CS M51A and EE M16 Summer 2016 Section 1

Logic Design of Digital Systems

Dr. Yutao He

Verilog Lab #2 - Design of Combinational Systems

Due: July 31st, 2016

(1) **Name:** Liu (Last) Jerry (First)

Student ID: 404474229

Signature: Jerry Liu

(2) **Name:** Chen (Last) Haojie (First)

Student ID: 204449491

Signature: Haojie Chen

Date: 07/31/2016

Result	
Correctness	
Creativity	
Report	
Total Score	

Project 2 Report, Title Page Above

Jerry Liu, 404474229
Haojie Chen, 204449491

July 31, 2016

Abstract

In this project, we will design a combinational circuit that converts BCD code to drive the display of a seven-segment LED device. When a decimal number from 0 to 9 is entered, each segment of the device will be shaded (ON) or unshaded (OFF) to display the number. We design this project from pencile-and-paper worksheet to Verilog implementation, shown in this report.

3 The Switching Functions of the Circuit

3.1 Binary-level Specification

Inputs: $\underline{x} = (x_3, x_2, x_1, x_0)$, $x_i \in \{0, 1\}$

Outputs: $\underline{z} = (a, b, c, d, e, f, g)$, $a, b, c, d, e, f, g \in \{0, 1\}$

Truth Table:

Decimal No.	$x_3x_2x_1x_0$	a	b	c	d	e	f	g
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	1	0	1	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	1	0	1	1
10	1010	-	-	-	-	-	-	-
11	1011	-	-	-	-	-	-	-
12	1100	-	-	-	-	-	-	-
13	1101	-	-	-	-	-	-	-
14	1110	-	-	-	-	-	-	-
15	1111	-	-	-	-	-	-	-

Table 1: Binary-level Truth Table

3.2 Switching Functions

The final switching functions we implement is shown below:

$a = ((x_2' + x_1 + x_0)' + (x_3 + x_2 + x_1 + x_0')')'$ $\quad b = ((x_2' + x_1 + x_0')' + (x_2' + x_1' + x_0)')'$

$c = ((x_2 + x_1' + x_0)' + (x_2 + x_1' + x_0)')'$ $\quad d = ((x_2' + x_1 + x_0)' + (x_3 + x_2 + x_1 + x_0')' + (x_2' + x_1' + x_0')')'$

$e = ((x_2' + x_1 + x_0)' + x_0)'$ $\quad f = ((x_3 + x_2 + x_0')' + (x_2 + x_1' + x_0)' + (x_2' + x_1' + x_0')')'$

$g = ((x_3 + x_2 + x_1)' + (x_2' + x_1' + x_0')')'$

3.3 Schematic

The schematic of our implementation is shown in Figure 1 and Figure 4 in Appendix.

