Combinational Logic Design using VHDL

Lab Exercises on December 4, 2020

Department of Electronics and Computer Engineering
Pulchowk Campus, Lalitpur

Ashlesh Pandey

PUL074BEX007

Contents

1 Introduction **1**
1.1 Combinational Logic Circuit 1
1.2 VHDL Basics 1
1.2.1 Entity 1
1.2.2 Architecture 2
1.2.3 Configuration 2
1.3 Modeling Styles in VHDL 2
1.3.1 Dataflow Modeling 2
1.3.2 Behavioral Modeling 2
1.3.3 Structural Modeling 2
1.4 VHDL Test Bench 3

2 Objectives **3**

3 Lab Experiment Environment **3**
3.1 Xilinx ISE (Integrated Synthesis Environment) Design Suite 3
3.2 Logic Friday 3

4 Lab Problems **3**

5 Observations **24**

6 Discussion **40**

Additional References **41**

List of Figures

1 Combinational Logic Circuit 1
2 Logic circuit to be implemented in Problem 1 4
3 BCD to Gray code boolean expression reduction using Karnaugh map 7
4 Boolean expression reduction using Karnaugh map- Problem 3 10
5 SOP and POS boolean expression reduction using Karnaugh map 14
6 4:1 MUX using three 2:1 MUX as basic building blocks 20
7 4-bit adder/subtractor implementation using four 1-bit adders 22
8 Observed waveform for Problem 1 24
9 Observed RTL schematic for Problem 1 25
10 Observed waveform for Problem 2 25
11 Observed RTL schematic for Problem 2 25
12 Observed waveform for Problem 3 26
13 Observed RTL schematic for Problem 3 26
14 Observed waveform for Problem 4: SOP 26
15 Observed RTL schematic for Problem 4: SOP 27
16 Observed waveform for Problem 4: POS 27
17 Observed RTL schematic for Problem 4: POS 27
18 Observed waveform for Problem 5 28
19 Observed RTL schematic for Problem 5 28
20 Observed waveform for Problem 6 28
21 Observed RTL schematic for Problem 6 29
22 Observed waveform for Problem 7: 4-bit adder 30
23 Observed waveform for Problem 7: 4-bit subtractor 35
24 Observed RTL schematic for Problem 7 40

List of Tables

1 Truth table for BCD to Gray code conversion 7
2 Truth table for Problem 3 10
3 Truth table for Problem 4 13
4 Truth table for 2:1 multiplexer 18
5 Truth table for 1-bit full-adder 22

Listings

1 Syntax for entity declaration . . . 1
2 Syntax for entity declaration using generic . . . 2
3 Syntax for architecture declaration . . . 2
4 Syntax for configuration declaration . . . 2

Problem 1

5 Dataflow model . . . 4
6 Behavorial model . . . 5
7 AND gate implementation . . . 5
8 OR gate implementation . . . 5
9 AND gate implementation of two variables with one being inverted . . . 5
10 Structural model . . . 6
11 Testbench for all posible cases . . . 6

Problem 2

12 Dataflow model . . . 8
13 Behavorial model . . . 8
14 XOR gate implementation using only NOR . . . 9
15 Structural model . . . 9
16 Testbench for all posible cases . . . 10

Problem 3

17 Dataflow model . . . 11
18 Behavorial model . . . 11
19 NOT gate implementation using only NOR . . . 11
20 3-input OR gate implementation using only NOR . . . 11
21 3-input AND gate implementation using only NOR . . . 12
22 Structural model . . . 12
23 Testbench for all posible cases . . . 12

Problem 4: SOP

24 Dataflow model . . . 14
25 Behavorial model . . . 14
26 NOT gate implementation using only NOR . . . 15
27 3-input OR gate implementation using only NOR . . . 15
28 3-input AND gate implementation using only NOR . . . 15
29 Structural model . . . 16
30 Testbench for all posible cases . . . 16

Problem 4: POS

31 Dataflow model . . . 16
32 Behavorial model . . . 17
33 Structural model . . . 17
34 Testbench for all posible cases . . . 18

Problem 5

35 2:1 MUX implementation using WHEN-ELSE statement 19
36 2:1 MUX implementation using IF-THEN-ELSE statement 19
37 Testbench for all possible cases 19

Problem 6

38 2:1 MUX implementation using WHEN-ELSE statement 20
39 4:1 MUX implementation using three 2:1 MUX: Structural model 21
40 Testbench for all possible cases 21

Problem 7

41 XOR gate implementation 23
42 Full adder implementation 23
43 4-bit adder/subtractor implementation using four 1-bit full-adder: Structural model 23
44 Testbench for all possible cases 24

1 Introduction

1.1 Combinational Logic Circuit

Combinational logic circuits consist of inputs, two or more basic logic gates and outputs. The logic gates are combined in such a way that the output state depends entirely on the input states. Combinational logic circuits don't have memory, timing or feedback loops, i.e. their operation is instantaneous and doesn't depend on the outputs. Examples of common combinational logic circuits include: half adders, full adders, multiplexers, demultiplexers, encoders and decoders.

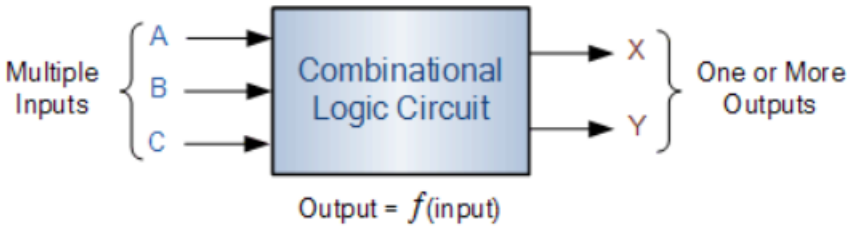


Figure 1: Combinational Logic Circuit

1.2 VHDL Basics

VHDL stands for Very High-Speed Integration Circuit HDL (Hardware Description Language). It is an IEEE (Institute of Electrical and Electronics Engineers) standard hardware description language that is used to describe and simulate the behavior of complex digital circuits. VHDL also includes design management features, and features that allow precise modeling of events that occur over time. VHDL is used for the following purposes:

- For describing hardware
- As a modeling language
- For simulation and synthesis of hardware
- For performance estimation

1.2.1 Entity

Entity is used to specify the input and output ports of the circuit. An entity usually has one or more ports that can be inputs (in), outputs (out), input-outputs (inout), or buffer. An entity may also include a set of generic values that are used to declare properties of the circuit.

Entity Declaration

```
ENTITY entity_name IS
    PORT (
    port_1_name : mode data_type;
    ort_2_name : mode data_type;
      .....
    Port_n_name : mode data_type
    );
END entity_name;
```

Listing 1: Syntax for entity declaration

If an entity is generic, then it must be declared before the ports. Generic does not have a mode, so it can only pass information into the entity.

Entity Declaration using generic

```
1  ENTITY entity_name IS
2      GENERIC (
3                  generic_1_name : data_type;
4                  generic_2_name : data_type;
5                    ........
6                  generic_n_name : data_type
7                   );
8    PORT (
9              port_1_name : mode data_type;
10              port_2_name : mode data_type;
11                ........
12              Port_n_name : mode data_type
13               );
14     END entity_name;
```

Listing 2: Syntax for entity declaration using generic

1.2.2 Architecture

Architecture is the actual description of the design, which is used to describe how the circuit operates. It can contain both concurrent and sequential statements.

Architecture Declaration

```
1  ARCHITECTURE architecture_name OF entity_name
       IS
2  BEGIN
3      (concurrent statements )
4  END architecture_name;
```

Listing 3: Syntax for architecture declaration

1.2.3 Configuration

A configuration defines how the design hierarchy is linked together. It is also used to associate architecture with an entity.

Configuration Declaration

```
1  CONFIGURATION configuration_name OF
       entity_name IS
2  FOR architecture_name
3    FOR instance_label : component_name
4        USE ENTITY library_name.entity_name(
             architecture_name);
5  END FOR;
6  END FOR;
7  END [CONFIGURATION] [configuration_name];
```

Listing 4: Syntax for configuration declaration

1.3 Modeling Styles in VHDL

1.3.1 Dataflow Modeling

Dataflow modeling can be described based on the boolean expression. It shows how the data flows from input to output. It works on concurrent execution.

1.3.2 Behavioral Modeling

Behavioral modeling is used to execute statements sequentially. It shows that how the system performs according to the current statement. Behavioral modeling may contain process statements, sequential statements, signal assignment statements, and wait statements.

1.3.3 Structural Modeling

Structural modeling is used to specify the functionality and structure of the circuit. Structural modeling contain signal declarations, component instances, and port maps in component instance.

1.4 VHDL Test Bench

An enclosing model called a test bench is used to test VHDL model. It similar to a real-life test bench where hardware that is under test is subjected to signal generators and outputs are observed. A VHDL test bench consists of an architecture body containing an instance of the component to be tested, known as unit under test (uut) and processes that generate sequences of values on signals connected to the component instance. The architecture body may also contain processes that test that the component instance produces the expected values on its output signals. Simulators are used to monitor the changes in the output signals.

2 Objectives

The primary objectives of this lab experiment is to understand programming concepts in VHDL for a Field Programmable Gate Array(FPGA). VHDL coding concepts for a FPGA will enable us to write codes capable of:

- Implementing combinational circuits.
- Implementing test benches to verify the working of combinational circuits.

3 Lab Experiment Environment

The lab experiments were performed virtually via simulation softwares. The basic usages of these tools will allow us to write, simulate and synthesize combinational circuts.

3.1 Xilinx ISE (Integrated Synthesis Environment) Design Suite

Xilinx ISE (Integrated Synthesis Environment) is a discontinued software tool from Xilinx for synthesis and analysis of HDL designs. It enables the developer to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. VHDL test benches are simulated on the ISE Simulator that provides a full-featured HDL simulator.

3.2 Logic Friday

Logic Friday is a freeware tool for students, hobbyists, and engineers who work with legacy digital logic circuits based on standard IC packages. Logic Friday takes the help of Espresso logic design minimiser to efficiently reduce the functions in an electronic design. Instead of using the traditional Karnaugh map method of min term reduction, the program manipulates the function iteratively to give a closely approximated result, eliminating redundancy. Truth tables and simlified boolean expressions are generated using Logic Friday for the purpose of checking the manually computed values.

4 Lab Problems

Problem 1
Write VHDL code to implement the logic circuit shown in below figure, which has 4 inputs (x1, x2, x3 and x4) and one output (f).

a. Provide the following architectural styles:
- **Dataflow style**
- **Behavioral style**

- **Structural style**

b. Write a VHDL test bench to verify the operation of the logic circuit.

c. Provide a simulation waveform depicting all possible input cases.

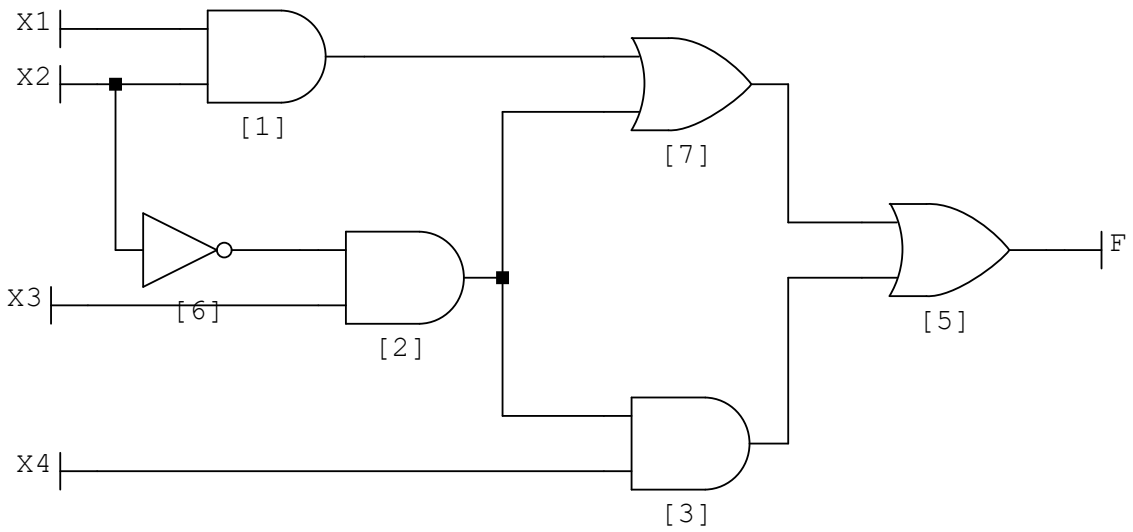


Figure 2: Logic circuit to be implemented in Problem 1

Figure 2 is a four input, one output combinational circuit. The gate diagram was re-created using Logic Friday which gave the following minimized logic equation for the output.

$$F = X1.X2 + X2'.X3$$

q1_df.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q1 IS PORT (
5          A, B, C, D : IN STD_LOGIC;
6          F : OUT STD_LOGIC
7  );
8  END q1;
9
10 ARCHITECTURE dataflow OF q1 IS
11 BEGIN
12         F <= (A AND B) OR (NOT B AND C);
13 END dataflow;
```

Listing 5: Dataflow model

q1_be.vhd

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY q1 IS PORT (
        A, B, C, D : IN STD_LOGIC;
        F : OUT STD_LOGIC
);
END q1;

ARCHITECTURE behavorial OF q1 IS
        SIGNAL F1, F2, F3, F4 : STD_LOGIC;

BEGIN
        be_proc : PROCESS (A, B, C, D, F1, F2, F3,
            F4)

        BEGIN
                F1 <= A AND B;
                F2 <= NOT B AND C;
                F <= F1 OR F2;
        END PROCESS be_proc;

END behavorial;
```

Listing 6: Behavorial model

and1.vhd

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY and1 IS PORT (
        i1, i2 : IN STD_LOGIC;
        o1 : OUT STD_LOGIC
);

END and1;

ARCHITECTURE dataflow OF and1 IS
BEGIN
        o1 <= i1 AND i2;
END dataflow;
```

Listing 7: AND gate implementation

or1.vhd

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY and1 IS PORT (
        i1, i2 : IN STD_LOGIC;
        o1 : OUT STD_LOGIC
);

END and1;

ARCHITECTURE dataflow OF and1 IS
BEGIN
        o1 <= i1 AND i2;
END dataflow;
```

Listing 8: OR gate implementation

andnot.vhd

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY andnot IS PORT (
        i1, i2 : IN STD_LOGIC;
        o1 : OUT STD_LOGIC
);

END andnot;

ARCHITECTURE dataflow OF andnot IS
BEGIN
        o1 <= NOT i1 AND i2;
END dataflow;
```

Listing 9: AND gate implementation of two variables with one being inverted

q1_st.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q1 IS PORT (
5          A, B, C, D : IN STD_LOGIC;
6          F : OUT STD_LOGIC
7  );
8  END q1;
9
10 ARCHITECTURE structural OF q1 IS
11         SIGNAL F1, F2 : STD_LOGIC;
12
13         COMPONENT and1 IS PORT (
14                 i1, i2 : IN STD_LOGIC;
15                 o1 : OUT STD_LOGIC
16                 );
17         END COMPONENT;
18         COMPONENT andnot IS PORT (
19                 i1, i2 : IN STD_LOGIC;
20                 o1 : OUT STD_LOGIC
21                 );
22         END COMPONENT;
23
24         COMPONENT or1 IS PORT (
25                 i1, i2 : IN STD_LOGIC;
26                 o1 : OUT STD_LOGIC
27                 );
28         END COMPONENT;
29
30 BEGIN
31         C1 : and1 PORT MAP(i1 => A, i2 => B, o1
                => F1);
32         C2 : andnot PORT MAP(i1 => B, i2 => C,
                o1 => F2);
33         C3 : or1 PORT MAP(i1 => F1, i2 => F2, o1
                => F);
34 END structural;
```

