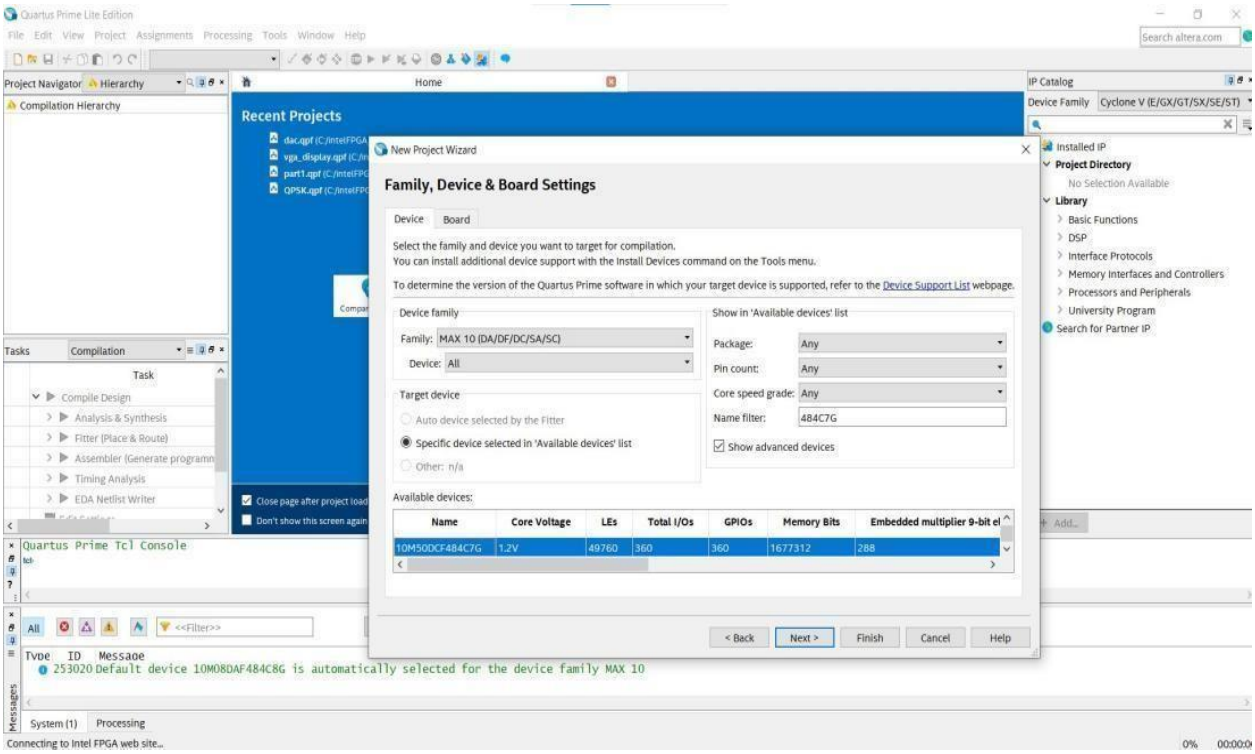



Run and view the waveforms and messages in the transcript

**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]**

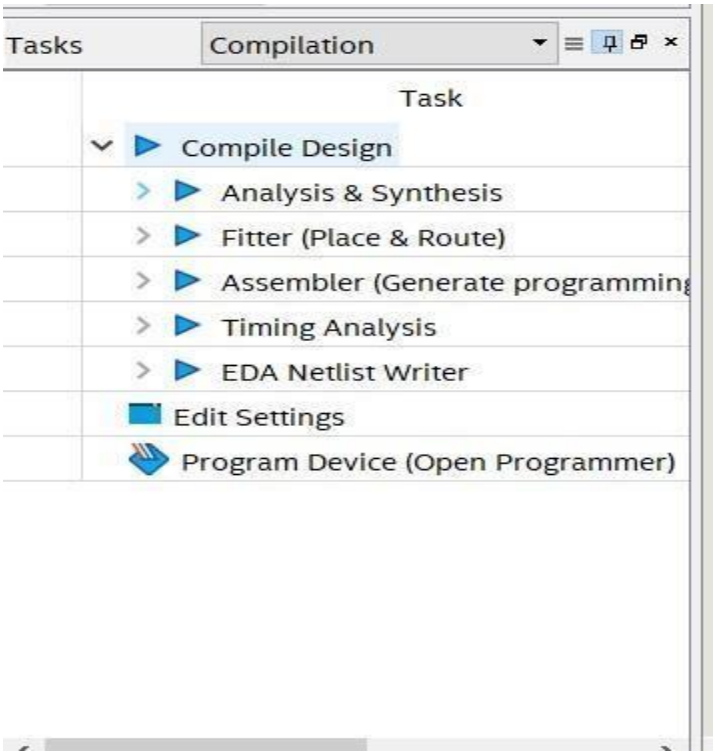
**set_location_assignment PIN_N2 -to blue[3]**

**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.

PROBABLE/SUGGESTED QUESTION BANK

1. What is Verilog?

2. What is the difference between (i) signal and variable (ii) generic & Parameter (iii) function & procedure (iv) task & function (v) always & initial (vi) register & variable (vii) signal & wire.

3. What are the operators in Verilog?

4. Which operator is having most priority?

5. What is meant by sensitivity list?

6. Give the Following syntax in HDL (i) if, for, function, procedure (ii) while, case.

7. What is the operating frequency of your FPGA? What are the advantages of using FPGA.

8. Expand FPGA & ASIC.

9. What are the data types in Verilog?

10. Write the truth table & Excitation table for D filp flop, SR , T, JK

11. What is meant by synthesis?

12. Write the flow chart for synthesis process?

13. What is the difference between combination circuit & sequential circuit?

14. What is the difference between latch & Flip flop?

15. In a pure combinational circuit is it necessary to mention all the inputs in sensitivity list? If yes, why?What is the difference between wire and reg?

16. Give only two xor gates one must function as buffer and another as not gate?

17. Build a 4:1 mux using only 2:1 mux?

18. What are the logical operators in Verilog?

19. What is the gate density of your FPGA?

20. What is data flow model, structural model, behavioral model?

21. What is the difference between SR flip flop & JK flip flop?

22. What is the difference between synchronous reset & Asynchronous reset?
23. What is mux and demux?
24. What is encoder and decoder?
25. What is the difference between encoder & priority encoder?
26. What are the different types of buffers are in Verilog HDL?
27. Write the syntax for casex and casez. What is screen time?
28. Draw the simulation waveform for D-latch using signal assignment and variable assignment statements inside the process.
29. What is test bench? Why we need test bench
30. What is task? Give example
31. What is Function? Give Example
32. What is Instantiation? Which are different types of Instantiation
33. What is elaboration?
34. What are the ways of generating the clock?
35. What is functional simulation?
36. What is sensitivity list?
37. What is RTL?
38. Why do we use clock division?
39. What is compilation?
40. What is the difference between FPGA and ASIC?
41. What is the flow of FPGA and ASIC?
42. List the different VLSI design styles?
43. Differentiate between function and tasks