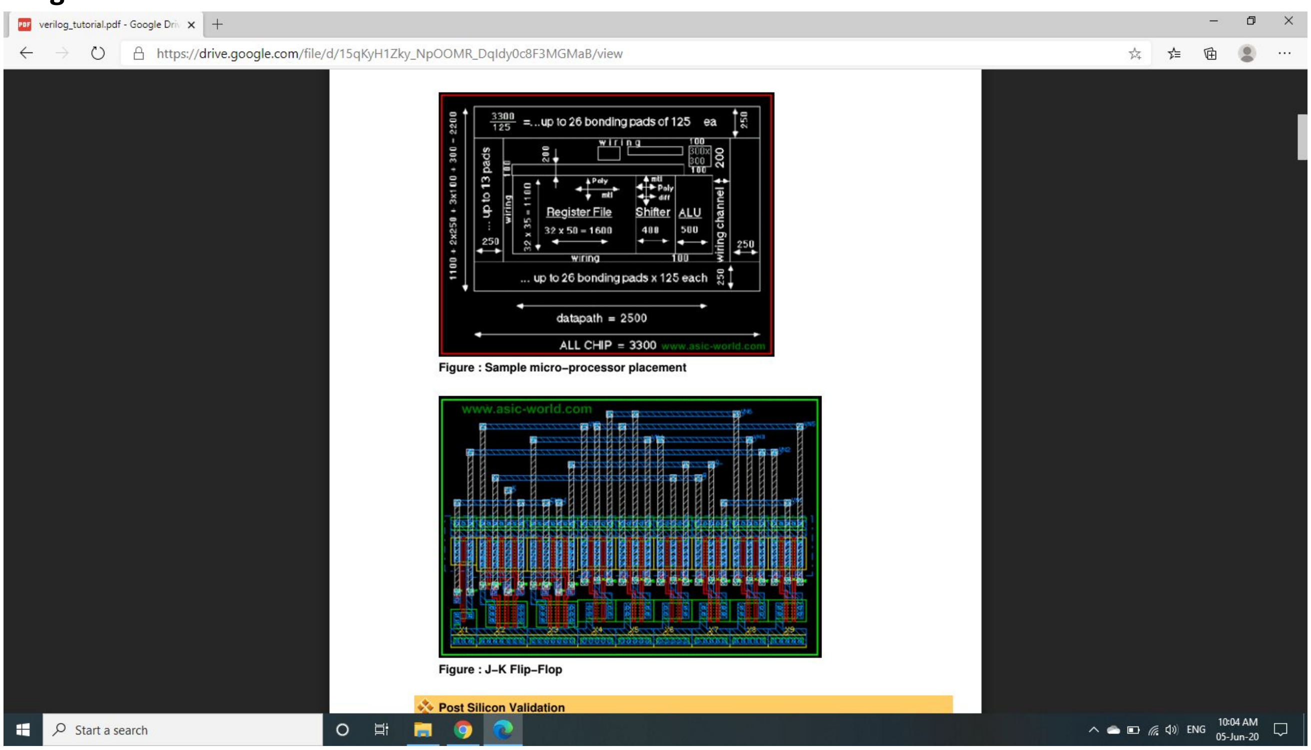
REPORT JUNE 05

Date:	05 JUNE 2020	Name:	Rakshith B
Course:	Digital Design Using HDL	USN:	4AL16EC409
Topic:	Verilog Tutorials and practice programs,Building/ Demo projects using FPGA , Implement a verilog module to count number of 0's in a 16 bit number in the compiler.	Semester & Section:	6th SEM B
Github Repository:	Rakshith-B		

FORENOON SESSION DETAILS

Image of session



Report –

What is HDL ?

A hardware description Language Is a language used to describe a digital system, for example, a network switch, a microprocessor or a memory or a simple flip−flop. This just means that, by using a HDL one can describe any hardware (digital) at any level.

One can describe a simple Flip flop as that in above figure as well as one can describe a complicated designs having 1 million gates. Verilog is one of the HDL languages available in the industry for designing the Hardware. Verilog allows us to design a Digital design at Behavior Level,

Register Transfer Level (RTL), Gate level and at switch level. Verilog allows hardware designers to express their designs with behavioral constructs, deterring the details of implementation to a later stage of design in the final design.

