**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **04-06-2020** | **Name:** | **Karthik J** |
| **Course:** | **DSDV** | **USN:** | **4AL16EC030** |
| **Topic:** | Hardware modelling using Verilog | **Semester & Section:** | **8TH A** |
| **GitHub Repository:** | Karthik-J |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| NPTEL :: Computer Science and Engineering - NOC:Hardware modeling ...  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.   Doulos plans include innovations and extensions to EDA Playground in support of the roll-out of online training and blended learning solutions. “Today's engineers can access so much of the information they need online,“ says **John Aynsley, Doulos CTO**, “To some extent that goes for acquiring knowledge as well; but productive design requires engineers that are highly skilled; and skills are only sharpened through hands-on practise with the support of experts who know how to teach. The best of tomorrow's training solutions will blend all of the above. EDA Playground will play a key part in enabling this.“  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  What is EDA Playground? EDA Playground gives engineers immediate hands-on exposure to simulating SystemVerilog, Verilog, VHDL, C++/SystemC, and other HDLs. All you need is a web browser. The goal is to accelerate learning of design/testbench development with easier code sharing and simpler access to EDA tools and libraries.   * With a simple click, run your code and see console output in real time. * View waves for your simulation using [EPWave](http://epwave.readthedocs.org) browser-based wave viewer. * Save your code snippets (“Playgrounds”). * Share your code and simulation results with a web link. Perfect for web forum discussions or emails. Great for asking questions or sharing your knowledge. * Quickly try something out   + Try out a language feature with a small example.   + Try out a library that you’re thinking of using.  Example Usecases  * **Quick prototyping** – try out syntax or a library/language feature. * When **asking questions on** [Stack Overflow](http://stackoverflow.com/) or other online forums, attach a link to the code and simulation results. * Use during **technical interviews** to test candidates’ SystemVerilog/Verilog coding and debug skills. * Try verifying using **different verification frameworks**: UVM, SVUnit, plain Verilog, or Python.  Tools & Simulators For settings and options documentation, see [Tools & Simulators Options](https://eda-playground.readthedocs.io/en/latest/settings.html#tools-simulators-options-label)  Available tools and simulators are below. EDA Playground can support many different tools. [Contact us](http://www.doulos.com) to add your EDA tool to EDA Playground.  Playground |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date:** | | **03-06-2020** | **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!') | | | | |