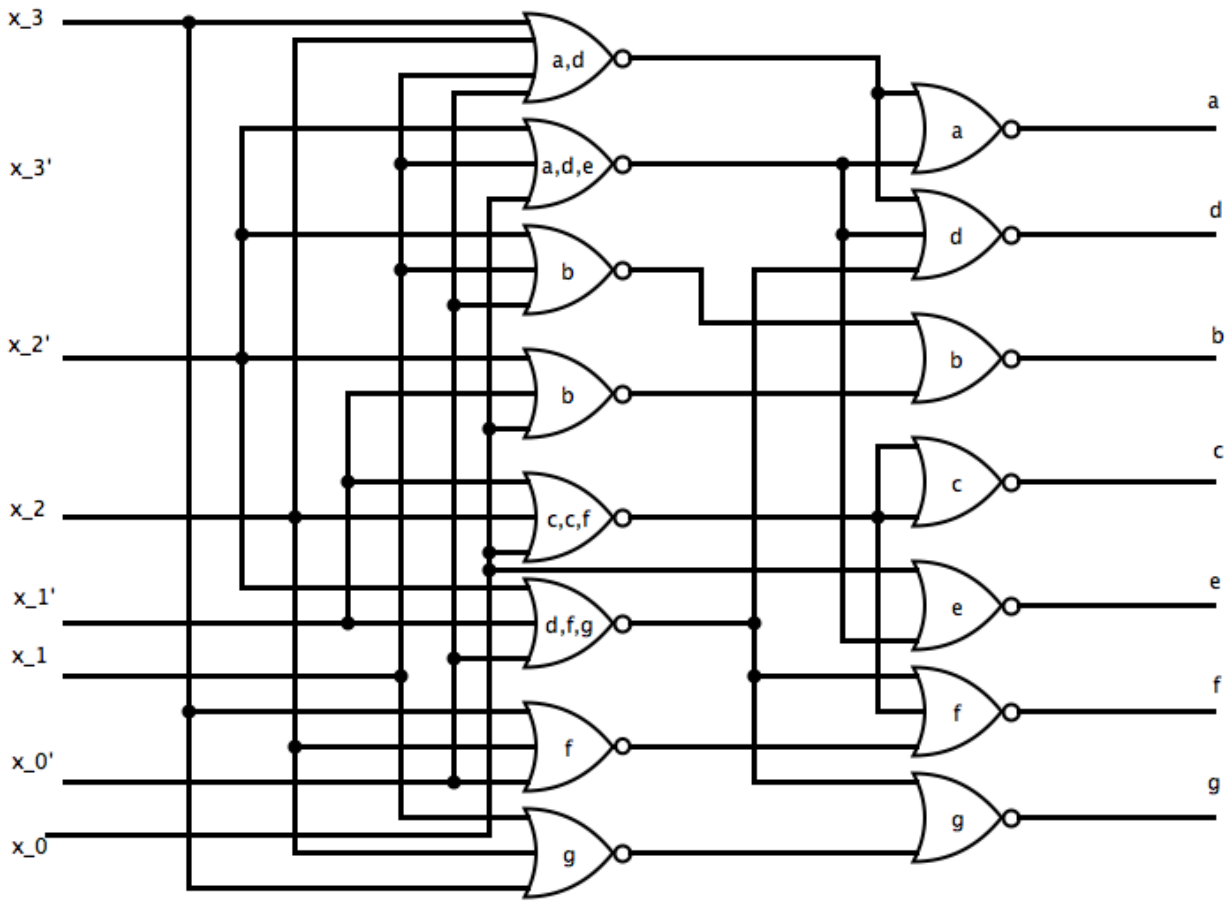


Figure 1: The Schematic Design of Minimum Number of Gates

4 Verilog File

4.1 Circuit Implementation

csm51a_proj2.v

```
module csm51a_proj2(
    input x0,
    input x1,
    input x2,
    input x3,
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
    output g
    );
    wire n1_out, n2_out, n4_out, n5_out, n7_out, n9_out;
    //  x_2' + x_1 + x_0: a, d, e
    nor n1(n1_out, !x2, x1, x0);
    //  x_3 + x_2 + x_1 + x_0': a, d
    nor n2(n2_out, x3, x2, x1, !x0);
    //  output a
    nor n3(a, n1_out, n2_out);

    //  x_2' + x_1 + x_0': b
    nor n4(n4_out, !x2, x1, !x0);
    //  x_2' + x_1' + x_0: b
    nor n5(n5_out, !x2, !x1, x0);
    //  output b
    nor n6(b, n4_out, n5_out);

    //  x_2 + x_1' + x_0: c, f
    nor n7(n6_out, x2, !x1, x0);
    //  output c
    nor n8(c, n6_out, n6_out);

    //  x_2' + x_1' + x_0': d, f, g
    nor n9(n7_out, !x2, !x1, !x0);
    //  output d
    nor n10(d, n1_out, n2_out, n7_out);

    //  output e
    nor n11(e, n1_out, x0);

    //  x_3 + x_2 + x_0': f
    nor n12(n8_out, x3, x2, !x0);
```

```
    nor n13(f, n6_out, n7_out, n8_out);

    //  x_3 + x_2 + x_1: g
    nor n14(n9_out, x3, x2, x1);
    //  output g
    nor n15(g, n7_out, n9_out);
endmodule
```