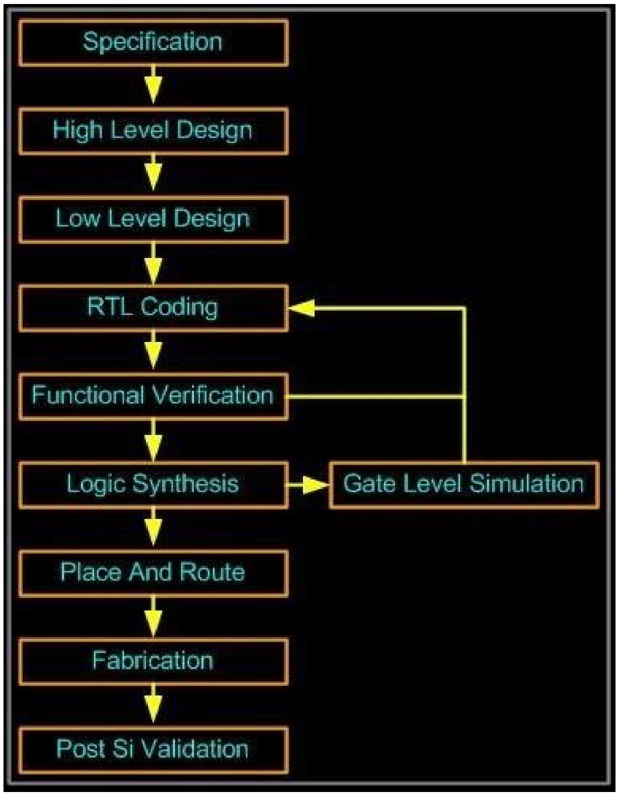
Design Styles:

- **Top Up Design**
- **Bottom Up Design**

Abstract Level of Verilog

- **Behavioral Level**
  This level describes a system by concurrent algorithms (Behavioral). Each algorithm itself is sequential, that means it consists of a set of instructions that are executed one after the other.Functions, Tasks and Always blocks are the main elements. There is no regard to the structural realization of the design.
- **Register Transfer Level**
  Designs using the Register−Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers. An explicit clock is used. RTL design contains exact timing possibilities, operations are scheduled to occur at certain times. Modern definition of a RTL code is "Any code that is synthesizable is called RTL code".
- **Gate Level**
  Within the logic level the characteristics of a system are described by logical links and their timing properties. All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc gates). Using gate level modeling might not be a good idea for any level of logic design. Gate level code is generated by tools like synthesis tools and this netlist is used for gate level simulation and for backend.

Typical Design Flow



Specification:
This is the stage at which we define what are the important parameters of the system/design that you are planning to design. Simple example would be, like I want to design a counter, it should be 4 bit wide, should have synchronous reset, with active high enable, When reset is active, counter output should go to "0". You can use Microsoft Word, or GNU Abiword or Openoffice for entering the specification.

High Level Design
This is the stage at which you define various blocks in the design and how they communicate. Lets assume that we need to design a microprocessor, High level design means splitting the design into blocks based on their function, In our case various blocks are registers, ALU, Instruction Decode, Memory Interface, etc. You can use Microsoft Word, or KWriter or Abiword or Openoffice for entering high level design.

Micro Design /Low Level Design
Low level design or Micro design is the phase in which, designer describes how each block is implemented. It contains details of State machines, counters, Mux, decoders, internal registers. For state machine entry you can use either Word, or special tools like StateCAD. It is always a good idea if waveform is drawn at various interfaces. This is the phase, where one spends a lot of time.

RTL Coding
In RTL coding, Micro Design is converted into Verilog/VHDL code, using synthesizable constructs of the language. Normally we use vim editor, but I prefer conTEXT and Nedit editor, it all depends

on which editor you like. Some use Emacs.

Simulation

Simulation is the process of verifying the functional characteristics of models at any level of abstraction. We use simulators to simulate the Hardware models. To test if the RTL code meets the functional requirements of the specification, see if all the RTL blocks are functionally correct. To achieve this we need to write a testbench, which generates clk, reset and required test vectors. A sample testbench for a counter is as shown below. Normally we spend 60−70% of time in verification of design.

