Bilkent University Electrical and Electronics Department

EE102-03 Lab 2 Report:

Arithmetic Logic Unit (ALU)

Cankut Bora Tuncer – 22001770

Purpose:

The purpose of this lab was to design an arithmetic logic unit (ALU) capable of performing at least one bitwise and one shift operation via VHDL. The mathematical functions are: summation, multiplication, subtraction, comparison, barrel-shifter, XNOR, AND, OR gates. The VHDL code is designed in a modular fashion. At the end of the lab, the code would be implemented on the Basys3.

Methodology:

The initial task was to decide which eight operations to choose from and how many bits to functionalize. It is decided to use two 3-bit unsigned numbers, in_a and in_b. After deciding on the functionalities, another essential task was implementing the switching operation to toggle between 8 processes. RTL schematic is used to visualize disconnections between pins and test if the pins are driven appropriately. The test bench is initialized to verify that all of the eight operations are working accordingly, and the waveform is recorded.

The constraint file is created, and the bitstream is generated. For the final step, the functions are simulated via. switches and LED.

Design Specifications:

There are 3 inputs and 1 output. The user first selects which operation to implement by inputing the 3 bit input, sel(sel(0), sel(1), sel(2)). Next, the user gives 2 3-bit numbers for in_a(in_a1, in_a2, in_a3) and in_b(in_b1, in_b2, in_b3). The 4 bit output (out_m1, out_m2, out_m3, out_m4) is projected via. LED's.

The VHDL code is constructed in a modular fashion. "main.VHDL" is the top-level module that toggles between user selections and outputs the results of the selected operation. There are eight sub-modules under the top module:

- summation.vhdl
- multiplication.vhdl
- substractor.vhdl
- comparator.vhdl
- barrel_shifter.vhdl
- xnorgate.vhdl
- andgate.vhdl
- orgate.vhdl

Furthermore, in order to sustain the modularity of the design, there are two more sub_modules under the above-mentioned sub_modules:

- full_adder.vhdl
- half_adder.vhdl

One possible approach could be including further sub_modules into the adders such as and gate or gate and xor gate, but these gates are already implemented in VHDL. Thus, it is aimed to keep things simple by not creating further sub_modules.

For all but the subtraction operation, it is operated by two 3-bit unsigned numbers. The subtracter inputs two 3-bit signed numbers and outputs one 3-bit signed number. For the multiplication operation, overflow is possible. For future implementations, an overflow detector can be added. For the switching operation, if-else clause is used.

The operations can be summarized as:

SELECT/OPERATION	INPUT/OUTPUT	EXAMPLE
"000" / Summation	in_a & in_b / in_a + in_b	"101" + "011" = "1000"
"001" / Multiplication	in_a & in_b / in_a * in_b	"111" * "101" = "0011"
"010" / Substraction	in_a & in_b / in_a - in_b	"100" - "001" = "011"
"011" / Comparator	in_a & in_b / in_a >,=,< in_b	"000" & "111" = "0010"
"100" / Barrel Shifter	in_a / "0" in_a3 in_a1 in_a2	"110" = "0011"
"101" / Xnor Gate	in_a & in_b / in_a xnor in_b	"111" XNOR "011" = "0000"
"110" / And Gate	in_a & in_b / in_a and in_b	"111" AND "111" = "0001"
"111" / Or Gate	in_a & in_b / in_a or in_b	"000" OR "000" = "0000"

Figure 1.1 The Operation Table

Results:

The proposed design is first initialized via. VHDL. Due to the complexity of the process, after each creation of a module, the code is simulated with preselected values in the

testbench. The finalized RTL schematics can be seen in Figure 1.2 below. The individual modules of the design can be seen in the Appendix section.

