**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **05-06-2020** | **Name:** | **Karthik J** |
| **Course:** | **DSDV** | **USN:** | **4AL16EC030** |
| **Topic:** | Hardware modelling using Verilog | **Semester & Section:** | **8TH A** |
| **GitHub Repository:** | Karthik-J |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| What is [FPGA](https://en.wikipedia.org/wiki/Field-programmable_gate_array)? FPGA stands for Field Programmable Gate Array.1. Field-Programmable: An [FPGA](https://www.fpga4student.com/2017/08/what-is-fpga-programming.html) 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.Thus, FPGAs are nothing, but logic blocks and interconnects that can be programmable by [Hardware Description Languages](https://en.wikipedia.org/wiki/Hardware_description_language) (Verilog HDL/ VHDL) to perform different complex functions. In fact, [FPGA](https://www.fpga4student.com/2017/08/what-is-fpga-programming.html)s 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.  What is FPGA  As mentioned above, there are necessary logic gates on FPGA such as XOR, AND and OR in order to implement the above adder on FPGA. To demonstrate the operation of the adder on FPGA, either Verilog or VHDL can be easily used to connect those gates together as shown in the logic diagram of the adder. An example Verilog code for the adder: -- FPGA Project: What is an FPGA?  -- Verilog example code for Adder on FPGA  module fpga4student\_adder(input A,B,Ci, output S,Co);  wire tmp1,tmp2,tmp3; //FPGA projects  xor u1(tmp1,A,B); // Verilog projects  and u2(tmp2,A,B);  and u3(tmp3,tmp1,Ci);  or u4(Co,tmp2,tmp3);  xor u5(S,tmp1,Ci);  endmodule An example VHDL code for the adder: -- FPGA projects: What is an FPGA?  -- VHDL example code for adder on FPGA  library ieee;  use ieee.std\_logic\_1164.all;  entity fpga4student\_Adder is  port( A, B, Ci : in std\_logic;  S, Co : out std\_logic);  end fpga4student\_Adder;  architecture structural of fpga4student\_Adder is  signal tmp1, tmp2, tmp3: std\_logic;  begin  tmp1 <= A xor B;  tmp2 <= A and B;  tmp3 <= tmp1 and Ci;  Co <= tmp2 or tmp3;  S <= tmp1 xor Ci;  end structural;  The [Verilog/ VHDL](https://www.fpga4student.com/2017/08/verilog-vs-vhdl-explain-by-example.html) code for the adder simply does the "wiring" job by connecting the logic gates on FPGA together to be functional as a full adder. After verifying the code by simulation, we can synthesize, run the adder on FPGA and see how it works. For simulation, there are various available simulators that we can use, but for students, I would recommend you to use ModelSim of Mentor Graphics. You can see more simulators [here](https://en.wikipedia.org/wiki/List_of_HDL_simulators). For FPGA synthesis and FPGA programming, you can use Quartus II for Altera's FPGA Boards and Xilinx ISE or Vivado for Xilinx's FPGA Boards.  As an FPGA designer for several years, here are five reasons why I love FPGA design: 1. Very fast on-chip (FPGA) demonstrationThe top reason why I love [FPGA design](https://www.fpga4student.com/2017/08/what-is-fpga-programming.html) is that it is very fast to verify a design on FPGA. While ASICs could take several months only for tape-out and another latency for PCB design, everything is settled on FPGA and we just need to download the program file to FPGA using a specific FPGA programming tool and see how it works on FPGA. FPGA boards of Xilinx and Altera provide necessary  IOs and additional support circuits such as LCD, single LEDs, 7-segment LEDs, communication ports (USB, UART, VGA, HDMI, PS2, FMC, etc.), ADCs, DACs, CODEC, etc. so that FPGA can easily communicate with other chips for the verification process.2. Simple and fast design process on FPGAAnother great thing to say about FPGA is that the design process is pretty simple and really easy to learn. The design flow for ASICs is very complicated and time-consuming since it needs a lot of complex steps for designing, verification, and implementation. On the other hand, FPGA design process mostly avoids sophisticated and time-consuming steps like Floor-planing, Timing Analysis, Physical Implementation, etc. because FPGA is already a characterized and verified chip. Of course, when needed, FPGA vendors also provide necessary tools for floorplanning and timing analysis to enable users optimizing performance for niche very-demanding designs. In fact, FPGA design flow only takes several steps such as HDL design and coding, functional