Date:	**05-06-2020**	**Name:**	**Kishan shetty**
Course:	**Digital Design Using HDL**	**USN:**	**4AL17EC041**
Topic:	**Verilog Tutorials and practice programs,Building/ Demo projects using FPGA**	**Semester & Section:**	**6th sem & Asec**
Github Repository:	**Kishanshetty-041**		

FORENOON SESSION DETAILS

REPORT

N-bit Adder Design in Verilog

The N-bit Adder is simply implemented by connecting 1 Half Adder and N-1 Full Adder in series. The Verilog code for N-bit Adder is designed so that the N value can be initialized independently for each instantiation. To do it, the Verilog code for N-bit Adder uses Generate Statement in Verilog to create a chain of full adders for implementing the N-bit Adder.

Verilog code

```
// Verilog project: Verilog code for N-bit Adder

 // Top Level Verilog code for N-bit Adder using Structural Modeling

module N_bit_adder(input1,input2,answer);

parameter N=32;

input [N-1:0] input1,input2;

output [N-1:0] answer; wire carry_out;

wire [N-1:0] carry;

 genvar i;

generate for(i=0;i ;i<N;i=i+1)

 begin: generate_N_bit_Adder
```

```
 if(i==0)
 half_adder f(input1[0],input2[0],answer[0],carry[0]);
 else
 full_adder f(input1[i],input2[i],carry[i-1],answer[i],carry[i]);
 end
 assign carry_out = carry[N-1];
 endgenerate
endmodule




// Verilog project: Verilog code for N-bit Adder
// Verilog code for half adder
module half_adder(x,y,s,c);
 input x,y;
output s,c;
assign s=x^y;
assign c=x&y;
endmodule
 // half adder



// Verilog project: Verilog code for N-bit Adder
 // Verilog code for full adder
module full_adder(x,y,c_in,s,c_out);
 input x,y,c_in; output s,c_out;
assign s = (x^y) ^ c_in;
 assign c_out = (y&c_in)| (x&y) | (x&c_in);
 endmodule
// full_adder
```

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.

Task (DAY - 5)

Implement a Verilog module to count number of 1's and 0's in a 16-bit number in compiler.

Verilog Code:

```
module num_zero_ones ( input [15:0] In,
 output reg [4:0] ones,
 output reg [4:0] zeros);
integer i, o, z;
always @ (In)
begin
```

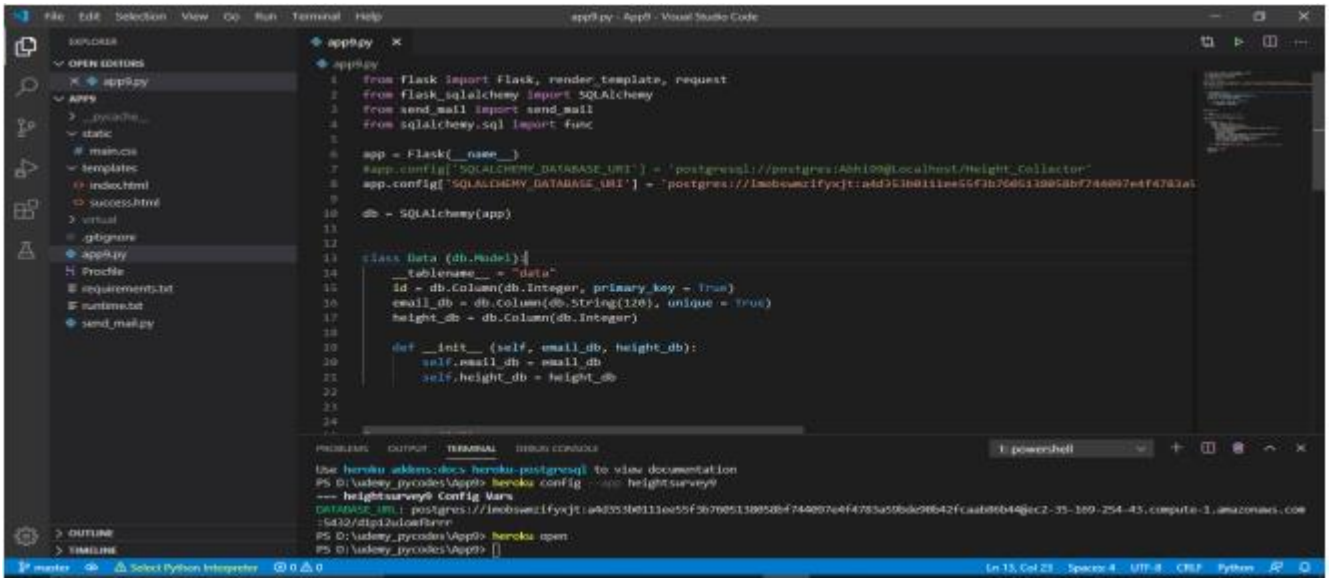
```
    o = 0; //initialize count variable.

    z = 0; //initialize count variable.

    for(i=0;i<16;i=i+1) //check for all the bits.

        if(In[i] == 1'b1) //check if the bit is '1'

            o = o + 1; //if its one, increment the count.

   z = 16-o; //number of zeros.

   ones = o;

   zeros = z;

end

endmodule
```

Date: **05-06- 2020** **Name: Kishan shetty**

Course: **Python** **USN: 4AL17EC041**

Topic: Application 9: Build a Data Collector Web App with PostGreSQL and Flask

AFTERNOON SESSION



REPORT:

Application 9: Build a Data Collector Web App with PostGreSQL and Flask

- Python code to build a data collector web application, which collects height data from the user and sends the survey result via e-mail.
- Some of the functions/modules used in this application:

 SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and highperforming database access, adapted into a simple and Pythonic domain language.

 SQL functions which are known to SQLAlchemy with regards to database-specific rendering, return types and argument behavior. Generic functions are invoked like all SQL functions, using the func attribute.

 email.mime module can create a new object structure by creating Message instances, adding attachments and all the appropriate headers manually. For MIME messages the email package provides some convenient subclasses to make things easier.

 The smtplib module defines an SMTP client session object that can be used to send

mail to any Internet machine with an SMTP or ESMTP listener daemon. For details of SMTP and ESMTP operation, consult RFC 821 (Simple Mail Transfer Protocol) and RFC 1869 (SMTP Service Extensions).

 Extended HELO (EHLO) is an Extended Simple Mail Transfer Protocol (ESMTP) command sent by an email server to identify itself when connecting to another email server to start the process of sending an email. The EHLO command tells the receiving server it supports extensions compatible with ESMTP.

 The starttls () command extends the Transport Layer Security (TLS) protocol in order to encrypt the information transmitted using the TLS protocol. Starttls () is mainly used as a protocol extension for communication by e-mail, based on the protocol's SMTP, IMAP and POP.