**A**

**Laboratory Manual for**

**VLSI Design**

**(3151105)**

**B. E. Electronics and Communication**

**Semester 5**





**Directorate of Technical Education, Gandhinagar, Gujarat**

**VGEC, Chandkheda, Ahmedabad**

**Department of Electronics & Communication Engineering**

**Certificate**

This is to certify that Mr./Ms. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ Enrollment No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of B. E. Semester-V Electronics and Communication Engineering of this Institute (GTU Code: 11) has satisfactorily completed the Practical / Tutorial work for the subject **VLSI Design (3151105)** for the academic year 2023-24.

Place: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Name and Sign of Faculty member**

**Head of the Department**

**Practical – Course Outcome Matrix**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Course Outcomes (COs):**   1. **To understand fabrication and operation of MOSFET device.** 2. **To obtain and verify the mathematical model of MOSFET device.** 3. **Analyze and design Static and Dynamic CMOS Circuits.** 4. **To understand techniques of DFT.** 5. **To write programs in Verilog/VHDL for combinational and sequential circuits and realize them on FPGA.** | | | | | | |
| **Sr. No.** | **Objective(s) of Experiment** | **CO**  **1** | **C**  **O2** | **CO**  **3** | **CO**  **4** | **CO**  **5** |
| 1. | Introduction to FPGA and CPLD. |  |  |  |  | **√** |
| 2. | Introduction to Hardware Description Languages (HDLs). |  |  |  |  | **√** |
| 3. | Introduction to Xilinx / Tool |  |  |  |  | **√** |
| 4. | Implement all the basic Logic Gates and Boolean functions using different modeling styles in Verilog/ VHDL:   1. Structural modeling 2. Dataflow modeling 3. Behavioural modeling |  |  | **√** |  | **√** |
| 5. | Design Adder & Subtractor using Verilog / VHDL. a. Half Adder (structural and dataflow modeling)   1. Full Adder using Half Adder (structural modeling) 2. Full Adder (dataflow and behavioural modeling) 3. Half Subtractor 4. Full Subtractor |  |  | **√** |  | **√** |
| 6. | Design Binary-to-Gray & Gray-to-Binary encoder using Verilog/VHDL |  |  | **√** |  | **√** |
| 7. | Design Multiplexer and Demultiplexer using Verilog/  VHDL.   1. 2:1 Mux (dataflow and behavioural modeling) 2. 4:1 Mux (structural and dataflow modeling) 3. 8:1 Mux (using 4:1 and 2:1 Mux: structural modeling) 4. 16:1 Mux (using behavioural modeling & 4:1 Mux:   structural modeling)   1. 1:8 Demux |  |  | **√** |  | **√** |
| 8. | Design 2:4, 3:8, 4:16 Decoders using Verilog/VHDL |  |  | **√** |  | **√** |
| 9. | Design 4\*2, 8\*3 Priority Encoder using Verilog / VHDL. |  |  | **√** |  | **√** |
| 10. | Design 4-bit Comparator using Verilog / VHDL. |  |  | **√** |  | **√** |
| 11. | Design BCD and Ripple Carry Adder using Verilog / VHDL. |  |  | **√** |  | **√** |
| 12. | Design of S-R and D latches using structural / behavioural modelling using Verilog/VHDL. |  |  | **√** |  | **√** |
| 13. | Design positive edge triggered D-FF with asynchronous /synchronous active high reset using Verilog / VHDL. |  |  | **√** |  | **√** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 14. | Design serial in serial out and serial in parallel out shift registers using Verilog/VHDL. |  |  | **√** |  | **√** |
| 15. | Design Counter using Verilog/VHDL. a. BCD Counter.  b. Up-Down Counter |  |  | **√** |  | **√** |
| 16. | Introduction to Microwind tool. | **√** |  | **√** |  |  |
| 17. | Implementation of Resistive load and CMOS inverters using Microwind. | **√** |  | **√** |  |  |
| 18. | Implementation of CMOS NAND and NOR gate using Microwind. | **√** |  | **√** |  |  |

# Index

**(Progressive Assessment Sheet)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sr. No. | Title of Experiment | Page No. | Date of perform ance | Date of submiss ion | Assessme  nt  Marks | Sign. of  Teacher with date | Remar ks |
| 1 | Introduction to FPGA and CPLD. |  |  |  |  |  |  |
| 2 | Introduction to Hardware Description Languages (HDLs). |  |  |  |  |  |  |
| 3 | Introduction to Xilinx / Tool |  |  |  |  |  |  |
| 4 | Implement all the basic Logic Gates and Boolean functions using different modeling styles in Verilog/ VHDL:   1. Structural modeling 2. Dataflow modeling 3. Behavioural modeling |  |  |  |  |  |  |
| 5 | Design Adder & Subtractor using Verilog / VHDL.   1. Half Adder (structural and dataflow modeling) 2. Full Adder using Half Adder (structural modeling) 3. Full Adder (dataflow and behavioural modeling) 4. Half Subtractor 5. Full Subtractor |  |  |  |  |  |  |
| 6 | Design Binary-to-Gray & Gray-to-Binary encoder using Verilog/VHDL |  |  |  |  |  |  |
| 7 | Design Multiplexer and Demultiplexer using  Verilog/ VHDL.   1. 2:1 Mux (dataflow and behavioural modeling) 2. 4:1 Mux (structural and dataflow modeling) 3. 8:1 Mux (using 4:1 and 2:1 Mux: structural modeling) 4. 16:1 Mux (using behavioural modeling & 4:1 Mux: structural modeling) 1:8 Demux |  |  |  |  |  |  |
| 8 | Design 2:4, 3:8, 4:16 Decoders using Verilog/VHDL |  |  |  |  |  |  |
| 9 | Design 4\*2, 8\*3 Priority Encoder using Verilog / VHDL. |  |  |  |  |  |  |
| 10 | Design 4-bit Comparator using Verilog / VHDL. |  |  |  |  |  |  |
| 11 | Design BCD and Ripple Carry Adder using Verilog / VHDL. |  |  |  |  |  |  |
| 12 | Design of S-R and D latches using structural / behavioral modeling usingVerilog/VHDL. |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 13 | Design positive edge triggered D-FF with asynchronous /synchronous active high reset using Verilog / VHDL. |  |  |  |  |  |  |
| 14 | Design serial in serial out and serial in parallel out shift registers using Verilog/VHDL. |  |  |  |  |  |  |
| 15 | Design Counter using Verilog/VHDL. a. BCD Counter.  b. Up-Down Counter |  |  |  |  |  |  |
| 16 | Introduction to Microwind tool. |  |  |  |  |  |  |
| 17 | Implementation of Resistive load and CMOS inverters using Microwind. |  |  |  |  |  |  |
| 18 | Implementation of CMOS NAND and NOR gate using Microwind. |  |  |  |  |  |  |
|  | Total |  |  |  |  |  |  |

# Experiment No: 1

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Introduction to FPGA & CPLD.**

**Competency and Practical Skills:**

**Relevant CO:**

**Objectives:**

**Equipment / Instruments:** Laptop or Computer with Xilinx / LTspiceTools.

**Basic Theory:**

**What is an FPGA?**

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. This feature distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom manufactured for specific design tasks. Although one-time programmable (OTP) FPGAs are available, the dominant types are SRAM based which can be reprogrammed as the design evolves.

**What are the differences between an ASIC and an FPGA?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **ASIC (Application-Specific Integrated Circuit)** | **FPGA (Field Programmable Gate Array** |
| Definition | Custom-designed chip for a specific task | Reconfigurable chip, programmable after fabrication |
| Flexibility | Fixed – cannot be changed after fabrication | Reprogrammable – can be updated anytime |
| Performance | Very high (optimized for speed & efficiency) | Moderate (slower due to programmable fabric) |
| Power Efficiency | Low power consumption (highly optimized) | Higher power consumption (less efficient) |
| Time to Market | Long (design + fab cycle) | Short (can configure immediately) |
| Example | |  | | --- | |  |  |  | | --- | | Apple A-series, Snapdragon, Nvidia GPU | | Xilinx, Intel (Altera) FPGA boards |

**FPGA Architecture:**

A basic FPGA architecture shown below consists of thousands of fundamental elements called configurable logic blocks (CLBs) surrounded by a system of programmable interconnects, called a fabric, that routes signals between CLBs. Input/output (I/O) blocks interface between the FPGA and external devices. Depending on the manufacturer, the CLB may also be referred to as a logic block (LB), a logic element (LE) or a logic cell (LC).

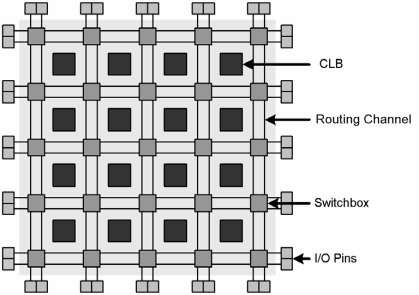


Fig 1.1 General Architecture of FPGA

**Explain the structure of Switch matrix:**

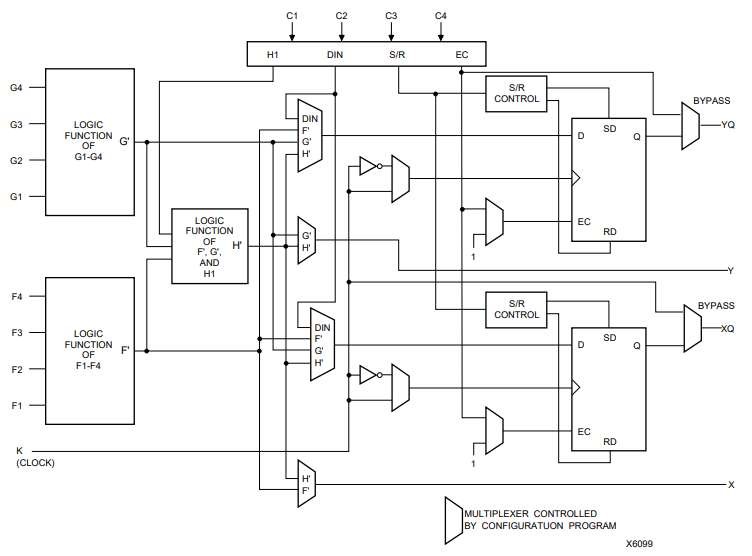


Fig 1.2 Simplified Block Diagram of XC4000-Families CLB

The XC4000 families achieve high speed through advanced semiconductor technology and through improved architecture, and supports system clock rates of up to 50 MHz. Compared to older Xilinx FPGA families, the XC4000 families are more powerful, offering on-chip RAM and wide-input decoders. They are more versatile in their applications, and design cycles are faster due to a combination of increased routing resources and more sophisticated software. And last, but not least, they more than double the available complexity, up to the 20,000-gate level. The XC4000 families have 16 members, ranging in complexity from 2,000 to 25,000 gates.

The CLBs provide the functional elements for constructing the user’s logic. The most important one is a more powerful and flexible CLB surrounded by a versatile set of routing resources, resulting in more “effective gates per CLB.” The principal CLB elements are shown in Figure 1.2. Each new CLB also packs a pair of flip-flops and two independent 4-input function generators. The two function generators offer designers plenty of flexibility because most combinatorial logic functions need less than four inputs. Consequently, the design-software tools can deal with each function generator independently, thus improving cell usage.

