**ABES Engineering College, Ghaziabad**

# Affiliated to Dr. APJ AKTU Lucknow

**Department of Computer science and engineering**



**LAB MANUAL**

**Session 2020-21 (Odd Semester)**

**Name** : Vipin Kumar.

**Subject Name :** Computer Organization Lab

**Roll No.** : 1900320100198

**Subject Code :** KCS - 352

**Semester :** B.Tech. (CSE IIIrd Sem)

**Vision**

The Department of Information Technology endeavours to be recognized globally for outstanding education and research leading to well qualified engineers, who are innovative, entrepreneurial and successful in advanced fields of IT to cater the ever-changing industrial demands and social needs.

**Mission**

To create a healthy environment for the development of innovative professional and researchers to fulfil its commitment to research & education of the highest quality with industry requirement and social acceptance at large.

**Program Educational Objectives (PEOs)**

The broader objective of IT Department is to transform the students admitted to the program into globally competitive professionals having sound knowledge of fundamentals and capable with core competency in logical & computation analysis to solve Engineering Problems. The detailed Program Educational Objectives (PEOs) are as follows:

**PEO 1:** To impart exhaustive knowledge to the students in all the sub – domains of Information Technology and the fast-evolving IT tools to:

a) Take up key assignments in IT and associated industries.

b) Undertake and excel in higher studies and Research & Development in IT related fields and Management.

**PEO 2:** To impart solid foundation in Computational Intelligence, Science and Interdisciplinary courses.

**PEO 3:** To design & develop novel products and innovative solutions for real life problems in Information Technology field and related domains.

**PEO 4:** To inculcate a conviction in the students to believe in self, impart professional and ethical attitude, nurture to be an effective team member, infuse leadership qualities, build proficiency in soft – skills and abilities to relate Engineering with the social issues.

**PEO 5:** To provide a conducive and disciplined Academic environment, quality of teaching with innovative & modern methods of pedagogy establishing the relevance of technical education as per the needs of the industry and society at large.

**Program Outcomes (POs)**

**PO-1: Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO-2: Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**PO-3: Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO-4: Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO-5: Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

**PO-6: The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO-7: Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO-8: Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO-9: Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO-10: Communication**: Communicate effectively on complex engineering activities with

the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO-11: Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO-12: Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**Program Specific Outcomes (PSOs)**

**PSO-1:** An ability to specify, analyse & design a usable computing system that efficiently utilizes system software and hardware to cover current user requirement in a socially and economically acceptable form.

**PSO-2:** An ability to state, design and implement a secure and reliable information communication system by using concepts of computer networks, network security, information theory and parallel algorithm.

**PSO-3:** An ability to state, design and implement knowledge-based discovery and machine-based learning in computer system by using the various concepts of soft computing, neural networks, image processing, digital system design and artificial intelligence.

**PSO-4:** An ability to analyse and design an efficient information management system which uses the concepts & tools of databases, data mining and information coding and deliver technological solutions for the end users.

**Course Objectives (Cos)**

**CO1:** Understand the simulator-logisim.

**CO2:** To analysis and verify the circuits like logic gates, adder etc.

**CO3:** To implement decoder, registers, ALU.

**CO4:** Design data path, control unit.

**CO5:** To write an algorithm, and check scalability.

**Tools Used:**

Logisim Simulator

**LIST OF EXPERIMENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.**  **No.** | **Experiment Name** | **CO**  **Mapping** | **Page No.** |
| 1. | Implementation of basic logic gates. | CO1 |  |
| 2. | Designing large Memory Unit using smaller size memory units. | CO4 |  |
| 3. | Implementi.ng 4\*1 and 8\*1 multiplexer. | CO1 |  |
| 4. | Implementing HALF ADDER, FULL ADDER using basic logic gates. | CO1 |  |
| 5. | Implementing Binary -to -Gray, Gray -to -Binary code conversions. | CO1 |  |
| 6. | Design the data path of a computer from its register transfer language. | CO1 |  |
| 7. | Design a bus system using Multiplexer for data transfer from register. | CO1 |  |
| 8. | Design of an 8-bit ARITHMETIC LOGIC UNIT. | CO1 |  |
| 9. | Implement an array multiplier of 2\*2 and 4\*3 bit. | CO1 |  |
| 10. | Implement 4 bit binary incrementer and decrementer. | CO1 |  |

**Name of Student Faculty Sign**

( Vipin Kumar ) (Mr. Anmol Jain)

**EXPERIMENT NO. ….1….**

**PROBLEM STATEMENT:** Implementation of basic logic gates.

**APPRATUS REQUIRED:**

1.AND Gate

2. OR Gate

3. NOT Gate

4. NAND Gate

5. NOR Gate

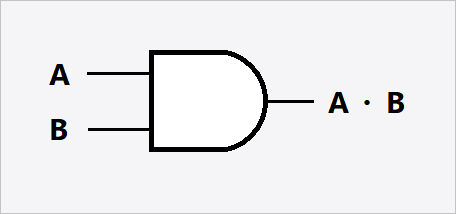
6. XOR Gate

**THEORY:** Logic gate is a circuit with one output and two or more input channels. An outputsignal occurs only for certain combination of input signals. Logic circuits are used to perform various computer functions.

**AND GATE:** It has two or more input and one output. It follows the rule “**The output of an AND Gate assumes 1 state if all its inputs assumes 1 state”.** Boolean Expression for “AND Gate” is:

**Y=A.B**

Symbol is:

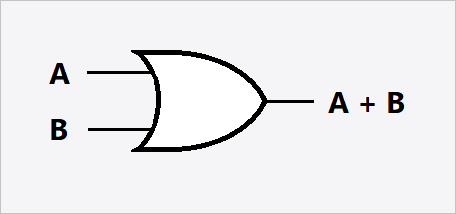


**OR GATE:** It has two or more inputs and one output. It follows the rule “**The output of an OR Gate assumes 1 state if one or more inputs assume 1 state”.**

Boolean Expression for “OR Gate” is:

**Y=A+B**

Symbol is:

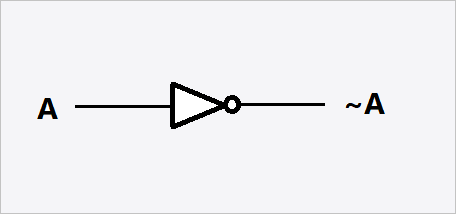


**NOT GATE:** It has only one input and one output. It follows the rule “**The output of a NOT Gate is 1 if input is 0 and vice-versa”.**

Boolean Expression for “NOT Gate” is:

**Y= ~A**

Symbol is:

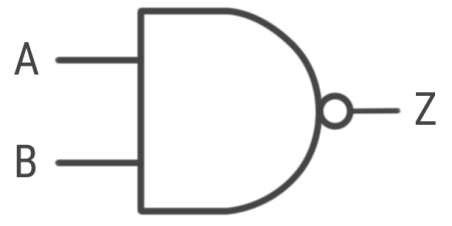


**NAND GATE:** It is the combination of “AND” and “NOT” gates. When the output of an “AND” gate is connected to the input of a “NOT” gate, the circuit is called “**NAND Gate**” and follow the rule **“The output of a NAND Gate is 1 only if at least one of its inputs is 0”.**

Boolean Expression for “NAND Gate” is:

**Y= ~ (A.B)**

Symbol is:

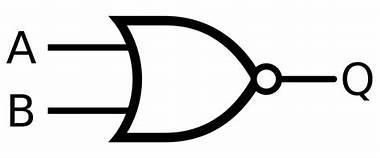


**NOR GATE:** It is the combination of “OR” and “NOT” gates. When the output of an OR gate is connected to the input of a NOT gate, the circuit is called **“NOR Gate”** and follows the rule **“The output of a NOR Gate is 1 only if the both inputs are zero”.**

Boolean Expression for “NOR Gate” is:

**Y= ~ (A+B)**

Symbol is:

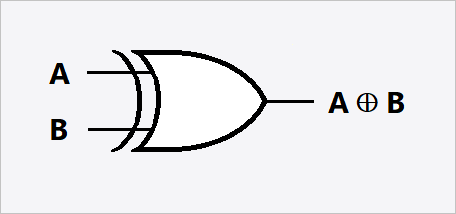


**XOR GATE:** The EX- OR gate operation is widely used in digital circuit. It is not abasic operation and can be performed using the basic gates – AND, OR, and NOT or universal gates NAND or NOR. It is similar to the basic OR gates except that **“The output is low, when** **both the inputs are same (0 & 1). The output of the gate is high (1) when both input are differ”**.

Boolean Expression for “XOR Gate” is:

**Y = A (~B) + (~A) B**

Symbol is:



**TRUTH TABLE:**

**AND GATE:**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y=A.B** |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

**OR GATE:**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Y=A+B** |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

**NOT GATE:**

|  |  |
| --- | --- |
| **A** | **~A** |
| 0 | 1 |
| 1 | 0 |

**NAND GATE:**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **~ (A.B)** |
| 0 | 0 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

4

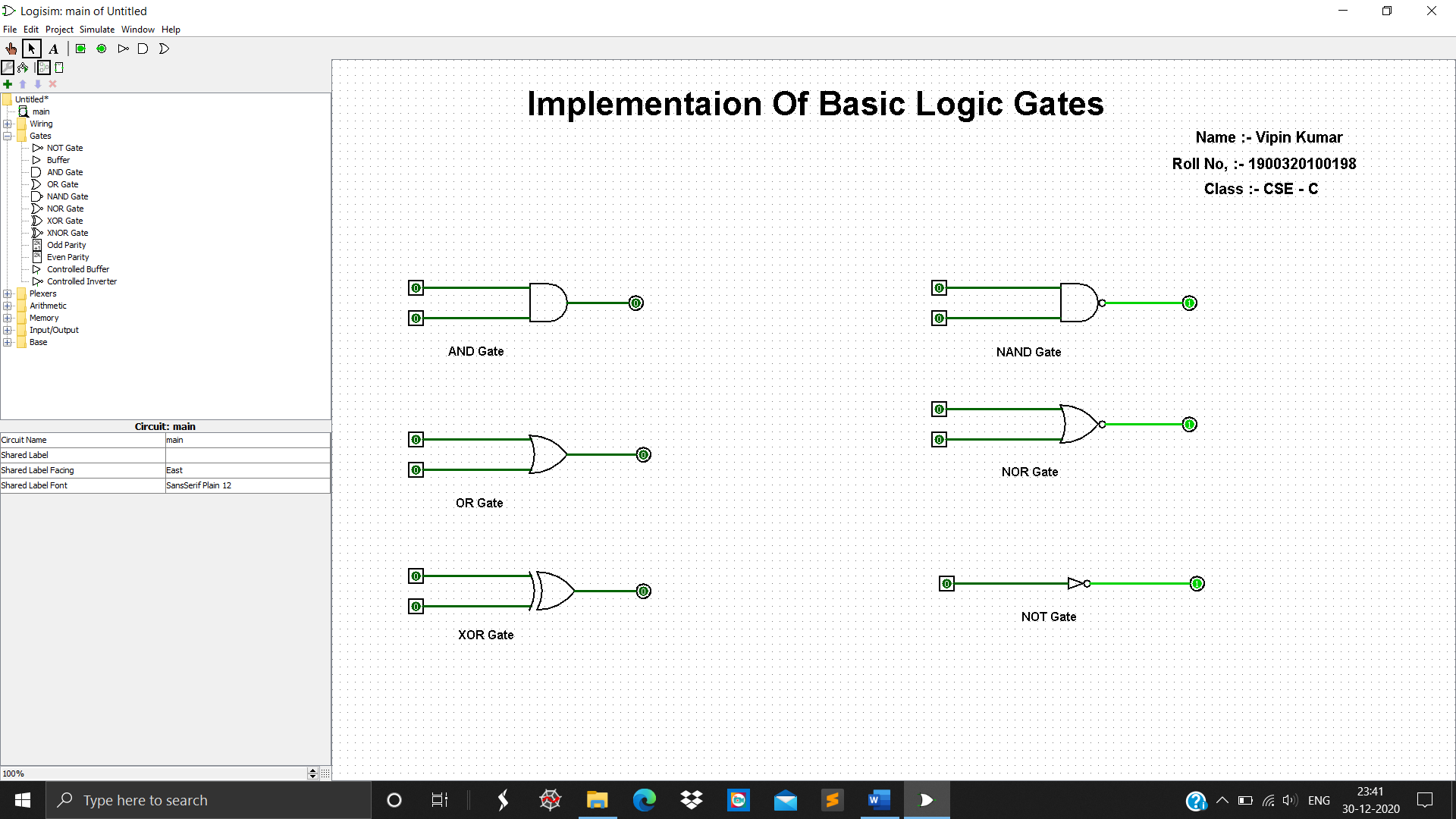
**NOR GATE:**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **~ (A+B)** |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |

