

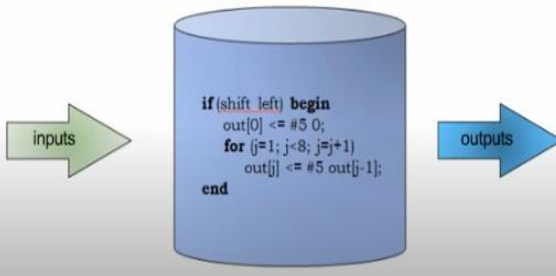
## DAILY ASSESSMENT FORMAT

<b>Date:</b>	<b>02/06/2020</b>	<b>Name:</b>	<b>Shilpa S</b>
<b>Course:</b>	<b>DIGITAL DESIGN USING HDL</b>	<b>USN:</b>	<b>4AL14EC078</b>
<b>Topic:</b>	FPGA Basics: Architecture, Applications and Uses Verilog HDL Basics by Intel Verilog Testbench code to verify the design under test (DUT)	<b>Semester &amp; Section:</b>	<b>8(A)</b>
<b>Github Repository:</b>	<b>Shilpa_online</b>		

### FORENOON SESSION DETAILS

#### Behavior Modeling

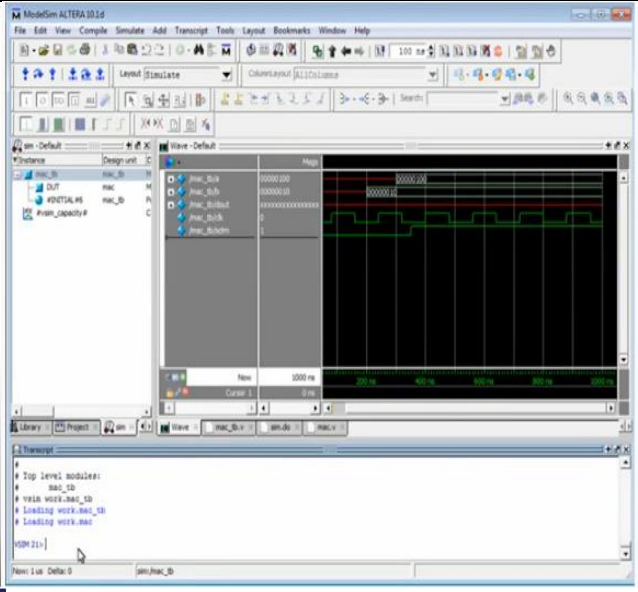
- Only the functionality of the circuit, no structure
- Synthesis tool creates correct logic




```

if (shift_left) begin
    out[0] <= #5 0;
    for (j=1; j<8; j=j+1)
        out[j] <= #5 out[j-1];
    end
                
```

**ALTERA**  
VERILOG HDL





```


module shift_test;
    reg clk, clr, in;    wire out;    integer i;
    shiftreg_4bit SR (clk, clr, in, out);


    initial
        begin clk = 1'b0; #2 clr = 0; #5 clr = 1; end


    always #5 clk = ~clk;

    initial begin #2;
        repeat (2)
            begin #10 in=0; #10 in=0; #10 in=1; #10 in=1; end
        end

    initial
        begin
            $dumpfile ("shifter.vcd");
            $dumpvars (0, shift_test);
            #200 $finish;
        end
    endmodule
                
```




Hardware Modeling Using Verilog

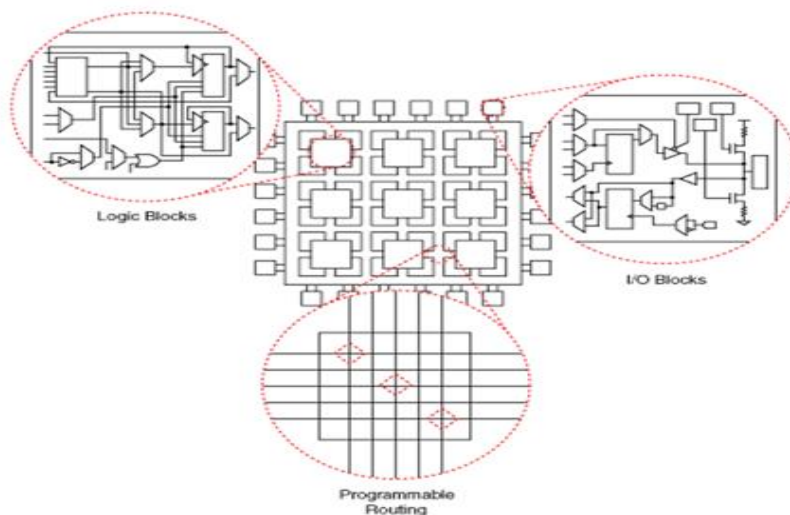


## Report-

The field-programmable gate array (FPGA) is an integrated circuit that consists of internal hardware blocks with user-programmable interconnects to customize operation for a specific application.