**FPGA Applications:**

* **Aerospace & Defense -** Radiation-tolerant FPGAs along with intellectual property for image processing, waveform generation, and partial reconfiguration for SDRs.
* **ASIC Prototyping -** ASIC prototyping with FPGAs enables fast and accurate SoC system modeling and verification of embedded software
* **Automotive -** Automotive silicon and IP solutions for gateway and driver assistance systems, comfort, convenience, and in-vehicle infotainment. - Learn how AMD FPGA's enable Automotive Systems
* **Broadcast & Pro AV** - Adapt to changing requirements faster and lengthen product life cycles with Broadcast Targeted Design Platforms and solutions for high-end professional broadcast systems.
* **Consumer Electronics -** Cost-effective solutions enabling next generation, full-featured consumer applications, such as converged handsets, digital flat panel displays, information appliances, home networking, and residential set top boxes.
* **Data Center -** Designed for high-bandwidth, low-latency servers, networking, and storage applications to bring higher value into cloud deployments.
* **High Performance Computing and Data Storage** - Solutions for Network Attached Storage (NAS), Storage Area Network (SAN), servers, and storage appliances.
* **Industrial -** AMD FPGAs and targeted design platforms for Industrial, Scientific and Medical (ISM) enable higher degrees of flexibility, faster time-to-market, and lower overall non-recurring engineering costs (NRE) for a wide range of applications such as industrial imaging and surveillance, industrial automation, and medical imaging equipment.
* **Medical** - For diagnostic, monitoring, and therapy applications, the Virtex FPGA and Spartan™ FPGA families can be used to meet a range of processing, display, and I/O interface requirements.
* **Video & Image Processing -** FPGAs and targeted design platforms enable higher degrees of flexibility, faster time-to-market, and lower overall non-recurring engineering costs (NRE) for a wide range of video and imaging applications.
* **Wired Communications** - End-to-end solutions for the Reprogrammable Networking Linecard Packet Processing, Framer/MAC, serial backplanes, and more
* **Wireless Communications** - RF, base band, connectivity, transport and networking solutions for wireless equipment, addressing standards such as WCDMA, HSDPA, WiMAX and others.

**CPLD**

A Complex Programmable Logic Device (CPLD) is a combination of a fully programmable AND/OR array and a bank of microcells. The AND/OR array is reprogrammable and can perform a multitude of logic functions. Microcells are functional blocks that perform combinatorial or sequential logic, and also have the added flexibility for true or complement, along with varied feedback paths.

Traditionally, CPLDs have used analog sense amplifiers to boost the performance of their architectures. This performance boost came at the cost of very high current requirements.

**Advantages of CPLD:**

CPLDs perform a variety of useful functions in systems design due to their unique capabilities and as the market leader in programmable logic solutions, AMD provides a total solution to a designer's CPLD needs. Understanding the features and benefits of using CPLDs can help enable ease of design, lower development costs, and speed products to market.

**Comparison of CPLD and FPGA**

|  |  |  |
| --- | --- | --- |
| **Feature** | CPLD | FPGA |
| Architecture | |  | | --- | |  |  |  |  | | --- | --- | |  | Consists of a few large logic blocks connected via a programmable interconnect | | Consists of thousands/millions of small logic blocks (LUTs) with rich interconnect |
| Logic Capacity | Low to medium (hundreds to a few thousand gates) | Very high (hundreds of thousands to millions of gates) |
| Configuration | Non-volatile (retains configuration after power-off) | Volatile (needs external memory or flash to load configuration on power-up) |
| Speed | Faster for simple logic due to predictable timing | Higher overall performance for complex designs but less predictable timing |
| Power Consumption | Low, good for simple applications | Higher (due to larger logic and routing resources) |
| cost | Lower cost (for small designs) | Higher cost (but scales better for large designs) |

**Conclusion**:

**Suggested Reference:**

<https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>https://www.xilinx.com/products/silicon-devices/cpld/cpld.html <https://media.digikey.com/pdf/data%20sheets/xilinx%20pdfs/xc4000,a,h.pdf>

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 2

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Introduction to Hardware Description Languages (HDLs).**

**Competency and Practical Skills:**

**Relevant CO:**

**Objectives:**

**Equipment / Instruments:** Laptop or Computer with Xilinx / Tool

**Basic Theory:**

**What Are Hardware Description Languages (HDLs)?**

HDLs are indeed similar to programming languages but not exactly the same. We utilize a programming language to build or create software, whereas we use a hardware description language to describe or express the behavioral characteristics of digital logic circuits.

We utilize HDLs for designing processors, motherboards, CPUs (i.e., computer chips), as well as various other digital circuitry.

**What Is VHDL?**

Very High-Speed Integrated Circuit Hardware Description Language (VHDL) is a description language used to describe hardware. It is utilized in electronic design automation to express mixedsignal and digital systems, such as ICs (integrated circuits) and FPGA (field-programmable gate arrays). We can also use VHDL as a general-purpose parallel programming language.

We utilize VHDL to write text models that describe or express logic circuits. If the text model is part of the logic design, the model is processed by a synthesis program. The next step in the process incorporates a simulation program to test the logic design. During this step, we utilize the simulation models to characterize the logic circuits that interface to the design. We refer to this collection of simulation models as a testbench.

Typically, a VHDL simulator is an event-driven simulator which means that we add each transaction to an event queue for a particular scheduled time. For example, if a signal assignment occurs after one nanosecond, we add the event to the queue as time + 1 ns. Although a zero delay is allowed, it still must be scheduled, and for these scenarios we utilize a Delta delay.

**VHDL Functionality**

* These simulations alternate between two modes:

* Statement Execution: In this mode, the triggered statements are evaluated. Event Processing: During this mode, the events in the queue are processed.

Though there is an inherent similarity in hardware designs, VHDL has processes that can make the necessary accommodations. However, these processes differ in syntax from the parallel processes in tasks (Ada).

Similarly to Ada, VHDL is a predefined part of the programming language, plus, it is not case sensitive. However, VHDL provides various features that are unavailable in Ada, e.g., an extensive set of Boolean operators which include nor and nand. These additional features enable VHDL to precisely represent operations that are customary in hardware.

Another feature of VHDL is it has file output and input capabilities that you can utilize as a general purpose language for text processing. Although, we typically see them in use by a simulation testbench for data verification or stimulus. Specific VHDL compilers build executable binaries, which afford the option to use VHDL to write a testbench for functionality verification designs utilizing files on the host computer to compare expected results, user interaction, and define stimuli.

Note: **Ada** is a statically typed, structured, object-oriented, and imperative high-level programming language; it is an extension that derives from Pascal and other programming languages.

**What Is Verilog?**

As I am sure you are aware, Verilog is also a Hardware Description Language. It employs a textual format to describe electronic systems and circuits. In the area of electronic design, we apply Verilog for verification via simulation for testability analysis, fault grading, logic synthesis, and timing analysis.

Verilog is also more compact since the language is more of an actual hardware modeling language. As a result, you typically write fewer lines of code, and it elicits a comparison to the C language. However, Verilog has a superior grasp on hardware modeling as well as a lower level of programming constructs. Verilog is not as wordy as VHDL, which accounts for its compact nature. Although VHDL and Verilog are similar, their differences tend to outweigh their similarities.

Verilog HDL is an IEEE standard (IEEE 1364). It received its first publication in 1995, with a subsequent revision in 2001. SystemVerilog, which is the 2005 revision of Verilog, is the latest publication of the standard. We call the IEEE Verilog standard document the LRM (Language Reference Manual). Currently, the IEEE 1364 standard defines the PLI (Programming Language Interface).

Note: The PLI is a collective of software routines that allows a bidirectional interface between other languages such as C and Verilog.

**VHDL vs Verilog**

|  |  |  |
| --- | --- | --- |
| **Sr.**  **No.** | **VHDL** | **Verilog** |
| 1 | Strongly typed | Weakly typed |
| 2 | Easier to understand | Less code to write |
| 3 | More natural in use | More of a hardware modeling language |
| 4 | Non-C-like syntax | Similarities to the C language |
| 5 | Variables must be described by data type | A lower level of programming constructs |
| 6 | Widely used for FPGAs and military | A better grasp on hardware modeling |
| 7 | More difficult to learn | Simpler to learn |

The most important thing to remember when you are writing HDL code is that you are describing real hardware, not writing a computer program. The most common beginner’s mistake is to write HDL code without thinking about the hardware you intend to produce. If you don’t know what hardware you are implying, you are almost certain not going to get what you want. Instead, begin by sketching a block diagram of your system, identifying which portions are combinational logic, which portions are sequential circuits or finite state machines, and so forth. Then, write HDL code for each portion, using the correct idioms to imply the kind of hardware you need.

**Conclusion**:

**Suggested Reference:**

[https://resources.pcb.cadence.com/blog/2020-hardware-description-languages-vhdl-vs-verilogand-their-functional-uses](https://resources.pcb.cadence.com/blog/2020-hardware-description-languages-vhdl-vs-verilog-and-their-functional-uses)

https://www.sciencedirect.com/topics/computer-science/hardware-description-languages

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 3

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Introduction to Xilinx / Tool**

**Competency and Practical Skills:**

**Relevant CO:**

**Objectives:**

**Equipment / Instruments:** Laptop or Computer with Xilinx / Tool

**Basic:**

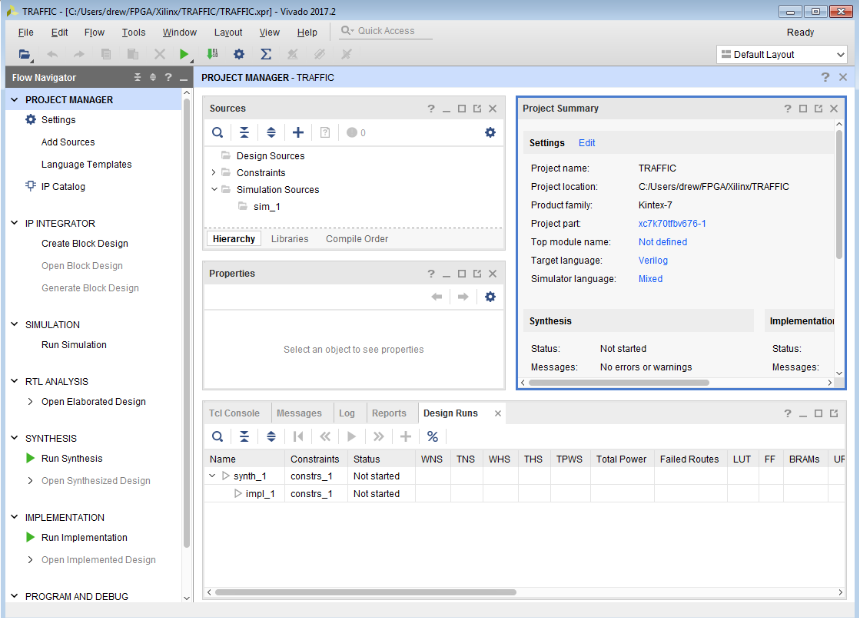
The Xilinx ISE software controls all aspects of the design flow. Through the Project Navigator interface, one can access all of the design entry and design implementation tools. One can also access the files and documents associated with their project.

Fig. 2.1 Project Navigator

The Project Navigator interface is divided into four panel sub-windows, as seen in Figure 2.1.

* On the top left are the Start, Design, Files, and Libraries panels, which include display and access to the source files in the project as well as access to running processes for the currently selected source. The Start panel provides quick access to opening projects as well as frequently access reference material, documentation and tutorials.

* At the bottom of the Project Navigator are the Console, Errors, and Warnings panels, which display status messages, errors, and warnings.

* To the right is a multi-document interface (MDI) window referred to as the Workspace. The Workspace enables person to view design reports, text files, schematics, and simulation waveforms.

* Each window can be resized, undocked from Project Navigator, moved to a new location within the main Project Navigator window, tiled, layered, or closed.

* One can use the View > Panels menu commands to open or close panels. You can use the Layout > Load Default Layout to restore the default window layout.

* The Design panel provides access to the View, Hierarchy, and Processes panes.

* The View pane radio buttons enable you to view the source modules associated with the Implementation or Simulation Design View in the Hierarchy pane. If you select Simulation, you must select a simulation phase from the drop-down list.

* The Hierarchy pane displays the project name, the target device, user documents, and design source files associated with the selected Design View.

* The View pane at the top of the Design panel allows you to view only those source files associated with the selected Design View, such as Implementation or Simulation.

* Each file in the Hierarchy pane has an associated icon. The icon indicates the file type (HDL file, schematic, core, or text file, for example).