Synthesis

Synthesis is a process in which synthesis tools like a design compiler or Synplify takes the RTL in Verilog or VHDL, target technology, and constraints as input and maps the RTL to target technology primitives. Synthesis tool after mapping the RTL to gates, also does the minimal amount of timing analysis to see if the mapped design meets the timing requirements. (Important thing to note is, synthesis tools are not aware of wire delays, they know only gate delays). After the synthesis there are couple of things that are normally done before passing the netlist to backend (Place and Route)

Place & Route

Gate Level netlist from the synthesis tool is taken and imported into place and route tool in Verilog netlist format. All the gates and flip−flops are places, Clock tree synthesis and reset is routed. After this each block is routed. Output of the P&R tool is GDS file, this file is used by foundry for fabricating the ASIC. Normally the P&R tool are used to output the SDF file, which is back annotated along with the gate level netlist from P&R into static analysis tool like Prime Time to do timing analysis.

```
module delay_example();

wire out1,out2,out3,out4,out5,out6;
reg b,c;

// Delay for all transitions
or #5 u_or (out1,b,c);
// Rise and fall delay
and #(1,2) u_and (out2,b,c);
// Rise, fall and turn off delay
nor #(1,2,3) u_nor (out3,b,c);
//One Delay, min, typ and max
nand #(1:2:3) u_nand (out4,b,c);
//Two delays, min,typ and max
buf #(1:4:8,4:5:6) u_buf (out5,b);
//Three delays, min, typ, and max
notif1 #(1:2:3,4:5:6,7:8:9) u_notif1 (out6,b,c);

//Testbench code
initial begin
  $monitor ( "Time = %g b = %b c=%b out1=%b out2=%b out3=%b out4=%b
out5=%b out6=%b" , $time, b, c ,
```

```
out1, out2, out3, out4, out5, out6);
 b = 0;
 c = 0;
 #10 b = 1;
 #10 c = 1;
 #10 b = 0;
 #10 $finish;
end

endmodule
```

Verilog Operators

```
module arithmetic_operators();

initial begin
 $display ( " 5 + 10 = %d" , 5 + 10);
 $display ( " 5 - 10 = %d" , 5 - 10);
 $display ( " 10 - 5 = %d" , 10 - 5);
 $display ( " 10 * 5 = %d" , 10 * 5);
 $display ( " 10 / 5 = %d" , 10 / 5);
 $display ( " 10 / -5 = %d" , 10 / -5);
 $display ( " 10 %s 3 = %d" ,
 $display ( " +5 = %d" , +5);
 $display ( " -5 = %d" , -5);
 #10 $finish;
end

endmodule
```

Logical Operator

```
module logical_operators();

initial begin
 // Logical AND
 $display ( "1'b1 && 1'b1 = %b" , (1'b1 && 1'b1));
 $display ( "1'b1 && 1'b0 = %b" , (1'b1 && 1'b0));
 $display ( "1'b1 && 1'bx = %b" , (1'b1 && 1'bx));
 // Logical OR
 $display ( "1'b1 || 1'b0 = %b" , (1'b1 || 1'b0));
 $display ( "1'b0 || 1'b0 = %b" , (1'b0 || 1'b0));
 $display ( "1'b0 || 1'bx = %b" , (1'b0 || 1'bx));
 // Logical Negation
 $display ( "! 1'b1 = %b" , (! 1'b1));
 $display ( "! 1'b0 = %b" , (! 1'b0));
 #10 $finish;
end

endmodule
```