4.2 Testbench

csm51a_proj2_tb.v

```
module csm51a_proj2_tb(
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
    output g
    );
    //  4-bit is enough for BCD
    reg [3:0] i;

    //  Input
    reg x3, x2, x1, x0;

    csm51a_proj2 plusOneSecond1(.x3(x3), .x2(x2), .x1(x1), .x0(x0), .a(a), .b(b), .

    initial
    begin
        for (i = 0; i < 10; i = i + 1)
        begin
            //  assign each bit separately
            {x3, x2, x1, x0} = i;
            //  delay
            #5;
        end
        $finish;
    end
endmodule
```

5 Simulation Result

Here we want to construct and test all possible cases in the truth table in the appendix: Table 4. The results are shown clearly in the Figure 2:

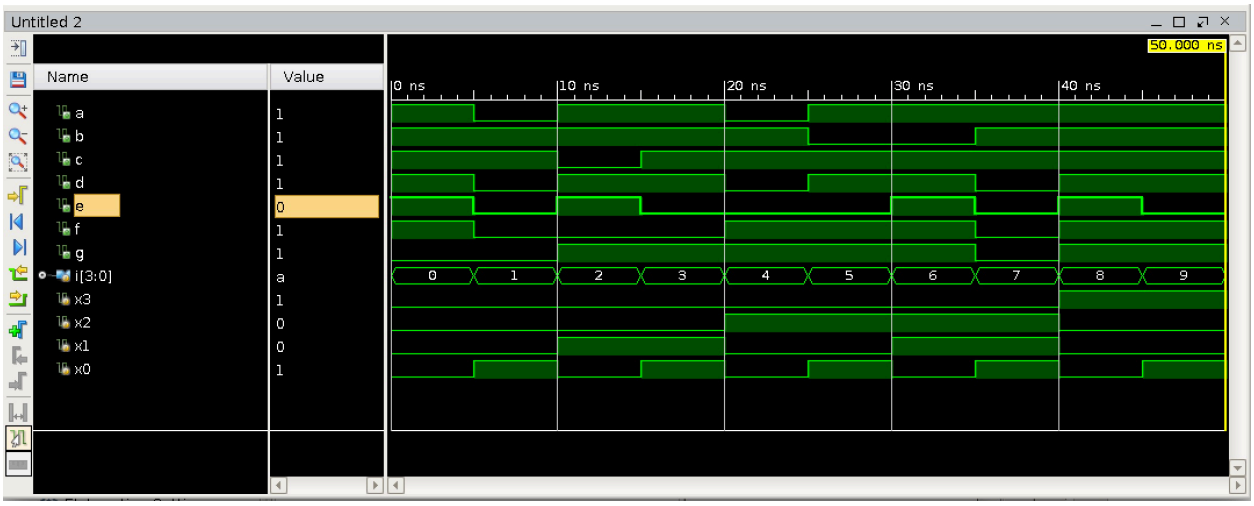


Figure 2: Screenshot of the Simulation Result

As we can see, the output a, b, c, d, e, f, g behaves exactly as what the truth table, Table 4, specifies. Note that green regions correspond to the time for a specific variable to have value 1, and for each variable, its value variations are represented on the same line as the variable.

6 The Design Review

Through this project, we have learned how to:

1. Encode and decode multiple inputs and outputs using different encoding schemes
2. Use the encoded information to draw truth table
3. Use the results in truth table to draw Karnaugh maps in order to get the minimal switching expression for each output in the form of product of sums (canonical form)
4. Transform the OR-AND network into two-level NOR-NOR network using bubble logic
5. Share terms to reduce the number of NOR gates by changing switching expressions.

During the implementation, the part that requires the most work is to minimize the switching expression for each output. To solve this, we have two possible approach: using Quine-McCluskey algorithm or Karnaugh maps. We decide to use Karnaugh maps to do the minimization, and then use QM algorithm to check the final results.
Other than to minimize the switching expressions, we have also minimized the network by sharing terms to reduce the number of repeated NOR gates. This method reduces the original 24-gate network down to 18-gate network, decreasing the delay time largely. The schematic is shown in Figure 3
However, minimal expression for each output does not give us the minimum number of gates. We

found that instead of keeping minimization for every output, we can actually share more terms by not using minimal expression for some of the outputs by looking at K-Maps. This further reduces the number of gates from 18 to 15 as shown in Figure 1.
When we implement the network using Verilog in the Vivado software, we have found that the schematic automatically generated by the software cannot display NOR gates directly. Therefore, we decide to use another software called Logisim to draw the schematic, and the final result displays the network all in NOR gates just as we expected.
To us, the most important aspect of this project is to wisely transform the original OR-AND network into NOR-NOR network to efficiently reduce the size and delay of the network. It shows us that modification on the type of gates to create an equivalent network sometimes yields more efficient results, and we should be familiar with the transformations between different types of network in order to make smarter design.