* For a complete list of possible source types and their associated icons, see the “Source File Types” topic in the ISE Help.

* From Project Navigator, select Help > Help Topics to view the ISE Help. If a file contains lower levels of hierarchy, the icon has a plus symbol (+) to the left of the name. One can expand the hierarchy by clicking the plus symbol (+). One can open a file for editing by double-clicking on the filename.

* The Console provides all standard output from processes run from Project Navigator. It displays errors, warnings, and information messages.

* The Workspace is where design editors, viewers, and analysis tools open. These include ISE Text Editor, Schematic Editor, Constraint Editor, Design Summary/Report Viewer, RTL and Technology Viewers, and Timing Analyzer.

* Other tools such as the PlanAhead™ software for I/O planning and floorplanning, ISim, third-party text editors, XPower Analyzer, and iMPACT open in separate windows outside the main Project Navigator environment when invoked.

* The Design Summary provides a summary of key design data as well as access to all of the messages and detailed reports from the synthesis and implementation tools.

* The summary lists high-level information about your project, including overview information, a device utilization summary, performance data gathered from the Place and Route (PAR) report, constraints information, and summary information from all reports with links to the individual reports.

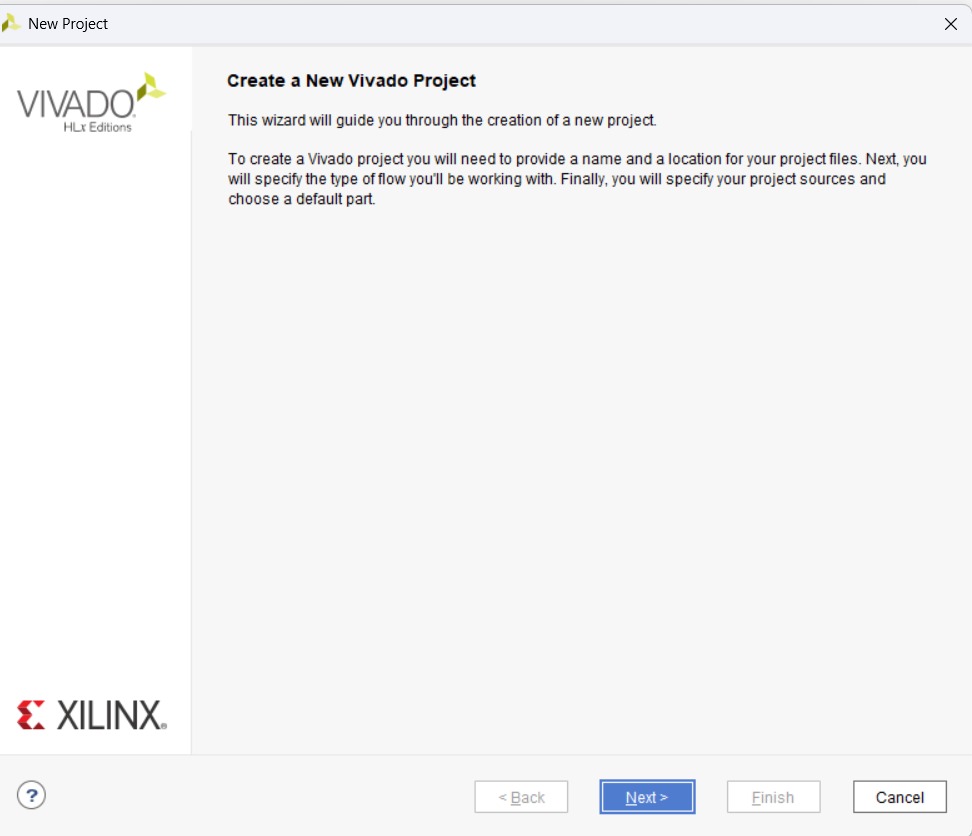
**Steps for implementing practical:**

1. **Open Xilinx ISE**
   * Launch the ISE Design Suite.
   * You’ll see the Project Navigator window.



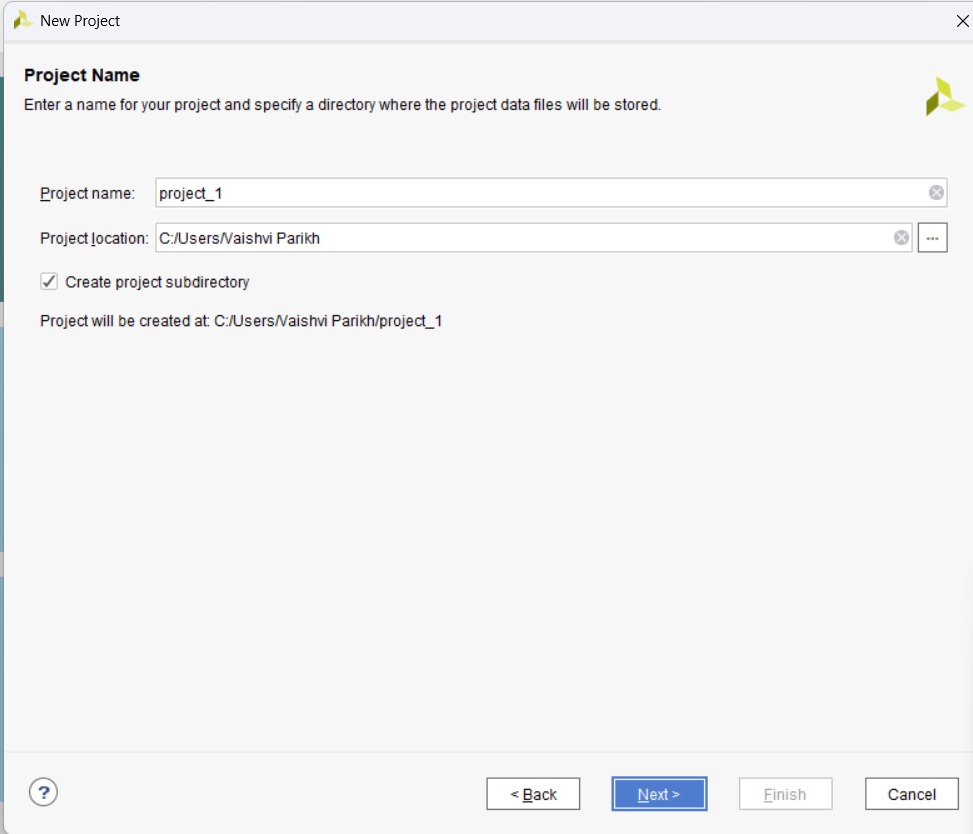
1. **Create a New Project**

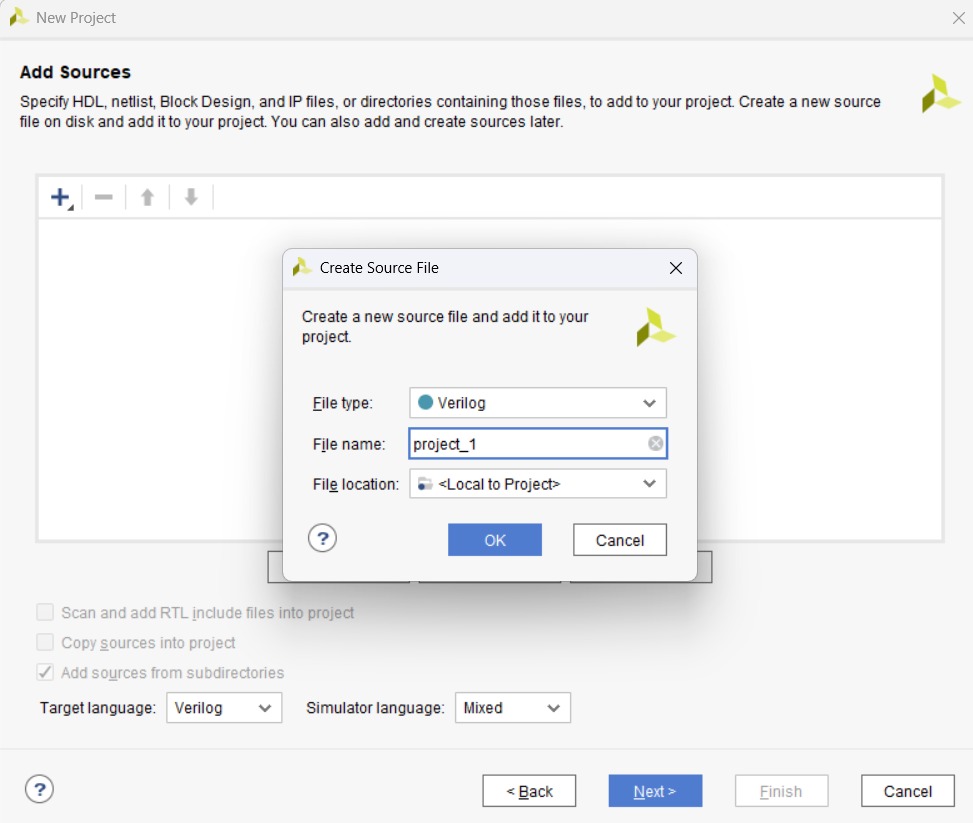
* Click File → New Project.

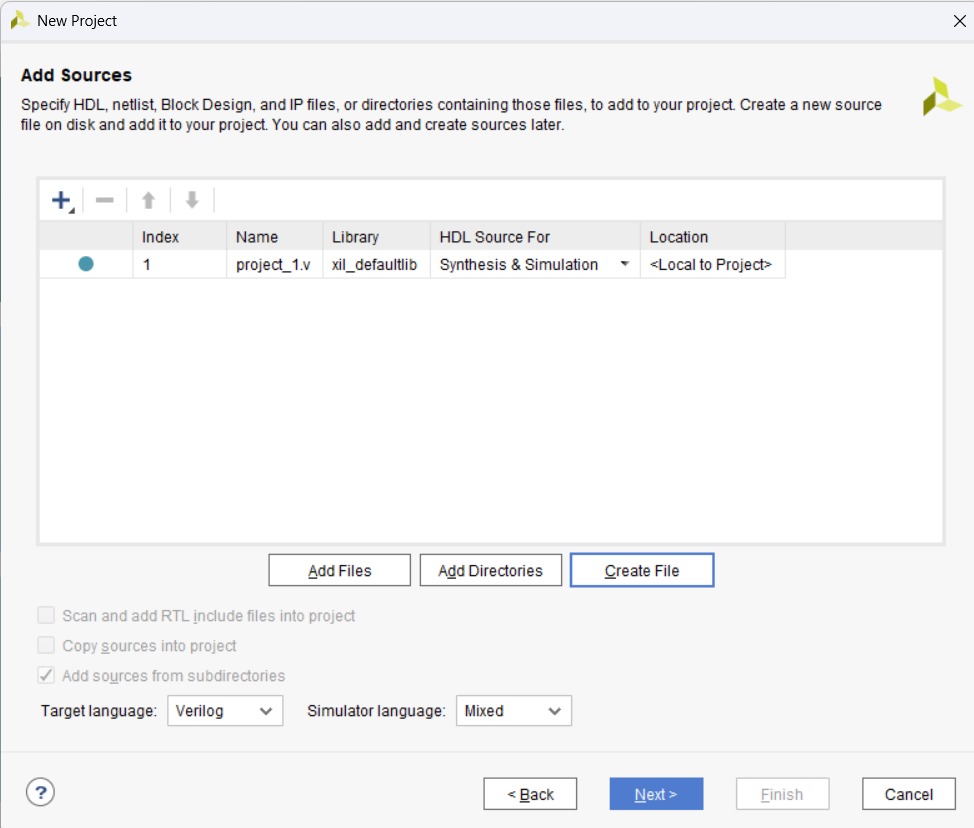


1. **Enter:**
   * Project Name (e.g., AND Gate)
   * Project Location
   * Top-Level Source Type = HDL (VHDL/Verilog).

