Kammavari Sangham (R) 1952

K. S. GROUP OF INSTITUTIONS

# K. S. POLYTECHNIC

**Approved by AICTE, New Delhi; Recognized by DTE, Govt., of Karnataka**

#15, Near Vajarahalli, Mallasandra, off kanakapura Road, Bengaluru – 560109

# Logic Design using Verilog : 20EC32P Lab Manual

**Prepared by:**

***Mr.Chidananda** D.ECE., B.E., M.Tech*

*(Senior Lecturer, Dept. of E&C Engg.)*

**Government of Karnataka**
**DEPARTMENT OF COLLEGIATE AND TECHNICAL EDUCATION**

| | | | |
|---|---|---|---|
| **Programme** | Electronics and Communication | **Semester** | III |
| **Course Code** | 20EC32P | **Type of Course** | Program Core |
| **Course Name** | Logic Design using Verilog | **Contact Hours** | 8 hours/week<br>104 hours/semester |
| **Teaching Scheme** | L:T:P :: 3:1:4 | **Credits** | 6 |
| **CIE Marks** | 60 | **SEE Marks** | 40 |

## 1. Rationale

Digital Electronics is a field of electronics involving the study of digital signals and engineering of devices that use or produce them. It is very important in today's life because if digital circuits are used instead of analog circuits the signals can be transmitted without degradation due to noise. Also in a digital system information stored is easier than that of analog systems. The functionality of digital circuits can be changed easily with the help of software without changing the actual circuit. Verilog, a Hardware Description Language, is used for describing digital electronic circuits and systems. It is used for verification of digital circuits through simulation, for timing analysis, for test analysis and for logic synthesis**.**

## 2. Course Outcomes: On successful completion of the course, the students will be able to:

| | |
|---|---|
| CO-01 | List the types of Verilog modeling and the use of each model for specific application |
| CO-02 | Design and construct a sequential circuit for a given application and test the circuit to obtain the desired result/output. |
| CO-03 | Compare and contrast combinational and sequential circuits and simulate a given circuit using Verilog descriptions to test to obtain the desired result/output |
| CO-04 | List the various types of A to D, D to A converters along with memory and for a given application select the appropriate converters and/or memory types to be used to obtain the given result/output. |

## 3. Course Content

| **Week** | **CO** | **PO** | **Lecture (Knowledge Criteria)** | **Tutorial (Activity Criteria)** | **Practice (Performance Criteria)** |
|---|---|---|---|---|---|
| | | | 3 hours/week | 1 hour/week | 4 hours/week (2 hours/batch twice in a week) |
| 1 | 1 | 1,4,5,6,7 | 1. VLSI - Introduction, Importance & Need.<br>HDL- Introduction, Importance, Need & Types.<br><br>2. Introduction to Verilog HDL, Types of modeling- Switch level, Structural, Data flow and Behavioral.<br><br>3. Basic Concepts- Lexical conventions, comments, keywords, identifiers, strings. | Refer Table 1 | 1. Familiarization of Xilinx software.<br><br>2. Familiarization of FPGA/CPLD KIT. |

| | | | | | |
|---|---|---|---|---|---|
| 2 | 1 | 1,2,4 | 1. Data types -Value Set, Wires, Nets, Registers, Vectors, Integers, Real, Time, Parameters, Arrays, Strings.<br><br>2. Operators- Arithmetic, Logical, Relational, Bit-wise.<br><br>3. Reduction, Shift, Concatenation, Replication, Conditional operators. Operator Precedence. | Refer Table 1 | 1. Demonstrate and Practice simple examples using different data types.<br><br>2. Compute the output for expressions having different operators using simple programs. |
| 3 | 1,3 | 1,2,3,6 | 1. Program structure- Module declaration, port declaration, port connection.<br><br>2. Gate level modeling for basic gates.<br><br>3. Gate level Verilog description for half adder, full adder. | Refer Table 1 | Write the verilog code, simulate and download to FPGA/CPLD kit for the following<br><br>1. 2 input basic gates using gate level modelling.<br><br>2. Full adder and full subtractor using gate level modelling. |
| 4 | 1,3 | 1,2,3,4,6 | 1. Data flow modeling- Continuous assignment, Module instantiations, net declaration, delays, expressions.<br><br>2. Data flow Verilog description of multiplexer and demultiplexer.<br><br>3. Data flow Verilog description for 4-bit comparator | Refer Table 1 | Write the verilog code, simulate and download to FPGA/CPLD kit for the following.<br><br>1. 4:1 Mux and 1:4 Demux using data flow modeling.<br><br>2. Comparator using data flow modeling. |
| 5 | 1,3 | 2,3,4,6 | 1. System tasks-display, strobe, monitor, reset, stop, finish. Compiler directives- include, define. Behavioral modeling- Always and Initial statements.<br><br>2. Procedural Assignments- Blocking and non-blocking assignments.<br>Timing Control-Delay, Event<br><br>3. Conditional statements-if, if-else, Case, Loops- While, For, Repeat, Forever. | Refer Table 1 | 1a.Write and execute simple programs to illustrate conditional statements.<br>1b. Write and execute simple programs to illustrate loops.<br><br>2. Write the verilog code, simulate and download to FPGA/CPLD kit for a 4-bit ALU with any 2 arithmetic and logical operations. |
| 6 | 1,3 | 1,2,3,4,6 | 1. Behavioral Verilog description for BCD to seven segment decoder for common anode display using if-else, Case.<br><br>2. Traffic light controller using Behavioral description.<br><br>3. Test bench- Need, Importance, testbench for half adder. | Refer Table 1 | 1. Write the verilog code, simulate and download to FPGA/CPLD kit for a BCD to seven segment decoder using case statement.<br><br>2. Write and simulate a Test bench for half adder. |

| | | | | | |
|---|---|---|---|---|---|
| 7 | 2 | 1,2,3,4,6,7 | 1. Sequential circuits - Introduction. Flip flops- types, SR flip flop- Gate level circuit using NAND gates, truth table, working, timing diagram.<br><br>2. JK, JK-MS flip flops-Logic circuit, truth table, working, timing diagram.<br><br>3. D, T flip flops-Logic circuit, truth table, working, timing diagram. Relevance of Asynchronous inputs to flip-flops. | Refer Table 1 | 1. Construct and test clocked SR Flip flop using NAND gates in digital trainer kit.<br><br>2. Implement D and T Flip flops using JK flip flop in digital trainer kit and observe the timing diagram. |
| 8 | 2,3 | 1,2,3,4 | 1. Verilog description of SR flip flops using data flow modeling.<br><br>2. Verilog description of JK flip flop using behavioral modeling.<br><br>3. Registers- Classification of registers, realization of simple (3 or 4 bit) SISO using flip-flops. | Refer Table 1 | Write the verilog code, simulate and download to FPGA/CPLD kit for the following.<br><br>1. SR, JK flip flops using data flow modeling<br>2.D, T flip flops using behavioral modeling |
| 9 | 2,3 | 1,2,3,4,6,7 | 1. Realization of SIPO, PISO and PIPO using flip flops.<br><br>2. Concept of universal shift-register. Ring counter and Johnson's counter (3 bit).<br><br>3. Verilog description of any one shift register using any modeling. | Refer Table 1 | Construct and verify the working of the following using suitable IC in digital trainer kit<br><br>1. SISO, SIPO, PISO and PIPO(4-bit) shift registers.<br>2. Ring and Johnson counter(4-bit). |
| 10 | 3 | 1,3,4,6,7 | 1. Counters - definition, classification, modulus. Working and realization of asynchronous (3 bit/4 bit) counters using flip-flops.<br><br>2. Working and realization of synchronous (3-bit/ 4-bit) counters and their comparison.<br><br>3. Realization of partial mod (mod n) counters-asynchronous, synchronous. | Refer Table 1 | Construct and verify the working of the following using digital trainer kit<br><br>1. 3 bit ripple counter using IC 7476.<br>2. 4 bit counter as a frequency divider. |
| 11 | 3,4 | 1,2,6,7 | 1. Realization of higher-mod counters using lower-mod counters. Concept of up/ down counters.<br><br>2. Verilog description of any one counter using any modeling.<br><br>3. Data converters- Need for DAC and ADC, DAC specifications, types, working of Weighted resistor type. | Refer Table 1 | 1. Write the verilog code, simulate and download to FPGA/CPLD kit for an up/down counter using behavioral modeling.<br><br>2. Construct/Simulate and verify the working of R-2R DAC. |
| 12 | 4 | 1,2,3,4,6,7 | 1. ADC specifications. types, working of Flash ADC. | Refer Table 1 | 1. Construct/Simulate and verify the working of Flash ADC. |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
|  |  |  | 2. Working of Successive approximation and dual slope ADCs.<br><br>3. Memory devices- Introduction, classification based on different criteria, read and write operations. |  | 2. Illustrate the storing and retrieving of data in RAM using suitable IC. |
| 13 | 4 | 1,2,3, 4,7 | 1. Introduction to PLDs- PAL, PLA, CPLD, FPGA, ASIC.<br>IC Design Verification – Types & Stages.<br><br>2. PAL- Architecture, Implementation of a Boolean expressions using PAL.<br><br>3. PLA-Architecture, Implementation of a Boolean expressions using PLA. | Refer Table 1 | 1. Implementation of Boolean expressions using PAL.<br><br>2. Implementation of Boolean expressions using PLA. |
| **Total in hours** |  |  | **39** | **13** | **52** |

**Note: 1) In Practice sessions Video demonstration should be followed by MCQs/Quiz/Subjective questions and the evaluation has to be documented.**

**2) In Practice sessions, all circuits should be simulated using suitable software before its construction and verification.**

**TABLE 1: Suggested activities for tutorials**

**The list is shared as an example and not inclusive of all possible activities of the course.**

**The list of activities for one week can be shared among teams in a batch of students.**

| **Week No.** | **Suggested activities for tutorials** |
|---|---|
| **01** | 1. Explain the typical design flow for VLSI IC Circuits.<br><br>2. Give a presentation on comparison of different types of HDLs.<br><br>3. Give a presentation on comparison of different types of modeling in Verilog. |
| **02** | 1. Prepare a report on declaration and initialization of variables of different data types in Verilog.<br><br>2. Prepare a report on hierarchy of operators. |
| **03** | 1. Explain basic components of a module? Which components are mandatory?<br><br>2. Prepare a report on Hierarchical names for variables.<br><br>3. Write and explain a Verilog code for 4:1 mux and 1:4 demux using gate level modeling. |

| | |
|---|---|
| **04** | 1. Write and explain the Verilog code for full adder using data flow modeling.<br><br>2 Write and explain the Verilog code for 8:1 mux using data flow modeling. |
| **05** | 1. Give a presentation on the differences between tasks and functions<br><br>2. Illustrate the use of system tasks with examples.<br><br>3. Illustrate the use of gate delays to model timing for a simple logic equation. |
| **06** | 1. Compare if-else and case statements with the help of examples.<br><br>2. Compare all loops with the help of examples.<br><br>3. Write and explain the verilog code for full subtractor and 1:8 demux using behavioral modeling.<br><br>4. Explain the Verilog Test bench with an example to verify the HDL designs. |
| **07** | 1. Prepare a report on differences between Combinational and Sequential circuits with examples.<br><br>2. Give a presentation on application of flip flop as bounce elimination switch.<br><br>3. Demonstrate the working of flip flop as a one bit memory element. |
| **08** | 1. Prepare a report on flip flop ICs and their features.<br><br>2. Give a presentation on eliminating race -around condition in JK flip flop.<br><br>3. Compare the advantages and disadvantages of all flip flops. |
| **09** | 1. Prepare a report on shift register ICs and their features.<br><br>2. Give a presentation on applications of shift registers in real life.<br><br>3. Demonstrate the working of IC 7495 as shift register. |
| **10** | 1. Prepare a report on differences between asynchronous and synchronous counters.<br><br>2. Give a presentation on how counters can be used in a simple car parking system.<br><br>3. Give a presentation on implementation of footfall counter for various purposes |
| **11** | 1. Prepare a report & explain the specifications of DAC and ADC ICs.<br><br>2. Give a presentation on any application of DAC in real life.<br><br>3. Give a presentation on any application of ADC in real life. |

| | |
|---|---|
| **12** | 1. Prepare a report & explain the types of RAM and ROM.<br>2. Give a presentation on usage of RAM and ROM in different digital devices. |
| **13** | 1. Study the latest technological changes in this course and present the impact of these changes on industry.<br>2. Prepare a report on CPLD, FPGA and ASIC and its applications.<br>3. Give a presentation on importance or scope of Design Verification in Integrated circuit designs. |

**LINKS.**

1. https://verilogguide.readthedocs.io/en/latest/verilog/testbench.html
2. https://youtu.be/XES0QUi8ttY(week 11, exp 2)
3. https://www.youtube.com/watch?v=krmXg-WTbIU (week 12, exp 1)
4. http://www.asicguru.com/verilog/tutorial/system-tasks-and-functions/68/.
5. https://youtu.be/vHlg_QLGIQ (week 7,exp 3)
6. https://youtu.be/AtX5x53FcLI (week 9,exp 3)
7. https://youtu.be/Bx_4rsUAGoM
8. https://www.irisys.net/people-counting.