**XOR GATE:**

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A(~B)+(~A)B** |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

**IMPLEMENTATION (LOGISIM):**

****

**RESULT:**

In AND GATE when pass values 1 and 0 output is 0

In OR GATE when pass values 1 and 0 output is 1

In NOT GATE when pass value is 1 output is 0

In NAND GATE when pass values 1 and 1 output is 0

In NOR GATE when pass values 1 and 0 output is 0

In XOR GATE when pass values 1 and 0 output is 1

All values are matched with tables.

**EXPERIMENT NO. ….2….**

**PROBLEM STATEMENT:** Designing large Memory Unit using smaller size memoryunits.

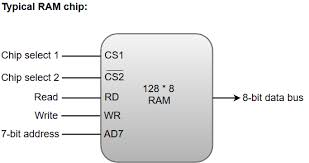
**APPRATUS REQUIRED:**

1. Decoder

2. 4 RAMs

**THEORY:** A memory unit is a collection of storage cells together with associated circuits needed to transform information in and out of the device. Memory cells which can be accessed for information transfer to or from any desired random location is called random access memory (RAM).

The block diagram of a memory unit is:



A basic RAM cell has been provided here as a component which can be used to design larger memory units.

Formula for numbers of IC required is:

**No. of IC required = Size of memory unit (required)**

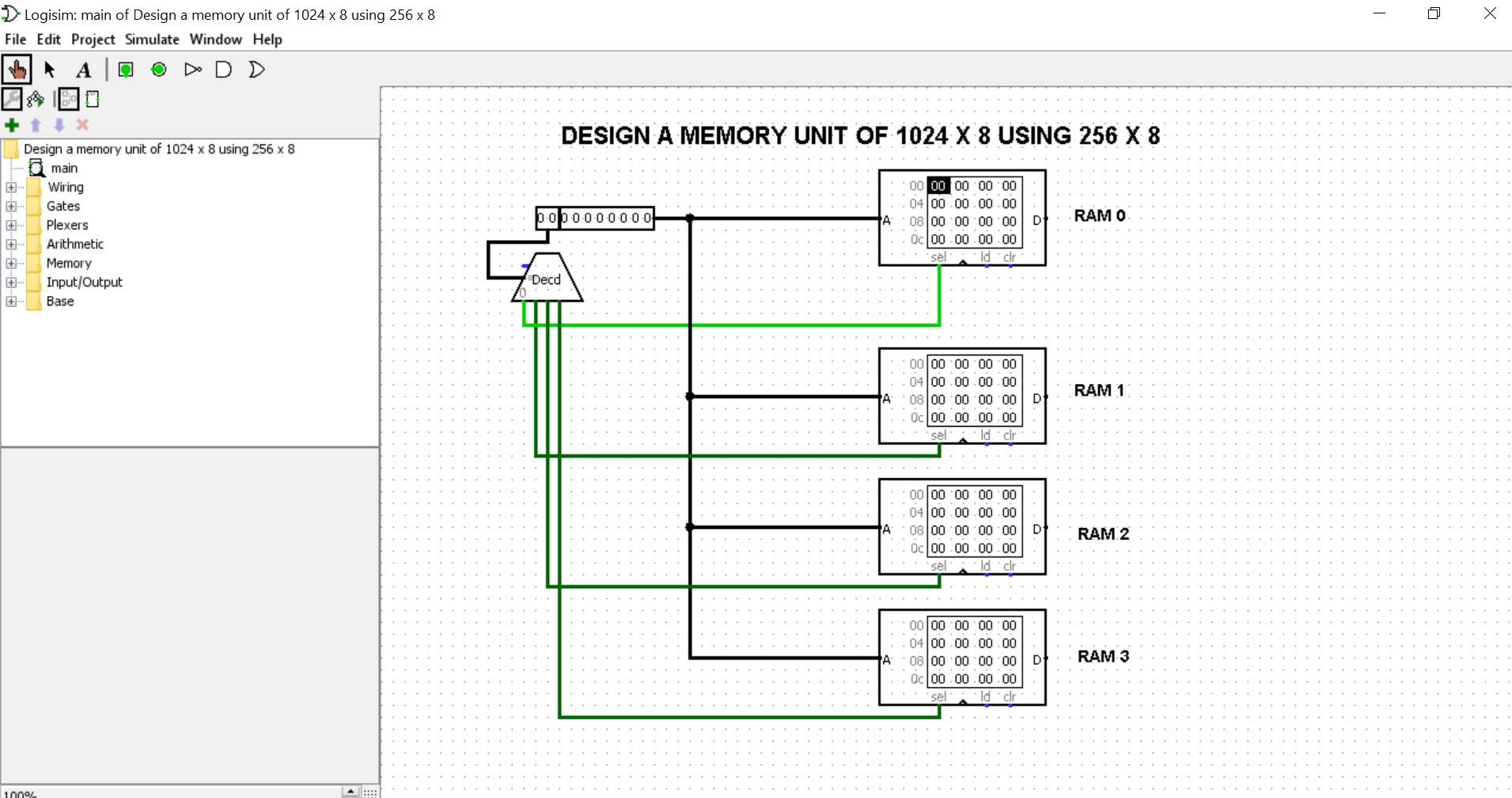
**Size of memory unit (given)**

Formula for Size of Decoder is:

**Size of Decoder = n \* 2^n**

Where n = no. of address line required – no. of address line given

**IMPLEMENTATION (LOGISIM):**



**RESULT:**

For Designing a larger size memory we need to fin no. of IC required. In this case,

No. of IC required is = 1024\*8 / 256\*8 = 4

So, we need 4 RAM in this circuit.

No. of Address Line for 1024 =10

No. of Address Line for 256 = 8

n = 10-8 = 2

size of decoder = 2 x 2^2 = 2 x 4.

**EXPERIMENT …3…**

**PROBLEM STATEMENT:** Implementation of 4\*1 and 8\*1 multiplexer.

**APPARATUS REQUIRED** :

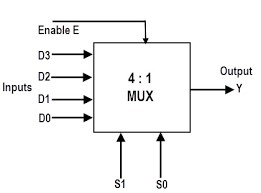
1. Bread board Trainer Kit.

1. IC-7411 (3 input AND Gate).
2. IC-7432 (OR Gate).
3. IC-7404 (NOT Gate).
4. IC-7410 (3 input NAND Gate).
5. IC-74153. MUX IC.
6. Connecting wires etc.

**THEORY**: Multiplexer means transmitting a large number of information units over a smallernumber of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally there are 2n input lines and n- selection lines whose bit combination determine which input is selected.