Bit Wise Operator

```
module bitwise_operators();

initial begin
 // Bit Wise Negation
 $display ( " ~4'b0001 = %b" , (~4'b0001));
 $display ( " ~4'bx001 = %b" , (~4'bx001));
 $display ( " ~4'bz001 = %b" , (~4'bz001));
 // Bit Wise AND
 $display ( " 4'b0001 & 4'b1001 = %b" , (4'b0001 & 4'b1001));
 $display ( " 4'b1001 & 4'bx001 = %b" , (4'b1001 & 4'bx001));
 $display ( " 4'b1001 & 4'bz001 = %b" , (4'b1001 & 4'bz001));
 // Bit Wise OR
 $display ( " 4'b0001 | 4'b1001 = %b" , (4'b0001 | 4'b1001));
 $display ( " 4'b0001 | 4'bx001 = %b" , (4'b0001 | 4'bx001));
 $display ( " 4'b0001 | 4'bz001 = %b" , (4'b0001 | 4'bz001));
 // Bit Wise XOR
 $display ( " 4'b0001 ^ 4'b1001 = %b" , (4'b0001 ^ 4'b1001));
 $display ( " 4'b0001 ^ 4'bx001 = %b" , (4'b0001 ^ 4'bx001));
 $display ( " 4'b0001 ^ 4'bz001 = %b" , (4'b0001 ^ 4'bz001));
 // Bit Wise XNOR
 $display ( " 4'b0001 ~^ 4'b1001 = %b" , (4'b0001 ~^ 4'b1001));
 $display ( " 4'b0001 ~^ 4'bx001 = %b" , (4'b0001 ~^ 4'bx001));
 $display ( " 4'b0001 ~^ 4'bz001 = %b" , (4'b0001 ~^ 4'bz001));
 #10 $finish;
end

endmodule
```

Behavioral Modeling

```
module avoid_latch_else ();


reg q;
reg enable, d;


always @ (enable or d)
if (enable) begin
 q = d;
end else begin
 q = 0;
end
```

```
initial begin
 $monitor ( " ENABLE = %b D = %b Q = %b" ,enable,d,q);
 #1 enable = 0;
 #1 d = 0;
 #1 enable = 1;
 #1 d = 1;
 #1 d = 0;
 #1 d = 1;
 #1 d = 0;
 #1 d = 1;
 #1 enable = 0;
 #1 $finish;
end
endmodule
```

Task and Function

```
module task_calling (temp_a, temp_b, temp_c, temp_d);
input [7:0] temp_a, temp_c;
output [7:0] temp_b, temp_d;
reg [7:0] temp_b, temp_d;
`include "mytask.v"
always @ (temp_a)
begin
 convert (temp_a, temp_b);
end
always @ (temp_c)
begin
 convert (temp_c, temp_d);
end
endmodule
```

Test Bench code

```
module counter_tb;
reg clk, reset, enable;
wire [3:0] count;
```

```
counter U0 (
clk (clk),
reset (reset),
enable (enable),
count (count)
);
initial
begin
 clk = 0;
 reset = 0;
 enable = 0;
end
always
#5 clk = !clk;
endmodule
```

Implement a verilog module to count number of 0's in a 16 bit number in compiler.

```
module num_zeros_for(
    input [15:0] A,
    output reg [4:0] ones
    );

integer i;

always@(A)
begin
    ones = 0;
    for(i=0;i<16;i=i+1)
        if(A[i] == 0'b1)
            ones = ones + 1;
end

endmodule
```