Listing 10: Structural model

q1_tb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY q1_tb IS
6  END q1_tb;
7
8  ARCHITECTURE behavioral OF q1_tb IS
9          -- component declaaration for the Unit Under
                Test (UUT)
10         COMPONENT q1
11                 PORT (
12                         A : IN STD_LOGIC;
13                         B : IN STD_LOGIC;
14                         C : IN STD_LOGIC;
15                         D : IN STD_LOGIC;
16                         F : OUT STD_LOGIC
17                 );
18         END COMPONENT;
19
20         SIGNAL input_vector :
                STD_LOGIC_VECTOR(3 DOWNTO
                0) := "0000";
21         SIGNAL output : STD_LOGIC;
22
23 BEGIN
24         --INSTANTIATE the unit under test
25         uut : q1 PORT MAP(
26                 A => input_vector(3),
27                 B => input_vector(2),
28                 C => input_vector(1),
29                 D => input_vector(0),
30                 F => output
31         );
32
33         --stimulus process
34         stim_proc : PROCESS
35
36         BEGIN
37                 FOR index IN 0 TO 15 LOOP
38                         input_vector <=
                                STD_LOGIC_VECTOR(
                                to_unsigned(index, 4));
39                         WAIT FOR 50 ns;
40                 END LOOP;
41         END PROCESS;
42 END behavioral;
```

Listing 11: Testbench for all posible cases

Problem 2
Write VHDL code to design a logic circuit that implements the truth table of BCD-to-Gray code converter.

a. Use Karnaugh maps to simplify the output function.

b. Provide the following architectural styles:

- **Dataflow style**
- **Behavioral style**
- **Structural style using only NOR gates**

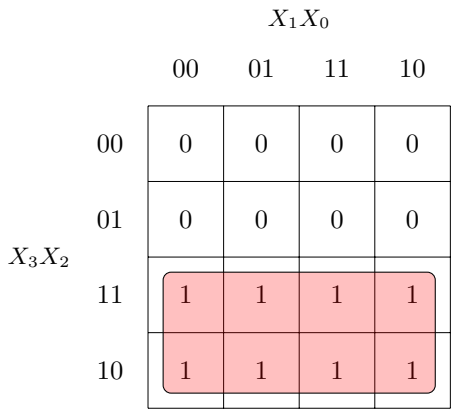
c. Write a VHDL test bench to verify the operation of the logic circuit.

d. Provide a simulation waveform depicting all possible input cases.

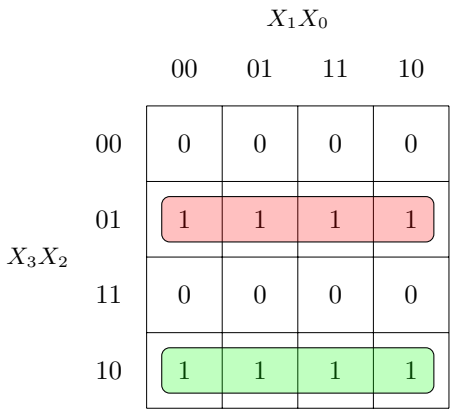
4-bit BCD numbers				Gray code numbers			
X3	X2	X1	X0	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Table 1: Truth table for BCD to Gray code conversion

Table 1 was entered into Logic Friday as a 4-input, 4-output truth table. The boolean equations for the outputs are,
$Y3 = X3$,
$Y2 = X3'X2 + X3X2'$,
$Y1 = X2'X1 + X2X1'$,
$Y0 = X1'X0 + X1X0'$

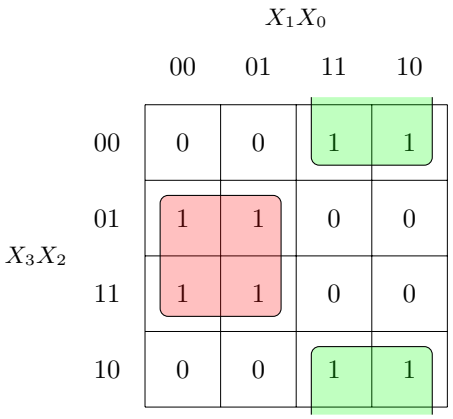


(3.1) K-Map reduction for Y3

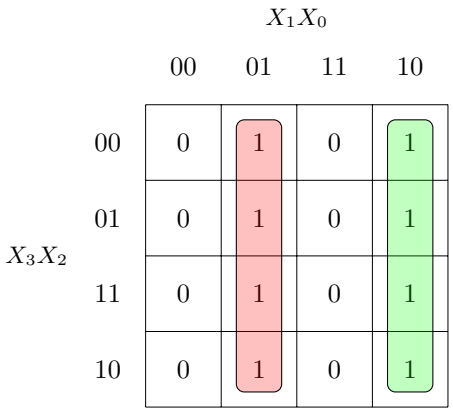


(3.2) K-Map reduction for Y2

Figure 3: BCD to Gray code boolean expression reduction using Karnaugh map



(3.3) K-Map reduction for Y1



(3.4) K-Map reduction for Y0

Figure 3: BCD to Gray code boolean expression reduction using Karnaugh map (continued)

From Figure 3 we can deduct the boolean expressions for the gray code outputs as:

$$Y3 = X3$$
$$Y2 = X2 \oplus X3$$
$$Y1 = X1 \oplus X2$$
$$Y0 = X0 \oplus X1$$

q2_df.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q2 IS PORT (
5          X3, X2, X1, X0 : IN STD_LOGIC;
6          Y3, Y2, Y1, Y0 : OUT STD_LOGIC
7  );
8  END q2;
9
10 ARCHITECTURE dataflow OF q2 IS
11
12 BEGIN
13         Y3 <= X3;
14         Y2 <= X2 XOR X3;
15         Y1 <= X1 XOR X2;
16         Y0 <= X0 XOR X1;
17 END dataflow;
```

Listing 12: Dataflow model

q2_be.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q2 IS PORT (
5          X3, X2, X1, X0 : IN STD_LOGIC;
6          Y3, Y2, Y1, Y0 : OUT STD_LOGIC
7  );
8  END q2;
9
10 ARCHITECTURE behavorial OF q2 IS
11
12 BEGIN
13         example : PROCESS (X3, X2, X1, X0)
14
15         BEGIN
16                 Y3 <= X3;
17                 Y2 <= X2 XOR X3;
18                 Y1 <= X1 XOR X2;
19                 Y0 <= X0 XOR X1;
20         END PROCESS example;
21
22 END behavorial;
```

Listing 13: Behavorial model

xor_nor.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY xor_nor IS PORT (
5          a, b : IN STD_LOGIC;
6          o : OUT STD_LOGIC
7  );
8  END xor_nor;
9
10 ARCHITECTURE behavorial OF xor_nor IS
11         SIGNAL nota, notb, xnorab : STD_LOGIC;
12
13 BEGIN
14         xor_nor : PROCESS (a, b, nota, notb, xnorab)
15
16         BEGIN
17                 nota <= a NOR a;
18                 notb <= b NOR b;
19                 xnorab <= (a NOR notb) NOR (nota
                           NOR b);
20                 o <= xnorab NOR xnorab;
21
22         END PROCESS xor_nor;
23
24 END behavorial;
```

Listing 14: XOR gate implementation using only NOR

q2_st.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q2 IS PORT (
5          X3, X2, X1, X0 : IN STD_LOGIC;
6          Y3, Y2, Y1, Y0 : OUT STD_LOGIC
7  );
8  END q2;
9
10 ARCHITECTURE structural OF q2 IS
11
12         COMPONENT xor_nor IS PORT (
13                 a, b : IN STD_LOGIC;
14                 o : OUT STD_LOGIC
15                 );
16         END COMPONENT;
17
18 BEGIN
19         C0 : xor_nor PORT MAP(a => X3, b => '0',
                   o => Y3);
20         C1 : xor_nor PORT MAP(a => X3, b => X2,
                   o => Y2);
21         C2 : xor_nor PORT MAP(a => X2, b => X1,
                   o => Y1);
22         C3 : xor_nor PORT MAP(a => X1, b => X0,
                   o => Y0);
23
24 END structural;
```

Listing 15: Structural model

q2_tb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY q2_tb IS
6  END q2_tb;
7
8  ARCHITECTURE behavioral OF q2_tb IS
9          -- component declaaration for the Unit Under
               Test (UUT)
10         COMPONENT q2
11                 PORT (
12                         X3 : IN STD_LOGIC;
13                         X2 : IN STD_LOGIC;
14                         X1 : IN STD_LOGIC;
15                         X0 : IN STD_LOGIC;
16                         Y3 : OUT STD_LOGIC;
17                         Y2 : OUT STD_LOGIC;
18                         Y1 : OUT STD_LOGIC;
19                         Y0 : OUT STD_LOGIC
20                 );
21         END COMPONENT;
22
23         SIGNAL input_vector :
               STD_LOGIC_VECTOR(3 DOWNTO
               0) := "0000";
24         SIGNAL output_vector :
               STD_LOGIC_VECTOR(3 DOWNTO
               0) := "0000";
25
26 BEGIN
27         --INSTANTIATE the unit under test
28         uut : q2 PORT MAP(
29                 X3 => input_vector(3),
30                 X2 => input_vector(2),
31                 X1 => input_vector(1),
32                 X0 => input_vector(0),
33
34                 Y3 => output_vector(3),
35                 Y2 => output_vector(2),
36                 Y1 => output_vector(1),
37                 Y0 => output_vector(0)
38         );
39
40         --stimulus process
41         stim_proc : PROCESS
42
43         BEGIN
44                 FOR index IN 0 TO 15 LOOP
45                         input_vector <=
                               STD_LOGIC_VECTOR(
                               to_unsigned(index, 4));
46                         WAIT FOR 50 ns;
47                 END LOOP;
48         END PROCESS;
49 END behavioral;
```

Listing 16: Testbench for all posible cases

Problem 3
Write VHDL code to implement the logic function (F) with the three input variables x1, x2, and x3. The function (F) is equal to 1 if and only if two variables are equal to 1; otherwise, it is equal to zero.

a. Draw a truth table for the function (F), and use Karnaugh maps to simplify.

b. Provide the following architectural styles:
- **Dataflow style**
- **Behavioral style**
- **Structural style using only NOR gates**

c. Write a VHDL test bench to verify the operation of the logic circuit.

d. Provide a simulation waveform depicting all possible input cases.

Input			Output
X3	X2	X1	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Table 2: Truth table for Problem 3

		X_2X_1			
		00	01	11	10
X_3	0	0	0	1	0
	1	0	1	0	1

Figure 4: Boolean expression reduction using Karnaugh map- Problem 3

From Figure 4 we can deduct the boolean expression for the output as:

$$Y = X1X2X3' + X1X2'X3 + X1'X2X3$$

q3_df.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q3 IS PORT (
5          X1, X2, X3 : IN STD_LOGIC;
6          F : OUT STD_LOGIC
7  );
8  END q3;
9
10 ARCHITECTURE dataflow OF q3 IS
11 BEGIN
12         F <= (X1 AND X2 AND NOT X3) OR (X3
                   AND (X1 XOR X2));
13 END dataflow;
```

Listing 17: Dataflow model

q3_be.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q3 IS PORT (
5          X1, X2, X3 : IN STD_LOGIC;
6          F : OUT STD_LOGIC
7  );
8  END q3;
9
10 ARCHITECTURE behavioral OF q3 IS
11         SIGNAL A1, A2, A3 : STD_LOGIC;
12
13 BEGIN
14
15         PROCESS (X1, X2, X3, A1, A2, A3)
16         BEGIN
17                 A1 <= X1 AND X2 AND NOT X3;
18                 A2 <= X1 XOR X2;
19                 A3 <= A2 AND X3;
20                 F <= A1 OR A3;
21         END PROCESS;
22 END behavioral;
```

Listing 18: Behavorial model

q3_not.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q3_not IS
5          PORT (
6                  a : IN STD_LOGIC;
7                  o : OUT STD_LOGIC);
8  END q3_not;
9
10 ARCHITECTURE behavorial OF q3_not IS
11
12 BEGIN
13         q3_not : PROCESS (a)
14
15         BEGIN
16                 o <= a NOR a;
17         END PROCESS q3_not;
18
19 END behavorial;
```

Listing 19: NOT gate implementation using only NOR

q3_or.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q3_or IS PORT (
5          a, b, c : IN STD_LOGIC;
6          o : OUT STD_LOGIC
7  );
8  END q3_or;
9
10 ARCHITECTURE behavorial OF q3_or IS
11         SIGNAL G1, G2, G3 : STD_LOGIC;
12
13 BEGIN
14         q3_or : PROCESS (a, b, c, G1, G2, G3)
15
16         BEGIN
17                 G1 <= a NOR b;
18                 G2 <= G1 NOR G1;
19                 G3 <= G2 NOR c;
20                 o <= G3 NOR G3;
21         END PROCESS q3_or;
22
23 END behavorial;
```

Listing 20: 3-input OR gate implementation using only NOR

q3_and.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q3_and IS PORT (
5          a, b, c: IN STD_LOGIC;
6          o : OUT STD_LOGIC
7  );
8  END q3_and;
9
10 ARCHITECTURE behavorial OF q3_and IS
11         SIGNAL F1, F2, F3, F4, F5 : STD_LOGIC;
12 BEGIN
13         q3_and : PROCESS (a, b, c, F1, F2, F3, F4, F5
               )
14         BEGIN
15                 F1 <= a NOR a;
16                 F2 <= b NOR b;
17                 F3 <= c NOR c;
18                 F4 <= F1 NOR F2;
19                 F5 <= F4 NOR F4;
20                 o <= F5 NOR F3;
21
22         END PROCESS q3_and;
23 END behavorial;
```

Listing 21: 3-input AND gate implementation using only NOR

q3_st.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q3 IS PORT (
5          X1, X2, X3 : IN STD_LOGIC;
6          F : OUT STD_LOGIC
7  );
8  END q3;
9
10 ARCHITECTURE structural OF q3 IS
11         SIGNAL A1, A2, A3, A4, A5, A6 :
               STD_LOGIC;
12
13         COMPONENT q3_and IS PORT (
14                 a, b, c : IN STD_LOGIC;
15                 o : OUT STD_LOGIC
16                 );
17         END COMPONENT;
18
19         COMPONENT q3_or IS PORT (
20                 a, b, c : IN STD_LOGIC;
21                 o : OUT STD_LOGIC
22                 );
23         END COMPONENT;
24
25         COMPONENT q3_not IS PORT (
26                 a : IN STD_LOGIC;
27                 o : OUT STD_LOGIC
28                 );
29         END COMPONENT;
30
31 BEGIN
32         N1 : q3_not PORT MAP(a => X1, o => A1);
33         N2 : q3_not PORT MAP(a => X2, o => A2);
34         N3 : q3_not PORT MAP(a => X3, o => A3);
35         N4 : q3_and PORT MAP(a => A1, b => X2,
               c => X3, o => A4);
36         N5 : q3_and PORT MAP(a => A2, b => X1,
               c => X3, o => A5);
37         N6 : q3_and PORT MAP(a => A3, b => X1,
               c => X2, o => A6);
38         N7 : q3_or PORT MAP(a => A4, b => A5, c
               => A6, o => F);
39 END structural;
```

Listing 22: Structural model

q3_tb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY q3_tb IS
6  END q3_tb;
7
8  ARCHITECTURE behavioral OF q3_tb IS
9          COMPONENT q3
10                 PORT (
11                         X1, X2, X3 : IN STD_LOGIC;
12                         F : OUT STD_LOGIC
13                 );
14         END COMPONENT;
15
16         SIGNAL input : STD_LOGIC_VECTOR(2
               DOWNTO 0) := "000";
17         SIGNAL output : STD_LOGIC := '0';
18
19 BEGIN
20         uut : q3 PORT MAP(
21                 X3 => input(2),
22                 X2 => input(1),
23                 X1 => input(0),
24                 F => output
25         );
26
27         stim_proc : PROCESS
28
29         BEGIN
30                 FOR index IN 0 TO 7 LOOP
31                         input <=
                               STD_LOGIC_VECTOR(
                               TO_UNSIGNED(index,
                               3));
32                         WAIT FOR 50 ns;
33                 END LOOP;
34         END PROCESS;
35
36 END behavioral;
```

Listing 23: Testbench for all posible cases

Problem 4
Write VHDL code to implement the implicit sum of products (SOP) and product of sums (POS) logic functions.

$$f(x_1, x_2, x_3, x_4) = \sum (m_0, m_1, m_4, m_5, m_8, m_9, m_{14}, m_{15})$$
$$f(x_1, x_2, x_3, x_4) = \prod (M_0, M_1, M_5, M_8, M_9, M_{13}, M_{15})$$

a. Draw a truth table for the function (F), and use Karnaugh maps to simplify.

b. Provide the following architectural styles:
- **Dataflow style**
- **Behavioral style**
- **Structural style using only NOR gates**

c. Write a VHDL test bench to verify the operation of the logic circuit.

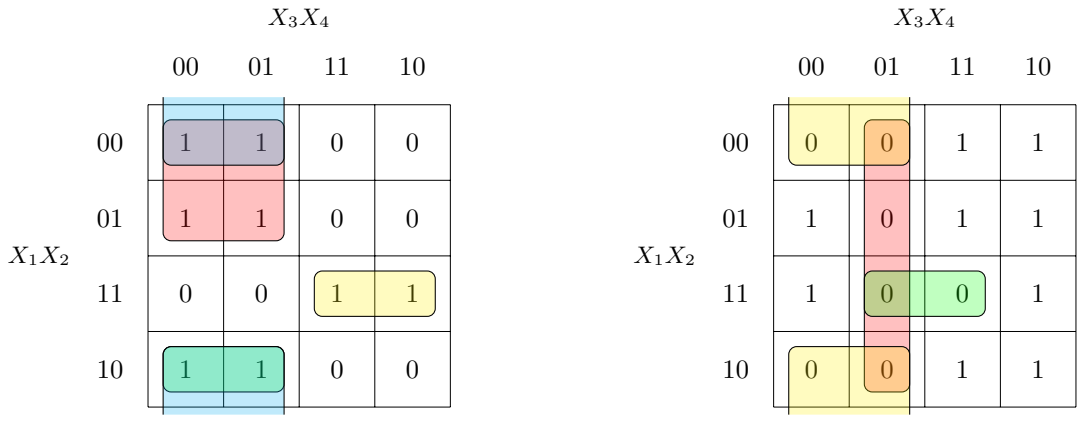
d. Provide a simulation waveform depicting all possible input cases.

Input				Output	
X1	X2	X3	X4	F1	F2
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	0	1
1	1	0	1	0	0
1	1	1	0	1	1
1	1	1	1	1	0

F1 is the output for SOP logic function and F2 is the output for POS logic function.

Table 3: Truth table for Problem 4

Table 3 was entered into Logic Friday as a 4-input, 2-output truth table. The boolean equations for the outputs are,
$F1 = X1X2X3 + X2'X3' + X1'X3'$,
$F2 = (X1' + X2' + X4')(X3 + X4')(X2 + X3)$



(5.1) K-Map reduction for F1

(5.2) K-Map reduction for F2

Figure 5: SOP and POS boolean expression reduction using Karnaugh map

From Figure 5 we can deduct the boolean expression for the output as:

$$F1 = X1'X3' + X1X2X3 + X2'X3'$$
$$F2 = (X3 + X4')(X1' + X2' + X4')(X2 + X3)$$

q4sop_df.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_sop IS PORT (
5          X1, X2, X3, X4 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q4_sop;
9
10 ARCHITECTURE dataflow OF q4_sop IS
11
12 BEGIN
13         Y <= (NOT X1 AND NOT X3) OR (NOT
               X2 AND NOT X3) OR (X1 AND X2
               AND X3);
14 END dataflow;
```

