Lab Report 4: Arithmetic Logic Unit
EE 102
Section 02
Bilkent University

March 13th, 2024

Doğa Zeynep Tarman
22303157

Purpose of the Experiment

The main purpose of this laboratory work was to design an ALU (Arithmetic Logic Unit) which performs certain mathematical operations such as addition or subtraction. After designing, we were expected to implement everything on Basys 3.

Methodology

The first thing we had to do was to decide on six different functions other than addition and subtraction to implement using VHDL. I also had to keep in mind that we were obliged to use specified functions which were addition and subtraction. Additionally, we also had to make sure that among the other six functions that we chose, there were at least one bitwise and one shift operation function. Considering these, I have started designing my ALU. The functions that I have chosen can be seen in Table 1. I first started by writing a file named “main.vhd”. This was the

Design Specifications

My design is an arithmetic logic unit which can compute eight unique operations. In order to indicate which operation to do, I decided to include a selecting signal that does this functionality. I have specified these operations in Table 1.

Select signal	Operation	Example Representation
000	Addition	x + y
001	Subtraction	x - y
010	NAND Gate	NOT (x AND y)
011	AND Gate	x AND y
100	OR Gate	x OR y
101	NOT Operation	NOT x
110	XOR Gate	x XOR y
111	Left Shift	$x_0x_1x_2x_3 \rightarrow x_1x_2x_3x_0$

Table 1: Operations

Results

After completing the function writing and trial part, I was able to observe this on both my Basys 3 and on screen by RTL Schematics. These schematics provided me with a better understanding of what was going on inside the circuit that I have designed. I have included the pictures of these schematics below, named Figure 1-9, to represent my findings.