### ➤ What is FPGA?

- The field-programmable gate array (FPGA) is an integrated circuit that consists of internal hardware blocks with user-programmable interconnects to customize operation for a specific application.
- The interconnects can readily be reprogrammed, allowing an FPGA to accommodate changes to a design or even support a new application during the lifetime of the part.



- The FPGA has its roots in earlier devices such as programmable read-only memories (PROMs) and programmable logic devices (PLDs).
- These devices could be programmed either at the factory or in the field, but they used fuse technology (hence, the expression “burning a PROM”) and could not be changed once programmed.
- In contrast, FPGA stores its configuration information in a re-programmable medium such as static RAM (SRAM) or flash memory. FPGA manufacturers

include Intel, Xilinx, Lattice Semiconductor, Microchip Technology and Microsemi.

- An FPGA-based design begins by defining the required computing tasks in the development tool, then compiling them into a configuration file that contains information on how to hook up the CLBs and other modules.
- The process is similar to a software development cycle except that the goal is to architect the hardware itself rather than a set of instructions to run on a predefined hardware platform.
- Designers have traditionally used a hardware description language (HDL) such as VHDL or Verilog to design the FPGA configuration.

➤ **FPGA Uses:**

- An Attractive Choice for Certain Applications. The ability to configure the hardware of the FPGA, reconfigure it when needed and optimize it for a particular set of functions makes the FPGA an attractive option in many applications.
- FPGAs are often used to provide a custom solution in situations in which developing an ASIC would be too expensive or time-consuming. An FPGA application can be configured in hours or days instead of months.
- Of course, the flexibility of the FPGA comes at a price: An FPGA is likely to be slower, require more PCB area and consume more power than an equivalent ASIC.
- Even when an ASIC will be designed for high-volume production, FPGAs are widely used for system validation, including pre-silicon validation, post-silicon validation and firmware development.
- This allows manufacturers to validate their design before the chip is produced in the factory.

➤ **FPGA Applications:**

- Many applications rely on the parallel execution of identical operations; the ability to configure the FPGA's CLBs into hundreds or thousands of identical processing blocks has applications in image processing, artificial intelligence (AI),
- Data center hardware accelerators, enterprisenetworking and automotive advanced driver assistance systems (ADAS).
- Many of these application areas are changing very quickly as requirements evolve and new protocols and standards are adopted.
- FPGAs enable manufacturers to implement systems that can be updated when necessary.
- A good example of FPGA use is high-speed search: Microsoft is using FPGAs in its data centers to run Bing search algorithms.
- The FPGA can change to support new algorithms as they are created. If needs change, the design can be repurposed to run simulation or modeling routines in an HPC application.
- This flexibility is difficult or impossible to achieve with an ASIC. Other FPGA uses include aerospace and defense, medical electronics, digital television, consumer electronics, industrial motor control, scientific instruments, cybersecurity systems and wireless communications.

### ➤ **FPGA History: What Comes Next?**

With these emerging applications, the FPGA market is growing at a healthy clip: It was valued at \$5.34 billion in 2016 and is expected to grow to \$9.50 billion in 2023, according to industry researchers MarketsandMarkets. That's a compound annual growth rate (CAGR) of 8.5 percent, compared to a CAGR of about 2 percent for the much larger (\$74 billion) general microprocessor market. The exponential growth of data, and the emergence of fast-changing fields such as AI, machine learning, HPC and

genomics, require architectures that are fast, flexible and adaptable. FPGAs are well-positioned to take advantage of these new opportunities.

### ➤ **HDL:**

Verilog is a **HARDWARE DESCRIPTION LANGUAGE (HDL)**. It is a language used for describing a digital system like a network switch or a microprocessor or a memory or a flip-flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits. Verilog supports a design at many levels of abstraction. The major three are –

- Behavioral level
- Register-transfer level
- Gate level.

### ➤ **Register–Transfer Level**

Designs using the Register–Transfer Level specify the characteristics of a circuit using operations and the transfer of data between the registers. Modern definition of an RTL code is "Any code that is synthesizable is called RTL code".

