**CS225- Lab 9**

**Design using HDL(Verilog)**

The goal of this lab10 is to familiarize the students with describing computer architectural blocks in verilog Hardware Description Language (HDL). Further we will simulate various hardware building blocks with appropriate stimulus. In this section we describe various some more logic circuits.

Combinational Logic

*Task1:The seven-segment light-emitting diode (LED) display depicted in Figure is a useful circuit in many applications using prototyping boards. Module Seven\_Seg\_Display accepts 4-bit words representing binary coded decimal (BCD) digits and displays their decimal value. The display has active-low illumination outputs, and can be implemented with combinational logic. The description synthesizes into a combinational circuit. Several of the input codes are unused and should not occur under ordinary operation. One possibility is to assign don't-cares to those codes. However, this would display an output if such an input code occurred. Instead, the default assignment blanks the display for all unused codes. This prevents a bogus display condition. If the default assignment is omitted, an event of an input that is not decoded will be detected by the event control expression of the cyclic behavior, but will not cause Display to be an assigned value. Simulate the following and study the behavior.*

**module Seven\_Seg\_Display (Display, BCD);**

**output [6: 0] Display;**

**input [3: 0] BCD;**

**reg [6: 0] Display;**

**// abc\_detg**

**parameter BLANK =7'b111\_1111;**

**parameter ZERO = 7'b000\_0001; // h01**

**parameter ONE = 7'b100\_1111; // h4f**

**parameter TWO = 7'b001\_0010; // h12**

**parameter THREE =7'b000\_0110; // h06**

**parameter FOUR = 7'b100\_1100; // h4c**

**parameter FIVE = 7'b010\_0100; // h24**

**parameter SIX = 7'b010\_0000; // h20**

**parameter SEVEN= 7'b000\_1111; // hOf**

**parameter EIGHT =7'b000\_0000; // hOO**

**parameter NINE = 7'b000\_0100; // h04**

**always @ (BCD)**

**case (BCD)**

**0: Display = ZERO;**

**1: Display = ONE;**

**2: Display = TWO;**

**3: Display = THREE;**

**4: Display = FOUR;**

**5: Display = FIVE;**

**6: Display = SIX;**

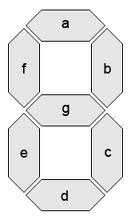
**7: Display = SEVEN;**

**8: Display = EIGHT;**

**9: Display = NINE; default:Display = BLANK;**

**endcase**

**endmodule**



*Task2:****Using a single continuous assignment. develop and verify a behavioral model***

***implementing a Boolean equation describing the logic of the circuit below. Use***

***the following names for the testbench, the model, and its ports: tb\_Combo\_CA(),***

***and Combo \_ CA (Y, A, B, C, D), respectively. Note: The test bench will have no***

***ports. Exhaustively simulate the circuit and provide graphical and text output***

***demonstrating that the model is correct.***

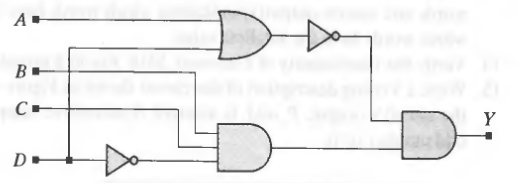
**module Combo\_CA (Y,A, B, C, D);**

**output Y;**

**input A, B, C, D;**

**assign Y = (~(A | D)) & (B & C & ~D);**

**endmodule**

****

Task 3: Decoders

Decoder is a combinational circuit that converts binary information from an n input lines to a maximum of 2n output lines. If the n-bit coded information has unused combinations, the decoder may have fewer than 2n outputs. Consider decoder\_2\_to\_4 below and verilog description.



module decoder\_2\_to\_4(d3, d2, d1, d0,s1, s0);

input s1, s0;

output d3, d2, d1, d0;

wire d3, d2, d1, d0; // wires for outputs

wire s1\_n, s0\_n; // interconnection wires

wire a3, a2, a1, a0; // interconnection wires

// Structural model of decoder

not n1 (s1\_n, s1),

not n2 (s0\_n, s0);

and a0 (d0,s1\_n,so\_n);

and a1 (d0,s1\_n,so);

and a2 (d0,s1,so\_n);

and a3 (d0,s1,so);

endmodule

Simulate the following decoder design

module decoder\_case (binary\_in,decoder\_out,enable);

input [3:0] binary\_in ; // 4 bit binary input

input enable ; // Enable for the decoder

output [15:0] decoder\_out ; // 16-bit out

reg [15:0] decoder\_out ;

always @ (enable or binary\_in)

begin

decoder\_out = 0;

if (enable) begin

case (binary\_in)

