

# DAILY ASSESSMENT FORMAT

Date:	05-06-2020	Name:	Abhishek
Course:	DIGITAL DESIGN USING HDL	USN:	4a17ec001
Topic:	1]Verilog Tutorials and practice programs 2]Building/ Demo projects using FPGA 3]Task	Semester & Section:	6 & 'A'
Github Repository:	Abhishek-online-courses		

## FORENOON SESSION DETAILS

### Image of session



### What is an FPGA?

What is **FPGA**? FPGA stands for Field Programmable Gate Array. Let's analyze the term:

1. **Field-Programmable**: An **FPGA** is manufactured to be easily reconfigured by developers, designers or customers. To program an FPGA as a specific configuration, Verilog HDL or VHDL (Hardware Description Language) is used as the standard language for FPGA programming.
2. **Gate-Array**: An FPGA consists of an array of programmable logic gates/ blocks such as AND, OR, XOR, NOT, memory elements, DSP components, etc., and reconfigurable interconnects which are to connect logic gates together for performing a specific function.

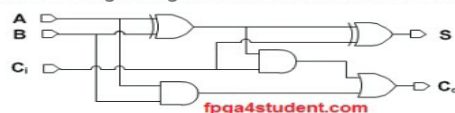
### What is an FPGA?



Thus, FPGAs are nothing, but logic blocks and interconnects that can be programmable by **Hardware Description Languages** (Verilog HDL/ VHDL) to perform different complex functions.

In fact, **FPGAs** can be used to implement almost any DSP algorithm. Some FPGAs also obtain embedded soft-core processors such as Xilinx's MicroBlaze, Altera's Nios II, etc. so that we can use C, C++, etc. to program the processor like what we do with a microcontroller. Besides, the soft processors can communicate with hardware accelerators to speed up complex DSP operations so that we can obtain a better flexible embedded system for niche applications.

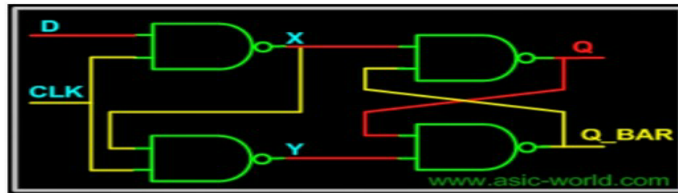
Let's take a very basic example on how to use an FPGA. Let's assume that you are designing a 1-bit full adder and you already obtained the logic diagram of the adder as shown in the figure below.



## Report –

### Verilog Tutorials and practice programs :

#### D-Flip flop from NAND Gate



#### Verilog Code

```
1 module dff_from_nand();
2 wire Q,Q_BAR;
3 reg D,CLK;
4
5 and U1 (X,D,CLK);
6 and U2 (Y,X,CLK);
7 and U3 (Q,Q_BAR,X);
8 and U4 (Q_BAR,Q,Y);
9
10 // Testbench of above code
11 initial begin
12     $monitor( "CLK = %b D = %b Q = %b Q_BAR = %b" ,CLK, D, Q, Q_BAR);
13     CLK = 0;
14     D = 0;
15     #3 D = 1;
16     #3 D = 0;
17     #3 $finish;
18 end
19
20 always #2 CLK = ~CLK;
21
22 endmodule
```

### Building/ Demo projects using FPGA :

Some of the FPGA projects can be FPGA tutorials such as [What is FPGA Programming](#), [image proc on FPGA](#), [matrix multiplication on FPGA](#) Xilinx using Core Generator, [Verilog vs VHDL: Expl Examples and how to load text files or images into FPGA](#). Many others FPGA projects provide st with full Verilog/ VHDL source code to practice and run on FPGA boards. Some of them can be us another bigger FPGA projects.

Following are the [FPGA](#) projects on [fpga4student.com](#):

1. [What is an FPGA? How does FPGA work?](#)
2. [Basys 3 FPGA OV7670 Camera](#)
3. [How to load text file or image into FPGA](#)
4. [Image processing on FPGA using Verilog](#)
5. [License Plate Recognition on FPGA](#)
6. [Alarm Clock on FPGA using Verilog](#)
7. [Digital Clock on FPGA using VHDL](#)
8. [Simple Verilog code for debouncing buttons on FPGA](#)
9. [Traffic Light Controller on FPGA](#)
10. [Car Parking System on FPGA in Verilog](#)
11. [VHDL code for comparator on FPGA](#)
12. [Verilog code for Multiplier on FPGA](#)
13. [N-bit Ring Counter in VHDL on FPGA](#)
14. [Verilog implementation of Microcontroller on FPGA](#)
15. [Verilog Carry Look Ahead Multiplier on FPGA](#)
16. [VHDL Matrix Multiplication on FPGA Xilinx](#)
17. [Fixed Point Matrix Multiplication on FPGA using Verilog](#)
18. [Verilog Divider on FPGA](#)
19. [VHDL code for Microcontroller on FPGA](#)
20. [VHDL code for FIR Filter on FPGA](#)
21. [Verilog code for Digital logic components on FPGA](#)
22. [Delay Timer Implementation on FPGA using Verilog](#)
23. [Single-Cycle MIPS processor on FPGA using Verilog](#)
24. [FIFO Verilog Implementation on FPGA](#)
25. [FIFO VHDL Implementation on FPGA](#)
26. [Verilog D Flip Flop on FPGA](#)
27. [Comparator Design on FPGA using Verilog](#)
28. [D Flip Flop on FPGA using VHDL](#)
29. [Full Adder Design on FPGA using Verilog](#)
30. [Full Adder Design on FPGA using VHDL](#)