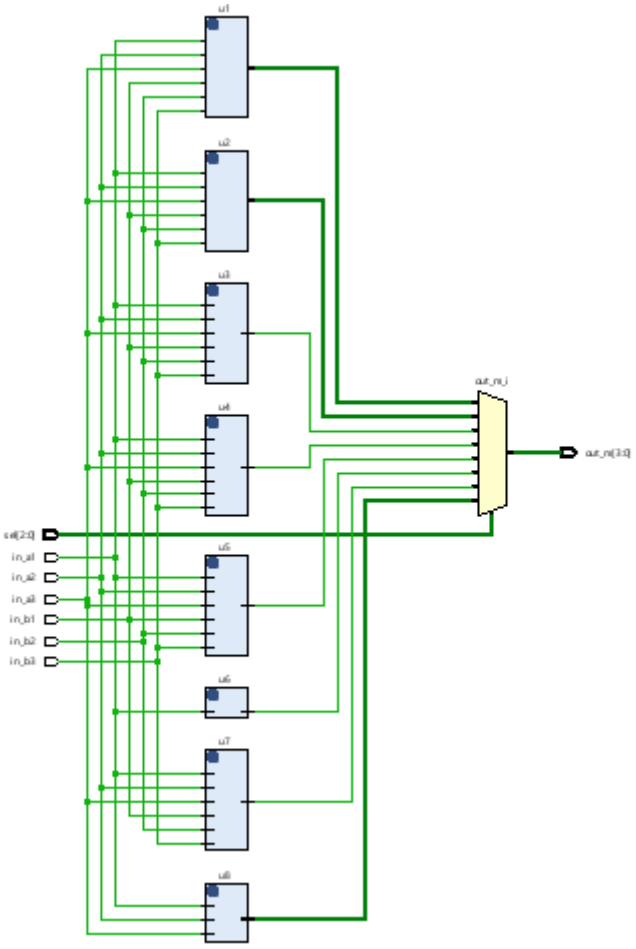


Figure 1: RTL Schematic of the Design

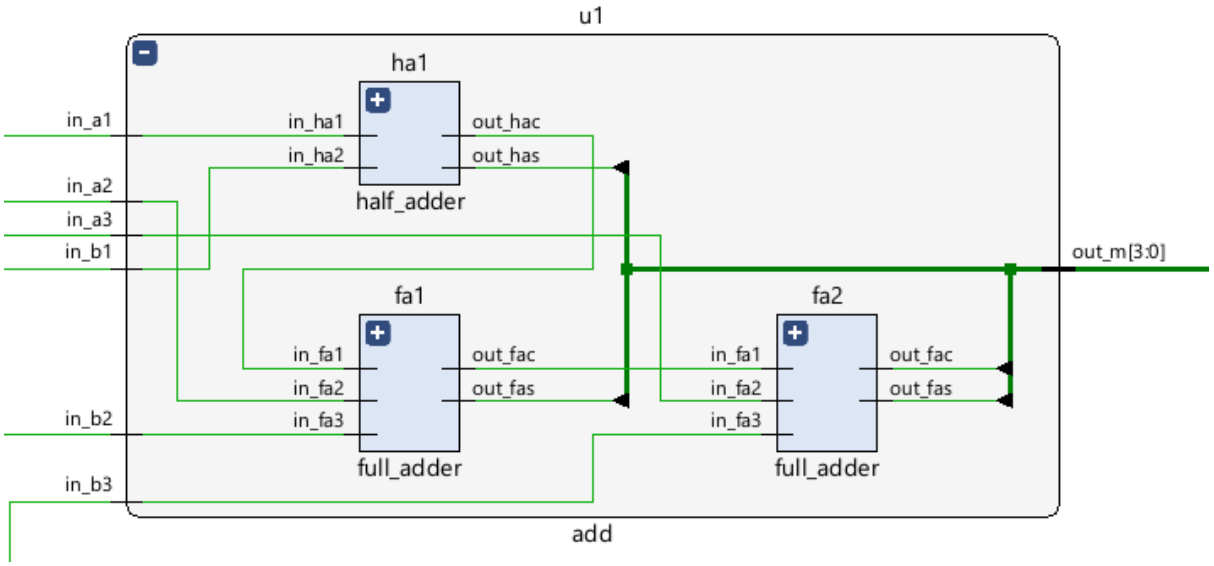


Figure 2: RTL Schematic of Addition

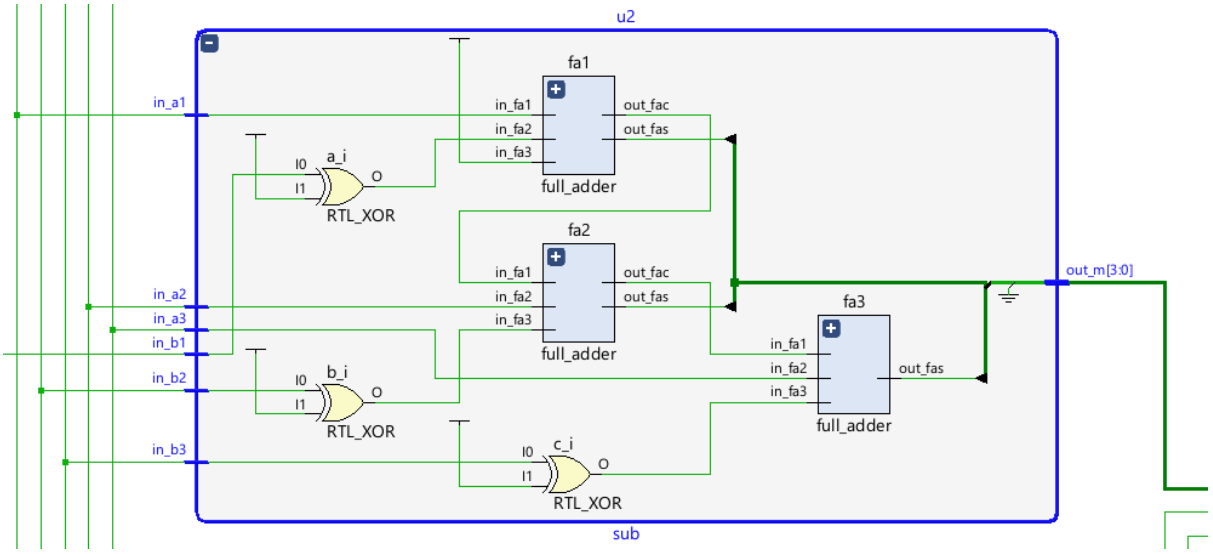


Figure 3: RTL Schematic of Subtraction

