**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **01-06-2020** | **Name:** | **Karthik J** |
| **Course:** | **DSDV** | **USN:** | **4AL16EC030** |
| **Topic:** | Industry applications of FPGA,  FPGA Business Fundamentals  FPGA vs ASIC Design flow  FPGA Basics-A look under the hood. | **Semester & Section:** | **8TH A** |
| **GitHub Repository:** | Karthik-J |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image Section**           * **ASIC:** * ASIC stands for Application-Specific Integrated Circuit. Furthermore, as the name implies, it is a chip that serves the purpose for which it has been designed and does not permit reprogramming or modification. * Which, in turn, means that it cannot perform another function or execute another application once programming is complete.Since the ASIC’s design is for a specific function, * This determines how the chip receives its programming. The programming process itself consists of drawing the resultant circuit permanently into the silicon.In terms of applications * ASIC chip technology is in use in [electronic devices](https://resources.pcb.cadence.com/blog/2019-how-sustainable-electronics-materials-are-the-future-of-the-pcb-industry) such as laptops, smartphones, and TVs, to give you an idea of the scope of their use. * **FPGA:** * [Field Programmable Gate Array or FPGA](https://www.cadence.com/content/cadence-www/global/en_US/home/tools/pcb-design-and-analysis/design-authoring/fpga-planner.html) is in direct competition with ASIC chip technology. * Also, FPGA is, in essence, a chip that can be programmed and reprogrammed to perform numerous functions at any single point in time.Furthermore, a single chip is comprised of thousands of units called logic blocks, that are linked with programmable interconnects. * The FPGA’s circuit is made by connecting several configurable blocks, and it has a rigid internal structure. In summary, an FPGA is essentially a programmable version of an ASIC.      * Overall, the FPGA affords general functionality that allows programming to your specifications. * However, like most things in life, there are side effects of FPGA’s versatility. In this case, it is an increased cost, increased internal delay, and limited analog functionality. * **FPGA versus ASIC: A Side-By-Side Comparison** * **NRE**: NRE stands for Non-Recurring Engineering costs. As you can imagine, with the words recurring and costs, in the same sentence, every business is concerned when they hear those two words. * So, it is safe to say that this is an essential deciding factor. Moreover, in the case of ASIC, this is exceptionally high, whereas, with FPGA, it is nearly non-existent. * However, in the grand scheme, the total cost gets lower and lower the more significant the quantity you require in terms of ASIC. Furthermore, FPGA can cost you more overall since its individual costs are higher per unit than ASIC. * **Design Flow**: Every engineer and PCB designer prefer a more trouble-free and simplistic design process. Just because what you do is complex, does not mean that you want the process itself to be complicated. * Therefore, in terms of the simplicity of design flow, FPGA is hands down less complicated than ASIC. * This is due to the FPGA’s flexibility, versatility, shorter time to market, and the fact that it is reprogrammable. Whereas, with ASIC, it is more involved in terms of design flow because it is not reprogrammable, and it requires costly [dedicated EDA tools](https://resources.pcb.cadence.com/blog/2019-eda-tools-for-pcb-design-what-you-should-be-looking-for) for the design process. * **Performance and Efficiency**: In terms of performance, ASICs outperforms FPGAs by a small margin, primarily due to lower power consumption and the various possible functionalities that you can layer onto a single chip. * Also, FPGA has a more rigid internal structure, whereas, with an ASIC, you can design it to excel in power consumption or speed. * **Cost**: Even with the increased NRE cost, ASICs are thought to be more cost-effective, all things considered as compared to FPGAs, which are only profitable when developed in smaller quantities. * **Power Consumption**: As I mentioned previously, ASICs require less power and thus provide a better option than the higher power consumption FPGA. Especially with electronic devices that are battery operated. * **Size**: In terms of size, it is a matter of physics. With an ASIC, its design is for one functionality; therefore, it consists of precisely the number of gates required for the desired application. However, with FPGA’s multifunctionality, a single unit will be significantly larger, because of its internal structure and a specific size that you cannot change. * **Time to Market**: Also, as mentioned earlier, FPGA affords a faster time to market than ASIC due to its simplicity in terms of the design flow. Moreover, ASICs also require layouts, back end processes, and advanced verification, all of which are time-consuming. * **Configuration**: Overall, the most apparent difference between FPGA and ASIC is programmability. Therefore, the logical conclusion here is FPGA offers more options in terms of flexibility. FPGAs are not only flexible, but they also provide “hot-swappable” functionality that allows modification even while in use. * **Operating Frequency**: In terms of design specifications, FPGAs have limited operating frequencies. This is one of those side effects of its flexibility (reprogrammable). However, with ASICs more focused approach to functionality, it can operate at higher frequencies.   **TASK FOR DAY 1**  **Verilog code for NAND gate – All modelling styles**  **GATE LEVEL MODELLING**    module NAND\_2\_gate\_level (output Y, input A, B);  wire Yd;  and (Yd,A,B);  not(Y,Yd);  endmodule  **DATA FLOW MODELLING**  module NAND\_2\_data\_flow(output Y,input A,B);  assign Y = ~(A & B);  endmodule  **BEHAVIOURAL MODELLING**  module NAND\_2\_behavioural(output reg Y,input A,B) ;  always @(A or B) begin  if( A == 1’b1 & B == 1’b1) begin  Y =1’b0;  end  else  Y = 1’b1;  end  endmodule  **TESTBENCH OF THE NAND GATE USING VERILOG**  `include "NAND\_2\_behavioral.v"  module NAND\_2\_behavioral\_tb;  reg A, B;  wire Y;  NAND\_2\_behavioral Indtance0 (Y, A, B);  initial begin  A = 0; B = 0;  #1 A = 0; B = 1;  #1 A = 1; B = 0;  #1 A = 1; B = 1;  end  initial begin  $monitor ("%t | A = %d| B = %d| Y = %d", $time, A, B, Y);  $dumpfile("dump.vcd");  $dumpvars();  end  endmodule |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date:** | | **01-06-2020** | **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** | | | | |
|  | 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!') | | | | |