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

- Verilog code :

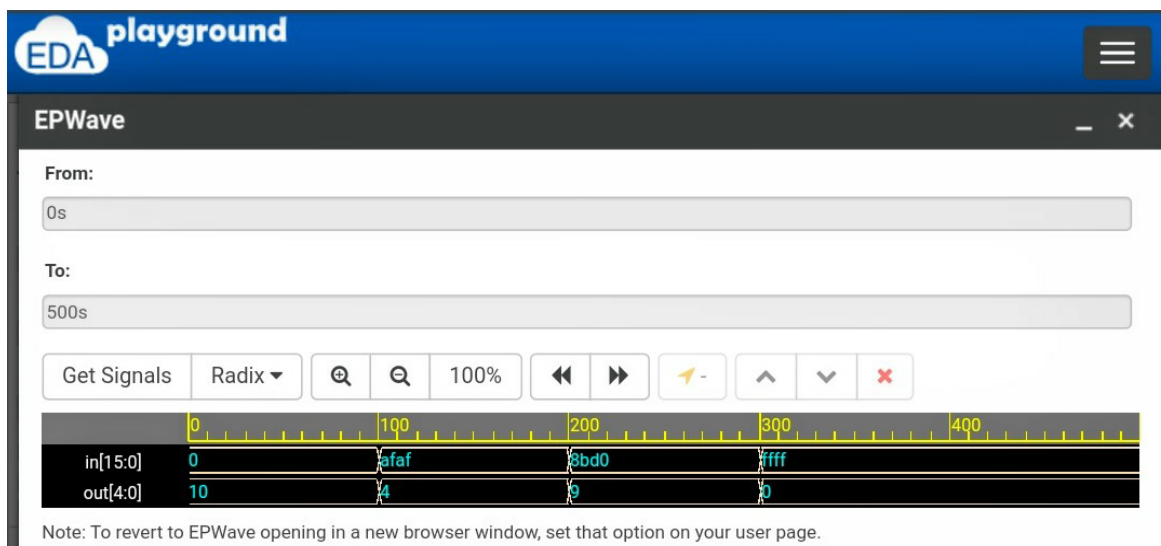
The screenshot shows the EDA Playground interface. On the left, there's a sidebar with 'Languages & Libraries' and 'Tools & Simulators'. The main area is split into two panels: 'testbench.sv' and 'design.sv'. The 'testbench.sv' panel contains code for setting up a stimulus and monitoring the output. The 'design.sv' panel contains the Verilog module 'count\_0' which counts the number of zeros in a 16-bit input. Below the code panels, there's a console window showing the output of the simulation, which lists the input values and the corresponding number of zeros.

```
// Code your SV/Verilog Testbench
module stimulus;
  reg [15:0] in;
  wire [4:0] out;
  count_0 c0(out,in);
  initial
  begin
    $dumpfile("dump1.vcd");
    $dumpvars(1,stimulus);
    in=16'h0000;
    #100;
    in=16'hAFAF;
    #100;
    in=16'h8BD0;
    #100;
    in=16'hFFFF;
    #200;
  end
  initial $monitor($time," -
Input=%b ___ No of
zeroes=%d",in,out);
endmodule

// Code your SV/Verilog Design
module count_0(out,in);
  input [15:0] in;
  output reg [4:0] out;
  integer i;
  always@(in)
  begin
    out=0;
    for(i=0;i<16;i=i+1)
      if (in[i]==1'b0)
        out=out+1;
  end
endmodule
```

[2020-06-05 06:00:14 EDT] iverilog '-Wall' design.sv testbench.sv && unb  
VCD info: dumpfile dump1.vcd opened for output.  
0 - Input=0000000000000000 \_\_\_ No of zeroes=16  
100 - Input=1010111110101111 \_\_\_ No of zeroes= 4  
200 - Input=1000101111010000 \_\_\_ No of zeroes= 9  
300 - Input=1111111111111111 \_\_\_ No of zeroes= 0

- Output :