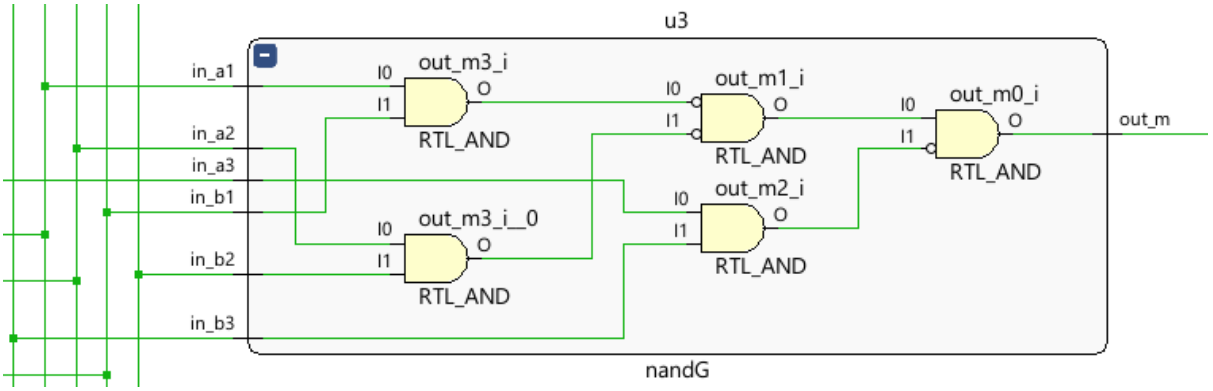


Figure 4: RTL Schematic of NAND

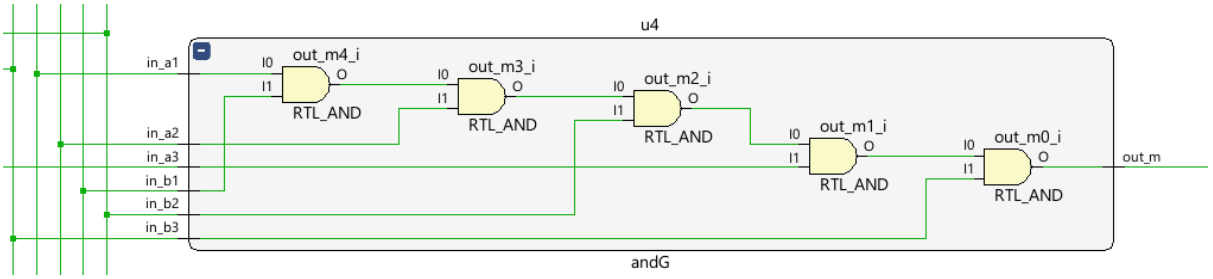


Figure 5: RTL Schematic of AND

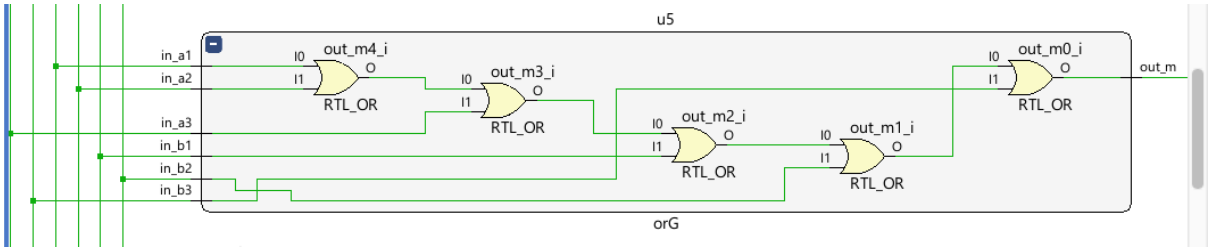


Figure 6: RTL Schematic of OR

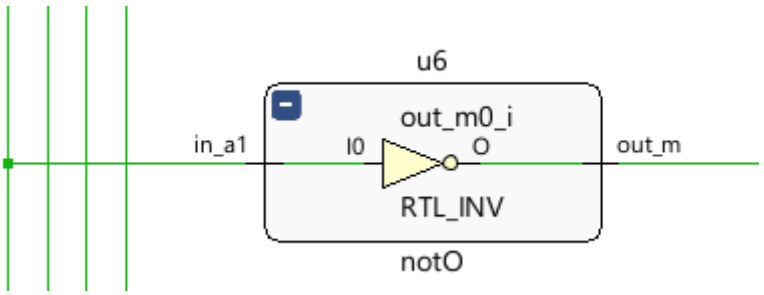