## 4. CIE and SEE Assessment Methodologies

| Sl. No | Assessment | Test Week | Duration In minutes | Max marks | Conversion |
|---|---|---|---|---|---|
| 1. | CIE-1 Written Test | 5 | 80 | 30 | Average of three tests 30 |
| 2. | CIE-2 Written Test | 9 | 80 | 30 | |
| 3 | CIE-3 Written Test | 13 | 80 | 30 | |
| 4. | CIE-4 Skill Test-Practice | 6 | 180 | 100 | Average of two skill tests 20 |
| 5 | CIE-5 Skill Test-Practice | 12 | 180 | 100 | |
| 6 | CIE-6 Portfolio continuous evaluation of Activity through Rubrics | 1-13 | | 10 | 10 |
| | | | | Total CIE Marks | 60 |
| | Semester End Examination (Practice) | | 180 | 100 | 40 |
| | | | | **Total Marks** | **100** |

## 5. Format for CIE (1,2,3) Written Test

| | | | | | | |
|---|---|---|---|---|---|---|
| Course Name | **Logic Design Using Verilog** | | Test | I/II/III | Sem | III/IV |
| Course Code | **20EC32P** | | Duration | 80 Min | Marks | 30 |
| **Note:** Answer any one full question from each section. Each full question carries 10 marks. | | | | | | |
| Section | Assessment Questions | | Cognitive Levels | | Course Outcome | Marks |
| I | 1 | | | | | |

| | | | | |
|---|---|---|---|---|
| | 2 | | | |
| II | 3 | | | |
| | 4 | | | |
| III | 5 | | | |
| | 6 | | | |
| Note for the Course coordinator: Each question may have one, two or three subdivisions. Optional questions in each section carry the same weightage of marks, Cognitive level and course outcomes. | | | | |

**5. (a) Format for CIE-4 Skill Test -Practice.**

| SL. No. | COs | Particulars/Dimension | Marks |
|---|---|---|---|
| 1 | 1 | List the types of Verilog modelling and the use of each model for specific application. | 20 |
| 2 | 3 | Write two Verilog programs on combinational circuits for a given application -40 Marks<br>Simulation - 20 Marks<br>Download to FPGA kit - 10 Marks | 70 |
| 3 | 1,3 | PortFolio evaluation of Practice sessions through rubrics | 10 |
| | | **Total Marks** | **100** |

**5. (b) Format for CIE-5 Skill Test - Practice.**

| SL. No. | COs | Particulars/Dimension | Marks |
|---|---|---|---|
| 1 | 2 | Write a Sequential circuit for a given application -20 Marks<br>Conduction using DTK -20 Marks<br>Output -10 Marks | 50 |
| 2 | 3 | Write a Verilog program on Sequential circuits for a given application - 10 Marks<br>Simulation -5 Marks<br>Output - 5 Marks | 20 |
| 3 | 4 | Identify various types of A to D, D to A converters/ memory for a given application & select the appropriate converters/ memory types needed to obtain the required output. | 20 |
| 4 | 2,3,4 | Portfolio evaluation of Practice sessions through rubrics. | 10 |
| | | **Total Marks** | **100** |

**6. Rubrics for Assessment of Activity (Qualitative Assessment)**

| Sl. No. | Dimension | Beginner | Intermediate | Good | Advanced | Expert | Students Score |
|---|---|---|---|---|---|---|---|
| | | 2 | 4 | 6 | 8 | 10 | |
| 1 | | Descriptor | Descriptor | Descriptor | Descriptor | Descriptor | 8 |
| 2 | | Descriptor | Descriptor | Descriptor | Descriptor | Descriptor | 6 |
| 3 | | Descriptor | Descriptor | Descriptor | Descriptor | Descriptor | 2 |
| 4 | | Descriptor | Descriptor | Descriptor | Descriptor | Descriptor | 2 |
| | Average Marks= (8+6+2+2)/4=4.5 | | | | | | **5** |

***Note:*** Dimension and Descriptor shall be defined by the respective course coordinator as per the activities

**7. Reference:**

| Sl. No. | Description |
|---|---|
| 1 | Fundamentals of Digital Logic with Verilog Design by Stephen Brown and Zvonko Vranesic |
| 2 | Verilog HDL by Samir Palnikar |
| 3 | Introduction to Verilog-Peter M Nyasulu |
| 4 | Verilog Tutorial-Deepak Kumar Tala |

**8. SEE Scheme of Evaluation**

| SL. No. | COs | Particulars/Dimension | Marks |
|---|---|---|---|
| 1 | 1 | List the types of Verilog modelling and the use of each model for specific application | 10 |
| 2 | 3 | Write a Sequential circuit for a given application -10 Marks<br>Conduction using DTK -10 Marks<br>Output -10 Marks | 30 |
| 3 | 2 | Write a Verilog program for a given application - 10 Marks<br>Simulation - 10 Marks<br>Download to FPGA kit- - 10 Marks | 30 |
| 4 | 4 | Identify various types of A to D, D to A converters and memory and for a given application & select the appropriate converters and/or memory types needed to obtain the given output. | 10 |
| 5 | 1,2, 3,4 | Viva-Voce | 20 |
| | | **Total Marks** | **100** |

**9. Equipment/software list with Specification for a batch of 20 students**

| Sl. No. | Particulars | Specification | Quantity |
|---|---|---|---|
| 1 | Computers | Intel Core i5 11th gen/8GB RAM/1 TB HDD/256GB SSD/ Graphics 2 GB | 20 |
| 2 | Xilinx software | | |
| 3 | Digital trainer kits | | 20 |
| 4 | Verilog kits | | 20 |
| 5 | Dual trace oscilloscope | 20-30MHz | 10 |
| 6 | Digital multimeters | | 05 |
| 7 | Patch cards | different length | 250 |
| 8 | Digital IC Tester | | 02 |
| 9 | ICs 7400,7402,7404,7408,7432,7486,7442, 7445,7446,7474,7476,7427,7489,7490, 7494,7495,74141,74148,74153,74157, 74155,74193,74194,DAC0808,ADC-0800,741 | | 10 each |

## Week-1
## Experiment - 1

**Aim: Familiarization of Xilinx software**

The Xilinx ISE development environment allows an engineer to enter a design in a several source formats, including Verilog. ISE allows the design to be expressed in a hierarchal way as the project develops. This allows the engineer to easily change modules as needed and facilitates easy testing.

In an ISE design, all design files are called SOURCES. A source is a file that specifies part of the overall design. These include HDL files in various formats (Verilog, VHDL, ABLE), schematic entry files, and state diagrams. Other types of sources are things like test fixtures and constraint files. Some source types may be new to you, but they are integral to the ISE approach.

All source files are organized into a hierarchy that starts with the TOP-LEVEL MODULE. There are several ways to enter a working design into ISE. This tutorial will illustrate the preferred method. In the preferred method, the top-level module is a schematic file (.sch). Having a schematic as the top-level module allows all sub modules to be combined in the top-level module as they would be in a block diagram. A top-level schematic module also allows signal markers to be attached to the design more easily. Synthesizing a design in ISE comes down to two things. Gathering all the needed sources, and running processes on those sources. In ISE, a process is an operation that is performed on or with a source file. Different Processes are available for different sources. Selecting a source will enable a list of processes available to that source. Any processes run will be run down the hierarchy.

**Design Entry**

- Click on Xilinx ISE 9.1 icon.
- Go to file, click on new project and give the project name.
- Select the top level source type as HDL then click on next.
- Select the family name as Spartan2, device name as XC2S30, package name as TQ144 and speed as -6. Then click on next.
- Click on new source then select Verilog module and give the file name and click on next.
- Declare the file name and click on next and finish.
- Now type the given Verilog code on the program window and click on save.
- Double click on Synthesis option to verify any syntax errors in the program.
- Click on view RTL schematic to view the logic diagram for the entered program.
- Ensure that you get a green tick (✔) mark that indicates that there are no errors in the code you entered.
- Right click on module name and select new source, now select test bench waveform to verify the truth table output of the code you entered.
- Give the waveform name, then click on next and finish.
- In the initial timing and clock wizard window select he following options - Rising edge, Combinational clock or internal clock, Then click on next.
- Now a new signal window appears in which we need to manually allocate the data as per the truth table inputs.
- After allocating the data click on save.
- In the source window click on Behavioural simulation and then click processes option and select Xilinx ISE simulator, double click on simulate behavioural model.
- A new window appears with the specified input and corresponding output.
- The obtained output can be verified by comparing the truth table.

Result: Study on Familiarization of Xilinx software is completed successfully

## Week-1
## Experiment – 2

**Aim: Familiarization of FPGA/CPLD Kit**

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.

The Spartan-II family of FPGAs have a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input/Output Blocks (IOBs). There are four Delay-Locked Loops (DLLs), one at each corner of the die. Two columns of block RAM lie on opposite sides of the die, between the CLBs and the IOB columns. These functional elements are interconnected by a powerful hierarchy of versatile routing Channels. Spartan-II FPGAs are customized by loading configuration data into internal static memory cells. Unlimited reprogramming cycles are possible with this approach. Stored values in these cells determine logic functions and interconnections implemented in the FPGA. Configuration data can be read from an external serial PROM (master serial mode), or written into the FPGA in slave serial, slave parallel, or Boundary Scan modes. Spartan-II FPGAs are typically used in high-volume applications where the versatility of a fast programmable solution adds benefits. Spartan-II FPGAs are ideal for shortening product development cycles while offering a cost-effective solution for high volume production.

Spartan-II FPGAs achieve high-performance, low-cost operation through advanced architecture and semiconductor technology. Spartan-II devices provide system clock rates up to 200 MHz. In addition to the conventional benefits of high volume programmable logic solutions, Spartan-II FPGAs also offer on-chip synchronous single port and dual-port RAM (block and distributed form), DLL clock drivers, programmable set and reset on all flip-flops, fast carry logic, and many other features.

**Assigning FPGA Package Pins to IO Markers:**

- With the top-level schematic selected in the Sources/Module View tab, expand User Constraints item in the Process View tab, then double-click on Assign Package Pins and click Yes in the following pop-up window.
- UCF file will be opened within Xilinx PACE - a separate program for assigning constraints. Now enter the Location of the I/O pins. The Locations or pin numbers are marked on the Spartan 2 kit, Insert the respective values in the Loc column for the pins to be assigned.
- Save the .ucf file and close the Xilinx PACE window.
- Start by expand the Generate Programming File item in the processes view and double-click on Configure Device (iMPACT).
- On the Configure Devices dialog box that appears, ensure Boundary Scan Mode is selected
- Click Next, select Automatically connect to cable and identify Boundary Scan chain and then click Finish
- A new application window (iMPACT) will appear.
- Right click on the xc2s30 IC to assign the configuration file and select the bit file (test.bit) and click open.
- Select the program option and click on Ok.

