Hardware Lab (CS 224) Assignment 2

Group No. 18

Team Members:
Daksh Agarwal (230101123)
Aditya Shukla (230101115)
Kartik Maheshwari (230101116)
Ryan Aggarwal (230101117)

January 26, 2025

Contents

1 Problem Statement

The objective of this assignment is to design and implement a Gray code to Binary code conversion circuit. The specifications are as follows:

- Input: A 4-bit Gray code.
- Output: A 4-bit Binary code.
- Use only 2-input logic gates (AND, OR, NOR, NAND gates) and the corresponding ICs.
- For a variable (a), either produce its complement (a') using gates or assume both are available at the input as convenient.

The implementation is to be demonstrated on a breadboard, ensuring adherence to the provided guidelines and optimization techniques.

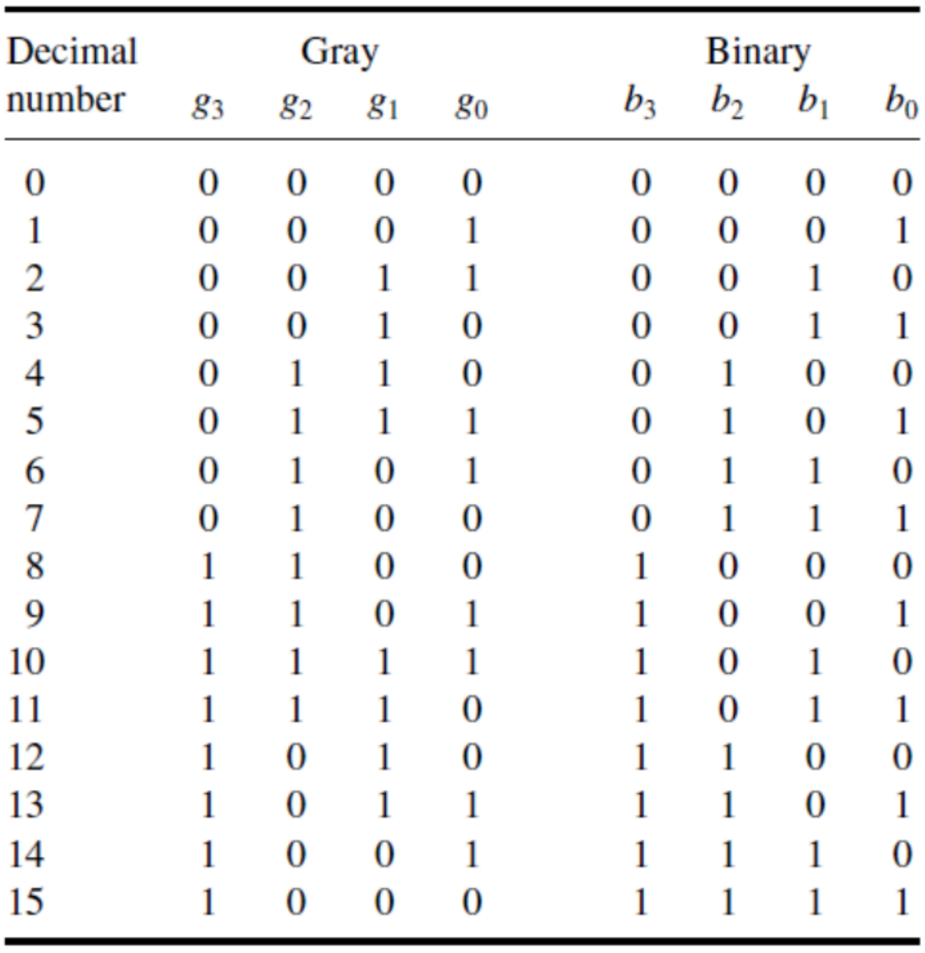
2 Process to Obtain the Design

2.1 Deriving Boolean Expressions Using K-Maps

To design the Gray code to Binary code conversion circuit, we began by deriving the Boolean expressions for each Binary output bit (b_3, b_2, b_1, b_0) in terms of the Gray code inputs (g_3, g_2, g_1, g_0). We used Karnaugh maps (K-maps) to simplify the logic and obtain the minimized Boolean expressions.