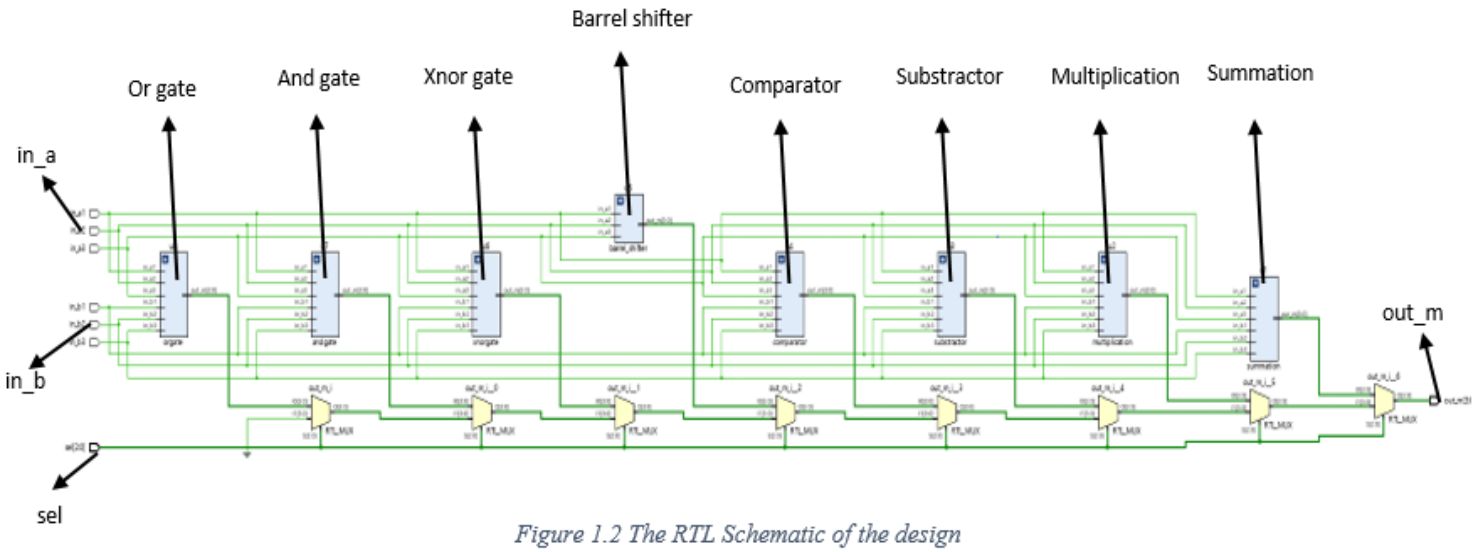


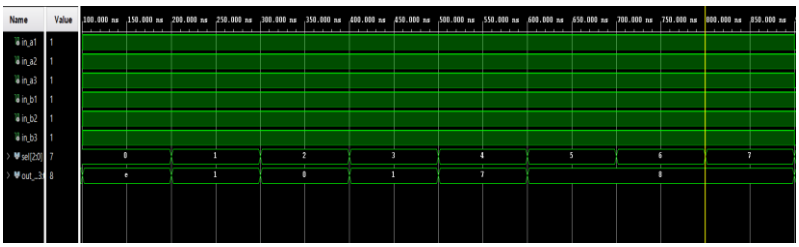
Figure 1.2 The RTL Schematic of the design

For simulation purposes, there used three different input selections and each of them is processed into 8 of the operations:

For the first input combination:

in_a : “111”

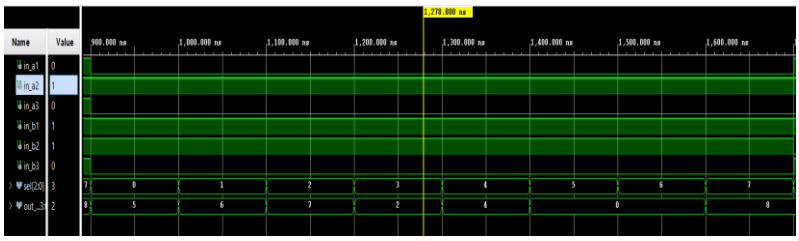
in_b: “111”



For the second input combination:

in_a: “010”

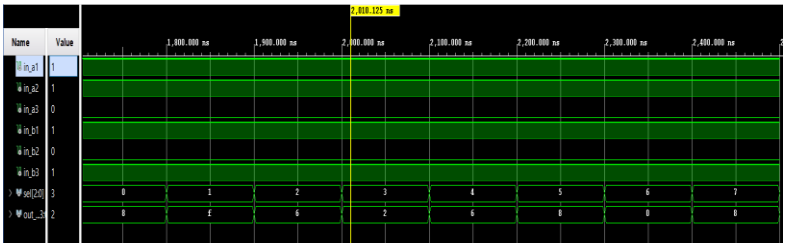
in_b: “110”



For the third input combination:

in_a: “011”

in_b: “101”



For the implementation, it is used the Basys3 board. The pin set follows as:

in_a1: V17 switch

in_a2: V16 switch

in_a3: W16 switch

in_b1: W15 switch

in_b2: V15 switch

in_b3: W14 switch

sel(0): U1 switch

sel(1): T1 switch

sel(2): R2 switch

out_m(0): U16 LED

out_m(1): E19 LED

out_m(2): U19 LED

out_m(3): V19 LED

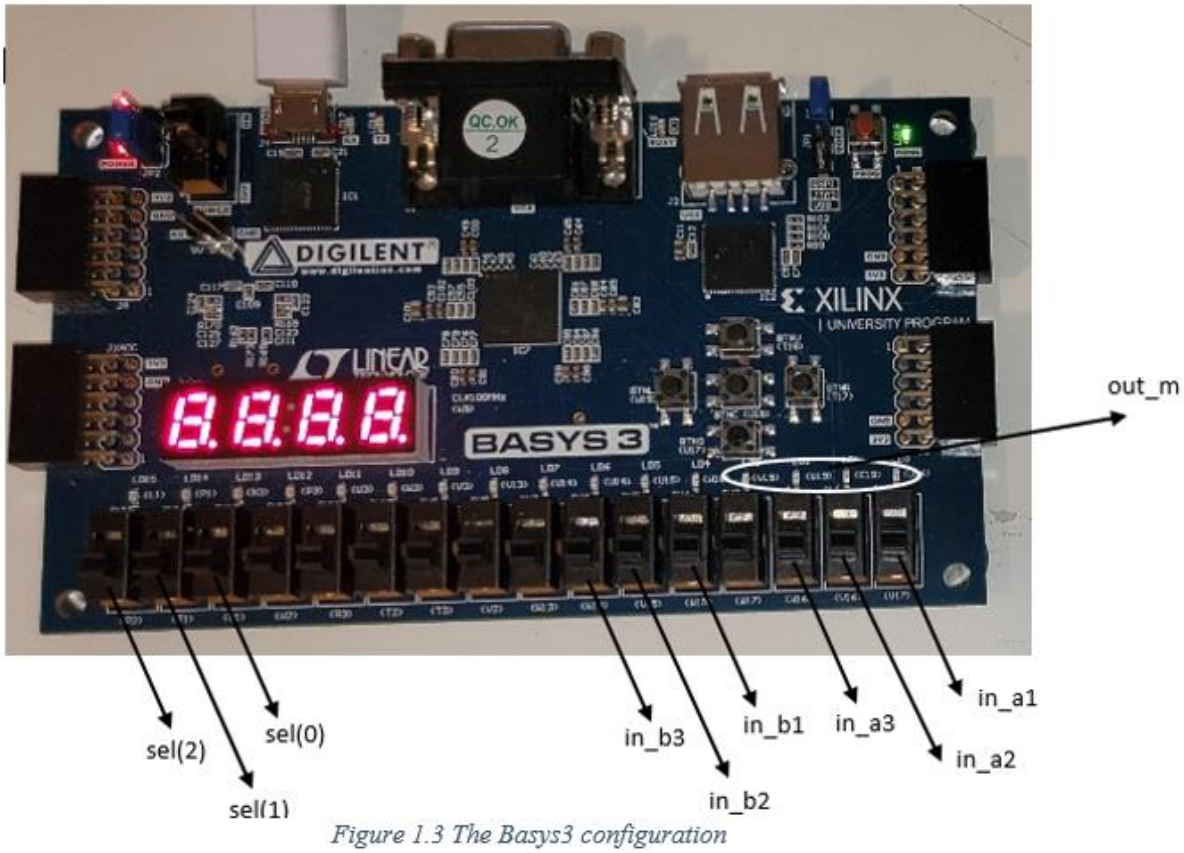


Figure 1.3 The Basys3 configuration

In order to verify that the Basys3 implementation is working accordingly, it is choosen 3 different cases from the above mentioned waveform.

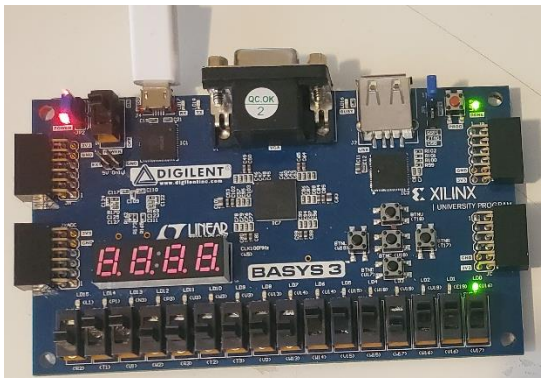
Case 1:

in_a: “111”

in_b: “111”

sel: “011”

expected out_m: “0001”



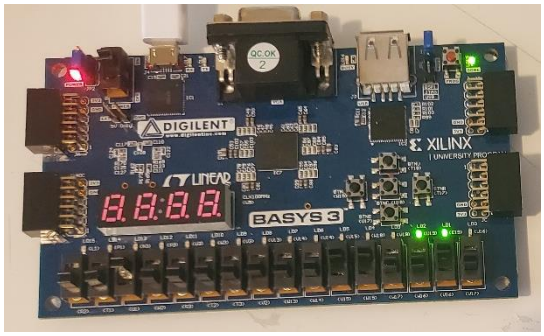
Case 2:

in_a: “010”

in_b: “011”

sel: “001”

expected out_m: “0110”



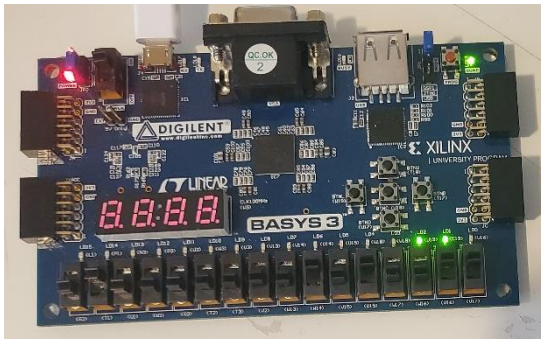
Case 3:

in_a: “011”

in_b: “101”

sel: “010”

expected out_m: “110”



Conclusion:

In this lab, we implemented an Arithmetic Logic Unit using VHDL and Basys3. Eight different operations are selected, and three 3-bit numbers are inputted into the design, resulting in a 4-bit output. The design is implemented in a modular fashion. During the designing process, there were some design-related problems. For instance, the inputs in_a and in_b used to drive all the sub_modules in the initial design. This gave a driver error. The inputs are assigned to signals then distributed over the modules, which acted like a buffer to solve it. Also, deciding on the switching algorithm was another hurdle. In conditional assignments, it is not allowed to initialize submodules by port map. Again to overcome this, the outputs from the modules are assigned to unique signals. By using if-else clause, the signals are connected to the final output, out_m, using the if-else clause. The design can be improved by increasing the output bit or adding an overflow detector. The waveform and FPGA implementation is working as expected. The VHDL code is implemented on the recent(2021) version of the Vivado; thus the syntax could differ from the older versions.