Figure 7: RTL Schematic of NOT

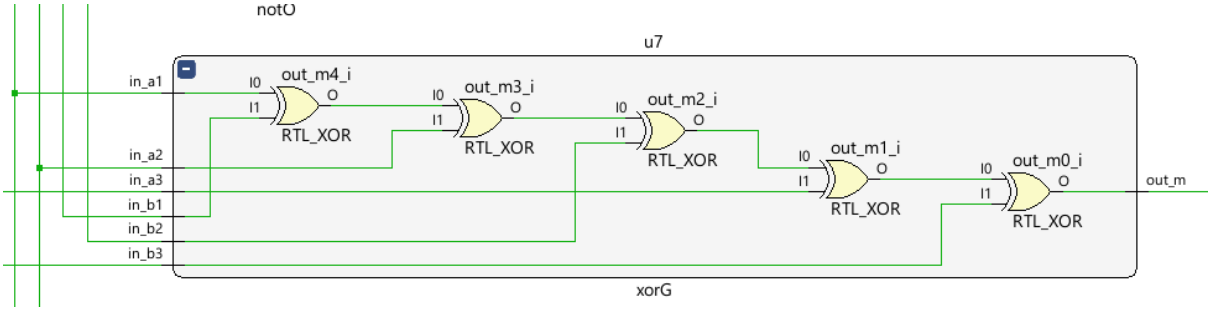


Figure 8: RTL Schematic of XOR

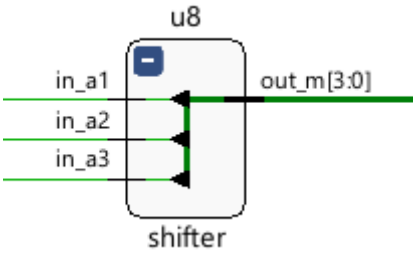


Figure 9: RTL Schematic of the Shifter

I then observed the test bench result.