2.1.1 Identify the Truth Table

We first constructed the truth table for the Gray code to Binary code conversion. The truth table maps each 4-bit Gray code input to its corresponding 4-bit Binary code output. The truth table is shown below:

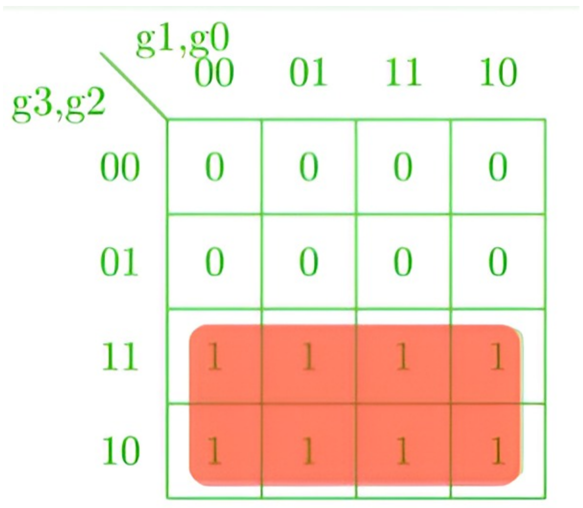


Decimal	Gray				Binary			
number	g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

Figure 1: Gray Code to Binary Code Truth Table

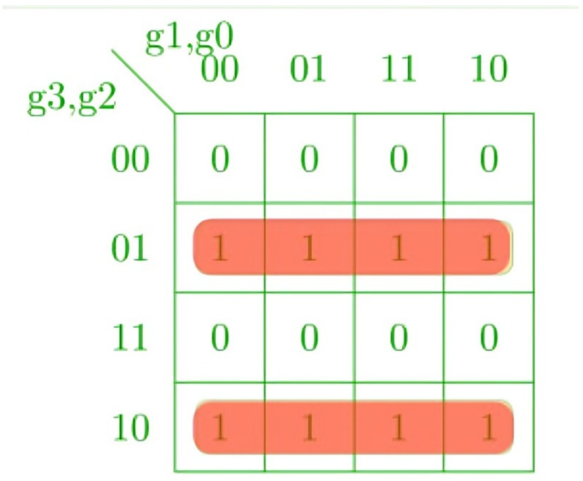
2.1.2 Plot the Minterms on K-Maps

For each Binary output bit (b_3, b_2, b_1, b_0), we plotted the minterms on a 4-variable K-map. The K-maps for each output bit are shown below:



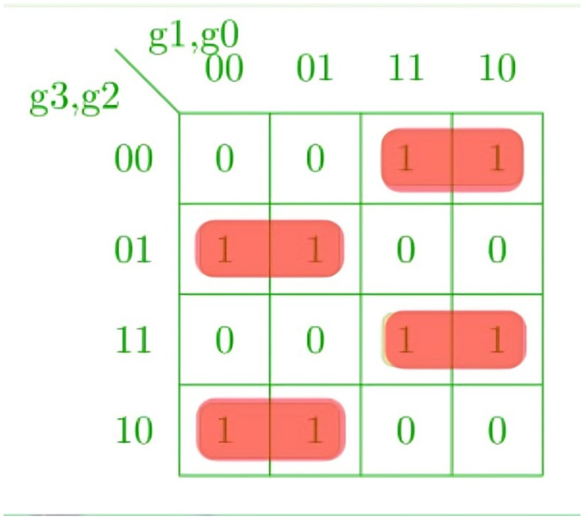
g3,g2 \ g1,g0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

Figure 2: K-map for b_3



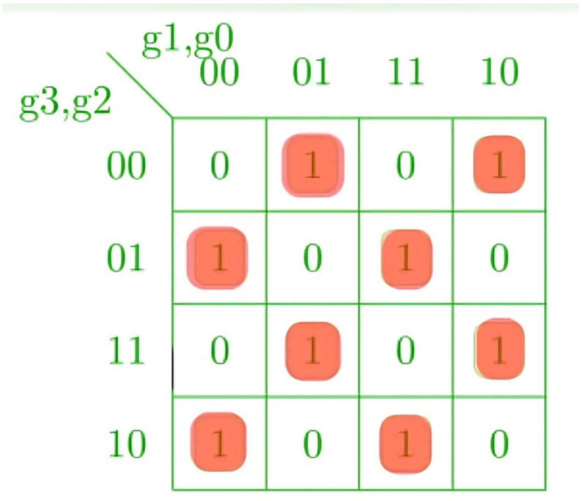
g3,g2 \ g1,g0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Figure 3: K-map for b_2



g3,g2 \ g1,g0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

Figure 4: K-map for b_1



g3,g2 \ g1,g0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Figure 5: K-map for b_0

2.1.3 Derive the Boolean Expressions

From the grouped minterms in the K-maps, we derived the Boolean expressions for each Binary output bit. The expressions are as follows:

$$\begin{aligned} b_3 &= g_3 \\ b_2 &= g_3 \oplus g_2 \\ b_1 &= g_3 \oplus g_2 \oplus g_1 \\ b_0 &= g_3 \oplus g_2 \oplus g_1 \oplus g_0 \end{aligned}$$

These expressions were obtained by analyzing the patterns in the K-maps and grouping the minterms to minimize the number of terms in the Boolean expressions.