Result: Study on Familiarization of FPGA/CPLD Kit is completed successfully

**Week-2**
**Experiment - 1**

**Aim: Demonstrate and Practice simple examples using different data types**

**Verilog Description for two input Arithmetic operations**

Consider the below Arithmetic operations

Y1 = A+B

Y2 = A-B

Y3 = A*B

Y4 = A/B

Y5 = A%B

**Verilog Code:**

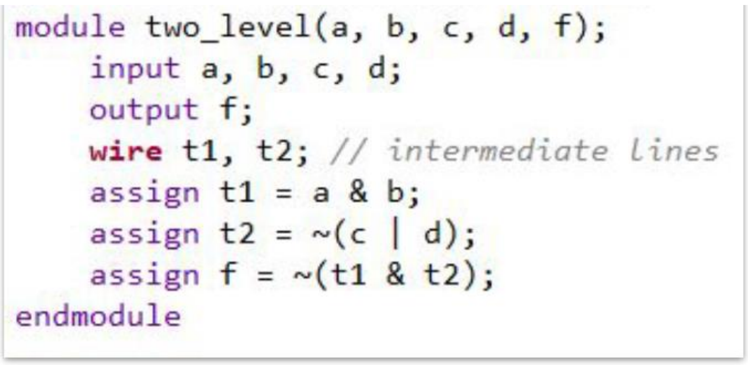
```
module Arithmetic (A, B, Y1, Y2, Y3, Y4, Y5);
      input  [2:0]  A,  B;
      output[4:0]  Y1,  Y2,  Y3,  Y4,  Y5;
      reg  [4:0]  Y1,  Y2,  Y3,  Y4,  Y5;
      always  @(A  or  B)
      begin
           Y1=A+B; //addition
           Y2=A-B; //subtraction
           Y3=A*B; //multiplication
           Y4=A/B; //division
           Y5=A%B; //modulus  of  A  divided  by  B
       end
 endmodule.
```

Result : Practicing simple examples using different data types is completed successfully

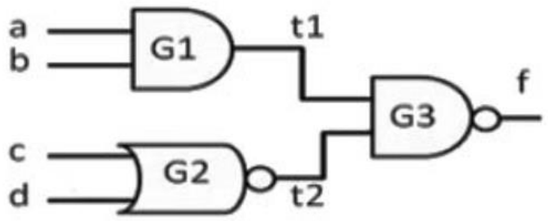
**Week-2**
**Experiment - 1**

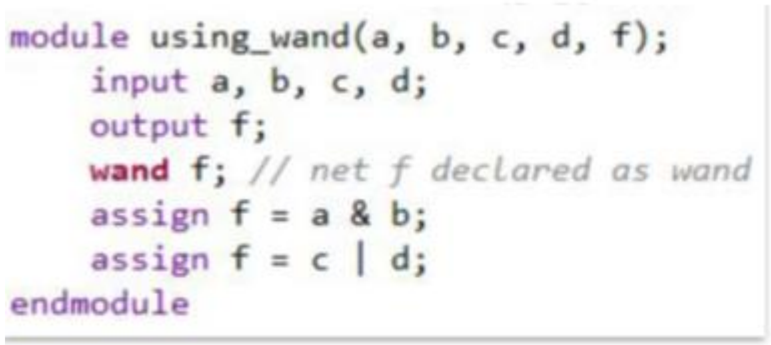
Aim: Compute the output for expressions having different operators using simple programs

Two level combinational circuit

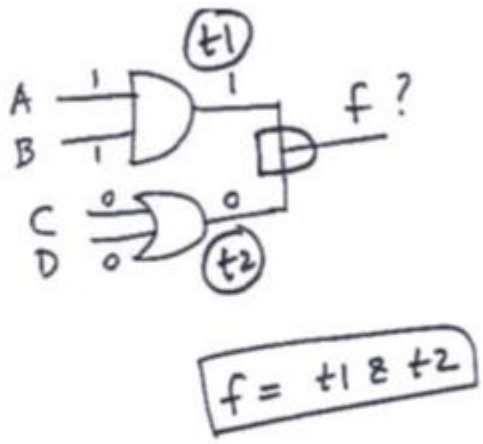


```
module two_level(a, b, c, d, f);
    input a, b, c, d;
    output f;
    wire t1, t2; // intermediate lines
    assign t1 = a & b;
    assign t2 = ~(c | d);
    assign f = ~(t1 & t2);
endmodule
```





```
module using_wand(a, b, c, d, f);
    input a, b, c, d;
    output f;
    wand f; // net f declared as wand
    assign f = a & b;
    assign f = c | d;
endmodule
```

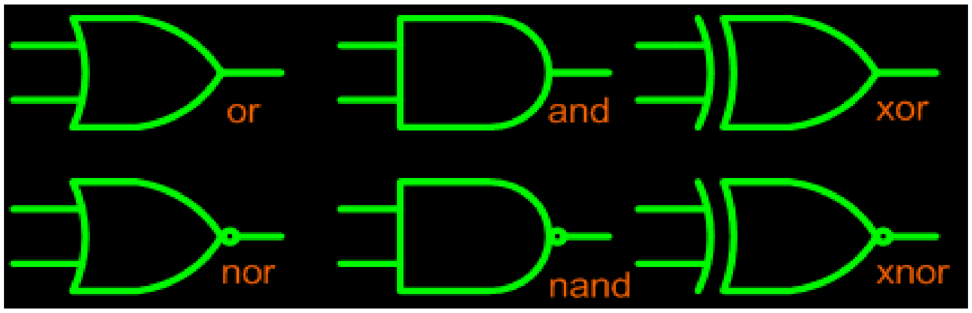


*Here, function realized will be*
*f = (A & B) & (C | D)*

# Week-3
# Experiment – 1

**Aim : Write the verilog code,simulate and download to FPGA/CPLD kit for the 2 input basic gates using gate level modelling.**

## Logic Diagram



## VERILOG Program

**AND Gate**

```
moduleandstr(x,y,z);

inputx,y;

output z;

and g1(z,x,y);

endmodule
```

**NAND Gate**

```
modulenandstr(x,y,z);

inputx,y;

output z;

nand g1(z,x,y);

endmodule
```

**OR Gate**

```
module orstr(x,y,z);

inputx,y;

output z;

or g1(z,x,y);

endmodule
```

**NOR Gate**

```
modulenorstr(x,y,z);

inputx,y;

output z;

nor g1(z,x,y);

endmodule
```

**XOR Gate**

```
module xorstr(x,y,z);

inputx,y;

output z;

xor g1(z,x,y);

endmodule
```

**XNOR Gate**

```
modulexnorstr(x,y,z);

inputx,y;

output z;

xnor g1(z,x,y);

endmodule
```

**NOT Gate**

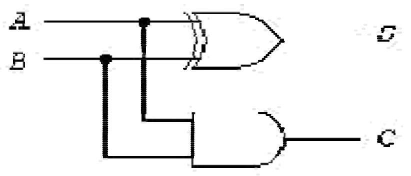
```
module  notstr(x,z);

input  x;

output z;

not g1(z,x);

endmodule
```

**Result: output for 2 input basic gates using gate level modeling is verified sucessfully**

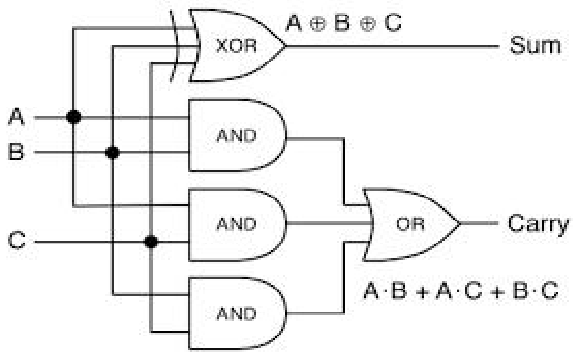
# Week-3
# Experiment - 2

**Aim: Write the verilog code, simulate and download to FPGA/CPLD kit for Full adder and full subtractor using gate level modelling.**

**Logic Diagram**



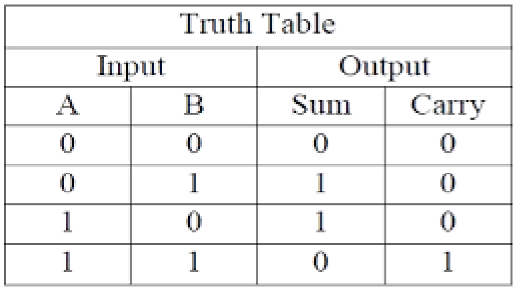
**Half adder**



**Full adder**

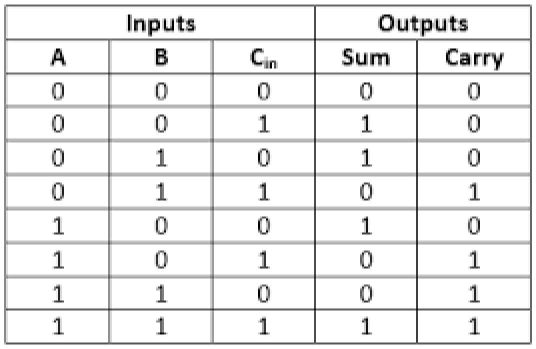
## Truth table

**Half adder**



| Truth Table | | | |
|---|---|---|---|
| Input | | Output | |
| A | B | Sum | Carry |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**Full adder**



| Inputs | | | Outputs | |
|---|---|---|---|---|
| A | B | $C_{in}$ | Sum | Carry |
| 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 |

## VERILOG Program

**HALF ADDER:**

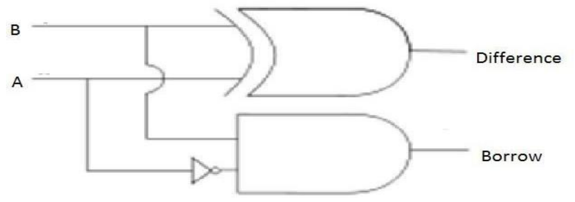
```
modulehalfaddstr(sum,carry,a,b);

outputsum,carry;

inputa,b;

xor(sum,a,b);

and(carry,a,b);

endmodule
```

**FULL ADDER:**

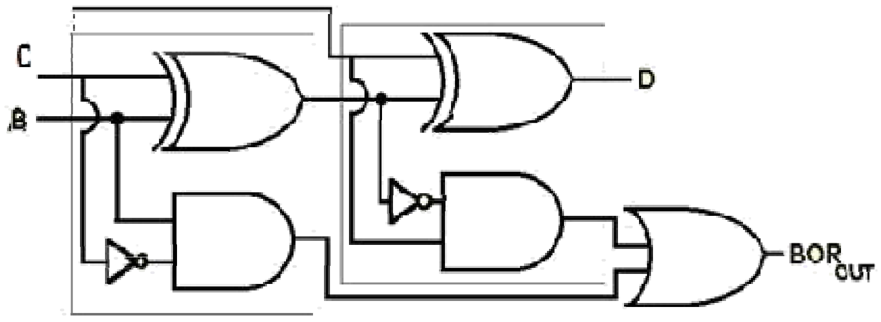
```
module
fulladdstr(sum,carry,a,b,c);

outputsum,carry;

inputa,b,c;

xor  g1(sum,a,b,c);

and g2(x,a,b);

and g3(y,b,c);

and g4(z,c,a);

or g5(carry,x,z,y);

endmodule
```

# Gate level Verilog description for half Subtractor, full Subtractor

**Logic Diagram**



**Half subtractor**



**Full Subtractor**

**Truth Table**

**Half subtractor**

| Inputs | | Outputs | |
|---|---|---|---|
| A | B | Difference | Borrow |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