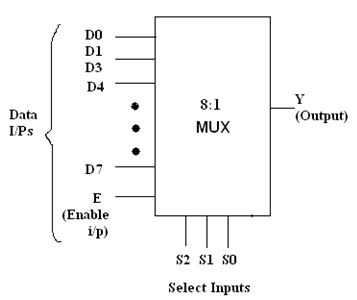
##### **BLOCK DIAGRAM OF MUX:**

**4×1 MUX**



#### 8×1 MUX

#### 

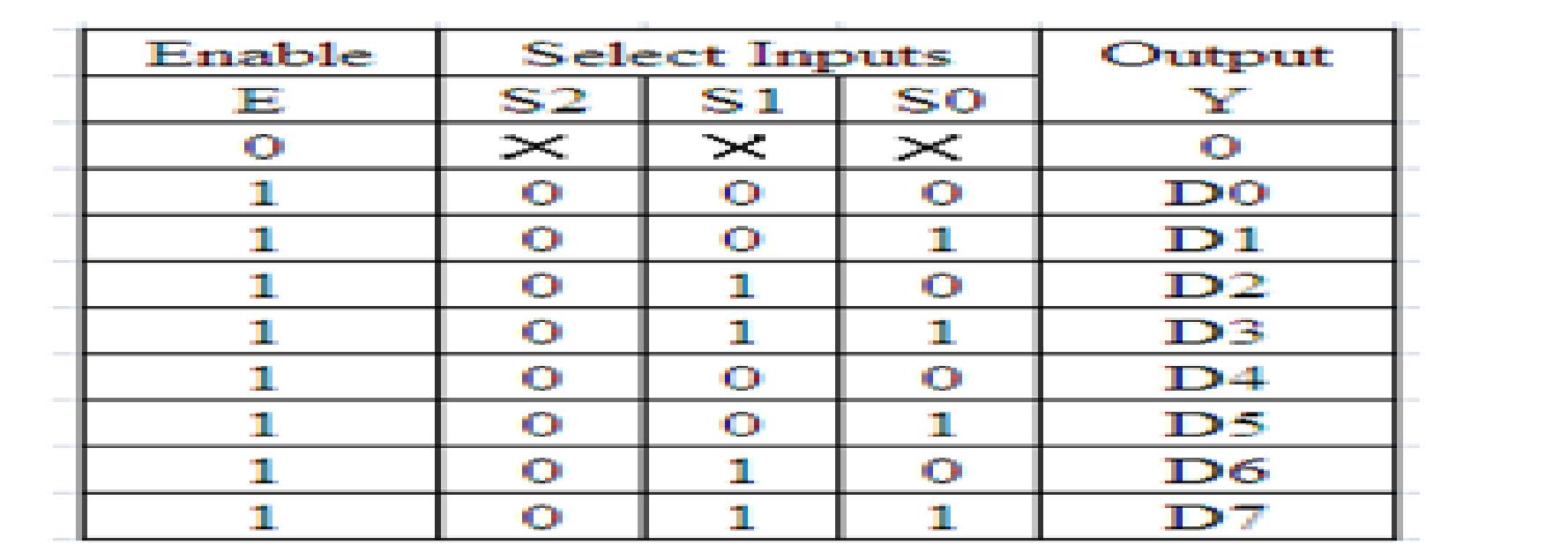


**OBSERVATION TABLE:**

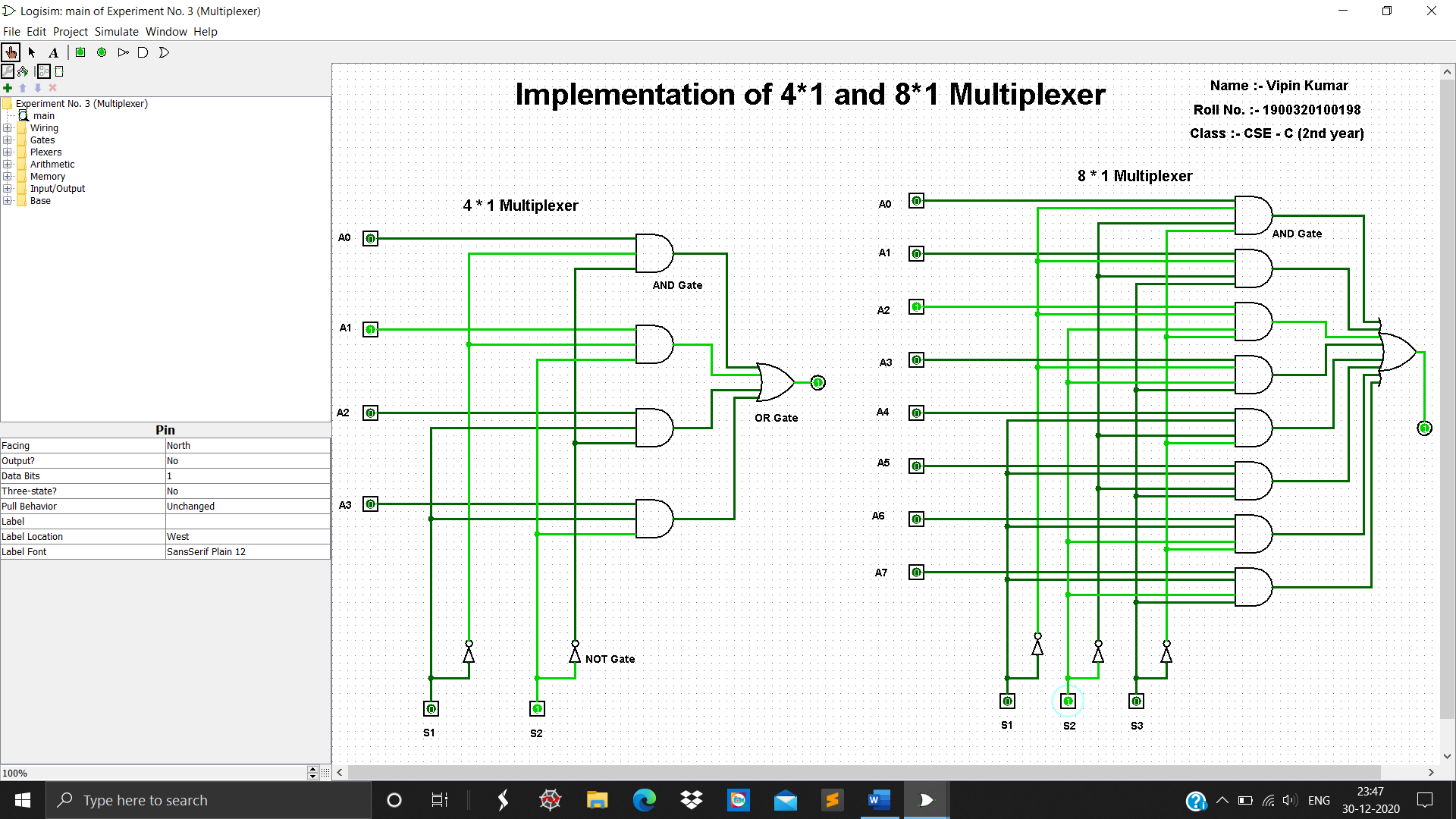
##### **FOR 4:1 MUX**

|  |  |  |  |
| --- | --- | --- | --- |
| INPUT | SELECT LINE | | OUTPUT |
| G | A | B | Y |
| 1 | 0 | 0 | C0 |
| 1 | 0 | 1 | C1 |
| 1 | 1 | 0 | C2 |
| 1 | 1 | 1 | C3 |

##### **FOR 8:1 MUX**



**IMPLEMENTATION (LOGISIM):**

****

**RESULT:**  Multiplexer Tables are verified.

**EXPERIMENT NO….4**

**PROBLEM STATEMENT:** Implementation of HALF ADDER, FULL ADDER using basic logic gates

**APPRATUS REQUIRED:**

1. Bread board Trainer Kit.

2. IC-7486 (X-OR Gate).

3. IC-7408 (AND Gate).

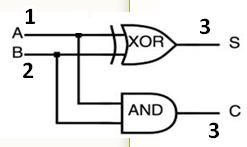
4. IC-7432 (OR Gate).

5. IC-7404(NOT Gate).

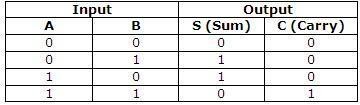
6. Connecting wires etc

**THEORY:**

**HALF ADDER:** A combinational circuit which performs the addition of two bits is called half adder. The input variables designate the augend and the add end by whereas the output variables produce the sum and carry bits

****

**TRUTH TABLE:**

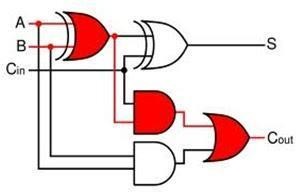
****

**THEORY:**

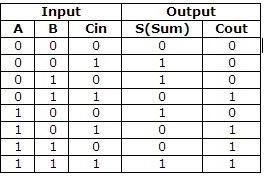
**FULL ADDER:**

Adder circuit is a combinational digital circuit that is used for adding two numbers. A typical Adder circuit produces a sum bit (denoted by S) and carry bit (denoted by C) as the output.

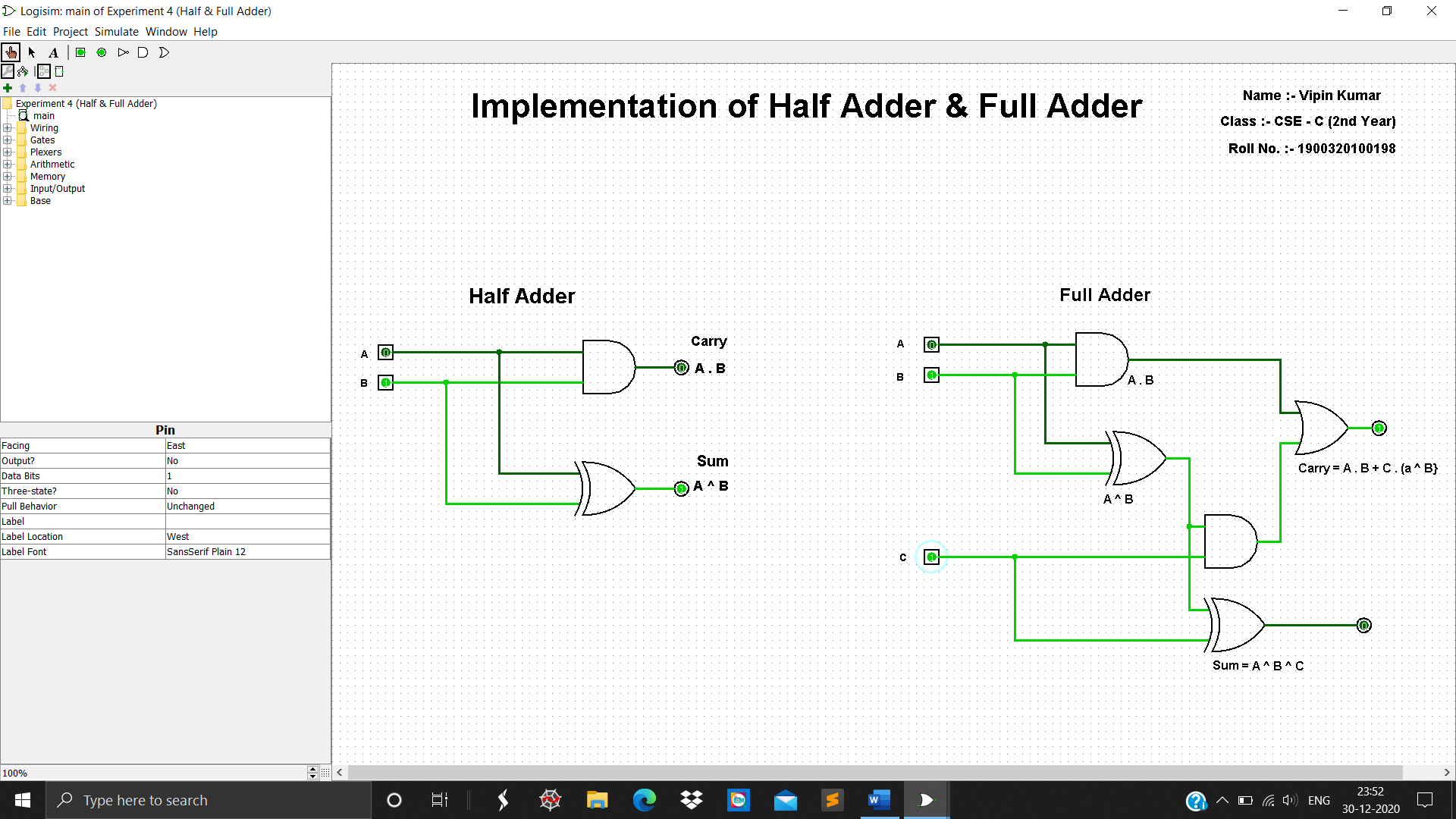
A full adder is a combinational circuit that forms the arithmetic sum of input, it consists of three inputs and two outputs. A full adder is useful to add three bits at a time but a half adder cannot do so. In full adder sum output will be taken from X-OR Gate, carry, output will be taken from OR Gate