Click Next.

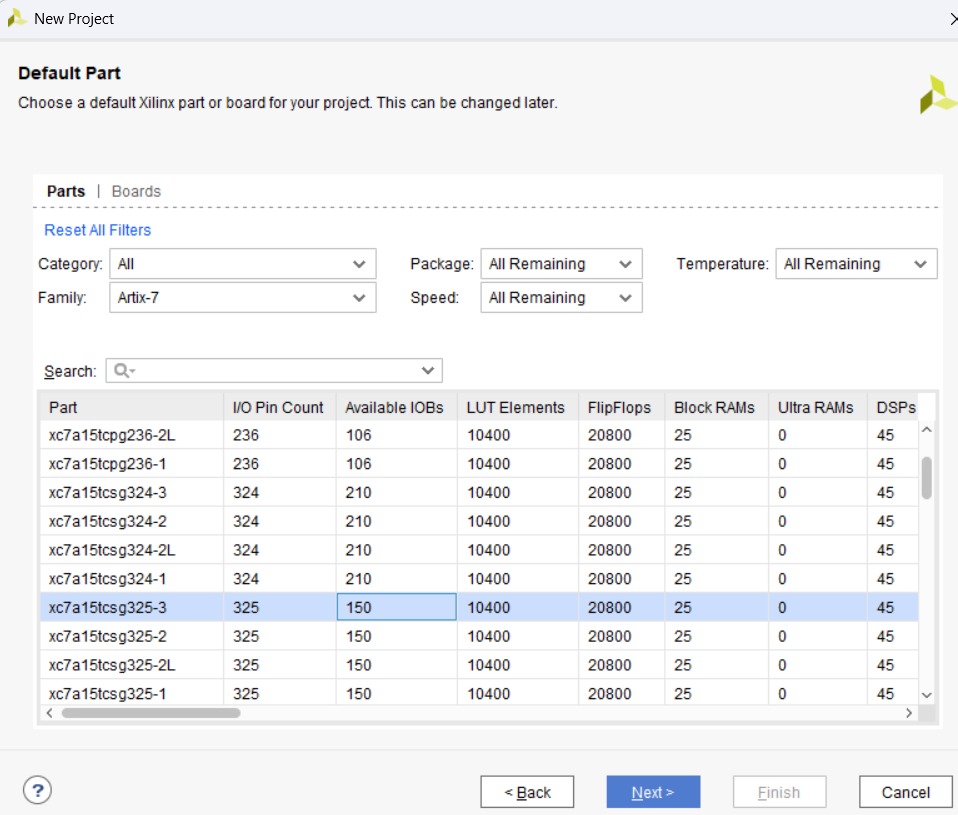






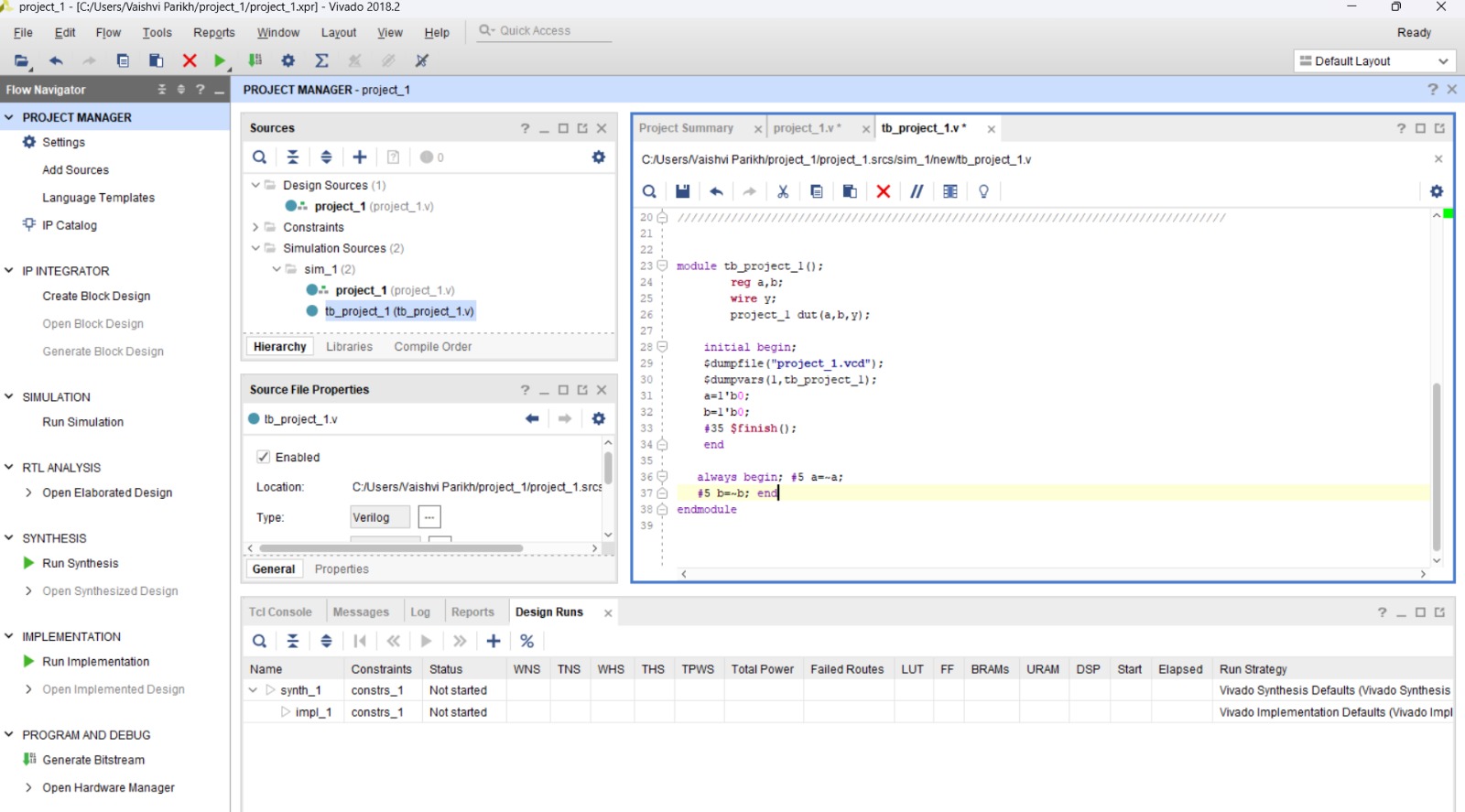
**Select Device**

* Choose the FPGA family (e.g., Spartan-3, Virtex-5, etc.).
* Example:
  + Family: Spartan3E
  + Device: xc3s500e
  + Package: fg320
  + Speed: -4
  + Preferred Language: Verilog/VHDL
* Click Next → Finish.



**Add Source Files**

* In the Sources Window, right-click → New Source.
* Select Verilog Module (or VHDL).
* Enter filename and ports (inputs/outputs).
* Finish → code editor opens.



**Conclusion:**

**Suggested Reference:**

https://www.xilinx.com/htmldocs/xilinx13\_3/ise\_tutorial\_ug695.pdf

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 4

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Implement all the basic Logic Gates and Boolean functions using different modeling styles in Verilog/ VHDL: (a) Structural modeling**

1. **Dataflow modeling**
2. **Behavioural modeling**

**Competency and Practical Skills:** Basic Digital Design

**Relevant CO:** CO5

**Objectives:** Understanding of different modeling styles in Verilog HDL.

**Equipment / Instruments:** Laptop or Computer with Xilinx / Tool

**Basic Theory:**

**Part – 1 Logic Gate Implementation**

|  |  |
| --- | --- |
| **Logic Gate** | **Truth Table** |
| **Inverter** | |  |  | | --- | --- | | **A** | **X** | | **0** | **1** | | **1** | **0** | |
| **OR Gate** | |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | **0** | **0** | **0** | | **0** | **1** | **1** | | **1** | **0** | **1** | | **1** | **1** | **1** | |
| **AND Gate** | |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | **0** | **0** | 0 | | **0** | **1** | **0** | | **1** | **0** | **0** | | **1** | **1** | **1** | |
| **NAND Gate** | |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | **0** | **0** | 1 | | **0** | **1** | **1** | | **1** | **0** | **1** | | **1** | **1** | **0** | |
| **NOR Gate** | |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | **0** | **0** | 1 | | **0** | **1** | **0** | | **1** | **0** | **0** | | **1** | **1** | **0** | |
| **XOR Gate** | |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | **0** | **0** | 0 | | **0** | **1** | **1** | | **1** | **0** | 1 | | **1** | **1** | 0 | |
| **XNOR Gate** | |  |  |  | | --- | --- | --- | | **A** | **B** | **Y** | | **0** | **0** | 1 | | **0** | **1** | 0 | | **1** | **0** | 0 | | **1** | **1** | **1** | |

**NOT/OR :**

|  |  |
| --- | --- |
| 1. **Structural Modeling** | `timescale 1ns / 1ps  module gate\_demo (  input A,  input B,  output X,  output Y  );  not N1 (X, A);  or OR1(Y, A, B);  endmodule |
| 1. **Dataflow Modeling** | `timescale 1ns / 1ps  module gate\_demo (  input A,  input B,  output X,  output Y  );  assign #5 X = ~A;  assign #2 Y = A | B;  endmodule |
| 1. **Behavioural Modeling** | `timescale 1ns / 1ps  module gate\_demo (  input A,  input B,  output reg X,  output reg Y  );  always @(A, B) begin  X = ~A;  Y = A | B;  end  endmodule |
| **Test Bench:** | `timescale 1ns / 1ps  module gate\_tb;  reg A, B;  wire X, Y;  gate\_demo uut (  .A(A),  .B(B),  .X(X),  .Y(Y)  );  initial begin  A = 0; B = 0;  #10 A = 0; B = 1;  #10 A = 1; B = 0;  #10 A = 1; B = 1;  #10 A = 0; B = 0;  #10 $finish;  end  endmodule |
| **Simulation waveforms:** |  |

**AND Gate:**

|  |  |
| --- | --- |
| **Verilog code** | **Dataflow modeling**  `timescale 1ns / 1ps  module and\_2\_1\_1(  input a, b,  output c  );  assign c = a & b;  endmodule  **Structural modeling**  `timescale 1ns / 1ps  module and\_2\_1\_1(  input a, b,  output c  );  and g1(c, a, b);  endmodule  **Behavioral modeling**  `timescale 1ns / 1ps  module and\_2\_1\_1(  input a, b,  output reg c  );  always @(a or b) begin  c = a & b;  end  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_and\_2\_1\_1();  reg a,b;  wire c;  and\_2\_1\_1 dut(a,b,c);    initial begin;  $dumpfile("and\_2\_1\_1.vcd");  $dumpvars(1,tb\_and\_2\_1\_1);  a=1'b0;  b=1'b0;  #35 $finish();  end    always begin;  #5 a=~a;  #5 b=~b;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**OR Gate:**

|  |  |
| --- | --- |
| **Verilog code** | **Dataflow modeling**  `timescale 1ns / 1ps  module or\_1(  input a,  input b,  output c  );  assign c = a | b;  endmodule  **-----------------------------------------------------------------------------------------------**  **Structural modeling using gate-level primitive**  `timescale 1ns / 1ps  module or\_1(  input a,  input b,  output c  );    or o1(c, a, b); //    endmodule  **Behavioral modeling**  `timescale 1ns / 1ps  module or\_1(  input a,  input b,  output reg c  );    always @(a or b) begin  c = a | b;  end    endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_or\_1();  reg a,b;  wire c;  or\_1 dut(a,b,c);    initial begin;  $dumpfile("or\_1.vcd");  $dumpvars(1,tb\_or\_1);  a=1'b0;  b=1'b0;  #35 $finish();  end  always begin;  #5 a=~a;  #5 b=~b;  end    endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**NAND Gate:**

|  |  |
| --- | --- |
| **Verilog code** | **Dataflow modeling**  `timescale 1ns / 1ps  module nand\_1(  input a, b,  output c  );    assign c = ~(a & b);    endmodule  **Structural modeling using gate-level primitive**  `timescale 1ns / 1ps  module nand\_1(  input a, b,  output c  );    nand na1(c, a, b);    endmodul  **Behavioral modeling**  `timescale 1ns / 1ps  module nand\_1(  input a, b,  output reg c  );    always @(a or b) begin  c = ~(a & b);  end    endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_nand\_1();  reg a,b;  wire c;  nand\_1 dut(a,b,c);  initial begin;  $dumpfile("nand\_1.vcd");  $dumpvars(1,tb\_nand\_1);  a=1'b0;  b=1'b0;  #35 $finish();  end  always begin;  #5 a=~a;  #5 b=~b;  end    endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| Simulation waveforms: |  |