Listing 24: Dataflow model

q4sop_be.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_sop IS PORT (
5          X1, X2, X3, X4 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q4_sop;
9
10 ARCHITECTURE behavorial OF q4_sop IS
11         SIGNAL Y1, Y2, Y3 : STD_LOGIC;
12
13 BEGIN
14
15         PROCESS (X1, X2, X3, X4, Y1, Y2, Y3)
16
17         BEGIN
18                 Y1 <= NOT X1 AND NOT X3;
19                 Y2 <= NOT X2 AND NOT X3;
20                 Y3 <= X1 AND X2 AND X3;
21                 Y <= Y1 OR Y2 OR Y3;
22
23         END PROCESS;
24
25 END behavorial;
```

Listing 25: Behavorial model

q4_not.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_not IS
5          PORT (
6                  a : IN STD_LOGIC;
7                  o : OUT STD_LOGIC);
8  END q4_not;
9
10 ARCHITECTURE behavorial OF q4_not IS
11
12 BEGIN
13         q4_not : PROCESS (a)
14
15         BEGIN
16                 o <= a NOR a;
17         END PROCESS q4_not;
18
19 END behavorial;
```

Listing 26: NOT gate implementation using only NOR

q4_or.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_or IS PORT (
5          a, b, c : IN STD_LOGIC;
6          o : OUT STD_LOGIC
7  );
8  END q4_or;
9
10 ARCHITECTURE behavorial OF q4_or IS
11         SIGNAL G1, G2, G3 : STD_LOGIC;
12
13 BEGIN
14         q4_or : PROCESS (a, b, c, G1, G2, G3)
15
16         BEGIN
17                 G1 <= a NOR b;
18                 G2 <= G1 NOR G1;
19                 G3 <= G2 NOR c;
20                 o <= G3 NOR G3;
21         END PROCESS q4_or;
22
23 END behavorial;
```

Listing 27: 3-input OR gate implementation using only NOR

q4_and.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_and IS PORT (
5          a, b, c : IN STD_LOGIC;
6          o : OUT STD_LOGIC
7  );
8  END q4_and;
9
10 ARCHITECTURE behavorial OF q4_and IS
11         SIGNAL F1, F2, F3, F4, F5 : STD_LOGIC;
12 BEGIN
13         q4_and : PROCESS (a, b, c, F1, F2, F3, F4, F5
               )
14         BEGIN
15                 F1 <= a NOR a;
16                 F2 <= b NOR b;
17                 F3 <= c NOR c;
18                 F4 <= F1 NOR F2;
19                 F5 <= F4 NOR F4;
20                 o <= F5 NOR F3;
21
22         END PROCESS q4_and;
23 END behavorial;
```

Listing 28: 3-input AND gate implementation using only NOR

q4sop_st.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_sop IS PORT (
5          X1, X2, X3, X4 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q4_sop;
9
10 ARCHITECTURE structural OF q4_sop IS
11         SIGNAL A1, A2, A3, A4, A5, A6 :
               STD_LOGIC;
12
13         COMPONENT q4_and IS PORT (
14                 a, b, c : IN STD_LOGIC;
15                 o : OUT STD_LOGIC
16                 );
17         END COMPONENT;
18
19         COMPONENT q4_or IS PORT (
20                 a, b, c : IN STD_LOGIC;
21                 o : OUT STD_LOGIC
22                 );
23         END COMPONENT;
24
25         COMPONENT q4_not IS PORT (
26                 a : IN STD_LOGIC;
27                 o : OUT STD_LOGIC
28                 );
29         END COMPONENT;
30
31 BEGIN
32         N1 : q4_not PORT MAP(a => X1, o => A1);
33         N2 : q4_not PORT MAP(a => X2, o => A2);
34         N3 : q4_not PORT MAP(a => X3, o => A3);
35         N4 : q4_and PORT MAP(a => A1, b => A3,
                 c => '1', o => A4);
36         N5 : q4_and PORT MAP(a => A2, b => A3,
                 c => '1', o => A5);
37         N6 : q4_and PORT MAP(a => X1, b => X2,
                 c => X3, o => A6);
38         N7 : q4_or PORT MAP(a => A4, b => A5, c
                  => A6, o => Y);
39 END structural;
```

Listing 29: Structural model

q4sop_tb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY q4_sop_tb IS
6  END q4_sop_tb;
7
8  ARCHITECTURE behavioral OF q4_sop_tb IS
9          -- component declaaration for the Unit Under
               Test (UUT)
10         COMPONENT q4_sop
11                 PORT (
12                         X1 : IN STD_LOGIC;
13                         X2 : IN STD_LOGIC;
14                         X3 : IN STD_LOGIC;
15                         X4 : IN STD_LOGIC;
16                         Y : OUT STD_LOGIC
17                 );
18         END COMPONENT;
19
20         SIGNAL input_vector :
               STD_LOGIC_VECTOR(3 DOWNTO
               0) := "0000";
21         SIGNAL output : STD_LOGIC := '0';
22
23 BEGIN
24         --INSTANTIATE the unit under test
25         uut : q4_sop PORT MAP(
26                 X1 => input_vector(3),
27                 X2 => input_vector(2),
28                 X3 => input_vector(1),
29                 X4 => input_vector(0),
30                 Y => output
31         );
32
33         --stimulus process
34         stim_proc : PROCESS
35
36         BEGIN
37                 FOR index IN 0 TO 15 LOOP
38                         input_vector <=
                               STD_LOGIC_VECTOR(
                               to_unsigned(index, 4));
39                         WAIT FOR 50 ns;
40                 END LOOP;
41         END PROCESS;
42 END behavioral;
```

Listing 30: Testbench for all posible cases

q4pos_df.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_pos IS PORT (
5          X1, X2, X3, X4 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q4_pos;
9
10 ARCHITECTURE dataflow OF q4_pos IS
11
12 BEGIN
13         Y <= (X3 OR NOT X4) AND (X2 OR X3)
                AND (NOT X1 OR NOT X2 OR NOT
                X4);
14 END dataflow;
```

Listing 31: Dataflow model

q4pos_be.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_pos IS PORT (
5          X1, X2, X3, X4 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q4_pos;
9
10 ARCHITECTURE behavorial OF q4_pos IS
11         SIGNAL Y1, Y2, Y3 : STD_LOGIC;
12
13 BEGIN
14
15         PROCESS (X1, X2, X3, X4, Y1, Y2, Y3)
16
17         BEGIN
18                 Y1 <= X3 OR NOT X4;
19                 Y2 <= X2 OR X3;
20                 Y3 <= NOT X1 OR NOT X2 OR
                       NOT X4;
21                 Y <= Y1 AND Y2 AND Y3;
22
23         END PROCESS;
24
25 END behavorial;
```

Listing 32: Behavorial model

q4pos_st.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q4_pos IS PORT (
5          X1, X2, X3, X4 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q4_pos;
9
10 ARCHITECTURE structural OF q4_pos IS
11         SIGNAL A1, A2, A3, A4, A5, A6 :
                STD_LOGIC;
12
13         COMPONENT q4_and IS PORT (
14                 a, b, c : IN STD_LOGIC;
15                 o : OUT STD_LOGIC
16                 );
17         END COMPONENT;
18
19         COMPONENT q4_or IS PORT (
20                 a, b, c : IN STD_LOGIC;
21                 o : OUT STD_LOGIC
22                 );
23         END COMPONENT;
24
25         COMPONENT q4_not IS PORT (
26                 a : IN STD_LOGIC;
27                 o : OUT STD_LOGIC
28                 );
29         END COMPONENT;
30
31 BEGIN
32         N1 : q4_not PORT MAP(a => X1, o => A1);
33         N2 : q4_not PORT MAP(a => X2, o => A2);
34         N3 : q4_not PORT MAP(a => X4, o => A3);
35         N4 : q4_or PORT MAP(a => A1, b => A2, c
                => A3, o => A4);
36         N5 : q4_or PORT MAP(a => X3, b => A3, c
                => '0', o => A5);
37         N6 : q4_or PORT MAP(a => X2, b => X3, c
                => '0', o => A6);
38         N7 : q4_and PORT MAP(a => A4, b => A5,
                c => A6, o => Y);
39 END structural;
```

Listing 33: Structural model

q4pos_tb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY q4_pos_tb IS
6  END q4_pos_tb;
7
8  ARCHITECTURE behavioral OF q4_pos_tb IS
9          -- component declaaration for the Unit Under
               Test (UUT)
10         COMPONENT q4_pos
11                 PORT (
12                         X1 : IN STD_LOGIC;
13                         X2 : IN STD_LOGIC;
14                         X3 : IN STD_LOGIC;
15                         X4 : IN STD_LOGIC;
16                         Y : OUT STD_LOGIC
17                 );
18         END COMPONENT;
19
20         SIGNAL input_vector :
               STD_LOGIC_VECTOR(3 DOWNTO
               0) := "0000";
21         SIGNAL output : STD_LOGIC := '0';
22
23 BEGIN
24         --INSTANTIATE the unit under test
25         uut : q4_pos PORT MAP(
26                 X1 => input_vector(3),
27                 X2 => input_vector(2),
28                 X3 => input_vector(1),
29                 X4 => input_vector(0),
30                 Y => output
31         );
32
33         --stimulus process
34         stim_proc : PROCESS
35
36         BEGIN
37                 FOR index IN 0 TO 15 LOOP
38                         input_vector <=
                               STD_LOGIC_VECTOR(
                               to_unsigned(index, 4));
39                         WAIT FOR 50 ns;
40                 END LOOP;
41         END PROCESS;
42 END behavioral;
```

Listing 34: Testbench for all posible cases

Problem 5
Write VHDL code to implement a 2:1 MUX having inputs x1 and x2, select line s and output y.

a. Provide the following architectural implementations:

- **Using WHEN-ELSE statement**
- **Using IF-THEN-ELSE statement**

b. Write a VHDL test bench to verify the operation of the 2:1 MUX.

c. Provide a simulation waveform depicting all possible input cases.

Select	Input		Output
S	X2	X1	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Table 4: Truth table for 2:1 multiplexer

Table 4 was entered into Logic Friday as a 3-input, 1-output truth table. The boolean equation for the output is, $Y = SX2 + S'X1$

q5_when_else.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q5_mux2_1 IS PORT (
5          S, X1, X2 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q5_mux2_1;
9
10 ARCHITECTURE dataflow OF q5_mux2_1 IS
11 BEGIN
12         Y <= '1' WHEN ((NOT S AND X1) OR (S
               AND X2)) = '1' ELSE
13                 '0';
14 END dataflow;
```

Listing 35: 2:1 MUX implementation using WHEN-ELSE statement

q5_if_then_else.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q5_mux2_1 IS PORT (
5          S, X1, X2 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q5_mux2_1;
9
10 ARCHITECTURE behavioral OF q5_mux2_1 IS
11 BEGIN
12
13         PROCESS (S, X1, X2)
14         BEGIN
15                 IF ((NOT S AND X1) OR (S AND
                       X2)) = '1' THEN
16                         Y <= '1';
17                 ELSE
18                         Y <= '0';
19                 END IF;
20         END PROCESS;
21 END behavioral;
```

Listing 36: 2:1 MUX implementation using IF-THEN-ELSE statement

q5_tb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY q5_mux2_1_tb IS
6  END q5_mux2_1_tb;
7
8  ARCHITECTURE behavioral OF q5_mux2_1_tb IS
9          COMPONENT q5_mux2_1
10                 PORT (
11                         S : IN STD_LOGIC;
12                         X1 : IN STD_LOGIC;
13                         X2 : IN STD_LOGIC;
14                         Y : OUT STD_LOGIC
15                 );
16         END COMPONENT;
17
18         SIGNAL input : STD_LOGIC_VECTOR(2
               DOWNTO 0) := "000";
19         SIGNAL output : STD_LOGIC;
20
21 BEGIN
22         uut : q5_mux2_1 PORT MAP(
23                 S => input(2),
24                 X2 => input(1),
25                 X1 => input(0),
26                 Y => output
27         );
28
29         stim_proc : PROCESS
30
31         BEGIN
32                 FOR index IN 0 TO 7 LOOP
33                         input <=
                               STD_LOGIC_VECTOR(
                               TO_UNSIGNED(index,
                               3));
34                         WAIT FOR 50 ns;
35                 END LOOP;
36         END PROCESS;
37
38 END behavioral;
```

Listing 37: Testbench for all possible cases

Problem 6
Write VHDL code to implement a 4:1 MUX having inputs x1, x2, x3 and x4, select lines s1, s0 and output y using three 2:1 multiplexers as the basic building blocks.

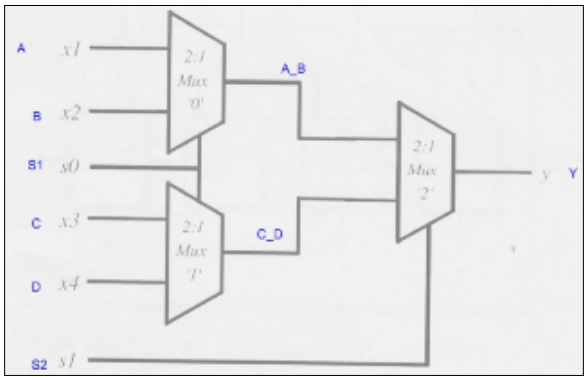


Figure 6: 4:1 MUX using three 2:1 MUX as basic building blocks

a. Use a hierarchical design approach:

1. Create component definitions in separate (.vhd) files. Use either Dataflow or Behavioral or Structural design styles.
2. Use Structural design style for the 4:1 MUX architecture:
   (a) Make use of 2:1 MUX component declaration
   (b) Make use of 2:1 MUX component instantiation.

b. Write a VHDL test bench to verify the operation of the 4:1 MUX.

c. Provide a simulation waveform depicting all possible input cases.

q5_when_else.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q5_mux2_1 IS PORT (
5          S, X1, X2 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q5_mux2_1;
9
10 ARCHITECTURE dataflow OF q5_mux2_1 IS
11 BEGIN
12         Y <= '1' WHEN ((NOT S AND X1) OR (S
               AND X2)) = '1' ELSE
13               '0';
14 END dataflow;
```

Listing 38: 2:1 MUX implementation using WHEN-ELSE statement

q6_st.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q6_mux4_1 IS PORT (
5          X1, X2, X3, X4, S0, S1 : IN STD_LOGIC;
6          Y : OUT STD_LOGIC
7  );
8  END q6_mux4_1;
9
10 ARCHITECTURE structural OF q6_mux4_1 IS
11         SIGNAL F1, F2 : STD_LOGIC;
12
13         COMPONENT q5_mux2_1 IS PORT (
14                 X1, X2, S : IN STD_LOGIC;
15                 Y : OUT STD_LOGIC
16                 );
17         END COMPONENT;
18
19 BEGIN
20         M0 : q5_mux2_1 PORT MAP(X1 => X1, X2
                => X2, S => S0, Y => F1);
21         M1 : q5_mux2_1 PORT MAP(X1 => X3, X2
                => X4, S => S0, Y => F2);
22         M2 : q5_mux2_1 PORT MAP(X1 => F1, X2
                => F2, S => S1, Y => Y);
23 END structural;
```

Listing 39: 4:1 MUX implementation using three 2:1 MUX: Structural model

q6_tb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY q6_mux4_1_tb IS
6  END q6_mux4_1_tb;
7
8  ARCHITECTURE behavioral OF q6_mux4_1_tb IS
9          -- component declaaration for the Unit Under
               Test (UUT)
10         COMPONENT q6_mux4_1
11                 PORT (
12                         X1, X2, X3, X4, S0, S1 : IN
                               STD_LOGIC;
13                         Y : OUT STD_LOGIC
14                 );
15         END COMPONENT;
16
17         SIGNAL select_vector :
               STD_LOGIC_VECTOR(1 DOWNTO
               0) := "00";
18         SIGNAL input_vector :
               STD_LOGIC_VECTOR(3 DOWNTO
               0) := "0000";
19         SIGNAL output : STD_LOGIC := '0';
20
21 BEGIN
22         --INSTANTIATE the unit under test
23         uut : q6_mux4_1 PORT MAP(
24                 S0 => select_vector(0),
25                 S1 => select_vector(1),
26                 X1 => input_vector(3),
27                 X2 => input_vector(2),
28                 X3 => input_vector(1),
29                 X4 => input_vector(0),
30                 Y => output
31         );
32
33         --stimulus process
34         stim_proc : PROCESS
35
36         BEGIN
37                 FOR selector IN 0 TO 3 LOOP
38                         select_vector <=
                               STD_LOGIC_VECTOR(
                               to_unsigned(selector, 2));
39                         FOR index IN 0 TO 15 LOOP
40                         input_vector <=
                               STD_LOGIC_VECTOR(
                               to_unsigned(index, 4));
41                                 WAIT FOR 40 ns;
42                         END LOOP;
43                 END LOOP;
44         END PROCESS;
45 END behavioral;
```

Listing 40: Testbench for all possible cases

Problem 7
Write VHDL code to implement a 4-bit adder/subtractor using four 1-bit full adders.

a. Use a Structural architecture style with hierarchical design approach:

- Use 1-bit adder as the basic building block.
- Implement the 4-bit adder/subtractor using four 1-bit full adders.

b. Write a VHDL test bench to verify the operation of the 4-bit adder/subtractor.

c. Provide a simulation waveform depicting all possible input cases.