2.2 Implementing XOR Using 4 NAND Gates

This section demonstrates an XOR implementation using precisely four NAND gates.

2.2.1 Logical Derivation

Let A and B be the inputs. The XOR operation can be realized through the following sequence of NAND operations:

1. Compute intermediate signal $N_1 = \overline{A \cdot B}$ (NAND of original inputs)
2. Compute $N_2 = \overline{A \cdot N_1}$ (NAND of A with N_1)
3. Compute $N_3 = \overline{B \cdot N_1}$ (NAND of B with N_1)
4. Final output $X = \overline{N_2 \cdot N_3}$ (NAND of intermediate signals)

The complete logical expression becomes:

$$XOR(A, B) = \overline{\left(\overline{A \cdot \overline{A \cdot B}}\right) \cdot \left(\overline{B \cdot \overline{A \cdot B}}\right)}$$

2.2.2 Truth Table Verification

Table 1: XOR Implementation Truth Table

A	B	N_1	N_2	N_3	X	XOR
0	0	1	1	1	0	0
0	1	1	1	0	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0

2.2.3 Circuit Diagram

The circuit operates as follows:

- First NAND gate (top) processes inputs A and B
- Second and third NAND gates combine the original inputs with the first gate's output

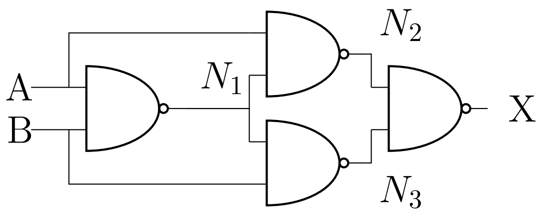


Figure 6: XOR Implementation Using Four NAND Gates

- Final NAND gate produces the XOR result from the intermediate signals

This implementation maintains optimal component count.

2.3 ICs Used and Gates Required

For the implementation of the Boolean function, we used only NAND gates to adhere to the assignment constraints. The following ICs and gates were required:

- We used the `7400` IC, which contains four 2-input NAND gates.
- A total of 12 NAND gates were used to implement the entire circuit.

3 Circuit Diagram

In this section, we present the step-by-step design of the Gray code to Binary code conversion circuit. The design process involves three stages:

1. **Circuit using XOR gates**: The initial implementation uses XOR gates to perform the conversion.
2. **Creating an XOR gate using NAND gates**: Since only 2-input NAND gates are allowed, we designed an XOR gate using NAND gates.
3. **Final circuit using only NAND gates**: The complete circuit is implemented using only NAND gates, adhering to the assignment constraints.

3.1 Circuit Using XOR Gates

The initial circuit uses XOR gates to convert Gray code to Binary code. The most significant bit (MSB) of the Binary code is directly taken from the Gray code input, while the remaining bits are computed using XOR operations. The circuit diagram for this implementation is shown below:

In this diagram:

- XOR gates are used to compute the Binary bits b_2, b_1, and b_0.
- The MSB b_3 is directly taken from the Gray code input g_3.

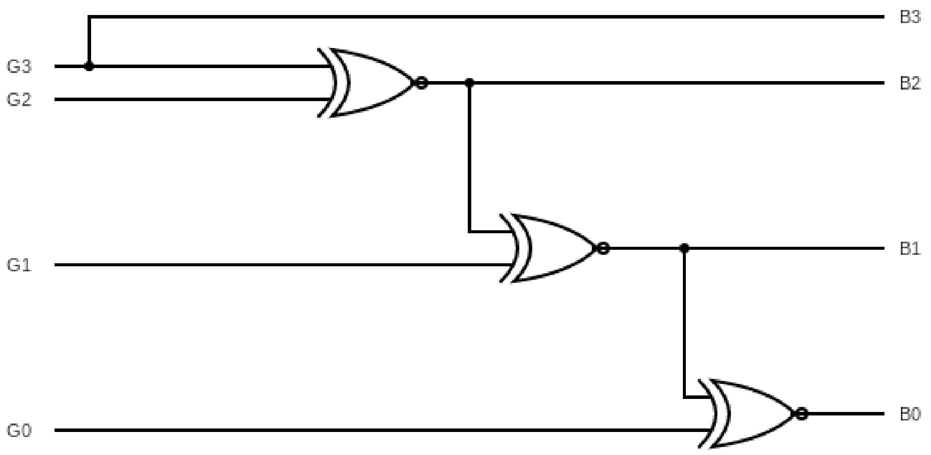


Figure 7: Circuit using XOR gate.