****

**TRUTH TABLE:**

****

**LOGISIM RESULT:**

****

**RESULT:** All output verified the result of truth table

**EXPERIMENT NO….5**

**PROBLEM STATEMENT:** Implementing Binary -to -Gray, Gray -to -Binary code conversions.

**APPARATUS REQUIRED**: 1. Bread board Trainer Kit.

2. IC-7486.

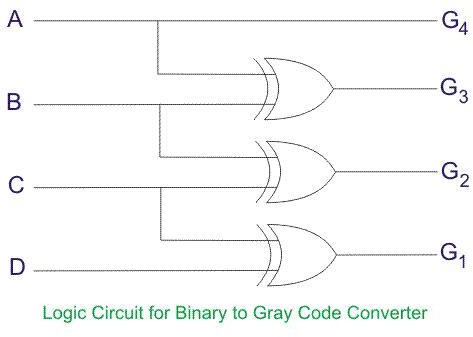
3.Connecting wires etc

**Theory: Binary-to-Gray Conversion**

For Binary-to-Gray Conversion, follow the steps below

**Step 1:** Record the MSB as it is.

**Step 2:** Add(Apply X-OR) the previous bit to the next bit, recording the sum and neglecting the carry

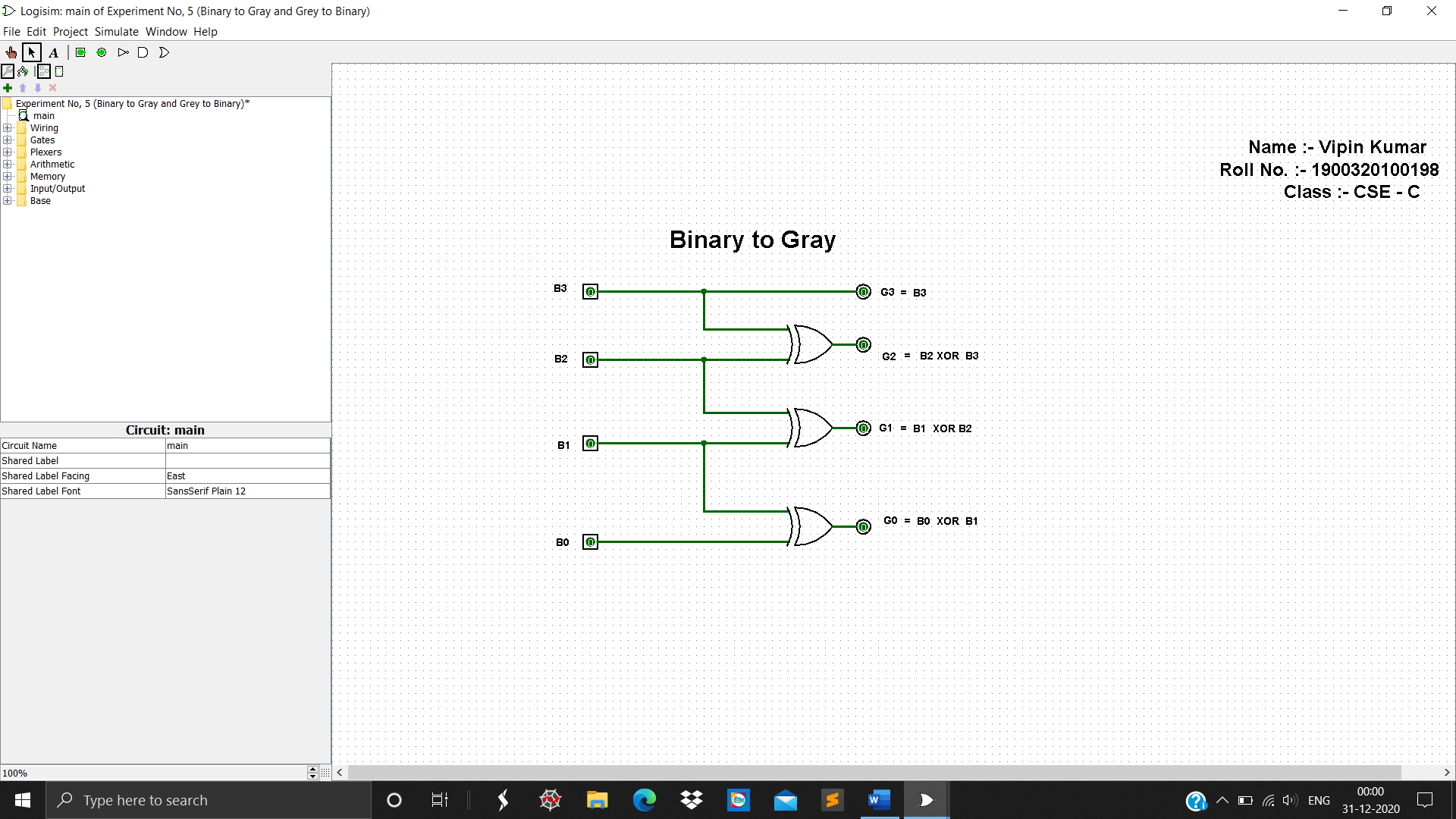
** Step 3:** Record Successive sums until completed

Circuit Diagram Of Gray-to-Binary Conversion:

**OBSERVATION TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Input (4 bit Binary Code)** | **Output (4 bit Gray Code)** |
| 0 | **0000** | **0000** |
| 1 | **0001** | **0001** |
| 2 | **0010** | **0011** |
| 3 | **0011** | **0010** |
| 4 | **0100** | **0110** |
| 5 | **0101** | **0111** |
| 6 | **0110** | **0101** |
| 7 | **0111** | **0100** |

**LOGISIM RESULT:**

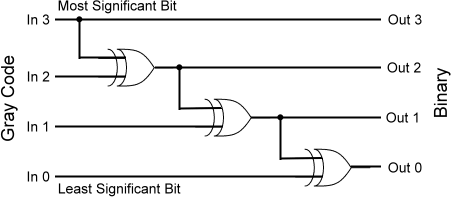
****

**Gray-to-Binary Conversion**

For Gray-to Binary Conversion, follow the steps below

**Step 1:** The MSB of Gray and Binary are same. So write it directly.

**Step 2:** Add(Apply X-OR) binary MSB to the next bit of Gray Code. Record the result and ignore the carries

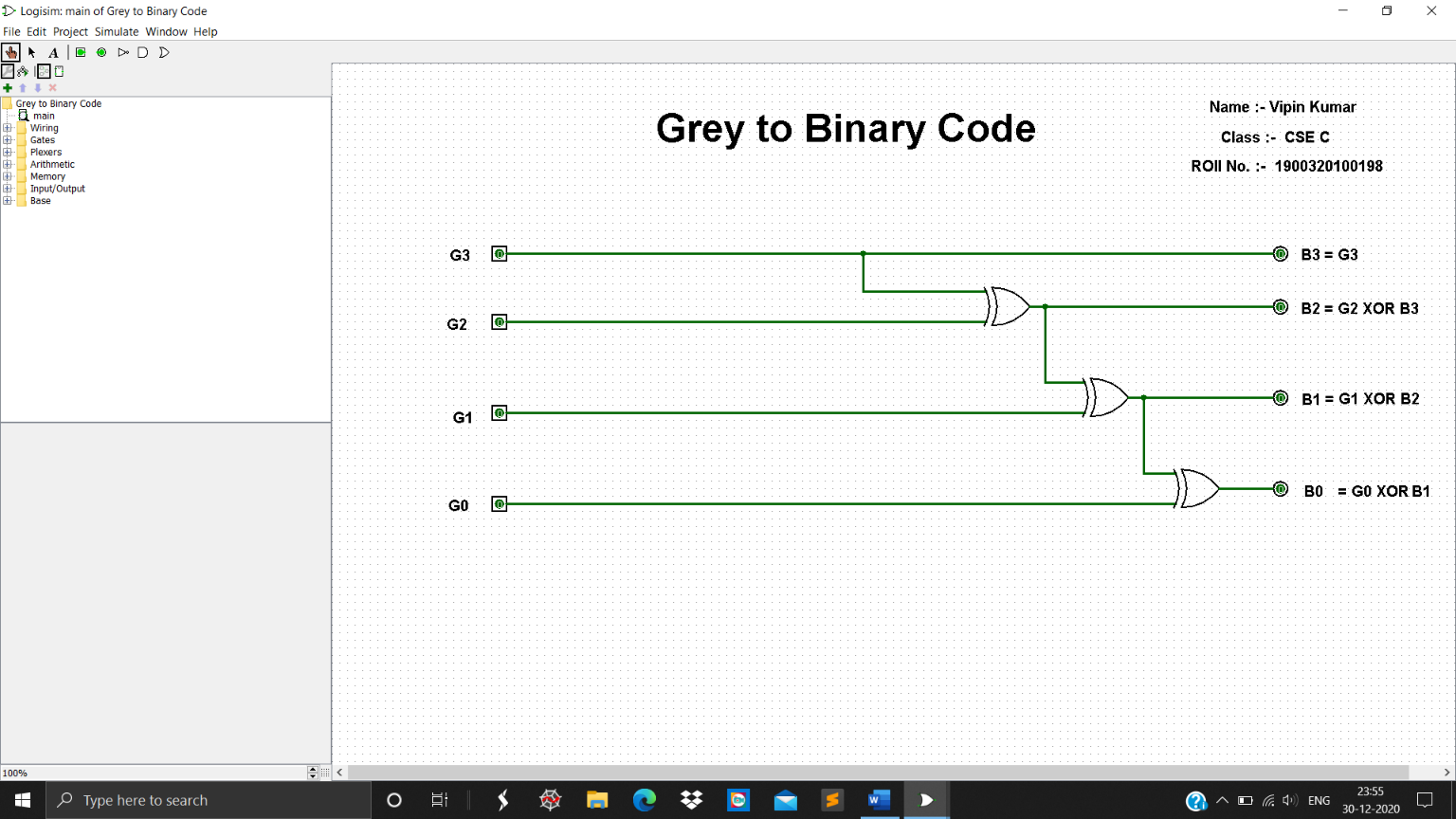
**Step 3:** Continue this process until the LSB is reached

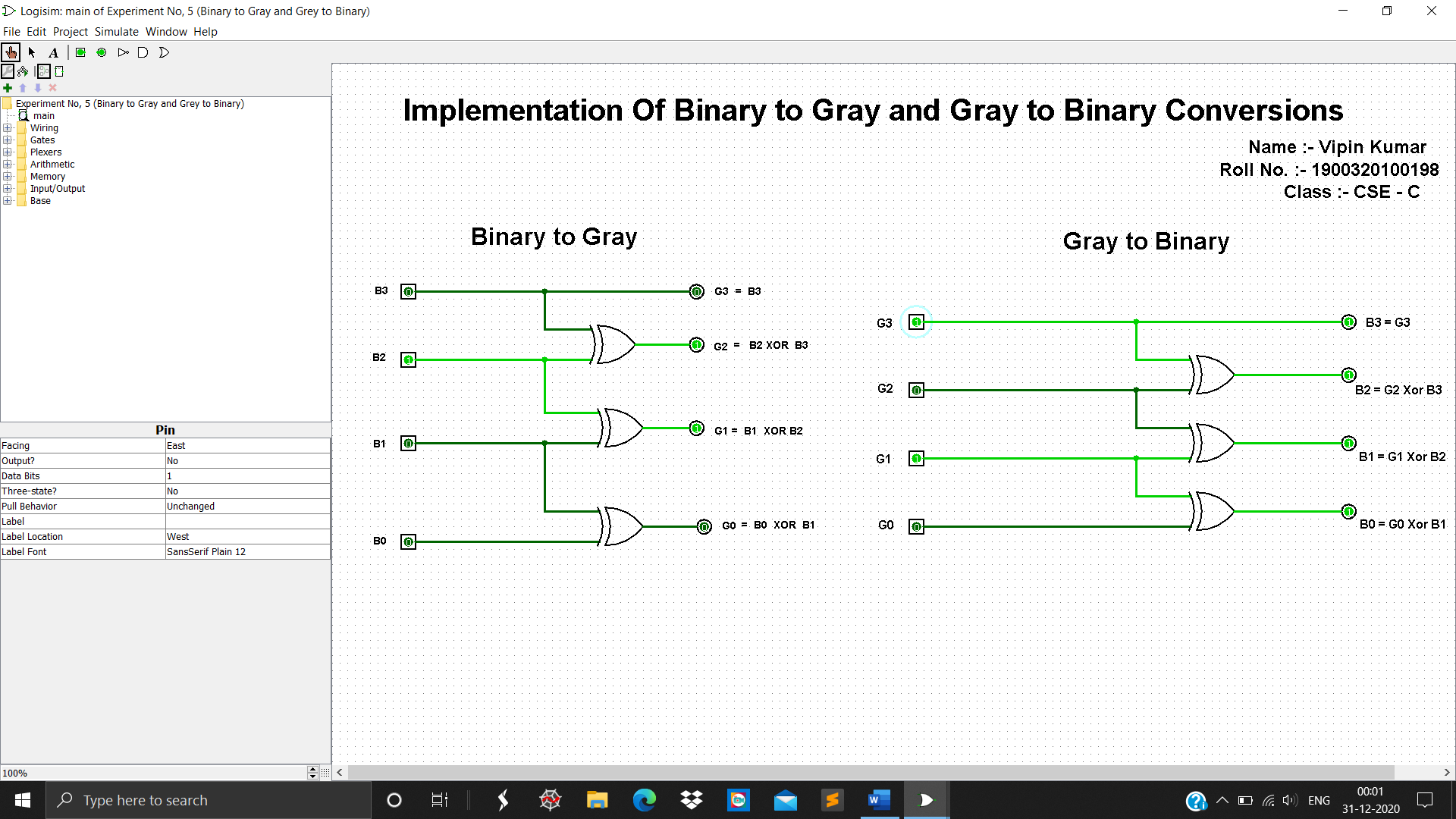
#### E:\CAO Material\CAO and DLD lab\Gray code conversion logic.pngCircuit Diagram of Gray-to-Binary Conversion:

**OBSERVATION TABLE:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Input (4 bit Gray Code)** | **Output (4 bit Binary Code)** |
| 0 | **0000** | **0000** |
| 1 | **0001** | **0001** |
| 2 | **0011** | **0010** |
| 3 | **0010** | **0011** |
| 4 | **0110** | **0100** |
| 5 | **0111** | **0101** |
| 6 | **0101** | **0110** |
| 7 | **0100** | **0111** |

**LOGISIM RESULT:**





**RESULT:** All outputs are verified

**EXPERIMENT NO….6**

**EXPERIMENT :** Design the data path of a computer from its register transfer language.

**PROBLEM STATEMENT:** While performing an instruction in a computer the data is transferred from one register to another register according to the architecture used in the computer. The register transfer the data according to their register transfer language(RTL). In this experiment we will design a data path which represent a register transfer Statement

x + yz : R2←R1

**APPRATUS REQUIRED:** 1. Registers

2. AND Gates

3. OR Gate

4. Connecting Wires

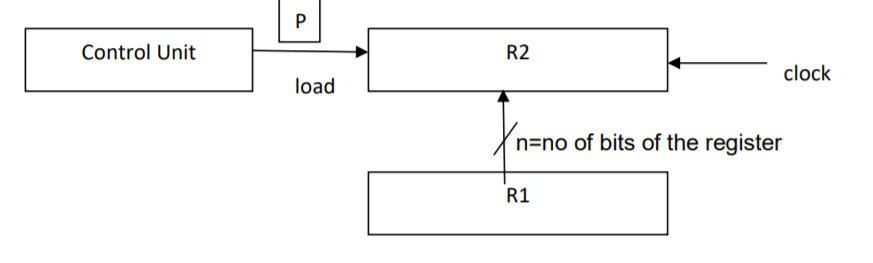
**THEORY:** The symbolic notation used to describe the micro-operation transfers among registers is called Register Transfer Language. The term “register transfer” implies the availability of hardware logic circuits that can perform a stated micro-operation and transfer the result of the operation to the same or another register. A statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination register has a parallel load capacity. If the transfer is to occur under a predetermined condition i.e.

If(P=1) then R2←R1

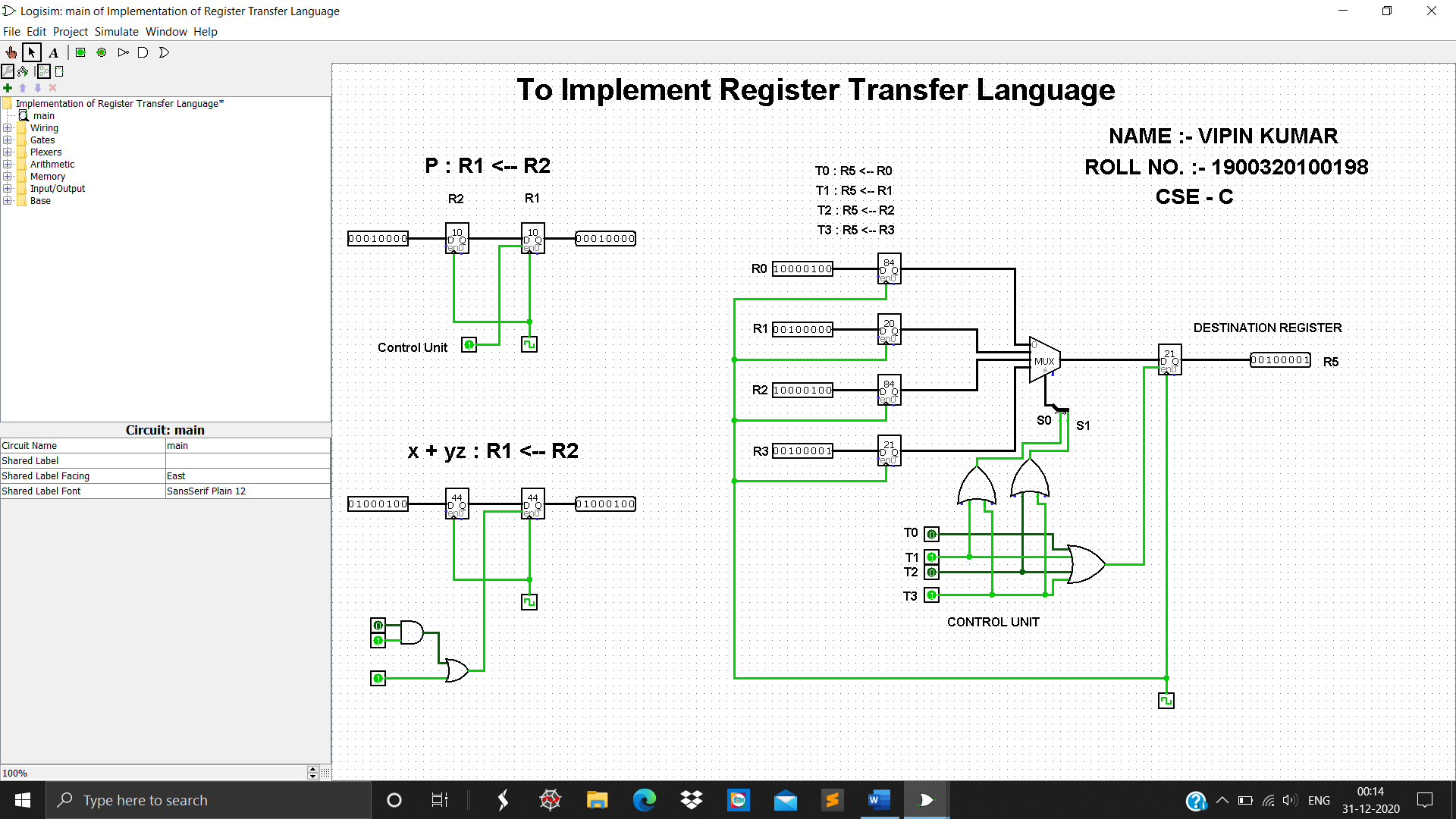
Where P is the control signal generated in the control section. A Control Function is a Boolean variable that is equal to 0 or 1

P: R2←R1

**BLOCK DIAGRAM:-**



**LOGISIM IMPLEMENTATION:-**

****

**RESULT:-** The circuit has been designed according to the given Register Transfer statement and verified.

**EXPERIMENT NO. ….7…**

**PROBLEM STATEMENT:** Design a bus system using Multiplexer for data transfer from register.

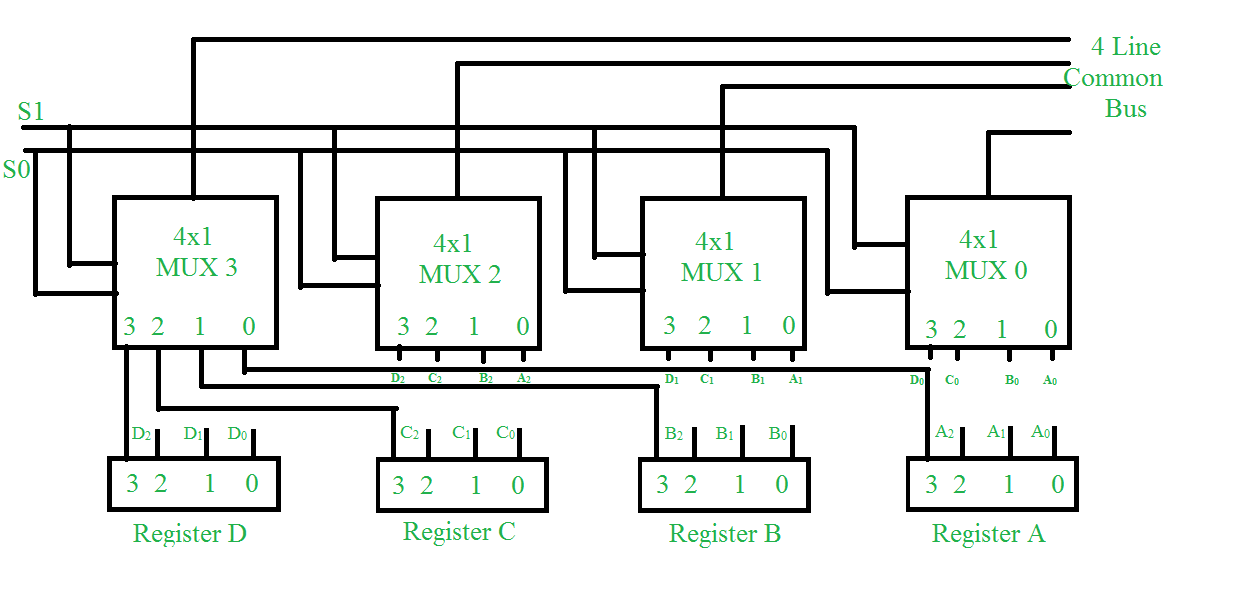
**APPRATUS REQUIRED:** 1.Decoder

**2.** Multiplexer

**3.** Registers

**4.** Connecting Wires

**THEORY:** A typical computer has many registers and we need to transfer the information between these registers. A way to transfer the information is using the common bus system The construction of this bus system for 4 registers is shown . The bus consists of 4×1 multiplexers with 4 inputs and 1 output and 4 registers with bits numbered 0 to 3. There are 2 select inputs S0 and S1 which are connected to the select inputs of the multiplexers.



The output 1 of register A is connected to input 0 of MUX 1 and similarly other connections are made as shown in the diagram. The data transferred to the bus depends upon the select lines. A table for the various combinations of select lines is shown below.