**Full Subtractor**

| Inputs | | | Outputs | |
|---|---|---|---|---|
| A | B | $\text{Borrow}_{in}$ | Diff | Borrow |
| 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 |

**Verilog Code**

**HALF SUBTRACTOR:**

```
modulehalfsubtstr(diff,borrow,a,  b);


outputdiff,borrow;

inputa,b;

xor(diff,a,b);

and( borrow,~a,b);

endmodule
```

**FULL SUBTRACTOR:**

```
module
fullsubtstr(diff,borrow,a,b,c);

outputdiff,borrow;

inputa,b,c;

wire  a0,q,r,s,t;

not(a0,a);

xor(x,a,b);

xor(diff,x,c);

and(y,a0,b);

and(z,~x,c);

or(borrow,y,z);

endmodule
```

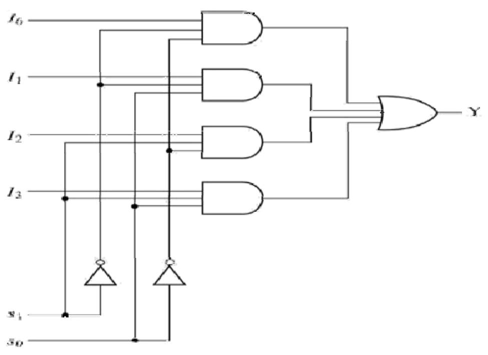
**Result: output for for Full adder and full subtractor using gate level modelling.
is verified successfully**

# Week-4
# Experiment – 1

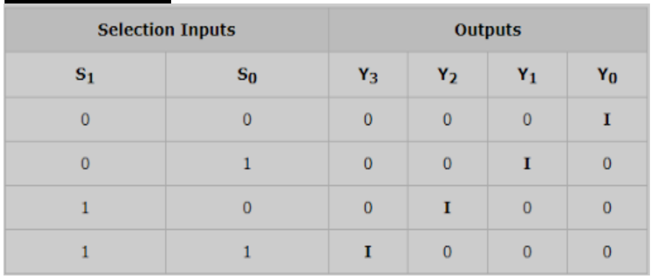
**Aim: Write the verilog code, simulate and download to FPGA/CPLD kit for 4:1 Mux and 1:4 Demux using data flow modeling.**

## <u>4:1 Multiplexer</u>

**<u>Logic Diagram</u>**



**<u>Truth Table</u>**

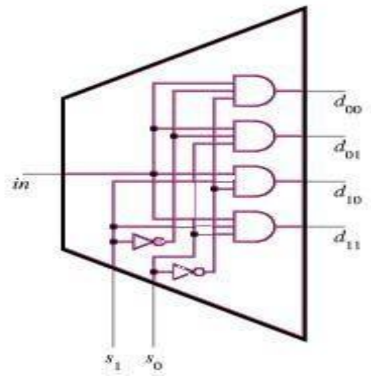


| Selection Inputs | | Outputs | | | |
|---|---|---|---|---|---|
| $S_1$ | $S_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 0 | 0 | 0 | 0 | 0 | I |
| 0 | 1 | 0 | 0 | I | 0 |
| 1 | 0 | 0 | I | 0 | 0 |
| 1 | 1 | I | 0 | 0 | 0 |

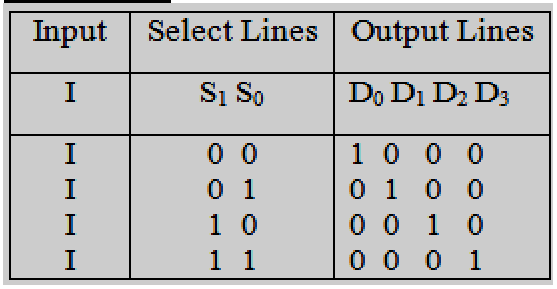
## <u>*Verilog Code*</u>

```
module  MUX41(i0,i1,i2,i3,s0,s1,y);
input  i0,i1,i2,i3,s0,s1;
output  y;
assign  y =((i0  &(~(s0))&(~(s1))))|(i1  &(~(s0))& s1)|(i2  & s0 & (~(s1)))|(i3  & s0 & s1);
endmodule
```

**1:4 De-Multiplexer**



**<u>Truth Table</u>**



| Input | Select Lines | Output Lines |
|---|---|---|
| I | $S_1$ $S_0$ | $D_0$ $D_1$ $D_2$ $D_3$ |
| I | 0 0 | 1 0 0 0 |
| I | 0 1 | 0 1 0 0 |
| I | 1 0 | 0 0 1 0 |
| I | 1 1 | 0 0 0 1 |

***Verilog Code***

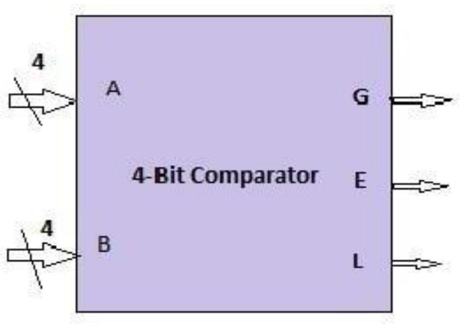
```
module  demux14df(in,d0,d1,d2,d3,s0,s1);
output  d0,d1,d2,d3;
input  in,s0,s1;
assign  d0 = in  & (~s0) & (~s1);
assign  d1 = in  & (~s0) & s1;
assign  d2 = in  & s0 & (~s1);
assign  d3 = in  & s0 & s1;
endmodule
```

**Result: output for 4:1 Mux and 1:4 Demux using data flow modeling is verified successfully**

## Week-4
## Experiment – 2

**Aim: Write the verilog code, simulate and download to FPGA/CPLD kit for Comparator using data flow modeling.**

**<u>Block Diagram</u>**



**<u>Truth Table</u>**

| A3 | A2 | A1 | A0 | B3 | B2 | B1 | B0 | A>B | A<B | A=B |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

***<u>Verilog Code</u>***

```
module  magcomp  (A,B,ALTB,AGTB,AEQB);
 input  [3:0]  A,B;
output  ALTB,AGTB,AEQB;
 assign  ALTB  = (A < B),  AGTB  = (A  > B),  AEQB  = (A  == B);
endmodule
```

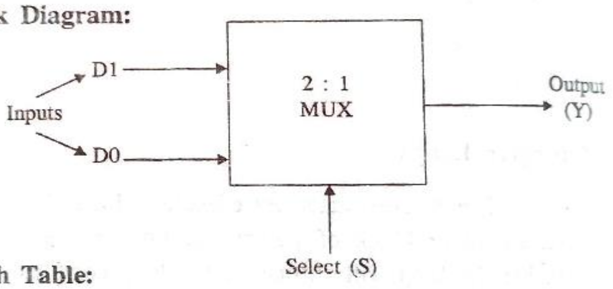
**Result: output for Comparator using data flow modeling is verified successfull**

## Week-5
## Experiment – 1a

**Aim: Write and execute simple programs to illustrate conditional statements**

**Consider 2:1 Mux to illustrate conditional statements**

Block Diagram:



Truth Table:

| Select | Output |
|---|---|
| S | Y |
| 0 | $D_0$ |
| 1 | $D_1$ |

Verilog Code :

```
module mux2to1( D0, D1, sel, Y_out);
     input D0;
     input D1;
     input sel;
     output Y_out;
     reg Y_out;
     always @(D0,D1,sel)
     begin
         if(sel == 0)
             Y_out = D0;
         else
             Y_out = D1;
         end
endmodule
```

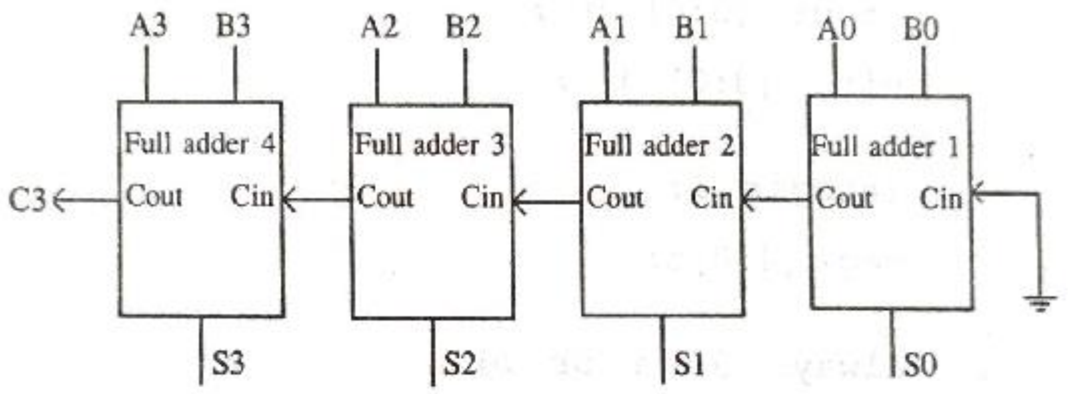
**Result: output for program to illustrate conditional statements is verified successfully**

# Week-5
# Experiment – 1b

**Aim: Write and execute simple program to illustrate loops.**

Consider a 4-bit parallel adder to illustrate verilog code using 'for' loop.

**Block Diagram:**



**Sample Truth Table:**

| A3 A2 A1 A0 | B3 B2 B1 B0 | S3 S2 S1 S0 | Cout |
|---|---|---|---|
| 1 1 1 0 | 0 1 1 0 | 0 1 0 0 | 1 |
| 0 1 0 1 | 1 0 1 0 | 1 1 1 1 | 0 |

**Expressions:**

S0 = a0 XOR b0 XOR cin

c0= (a0.b0) + (b0.cin) + (a0.cin)

**Verilog Code:**

```
module  adder_4bit  (a  ,b  ,sum  ,carry);

      output  [3:0]  sum  ;

      reg  [3:0]  sum  ;
```

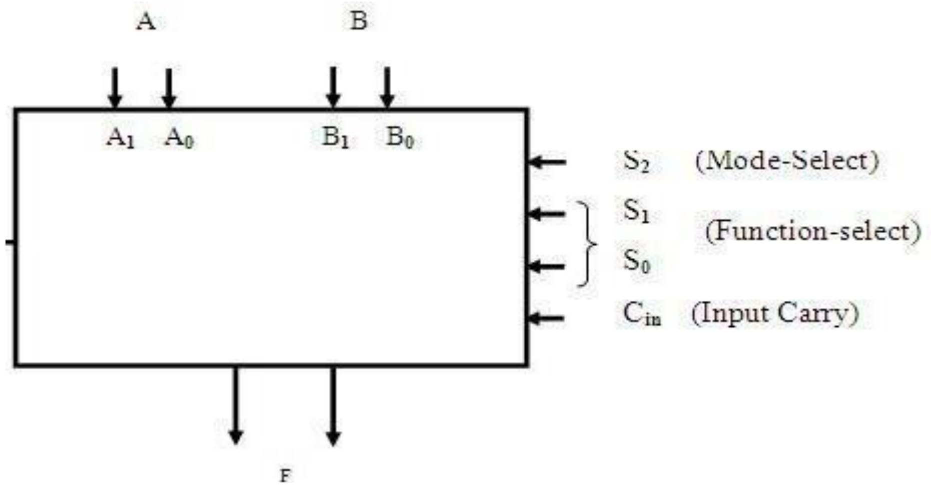
```
    output carry ;
    reg carry ;
    input [3:0] a ;
    wire [3:0] a ;
    input [3:0] b ;
    wire [3:0] b ;
    integer i;
    reg [4:0]s;
    always @ (a or b)
    begin
       .s[0] = 0;
    for (i=0;i<=3;i=i+1) begin
        sum [i] = a[i] ^ b[i] ^ s[i];
        s[i+1] = (a[i] & b[i]) | (b[i] &
        s[i]) | (s[i] & a[i]);
    end
    carry = s[4];
    end
endmodule
```