3.2 Creating an XOR Gate Using NAND Gates

Since the assignment restricts us to using only 2-input NAND gates, we designed an XOR gate using NAND gates. This design ensures that we can perform XOR operations without using dedicated XOR gates. The circuit diagram for creating an XOR gate using NAND gates is shown below:

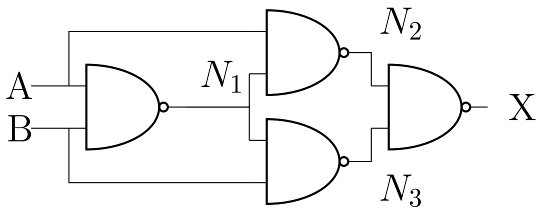


Figure 8: XOR Implementation Using Four NAND Gates

In this diagram:

- Four NAND gates are used to create a single XOR gate.
- This modular design can be replicated for each XOR operation in the Gray code to Binary code conversion.

3.3 Final Circuit Using Only NAND Gates

The final circuit replaces all XOR gates with the NAND-based XOR design. This ensures that the circuit adheres to the assignment constraints of using only 2-input NAND gates. The complete circuit diagram for the Gray code to Binary code conversion using only NAND gates is shown below:

In this diagram:

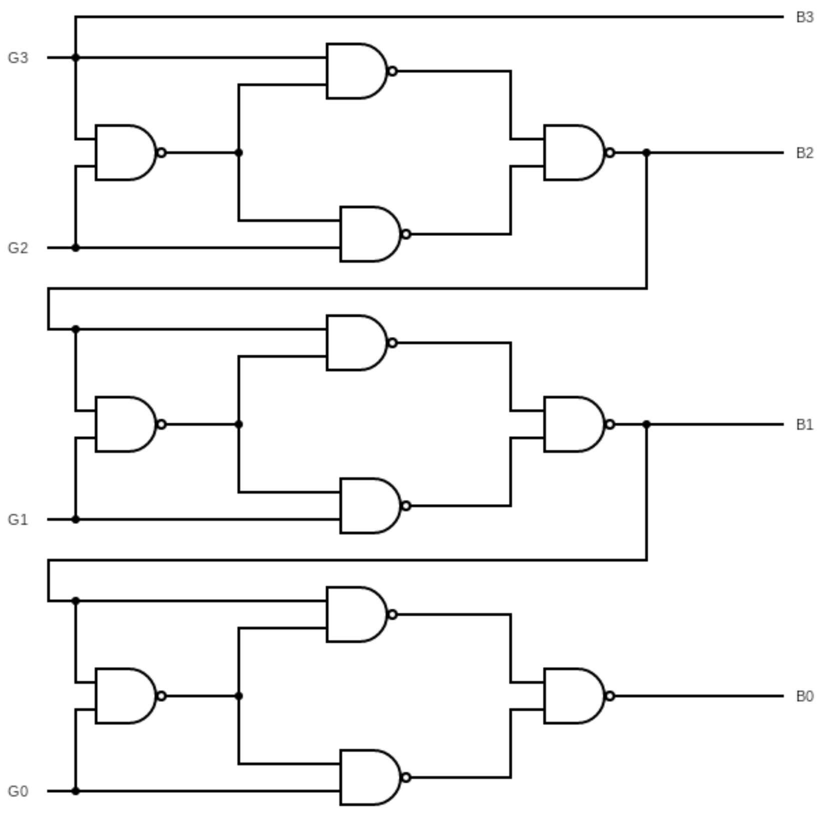


Figure 9: Circuit using NAND gates.