Appendix:

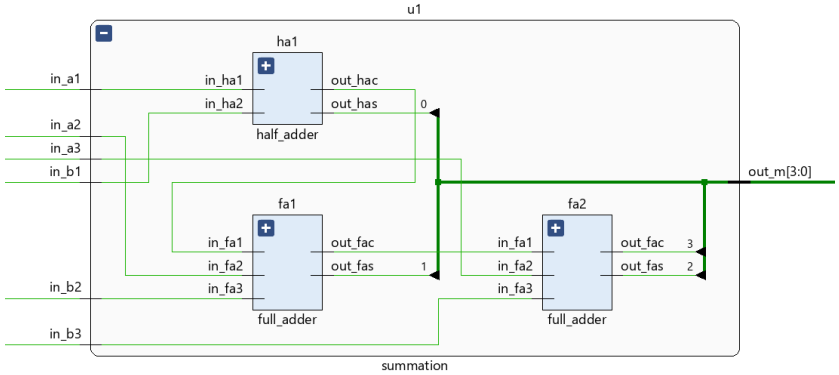


Figure A: Summation Schematic

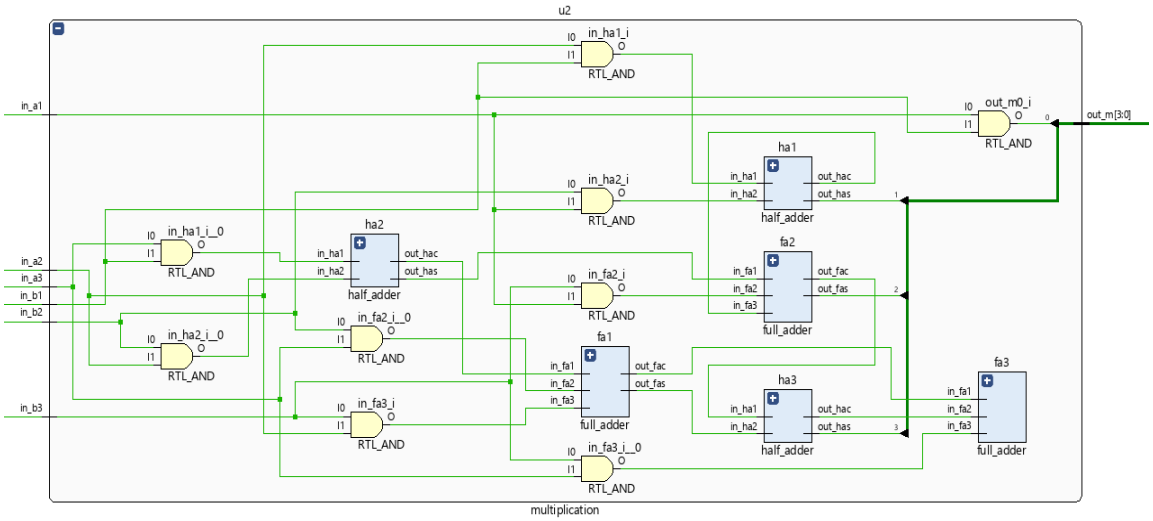


Figure B: Multipliation Schematic

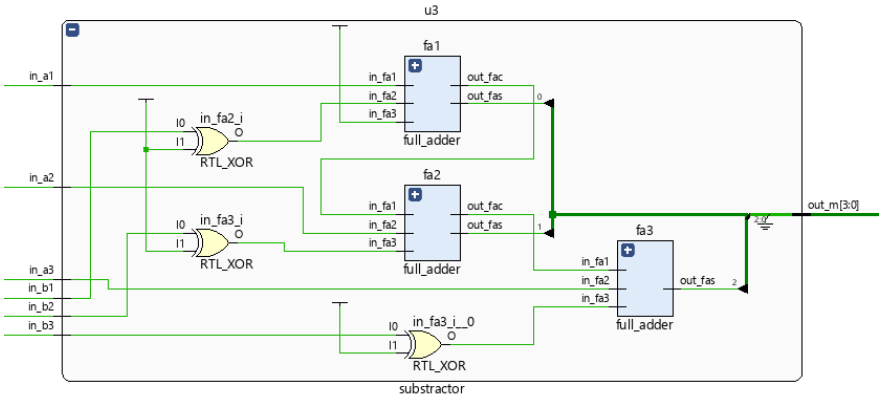


Figure C: Substraction Schematic

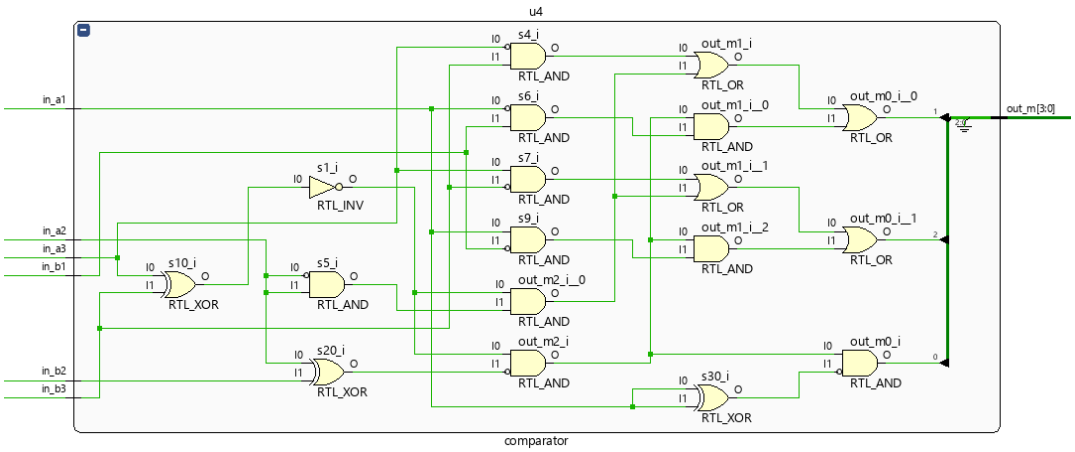


Figure D: Comparator Schematic

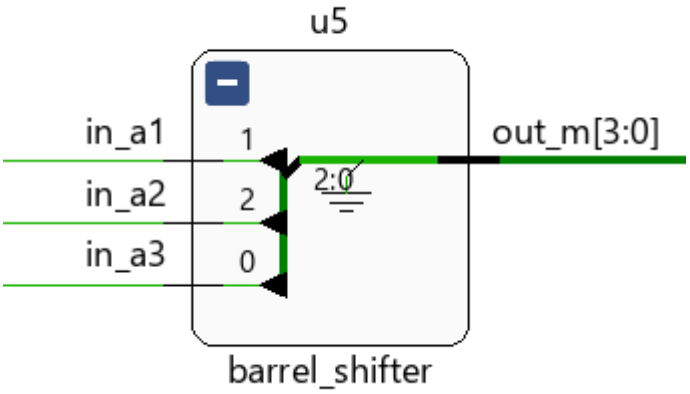


Figure E: Barrel-Shifter Schematic

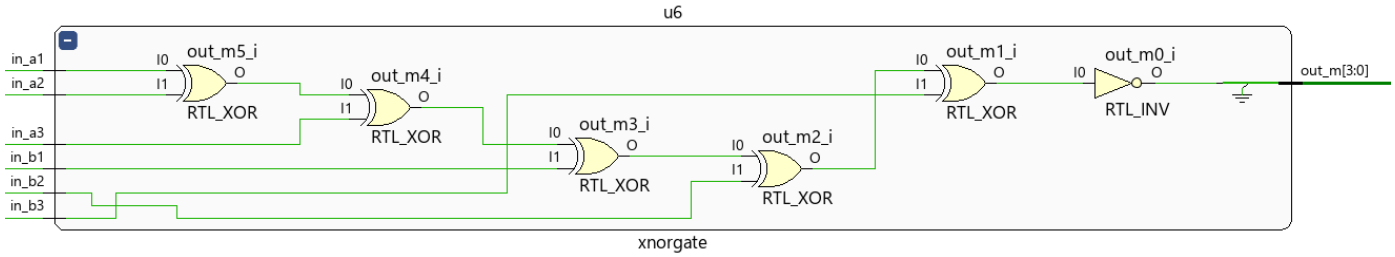


Figure F: Xnor gate Schematic

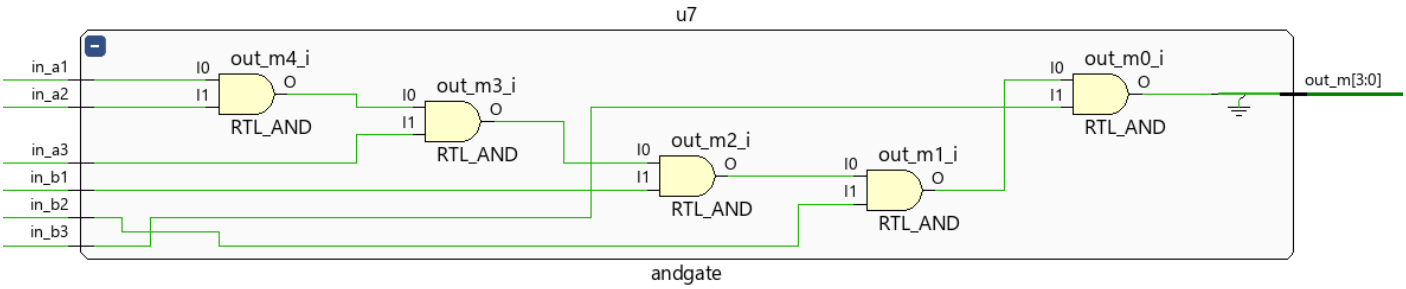


Figure G:And gate Schematic

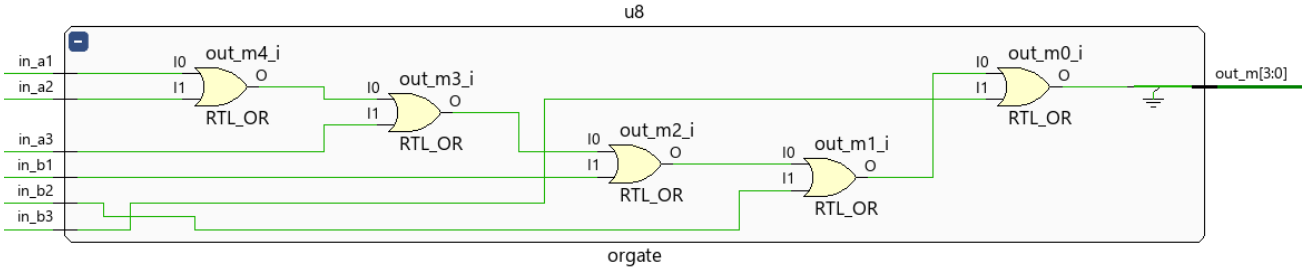


Figure H: Or gate Schematic

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 summation
    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 multiplication
    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 substractor
    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 comparator
    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 barrel_shifter
    Port (in_a1,in_a2, in_a3: in std_logic;
           out_m: out STD_LOGIC_vector(3 downto 0));
end component;
component xnorgate
    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 andgate
    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 orgate
    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;
signal ina1,ina2,ina3,inb1,inb2,inb3: std_logic;
signal out1, out2, out3, out4, out5, out6, out7, out8: std_logic_vector(3 downto 0);
begin
ina1 <= in_a1;
ina2 <= in_a2;
ina3 <= in_a3;
inb1 <= in_b1;
inb2 <= in_b2;
inb3 <= in_b3;
u1: summation
port map(in_a1 => ina1,in_a2 => ina2, in_a3 => ina3,