Input			Output	
X	Y	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 5: Truth table for 1-bit full-adder

Table 5 was entered into Logic Friday as a 3-input, 2-output truth table. The boolean equation for the outputs are,
$S = X \oplus Y \oplus Cin$
$Cout = XY + (X \oplus Y)Cin$

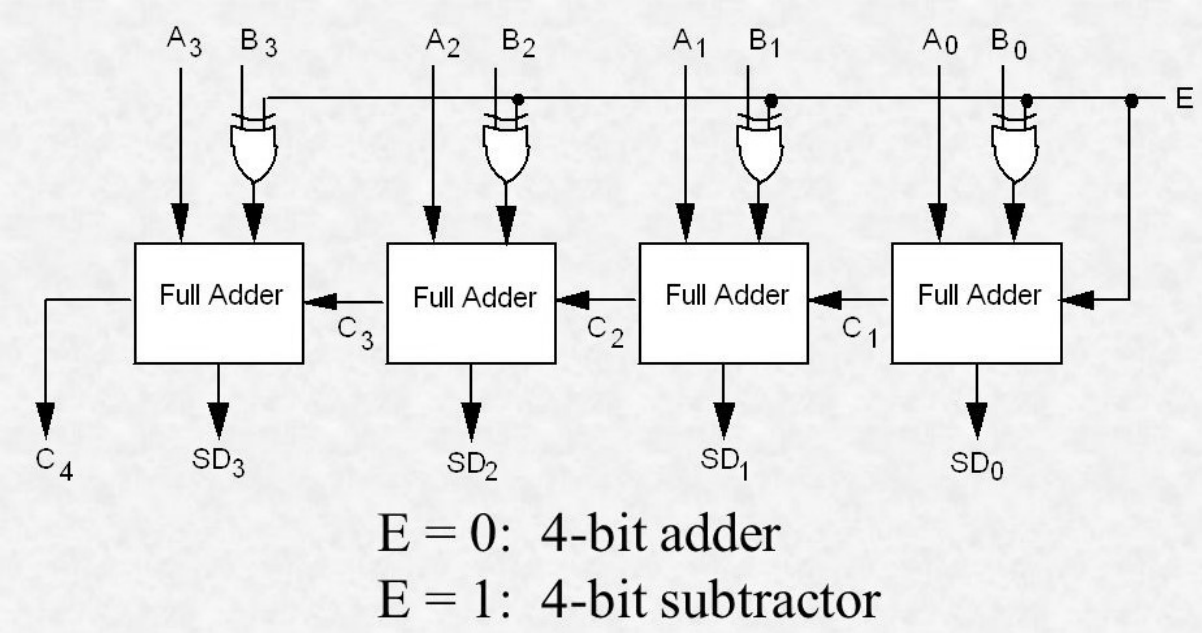


Figure 7: 4-bit adder/subtractor implementation using four 1-bit adders

q7_xor.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q7_xor IS PORT (
5          i1, i2 : IN STD_LOGIC;
6          o1 : OUT STD_LOGIC
7  );
8  END q7_xor;
9
10 ARCHITECTURE dataflow OF q7_xor IS
11 BEGIN
12         o1 <= i1 XOR i2;
13 END dataflow;
```

Listing 41: XOR gate implementation

full_adder.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY full_adder IS PORT (
5          i1, i2, cin : IN STD_LOGIC;
6          sum, cout : OUT STD_LOGIC
7  );
8  END full_adder;
9
10 ARCHITECTURE dataflow OF full_adder IS
11 BEGIN
12         sum <= i1 XOR i2 XOR cin;
13         cout <= (i1 AND i2) OR ((i1 XOR i2) AND
               cin);
14 END dataflow;
```

Listing 42: Full adder implementation

q7_st.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY q7_add_sub IS PORT (
5          X3, X2, X1, X0, Y3, Y2, Y1, Y0, A_S : IN
               STD_LOGIC;
6          S4, S3, S2, S1, S0 : OUT STD_LOGIC
7  );
8  END q7_add_sub;
9
10 ARCHITECTURE structural OF q7_add_sub IS
11         SIGNAL F0, F1, F2, F3, C1, C2, C3 :
               STD_LOGIC;
12
13         COMPONENT q7_xor IS PORT (
14                 i1, i2 : IN STD_LOGIC;
15                 o1 : OUT STD_LOGIC
16                 );
17         END COMPONENT;
18
19         COMPONENT full_adder IS PORT (
20                 i1, i2, cin : IN STD_LOGIC;
21                 sum, cout : OUT STD_LOGIC
22                 );
23         END COMPONENT;
24
25 BEGIN
26         A0 : q7_xor PORT MAP(i1 => A_S, i2 =>
                   Y0, o1 => F0);
27         A1 : full_adder PORT MAP(i1 => X0, i2 =>
                    F0, cin => A_S, sum => S0, cout =>
                   C1);
28
29         A2 : q7_xor PORT MAP(i1 => A_S, i2 =>
                   Y1, o1 => F1);
30         A3 : full_adder PORT MAP(i1 => X1, i2 =>
                    F1, cin => C1, sum => S1, cout => C2
                   );
31
32         A4 : q7_xor PORT MAP(i1 => A_S, i2 =>
                   Y2, o1 => F2);
33         A5 : full_adder PORT MAP(i1 => X2, i2 =>
                    F2, cin => C2, sum => S2, cout => C3
                   );
34
35         A6 : q7_xor PORT MAP(i1 => A_S, i2 =>
                   Y3, o1 => F3);
36         A7 : full_adder PORT MAP(i1 => X3, i2 =>
                    F3, cin => C3, sum => S3, cout => S4)
                   ;
37 END structural;
```

Listing 43: 4-bit adder/subtractor implementation using four 1-bit full-adder: Structural model

q7_tb.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  ENTITY q7_add_sub_tb IS
6  END q7_add_sub_tb;
7
8  ARCHITECTURE behavioral OF q7_add_sub_tb IS
9          -- component declaaration for the Unit Under
               Test (UUT)
10         COMPONENT q7_add_sub
11                 PORT (
12                         X3, X2, X1, X0, Y3, Y2, Y1, Y0,
                                   A_S : IN STD_LOGIC;
13                         S4, S3, S2, S1, S0 : OUT
                                   STD_LOGIC
14                 );
15         END COMPONENT;
16
17         SIGNAL addsub : STD_LOGIC := '1';
18         --Change addsub to '0' for 4-bit full-adder
19         SIGNAL input_vector1 :
               STD_LOGIC_VECTOR(3 DOWNTO
               0) := "0000";
20         SIGNAL input_vector2 :
               STD_LOGIC_VECTOR(3 DOWNTO
               0) := "0000";
21         SIGNAL output_vector :
               STD_LOGIC_VECTOR(4 DOWNTO
               0) := "00000";
22
23 BEGIN
24         --INSTANTIATE the unit under test
25         uut : q7_add_sub PORT MAP(
26                 A_S => addsub,
27                 X3 => input_vector1(3),
28                 X2 => input_vector1(2),
29                 X1 => input_vector1(1),
30                 X0 => input_vector1(0),
31
32                 Y3 => input_vector2(3),
33                 Y2 => input_vector2(2),
34                 Y1 => input_vector2(1),
35                 Y0 => input_vector2(0),
36
37                 S4 => output_vector(4),
38                 S3 => output_vector(3),
39                 S2 => output_vector(2),
40                 S1 => output_vector(1),
41                 S0 => output_vector(0)
42         );
43
44         --stimulus process
45         stim_proc : PROCESS
46
47         BEGIN
48                 FOR index1 IN 0 TO 15 LOOP
49                         input_vector1 <=
                               STD_LOGIC_VECTOR(
                               to_unsigned(index1, 4));
50                         FOR index2 IN 0 TO 15
                               LOOP
51                         input_vector2 <=
                               STD_LOGIC_VECTOR(
                               to_unsigned(index2, 4));
52                           WAIT FOR 50 ns;
53                         END LOOP;
54                 END LOOP;
55         END PROCESS;
56 END behavioral;
```

Listing 44: Testbench for all possible cases

5 Observations

The observations for the ISim simulator are printed to individual .pdf files using the print option available within the simulator. The time frames have been selected such that all the possible input cases are included at least once within the waveform. The expected outputs and the simulated waveform match leading us to a conclusion that the designed combinational circuits are functional. The RTL schematic for each question have been included with this report since it was also a major part of the observation. The RTL schematic shows the internal logic gates that are used starting from the top module. The schematics are taken for the structural models since it best describes the individual components in use.