**NOR Gate:**

|  |  |
| --- | --- |
| **Verilog code :** | **Dataflow modeling**  `timescale 1ns / 1ps  module nor\_1(  input a, b,  output c  );    assign c = ~(a | b); //    endmodule  **Structural modeling using gate-level primitive**  `timescale 1ns / 1ps  module nor\_1(  input a, b,  output c  );    nor no1(c, a, b);    endmodule  **Behavioral modeling**  `timescale 1ns / 1ps  module nor\_1(  input a, b,  output reg c  );    always @(a or b) begin  c = ~(a | b);  end    endmodule |
| **Verilog code (TB):** | `timescale 1ns / 1ps  module tb\_nor\_1();  reg a,b;  wire c;  nor\_1 dut(a,b,c);    initial begin;  $dumpfile("nor\_1.vcd");  $dumpvars(1, tb\_nor\_1);  a=1'b0;  b=1'b0;  #35 $finish();  end  always begin;  #5 a=~a;  #5 b=~b;  end  endmodule |
| RTL Schematic: |  |
| Synthesis Schematic: |  |
| Simulation waveforms: |  |

**XOR Gate:**

|  |  |
| --- | --- |
| **Verilog code** | **Dataflow modeling**  `timescale 1ns / 1ps  module xor\_2(  input a, b,  output c  );    assign c = a ^ b;    endmodule  **Structural modeling using gate-level primitive**  `timescale 1ns / 1ps  module xor\_2(  input a, b,  output c  );    xor xo1(c, a, b); //    endmodule  `timescale 1ns / 1ps  module xor\_2(  input a, b,  output reg c  );    always @(a or b) begin  c = a ^ b; // Behavioral modeling  end    endmodule |
| **Verilog code (TB):** | `timescale 1ns / 1ps  module tb\_xor\_2();  reg a,b;  wire c;  xor\_2 dut(a,b,c);  initial begin;    $dumpfile("xor\_2.vcd");  $dumpvars(1,tb\_xor\_2);  a=1'b 0;  b=1'b 0;  #35 $finish();  end  always begin;  #5 a=~a;  #5 b=~b;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**XNOR Gate:**

|  |  |
| --- | --- |
| **Verilog code** | **Dataflow modeling**  `timescale 1ns / 1ps  module xnor\_1(  input a, b,  output c  );    assign c = ~(a ^ b);    endmodule  **Structural modeling using gate-level primitive**  `timescale 1ns / 1ps  module xnor\_1(  input a, b,  output c  );    xnor xno1(c, a, b);    endmodule  **Behavioral modelling**  `timescale 1ns / 1ps  module xnor\_1(  input a, b,  output reg c  );    always @(a or b) begin  c = ~(a ^ b);  end    endmodule |
| **Verilog code (TB)** | module tb\_xnor\_1();  reg a,b;  wire c;  xnor\_1 dut(a,b,c);  initial begin;  $dumpfile("xnor\_1.vcd");  $dumpvars(1,tb\_xnor\_1);  a=1'b 0;  b=1'b 0;  #35 $finish();  end    always begin;  #5 a=~a;  #5 b=~b;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms**: |  |

**Part – 2 Boolean Function Implementation**

* 1. 𝒇 = 𝒙′𝒚′ + 𝒙𝒚
  2. 𝒈 = 𝒙′𝒚𝒛 + 𝒙𝒚𝒛′ + 𝒙′𝒚′𝒛′ + **𝒙𝒚z**

|  |  |
| --- | --- |
| **Verilog code** | timescale 1ns / 1ps  module fun\_2(  input x,y,z,  output f,g);  assign f=(~x&~y) | (x&y);  assign g= (~x&y&z)|(x&y&~z)|(~x&~y&~z)|(x&y&z);    endmodule  timescale 1ns / 1ps  module fun\_2(  input x,y,z,  output f,g);  wire a,b,c,d,e;  wire p,q,r,s;  not (a,x);  not (b,y);  not (c,z);  and (d,a,b);  and (e,x,y);  or (f,d,e);  and (p,a,y,z);  and (q,x,y,c);  and (r,a,b,c);  and (s,x,y,z);  or (g,p,q,r,s);  endmodule  `timescale 1ns / 1ps  module fun\_2(  input x,y,z,  output reg f,g);  always@(x,y,z) begin  f= (~x & ~y) | (x & y);  g=(~x&y&z)|(x&y&~z)|(~x&~y&~z)|(x&y&z);  end  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_fun\_2();  reg x,y,z;  wire f,g;  fun\_2 uut(x,y,z,f,g);  initial begin;  $dumpfile("fun\_2.vcd");  $dumpvars(1,tb\_fun\_2);  x=1'b0;  y=1'b0;  z=1'b0;  #35 $finish();  end    always begin;  #5 x=~x;  #5 y=~y;  #5 z=~z;  end  endmodule  testbench of both the functions |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**Conclusion:**

**Suggested Reference: ---**

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 5

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Design Adder & Subtractor using Verilog / VHDL. (a) Half Adder (Using structural and dataflow modeling) (b) Full Adder using Half Adder (Structure Method).**

1. **Full Adder (Using dataflow and behavioural modeling)**
2. **Half Subtractor**
3. **Full Subtractor**

**Competency and Practical Skills:** Basic Digital Design

**Relevant CO:** CO5

**Objectives:** Designing of adders and subtractor using different modeling styles in Verilog HDL.

**Equipment / Instruments:** Laptop or Computer with Xilinx / Tool

**Basic Theory:**

|  |  |  |
| --- | --- | --- |
| **Logic Gate** |  | **Truth Table** |
| **Half Adder** |  | |  |  |  |  | | --- | --- | --- | --- | | **A** | **B** | **S** | **C** | | **0** | **0** | **0** | **0** | | **0** | **1** | **1** | **0** | | **1** | **0** | **1** | **0** | | **1** | **1** | **0** | **1** | |
| **Full Adder**    **A**  **B**  **Sum**  **C**  **in**  **T**  **1**  **T**  **2**  **T**  **3** | **Cout** | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **A** | **B** | **Cin** | **Sum** | **Cout** | | **0** | **0** | **0** | **0** | **0** | | **0** | **0** | **1** | **1** | **0** | | **0** | **1** | **0** | **1** | **0** | | **0** | **1** | **1** | **0** | **1** | | **1** | **0** | **0** | **1** | **0** | | **1** | **0** | **1** | **0** | **1** | | **1** | **1** | **0** | **0** | **1** | | **1** | **1** | **1** | **1** | **1** | |
| **Half Subtractor** |  | |  |  |  |  | | --- | --- | --- | --- | | **A** | **B** | **D** | **Br** | | **0** | **0** | **0** | **0** | | **0** | **1** | **1** | **1** | | **1** | **0** | **1** | **0** | | **1** | **1** | **00** | **0** | |
| **Full Subtractor** | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **A** | **B** | **Cin** | **Diff** | **Bout** | | **0** | **0** | **0** | **0** | **0** | | **0** | **0** | **1** | **1** | **1** | | **0** | **1** | **0** | **1** | **1** | | **0** | **1** | **1** | **0** | **1** | | **1** | **0** | **0** | **1** | **0** | | **1** | **0** | **1** | **0** | **0** | | **1** | **1** | **0** | **0** | **0** | | **1** | **1** | **1** | **1** | **1** | |

**RTL Code:**

|  |  |
| --- | --- |
| **Verilog**  **code** | **Half Adder (Structural modelling):**      module HA (A, B, S, C);  input A, B;  output S, C;  xor x1(S, A, B);  and n1(C, A, B);  endmodule    **Half Adder (Dataflow modeling):**  module HA (A, B, S, C);  input A, B;  output S, C;    assign S = A ^ B;  assign C =A & B;  endmodule |
| **Verilog code (TB)** | module tb\_ha\_1();  reg a,b;  wire h,c;  ha\_1 uut (a,b,h,c);  initial begin  $monitor("Time=%0t | A=%b B=%b | Sum=%b Carry=%b",  $time, a, b, h, c);  a=1'b0;  b=1'b0;  #35 $finish();  end  always begin;  #5 a=~a;  #10 b=~b;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**(b) 1-bit Full Adder Implementation using Half Adders (structural modeling):**

**A**

**C**

**B**

**A**

**B**

**B**

**S**

**S**

**C**

**HA**

**1**

**HA**

**2**

**A**

**C**

**in**

**Sum**

**C**

**out**

**S**

**1**

**C**

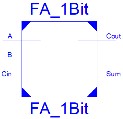
**2**

**C**

**1**

|  |  |
| --- | --- |
| **Verilog code** | **Half Adder:**  `timescale 1ns / 1ps  module HA (  input A,  input B,  output S,  output C  );  xor x1 (S, A, B);  and n1 (C, A, B);  endmodule  **Full Adder (1-Bit)**  module FA\_1Bit (  input A,  input B,  input Cin,  output Sum,  output Cout  );  wire S1, C1, C2;  HA HA1 (A, B, S1, C1);  HA HA2 (S1, Cin, Sum, C2);  or OR1 (Cout, C1, C2);  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_FA\_1Bit;  reg A, B, Cin;  wire Sum, Cout;  FA\_1Bit uut (  .A(A),  .B(B),  .Cin(Cin),  .Sum(Sum),  .Cout(Cout)  );  initial begin  $monitor("Time=%0t | A=%b B=%b Cin=%b | Sum=%b Cout=%b",  $time, A, B, Cin, Sum, Cout);  A=0; B=0; Cin=0; #10;  A=0; B=0; Cin=1; #10;  A=0; B=1; Cin=0; #10;  A=0; B=1; Cin=1; #10;  A=1; B=0; Cin=0; #10;  A=1; B=0; Cin=1; #10;  A=1; B=1; Cin=0; #10;  A=1; B=1; Cin=1; #10;  $stop;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**RTL Schematic of 1-bit FA Implementation:**





**(c)**  **1-bit Full Adder (Dataflow and behavioural modeling):**

|  |  |
| --- | --- |
| **Verilog code** | **Half Adder Module**  module HA (A, B, S, C); // HA - Half Adder  input A, B;  output S, C;  xor x1(S, A, B);  and n1(C, A, B);  endmodule  module FA\_1Bit( A, B, Cin, Sum, Cout);  input A, B, Cin;  output Sum, Cout;  wire S1, C1, C2; //Module instantiation  HA HA1(A, B, S1, C1);  HA HA2(S1, Cin, Sum, C2);  or OR1(Cout, C1, C2);  endmodule |
| **Verilog code (TB)** | `timescale 1ns/1ps // Simulation time unit / precision  module tb\_FA\_1Bit;  reg A, B, Cin;  wire Sum, Cout;  FA\_1Bit uut (  .A(A),  .B(B),  .Cin(Cin),  .Sum(Sum),  .Cout(Cout)  );  initial begin  $monitor("Time=%0t | A=%b B=%b Cin=%b | Sum=%b Cout=%b",  $time, A, B, Cin, Sum, Cout);  A = 0; B = 0; Cin = 0; #10;  A = 0; B = 0; Cin = 1; #10;  A = 0; B = 1; Cin = 0; #10;  A = 0; B = 1; Cin = 1; #10;  A = 1; B = 0; Cin = 0; #10;  A = 1; B = 0; Cin = 1; #10;  A = 1; B = 1; Cin = 0; #10;  A = 1; B = 1; Cin = 1; #10;  $stop;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