**Result: output for simple program to illustrate loops is verified successfully**

# Week-5
# Experiment – 2

**Aim: Write the verilog code, simulate and download to FPGA/CPLD kit for a ALU with any 2 arithmetic and logical operations.**

**Block Diagram**



**Verilog Code:**

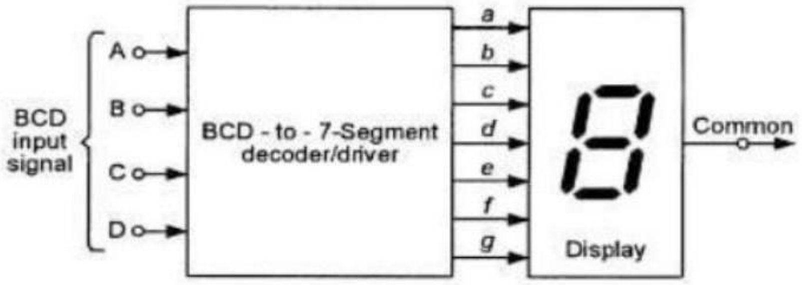
```
 module  alua  (a,b,s,y);
 input[1:0]a,b,s;
output[1:0]y;
 reg[1:0]y;
always  @(*)
 begin
case  (s)
2'b00:y=a+b;
2'b01:y=a-b;
 2'b10:y=a&b;
2'b11:y=a|b;
endcase
end
endmodule
```

**Result: output for 4-bit ALU with any 2 arithmetic and logical operations is verified successfully**

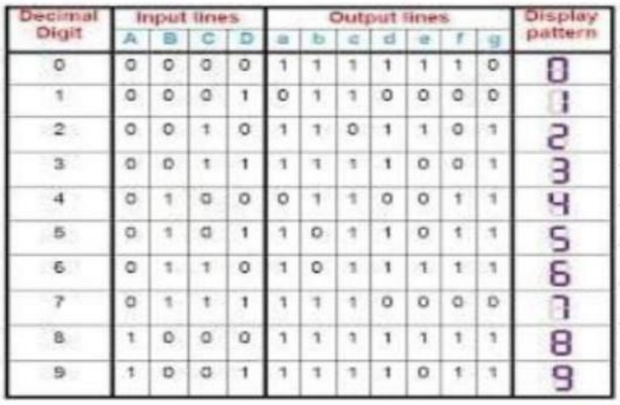
# Week-6
# Experiment – 1

**Aim: Write the verilog code, simulate and download to FPGA/CPLD kit for a BCD to seven segment decoder using case statement.**

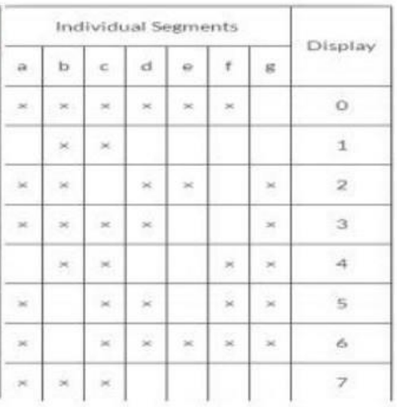
***Block Diagram***



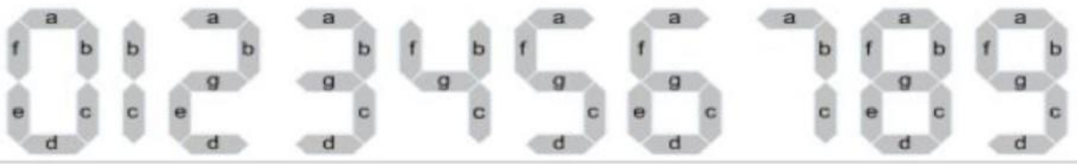
**Truth Table**



| Decimal Digit | Input lines | | | | Output lines | | | | | | | Display pattern |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | a | b | c | d | e | f | g | |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 9 |



| Individual Segments | | | | | | | Display |
|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | |
| × | × | × | × | × | × | | 0 |
| | × | × | | | | | 1 |
| × | × | | × | × | | × | 2 |
| × | × | × | × | | | × | 3 |
| | × | × | | | × | × | 4 |
| × | | × | × | | × | × | 5 |
| × | | × | × | × | × | × | 6 |
| × | × | × | | | | | 7 |



**Verilog Code :**

```
  module case(x,y);
  input[3:0]x;
  output[6:0]y;
  reg[6:0]y;
  always @ (x)
  begin
   case (x)
        0 : y =
   7'b0000001;
        1 : y =
   7'b1001111;
        2 : y =
   7'b0010010;
        3 : y =
 7'b0000110;
         4 : y =
 7'b1001100;
         5 : y =
 7'b0100100;
         6 : y =
 7'b0100000;
         7 : y =
 7'b0001111;
         8 : y =
 7'b0000000;
         9 : y =
 7'b0000100;
default : y = 7'b1111111;
 endcase
 end
endmodule
```

**Result: output for BCD to seven segment decoder using case statement is verified successfully.**

## Week-6
## Experiment – 2

**Aim: Write and simulate a Test bench for half adder.**

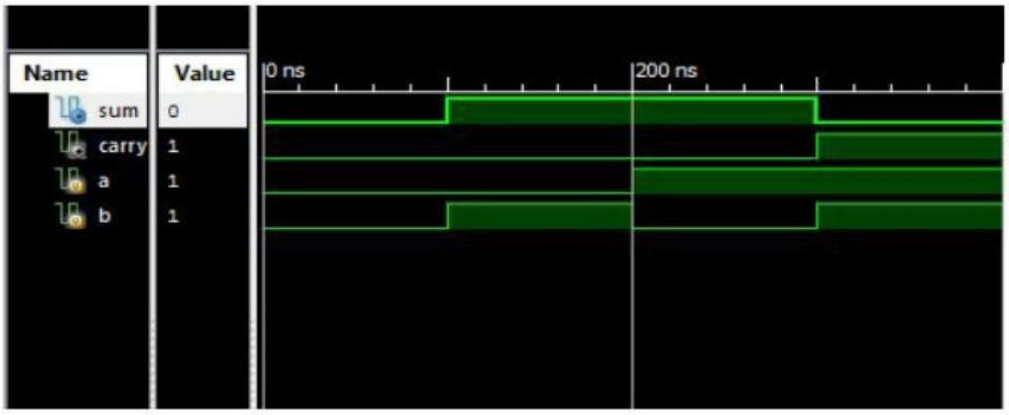
**Verilog Code**

```
module ha(sum,carry,a,b);
output sum,carry;
input a,b;
xor(sum,a,b);
and(carry,a,b);
endmodule
```

**Test Bench**

```
module ha_test;
        // Inputs
        reg a;
        reg b;
        // Outputs
        wire sum;
        wire carry;
        // Instantiate the Unit Under Test (UUT)
        ha uut (
                .sum(sum),
                .carry(carry),
                .a(a),
                .b(b)
        );

        initial begin
                // Initialize Inputs
                a = 0;
                b = 0;

                // Wait 100 ns for global reset to finish
                #100;

                a = 0;
                b = 1;

                // Wait 100 ns for global reset to finish
                #100;

                a = 1;
                b = 0;

                // Wait 100 ns for global reset to finish
                #100;

                a = 1;
                b = 1;
                // Wait 100 ns for global reset to finish
                #100;
                // Add stimulus here

        end

endmodule
```

**Simulation**



**Result: Test bench for half adder is simulated successfully.**

## Week-7
## Experiment – 1

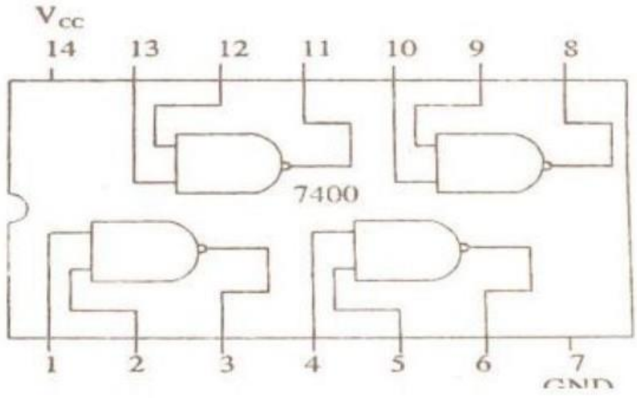
**Aim :**

To Realize and verify the function of clocked SR Flip-Flop using NAND logic gates.

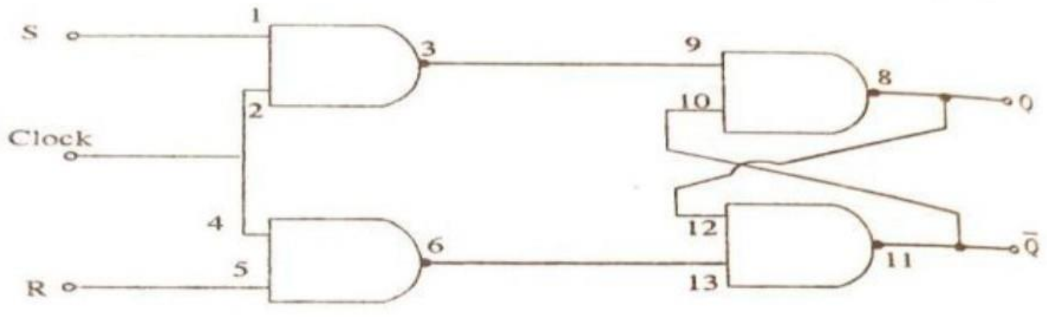
**Apparatus Required :**

- Digital trainer kit
- NAND gate IC 7400
- Patch chords.

**Pin Diagram of IC 7400 NAND Gate**



**Circuit Diagram of SR Flip Flop using only NAND gates**



***Note : Pin 16 = $V_{CC}$ and 7 = GND for IC 7400***

**Truth Table :**

| Clock | $Q_n$ Initial State | S | R | $Q_{n+1}$ Next state |
|---|---|---|---|---|
| 0 | X | X | X | No Change |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | Indeterminate |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | Indeterminate |

**Procedure :**

1. Check the working of all the components.
2. Insert the appropriate IC into the IC base.
3. Make connections as shown in the circuit diagram.
4. Apply the clock pulse manually, verify the Truth Table and observe the outputs.

**Result :**

Clocked SR FF is constructed using NAND gates and verified its functionality.

# Week-7
# Experiment – 2

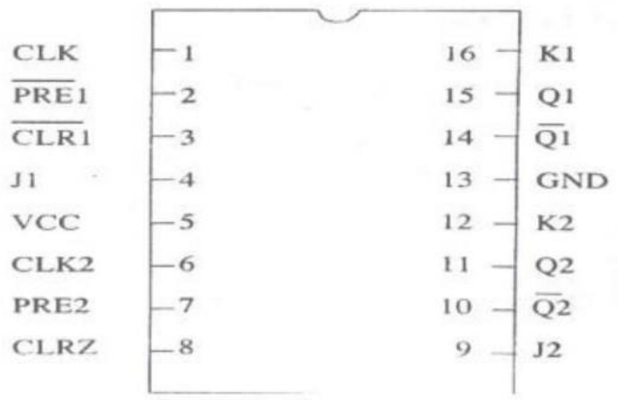
**Aim :**

To Realize and verify the truth table of T and D Flip-Flop using JK flip flop IC 7476.