output

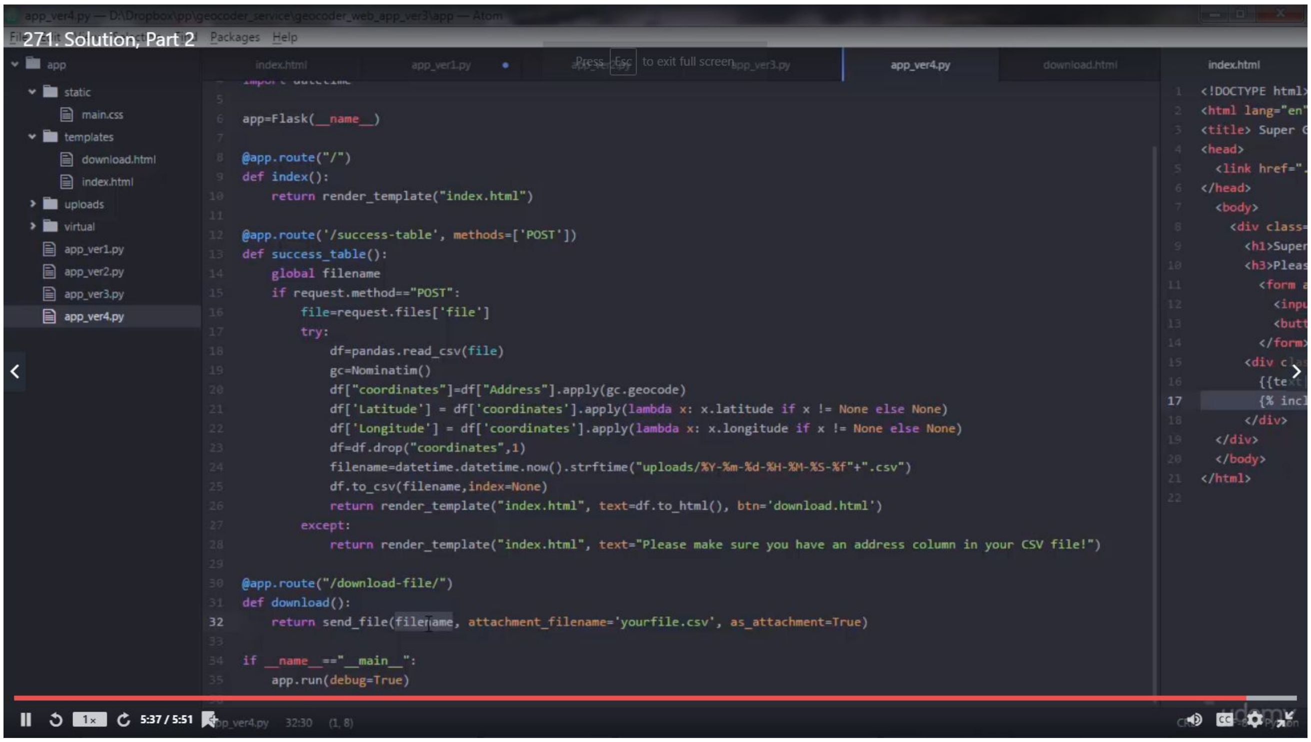
Input = "1010_0010_1011_0010" => Output = "01001" (9 in decimal)

Input = "0011_0110_1000_1011" => Output = "01000" (8 in decimal)

Date:	**05 JUNE 2020**	**Name:RAKSHITH B**	
Course:	**Python On Udemy**	**USN:4AL16EC409**	
Topic:	**Geocoder**	**Semester & Section:6 B**	

AFTERNOON SESSION DETAILS

Image of session:Output



app.py

```
from flask import Flask, render_template, request, send_file
from geopy.geocoders import ArcGIS
import pandas
import datetime


app=Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

@app.route('/success-table', methods=['POST'])
def success_table():
    global filename
    if request.method=="POST":
        file=request.files['file']
        try:
            df=pandas.read_csv(file)
            gc=ArcGIS(scheme='http')
            df["coordinates"]=df["Address"].apply(gc.geocode)
            df['Latitude'] = df['coordinates'].apply(lambda x: x.latitude if
x != None else None)
            df['Longitude'] = df['coordinates'].apply(lambda x: x.longitude
if x != None else None)
            df=df.drop("coordinates",1)

filename=datetime.datetime.now().strftime("sample_files/%Y-%m-%d-%H-%M-%S-%f"
+".csv")
            df.to_csv(filename,index=None)
            return render_template("index.html", text=df.to_html(),
btn='download.html')
        except Exception as e:
            return render_template("index.html", text=str(e))

@app.route("/download-file/")
def download():
```

```
    return send_file(filename, attachment_filename='yourfile.csv',
as_attachment=True)

if __name__=="__main__":
    app.run(debug=True)
```

download.html

```
<!DOCTYPE html>
<html lang="en">
<div class="download">
<a href={{url_for('download')}} target="blank"> <button class="btn">
Download </button></a>
</div>
</html>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<title> Super Geocoder </title>
<head>
  <link href="../static/main.css" rel="stylesheet">
</head>
  <body>
    <div class="container">
      <h1>Super Geocoder</h1>
      <h3>Please upload your CSV file. The values containing addresses should
be in a column named <em>address</em> or <em>Address</em></h3>
        <form action="{{url_for('success_table')}}" method="POST"
enctype="multipart/form-data">
          <input type="file" accept=".csv" name="file" />
          <button type="submit"> Submit </button>
        </form>
      <div class="output">
        {{text|safe}}
        {% include btn ignore missing %}
      </div>
  </div>
  </body>
```

```html
</html>
```