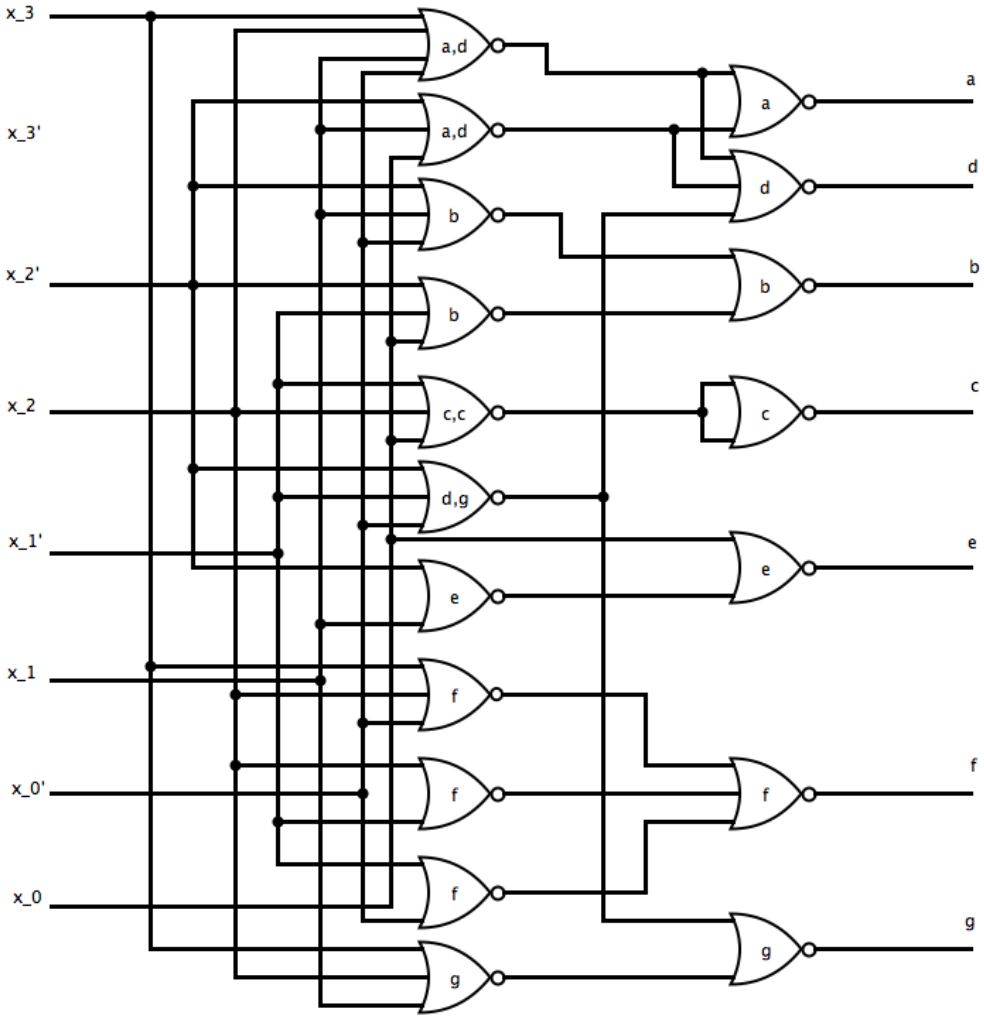


Figure 3: The Schematic Design of Minimal Expression for All Outputs

A Appendix - The Detailed Design Worksheet

A.1 Inputs and Outputs of the system

Inputs: $\underline{x} = (x_3, x_2, x_1, x_0)$, $x_i \in \{0, 1\}$
Outputs: $\underline{z} = (a, b, c, d, e, f, g)$, $a, b, c, d, e, f, g \in \{\text{ON, OFF}\}$

A.2 Encoding Schemes of Inputs and Outputs

The input and output encoding scheme is shown in Table 2 and Table 3 respectively.