1. **Half Subtractor**

|  |  |
| --- | --- |
| **Verilog code** | `timescale 1ns / 1ps  module half\_subtractor(  input a,b,  output D,B  );  assign D = a ^ b;  assign B = ~a & b;  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_top(  );  reg a, b;  wire D, B;    half\_subtractor hs(a, b, D, B);    initial begin  $monitor("At time %0t: a=%b b=%b, difference=%b, borrow=%b",$time, a,b,D,B);  a = 0; b = 0;  #100;  a = 0; b = 1;  #100;  a = 1; b = 0;  #100;  a = 1; b = 1;     end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

1. **Full Subtractor**

|  |  |
| --- | --- |
| **Verilog code :** | `timescale 1ns / 1ps  module full\_subtractor(  input a,b,Bin,  output D,Bout  );  assign D = a ^ b ^ Bin;  assign Bout = (~a & b) | (~(a ^ b) & Bin);  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_top(  );  reg a, b, Bin;  wire D, Bout;    full\_subtractor fs(a, b, Bin, D, Bout);    initial begin  $monitor("At time %0t: a=%b b=%b, Bin=%b, difference=%b, borrow=%b",$time, a,b,Bin,D,Bout);  a = 0; b = 0; Bin = 0; #1;  a = 0; b = 0; Bin = 1; #1;  a = 0; b = 1; Bin = 0; #1;  a = 0; b = 1; Bin = 1; #1;  a = 1; b = 0; Bin = 0; #1;  a = 1; b = 0; Bin = 1; #1;  a = 1; b = 1; Bin = 0; #1;  a = 1; b = 1; Bin = 1;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms** |  |

**Conclusion:**

**Suggested Reference:**

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 6

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Design Binary to Gray & Gray to Binary encoder using Verilog/VHDL.**

**Competency and Practical Skills:** Basic Digital Design

**Relevant CO:** CO5

**Objectives:** Designing of code converters

**Equipment / Instruments:** Laptop or Computer with Xilinx / Tool

**Basic Theory:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Decimal Equivalent** | **Binary Code** | | | |  | **Gray code** | |  |
|  | **B3** | **B2** | **B1** | **B0** | **G3** | **G2** | **G1** | **G0** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **1** | **0** | **0** | **0** | **1** | **0** | **0** | **0** | **1** |
| **2** | **0** | **0** | **1** | **0** | **0** | **0** | **1** | **1** |
| **3** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** |
| **4** | **0** | **1** | **0** | **0** | **0** | **1** | **1** | **0** |
| **5** | **0** | **1** | **0** | **1** | **0** | **1** | **1** | **1** |
| **6** | **0** | **1** | **1** | **0** | **0** | **1** | **0** | **1** |
| **7** | **0** | **1** | **1** | **1** | **0** | **1** | **0** | **0** |
| **8** | **1** | **0** | **0** | **0** | **1** | **1** | **0** | **0** |
| **9** | **1** | **0** | **0** | **1** | **1** | **1** | **0** | **1** |
| **10** | **1** | **0** | **1** | **0** | **1** | **1** | **1** | **1** |
| **11** | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **0** |
| **12** | **1** | **1** | **0** | **0** | **1** | **0** | **1** | **0** |
| **13** | **1** | **1** | **0** | **1** | **1** | **0** | **1** | **1** |
| **14** | **1** | **1** | **1** | **0** | **1** | **0** | **0** | **1** |
| **15** | **1** | **1** | **1** | **1** | **1** | **0** | **0** | **0** |

|  |  |  |
| --- | --- | --- |
| **Binary to Gray Converter** | | **Gray to Binary Converter** |
|  |  |  |
| **B0**  **B1**  **B2**  **B3** | **G3**  **G2**  **G1**  **G0** | Virtual Labs |

**Binary to Gray Converter:**

|  |  |
| --- | --- |
| **Verilog code** | module bin\_to\_gray(B, G);  input [3:0] B;  output [3:0] G;  assign G[3] = B[3];  assign G[2] = B[3] ^ B[2];  assign G[1] = B[2] ^ B[1];  assign G[0] = B[1] ^ B[0];    endmodule |
| **Verilog code (TB)** | module tb\_bin\_to\_gray;  reg [3:0] B;  wire [3:0] G;  bin\_g bg1 (  .B(B),  .G(G)  );  initial begin  $dumpfile("bin\_to\_gray.vcd");  $dumpvars(1, tb\_bin\_to\_gray);  B = 4'b0000;  #100 $finish;  end  always begin  #5 B = B + 1;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**Gray to Binary Encoder**

|  |  |
| --- | --- |
| **Verilog code** | `timescale 1ns / 1ps  module Gray\_to\_Binary #(parameter N = 4)(  input [N-1:0] gray, // Gray code input  output [N-1:0] binary // Binary output  );  assign binary[N-1] = gray[N-1]; // MSB same as Gray  genvar i;  generate  for(i = N-2; i >= 0; i = i - 1) begin  assign binary[i] = binary[i+1] ^ gray[i];  end  endgenerate  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_Gray\_to\_Binary(    );  parameter N = 4;  reg [N-1:0] gray;  wire [N-1:0] binary;    // Instantiate DUT  Gray\_to\_Binary #(N) uut (  .gray(gray),  .binary(binary)  );  initial begin  // Monitor values  $monitor("Time=%0t | Gray=%b | Binary=%b", $time, gray, binary);    // Apply test values  gray = 4'b0000; #10;  gray = 4'b0001; #10;  gray = 4'b0011; #10;  gray = 4'b0010; #10;  gray = 4'b0110; #10;  gray = 4'b0111; #10;  gray = 4'b0101; #10;  gray = 4'b0100; #10;  gray = 4'b1100; #10;  gray = 4'b1101; #10;  gray = 4'b1111; #10;  gray = 4'b1110; #10;  gray = 4'b1010; #10;  gray = 4'b1011; #10;  gray = 4'b1001; #10;  gray = 4'b1000; #10;    $stop;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**Conclusion:**

**Suggested Reference:**

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 7

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Design Multiplexer and Demultiplexer using Verilog / VHDL.**

1. **2:1 Mux (Dataflow and Behavioural modeling)**
2. **4:1 Mux (Structural and Dataflow modeling)**
3. **8:1 Mux (Using 4:1 and 2:1 Mux : Structural modeling)**
4. **16:1 Mux (Using Behavioural Modeling & 4:1 Mux : Structural modeling) e. 1:8 Demux**

**Competency and Practical Skills:** Basic Digital Design

**Relevant CO:** CO5

**Objectives:** Designing of Multiplexers and Demultiplexers

**Equipment / Instruments:** Laptop or Computer with Xilinx / Tool

**Basic Theory:**

**MUX:**

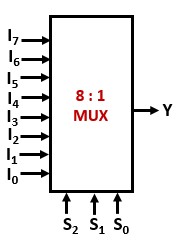
A digital logic circuit which is capable of accepting several inputs and generating a single output is known as multiplexer or MUX.

**(a) 2:1 Mux (Dataflow and Behavioural modeling)**

|  |  |
| --- | --- |
| **Verilog code** | **dataflow Modelling**  module mux2\_to\_1(I, S, Y);  input [1 : 0] I;  input S;  output Y;  assign Y = S ? I[1] : I[0];  endmodule  **behavioural Modelling**  module mux2\_to\_1(I, S, Y);  input [1 : 0] I;  input S;  output reg Y;    always @(I, S) begin  if (S == 0)  Y <= I[0];  else  Y <= I[1];  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_mux2\_1;  reg [1:0] I;  reg S;  wire Y;    mux2\_1 uut (  .I(I),  .S(S),  .Y(Y)  );    initial begin  $monitor("Time=%0t |  I=%b | S=%b | Y=%b",  $time, I, S, Y);    I = 2'b00; S = 0; #10;  I = 2'b01; S = 0; #10;  I = 2'b10; S = 0; #10;  I = 2'b11; S = 0; #10;  I = 2'b00; S = 1; #10;  I = 2'b01; S = 1; #10;  I = 2'b10; S = 1; #10;  I = 2'b11; S = 1; #10;    #10 $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

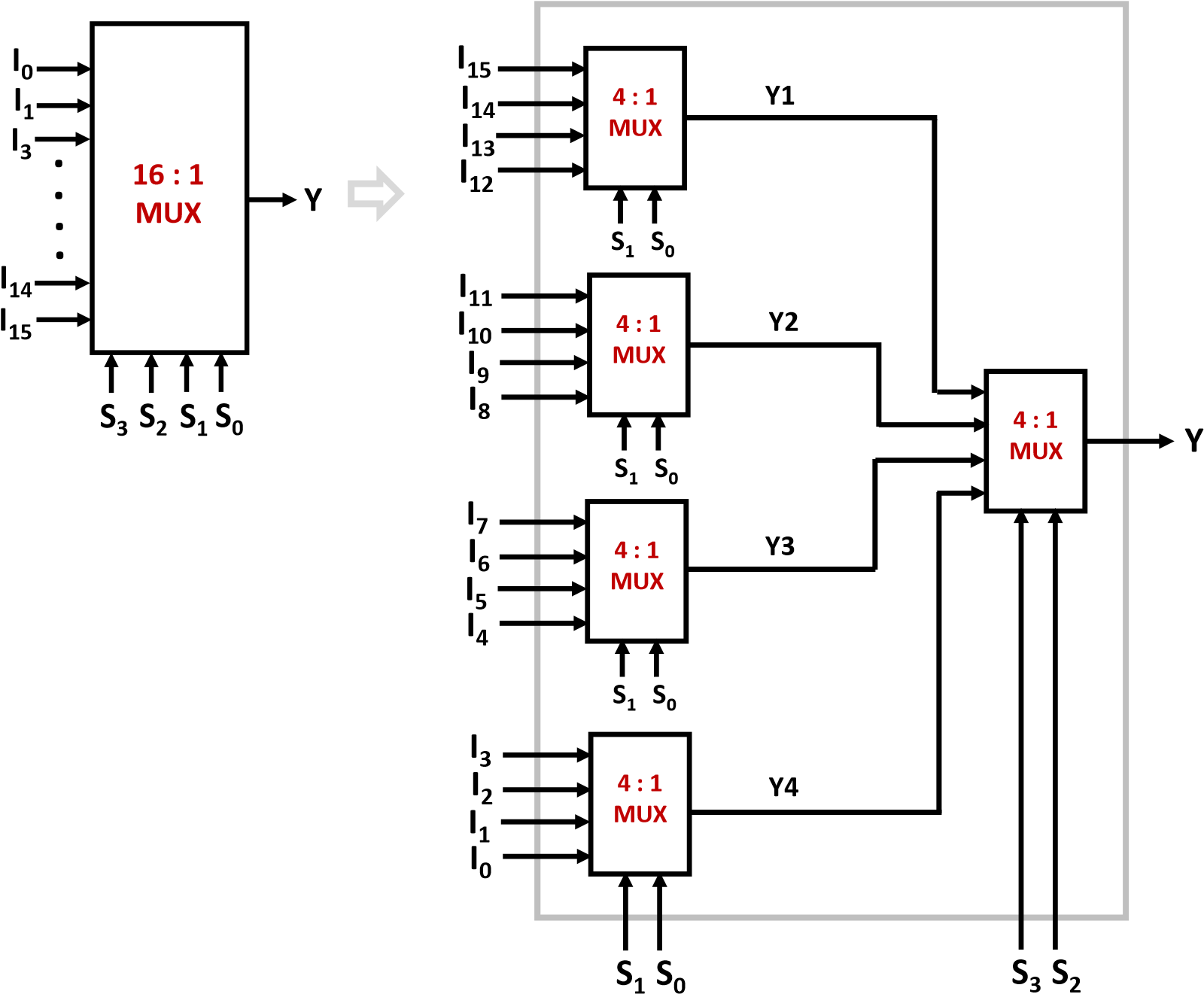
**(b) 4:1 Mux (Structural and Dataflow modeling)**