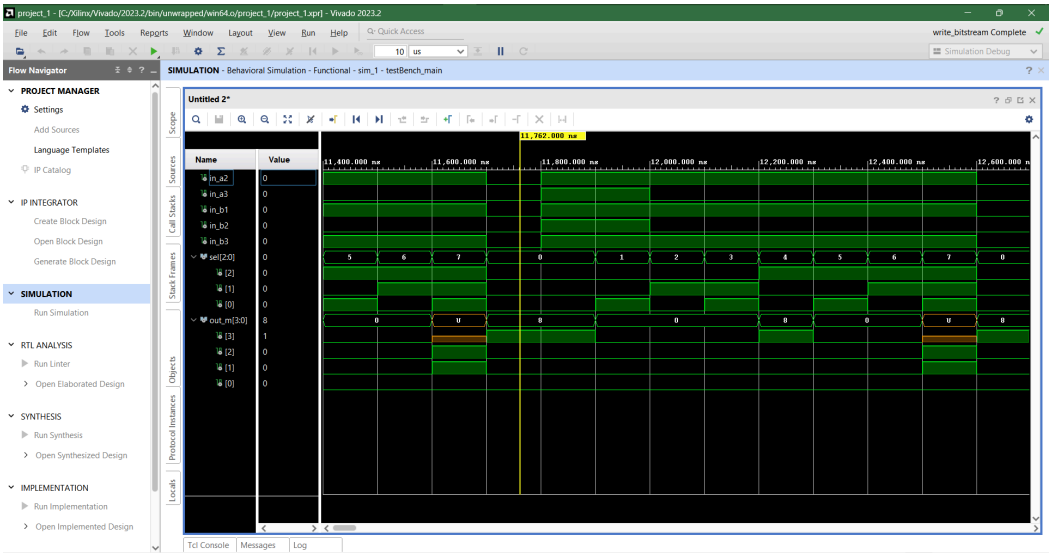
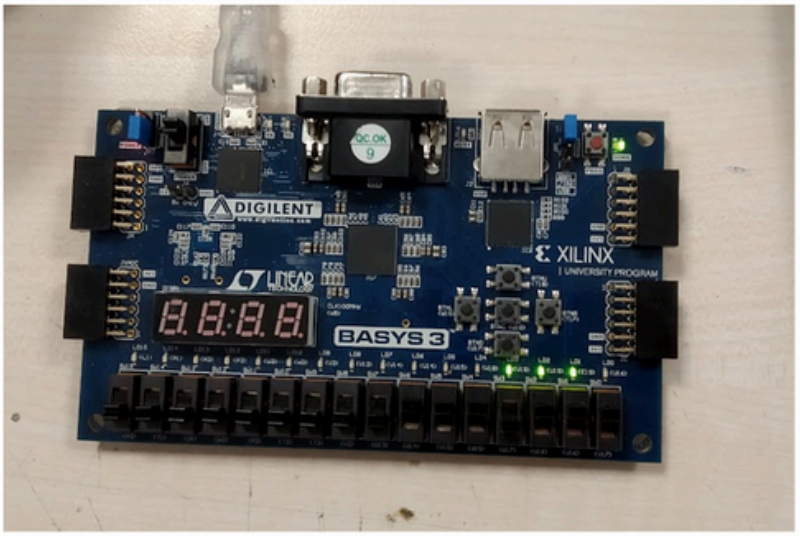


Figure 10: Test Bench Result

I then tried to see whether I could observe the same results on my Basys 3.



I was also able to observe the results on my board.

Selection: 000

in_a: 111

in_b: 111

7 + 7 = 14

Output: 1110

Figure 11: Basys 3 Doing Addition

Conclusion

In this laboratory work, we have tried to implement an arithmetic logic circuit. This circuit consisted of eight unique functions. These functions were selected by us. Using the functions, I was able to observe the outputs on my Basys 3. During the design and implementation processes, I have faced many problems. For example, I forgot to put a “;” at the end of one line, and kept searching for what the error was for quite a long time. Another problem I faced was in the constraints file. Although I was sure I wrote everything correctly, it did not work and it was quite a struggle to figure out what the problem was. Eventually, I realised that I had some indentation errors and wrote “U” instead of “V” in one of the lines. However, this simple error took so much time I almost failed to finish the task in time. Nevertheless, it was an important experience and it has taught me to be careful about errors like these in the future. I believe that in the future, I will not make such errors and spend significant amounts of time on them. Overall, this laboratory work was an enjoyable one which challenged us to try something out of our comfort zone and thus it has taught me a lot.