simulation, synthesis, timing or post-synthesis simulation if needed, and Place And Route. Furthermore, many FPGA design tools are free and very easy for users to learn and design. FPGA vendors provide free user guides and tutorials to facilitate user's learning process. It could take very short time for students to be familiar with FPGA design if they have a good background in digital logic design.3. FPGA's programmabilityThe highlight feature of FPGA we obviously could not omit is its programmability. While ASICs or microcontrollers are fixed in term of hardware (it can be programmable at the software level), FPGAs can be programmable at the hardware level. We can program FPGAs to perform almost any digital complex functionality and reconfigure it to whatever we want in the future if needed. FPGAs can be programmed as a microprocessor, a microcontroller, DSPs, VGA controllers, digital filters, etc.4. FPGA's high performanceAnother superb feature in [FPGA design](https://www.fpga4student.com/2017/08/what-is-fpga-programming.html) is high performance. While processor-based ASICs or DSPs are sequential executed, FPGAs exploits the hardware parallelism to obtain a breakthrough performance for demanding designs. Thus, FPGAs provides faster implementations that processor-based ASICs could not match. The FPGA's parallelism can be effectively exploited to implement digital signal processing algorithms in order to speed up the processing time.5. FPGA's flexibility FPGAs are more and more flexible for designers to make their own applications. As mentioned above, FPGA vendors provide their own soft processors such as Xilinx's MicroBlaze, Altera's Nios II, etc. so that designers can be more flexible in design and programming process. Indeed, on FPGA, you can use the soft processors with the software-level-reconfigurable capability(C, C++, etc.) for low and average speed applications, and FPGA hardware accelerators with the hardware-level-reconfigurable capability(Verilog/ VHDL) for high-speed operations. Thus, designers can obtain a suitable embedded system which meets their design requirements. For example, when designing an embedded real-time tracking system, designers can use the soft processor for camera interface and FPGA hardware accelerators for tracking processing.  There is no doubt that our daily lives are significantly affected by electronic engineering technology. This is true on the domestic scene, in our professional disciplines, in the workplace, and in leisure activities. Indeed, even at school, tomorrow's adults are exposed to and are coming to terms with quite- sophisticated electronic devices and systems. There is no doubt that revolutionary changes have taken place in a relatively short time and it is also certain that even more-dramatic advances will be made in the next decade.  Electronics as we know it today is characterized by reliability, low power dissipation, extremely low weight and volume, and low cost, coupled with an ability to cope easily with a high degree of sophistication and complexity. Electronics, and in particular the integrated circuit, has made possible the design of powerful and flexible processors which provide highly intelligent and adaptable devices for the user. Integrated circuit memories have provided the essential elements to complement these processors and, together with a wide range of logic and analog integrated circuitry, they have provided the system designer with components of considerable capability and extensive application. Furthermore, the revolutionary advances in technology have not yet by any means run their full course and the potential for future developments is exciting to say the least. Up until the 1950s electronic active device technology was dominated by the vacuum tube and, although a measure of miniaturization and circuit integration did take place, the technology did not lend itself to miniaturization as we have come to accept it today. Thus the vast majority of present-day electronics is the result of the invention of the transistor in 1947.  The invention of the transistor by William B. Shockley, Walter H. Brattain and John Bardeen of Bell Telephone Laboratories was followed by the development of the Integrated Circuit (IC). The very first IC emerged at the beginning of 1960 and since that time there have already been four generations of ICs: SSI (small scale integration), MSI (medium scale integration), LSI (large scale integration), and VLSI (very large-scale integration). Now we are beginning tp see the emergence of the fifth generation, ULSI (ultra large scale integration) which is characterized by complexities in excess of 3 million devices on a single IC chip. Further miniaturization is still to come and more revolutionary advances in the application of this technology must inevitably occur.  