         in_b1 => inb1,in_b2 => inb2, in_b3 => inb3,
         out_m => out1);
u2: multiplication
port map(in_a1 => ina1,in_a2 => ina2, in_a3 => ina3,
         in_b1 => inb1,in_b2 => inb2, in_b3 => inb3,
```

```
        out_m => out2);
u3: substractor
port map(in_a1 => ina1,in_a2 => ina2, in_a3 => ina3,
        in_b1 => inb1,in_b2 => inb2, in_b3 => inb3,
        out_m => out3);
u4: comparator
port map(in_a1 => ina1,in_a2 => ina2, in_a3 => ina3,
        in_b1 => inb1,in_b2 => inb2, in_b3 => inb3,
        out_m => out4);
u5: barrel_shifter
port map(in_a1 => ina1,in_a2 => ina2, in_a3 => ina3,
        out_m => out5);
u6: xnorgate
port map(in_a1 => ina1,in_a2 => ina2, in_a3 => ina3,
        in_b1 => inb1,in_b2 => inb2, in_b3 => inb3,
        out_m => out6);
u7: andgate
port map(in_a1 => ina1,in_a2 => ina2, in_a3 => ina3,
        in_b1 => inb1,in_b2 => inb2, in_b3 => inb3,
        out_m => out7);
u8: orgate
port map(in_a1 => ina1,in_a2 => ina2, in_a3 => ina3,
        in_b1 => inb1,in_b2 => inb2, in_b3 => inb3,
        out_m => out8);
```

```
process(sel, out1, out2, out3, out4, out5, out6, out7, out8)
begin
    if sel = "000" then
        out_m <= out1;
    elsif sel = "001" then
        out_m <= out2;
    elsif sel = "010" then
        out_m <= out3;
    elsif sel = "011" then
        out_m <= out4;
    elsif sel = "100" then
        out_m <= out5;
    elsif sel = "101" then
        out_m <= out6;
    elsif sel = "110" then
        out_m <= out7;
    elsif sel = "111" then
        out_m <= out8;
    else
        out_m <= "0000";
    end if;
end process;
end Behavioral;
```

summation.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- entitiy declare
entity summation 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 summation;
-- architecture declare
architecture Behavioral of summation is
--half adder module is declared
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;
--full adder module is declared
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 declaration for connections
signal hac,fac1: std_logic;
begin
    ha1:half_adder -- initiating the first half-adder
    port map(in_ha1 => in_a1, in_ha2 => in_b1,
            out_has => out_m(0), out_hac => hac);
    fa1:full_adder -- initiating the first 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 -- initiating the second 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;
```

multiplication.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- entitiy declare
entity multiplication 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 multiplication;
-- architecture declare
architecture Behavioral of multiplication is
```

```
--half adder module is declared
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;
--full adder module is declared
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 declaration for connections
signal sum1,sum2: std_logic;
signal carry1,carry2,carry3,carry4,carry5: std_logic;
signal dump2,dump3: std_logic;
signal sel: std_logic;
begin
out_m(0) <= in_a1 and in_b1;
ha1: half_adder -- initiating the first half-adder
port map(in_ha1 => (in_a2 and in_b1), in_ha2 =>(in_b2 and in_a1),
```