Appendix

Top Module (main.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity main is
	Port (
	in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in STD_LOGIC;
	sel: in std_logic_vector(2 downto 0);
	out_m: out std_logic_vector(3 downto 0)
	);
end main;

architecture Behavioral of main is
	component add
	Port (
	in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
	out_m: out std_logic_vector(3 downto 0)
	);
	end component;

	component sub
	Port (
	in_a1,in_a2, in_a3, in_b1, in_b2, in_b3 : in STD_LOGIC;
	out_m : out std_logic_vector(3 downto 0)
	);

	end component;

	component nandG
	Port (
	in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
	out_m: out std_logic
	);
	end component;

	component andG
	Port (
	in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
	out_m: out std_logic
	);
	end component;
```

```
        component orG
        Port (
        in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
        out_m: out std_logic
        );
        end component;

        component notO
        Port (
        in_a1 : in std_logic;
        out_m: out std_logic
        );
        end component;

        component xorG
        Port (
        in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
        out_m: out std_logic
        );
        end component;

        component shifter
        Port (
        in_a1, in_a2, in_a3 : in std_logic;
        out_m: out std_logic_vector(3 downto 0)
        );
        end component;
        signal out1, out2, out3, out4, out5, out6, out7, out8: std_logic_vector(3 downto
0);
begin
        u1: add port map (
        in_a1 => in_a1, in_a2 => in_a2, in_a3 => in_a3,
        in_b1 => in_b1, in_b2 => in_b2, in_b3 => in_b3,
        out_m => out1
        );

        u2: sub port map (
        in_a1 => in_a1, in_a2 => in_a2, in_a3 => in_a3,
        in_b1 => in_b1, in_b2 => in_b2, in_b3 => in_b3,
        out_m => out2
        );

        u3: nandG port map (
        in_a1 => in_a1, in_a2 => in_a2, in_a3 => in_a3,
```

```
in_b1 => in_b1, in_b2 => in_b2, in_b3 => in_b3,
out_m => out3(0)
);

u4: andG port map (
in_a1 => in_a1, in_a2 => in_a2, in_a3 => in_a3,
in_b1 => in_b1, in_b2 => in_b2, in_b3 => in_b3,
out_m => out4(0)
);

u5: orG port map (
in_a1 => in_a1, in_a2 => in_a2, in_a3 => in_a3,
in_b1 => in_b1, in_b2 => in_b2, in_b3 => in_b3,
out_m => out5(0)
);

u6: notO port map (
in_a1 => in_a1,
out_m => out6(0));

u7: xorG port map (
in_a1 => in_a1, in_a2 => in_a2, in_a3 => in_a3,
in_b1 => in_b1, in_b2 => in_b2, in_b3 => in_b3,
out_m => out7(0)
);

u8: shifter port map (
in_a1 => in_a1, in_a2 => in_a2, in_a3 => in_a3,
out_m => out8
);
process(sel)
begin
case sel is
when "000" =>
        out_m <= out1;
when "001" =>
        out_m <= out2;
when "010" =>
        out_m <= out3;
when "011" =>
        out_m <= out4;
when "100" =>
        out_m <= out5;
when "101" =>
```

```
            out_m <= out6;
        when "110" =>
            out_m <= out7;
        when "111" =>
            out_m <= out8;
        when others =>
            out_m <= (others => '0');
        end case;
        end process;
end Behavioral;
```

Addition (add.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add is
        Port (
        in_a1, in_a2, in_a3, in_b1, in_b2, in_b3 : in STD_LOGIC;
        out_m: out STD_LOGIC_VECTOR(3 downto 0)
        );
end add;

architecture Behavioral of add is

        component half_adder
        Port ( in_ha1 : in STD_LOGIC;
            in_ha2 : in STD_LOGIC;
            out_has : out STD_LOGIC;
            out_hac : out STD_LOGIC
        );
        end component;

        component full_adder
        Port ( in_fa1 : in STD_LOGIC;
            in_fa2 : in STD_LOGIC;
            in_fa3 : in STD_LOGIC;
            out_fas : out STD_LOGIC;
            out_fac : out STD_LOGIC
        );
        end component;

        signal hac, fac1 : std_logic;
```

```
begin

        ha1: half_adder
        port map (
        in_ha1 => in_a1, in_ha2 => in_b1,
        out_has => out_m(0), out_hac => hac
        );

        fa1: full_adder
        port map (
        in_fa1 => hac, in_fa2 => in_a2, in_fa3 => in_b2,
        out_fas => out_m(1), out_fac => fac1
        );

        fa2: full_adder
        port map (
        in_fa1 => fac1, in_fa2 => in_a3, in_fa3 => in_b3,
        out_fas => out_m(2), out_fac => out_m(3)
        );

end Behavioral;
```