As we look back proudly on 25 years of delivering engineering know-how to engineers worldwide, it's exciting that an EDA and semiconductor IP leader like Synopsys is supporting us in ensuring that aspiring designers can develop their skills by accessing such market-leading solutions.   This collaboration will enable broader and deeper adoption of key subjects like System Verilog and UVM,“ says **Michael Sanie, Synopsys Senior Director of Verification Marketing.** “Synopsys is supportive of initiatives from partners such as Doulos that will assist more customers to become effective and productive in addressing their verification challenges.  Playground |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date:** | | **05-06-2020** | **Name:** | **Karthik J** |  |
| **Course:** | | [Programming with Python: Hands-on Introduction for Beginners](https://www.udemy.com/course/python-programming-beginners/) | **USN:** | **4AL16EC030** |  |
| **Topic:** | |  | **Semester & Section:** | **8th A** |  |
|  | **AFTERNOON SESSION DETAILS** | | | | |
|  | **Image of session** | | | | |
|  | **Conditional statements**  Condition statements are a block of statements whose execution depends on a certain condition.  Different types of conditional statements in Python  1. If:  A “simple if” condition is one where a block of statements get executed if the condition mentioned in the “if” statement evaluates to true  Example:  distance = 100  if distance == 100:  print(“Distance is 100”)  2. If-Else:  An “If-Else” statement is one where a block of statements under “if” condition gets executed if the condition evaluates to true.  If the condition evaluates to false, the block of statements under “else” is executed.  Example:  distance = 200  if distance <= 100:  print(“Distance is less than or equal to 100”)  else:  print(“Distance is greater than 100”)  3. If-Elif-Else  An “If-Elif-Else” statement is one where multiple “if” conditions are evaluated one after another if an “if” statement evaluates to false. “elif” stands for else-if. If all the if conditions evaluates to false, the block of statements under “else” gets executed.  Example:  distance = 400  if distance <= 100:  print(“Distance is less than or equal to 100”)  elif distance <= 200:  print(“Distance is less than or equal to 200”)  elif distance <= 300: print(“Distance is 300”)  else: print(“Distance is greater than 300”)  4. Nested If  An if statement within another if statement is called a nested if statement.  Example:  distance = 50  if distance < 100:  if distance == 50:  print “Distance is 50” Dictionary A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values. Example Create and print a dictionary:  thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } print(thisdict) Python - Tuples A tuple is an immutable sequence of Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.  Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.  For example −  tup1 = ('physics', 'chemistry', 1997, 2000);  tup2 = (1, 2, 3, 4, 5 );  tup3 = "a", "b", "c", "d"; Accessing Values in Tuples To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.  For example −  tup1 = ('physics', 'chemistry', 1997, 2000);  tup2 = (1, 2, 3, 4, 5, 6, 7 );  print "tup1[0]: ", tup1[0];  print "tup2[1:5]: ", tup2[1:5]; Updating Tuples Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples  Example  tup1 = (12, 34.56);  tup2 = ('abc', 'xyz');  # Following action is not valid for tuples  # tup1[0] = 100;  # So, let's create a new tuple as follows  tup3 = tup1 + tup2;  print tup3; Delete Tuple Elements Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.  To explicitly remove an entire tuple, just use the **del** statement.  example −  tup = ('physics', 'chemistry', 1997, 2000);  print tup;  del tup;  print "After deleting tup : ";  print tup;  This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any-more − Basic Tuples Operations Tuples respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string. Indexing, Slicing, and Matrixes Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input −  L = ('spam', 'Spam', 'SPAM!') | | | | |