Problem 1

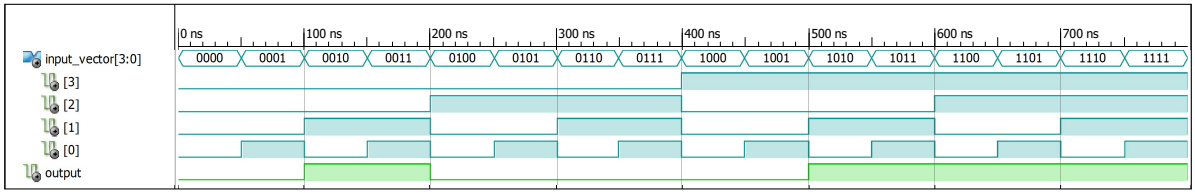


Figure 8: Observed waveform for Problem 1

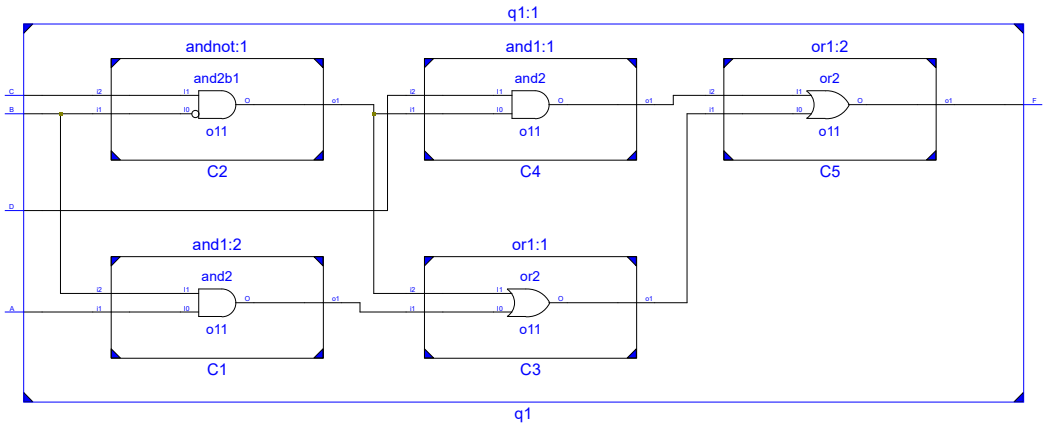


Figure 9: Observed RTL schematic for Problem 1

Problem 2

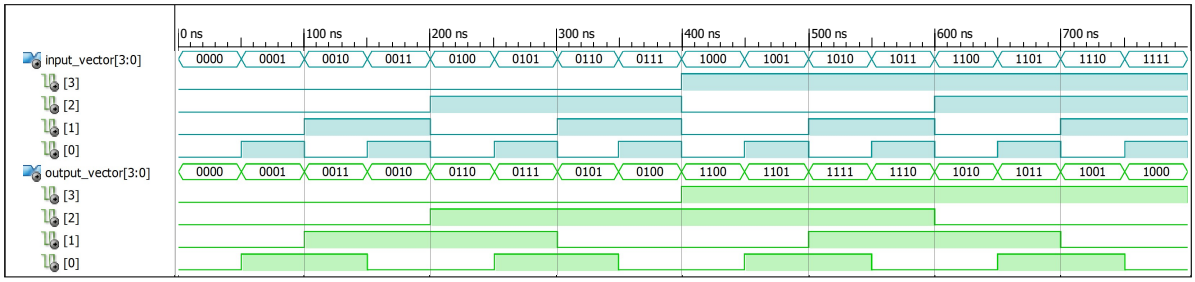


Figure 10: Observed waveform for Problem 2

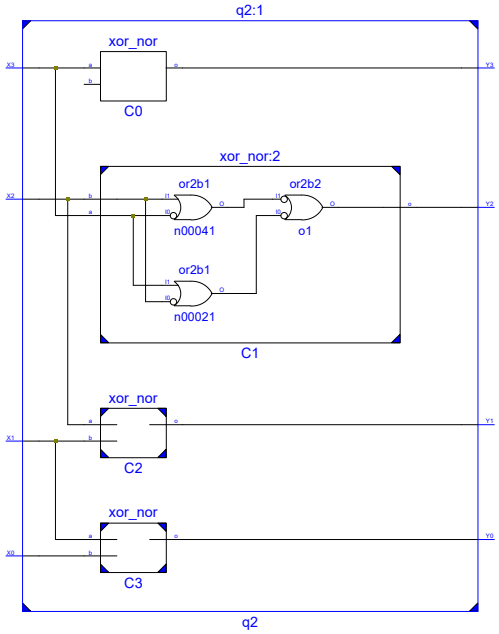


Figure 11: Observed RTL schematic for Problem 2

Problem 3

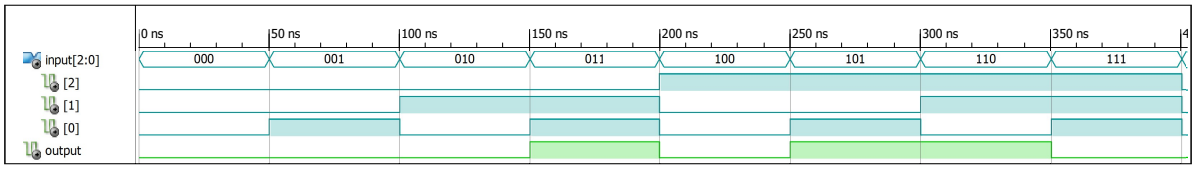


Figure 12: Observed waveform for Problem 3

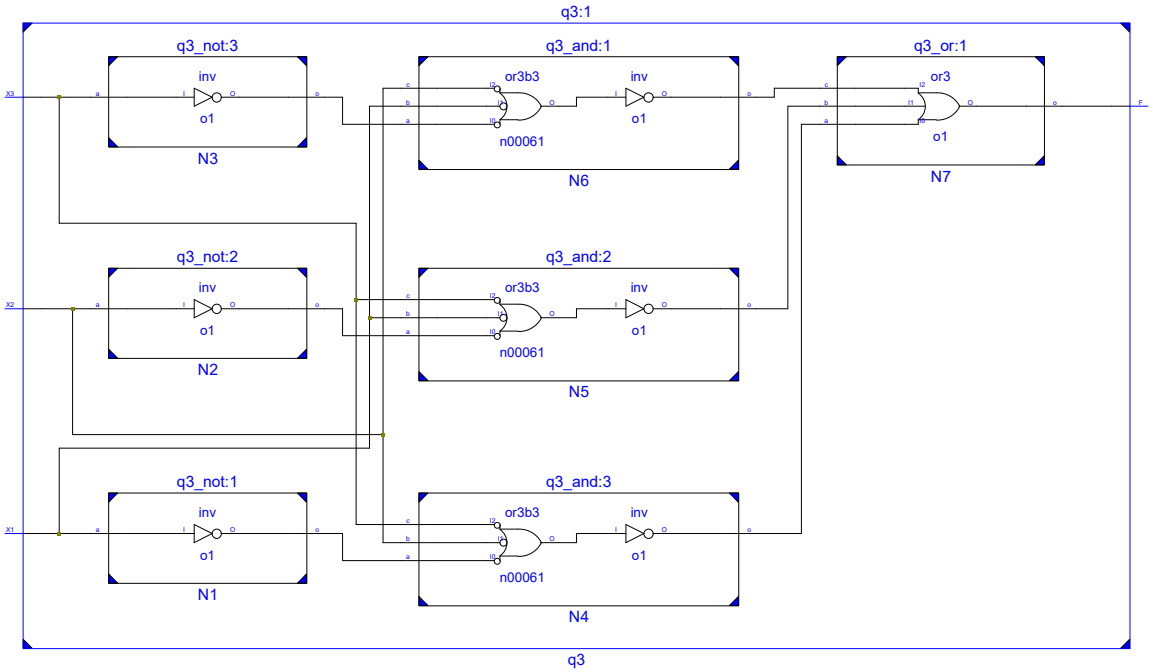


Figure 13: Observed RTL schematic for Problem 3

Problem 4: SOP

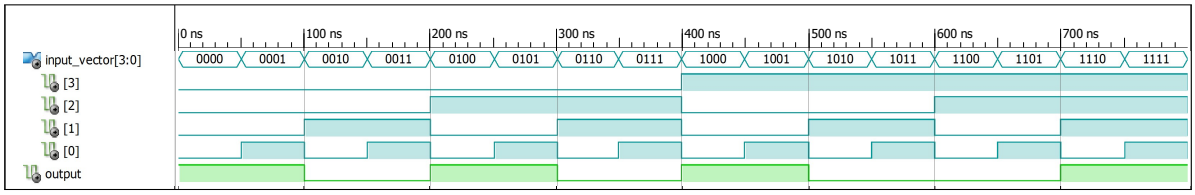


Figure 14: Observed waveform for Problem 4: SOP

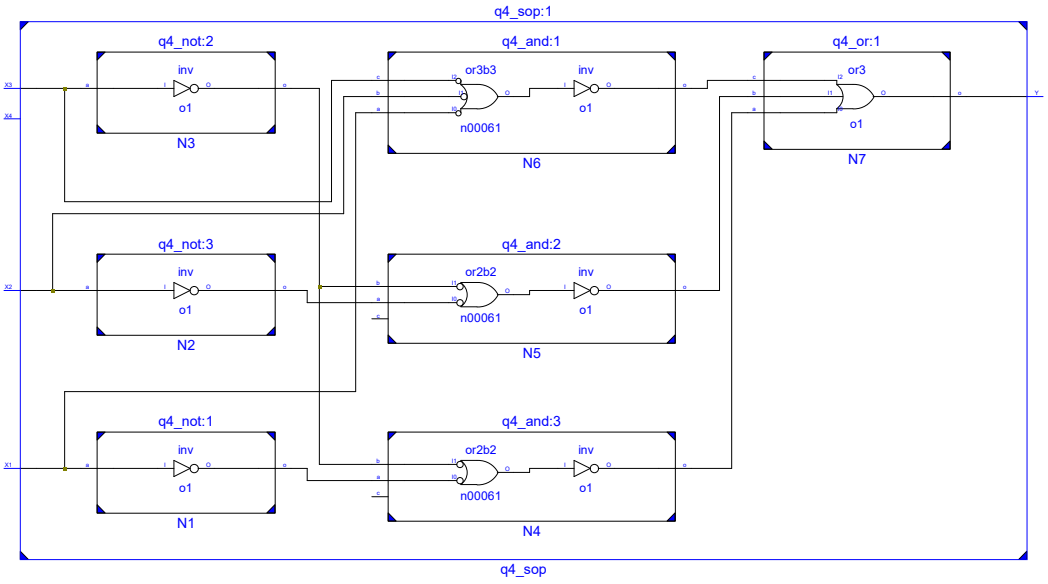


Figure 15: Observed RTL schematic for Problem 4: SOP

Problem 4: POS

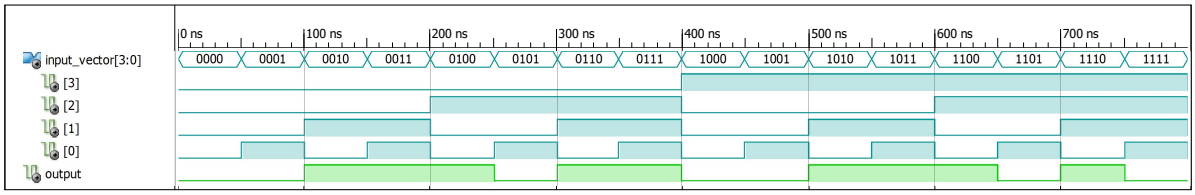


Figure 16: Observed waveform for Problem 4: POS

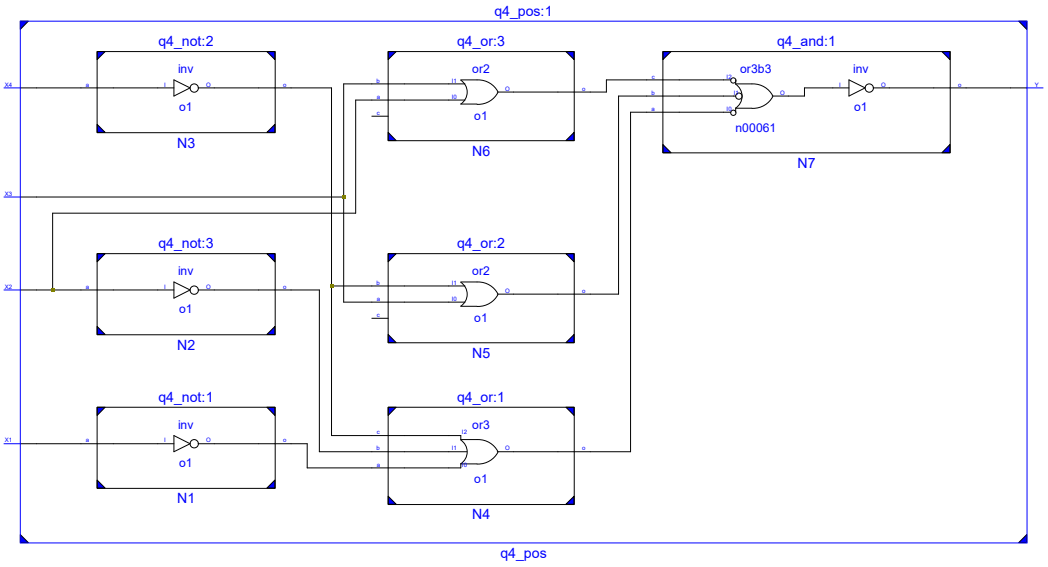


Figure 17: Observed RTL schematic for Problem 4: POS

Problem 5

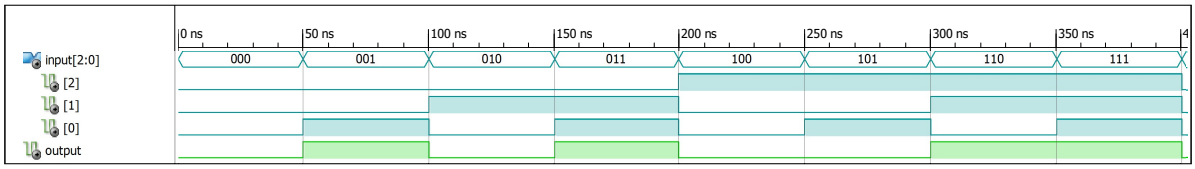


Figure 18: Observed waveform for Problem 5

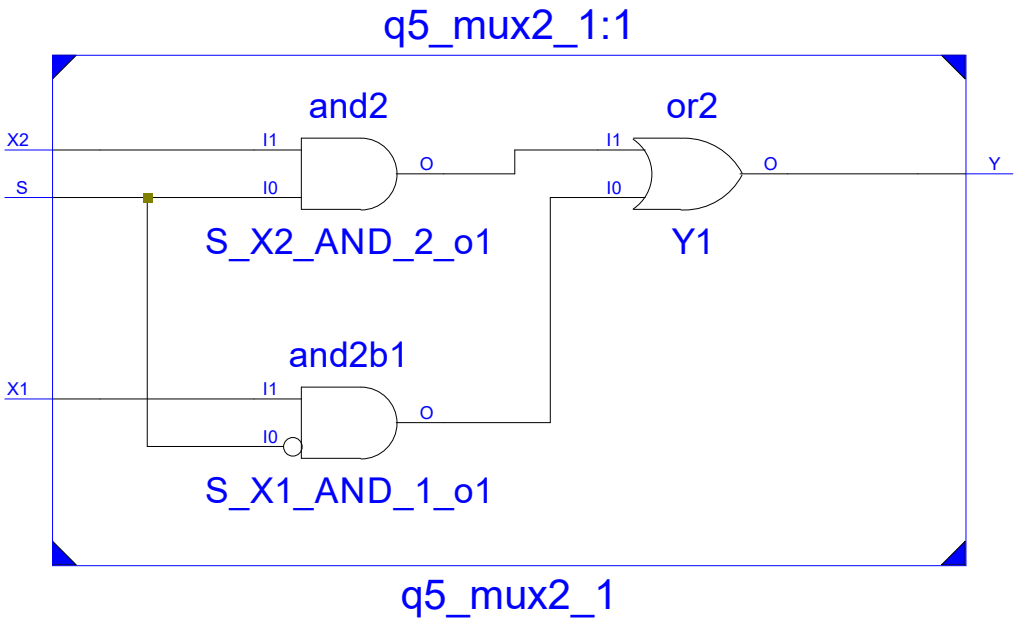
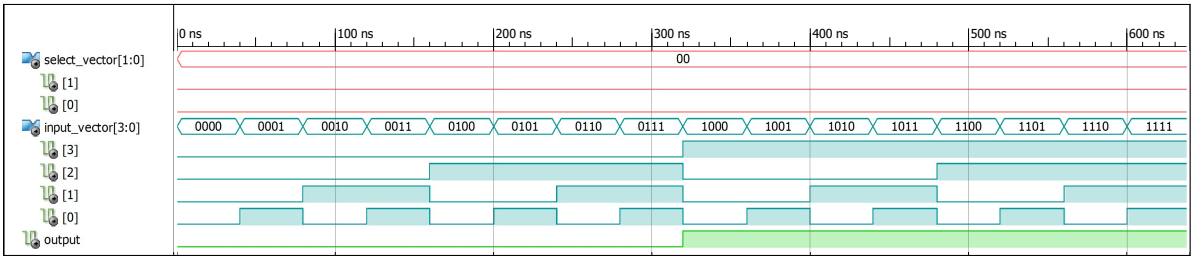


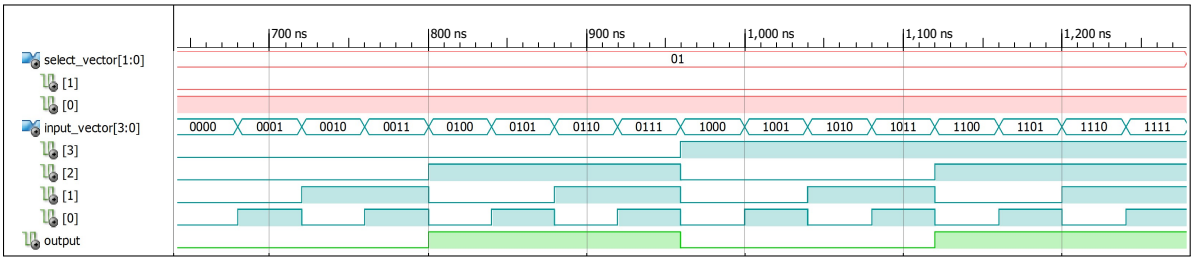
Figure 19: Observed RTL schematic for Problem 5

Problem 6

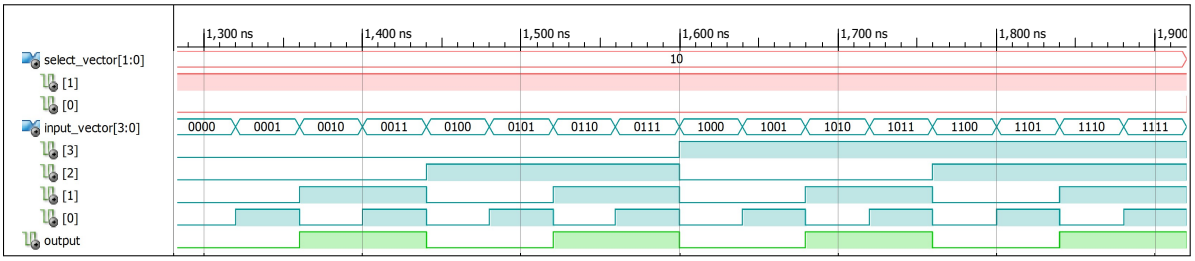


(20.1)

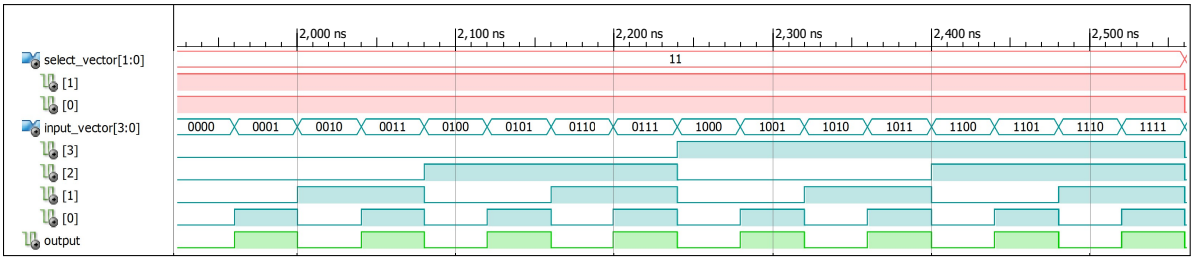
Figure 20: Observed waveform for Problem 6



(20.2)



(20.3)



(20.4)

Figure 3: Observed waveform for Problem 6 (continued)

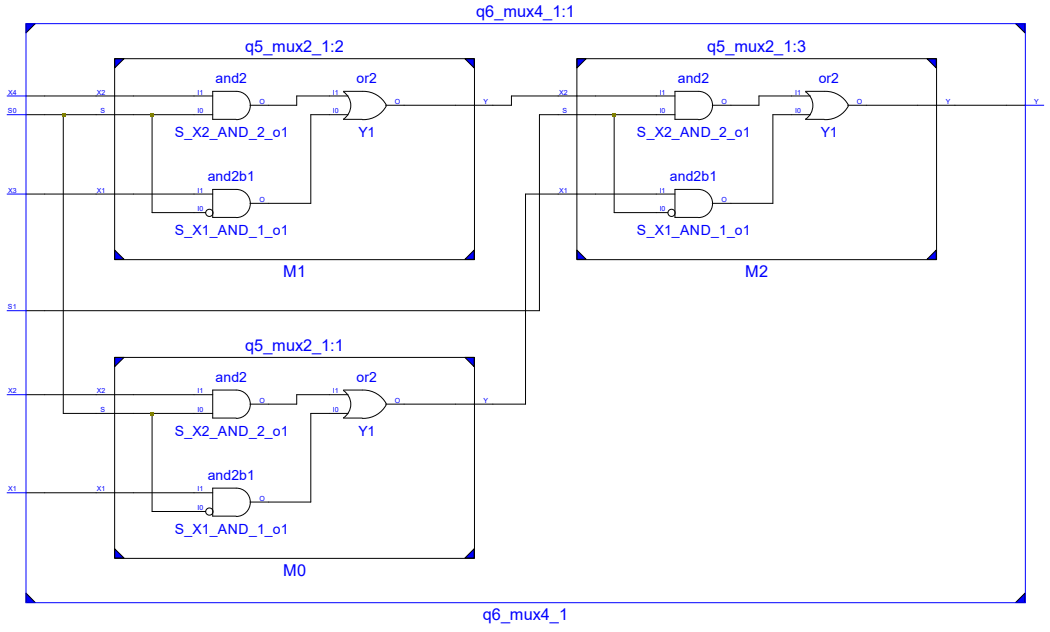
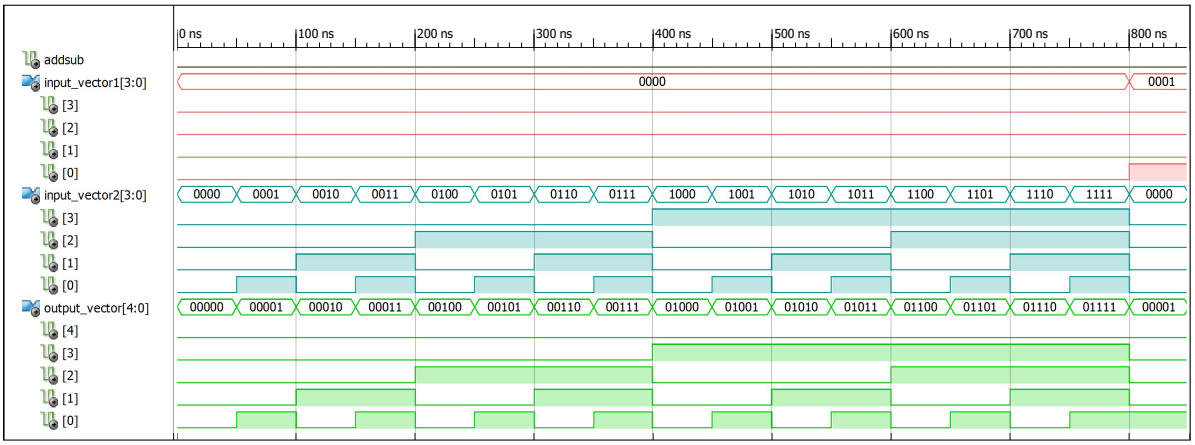
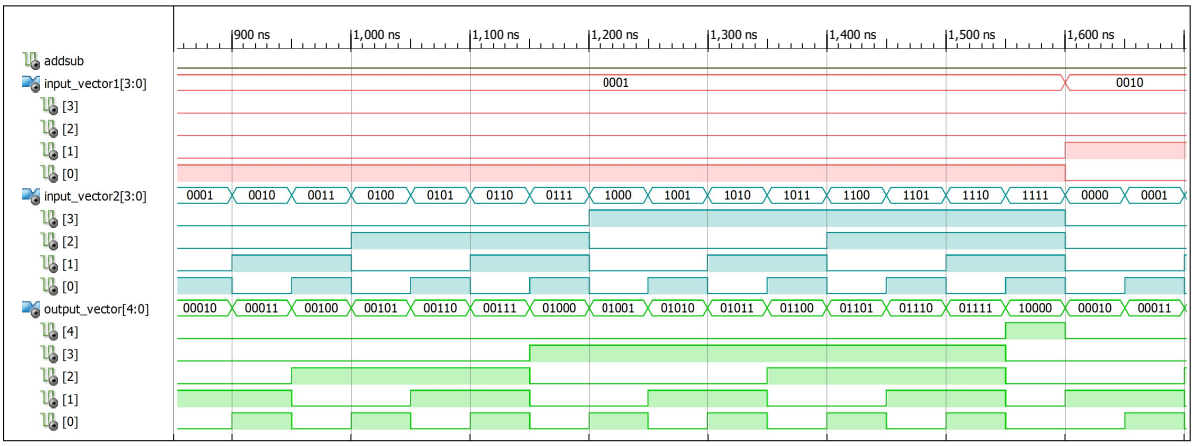


Figure 21: Observed RTL schematic for Problem 6

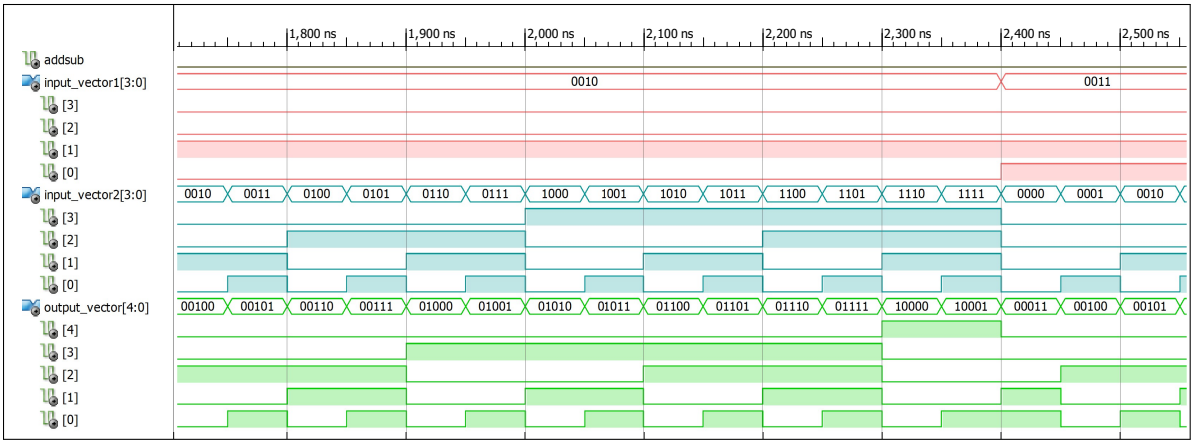
Problem 6



(22.1)