### ➤ **Gate Level**

Within the logical 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 (basic gates). Gate level modelling may not be a right idea for logic design. Gate level

code is generated using tools like synthesis tools and his netlist is used for gate level simulation and for backend.

### **TODAY'S TASK :**

**Implement a 4:1 MUX and write the test bench code to verify the module**

#### **STRUCTURAL:**

```
module and_gate(output a, input b, c, d);
```

```
assign a = b & c & d;
```

```
endmodule
```

```
module not_gate(output f, input e);
```

```
assign e = ~ f;
```

```
endmodule
```

```
module or_gate(output l, input m, n, o, p);
```

```
assign l = m | n | o | p;
```

```
endmodule
```

```
module m41(out, a, b, c, d, s0, s1);
```

```
output out;
```

```
input a, b, c, d, s0, s1;
```

```
wire s0bar, s1bar, T1, T2, T3;
```

```
not_gate u1(s1bar, s1);
```

```
not_gate u2(s0bar, s0);
```

```
and_gate u3(T1, a, s0bar, s1bar);
```

```
and_gate u4(T2, b, s0, s1bar);
```

```
and_gate u5(T3, c, s0bar, s1);
```

```
and_gate u6(T4, d, s0, s1);
```

```
or_gate u7(out, T1, T2, T3, T4);
```

```
endmodul
```

#### **TESTBENCH:**

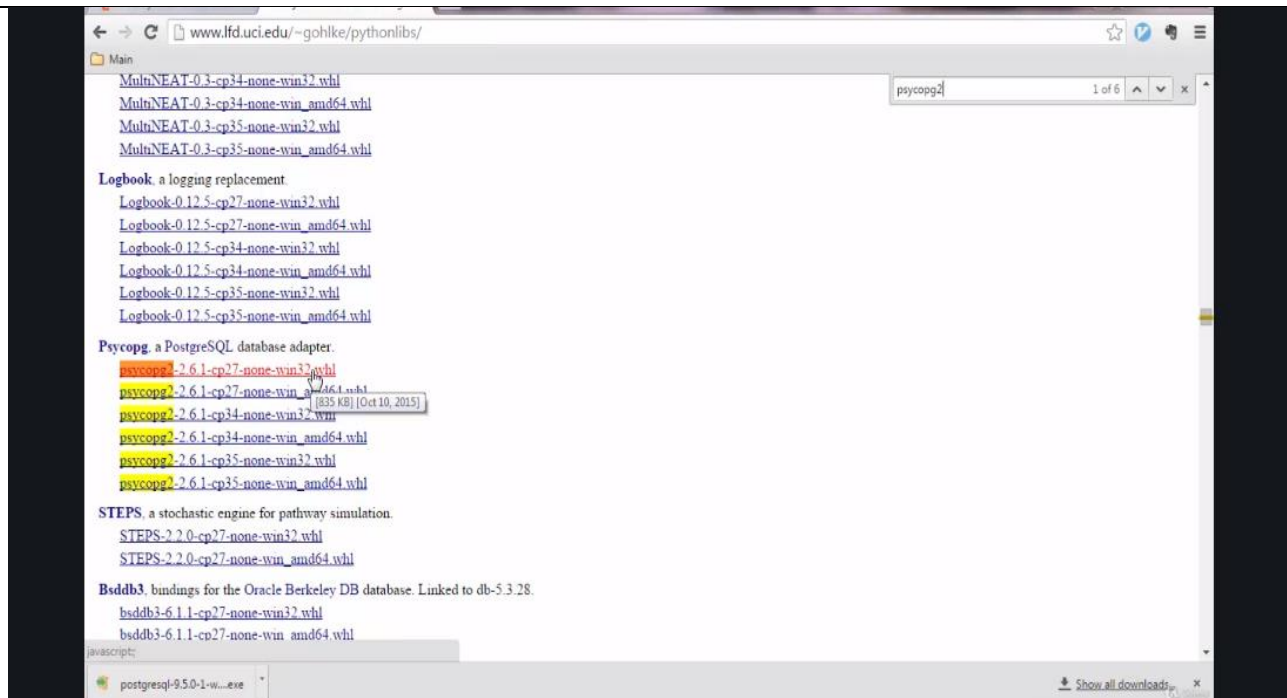
```
module top;
wire out;
reg a;
reg b;
reg c;
reg d;
reg s0, s1;
m41 name(.out(out), .a(a), .b(b), .c(c), .d(d), .s0(s0), .s1(s1));
initial
begin
a=1'b0; b=1'b0; c=1'b0; d=1'b0;
s0=1'b0; s1=1'b0;
#500 $finish;
end
always #40 a=~a;
always #20 b=~b;
always #10 c=~c;
always #5 d=~d;
always #80 s0=~s0;
always #160 s1=~s1;
always@(a or b or c or d or s0 or s1)
$monitor("At time = %t, Output = %d", $time, out);
endmodule;
```