4'h0 : decoder\_out = 16'h0001;

4'h1 : decoder\_out = 16'h0002;

4'h2 : decoder\_out = 16'h0004;

4'h3 : decoder\_out = 16'h0008;

4'h4 : decoder\_out = 16'h0010;

4'h5 : decoder\_out = 16'h0020;

4'h6 : decoder\_out = 16'h0040;

4'h7 : decoder\_out = 16'h0080;

4'h8 : decoder\_out = 16'h0100;

4'h9 : decoder\_out = 16'h0200;

4'hA : decoder\_out = 16'h0400;

4'hB : decoder\_out = 16'h0800;

4'hC : decoder\_out = 16'h1000;

4'hD : decoder\_out = 16'h2000;

4'hE : decoder\_out = 16'h4000;

4'hF : decoder\_out = 16'h8000;

endcase

end

end

endmodule

**Task 4 : Clock generators**

**Clock generators** are used in testbenches to provide a clock signal for testing the model

of a synchronous circuit. A flexible clock generator will be parameterized for a variety

of applications. The forever loop causes unconditional repetitive execution of state-

ments, subject to the disable statement, and is a convenient construct for describing

clocks.

module test\_clock();

parameter half\_cycle = 50;

reg clock;

initial

begin: clock\_label // Note: clock\_loop is a named block of statements

forever

begin

#half\_cycle clock = 1;

#half\_cycle clock = 0;

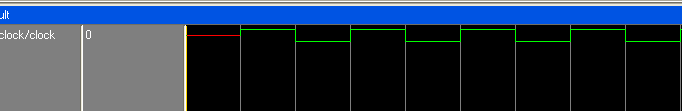
end

end

// always #100 clock = ~clock; another way of declaring clock

endmodule





**Task5: D Flipt flop**

module dff (q,d,clk,reset);

input d, clk, reset ;

output q;

reg q;

always @ ( posedge clk)

if (~reset)

q <= 1'b0;

else

q <= d;

endmodule



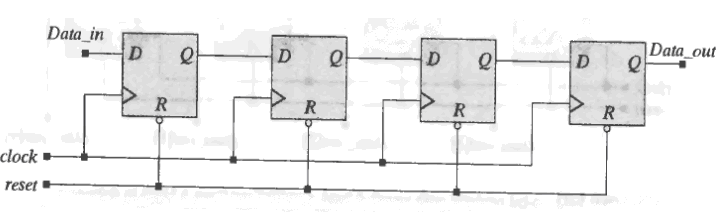
**Task 6: Shift register**

**Here,shift register declares an internal 4-bit register, Data\_reg, which creates Data\_out by a continuous assignment to the least significant bit (LSB) of the register and forms the register contents synchronously from a concatenation of the scalar Data\_in with the three leftmost bits of the register. Notice that the register variable,**

**Data\_reg, is referenced by concatenation in a nonblocking assignment before it is**

**assigned value in a synchronous behavior. This implies the need for memory, and**

**synthesizes to the flip-flop structure shown . Also, recall that the values on the RHS of the non blocking assignments are the values of the variables immediately before the active edge of the clock, and the values on the LHS are the values formed after the edge.**

****

module Shif\_reg4 (Data\_out, Data\_In, clock, reset);

output Data\_out;

input Data\_In, clock, reset;

reg [3: 0] Data\_reg;

assign Data\_out = Data\_reg[0];

always @ (posedge clock)