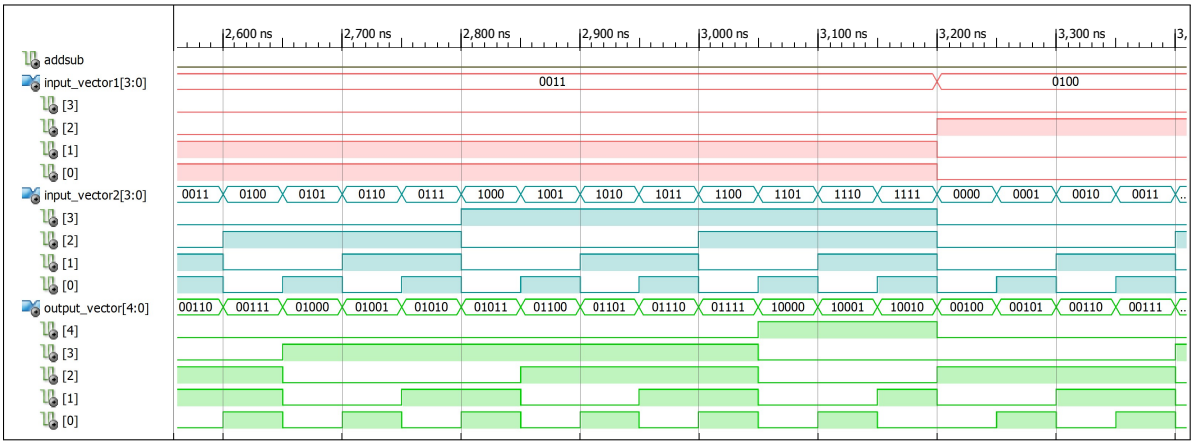


(22.2)

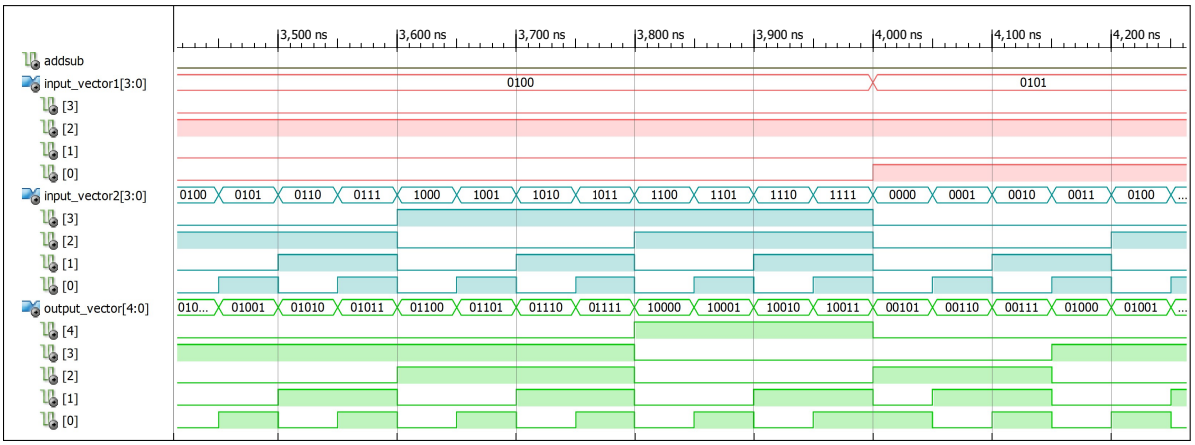


(22.3)

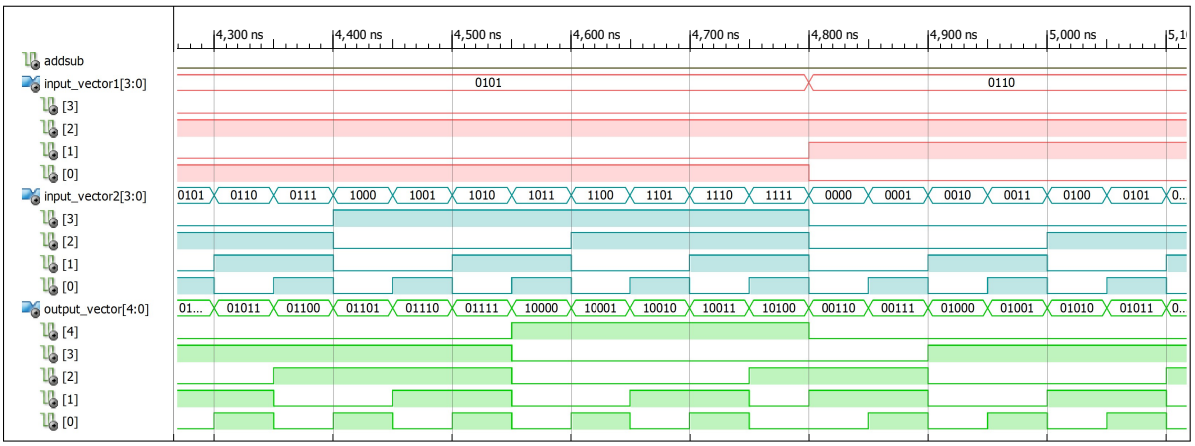
Figure 22: Observed waveform for Problem 7: 4-bit adder



(22.4)

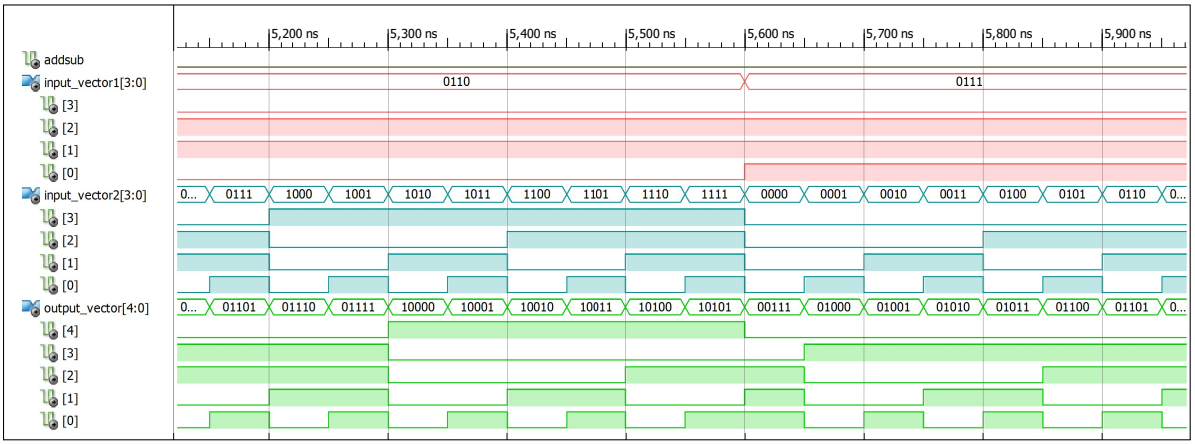


(22.5)

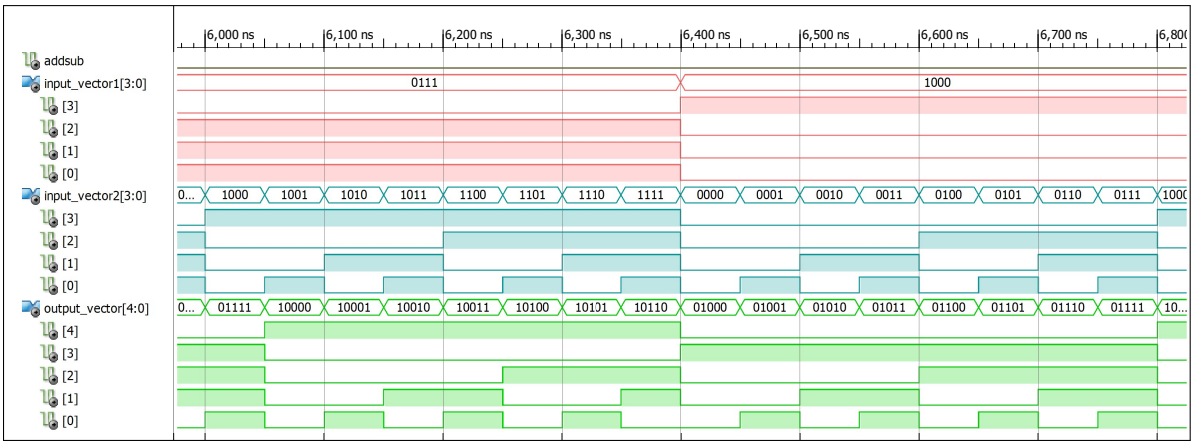


(22.6)

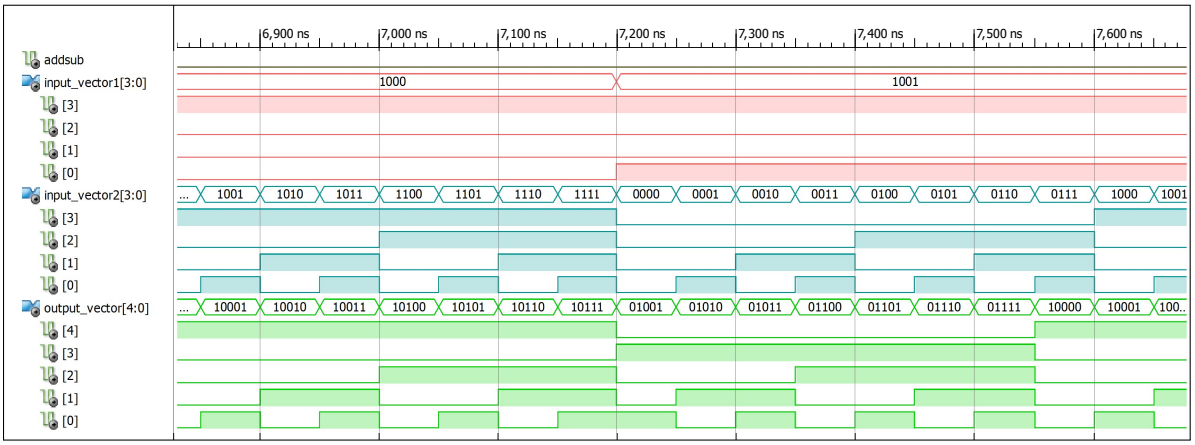
Figure 22: Observed waveform for Problem 7: 4-bit adder (continued)



(22.7)

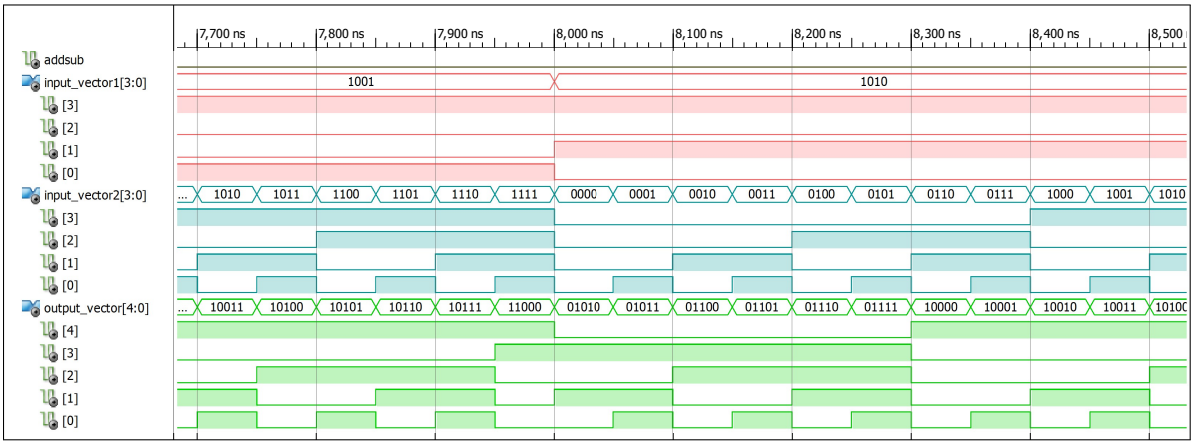


(22.8)

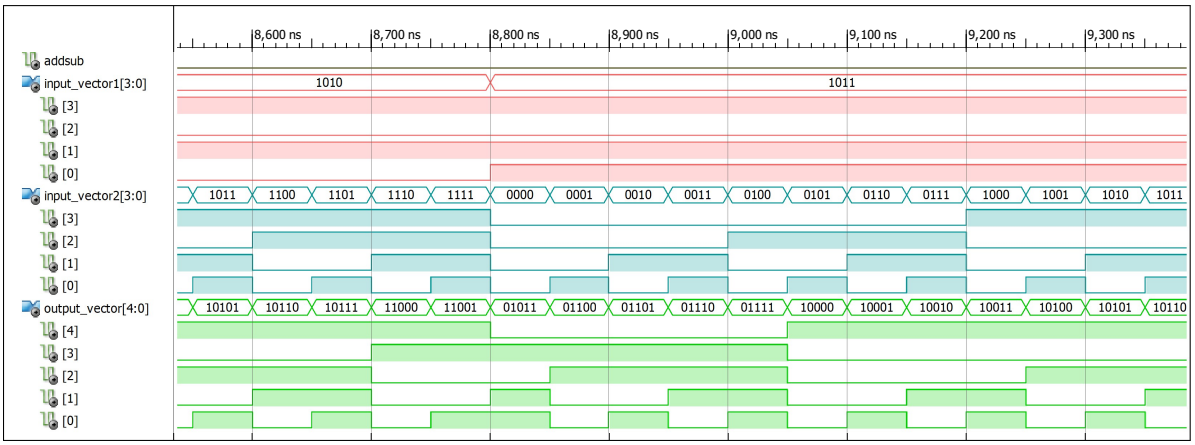


(22.9)

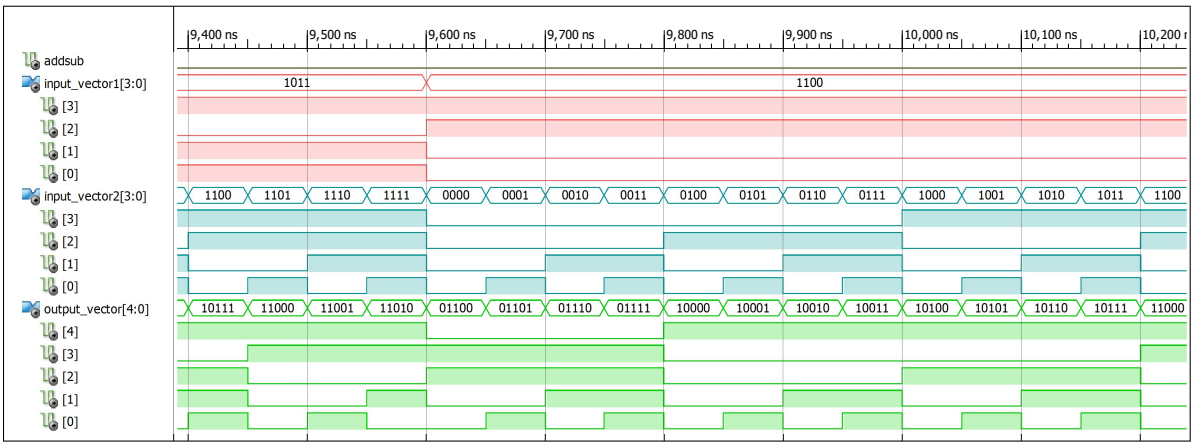
Figure 22: Observed waveform for Problem 7: 4-bit adder (continued)



(22.10)

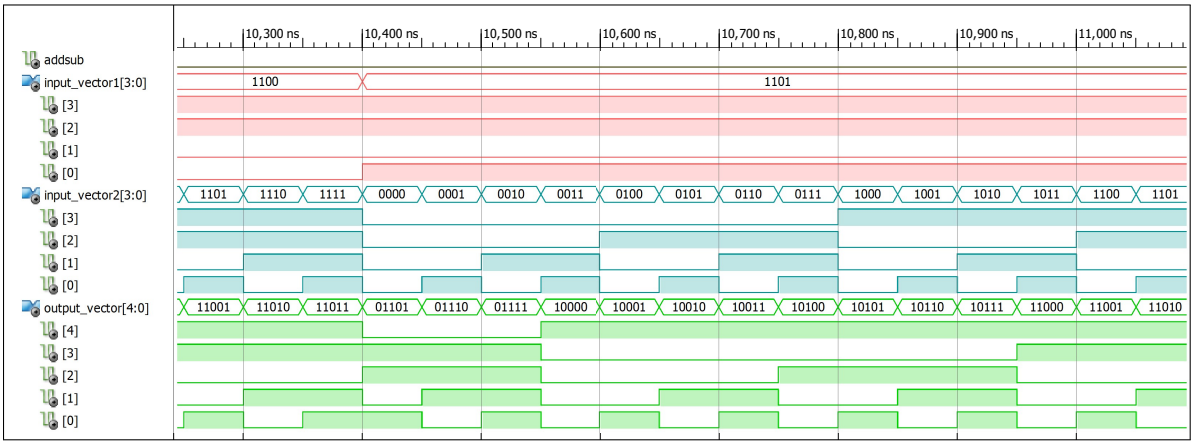


(22.11)