```
        out_has => out_m(1), out_hac => carry1);
ha2: half_adder -- initiating the second half-adder
port map(in_ha1 => (in_a3 and in_b1), in_ha2 =>(in_b2 and in_a2),
        out_has => sum1, out_hac => carry2);
fa1: full_adder -- initiating the first full-adder
port map(in_fa1 => carry2, in_fa2 =>(in_b2 and in_a3), in_fa3 => (in_b3 and in_a2),
        out_fas => sum2, out_fac => carry3);
fa2: full_adder -- initiating the second full-adder
port map(in_fa1 => sum1, in_fa2 =>(in_b3 and in_a1), in_fa3 => carry1,
        out_fas => out_m(2), out_fac => carry4);
ha3: half_adder -- initiating the third half-adder
port map(in_ha1 => carry4, in_ha2 =>sum2,
        out_has => out_m(3), out_hac => carry5);
fa3: full_adder -- initiating the third full-adder
port map(in_fa1 => carry3, in_fa2 =>carry5, in_fa3 => (in_b3 and in_a3),
        out_fas => dump2, out_fac => dump3);
end Behavioral;
```

substractor.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- entitiy declare
entity substractor 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 substractor;
-- architecture declare
architecture Behavioral of substractor is
--full adder module is declared
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 declaration for connections
signal dummy,fac1,fac2: std_logic;
begin
    fa1:full_adder -- initiating the first full-adder
    port map(in_fa1 => in_a1, in_fa2 => (in_b1 xor '1'), in_fa3 => '1',
             out_fas => out_m(0), out_fac => fac1);
    fa2:full_adder -- initiating the second full-adder
    port map(in_fa1 => fac1, in_fa2 => in_a2, in_fa3 => (in_b2 xor '1'),
             out_fas => out_m(1), out_fac => fac2);
    fa3:full_adder -- initiating the third full-adder
    port map(in_fa1 => fac2, in_fa2 => in_a3, in_fa3 => (in_b3 xor '1'),
             out_fas => out_m(2), out_fac => dummy);
    out_m(3) <= '0';
```

```
end Behavioral;
```

comparator.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- entitiy declare
entity comparator 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 comparator;
-- architecture declare
architecture Behavioral of comparator is
--signal declaration for connections
signal s1,s2,s3,s4,s5,s6,s7,s8,s9: std_logic;
begin
s1 <= not(in_a3 xor in_b3);
s2 <= not(in_a2 xor in_b2);
s3 <= not(in_a1 xor in_b1);
s4 <= (not in_a3) and in_b3;
s5 <= (not in_a2) and in_b2;
s6 <= (not in_a1) and in_b1;
s7 <= in_a3 and (not in_b3);
s8 <= in_a2 and (not in_b2);
s9 <= in_a1 and (not in_b1);
out_m(0) <= s1 and s2 and s3;  -- for a equals b.
```

```
out_m(1) <= s4 or (s1 and s5) or (s1 and s2 and s6); --for a less than b

out_m(2) <= s7 or (s1 and s8) or (s1 and s2 and s9); --for a greater than b

out_m(3) <= '0';

end Behavioral;
```

barrel_shifter.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity barrel_shifter is

    Port (in_a1,in_a2, in_a3: in STD_LOGIC;

          out_m: out STD_LOGIC_vector(3 downto 0));

end barrel_shifter;

architecture Behavioral of barrel_shifter is

signal a1, a2, a3: std_logic;

begin

a1<= in_a1;

a2<= in_a2;

a3<= in_a3;


out_m <= '0' & a2 & a1 & a3;

end Behavioral;
```

xnorgate.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity xnorgate 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):= "0000");
end xnorgate;
architecture Behavioral of xnorgate is
begin
out_m(3) <= (not(in_a1 xor in_a2 xor in_a3 xor in_b1 xor in_b2 xor in_b3));
end Behavioral;
```

andgate.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity andgate 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):= "0000");
end andgate;
architecture Behavioral of andgate is
begin
out_m(3) <= in_a1 and in_a2 and in_a3 and in_b1 and in_b2 and in_b3;
end Behavioral;
```

orgate.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity orgate 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):= "0000");
```

```
end orgate;
architecture Behavioral of orgate is
begin
out_m(3) <= in_a1 or in_a2 or in_a3 or in_b1 or in_b2 or in_b3;
end Behavioral;
```

half_adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity half_adder is
    Port ( in_ha1 : in STD_LOGIC;
           in_ha2 : in STD_LOGIC;
           out_has : out STD_LOGIC;
           out_hac : out STD_LOGIC);
end half_adder;
architecture Behavioral of half_adder is
begin
out_has <= in_ha1 xor in_ha2;
out_hac <= in_ha1 and in_ha2;
end Behavioral;
```