|  |  |
| --- | --- |
| **Verilog code** | **structural modelling**  module mux4\_to\_1(I, S, Y);  input [3 : 0] I;  input [1: 0] S;  output Y;  wire [1: 0] t;  mux2\_to\_1 mux1(I[1:0], S[0], t[0]);  mux2\_to\_1 mux1(I[3:2], S[0], t[1]);  mux2\_to\_1 mux1(t, S[1], Y);  endmodule  **dataflow modeling**  module mux4\_1(I, S, Y);  input [3:0] I;  input [1:0] S;  output Y;  assign Y = I[S];  endmodule  module mux2\_to\_1(I, S, Y);  input [1 : 0] I;  input S;  output Y;  assign Y = S ? I[1] : I[0];    endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_mux4\_1;  reg [3:0] I;  reg [1:0] S;  wire Y;  mux4\_1 uut (.I(I), .S(S), .Y(Y));  initial begin  // Case 1: Select each input when only one is HIGH  I = 4'b0001; S = 2'b00; #10;  I = 4'b0010; S = 2'b01; #10;  I = 4'b0100; S = 2'b10; #10;  I = 4'b1000; S = 2'b11; #10;    // Case 2: Random input patterns  I = 4'b1011; S = 2'b00; #10;  I = 4'b1011; S = 2'b01; #10;  I = 4'b1011; S = 2'b10; #10;  I = 4'b1011; S = 2'b11; #10;    // Case 3: All inputs HIGH  I = 4'b1111; S = 2'b00; #10;  I = 4'b1111; S = 2'b01; #10;  I = 4'b1111; S = 2'b10; #10;  I = 4'b1111; S = 2'b11; #10;    $finish;  end    endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

****©**8:1Mux(Using 4:1 and 2:1 Mux : Structural modelling)**

|  |  |
| --- | --- |
| **Verilog code** | `timescale 1ns / 1ps  module mux8to1(  input [7:0] d, // 8 data inputs  input [2:0] sel, // 3-bit select  output y // output  );  wire y0, y1; // outputs of 4:1 muxes    // First 4:1 MUX (for inputs d[3:0])  mux4to1 m1 (  .d(d[3:0]),  .sel(sel[1:0]),  .y(y0)  );    // Second 4:1 MUX (for inputs d[7:4])  mux4to1 m2 (  .d(d[7:4]),  .sel(sel[1:0]),  .y(y1)  );    // Final 2:1 MUX to choose between y0 and y1  mux2to1 m3 (  .d0(y0),  .d1(y1),  .sel(sel[2]),  .y(y)  );    endmodule      // 4:1 MUX  module mux4to1 (  input [3:0] d,  input [1:0] sel,  output y  );  assign y = (sel == 2'b00) ? d[0] :  (sel == 2'b01) ? d[1] :  (sel == 2'b10) ? d[2] :  d[3];  endmodule      // 2:1 MUX  module mux2to1 (  input d0, d1,  input sel,  output y  );  assign y = (sel) ? d1 : d0;    endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_mux8to1(  );  reg [7:0] d;  reg [2:0] sel;  wire y;    // Instantiate DUT (Device Under Test)  mux8to1 uut (  .d(d),  .sel(sel),  .y(y)  );    initial begin  // Apply test cases  $monitor("Time=%0t | sel=%b | d=%b | y=%b", $time, sel, d, y);    d = 8'b10101010; // Test pattern    sel = 3'b000; #10; // Expect d[0] = 0  sel = 3'b001; #10; // Expect d[1] = 1  sel = 3'b010; #10; // Expect d[2] = 0  sel = 3'b011; #10; // Expect d[3] = 1  sel = 3'b100; #10; // Expect d[4] = 0  sel = 3'b101; #10; // Expect d[5] = 1  sel = 3'b110; #10; // Expect d[6] = 0  sel = 3'b111; #10; // Expect d[7] = 1    $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

1. **16:1 Mux (Using Behavioural Modeling & 4:1 Mux : Structural modeling)**



|  |  |
| --- | --- |
| **Verilog code** | `timescale 1ns / 1ps  module mux16to1(  input [15:0] d, // 16 inputs  input [3:0] sel, // 4-bit select  output reg y // Output  );  wire [3:0] y4; // outputs of 4:1 muxes    // Four 4:1 MUXes (structural instantiation)  mux4to1 m0 (.d(d[3:0]), .sel(sel[1:0]), .y(y4[0]));  mux4to1 m1 (.d(d[7:4]), .sel(sel[1:0]), .y(y4[1]));  mux4to1 m2 (.d(d[11:8]), .sel(sel[1:0]), .y(y4[2]));  mux4to1 m3 (.d(d[15:12]), .sel(sel[1:0]), .y(y4[3]));    // Final selection using behavioral modeling  always @(\*) begin  case(sel[3:2])  2'b00: y = y4[0];  2'b01: y = y4[1];  2'b10: y = y4[2];  2'b11: y = y4[3];  endcase  end  endmodule    // 4:1 Multiplexer (Dataflow / Structural submodule)    module mux4to1 (  input [3:0] d,  input [1:0] sel,  output y  );  assign y = (sel == 2'b00) ? d[0] :  (sel == 2'b01) ? d[1] :  (sel == 2'b10) ? d[2] :  d[3];  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_mux16to1(  );  reg [15:0] d;  reg [3:0] sel;  wire y;    // Instantiate DUT  mux16to1 uut (  .d(d),  .sel(sel),  .y(y)  );    initial begin  $monitor("Time=%0t | sel=%b | y=%b | d=%h", $time, sel, y, d);    d = 16'hA55A; // 1010 0101 0101 1010 (test pattern)    // Sweep select lines  sel = 4'b0000; #10; // Expect d[0]  sel = 4'b0001; #10; // Expect d[1]  sel = 4'b0010; #10; // Expect d[2]  sel = 4'b0011; #10; // Expect d[3]  sel = 4'b0100; #10; // Expect d[4]  sel = 4'b0101; #10; // Expect d[5]  sel = 4'b0110; #10; // Expect d[6]  sel = 4'b0111; #10; // Expect d[7]  sel = 4'b1000; #10; // Expect d[8]  sel = 4'b1001; #10; // Expect d[9]  sel = 4'b1010; #10; // Expect d[10]  sel = 4'b1011; #10; // Expect d[11]  sel = 4'b1100; #10; // Expect d[12]  sel = 4'b1101; #10; // Expect d[13]  sel = 4'b1110; #10; // Expect d[14]  sel = 4'b1111; #10; // Expect d[15]    $finish;  end    endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

1. **1:8 DEMUX**

**DEMUX:** A digital combinational circuit which takes one input line and routs it to one of the multiple output lines depending on the status of the select lines is known as demultiplexer or

DEMUX.

**Y**

**6**

**Y**

**7**

**S**

**1**

**I**

**1**

**:**

**8**

**DEMUX**

**S**

**0**

**Y**

**5**

**Y**

**4**

**Y**

**2**

**Y**

**3**

**Y**

**1**

**Y**

**0**

**S**

**2**

|  |  |
| --- | --- |
| **Verilog code** | `timescale 1ns / 1ps  module demux1to8(  input din, // single input  input [2:0] sel, // 3-bit select  output reg [7:0] y // 8 outputs  );  always @(\*) begin  y = 8'b00000000; // default all outputs 0  case(sel)  3'b000: y[0] = din;  3'b001: y[1] = din;  3'b010: y[2] = din;  3'b011: y[3] = din;  3'b100: y[4] = din;  3'b101: y[5] = din;  3'b110: y[6] = din;  3'b111: y[7] = din;  endcase  end  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_demux1to8(  );  reg din;  reg [2:0] sel;  wire [7:0] y;    // Instantiate DUT  demux1to8 uut (  .din(din),  .sel(sel),  .y(y)  );    initial begin  $monitor("Time=%0t | din=%b | sel=%b | y=%b", $time, din, sel, y);    din = 1; // drive input high    sel = 3'b000; #10; // Expect y = 00000001  sel = 3'b001; #10; // Expect y = 00000010  sel = 3'b010; #10; // Expect y = 00000100  sel = 3'b011; #10; // Expect y = 00001000  sel = 3'b100; #10; // Expect y = 00010000  sel = 3'b101; #10; // Expect y = 00100000  sel = 3'b110; #10; // Expect y = 01000000  sel = 3'b111; #10; // Expect y = 10000000    $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**Conclusion:**

**Suggested Reference:**

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 8

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Design 2:4, 3:8, 4:16 Decoders using Verilog/VHDL.**

**Competency and Practical Skills:**

**Relevant CO:**

**Objectives:**

**Equipment / Instruments:** Laptop or Computer with Xilinx / Tool

**Basic Theory:** Decoder is a combinational circuit that has '*n*' input lines and maximum of 2*n*output lines. Depending on the code presented by input lines, one of the output lines will be active high, when the decoder is enabled. It reveals a decoder detects a particular code presented at input.

**(a) 2-to-4 Decoder :**

|  |  |
| --- | --- |
| **Verilog code** | module decoder2\_4( input [1:0] A,  output [3:0] Y  );  assign Y[0] = ~A[1] & ~A[0];  assign Y[1] = ~A[1] & A[0];  assign Y[2] = A[1] & ~A[0];  assign Y[3] = A[1] & A[0];  endmodule |
| **Verilog code (TB)** | module tb\_decoder2\_4;  reg [1:0] A;  wire [3:0] Y;  decoder2\_4 dut (  .A(A),  .Y(Y)  );  initial begin  $dumpfile("decoder2\_4.vcd");  $dumpvars(1, tb\_decoder2\_4);  A = 2'b00; #10;  A = 2'b01; #10;  A = 2'b10; #10;  A = 2'b11; #10;  $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

1. **3-to-8 Decoder (Behavioural Modeling):**

|  |  |
| --- | --- |
| **Verilog code** | module decoder3\_8( input [2:0] A,  output [7:0] Y  );  assign Y[0] = (~A[2] & ~A[1] & ~A[0]);  assign Y[1] = (~A[2] & ~A[1] & A[0]);  assign Y[2] = (~A[2] & A[1] & ~A[0]);  assign Y[3] = (~A[2] & A[1] & A[0]);  assign Y[4] = ( A[2] & ~A[1] & ~A[0]);  assign Y[5] = ( A[2] & ~A[1] & A[0]);  assign Y[6] = ( A[2] & A[1] & ~A[0]);  assign Y[7] = ( A[2] & A[1] & A[0]);  endmodule |
| **Verilog code (TB)** | module tb\_decoder3\_8;  reg [2:0] A;  wire [7:0] Y;  decoder3\_8 dut (  .A(A),  .Y(Y)  );  initial begin  $dumpfile("decoder3\_8.vcd");  $dumpvars(0, tb\_decoder3\_8);  A = 3'b000; #10;  A = 3'b001; #10;  A = 3'b010; #10;  A = 3'b011; #10;  A = 3'b100; #10;  A = 3'b101; #10;  A = 3'b110; #10;  A = 3'b111; #10;  $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**(c) 4-to-16 Decoder (Structural Modeling using 3-to-8 Decoder):**

|  |  |
| --- | --- |
| **Verilog code** | `timescale 1ns / 1ps  module decoder4to16(  input [3:0] in,  input en,  output [15:0] out  );  wire [7:0] d0, d1;  wire en0, en1;    assign en0 = en & ~in[3]; // Enable lower half when MSB=0  assign en1 = en & in[3]; // Enable upper half when MSB=1    // Instantiate two 3-to-8 decoders  decoder3to8 dec\_low (.in(in[2:0]), .en(en0), .out(d0));  decoder3to8 dec\_high(.in(in[2:0]), .en(en1), .out(d1));    assign out = {d1, d0}; // Concatenate outputs    endmodule    module decoder3to8 (  input [2:0] in,  input en,  output reg [7:0] out  );  always @(\*) begin  if (!en)  out = 8'b00000000;  else begin  case(in)  3'b000: out = 8'b00000001;  3'b001: out = 8'b00000010;  3'b010: out = 8'b00000100;  3'b011: out = 8'b00001000;  3'b100: out = 8'b00010000;  3'b101: out = 8'b00100000;  3'b110: out = 8'b01000000;  3'b111: out = 8'b10000000;  default: out = 8'b00000000;  endcase  end  end  endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_decoder4to16(  );  reg [3:0] in;  reg en;  wire [15:0] out;    // Instantiate DUT  decoder4to16 uut (  .in(in),  .en(en),  .out(out)  );    initial begin  $display("Time\tEn\tIn\t\tOut");  $monitor("%0t\t%b\t%h\t%b", $time, en, in, out);    // Test sequence  en = 0; in = 4'b0000; #10;  en = 1;    // Apply all inputs  in = 4'b0000; #10;  in = 4'b0001; #10;  in = 4'b0010; #10;  in = 4'b0011; #10;  in = 4'b0100; #10;  in = 4'b0101; #10;  in = 4'b0110; #10;  in = 4'b0111; #10;  in = 4'b1000; #10;  in = 4'b1001; #10;  in = 4'b1010; #10;  in = 4'b1011; #10;  in = 4'b1100; #10;  in = 4'b1101; #10;  in = 4'b1110; #10;  in = 4'b1111; #10;    en = 0; #10; // Disable test  $stop;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**Conclusion:**

