All instructions in the tutorial file are completed and snapshot was taken as followed:

Varilog implimentation:

A screenshot of a computer

Description automatically generated

Pins assigned:

A screenshot of a computer

Description automatically generated

Error free compilation:A screenshot of a social media post

Description automatically generated

Summary:A screenshot of a social media post

Description automatically generated

Hardware programer:A screenshot of a computer

Description automatically generated

Function Simulation:A screenshot of a social media post

Description automatically generated

1. The **reg** stands for register. It represents data storage elements in Verilog. They retain their value till next value is assigned to them.

The **wire** is used for connecting different elements. They can be treated as physical wires. They can be treated as physical wires. No values get stored in them.

1. **Wire** can only be used on the left side of a continuous assignment and often declared as an input.
2. Rules for **Inputs**: internally must always be of type net, externally the inputs can be connected to a variable of type reg or net.

Rules for **Output**: internally can be of type reg or net. Externally can be of datatype or net type net.

Rules for **Inouts**: internally or externally must always be type net.

1. **Continuous Assignment:** continuous assignment drives a value into a net. It is declared outside of procedural blocks.

**Blocking Assignment:** executed before the execution of the statements that follow it in a sequential block. The operator is “=”

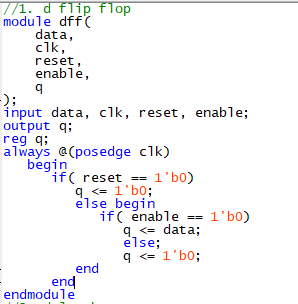
**Nonblocking Assignment:** it allows us to assign values without blocking the procedural flow. The operator is “<=”.

1. **Combinational logic** is time independent, and logic does not depend on the previous inputs.

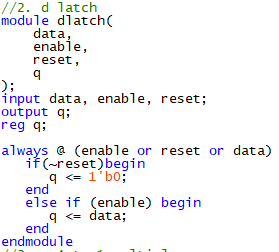
**Sequential logic** is dependent on the clock cycles and the output depends on present as well as past inputs.

1. Include all the branches of an if or case statement
2. Assign a value to every output signal in every branch.
3. Use default assignment at the start of the procedure, so every signal will be assigned.
4. << is binary logical shift, and <<< is arithmetic left shift.
5. wire[6:0] x[5:0];

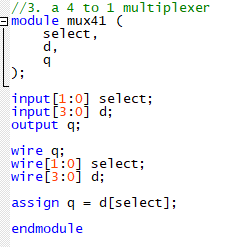
//1. d flip flop



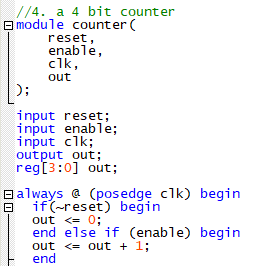
//2. d latch



//3. a 4 to 1 multiplexer



//4. a 4 bit counter





module seven seg\_decoder(input [3:0] x, output [6:0] hex\_LEDs);

reg [6:0] reg\_LEDs;

assign hex\_LEDs[0]=(~x[0]&x[1]&~x[2]&~x[3])|

(~x[0]&~x[1]&~x[2]&x[3])|(x[0]&x[1]&x[3])|

(x[0]&x[1]&x[2]) |(x[0]&x[2]&x[3]);

assign hex\_LEDs[1]=(x[1]&~x[2]&x[3]) |(x[0]&x[2]&x[3]) |(~x[0]&x[1]&x[2]&~x[3]);

assign hex\_LEDs[6:2]=reg\_LEDs[6:2];

always @(\*)

begin

case (x)

4'b0000: reg\_LEDs[6:2]=7'b10000; //7'b1000000 decimal 0

4'b0001: reg\_LEDs[6:2]=7'b11110; //7'b1111001 decimal 1

4'b0010: reg\_LEDs[6:2]=7'b01001; //7'b0100100 decimal 2

4'b0011: reg\_LEDs[6:2]=7'b01100; //7'b0110000 decimal 3

4'b0100: reg\_LEDs[6:2]=7'b00110; //7'b0011001 decimal 4

4'b0101: reg\_LEDs[6:2]=7'b00100; //7'b0010010 decimal 5

4'b0110: reg\_LEDs[6:2]=7'b00000; //7'b0000010 decimal 6

4'b0111: reg\_LEDs[6:2]=7'b11110; //7'b1111000 decimal 7

4'b1000: reg\_LEDs[6:2]=7'b00000; //7'b0000000 decimal 8

4'b1001: reg\_LEDs[6:2]=7'b00100; //7'b0010000 decimal 9

4'b1010: reg\_LEDs[6:2]=7'b00110; //7'b0011000 decimal q

4'b1011: reg\_LEDs[6:2]=7'b11000; //7'b1100011 decimal u

4'b1100: reg\_LEDs[6:2]=7'b00010; //7'b0001000 decimal a

4'b1101: reg\_LEDs[6:2]=7'b01010; //7'b0101011 decimal n

4'b1110: reg\_LEDs[6:2]=7'b00100; //7'b0010001 decimal y

4'b1111: reg\_LEDs[6:2]=7'b11111; //7'b1111111 decimal OFF

endcase

end

endmodule

module lab1part2(input [3:0] x, output [6:0] hex\_LEDs);

reg [6:0] reg\_LEDs;

assign hex\_LEDs[0]=(~x[0]&x[1]&~x[2]&~x[3])|

(~x[0]&~x[1]&~x[2]&x[3])|(x[0]&x[1]&x[3])|

(x[0]&x[1]&x[2]) |(x[0]&x[2]&x[3]);

assign hex\_LEDs[1]=(x[1]&~x[2]&x[3]) |(x[0]&x[2]&x[3]) |(~x[0]&x[1]&x[2]&~x[3]);

assign hex\_LEDs[6:2]=reg\_LEDs[6:2];

always @(\*)

begin

case (x)

4'b0000: reg\_LEDs[6:0]=7'b1000000; //7'b1000000 decimal 0

4'b0001: reg\_LEDs[6:0]=7'b1111001; //7'b1111001 decimal 1

4'b0010: reg\_LEDs[6:0]=7'b0100100; //7'b0100100 decimal 2

4'b0011: reg\_LEDs[6:0]=7'b0110000; //7'b0110000 decimal 3

4'b0100: reg\_LEDs[6:0]=7'b0011001; //7'b0011001 decimal 4

4'b0101: reg\_LEDs[6:0]=7'b0010010; //7'b0010010 decimal 5

4'b0110: reg\_LEDs[6:0]=7'b0000010; //7'b0000010 decimal 6

4'b0111: reg\_LEDs[6:0]=7'b1111000; //7'b1111000 decimal 7

4'b1000: reg\_LEDs[6:0]=7'b0000000; //7'b0000000 decimal 8

4'b1001: reg\_LEDs[6:0]=7'b0010000; //7'b0010000 decimal 9

4'b1010: reg\_LEDs[6:0]=7'b0011000; //7'b0011000 decimal q

4'b1011: reg\_LEDs[6:0]=7'b1100011; //7'b1100011 decimal u

4'b1100: reg\_LEDs[6:0]=7'b0001000; //7'b0001000 decimal a

4'b1101: reg\_LEDs[6:0]=7'b0101011; //7'b0101011 decimal n

4'b1110: reg\_LEDs[6:0]=7'b0010001; //7'b0010001 decimal y

4'b1111: reg\_LEDs[6:0]=7'b1111111; //7'b1111111 decimal OFF

endcase

end

endmodule