full_adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder is
    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 full_adder;
architecture Behavioral of full_adder is
begin
out_fas <= in_fa1 xor in_fa2 xor in_fa3;
out_fac <= (in_fa1 and in_fa2) or (in_fa1 and in_fa3) or (in_fa2 and in_fa3);
end Behavioral;
```

testBench_main.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity testBench_main is
end testBench_main;
architecture Behavioral of testBench_main is
component main
    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 component;
    signal in_a1, in_a2,in_a3,in_b1,in_b2,in_b3: std_logic;
    signal sel: std_logic_vector(2 downto 0);
    signal out_m: std_logic_vector(3 downto 0);
```

```
begin
    u1: main
    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, sel => sel, out_m =>out_m);
testBench_main: process
begin
    sel <= "000";
    in_a1 <= '0';
    in_a2 <= '0';
    in_a3 <= '0';
    in_b1 <= '0';
    in_b2 <= '0';
    in_b3 <= '0';
    wait for 100ns;
        sel <= "000";
        in_a1 <= '1';
        in_a2 <= '1';
        in_a3 <= '1';
        in_b1 <= '1';
        in_b2 <= '1';
        in_b3 <= '1';
    wait for 100ns;
        sel <= "001";
        in_a1 <= '1';
        in_a2 <= '1';
```

```
    in_a3 <= '1';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '1';
 wait for 100ns;
    sel <= "010";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '1';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '1';
wait for 100ns;
    sel <= "011";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '1';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '1';
wait for 100ns;
    sel <= "100";
    in_a1 <= '1';
    in_a2 <= '1';
```

```
    in_a3 <= '1';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '1';
wait for 100ns;
    sel <= "101";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '1';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '1';
wait for 100ns;
    sel <= "110";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '1';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '1';
wait for 100ns;
    sel <= "111";
    in_a1 <= '1';
    in_a2 <= '1';
```

```
        in_a3 <= '1';
        in_b1 <= '1';
        in_b2 <= '1';
        in_b3 <= '1';
-----------------------------------------------
    wait for 100ns;
        sel <= "000";
        in_a1 <= '0';
        in_a2 <= '1';
        in_a3 <= '0';
        in_b1 <= '1';
        in_b2 <= '1';
        in_b3 <= '0';
    wait for 100ns;
        sel <= "001";
        in_a1 <= '0';
        in_a2 <= '1';
        in_a3 <= '0';
        in_b1 <= '1';
        in_b2 <= '1';
        in_b3 <= '0';

     wait for 100ns;
        sel <= "010";
```

```
    in_a1 <= '0';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '0';
wait for 100ns;
    sel <= "011";
    in_a1 <= '0';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '0';
wait for 100ns;
    sel <= "100";
    in_a1 <= '0';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '1';
    in_b3 <= '0';

wait for 100ns;
```

```
        sel <= "101";
        in_a1 <= '0';
        in_a2 <= '1';
        in_a3 <= '0';
        in_b1 <= '1';
        in_b2 <= '1';
        in_b3 <= '0';
    wait for 100ns;
        sel <= "110";
        in_a1 <= '0';
        in_a2 <= '1';
        in_a3 <= '0';
        in_b1 <= '1';
        in_b2 <= '1';
        in_b3 <= '0';
    wait for 100ns;
        sel <= "111";
        in_a1 <= '0';
        in_a2 <= '1';
        in_a3 <= '0';
        in_b1 <= '1';
        in_b2 <= '1';
        in_b3 <= '0';
-----------------------------------------------
```

```
wait for 100ns;
    sel <= "000";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '0';
    in_b3 <= '1';
wait for 100ns;
    sel <= "001";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '0';
    in_b3 <= '1';
 wait for 100ns;
    sel <= "010";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '0';
    in_b3 <= '1';
```

```
wait for 100ns;
    sel <= "011";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '0';
    in_b3 <= '1';
wait for 100ns;
    sel <= "100";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '0';
    in_b3 <= '1';
wait for 100ns;
    sel <= "101";
    in_a1 <= '1';
    in_a2 <= '1';
    in_a3 <= '0';
    in_b1 <= '1';
    in_b2 <= '0';
    in_b3 <= '1';
```

```
    wait for 100ns;
        sel <= "110";
        in_a1 <= '1';
        in_a2 <= '1';
        in_a3 <= '0';
        in_b1 <= '1';
        in_b2 <= '0';
        in_b3 <= '1';
    wait for 100ns;
        sel <= "111";
        in_a1 <= '1';
        in_a2 <= '1';
        in_a3 <= '0';
        in_b1 <= '1';
        in_b2 <= '0';
        in_b3 <= '1';
   wait for 100ns;
end process;
end Behavioral;
```

calculatorconstraints.xdc

```
set_property PACKAGE_PIN V17 [get_ports {in_a1}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in_a1}]
set_property PACKAGE_PIN V16 [get_ports {in_a2}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in_a2}]
```

```
set_property PACKAGE_PIN W16 [get_ports {in_a3}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in_a3}]
set_property PACKAGE_PIN W15 [get_ports {in_b1}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in_b1}]
set_property PACKAGE_PIN V15 [get_ports {in_b2}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in_b2}]
set_property PACKAGE_PIN W14 [get_ports {in_b3}]
    set_property IOSTANDARD LVCMOS33 [get_ports {in_b3}]
set_property PACKAGE_PIN U1 [get_ports {sel[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sel[0]}]
set_property PACKAGE_PIN T1 [get_ports {sel[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sel[1]}]
set_property PACKAGE_PIN R2 [get_ports {sel[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {sel[2]}]
set_property PACKAGE_PIN U16 [get_ports {out_m[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out_m[0]}]
set_property PACKAGE_PIN E19 [get_ports {out_m[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out_m[1]}]
set_property PACKAGE_PIN U19 [get_ports {out_m[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out_m[2]}]
set_property PACKAGE_PIN V19 [get_ports {out_m[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out_m[3]}]
```