**Suggested Reference:**

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 9

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Design 4:2, 8:3 Priority Encoder using Verilog / VHDL.**

**Competency and Practical Skills:**

**Relevant CO:**

**Objectives:**

**Equipment / Instruments:** Laptop or Computer with Xilinx / Tool

**Basic Theory:** An Encoder is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2*n* input lines and ‘*n*’ output lines. It will produce a binary code depending on the input line activated.

**Priority Encoder:** A priority encoder produces correct code at the output even when multiple lines at the inputs are simultaneously active high (logic ‘1’). As shown above, the 4-to-2 priority encoder has four inputs (D3, D2, D1, D0) and two outputs (I1 and I0). Here, the input, D3 has the highest priority; whereas, the input, D0 has the lowest priority. In this case, even if more than one input lines are at logic ‘1’ at the same time, the output will be the binary code corresponding to the input, which is having higher priority.

**(a) 4 : 2 Priority Encoder**

|  |  |
| --- | --- |
| **Verilog code** | module encoder4\_2 (  input [3:0] I,  output [1:0] Y,  output valid  );  assign Y[1] = I[3] | I[2];  assign Y[0] = I[3] | (~I[2] & I[1]);  assign valid = I[3] | I[2] | I[1] | I[0];  endmodule |
| **Verilog code (TB)** | module tb\_encoder4\_2;  reg [3:0] I;  wire [1:0] Y;  wire valid;  encoder4\_2 dut (  .I(I),  .Y(Y),  .valid(valid)  );  initial begin  $dumpfile("encoder4\_2.vcd");  $dumpvars(0, tb\_encoder4\_2);  I = 4'b0000; #10;  I = 4'b0001; #10;  I = 4'b0010; #10;  I = 4'b0100; #10;  I = 4'b1000; #10;  I = 4'b1010; #10;  I = 4'b0111; #10;  I = 4'b0011; #10;  $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

1. **8:3 Priority Encoder**

|  |  |
| --- | --- |
| **Verilog code** | `timescale 1ns / 1ps  module priority\_encoder\_8to3(  input [7:0] D, // 8 input lines  output reg [2:0] Y, // Encoded 3-bit output  output reg valid // Valid = 1 when any input is active  );  always @(\*) begin  valid = 1; // Assume valid, then check inputs  casex(D)  8'b1xxxxxxx: Y = 3'b111; // D7 has highest priority  8'b01xxxxxx: Y = 3'b110; // D6  8'b001xxxxx: Y = 3'b101; // D5  8'b0001xxxx: Y = 3'b100; // D4  8'b00001xxx: Y = 3'b011; // D3  8'b000001xx: Y = 3'b010; // D2  8'b0000001x: Y = 3'b001; // D1  8'b00000001: Y = 3'b000; // D0 (lowest priority)  default: begin  Y = 3'b000;  valid = 0; // No input active  end  endcase  end    endmodule |
| **Verilog code (TB)** | `timescale 1ns / 1ps  module tb\_priority\_encoder\_8to3(  );  reg [7:0] D; // Input lines  wire [2:0] Y; // Encoded output  wire valid; // Valid output    // Instantiate DUT  priority\_encoder\_8to3 uut (  .D(D),  .Y(Y),  .valid(valid)  );    initial begin  // Monitor values  $monitor("Time=%0t | D=%b | Y=%b | valid=%b", $time, D, Y, valid);    // Test cases  D = 8'b00000000; #10; // No input  D = 8'b00000001; #10; // D0 active  D = 8'b00000010; #10; // D1 active  D = 8'b00000100; #10; // D2 active  D = 8'b00010000; #10; // D4 active  D = 8'b10000000; #10; // D7 active (highest priority)  D = 8'b10101010; #10; // Multiple active -> D7 should win  D = 8'b01010101; #10; // Multiple active -> D6 should win    $finish;  end    endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **Simulation waveforms:** |  |

**Conclusion:**

**Suggested Reference:**

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 10

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim: Design 4-bit Comparator using Verilog / VHDL.**

**Competency and Practical Skills:**

**Relevant CO:**

**Objectives:**

**Equipment / Instruments:** Laptop or Computer with Xilinx / Altera (Intel)Tools.

**Basic Theory:**

A **comparator** is a combinational circuit that compares two binary numbers and determines their relative magnitudes.

A **4-bit comparator** compares two 4-bit binary numbers, say **A = A3 A2 A1 A0** and **B = B3 B2 B1 B0**, and produces three outputs:

1. **A > B** → High when A is greater than B
2. **A = B** → High when A is equal to B
3. **A < B** → High when A is less than B

The comparison is done bit by bit starting from the **most significant bit (MSB)**:

* If **A3 ≠ B3**, then the larger number is decided immediately.
* If **A3 = B3**, then the next bit (A2 vs B2) is checked, and so on until A0 vs B0.
* If all bits are equal, then A = B.

**Applications of Comparator:**

* Digital systems for decision making
* Sorting operations
* Address decoding in memory systems
* Arithmetic and logical units

**Program Code:**

|  |  |
| --- | --- |
| **Verilog code** | module comparator\_4 (  input [3:0] A,  input [3:0] B,  output AeqB,  output AgtB,  output AlsB  );  assign AeqB = (A == B);  assign AgtB = (A > B);  assign AlsB = (A < B);  endmodule |
| **Verilog Testbench code** | module tb\_comparator\_4;  reg [3:0] A, B;  wire AeqB, AgtB, AlsB;  comparator\_4 dut (  .A(A),  .B(B),  .AeqB(AeqB),  .AgtB(AgtB),  .AlsB(AlsB)  );  initial begin  $dumpfile("comparator\_4.vcd");  $dumpvars(0, tb\_comparator\_4);  A = 4'b0000; B = 4'b0000; #10;  A = 4'b1010; B = 4'b0110; #10;  A = 4'b0011; B = 4'b0100; #10;  A = 4'b1111; B = 4'b1111; #10;  A = 4'b1000; B = 4'b1100; #10;  A = 4'b0111; B = 4'b0011; #10;  $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **waveform:** |  |

**Conclusion:**

**Quiz:**

**Suggested Reference:**

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |

# Experiment No: 11

**Aim: Design BCD and Ripple Carry Adder using Verilog / VHDL.**

**Competency and Practical Skills:**

**Relevant CO:**

**Objectives:**

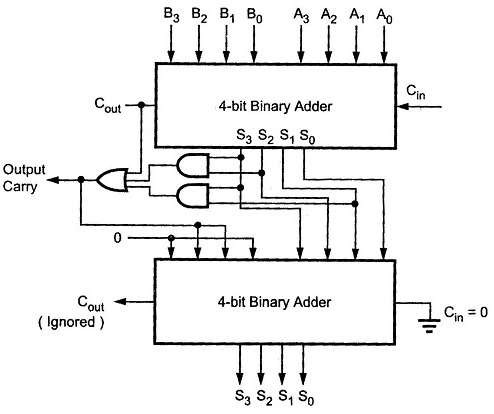
**Equipment / Instruments:** Laptop or Computer with Xilinx / Altera (Intel)Tools.

**Basic Theory:**

BCD Adder – Theory

* Adds two decimal digits in BCD form (0–9).
* If sum > 9, add 6 (0110) to correct result.
* Used in calculators and digital systems needing decimal outputs. Ripple Carry Adder – Theory
* Made of full adders connected in series.
* Carry output of one adder goes into the next → “ripple effect.”
* Simple design but slow for large numbers.

BCD Adder Block diagram:



**(A)BCD Adder:**

|  |  |
| --- | --- |
| **Verilog code** | module bcd\_adder (  input [3:0] A,  input [3:0] B,  input Cin,  output [3:0] Sum,  output [3:0] Correction,  output Cout  );  wire [4:0] temp\_sum;  wire correction\_needed;  assign temp\_sum = A + B + Cin;  assign correction\_needed = (temp\_sum > 9);  assign Correction = correction\_needed ? 4'b0110 : 4'b0000;  assign {Cout, Sum} = temp\_sum + Correction;  endmodule |
| **Verilog code(TB)** | module tb\_bcd\_adder;  reg [3:0] A, B;  reg Cin;  wire [3:0] Sum, Correction;  wire Cout;  bcd\_adder uut (  .A(A),  .B(B),  .Cin(Cin),  .Sum(Sum),  .Correction(Correction),  .Cout(Cout)  );  initial begin  $dumpfile("bcd\_adder.vcd");  $dumpvars(0, tb\_bcd\_adder);  A = 4'd5; B = 4'd3; Cin = 0; #10;  A = 4'd7; B = 4'd5; Cin = 0; #10;  A = 4'd9; B = 4'd9; Cin = 1; #10;  A = 4'd4; B = 4'd4; Cin = 1; #10;  $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **waveform:** |  |

(**B**) **Ripple Carry Adder:**

|  |  |
| --- | --- |
| **Verilog code** | module ripple\_adder (  input [3:0] A,  input [3:0] B,  input Cin,  output [3:0] Sum,  output Cout  );  wire C1, C2, C3;  FA\_2 FA0 (A[0], B[0], Cin, Sum[0], C1);  FA\_2 FA1 (A[1], B[1], C1, Sum[1], C2);  FA\_2 FA2 (A[2], B[2], C2, Sum[2], C3);  FA\_2 FA3 (A[3], B[3], C3, Sum[3], Cout);  endmodule |
| **Verilog code(TB)** | module tb\_ripple\_adder;  reg [3:0] A, B;  reg Cin;  wire [3:0] Sum;  wire Cout;  ripple\_adder uut (  .A(A),  .B(B),  .Cin(Cin),  .Sum(Sum),  .Cout(Cout)  );  initial begin  $dumpfile("ripple\_adder.vcd");  $dumpvars(0, tb\_ripple\_adder);  A = 4'b0001; B = 4'b0010; Cin = 0; #10;  A = 4'b0101; B = 4'b0110; Cin = 1; #10;  A = 4'b1111; B = 4'b0001; Cin = 0; #10;  A = 4'b1001; B = 4'b1001; Cin = 1; #10;  $finish;  end  endmodule |
| **RTL Schematic:** |  |
| **Synthesis Schematic:** |  |
| **waveform:** |  |

**Conclusion:**

**Suggested Reference:**

**References used by the students:**

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **1** | **2** | **3** | **4** | **5** | **Total** |
| **Marks** |  |  |  |  |  |  |