Decimal No.	0	1	2	3	4	5	6	7
BCD Code	0000	0001	0010	0011	0100	0101	0110	0111
Decimal No.	8	9	10	11	12	13	14	15
BCD Code	1000	1001	1010	1011	1100	1101	1110	1111

Table 2: Input and Output Encoding Scheme

Segment State	**Binary Code**
ON	1
OFF	0

Table 3: Output Encoding Scheme

A.3 Truth Table

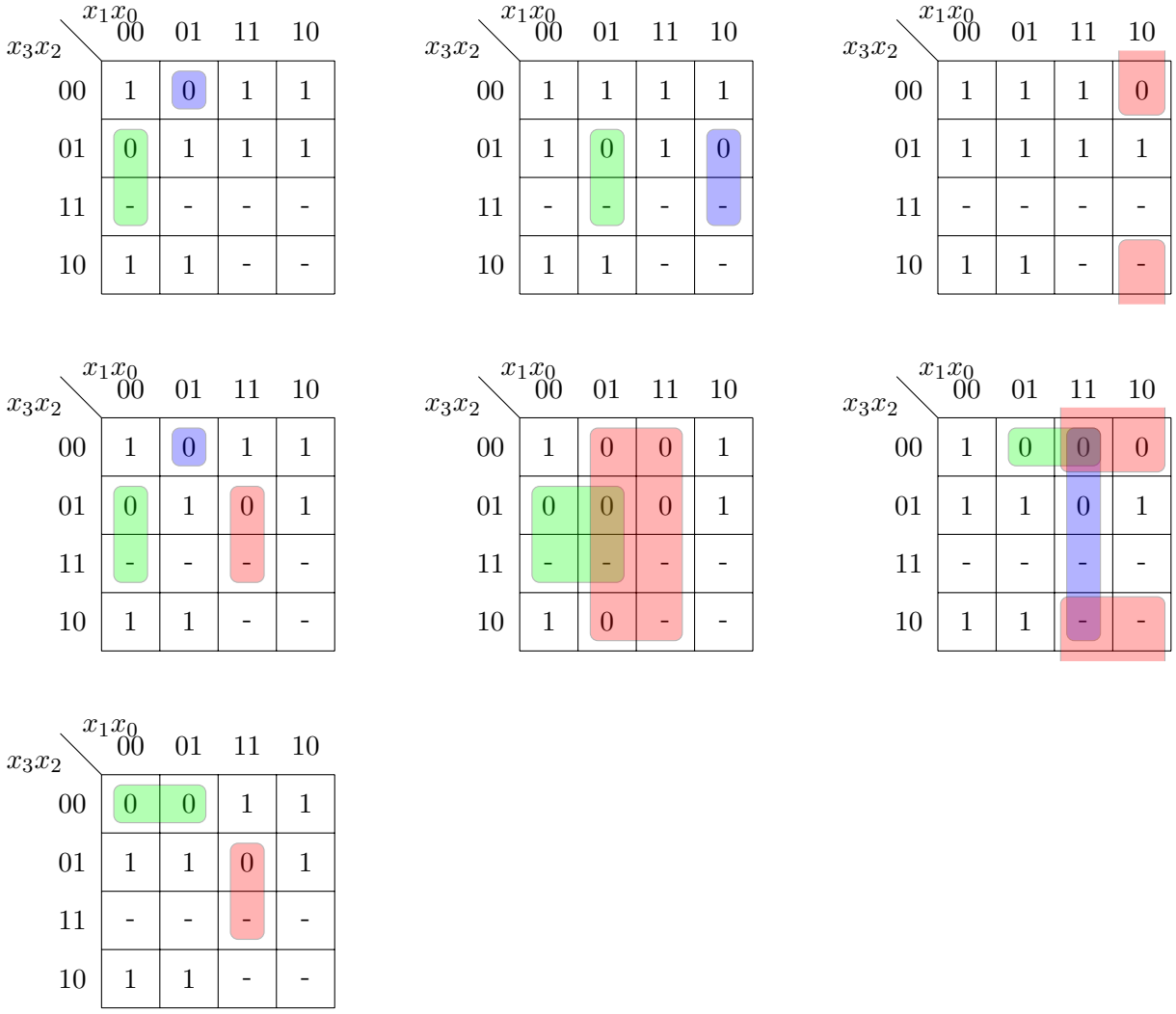
The Binary Truth Table is shown in Table 4