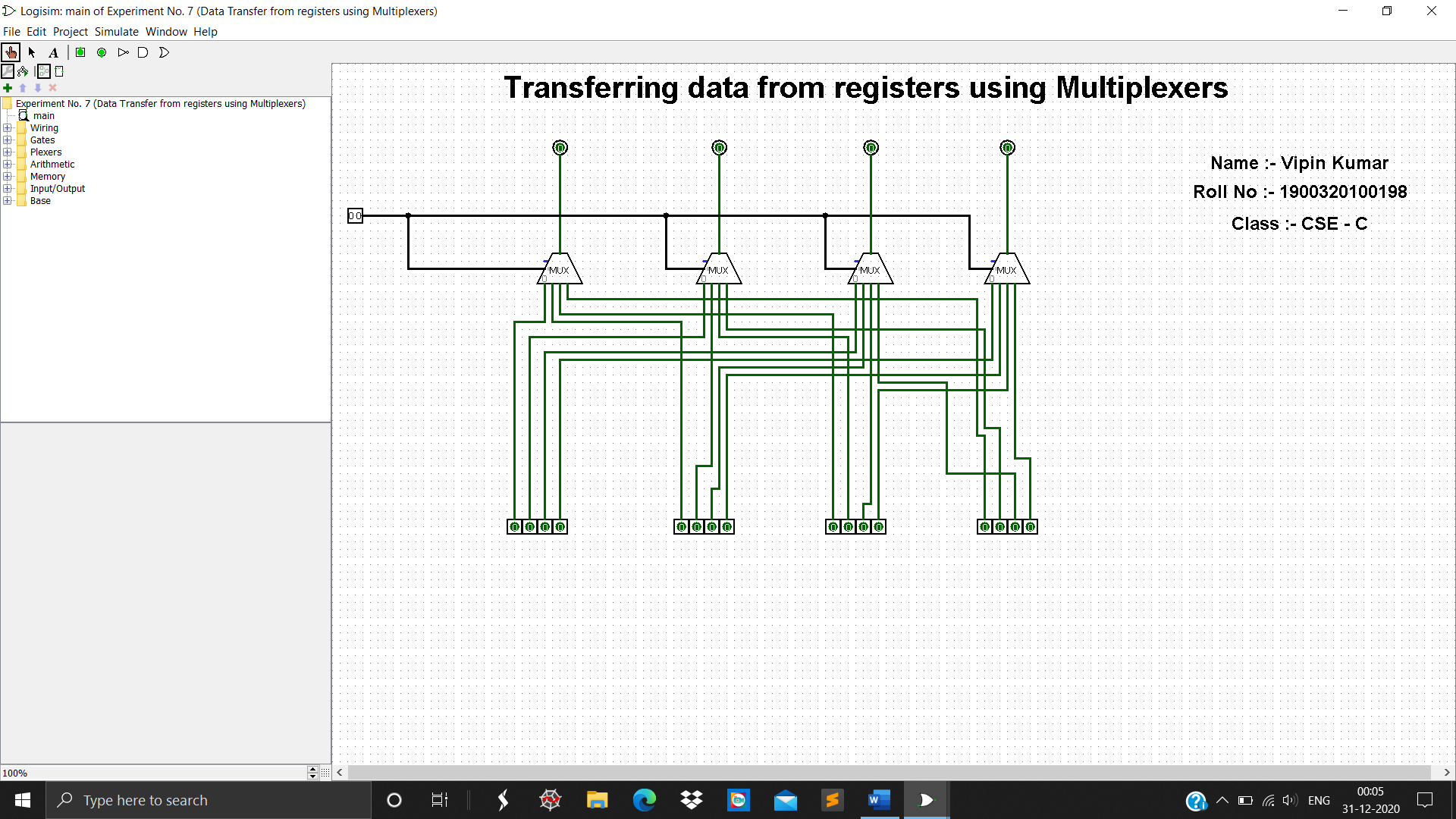
**OBSERVATION TABLE :**

|  |  |  |
| --- | --- | --- |
| **S1** | **S0** | **Register Selected** |
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |

As we can see that when S1S0=00, register A is selected because on 00 the 0 data inputs of all the multiplexers are applied to the common bus.

Since the 0 data inputs of all the multiplexers receive the inputs from the register A, thus register A gets selected. Similarly for other combinations of S1S0 other register are selected.

**IMPLEMENTATION(LOGISIM):**

****

**RESULT:**

The observation table has been verified.

**EXPERIMENT NO. ....8....**

**PROBLEM STATEMENT :**  Design a 8-bit ARITHMETIC LOGIC UNIT

.**APPRATUS REQUIRED:**

1. Binary Adders

2. Multiplexer

3. AND Gates

4. OR Gate

5. X-OR Gate

6. NOT Gate

7. Connecting Wires

**THEORY:**

**Arithmetic Circuit:-**

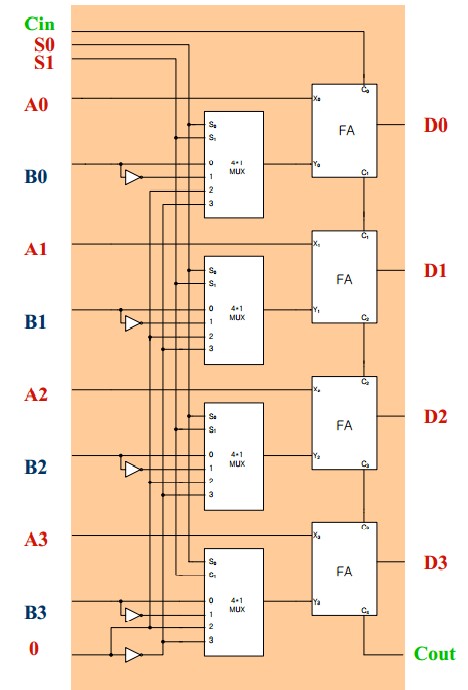
The basic component of an arithmetic circuit is the parallel adder. By controlling data inputs to the adder, it is possible to obtain different types of arithmetic operations. The circuit shows a 4-bit arithmetic circuit. It has four full-adder circuits that constitute the 4-bit adder and four Multiplexers for choosing different operations. There are two 4-bit inputs A and B and a 4-bit output D. The four inputs from A go directly to the X inputs of the binary adder. Each of the four inputs from B are connected to the data inputs of the

multiplexers. The multiplexers data inputs also receive the complement of B . The other two data inputs are connected to logic-0 and logic-1 . Logic-0 i s a fixed voltage value (0 volts for TTL integrated circuits) and the logic-1 signal can be generated through· an inverter whose input is 0. The fourmultiplexers are controlled by two selection inputs, S1 and S0. The input carry Cin goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.

The output of the binary adder is calculated from the following arithmetic sum:

D = A + Y + C in

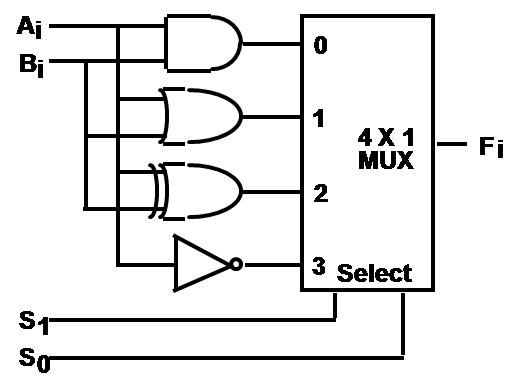
where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder. C in is the input carry, which can be equal to 0 or 1. Note that the symbol + in the equation above denotes an arithmetic plus. By controlling the value of Y with the two selection inputs S1 and S0 and making C in equal to 0 or 1, it is possible to generate the eight arithmetic micro-operations



**Logical Unit :**

Figure shows one stage of a circuit that generates the four basic logic rnicro-operations . It consists of four gates and a multiplexer. Each of the four logic operations is generated through a gate that performs the required logic. The outputs of the gates are applied to the data inputs of the multiplexer. The two selection inputs S1 and S0 choose one of the data inputs of the multiplexer and direct its value to the output.

The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated n times for i = 0, 1, 2, . . . , n - 1. The selection variables are applied to all stages.

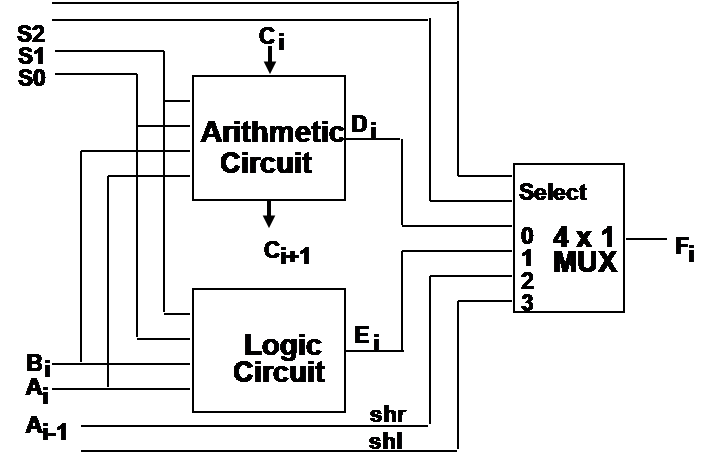


###### **A single stage of ALU circuit :**

One stage of an arithmetic logic shift unit is shown in Figure. The subscript i designates a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs S1 and S0. A 4 x 1 multiplexer at the output chooses between an arithmetic output in Ei; and a logic output in Hi. The data in the multiplexer are selected with inputs S3 and S2. The other two data inputs to the multiplexer receive inputs Ai-1 for the shift-right operation and Ai+1 for the shift-left operation.

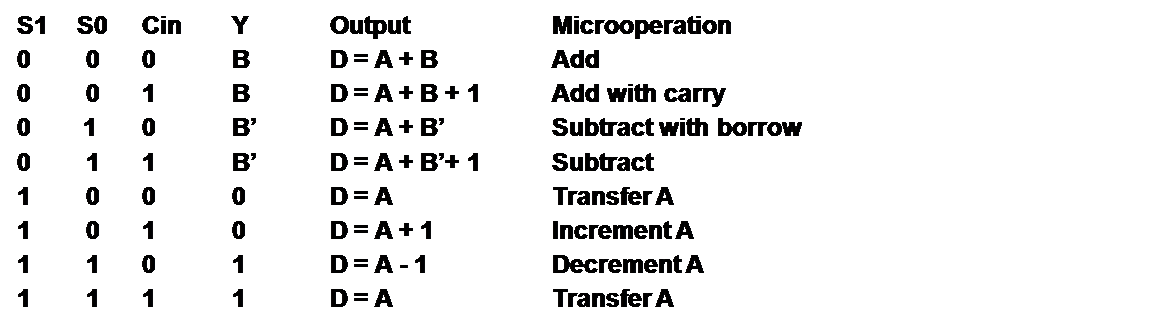
The diagram shows just one typical stage. The circuit of Figure must be repeated n times for an n-bit ALU. The output carry Ci+1 of a given arithmetic stage must be connected to the input carry Ci of the next stage in sequence. The input carry to the first stage is the input carry Cin which provides a selection variable for the arithmetic operations.

The circuit provides eight arithmetic operation, four logic operations, and two shift operations. Each operation is selected with the five variables S3, S2, S1, S0, and Cin. The input carry Cin is used for selecting an arithmetic operation only. Function Table lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with S3S2 = 00. The next four are logic operations and are selected with S3S2 = 01. The input carry has no effect during the logic operations and is marked with don't-care X ' s . The last two operations are shift operations and are selected with S3S2 = 10 and 11 . The other three selection inputs have to the shift.

