**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **2nd June 2020** | **Name:** | **Rashmitha** |
| **Course:** | **Digital design using HDL** | **USN:** | **4AL17EC077** |
| **Topic:** | **FPGA basics:architecture ,application and uses,verilog testbench code to verify the design under test** | **Semester & Section:** | **6th sem ‘B’ sec** |
| **Github Repository:** | **Rashmitha** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session**  C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot (226).png  C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot (227).png ****What is FPGA?**** The [**field-programmable gate array (FPGA)**](https://www.arrow.com/en/categories/programmable-devices/programmable-logic-devices/fpgas) 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**](https://www.arrow.com/en/manufacturers/intel)**,** Xilinx, [**Lattice Semiconductor**](https://www.arrow.com/en/manufacturers/lattice-semiconductor)**,**[**Microchip Technology**](https://www.arrow.com/en/manufacturers/microchip-technology) and **[Microsemi](https://www.arrow.com/en/manufacturers/microsemi).**  **FPGA Architecture :**  A basic FPGA architecture (Figure 1) consists of thousands of fundamental elements called configurable logic blocks (CLBs) surrounded by a system of programmable interconnects, called a fabric, that routes signals between CLBs. Input/output (I/O) blocks interface between the FPGA and external devices.Depending on the manufacturer, the CLB may also be referred to as a logic block (LB), a logic element (LE) or a logic cell (LC). An individual CLB is made up of several logic blocks. A lookup table (LUT) is a characteristic feature of an FPGA. An LUT stores a predefined list of logic outputs for any combination of inputs: LUTs with four to six input bits are widely used. Standard logic functions such as multiplexers (mux), full adders (FAs) and flip-flops are also common. ****CPLD vs FPGA:****Originally, FPGAs included the blocks in Figure 1 and little else, but now designers canchoose from products with a large range of features. Less complex devices such as simple programmable logic devices (SPLDs) and complex programmable logic devices (CPLDs) bridge the gap between discrete logic devices and entry-level FPGAs.Entry-level FPGAs emphasize low power consumption, low logic density and low complexity per chip. Higher-function devices add functional blocks dedicated to specific functions: Examples include clock management components, phase-locked loops (PLLs), high-speed serializers and deserializers, Ethernet MACs, PCI express controllers and high-speed transceivers. These blocks can either be implemented with CLBs—termed soft IP—or designed as separate circuits; i.e., hard IP. Hard IP blocks gain performance at the expense of reconfigurability..FPGA Design:How do we transform this collection of thousands of hardware blocks into the correct configuration to execute the application? 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 (Figure 4) or Verilog to design the FPGA configuration.****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, enterprise networking 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.  **Basics of 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  Behavioral level: This level describes a system by concurrent algorithms (Behavioural). Every algorithm is sequential, which means it consists of a set of instructions that are executed one by one. Functions, tasks and 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 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.  **Task for Day-2**  **Implement a 4:1 MUX and write the test bench code to verify the module**  module tb\_4to1\_mux;  reg [3:0] a,b,c,d;  wire [3:0] out;  reg [1:0] sel;  integer i;  mux\_4to1\_case mux0( .a (a),  .b (b),  .c (c),  .d(d),  .sel(sel),  .out(out));  initial begin  $monitor (“[%0t] sel=0x%0h a=0x%0h b=0x%0h c=0x%0h d=0x%0h out=0x%0h”, $time,sel,a,b,c,d);  sel <=0;  a <= $random;  b <= $random;  c<= $random;  d<= $random;  for(i=1; i<=4; i=i+1) begin  #5 sel <= I;  end  #5 $finish;  end  endmodule |
|  |

**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **2nd June 2020** | **Name:** | **Rashmitha** |
| **Course:** | **Python** | **USN:** | **4AL17EC077** |
| **Topic:** | **Interactive data visualization with bokeh,webscraping with python beautiful soup** | **Semester & Section:** | **6th sem ‘B’ sec** |
| **Github Repository:** | **Rashmitha** |  |  |

|  |
| --- |
| **AFTERNOON SESSION DETAILS** |
| **Image of session**  C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot (228).png  C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot (229).png  **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    2. #prepare some data 3. x=[3,7.5,10] 4. y=[3,6,9] 6. #prepare the output file 7. output\_file("Line.html") 9. #create a figure object 10. f=figure() 12. #create line plot 13. f.triangle(x,y) 15. #write the plot in the figure object   show(f)  **Snippet producing the circle based plot:**   1. #Making a basic Bokeh line graph 3. #importing Bokeh 4. from bokeh.plotting import figure 5. from bokeh.io import output\_file, show 7. #prepare some data 8. x=[3,7.5,10] 9. y=[3,6,9] 11. #prepare the output file 12. output\_file("Line.html") 14. #create a figure object 15. f=figure() 17. #create line plot 18. f.circle(x,y) 20. #write the plot in the figure object 21. show(f)   **Plot Properties:**  You can add a title to the plot, set the figure width and height, change title font, etc. Below is a summary of properties which can be added to change the style of the plot:   1. import pandas 2. from bokeh.plotting import figure, output\_file, show 4. p=figure(plot\_width=500,plot\_height=400, tools='pan',logo=None) 6. p.title.text="Cool Data" 7. p.title.text\_color="Gray" 8. p.title.text\_font="times" 9. p.title.text\_font\_style="bold" 10. p.xaxis.minor\_tick\_line\_color=None 11. p.yaxis.minor\_tick\_line\_color=None 12. p.xaxis.axis\_label="Date" 13. p.yaxis.axis\_label="Intensity" 15. p.line([1,2,3],[4,5,6]) 16. output\_file("graph.html") 17. show(p)   **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:   1. from bokeh.plotting import figure, output\_file, show 2. p = figure(plot\_width=500, plot\_height=400, tools = 'pan, reset') 3. p.title.text = "Earthquakes" 4. p.title.text\_color = "Orange" 5. p.title.text\_font = "times" 6. p.title.text\_font\_style = "italic" 7. p.yaxis.minor\_tick\_line\_color = "Yellow" 8. p.xaxis.axis\_label = "Times" 9. p.yaxis.axis\_label = "Value" 10. 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) 11. output\_file("Scatter\_plotting.html") 12. show(p)   **Request Headers:**  r = requests.get("http://www.pythonhow.com/example.html")  please use this instead:  r = requests.get("http://www.pyclass.com/example.html", headers={'User-agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:61.0) Gecko/20100101 Firefox/61.0'})  The rest of the code stays the same.  So, we're just changing the domain name from pythonhow to pyclass and we're adding a header argument. Some webpages don't like scripts sometimes, so adding a header allows the script to impersonate a web browser. |
|  |