Decimal No.	$x_3x_2x_1x_0$	a	b	c	d	e	f	g
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1
5	0101	1	0	1	1	0	1	1
6	0110	1	0	1	1	1	1	1
7	0111	1	1	1	0	0	0	0
8	1000	1	1	1	1	1	1	1
9	1001	1	1	1	1	0	1	1
10	1010	-	-	-	-	-	-	-
11	1011	-	-	-	-	-	-	-
12	1100	-	-	-	-	-	-	-
13	1101	-	-	-	-	-	-	-
14	1110	-	-	-	-	-	-	-
15	1111	-	-	-	-	-	-	-

Table 4: The Truth Table for the Display

A.4 Minimization

We use K-Map to minimize outputs because it is easier to type.
Below are K-Maps for a, b, c, d, e, f, g respectively.



So we get the minimal switching expression for every output from K-Maps.

$$a = (x_2' + x_1 + x_0)(x_3 + x_2 + x_1 + x_0') \qquad b = (x_2' + x_1 + x_0')(x_2' + x_1' + x_0)$$
$$c = (x_2 + x_1' + x_0) \qquad d = (x_2' + x_1 + x_0)(x_3 + x_2 + x_1 + x_0')(x_2' + x_1' + x_0')$$
$$e = (x_2' + x_1)(x_0') \qquad f = (x_3 + x_2 + x_0')(x_2 + x_1')(x_1' + x_0')$$
$$g = (x_3 + x_2 + x_1)(x_2' + x_1' + x_0')$$

A.5 Transformation

We will transform two-level OR-AND networks to two-level NOR-NOR networks by bubble logic.

$$
\begin{aligned}
a &= (x_2' + x_1 + x_0)(x_3 + x_2 + x_1 + x_0') \\
&= ((x_2' + x_1 + x_0)' + (x_3 + x_2 + x_1 + x_0')')' \\
b &= (x_2' + x_1 + x_0')(x_2' + x_1' + x_0) \\
&= ((x_2' + x_1 + x_0')' + (x_2' + x_1' + x_0)')' \\
c &= (x_2 + x_1' + x_0) \\
&= ((x_2 + x_1' + x_0)' + (x_2 + x_1' + x_0)')' \\
d &= (x_2' + x_1 + x_0)(x_3 + x_2 + x_1 + x_0')(x_2' + x_1' + x_0') \\
&= ((x_2' + x_1 + x_0)' + (x_3 + x_2 + x_1 + x_0')' + (x_2' + x_1' + x_0')')' \\
e &= (x_2' + x_1)(x_0') \\
&= ((x_2' + x_1)' + x_0)' \\
f &= (x_3 + x_2 + x_0')(x_2 + x_1')(x_1' + x_0') \\
&= ((x_3 + x_2 + x_0')' + (x_2 + x_1')' + (x_1' + x_0')')' \\
g &= (x_3 + x_2 + x_1)(x_2' + x_1' + x_0') \\
&= ((x_3 + x_2 + x_1)' + (x_2' + x_1' + x_0')')'
\end{aligned}
$$

A.6 Final Minimal Expression

We did not use the minimum expression for each output, since we can share more gates to reduce the total number of gates. The schematic design of "minimal expression for every output" is shown in Figure 3.

A.6.1 Switching Expressions

$$
\begin{aligned}
a &= ((x_2' + x_1 + x_0)' + (x_3 + x_2 + x_1 + x_0')')' & b &= ((x_2' + x_1 + x_0')' + (x_2' + x_1' + x_0)')' \\
c &= ((x_2 + x_1' + x_0)' + (x_2 + x_1' + x_0)')' & d &= ((x_2' + x_1 + x_0)' + (x_3 + x_2 + x_1 + x_0')' + (x_2' + x_1' + x_0')')' \\
e &= ((x_2' + x_1 + x_0)' + x_0)' & f &= ((x_3 + x_2 + x_0')' + (x_2 + x_1' + x_0)' + (x_2' + x_1' + x_0')')' \\
g &= ((x_3 + x_2 + x_1)' + (x_2' + x_1' + x_0')')'
\end{aligned}
$$