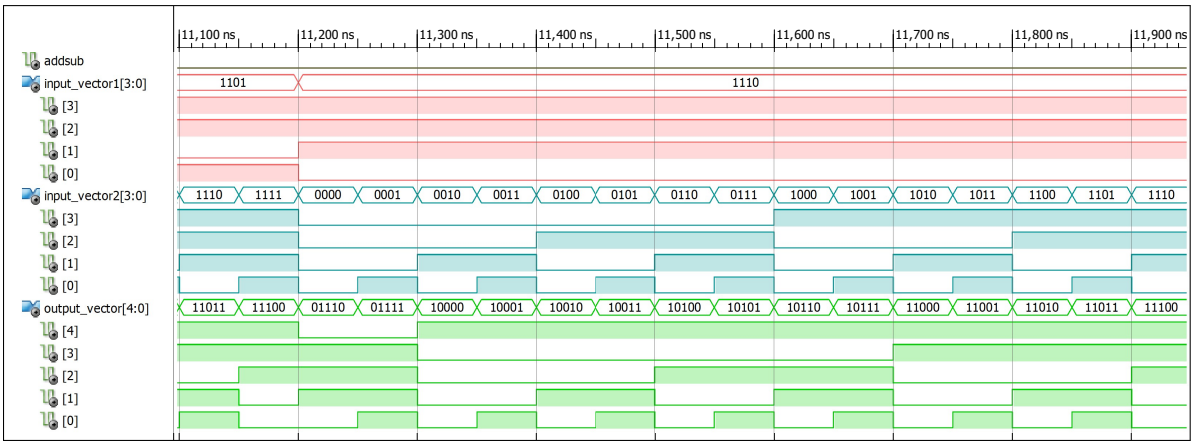


(22.12)

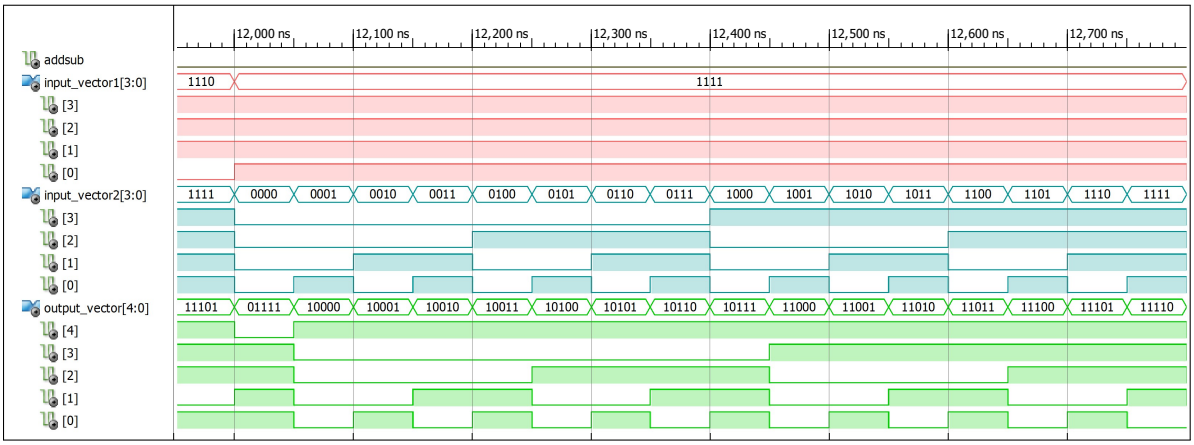
Figure 22: Observed waveform for Problem 7: 4-bit adder (continued)



(22.13)

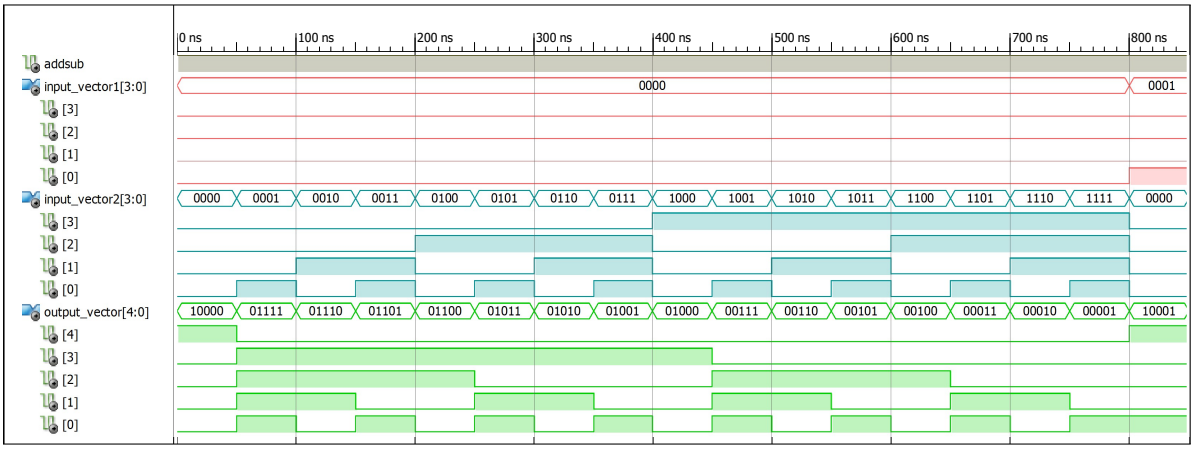


(22.14)

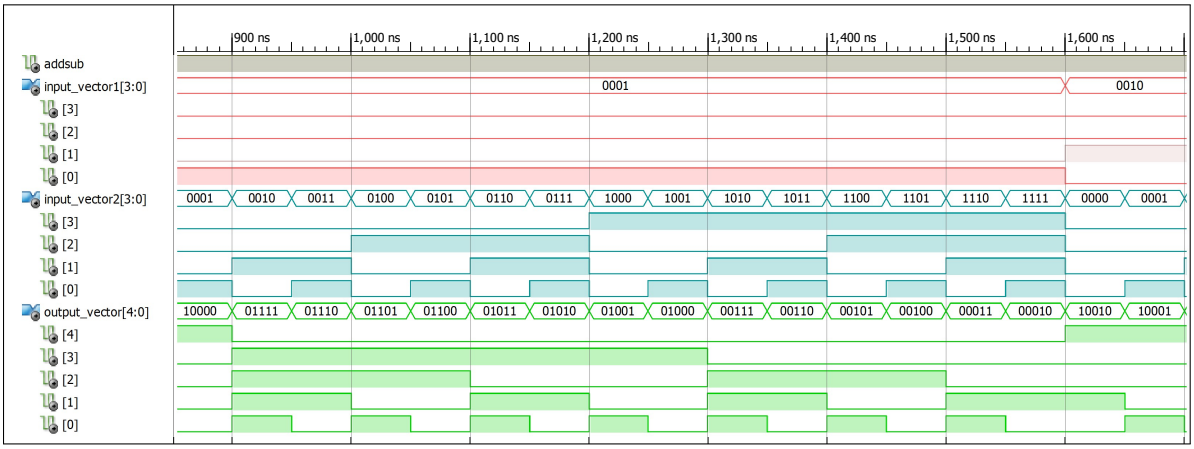


(22.15)

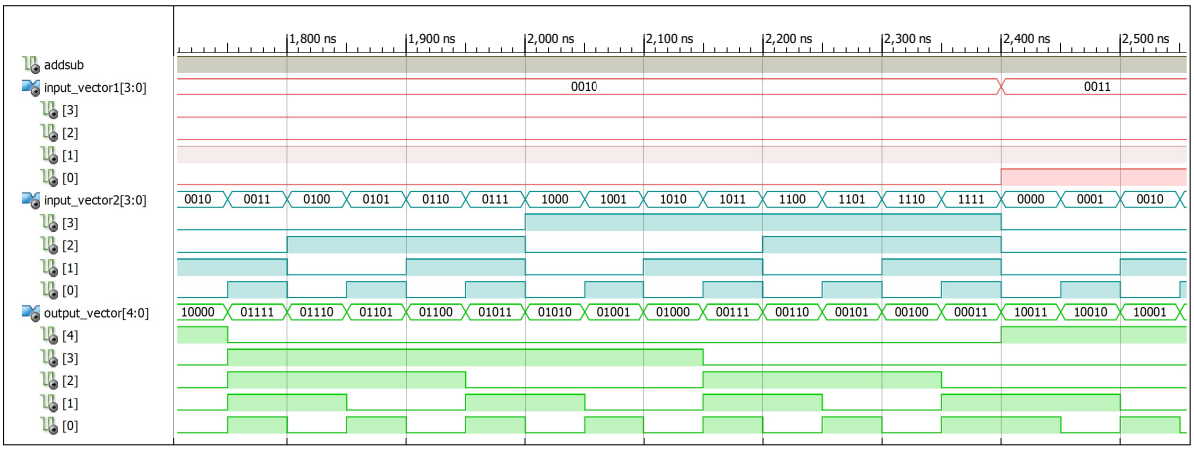
Figure 22: Observed waveform for Problem 7: 4-bit adder (continued)



(23.1)

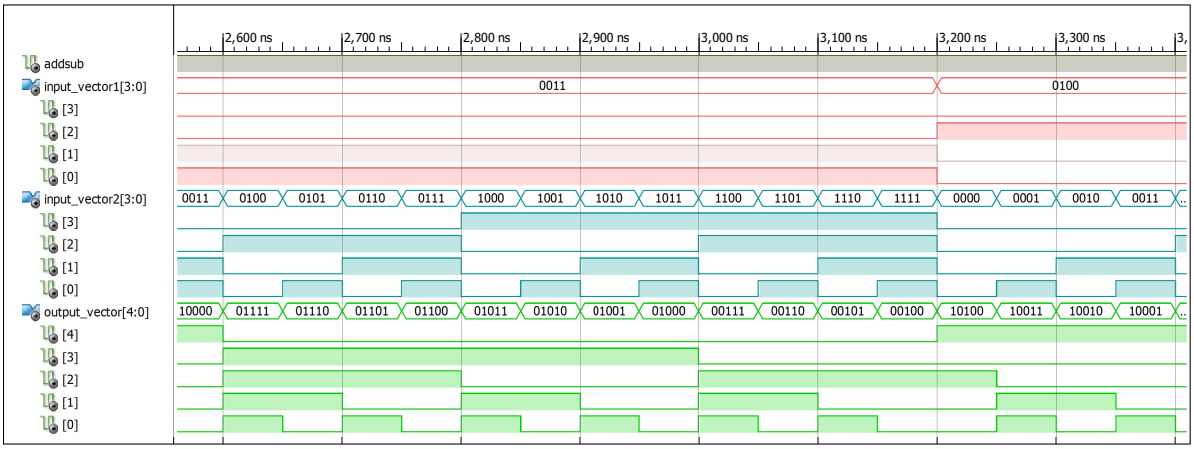


(23.2)

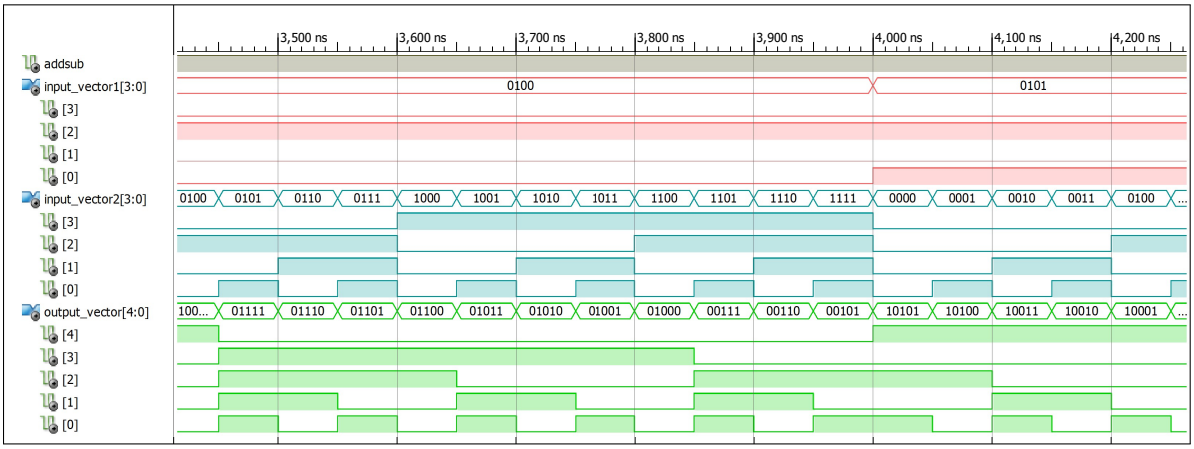


(23.3)

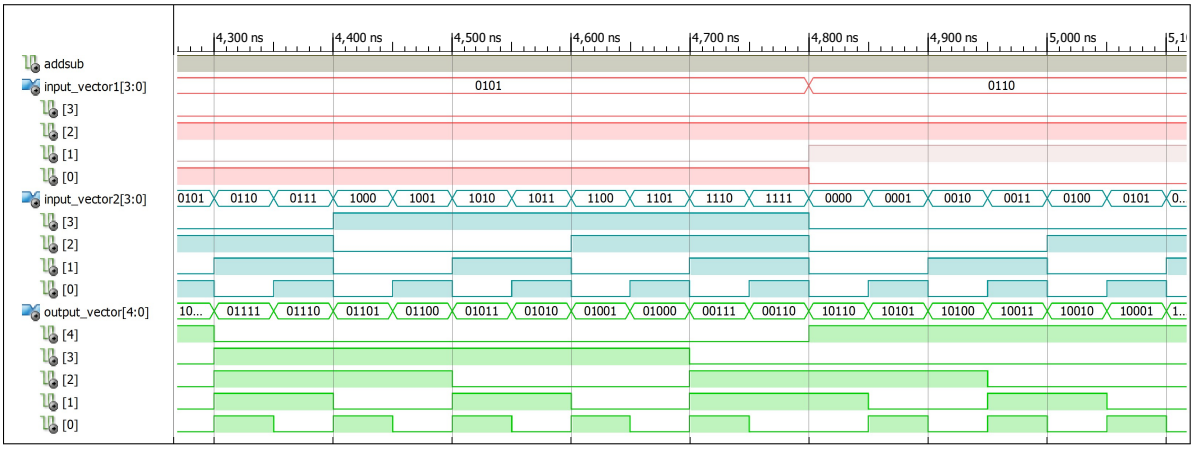
Figure 23: Observed waveform for Problem 7: 4-bit subtractor



(23.4)

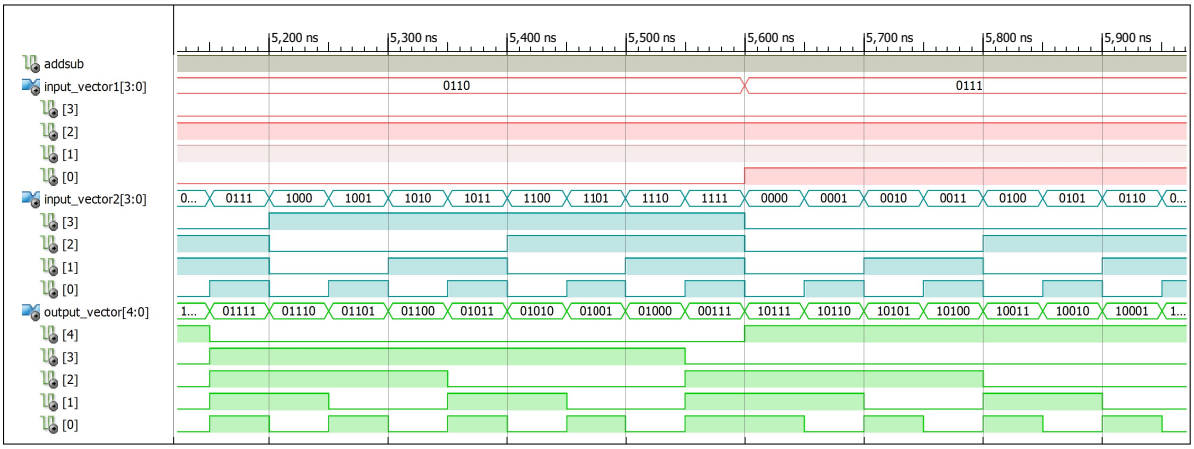


(23.5)

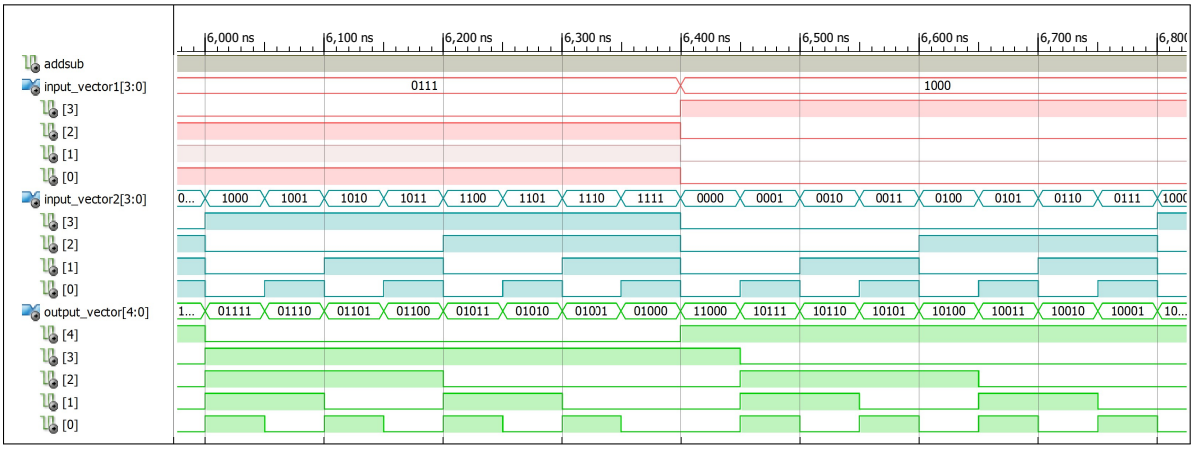


(23.6)