<b>Date:</b>	<b>02/06/2020</b>	<b>Name:</b>	<b>Shilpa S</b>
<b>Course:</b>	<b>Python</b>	<b>USN:</b>	<b>4AL14EC078</b>
<b>Topic:</b>	Interactive data visualization with bokeh, webscraping with python beautiful soup.	<b>Semester &amp; Section:</b>	<b>8( A)</b>
<b>Github Repository:</b>	<b>Shilpa_online</b>		

### AFTERNOON SESSION DETAILS

#### Image of session





## REPORT

If you haven't installed Bokeh yet, you can easily install it with pip from the terminal:

```
pip install bokeh
```

Or you use pip3:

```
pip3 install bokeh
```

Snippet producing the triangle based plot

```
#Making a basic Bokeh line graph
```

```
#importing Bokeh
```

```
from bokeh.plotting import figure
```

```
from bokeh.io import output_file, show
```

```
#prepare some data
```

```
x=[3,7.5,10]
y=[3,6,9]
.
.#prepare the output file
.output_file("Line.html")
.
.#create a figure object
.f=figure()
.
.#create line plot
.f.triangle(x,y)
.
.#write the plot in the figure object
.show(f)

#Snippet producing the circle based plot

#Making a basic Bokeh line graph

#importing Bokeh
from bokeh.plotting import figure
from bokeh.io import output_file, show

#prepare some data
x=[3,7.5,10]
y=[3,6,9]
.
.#prepare the output file
.output_file("Line.html")
```

```

.#create a figure object
.f=figure()

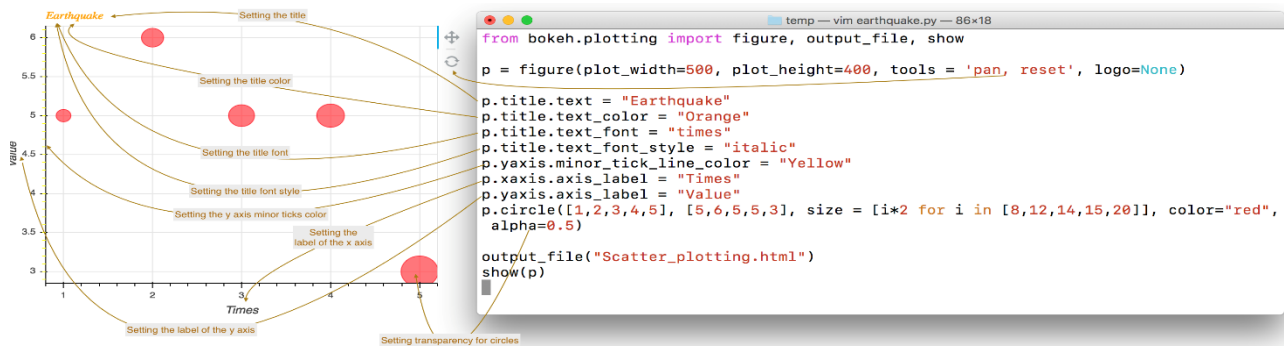
.#create line plot
.f.circle(x,y)

.#write the plot in the figure object
.show(f)

```

## Visual Attributes

Once you have built a basic plot, you can customize its visual attributes including changing the title color and font, adding labels for xaxis and yaxis, changing the color of the axis ticks, etc. All these properties are illustrated in the diagram below:



And here is the code if you want to play around with it:

```

from bokeh.plotting import figure, output_file, show

p = figure(plot_width=500, plot_height=400, tools = 'pan, reset')

p.title.text = "Earthquakes"
p.title.text_color = "Orange"
p.title.text_font = "times"

```

```
p.title.text_font_style = "italic"
p.yaxis.minor_tick_line_color = "Yellow"
p.xaxis.axis_label = "Times"
p.yaxis.axis_label = "Value"
p.circle([1,2,3,4,5], [5,6,5,5,3], size = [i*2 for i in [8,12,14,15,20]], color="red",
alpha=0.5)
.output_file("Scatter_plotting.html")
.show(p)
```

For a complete list of visual attributes, see the [Styling Visual Attributes](#) documentation page of Bokeh.