To keep gate counts as low as possible, we use this K-Map for output e, f to share more gates.

x_3x_2 \ x_1x_0	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	-	-	-	-
10	1	0	-	-

x_3x_2 \ x_1x_0	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	-	-	-	-
10	1	1	-	-

A.6.2 Schematic

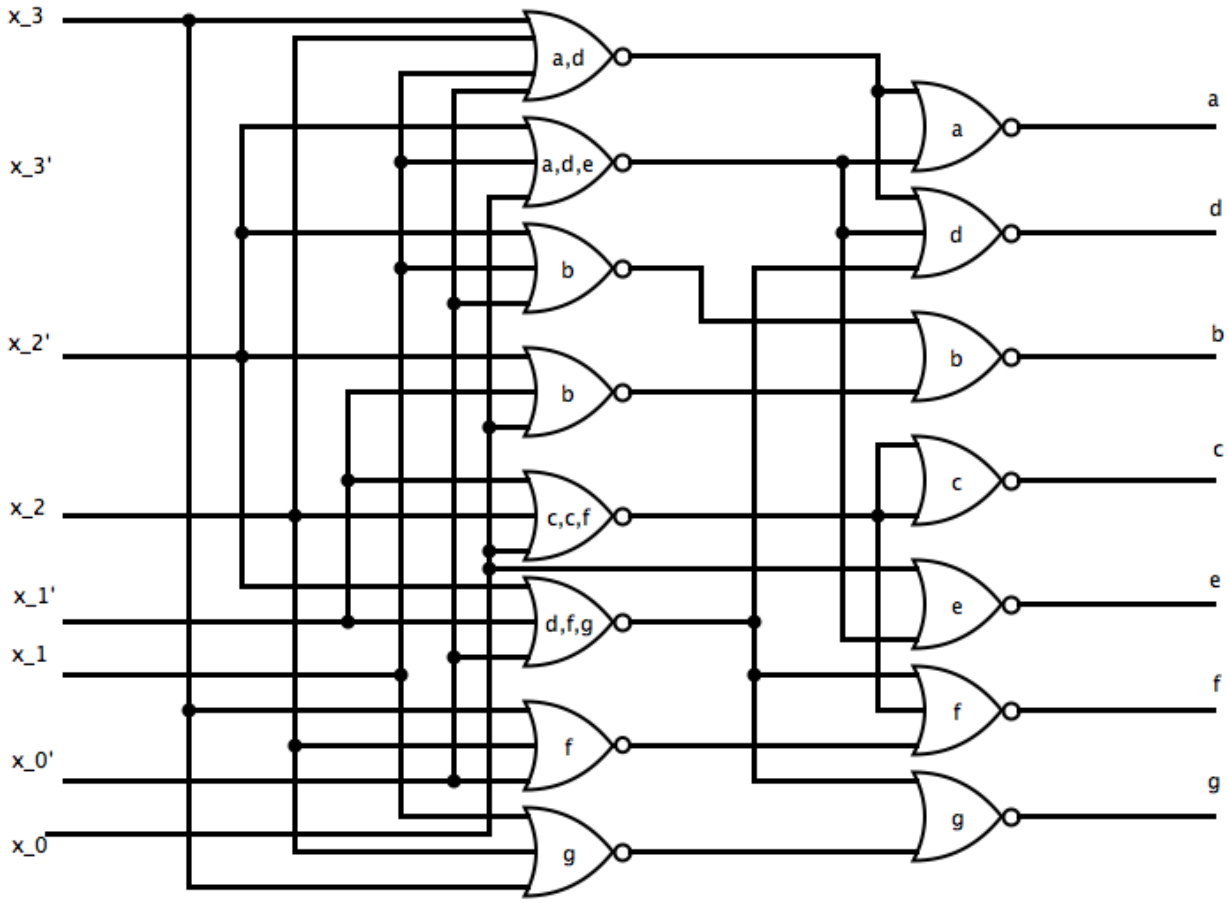


Figure 4: The Schematic Design of Minimum Number of Gates