**Experiment 4**

**Implementation of 2 input logic gates using NOR and NAND gates**

**Objective**

In this lab students will learn

* How NOR gate can be used as universal gate
* How NAND gate can be used as universal gate

**Components**

* IC74LS02×1
* IC74LS04×1
* IC74LS00×1
* AM2000 TRAINER
* Multimeter
* Cutter
* Single core wire
* Pair of Pliers

**Theory**

**Universal Gate**

A universal gate is a gate which can implement any Boolean function without the need of any other gate type. The NAND and NOR gates are universal gates. In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families. In fact, an AND gate is typically implemented as a NAND gate followed by an inverter not the other way around. Likewise, an OR gate is typically implemented as a NOR gate followed by an inverter not the other way around.

**Procedure:**

1. Connect the AM2000 trainer to the 220V AC power supply
2. Turn on the trainer and verify the voltage of the power supply usin the multimeter. It should be +5V exactly.
3. Install IC74LS02 and IC74LS04 on the trainer’s board.
4. Wire the circuit according to the diagram by consulting gate IC’s diagram from previous manuals.
5. Use any of the two logic switches from S2 to S9 for inputs A and B respectively.
6. For output indication use any of the LED’s from L0 to L15.
7. Supply the +5v and GND to the pins 14 and 7 of the IC respectively.
8. Test all the possible combinations of inputs and verify the output according to the truth tables of AND, OR, NOT, and NAND gate.
9. Now install IC 74LS00 and 74LS04 on the trainer and test how hand gate is used as universal gate.
10. Make truth table according to the results.

**In case of trouble:**

1. Check the power supply.
2. Check the Vcc and GND at pins 14 and 7 respectively.
3. Check all the wire connections.
4. Check the circuit wiring and remove the breaks.
5. Check the IC using truth table.

**Truth Table: [6]**

**NOR GATE IMPLEMENTATION**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input (A)** | | **Input (B)** | | **A+B** | **((A+B)’)’=A+B** | |
| **OR** | | | | |  | |
|  | |  | |  |  | |
|  | |  | |  |  | |
|  | |  | |  |  | |
|  | |  | |  |  | |
| **AND** | | | | |  | |
| **A** | **B** | **A’ B’** | | **(A’+B’)’** | **AB** | |
|  |  |  |  |  |  | |
|  |  |  |  |  |  | |
|  |  |  |  |  |  | |
|  |  |  |  |  |  | |
| **NAND** | | | | |  | |
| **A** | **B** | **A’** | | **B’** | **(A’+B’)’=AB** | **(AB)’** |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |

**NAND GATE IMPLEMENTATION**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input (A)** | | **Input (B)** | | **(A+B)’** | **((A+B)’)’=AB** | |
| **AND** | | | | |  | |
|  | |  | |  |  | |
|  | |  | |  |  | |
|  | |  | |  |  | |
|  | |  | |  |  | |
| **OR** | | | | |  | |
|  | **B** | **A’ B’** | | **(A’B’)’** | **A+B** | |
|  |  |  |  |  |  | |
|  |  |  |  |  |  | |
|  |  |  |  |  |  | |
|  |  |  |  |  |  | |
| **NOR** | | | | |  | |
| **A** | **B** | **A’** | | **B’** | **(A’B’)’=A+B** | **(A+B)’** |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |
|  |  |  | |  |  |  |

**Exercise:**

1. **Draw the equivalent logic diagram of NOT, AND, OR, and NAND gate using 2-input NOR gate. [4]**

**NOT AND**

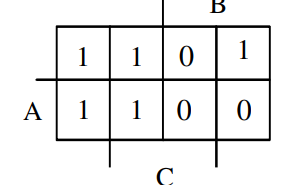
**OR NAND**

1. **Draw the equivalent logic diagram of NOT, AND, OR, and NOR gate using 2-input NAND gate. [4]**

**NOT AND**

**OR NOR**

1. **Implement the following function using multilevel NAND gate implementation. [3]**
2. **Implement the function in NOR-OR & AND-NOR logic. [4]**

****

**Invert-OR is equivalent to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. [2]**

**Invert-AND is equivalent to \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.**

**Conclusion: [2]**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**