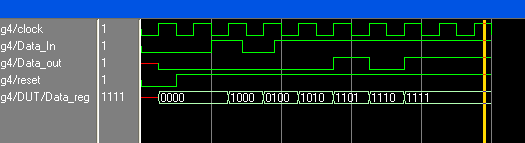
begin

if (reset == 1'b0) Data\_reg <= 4'b0;

else Data\_reg <= {Data\_In, Data\_reg[3:1]};

end

endmodule

****

**4-bit universal shift register**

A 4-bit universal shift register is an important unit of digital machines that employ a bit-slice architecture, with multiple identical slices of a 4-bit shift register chained together with additional logic to form a wider and more versatile datapath. Its features include synchronous reset, parallel inputs, parallel outputs, bidirectional serial input from either the LSB or the most significant bit (MSB), and bidirectional serial output to either the LSB or the MSB. In the serial-in, serial-out mode the machine can delay an input signal for 4 clock ticks, and act as a uni-directional shift register. In parallel-in, serial-out mode it operates as a parallel-to-serial converter, and in the serial-in, parallel-out mode it operates as a serial-to-parallel converter. Its parallel-in, parallel-out mode, combined with shift operations, allows it to perform any of the operations of less versatile unidirectional shift registers.

module counter (C\_OUT,CLK,reset);

output [3: 0] C\_OUT;

input CLK,reset;

reg [3:0] C\_OUT;

always @ (posedge CLK)

begin

if (reset) C\_OUT <= 4'b0000;

else

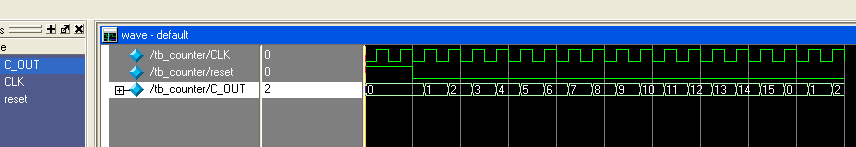
C\_OUT <= C\_OUT + 4'b0001;

end

endmodule

**Task 7: Counters**





Task: Write and verify the HDL beharioral description of a four-bit updown counter with parallel load using the following control inputs: (a) The counter has three control inputs for the three operations Load, Up, and Down. The order of precedence is Load, Up, and Down. (b) The counter has two selection inputs to specify four operations: Up, Down , Load, and no change.

module updown (OUT, Up, Down, Load, IN, CLK);

output [3: 0] OUT;

input [3: 0] IN;

input Up, Down, Load, CLK;

reg [3:0] OUT;

always @ (posedge CLK)

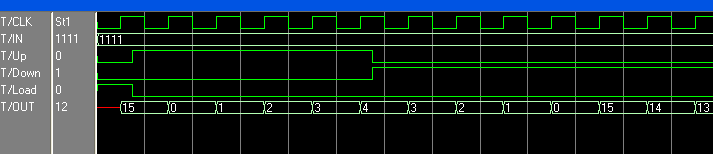
if (Load) OUT <= IN;

else if (Up) OUT <= OUT + 4'b0001;

else if (Down) OUT <= OUT - 4'b0001;

else OUT <= OUT;

endmodule



**Your Lab Assignment 10 (50 points)**

**Submission** Your submission must contain:

The source code of your design/testbench if any (for each of the problem) and answer document (no page limit). Code should reasonably well commented. Please submit document as a pdf (single) with file name rollNo\_Lab9.pdf.

Lab submission through [cs225.iitp@gmail.com](mailto:cs225.iitp@gmail.com) with subject: YourrollNo\_Lab9

Q1.

Design and test a verilog model for 4 bit Binary to Gray code Converter.

**(5 points)**

Q2:

Design and verify an 16 bit ALU which performs the following operation: Add, sub, xor, and, or, increment, left shift, right shift. The ALU should indicate a zero flag if the result operation is zero.

**(10 points)**

Q3: Design 16 to 1 Multiplexer and test using appropriate test bench.

Q4: Model 16 to 1 Multiplexer using 2-1 multiplers and test using appropriate test bench.

**(10 points)**

Q5: Model 6 to 64 decoder and test using appropriate test bench.

: **(5 points)**

Q6: Model 6 to 64 decoder using 3 to 8 decoders and test using appropriate test bench.

: **(10 points)**

Q7: Simulate the following models by writing appropriate test bench.

module decade\_counter ( output reg [3:0] q,

input clk );

always @(posedge clk)

q <= q == 9 ? 0 : q + 1;

endmodule

module decoded\_counter ( output ctrl,

input clk );

reg [3:0] count\_value;

always @(posedge clk)

count\_value <= count\_value + 1;

assign ctrl = count\_value == 4'b0111 ||

count\_value == 4'b1011;

endmodule

**(10 points)**