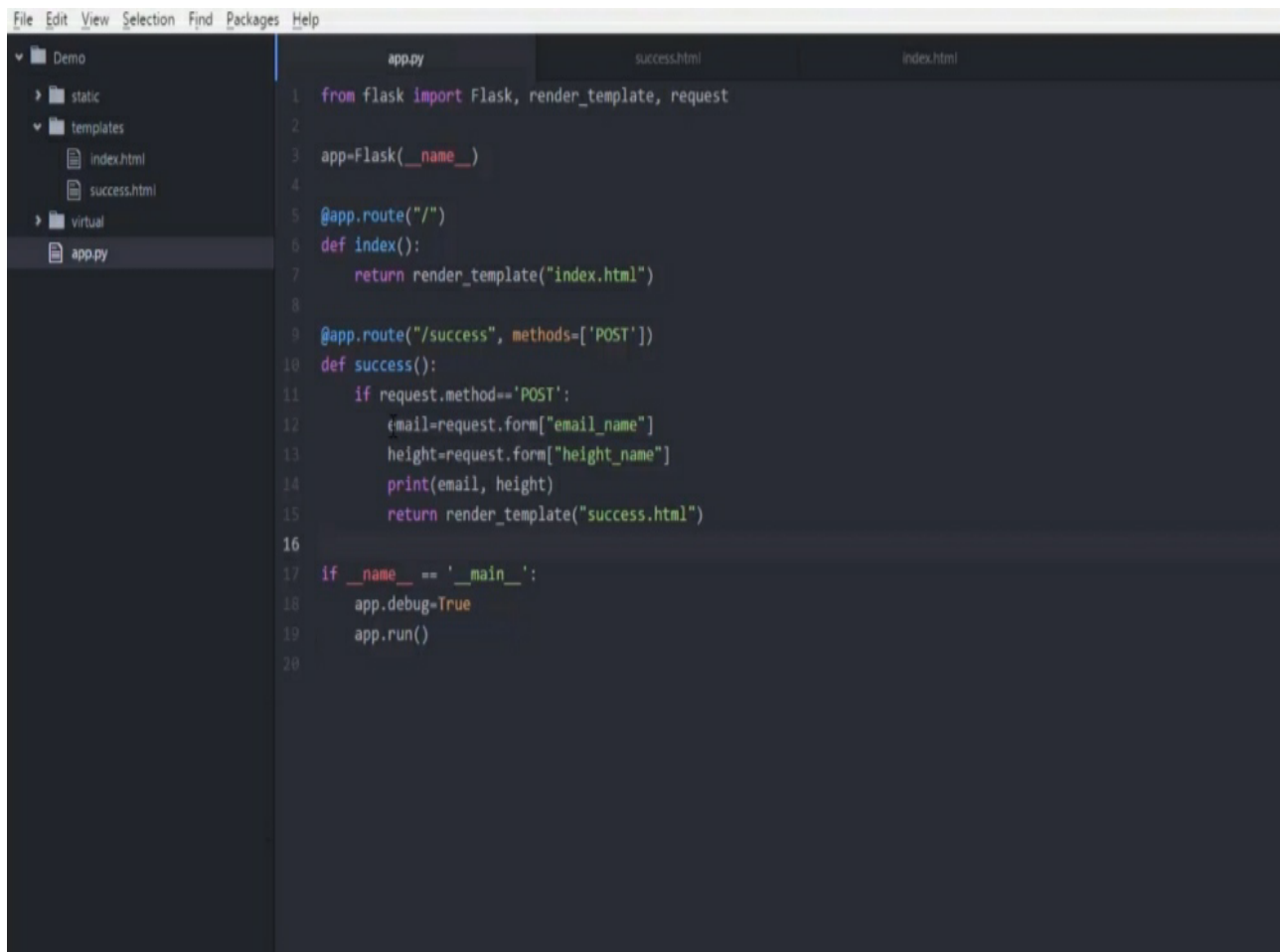


Note : The values displayed in EPWave output are in hexadecimal base.

Date:	05-06-2020	Name:	Abhishek
Course:	The Python Mega Course: Build 10 Real World Applications	USN:	4al17ec001
Topic:	Application 10: Build a Data Collector Web App with PostGreSQL and Flask	Semester & Section:	6 & 'A'

### AFTERNOON SESSION DETAILS

#### Image of session



The screenshot shows a code editor with a dark theme. On the left, a file explorer shows a project structure with folders 'static', 'templates', and 'virtual', and files 'index.html', 'success.html', and 'app.py'. The 'app.py' file is selected. The main editor area displays the following Python code:

```
1 from flask import Flask, render_template, request
2
3 app=Flask(__name__)
4
5 @app.route("/")
6 def index():
7     return render_template("index.html")
8
9 @app.route("/success", methods=['POST'])
10 def success():
11     if request.method=='POST':
12         email=request.form["email_name"]
13         height=request.form["height_name"]
14         print(email, height)
15         return render_template("success.html")
16
17 if __name__ == '__main__':
18     app.debug=True
19     app.run()
20
```

## Report –

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

- **SQLAlchemy** is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.
- **email.mime** module can create a new object structure by creating Message instances, adding attachments and all the appropriate headers manually.
- 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.
- **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.
- **Starttls ()** is mainly used as a protocol extension for communication by e-mail, based on the protocol's SMTP, IMAP and POP.