**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **05-06-2020** | **Name:** | **Kiran N** |
| **Course:** | **DIGITAL DESIGN USING**  **HDL** | **USN:** | **4al16ec031** |
| **Topic:** | **●Verilog Tutorials and practice**  **programs**  **●Building/ Demo projects using**  **FPGA** | **Semester & Section:** | **8th and A** |
| **Github Repository:** | **Kiran-course** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| Report  begin  q <= d;  q\_bar <= !d;  end  endmodule  One can describe a simple Flip flop as that in above figure as well as one can describe a  complicated designs having 1 million gates. Verilog is one of the HDL languages available in the  industry for designing the Hardware. Verilog allows us to design a Digital design at Behavior  Level,Register Transfer Level (RTL), Gate level and at switch level. Verilog allows hardware  designers to express their designs with behavioral constructs, deterring the details of  implementation to a later stage of design in the final design.  Many engineers who want to learn Verilog, most often ask this question, how much time it will take to learn Verilog?, Well my answer to them is "It may not take more then one week, if you happen to know at least one programming language".  Design Styles Verilog like any other hardware description language, permits the designers to  design a design in either Bottom−up or Top−down methodology.  Bottom−Up Design  The traditional method of electronic design is bottom−up. Each design is performed at the  gate−level using the standard gates ( Refer to the Digital Section for more details) With increasing  complexity of new designs this approach is nearly impossible to maintain. New systems consist of  ASIC or microprocessors with a complexity of thousands of transistors. These traditional  bottom−up designs have to give way to new structural, hierarchical design methods. Without these  new design practices it would be impossible to handle the new complexity.  Top−Down Design  The desired design−style of all designers is the top−down design. A real top−down design allows  early testing, easy change of different technologies, a structured system design and offers many  other advantages. But it is very difficult to follow a pure top−down design. Due to this fact most  designs are mix of both the methods, implementing some key elements of both design styles.  Figure shows a Top−Down design approach    Abstraction Levels of Verilog  Verilog supports a design at many different levels of abstraction. Three of them are very important:  •Behavioral level  •Register−Transfer Level  •Gate Level  Behavioral level  This level describes a system by concurrent algorithms (Behavioral). Each algorithm itself is  sequential, that means it consists of a set of instructions that are executed one after the other.  Functions, Tasks and Always 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 by operations and the transfer of data between the registers. An explicit clock is used. RTL  design contains exact timing possibility, operations are scheduled to occur at certain times. Modern  definition of a RTL code is "Any code that is synthesizable is called RTL code".  Gate Level  Within the logic 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 (AND, OR, NOT etc gates). Using  gate level modelling might not be a good idea for any level of logic design. Gate level code is generated by tools like synthesis tools and this netlist is used for gate level simulation and for backend.  Data Types  Verilog Language has two primary data types  •Nets − represents structural connections between components.  •Registers − represent variables used to store data.  Every signal has a data type associated with it:  •Explicitly declared with a declaration in your Verilog code.  Implicitly declared with no declaration but used to connect structural building blocks in your code.  •Implicit declaration is always a net type "wire" and is one bit wide.  Types of Nets  Each net type has functionality that is used to model different types of hardware (such as  PMOS,NMOS, CMOS, etc)  Register Data Types  Registers store the last value assigned to them until another assignment statement changes their value.  •Registers represent data storage constructs.  •You can create arrays of the regs called memories.  •register data types are used as variables in procedural blocks.  •A register data type is required if a signal is assigned a value within a procedural block  •Procedural blocks begin with keyword initial and always.  Some of the FPGA projects can be FPGA tutorials such as  What is FPGA Programming, image processing on FPGA, matrix multiplication on FPGA Xilinx using Core Generator, Verilog vs VHDL: Explain by Examples and how to load text files or images into FPGA  . Many others FPGA projects provide students with full Verilog/ VHDL source code to practice and run on FPGA boards. Some of them can be used for another bigger FPGA projects.  Task code  module num\_zero(input [15:0]A, output reg [4:0]zeros);  integer i;  always@(A)  begin  zeros=0;  for(i=0;i<16;i=i+1)  zeros=zeros+A[i];  end  endmodule  test bench code  module test;  reg [15:0]A;  wire [4:0] zeros;  num\_zero out (.A(A), .zeros(zeros));  initial begin  $dumpfile("dumo.vcd");  $dumpvars(1,test);  A=16'hFFFF;  #100;  A=16'hF56F;  #100;  A=16'h3FFF;  #100;  A=16'h0001;  #100;  A=16'hF10F;  #100;  A=16'hF822;  #100;  A=16'h7ABC; #100;  end  endmodule |