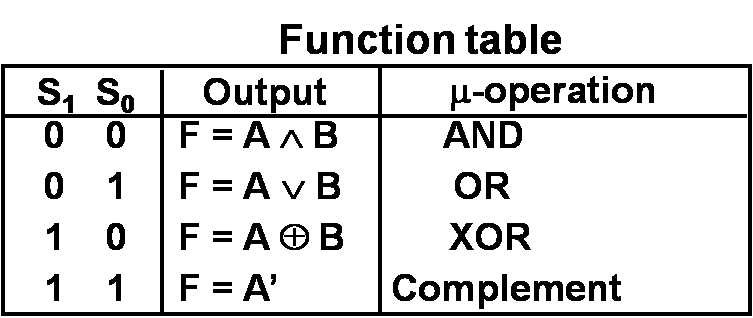


**TRUTH TABLE:**

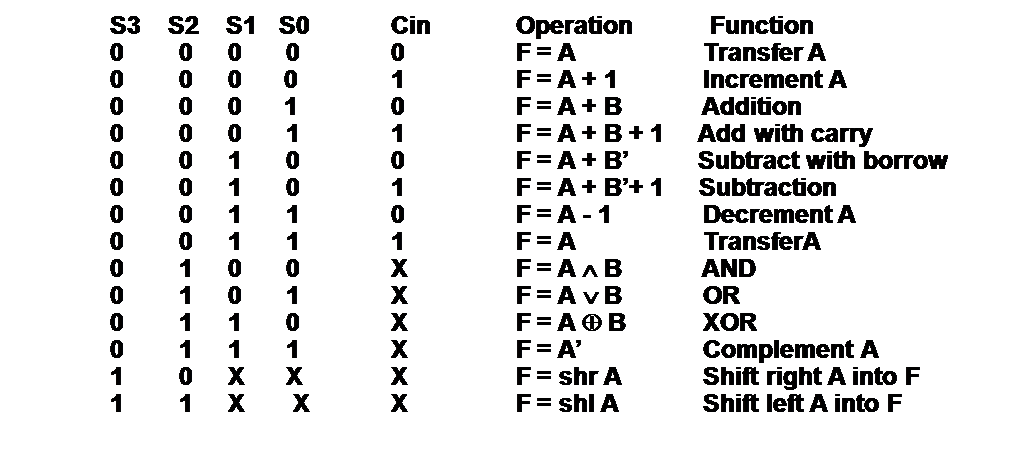
###### **Function Table of Arithmetic Circuit :**

****

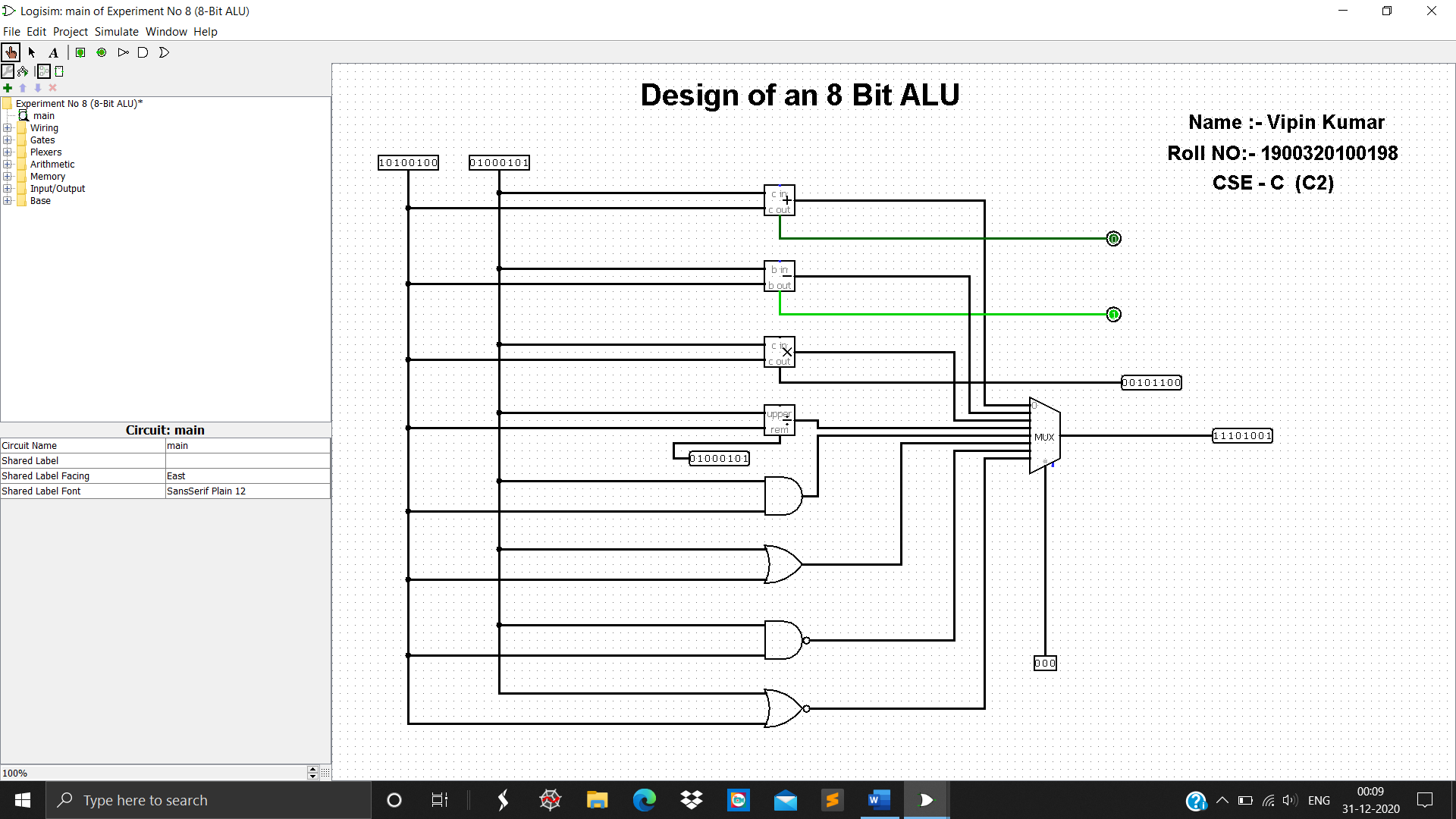
**Function table of logical circuit**:

****

**Function table of ALU circuit:**

****

**Implementation(LOGISIM):**



**RESULT:**

The ALU circuit has been designed and verified according to the function table of Arithmetic Logic Unit

**EXPERIMENT …9…**

**PROBLEM STATEMENT:** Implement an array multiplier of 2\*2 and 4\*3 bit.

**Apparatus Required:**

1.Bread Board Trainer Kit

2.Half Adders

3.And Gates

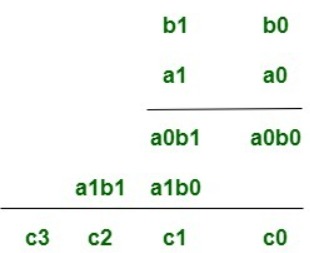
4.Connecting Wires

**THEORY:** An array multiplier is a digital combinational circuit used for multiplying two binary numbers by employing an array of full adders and half adders. This array is used for the nearly simultaneous addition of the various product terms involved. To form the various product terms, an array of AND gates is used before the Adder array.

Checking the bits of the multiplier one at a time and forming partial products is a sequential operation that requires a sequence of add and shift micro-operations. The multiplication of two binary numbers can be done with one micro-operation by means of a combinational circuit that forms the product bits all at once. This is a fast way of multiplying two numbers since all it takes is the time for the signals to propagate through the gates that form the multiplication array. However, an array multiplier requires a large number of gates, and for this reason it was not economical until the development of integrated circuits.

For implementation of array multiplier with a combinational circuit, consider the multiplication of two 2-bit numbers as shown in figure. The multiplicand bits are b1 and b0, the multiplier bits are a1 and a0, and the product is

**c3c2c1c0**



Assuming A = a1a0 and B= b1b0, the various bits of the final product term P can be written as:-

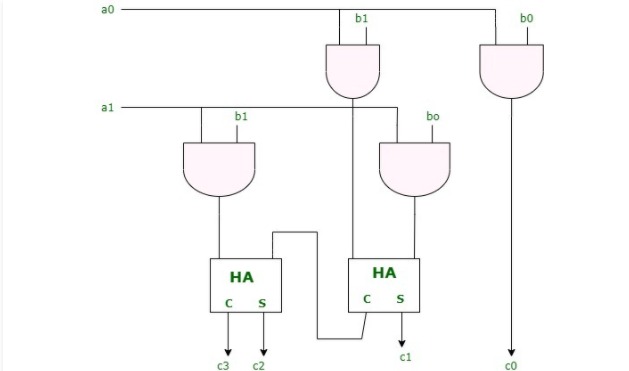
1. P(0)= a0b0

2. P(1)=a1b0 + b1a0

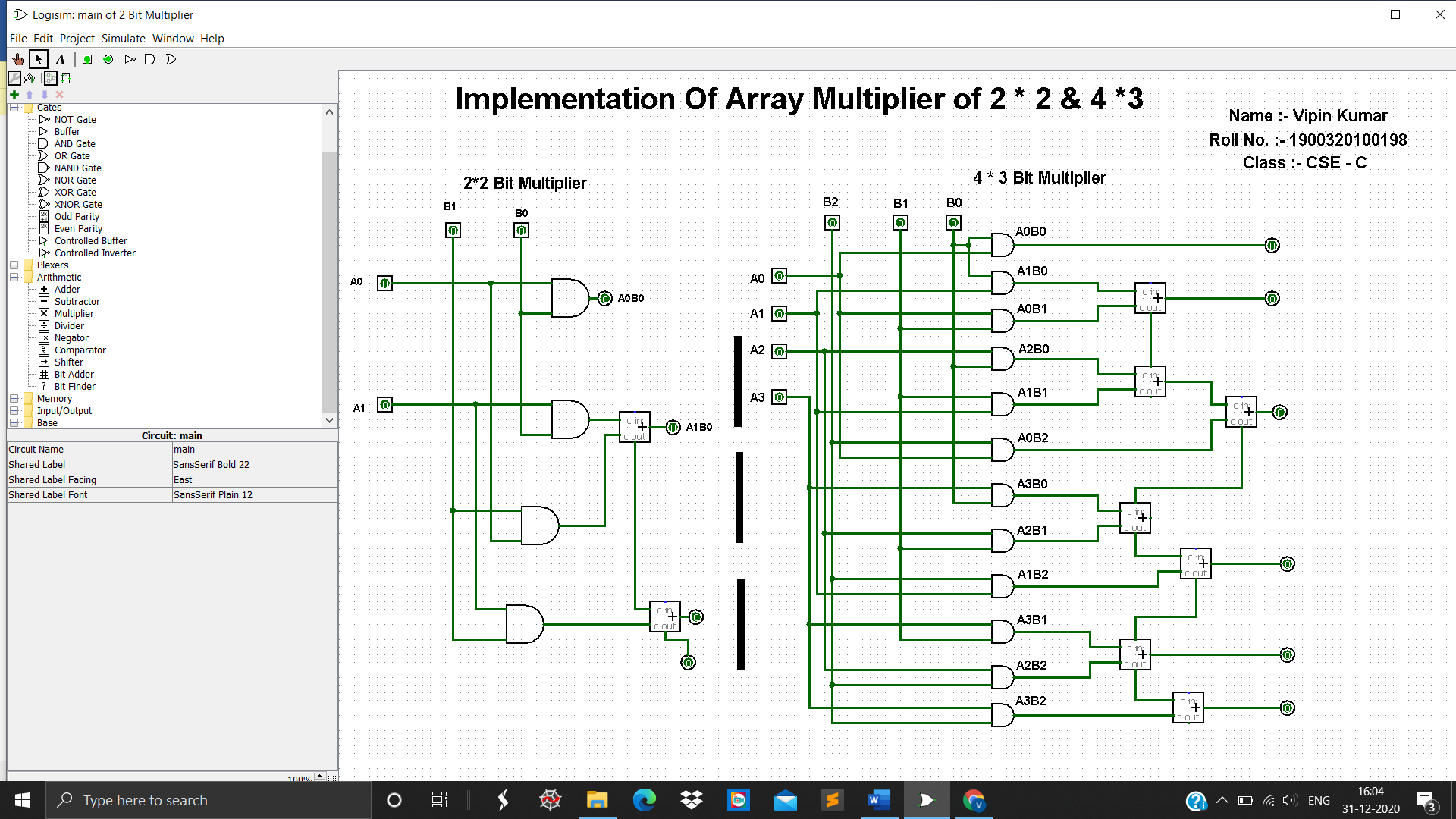
3. P(2) = a1b1 + c1 where c1 is the carry generated during the addition for the P(1) term.