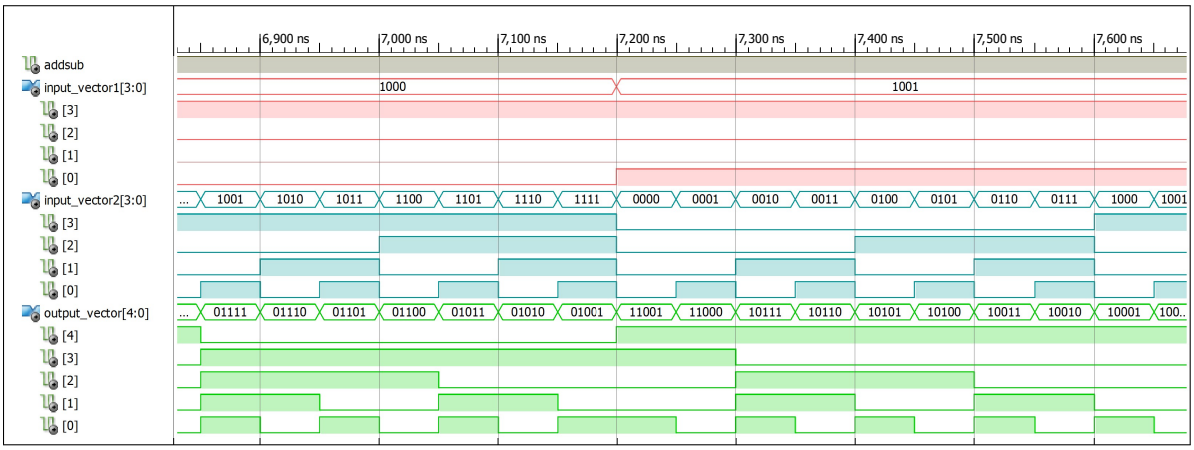
Figure 23: Observed waveform for Problem 7: 4-bit subtractor (continued)



(23.7)

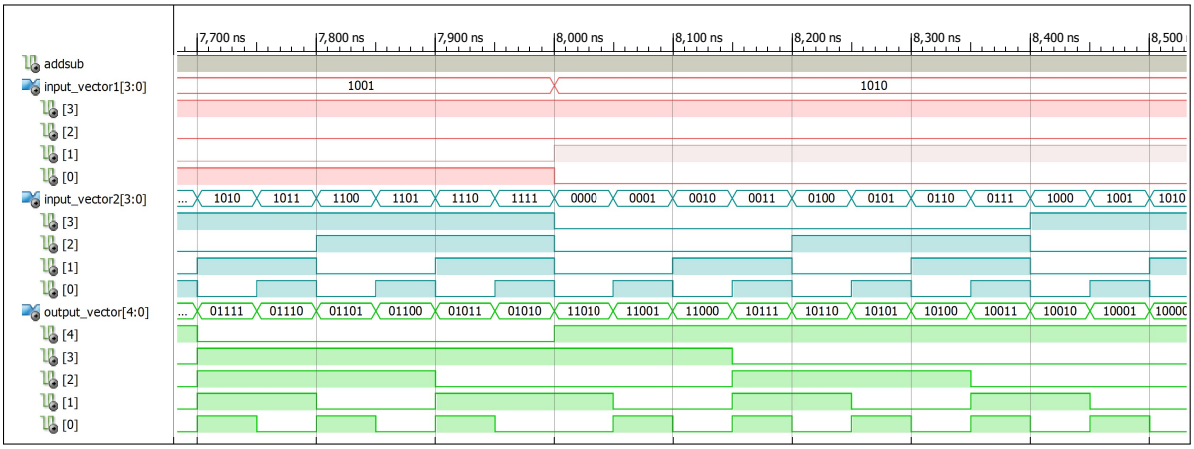


(23.8)

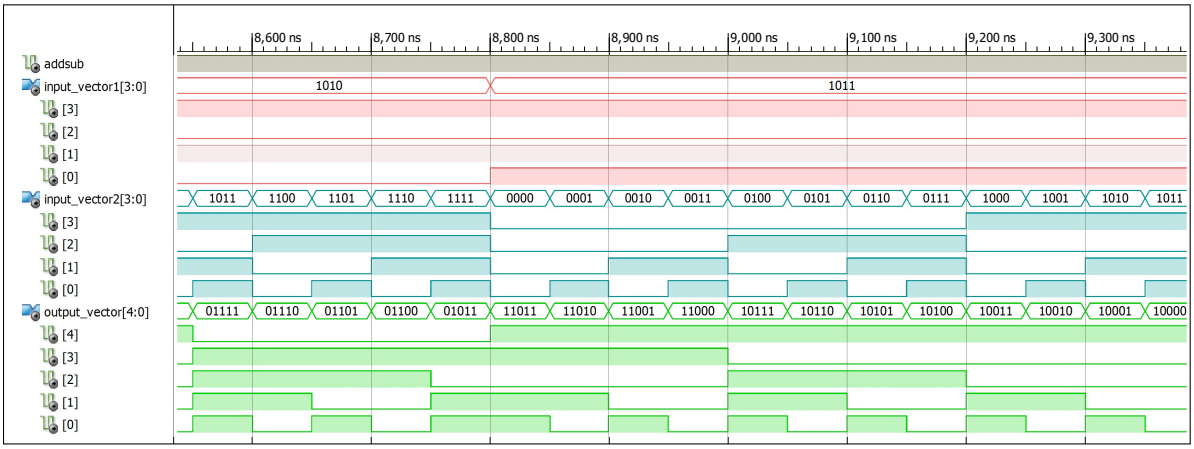


(23.9)

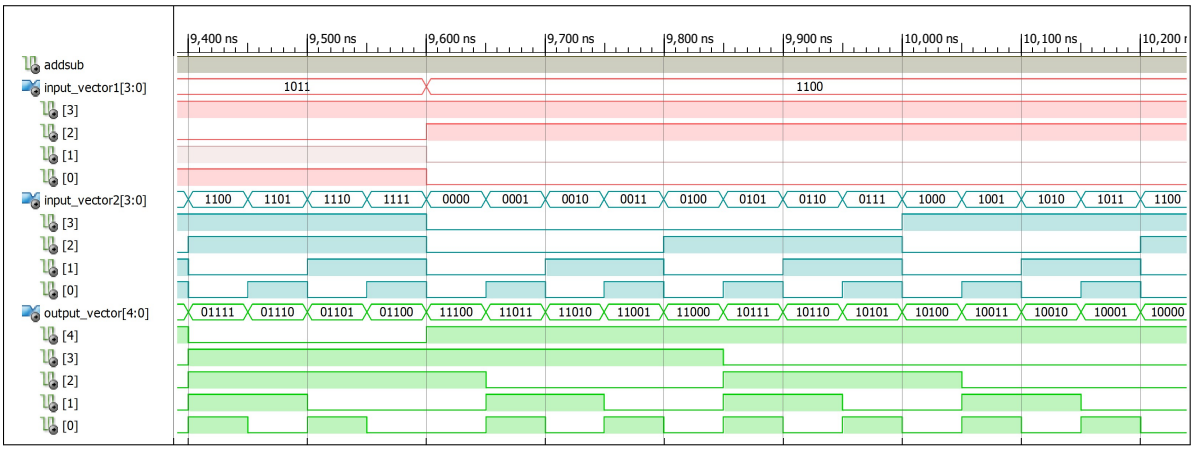
Figure 23: Observed waveform for Problem 7: 4-bit subtractor (continued)



(23.10)

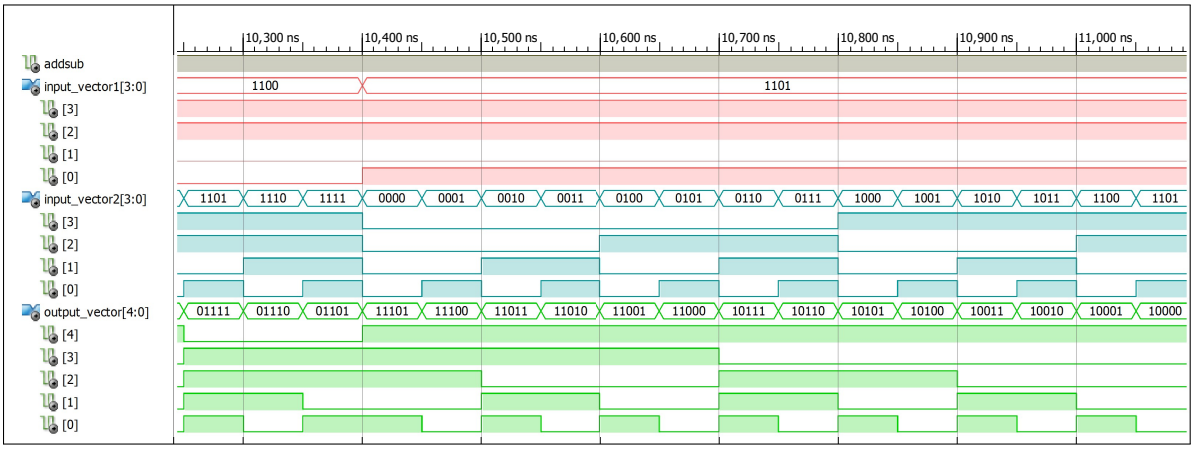


(23.11)

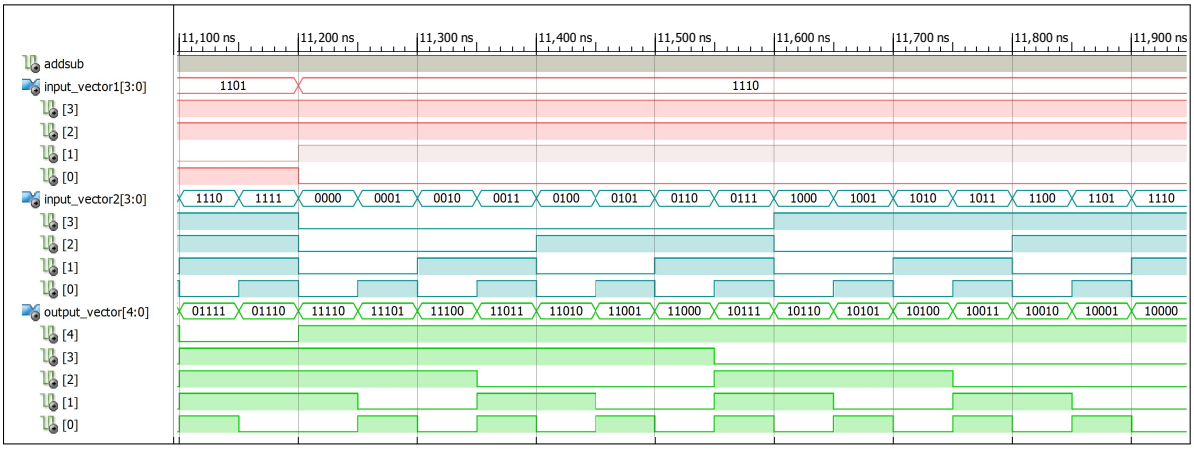


(23.12)

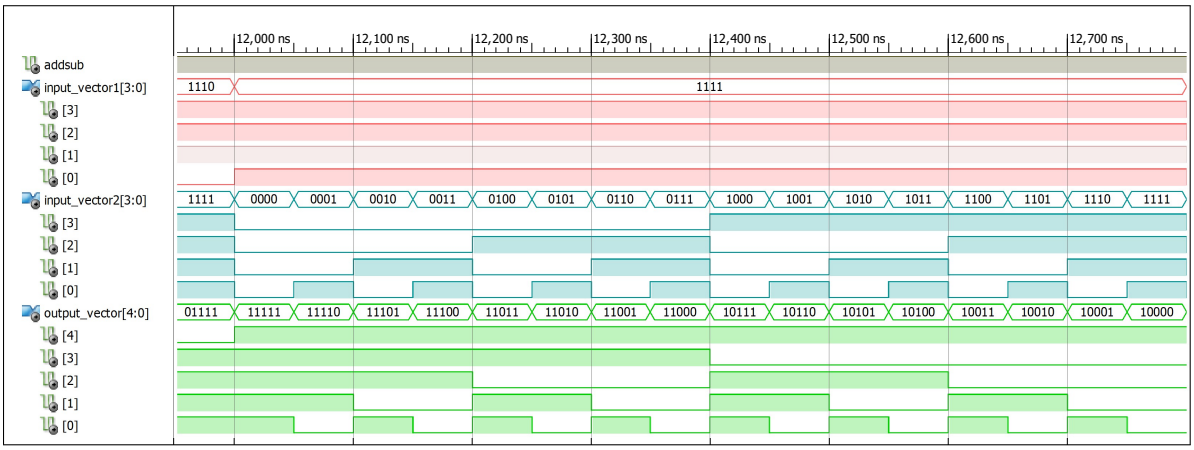
Figure 23: Observed waveform for Problem 7: 4-bit subtractor (continued)



(23.13)



(23.14)



(23.15)

Figure 23: Observed waveform for Problem 7: 4-bit subtractor (continued)

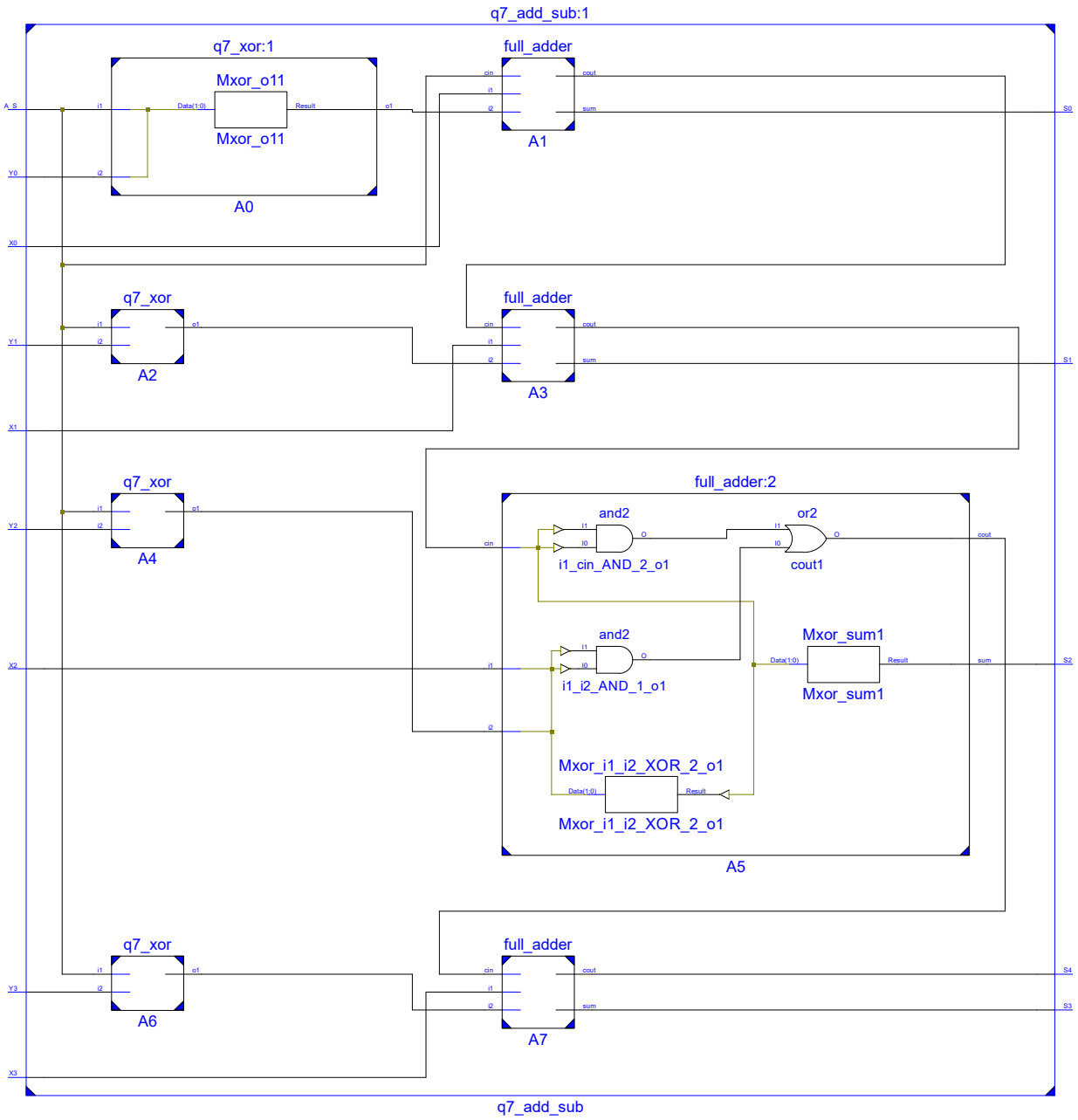


Figure 24: Observed RTL schematic for Problem 7

6 Discussion

In this lab experiment, various programs related to combinational logic circuits were written in VHDL using the Xilinx ISE Design Suite. The VHDL modules were written in dataflow, behavioral as well as structural architectural models. This allowed us to be familiar with VHDL programming concepts that are involved while designing a FPGA. The waveforms are results of ISim, a full-featured HDL simulator which was initiated by test bench programs written in VHDL. The test bench codes have been included with this report as per the instructions. RTL schematics of the combinational circuits have been included with this report since it provides a clear idea of the various modules that are used while in a complex circuit.

Additional References

[1] *VHDL Reference Manual*. en. [Online]. Available: URL: https://www.ics.uci.edu/~jmoorkan/vhdlref/Synario%20VHDL%20Manual.pdf.

[2] D. Sharma. *An Introduction to VHDL*. en. [Online]. Available: URL: https://www.ee.iitb.ac.in/~smdp/DKStutorials/vhdl-overview.pdf.

[3] *VHDL Reference Manual*. en. [Online]. Available: URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx13_3/ise_tutorial_ug695.pdf.