**Apparatus Required :**

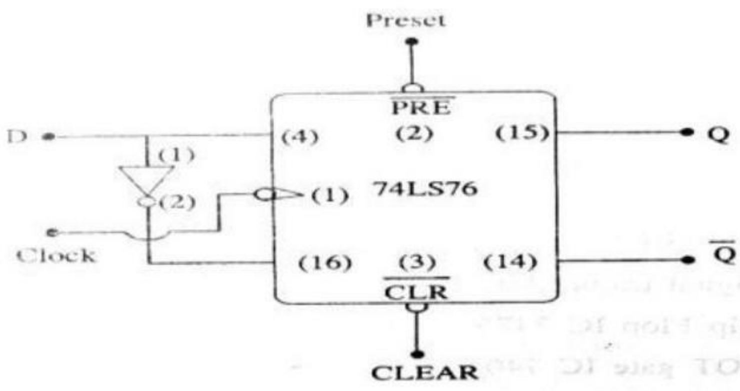
- Digital trainer kit
- Flip Flop IC 7476
- NOT gate IC 7404
- Patch chords.

**Pin Diagram of IC 7476 JK Flip Flop**



CLK 1
$\overline{\text{PRE1}}$ 2
$\overline{\text{CLR1}}$ 3
J1 4
VCC 5
CLK2 6
PRE2 7
CLRZ 8
16 K1
15 Q1
14 $\overline{\text{Q1}}$
13 GND
12 K2
11 Q2
10 $\overline{\text{Q2}}$
9 J2

**Circuit Diagram D Flip flop :**

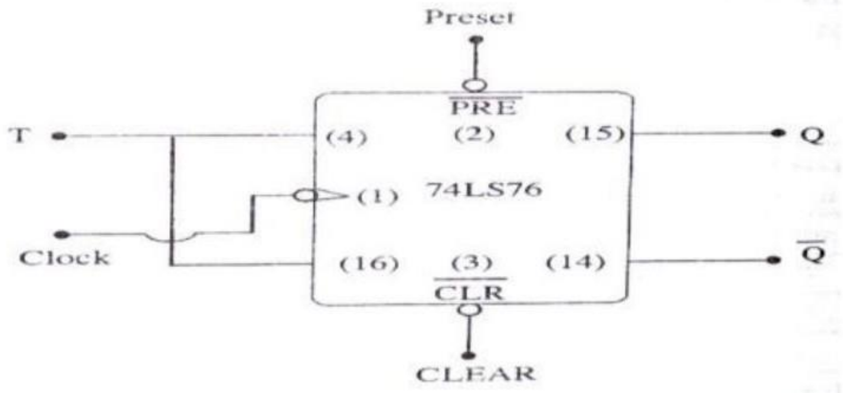


***Note : Pin 5 = $V_{CC}$ and 13 = GND for IC 7476. PRESET = $V_{CC}$, CLEAR = $V_{CC}$.***

**Truth Table of D Flip Flop**

| CLOCK | D | Q+ | $\overline{Q}$ + |
|---|---|---|---|
| 0 | X | Q | $\overline{Q}$ |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**Circuit Diagram T Flip Flop :**



***Note :*** *Pin 5 = $V_{CC}$ and 13 = GND*
*PRESET = $V_{CC}$, CLEAR = $V_{CC}$.*

| T | $Q_{n+1}$ |
|---|---|
| 0 | $Q_n$ |
| 1 | $\overline{Q_n}$ |

|  | Previous |  | Next |  |
|---|---|---|---|---|
| T | $Q_{Prev}$ | $Q'_{Prev}$ | $Q_{Next}$ | $Q'_{Next}$ |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

**Procedure :**

1. Check the working of all the components.
2. Insert the appropriate IC into the IC base.
3. Make connections as shown in the circuit diagram.
4. First connect CLR pin to GND and apply one clock pulse so that initially 0 is loaded
5. Then change CLR to VCC and apply clock pulses and input
6. Verify the Truth Table and observe the outputs.

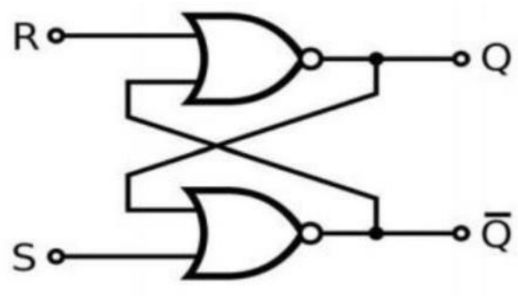
**Result :**

D-FF and T-FF are realized using JK FF IC 7476 and verified.

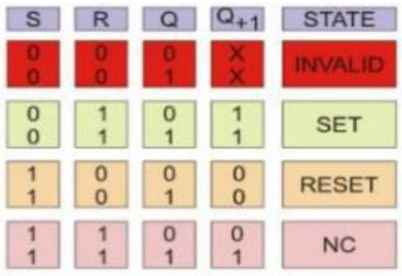
# Week-8
## Experiment – 1

Aim : Verilog description of SR flip flops using data flow modeling.

### Logic Diagram



### Truth table



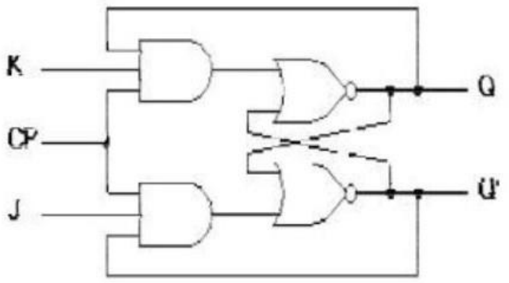
| S | R | Q | $Q_{+1}$ | STATE |
|---|---|---|---|---|
| 0 | 0 | 0 | X | INVALID |
| 0 | 0 | 1 | X | |
| 0 | 1 | 0 | 1 | SET |
| 0 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 0 | RESET |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 0 | NC |
| 1 | 1 | 1 | 1 | |

### Verilog Code

```
module srfp (s, r, q, q_n);
input s, r;
output q, q_n;
assign q_n = ~(s | q);
assign q = ~(r | q_n);
endmodule
```

## Verilog description of JK flip flop using behavioral modeling

### Logic Diagram



### Truth Table

| CP | J | K | $Q_{+1}$ | State |
|---|---|---|---|---|
| 1 | 0 | 0 | $Q_n$ | NO CHANGE |
| 1 | 0 | 1 | 0 | RESET |
| 1 | 1 | 0 | 1 | SET |
| 1 | 1 | 1 | $\overline{Q}_n$ | TOGGLES |

### Verilog Code:

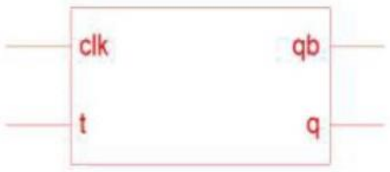
```
module jkflip_df (j,k,q,qn);
input j,k,q;
output qn;
wire w1,w2;
assign w1=~q;
assign w2=~k;
assign qn=(j & w1 | w2 &q);
endmodule
```

**Result: output for SR & JK Flip Flop is verified successfully**

# Week-8
# Experiment – 2

## Aim: Verilog description of T flip flops using behavioral modeling

### Block Diagram



### Truth Table

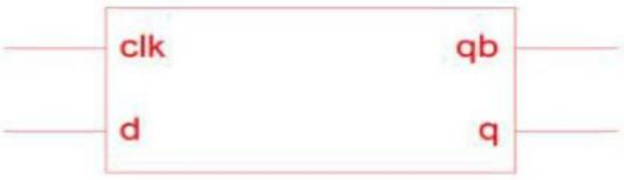
| CLK | T | Q | Qb | OPERATION |
|---|---|---|---|---|
| 0 | x | Q | Qb | No Change |
| 1 | 0 | Q | Qb | No Change |
| 1 | 1 | Qb | Q | Toggle |

### Verilog Code

```
module t_ff(t, clk, rst, q,
qb); input t, clk, rst;
output q,
qb;reg q,qb;
reg
[22:0]div;
reg clkdiv;
always @ (posedge
 clk) begin
div =
div+1'b1;
clkdiv =
div[22]; end
always @ (posedge
clkdiv) begin
if
 (rst==1
 )begin
 q=0;
  qb=1
  ;end
else
case
(t)
  1'b0 : begin q=q; qb=qb; end
   1'b1 : begin q=~(q); qb=~(qb);
    end default: begin end
endcase
end
endmodule
```

## Aim: Verilog description of D flip flops using behavioral modeling

### Block Diagram



### Truth Table

| CLK | D | Q | Qb |
|---|---|---|---|
| 0 | X | X | X |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Verilog Code

```
module d_ff(d, rst, clk, q,
qb); input d;
input       rst,
clk; output q
,qb;       reg
q,qb;
always@(posedge
clk) begin
if
(rst==1)
begin
q=0;
qb=1;
end
else
begin
q=d;
qb=~d
;end
end
endmodule
```

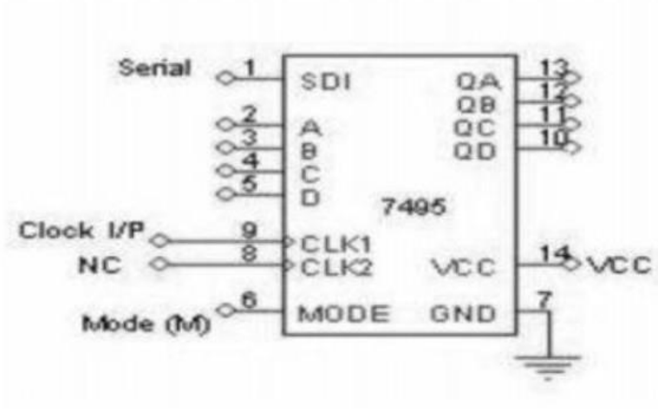
**Result: output for D & T Flip Flop is verified successfully**

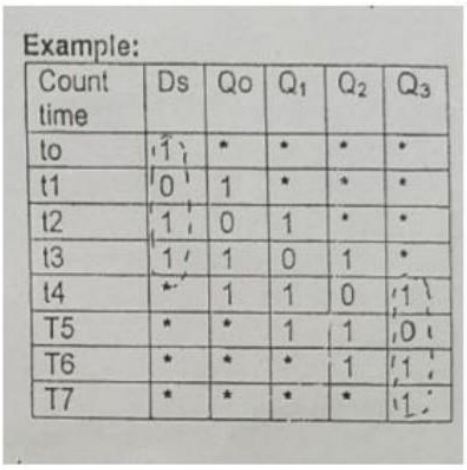
# Week-9
## Experiment – 1

Aim : To Construct and verify the working of the following using suitable IC in digital trainer kit -SISO, SIPO, PISO and PIPO (4-bit) shift registers.