Subtraction (sub.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sub is
        Port ( in_a1,in_a2, in_a3, in_b1, in_b2, in_b3 : in STD_LOGIC;
        out_m: out STD_LOGIC_vector(3 downto 0));

end sub;

architecture Behavioral of sub is

component full_adder
        Port ( in_fa1 : in STD_LOGIC;
        in_fa2 : in STD_LOGIC;
        in_fa3 : in STD_LOGIC;
        out_fas : out STD_LOGIC;
        out_fac : out STD_LOGIC);
end component;

signal abc,fac1,fac2, a, b, c: std_logic;
```

```
begin
	a <=  in_b1 xor '1';
	fa1:full_adder
	port map(in_fa1 => in_a1, in_fa2 => (a), in_fa3 => '1',
	out_fas => out_m(0), out_fac => fac1);
	b <= in_b2 xor '1';
	fa2:full_adder
	port map(in_fa1 => fac1, in_fa2 => in_a2, in_fa3 => (b),
	out_fas => out_m(1), out_fac => fac2);
	c <= in_b3 xor '1';
	fa3:full_adder
	port map(in_fa1 => fac2, in_fa2 => in_a3, in_fa3 => (c),
	out_fas => out_m(2), out_fac => abc);
	out_m(3) <= '0';
end Behavioral;
```

NAND Gate (nandG.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity nandG is
	Port (
	in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
	out_m: out std_logic
	);
end nandG;

architecture Behavioral of nandG is
begin
	process(in_a1, in_a2, in_a3, in_b1, in_b2, in_b3)
	begin
	out_m <= not (in_a1 and in_b1) and not (in_a2 and in_b2) and not (in_a3 and
in_b3);

	end process;
end Behavioral;
```

AND Gate (andG.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity andG is
	Port (
	in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
	out_m: out std_logic
	);
end andG;

architecture Behavioral of andG is
begin
	process(in_a1, in_a2, in_a3, in_b1, in_b2, in_b3)
	begin
	out_m <= in_a1 and in_b1 and in_a2 and in_b2 and in_a3 and in_b3;
	end process;
end Behavioral;
```

OR Gate (orG.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity orG is
	Port (
	in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
	out_m: out std_logic
	);
end orG;

architecture Behavioral of orG is
begin
	process(in_a1, in_a2, in_a3, in_b1, in_b2, in_b3)
	begin
	out_m <= in_a1 or in_a2 or in_a3 or in_b1 or in_b2 or in_b3;
	end process;
end Behavioral;
```

NOT Operation (notO.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity notO is
	Port (
	in_a1: in std_logic;
	out_m: out std_logic
```

```
        );
end notO;

architecture Behavioral of notO is
begin
        process(in_a1)
        begin
        out_m <= (not in_a1);
        end process;
end Behavioral;
```

XOR Gate (xorG.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity xorG is
        Port (
        in_a1, in_a2, in_a3, in_b1, in_b2, in_b3: in std_logic;
        out_m: out std_logic
        );
end xorG;

architecture Behavioral of xorG is
begin
        process(in_a1, in_a2, in_a3, in_b1, in_b2, in_b3)
        begin
        out_m <= (in_a1 xor in_b1 xor in_a2 xor in_b2 xor in_a3 xor in_b3);
        end process;
end Behavioral;
```

Left Shifter (shifter.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity shifter is
Port (in_a1, in_a2, in_a3: IN STD_LOGIC;
                out_m: OUT STD_LOGIC_VECTOR(3 downto 0));
end shifter;

architecture Behavioral of shifter is
```

```
begin

out_m(0) <=  in_a3;
out_m(1) <= in_a1;
out_m(2) <= in_a2;

end Behavioral;
```