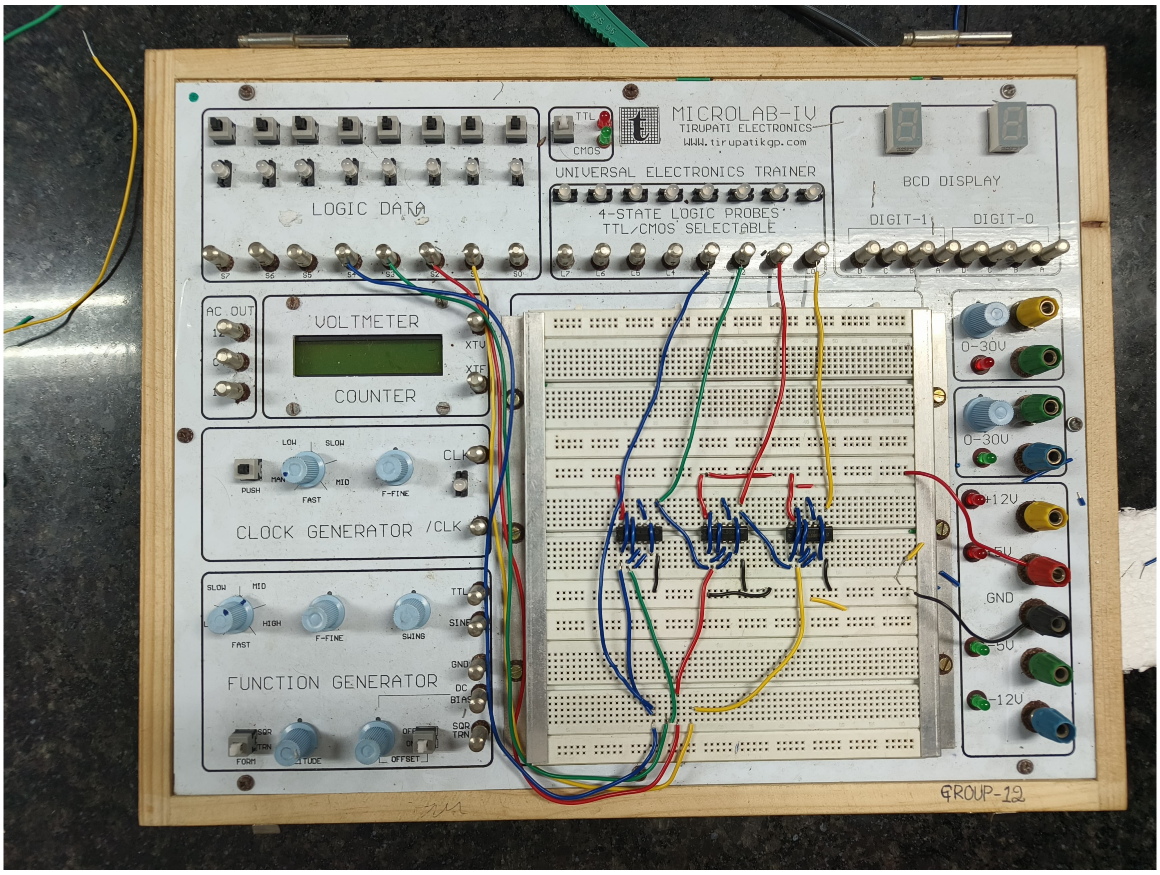
- Each XOR operation in the original circuit is replaced with the NAND-based XOR design.
- The circuit is optimized to minimize the number of NAND gates and ensure clean wiring.

3.4 Explanation of the Design Process

1. **Initial Design with XOR Gates**: The XOR-based design is straightforward but does not meet the assignment constraints of using only 2-input NAND gates.
2. **XOR Gate Using NAND Gates**: To comply with the constraints, we designed an XOR gate using NAND gates. This step ensures that the circuit can be implemented using only the allowed gates.
3. **Final NAND-Based Circuit**: The final circuit replaces all XOR gates with the NAND-based XOR design. This ensures that the circuit is fully compliant with the assignment requirements while maintaining functionality.

4 Picture of the Circuit

The following picture shows the actual implementation of the circuit on the breadboard. The ICs, hookup wires, and the arrangement of the components can be observed.



5 Things we kept in mind

While implementing the circuit on the breadboard, we followed a systematic approach to ensure clarity and organization. The following guidelines were kept in mind:

- **Ground Connections:** All ground connections were made using black wires to maintain consistency.
- **Power Connections:** The power connections ($+5V$) were made using red wires.
- **Input Wires:** Different colored wires were used for each input variable to distinguish them easily.
- **Intermediate Connections:** Blue wires were used for intermediate connections between gates.

- **Output Connections:** The final output connections were made using green wires to make them easily identifiable.