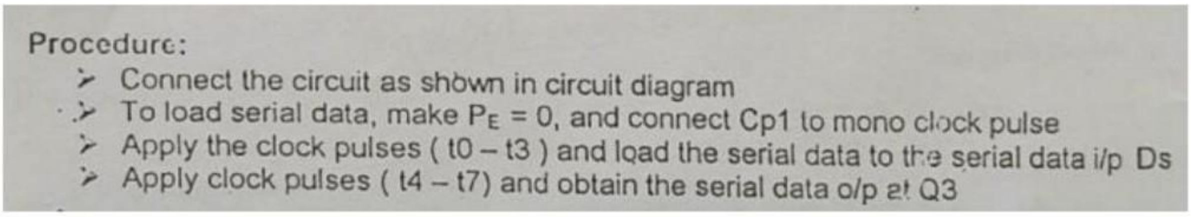
**SHIFT REGISTERS USING 7495**

**SISO:**



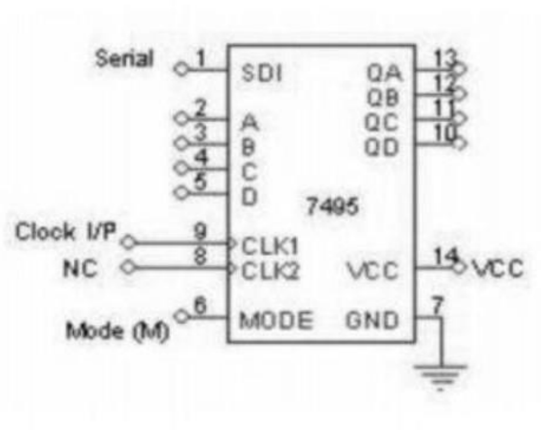

Example:

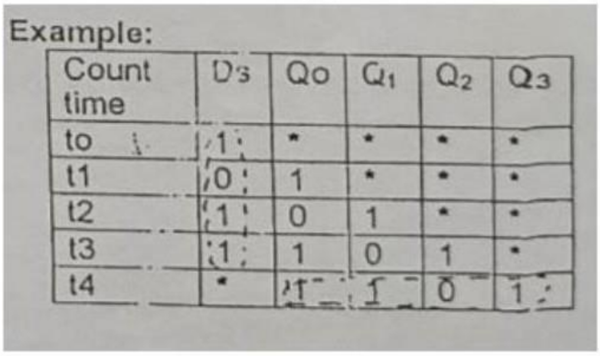
| Count time | Ds | Qo | $Q_1$ | $Q_2$ | $Q_3$ |
|---|---|---|---|---|---|
| to | 1 | • | • | • | • |
| t1 | 0 | 1 | • | • | • |
| t2 | 1 | 0 | 1 | • | • |
| t3 | 1 | 1 | 0 | 1 | • |
| t4 | • | 1 | 1 | 0 | 1 |
| T5 | • | • | 1 | 1 | 0 |
| T6 | • | • | • | 1 | 1 |
| T7 | • | • | • | • | 1 |


Procedure:

- Connect the circuit as shown in circuit diagram
- To load serial data, make $P_E = 0$, and connect Cp1 to mono clock pulse
- Apply the clock pulses ( t0 – t3 ) and load the serial data to the serial data i/p Ds
- Apply clock pulses ( t4 – t7) and obtain the serial data o/p at Q3

**SIPO(Right shift):**



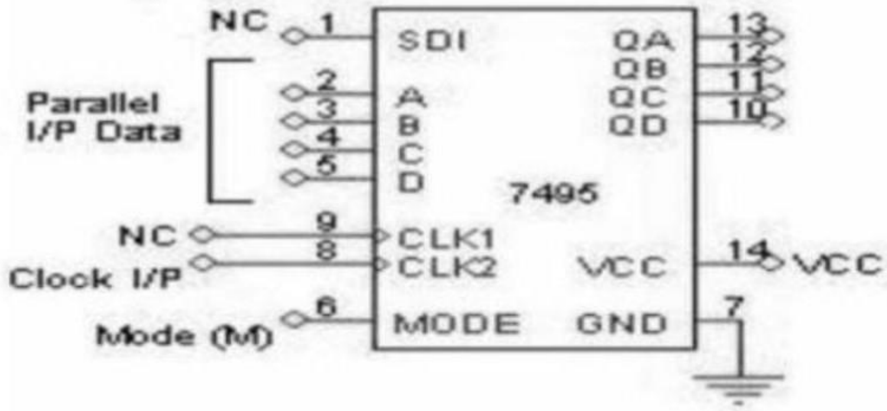

Example:

| Count time | Ds | Qo | $Q_1$ | $Q_2$ | $Q_3$ |
|---|---|---|---|---|---|
| to | 1 | • | • | • | • |
| t1 | 0 | 1 | • | • | • |
| t2 | 1 | 0 | 1 | • | • |
| t3 | 1 | 1 | 0 | 1 | • |
| t4 | • | 1 | 1 | 0 | 1 |

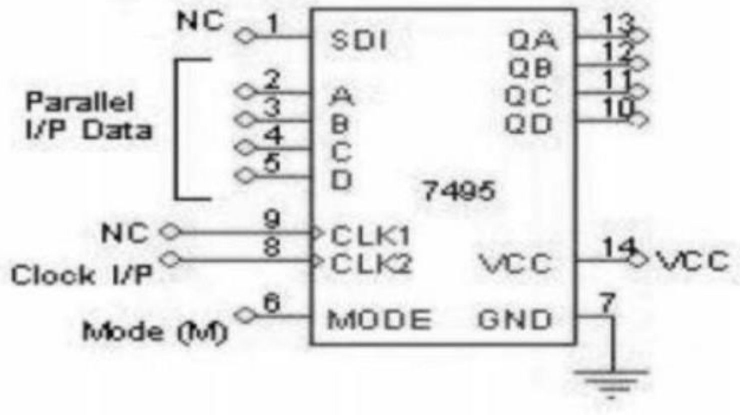
Procedure:

- Connect the circuit as shown in circuit diagram
- To load serial data, make $P_E = 0$, and connect Cp1 to mono clock pulse
- Apply the clock pulses ( t0 – t3 ) and load the serial data to the serial data i/p Ds
- Apply clock pulses ( t4) and obtain the parallel o/p at $Q_3$, $Q_2$, $Q_1$, $Q_0$

**PISO:**

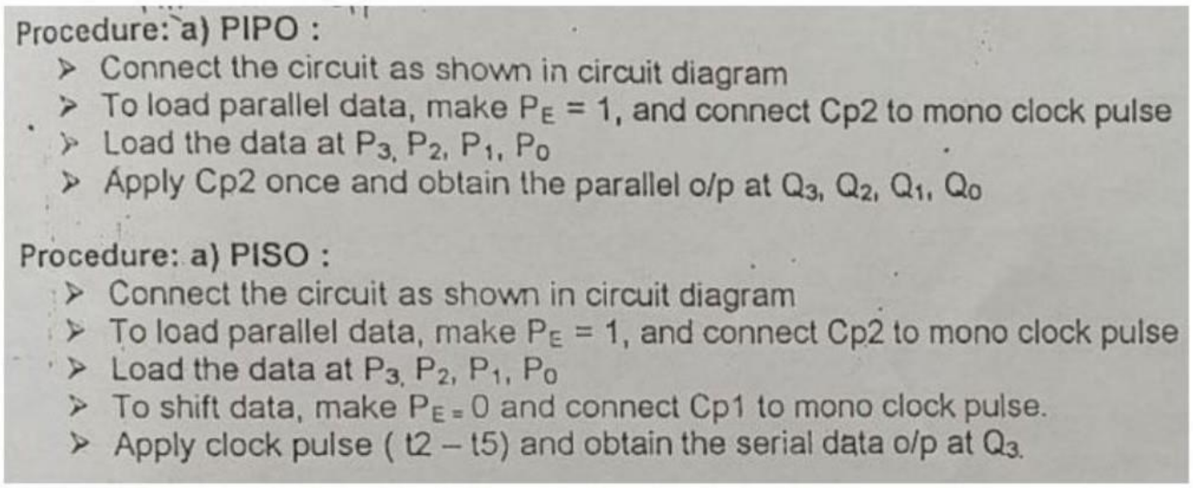


PIPO:



*Example:*

| Clock | Time | Qo | $Q_1$ | $Q_2$ | $Q_3$ | |
|---|---|---|---|---|---|---|
| Cp2 | to | * | * | * | * | |
| | t1 | 1 | 0 | 1 | 1 | → PIPO |
| Cp1 | t2 | * | 1 | 0 | 1 | |
| | t3 | * | * | 1 | 0 | → PISO |
| | t4 | * | * | * | 1 | |
| | t5 | * | * | * | * | |



Procedure: a) PIPO :

- Connect the circuit as shown in circuit diagram
- To load parallel data, make $P_E = 1$, and connect Cp2 to mono clock pulse
- Load the data at $P_3$, $P_2$, $P_1$, $P_0$
- Apply Cp2 once and obtain the parallel o/p at $Q_3$, $Q_2$, $Q_1$, $Q_0$

Procedure: a) PISO :

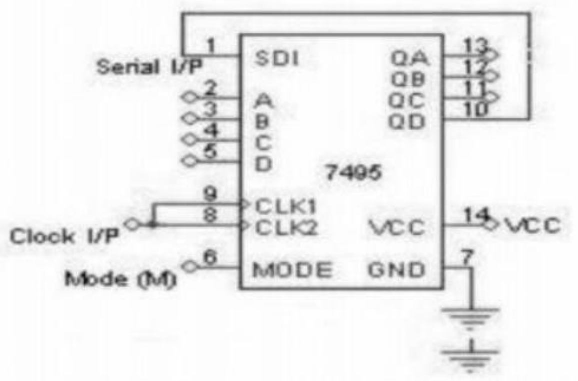
- Connect the circuit as shown in circuit diagram
- To load parallel data, make $P_E = 1$, and connect Cp2 to mono clock pulse
- Load the data at $P_3$, $P_2$, $P_1$, $P_0$
- To shift data, make $P_E = 0$ and connect Cp1 to mono clock pulse.
- Apply clock pulse ( t2 – t5) and obtain the serial data o/p at $Q_3$.

Result : 4-bit SISO, SIPO. PISO and PIPO are realized using shift register IC 7495 and tabulated the data movement in each mode

# Week-9
## Experiment – 2

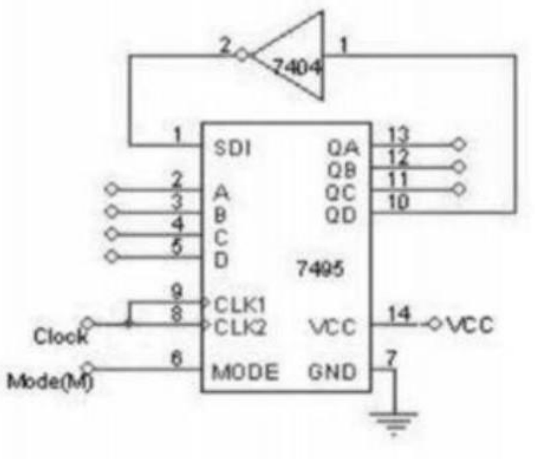
Aim: Construct and verify the working of the following using suitable IC in digital trainer kit - Ring and Johnson counter (4-bit).

### RING COUNTER USING 7495:



| Mode | Clock | QA | QB | QC | QD |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 2 | 0 | 1 | 0 | 0 |
| 0 | 3 | 0 | 0 | 1 | 0 |
| 0 | 4 | 0 | 0 | 0 | 1 |
| 0 | 5 | 1 | 0 | 0 | 0 |
| 0 | 6 | repeats | | | |

### JOHNSON COUNTER USING 7495



| Mode | Clock | QA | QB | QC | QD |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 2 | 1 | 1 | 0 | 0 |
| 0 | 3 | 1 | 1 | 1 | 0 |
| 0 | 4 | 1 | 1 | 1 | 1 |
| 0 | 5 | 0 | 1 | 1 | 1 |
| 0 | 6 | 0 | 0 | 1 | 1 |
| 0 | 7 | 0 | 0 | 0 | 1 |
| 0 | 8 | 0 | 0 | 0 | 0 |
| 0 | 9 | 1 | 0 | 0 | 0 |
| 0 | 10 | repeats | | | |

**Procedure:**

1.Check the working of all components

2.Insert the appropriate IC into IC Base

3.Make connections as shown in the circuit diagrams.

4. Apply the inputs, verify the Truth Table of the respective circuit and observe the outputs.

Result : Ring counter and Johnson counter using shift registers is implemented & output is verified

# Week-10
## Experiment – 1

**3 bit ripple counter using IC 7476 :**

**Aim :**

To Realize and verify the operation of 3-bit ripple counter using IC 7476.

**Apparatus Required :**

- Digital trainer kit
- JK Flip Flop IC 7476
- Patch chords.