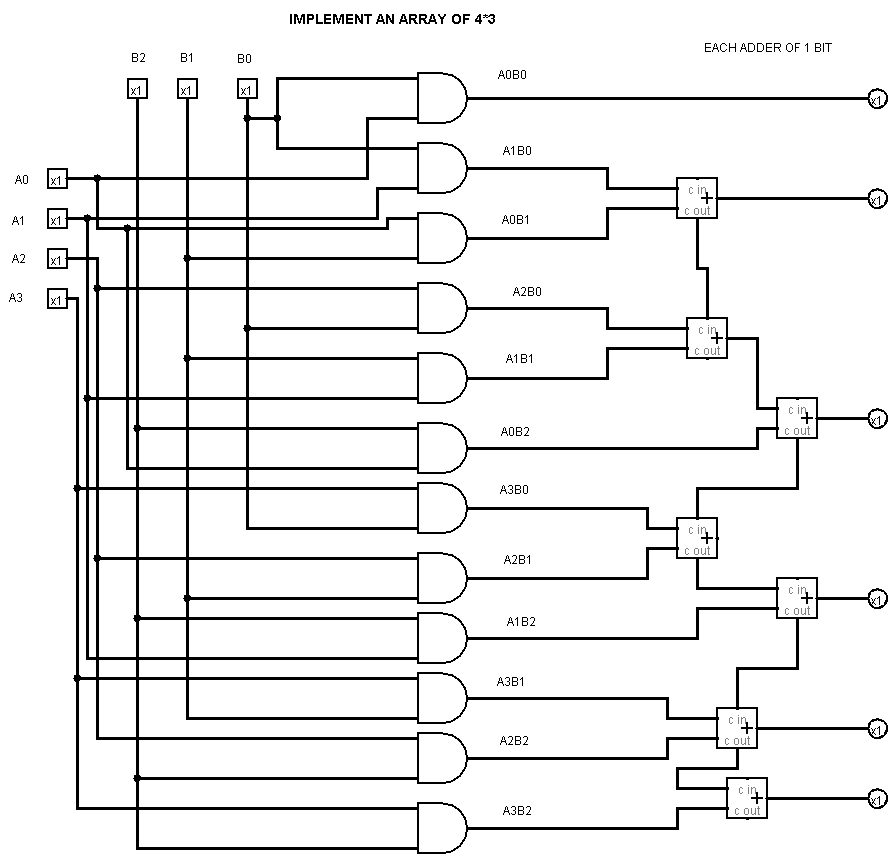
4. P(3) = c2 where c2 is the carry generated during the addition for the P(2) term.

**BLOCK DIAGRAM:**



**IMPLEMENTATION (LOGISIM):**

****

****

**RESULT:** All outputs are verified.

**EXPERIMENT …10…**

**PROBLEM STATEMENT:** Implement 4 bit binary incrementer and decrementer.

**Apparatus Required:**

1.Bread Board Trainer Kit

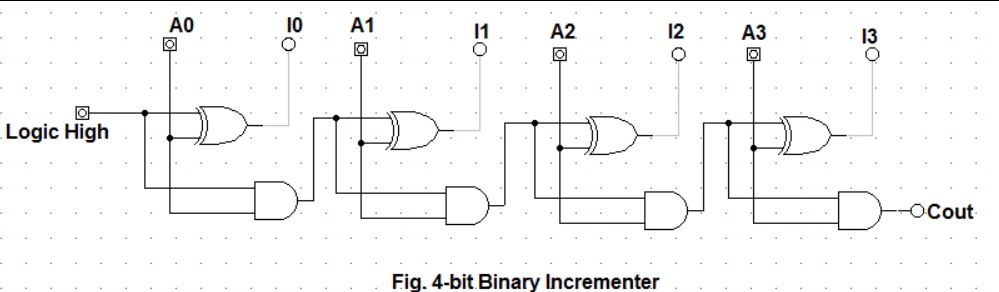
2.Half Adders

3.And Gates

4.Connecting Wires

**THEORY:** The binary incrementer increases the value stored in a register by ‘1’. For this, it simply adds ‘1’ to the existing value stored in a register. It is made by cascading ‘n’ half adders for ‘n’ number of bits i.e. the storage capacity of the register to be incremented. Hence, a 4-bit binary incrementer requires 4 cascaded half adder circuits.

**Block Diagram of Incrementer :**



**Observed Values:**

Following set of values were obtained in observation.

1. 0011 => 0100

2. 1010 => 1011

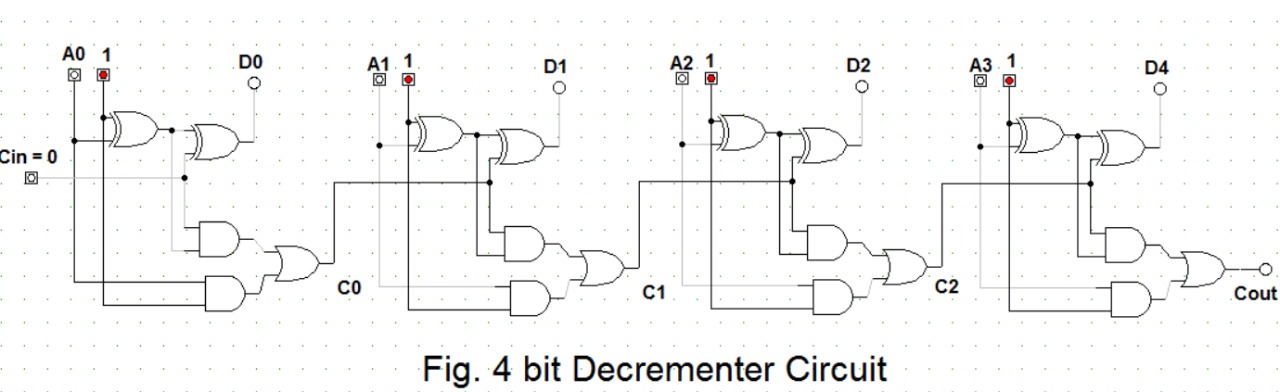
3. 1101 => 1110

4. 0010 => 0011

5. 1111 => 0000; Cout = 1

The binary decrementer decreases the value stored in a register by ‘1’. For this, we can simply add ‘1’ to the each bit of the existing value stored in a register. This is basically the concept of two's complement used for subtraction of '1' from given data. It is made by cascading ‘n’ full adders for ‘n’ number of bits i.e. the storage capacity of the register to be decremented. Hence, a 4-bit binary decrementer requires 4 cascaded full adder circuits. As stated above we add '1111' to 4 bit data in order to subtract '1' from it.

**Block Diagram of decrementer :**



**Observed Values:**

Following set of values were obtained in observation.

1. 0011 => 0010

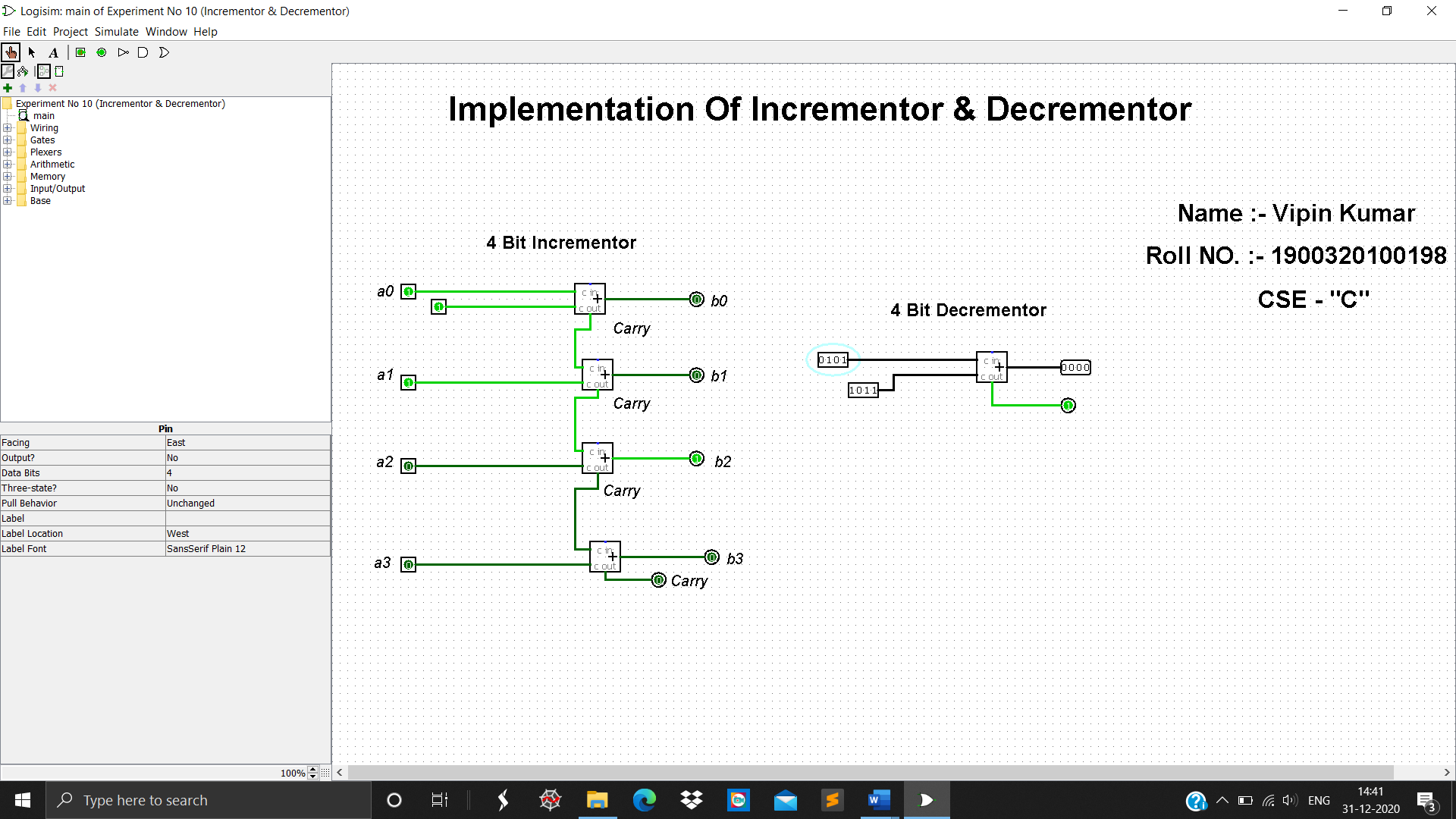
2. 1010 => 1001

3. 1101 => 1100

4. 0010 => 0001

5. 0000 =>1111

**IMPLEMENTATION (LOGISIM):**

****

**RESULT:** All outputs are verified.