**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | 02/06/20202 | **Name:** | Sheela Golasangi |
| **Course:** | DIGITAL DESIGN USING HDL | **USN:** | 4AL16EC068 |
| **Topic:** | FPGA Basics: Architecture,  Applications and Uses  Verilog HDL Basics by Intel  Verilog Testbench code to  verify the design under test  (DUT) | **Semester & Section:** | VIII  ‘B’ |
| **Github Repository:** | Sheela - Course |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
|  |
| **Report-**  The field-programmable gate array (FPGA) is an integrated circuit that consists of internal hardware blocks with user-programmable interconnects to customize operation for a specific application.  **What is FPGA?**  The field-programmable gate array (FPGA) 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, Xilinx, Lattice Semiconductor, Microchip Technology and Microsemi.  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 or  Verilog to design the FPGA configuration.  **FPGA Uses:**  An Attractive Choice for Certain Applications. The ability to configure the hardware of the FPGA, reconfigure it when needed and optimize it for a particular set of functions makes the FPGA an attractive option in many applications. FPGAs are often used to provide a custom solution in situations in which developing an ASIC would be too expensive or time-consuming. An FPGA application can be configured in hours or days instead of months. Of course, the flexibility of the FPGA comes at a price: An FPGA is likely to be slower, require more PCB area and consume more power than an equivalent ASIC.  Even when an ASIC will be designed for high-volume production, FPGAs are widely used for system validation, including pre-silicon validation, post-silicon validation and firmware development. This allows manufacturers to validate their design before the chip is produced in the factory.  **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. Other FPGA uses include aerospace and defense, medical electronics, digital television, consumer electronics, industrial motor control, scientific instruments, cybersecurity systems and wireless communications.  **FPGA History: What Comes Next?**  With these emerging applications, the FPGA market is growing at a healthy clip: It was valued at $5.34 billion in 2016 and is expected to grow to $9.50 billion in 2023, according to industry researchers Markets and Markets. That’s a compound annual growth rate (CAGR) of 8.5 percent, compared to a CAGR of about 2 percent for the much larger ($74 billion) general microprocessor market. The exponential growth of data, and the emergence of fast-changing fields such as AI, machine learning, HPC and genomics, require architectures that are fast, flexible and adaptable. FPGAs are well-positioned to take advantage of these new opportunities.  **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.  **TODAY’S TASK :**  I**mplement a 4:1 MUX and write the test bench code to verify the module**  STRUCTURAL:  module and\_gate(output a, input b, c, d);  assign a = b & c & d;  endmodule  module not\_gate(output f, input e);  assign e = ~ f;  endmodule  module or\_gate(output l, input m, n, o, p);  assign l = m | n | o | p;  endmodule  module m41(out, a, b, c, d, s0, s1);  output out;  input a, b, c, d, s0, s1;  wire s0bar, s1bar, T1, T2, T3;  not\_gate u1(s1bar, s1);  not\_gate u2(s0bar, s0);  and\_gate u3(T1, a, s0bar, s1bar);  and\_gate u4(T2, b, s0, s1bar);  and\_gate u5(T3, c, s0bar, s1);  and\_gate u6(T4, d, s0, s1);  or\_gate u7(out, T1, T2, T3, T4);  endmodul  **TESTBENCH:**  module top;  wire out;  reg a;  reg b;  reg c;  reg d;  reg s0, s1;  m41 name(.out(out), .a(a), .b(b), .c(c), .d(d), .s0(s0), .s1(s1));  initial  begin  a=1'b0; b=1'b0; c=1'b0; d=1'b0;  s0=1'b0; s1=1'b0;  #500 $finish;  end  always #40 a=~a;  always #20 b=~b;  always #10 c=~c;  always #5 d=~d;  always #80 s0=~s0;  always #160 s1=~s1;  always@(a or b or c or d or s0 or s1)  $monitor("At time = %t, Output = %d", $time, out);  endmodule; |