**Pin Diagram of IC 7476 JK Flip Flop**

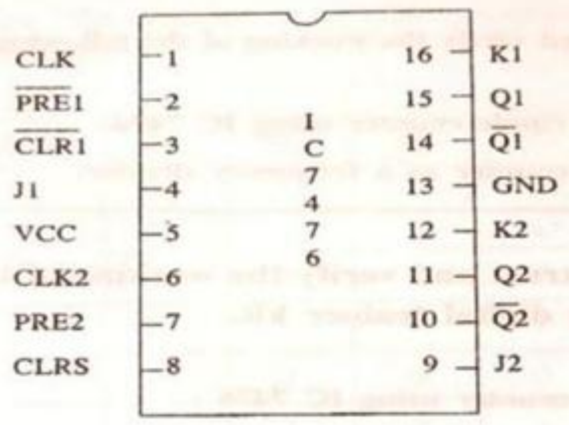
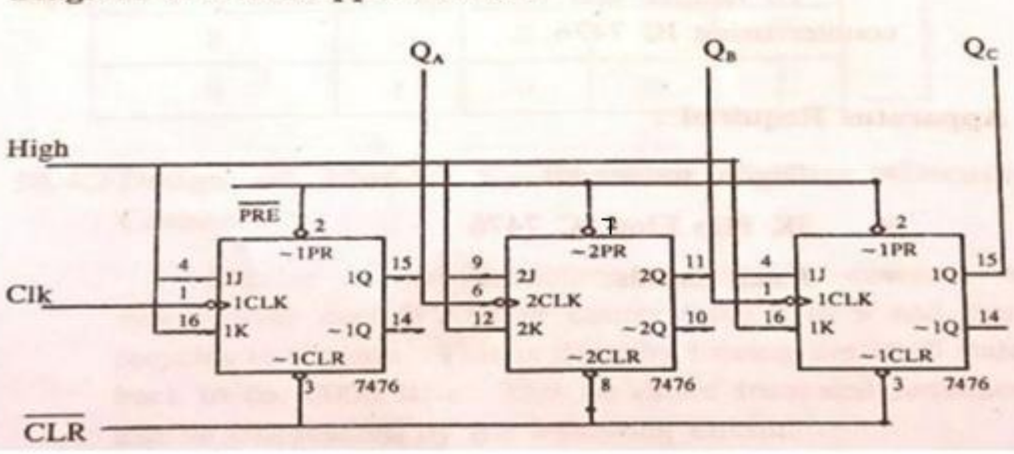
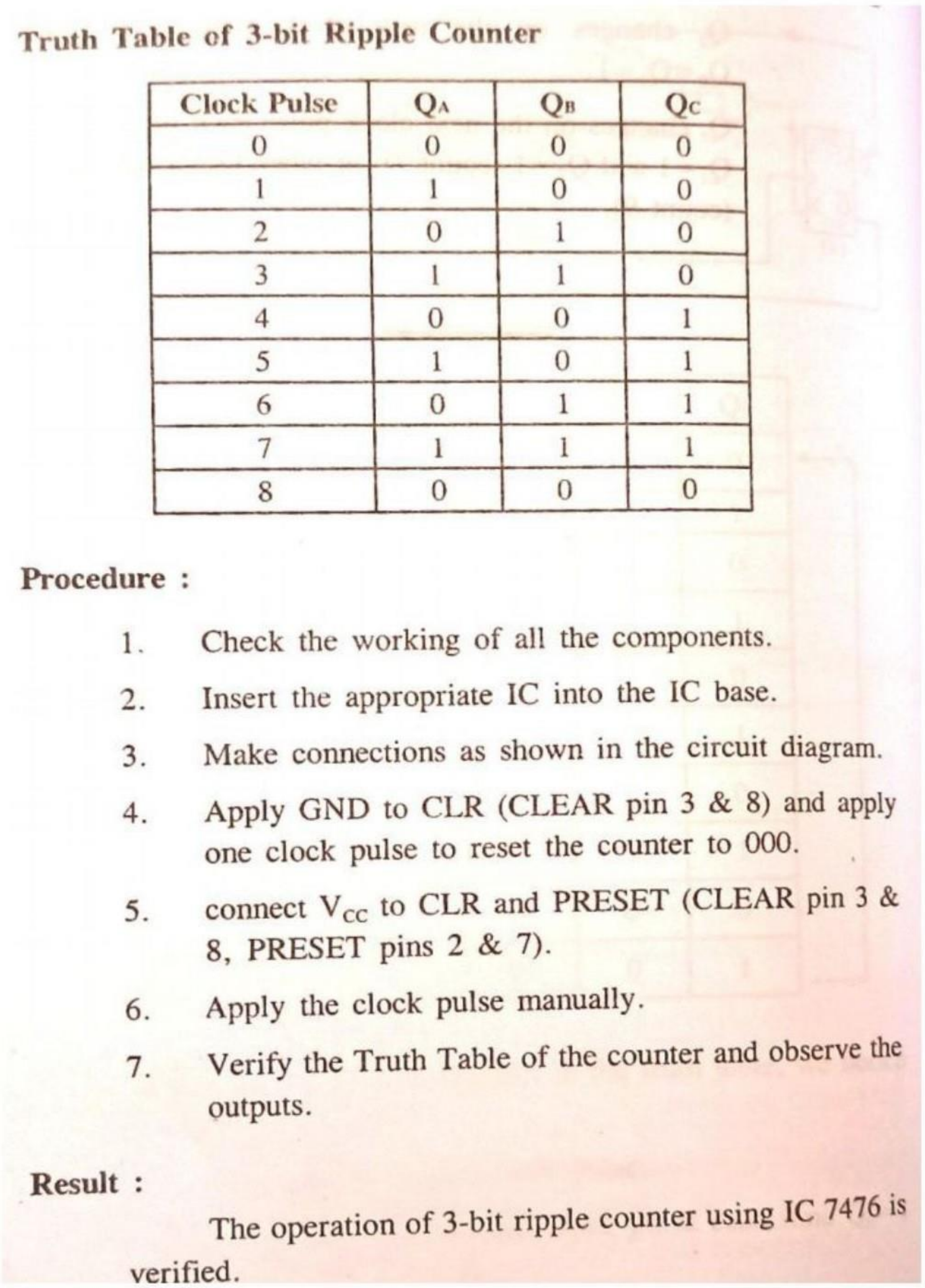


**Diagram of 3-bit Ripple Counter**





**Truth Table of 3-bit Ripple Counter**

| Clock Pulse | $Q_A$ | $Q_B$ | $Q_C$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 |
| 7 | 1 | 1 | 1 |
| 8 | 0 | 0 | 0 |

**Procedure :**

1. Check the working of all the components.
2. Insert the appropriate IC into the IC base.
3. Make connections as shown in the circuit diagram.
4. Apply GND to CLR (CLEAR pin 3 & 8) and apply one clock pulse to reset the counter to 000.
5. connect $V_{CC}$ to CLR and PRESET (CLEAR pin 3 & 8, PRESET pins 2 & 7).
6. Apply the clock pulse manually.
7. Verify the Truth Table of the counter and observe the outputs.

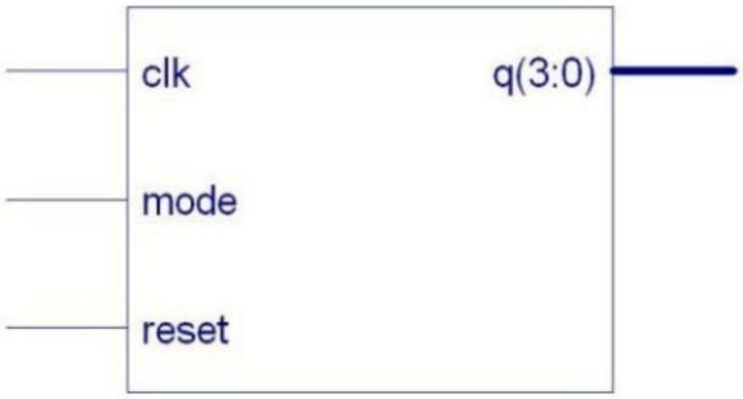
**Result :**

The operation of 3-bit ripple counter using IC 7476 is verified.

# Week-11
# Experiment – 1

**Aim: Verilog description for BCD up/down counter using behavior modeling**

**Block Diagram**



**Truth Table**

| Clock Count | Output bit Pattern QD | QC | QB | QA | Decimal Value |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 0 | 2 |
| 4 | 0 | 0 | 1 | 1 | 3 |
| 5 | 0 | 1 | 0 | 0 | 4 |
| 6 | 0 | 1 | 0 | 1 | 5 |
| 7 | 0 | 1 | 1 | 0 | 6 |
| 8 | 0 | 1 | 1 | 1 | 7 |
| 9 | 1 | 0 | 0 | 0 | 8 |
| 10 | 1 | 0 | 0 | 1 | 9 |

Verilog Code:

```
module updown
(clk,reset,mode,q); input
clk,reset,mode;
output[3:0]q;
reg[3:0]q;
reg
[22:0]div;
reg clkdiv;
always @ (posedge
  clk) begin
div =
div+1'b1;
```

```
clkdiv =
div[22]; end
always @ (posedge
clkdiv) begin
 if (reset)
  q=4'b0000
  ;
else
if(mode)
q=q+1'b1;
else
 q=q-1'b1;
 end
 endmodule
```

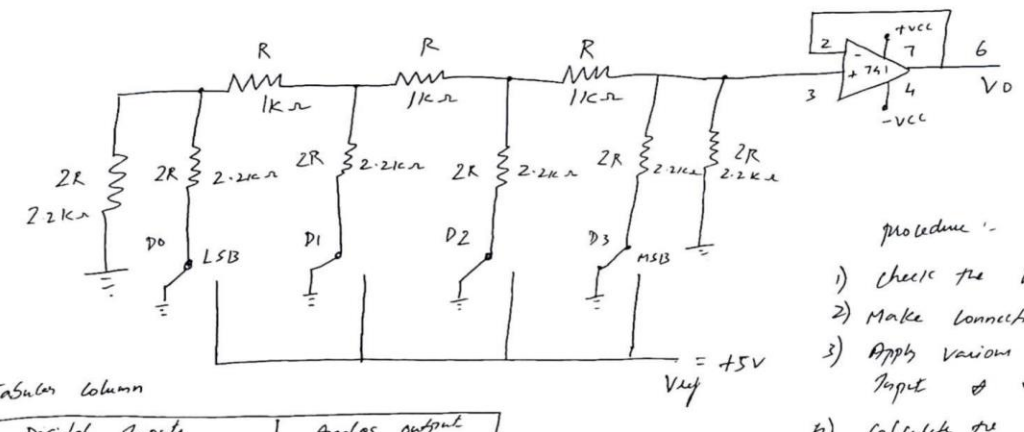
Result : Output for BCD up/down counter using behavior modeling is verified.

Week - 11

Experiment - 2

Aim:- Construct / Simulate and verify the working of R-2R DAC

Apparatus:- μA 741, 1KΩ, 2.2KΩ Resistors, Connecting wires & multimeter



Procedure:-

1) Check the working of all components
2) Make connections as shown in ckt diagram
3) Apply various combinations of the Input & verify the output
4) Calculate the ~~th~~ theoritical value and verify practical values.

Tabular column

| Digital Inputs | Analog output |
|---|---|
| $D_3$ $D_2$ $D_1$ $D_0$ | 0 |
| 0 0 0 0 | 0 |
| 0 0 0 1 | 0.2083 |
| 0 0 1 0 | 0.4166 |
| 0 0 1 1 | 0.6249 |

Result:- The working of R-2R DAC is verified successfully

Analog output voltage for a 4 bit DAC is given as

Example 1000

$$V_0 \quad [2^3 D_3 + 2^2 D_2 + 2^1 D_1 + 2^0 D_0]\left[\frac{V_R}{2^n}\right]\times\left[\frac{2K}{3R}\right]$$

$$= [2^3\cdot 1 + 0 + 0 + 0]\left[\frac{5}{16}\right]\left[\frac{2}{3}\right]$$

$$V_0 = \overset{1}{\cancel{8}} \times 5/\cancel{16}_{2} \times \cancel{2}/3 = 5/3 \text{ V [Theoritical]}$$

$$V_0 = \frac{V_{ref}\cdot R}{R + 2R} = \frac{5\times R}{3R} = 5/3 \text{ V (practical)}$$

$\underline{1.6V}$

$D_0$ being LSB & $D_3$ Being MSB bit