Bilkent University EEE102

Lab 04 Report:

Arithmetic Logic Unit

Burak Gökkaya 22202209

Purpose:

The aim of the this lab was to build an Arithmetic Logic Unit (ALU). It should have 8 functions includiing at least one bitwise, one shift, addition and subtraction operators.

Methodology:

Firstly we understood the what is ALU. Then we decided the which functions we will use. I choose:

- Adder (OP is '000')
- Subtracter (OP is '001')
- NAND (OP is '010')
- Comparator (OP is '011')
- XOR (OP is '100')
- Rotation (OP is '101')
- Shifting Left (OP is '110')
- Shifting Right (OP is '111')

Then I started to write the functions one by one. For example, for adder I used half adders and full adders. Then I generated the RTL schematic of my code.

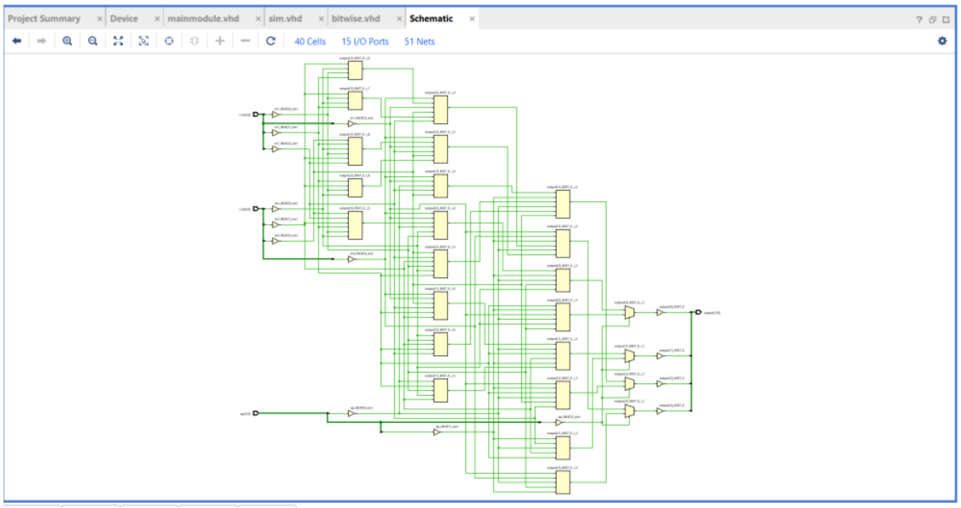

Figure 1: Schematic of system

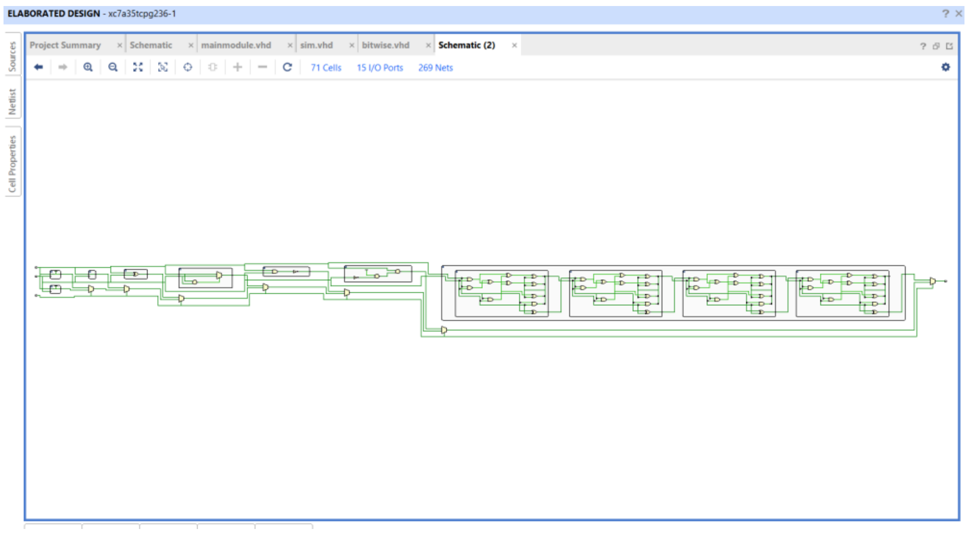

Figure 2: Elaborated Design

After that we simulated the code and it worked like what I wanted.

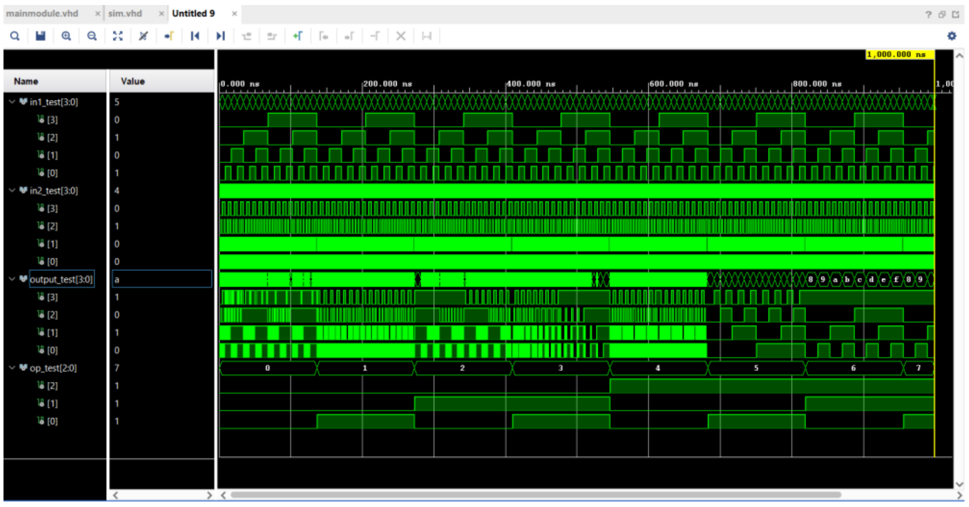

Figure 3: Simulation of code (op is select input)

After all of these stages, I generated bitstream and implemented the code on BASYS3.

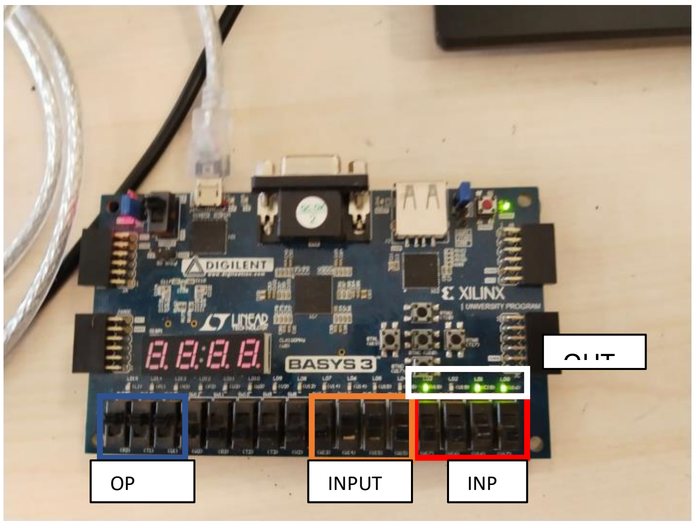

Figure 4: Shifting right to 0111

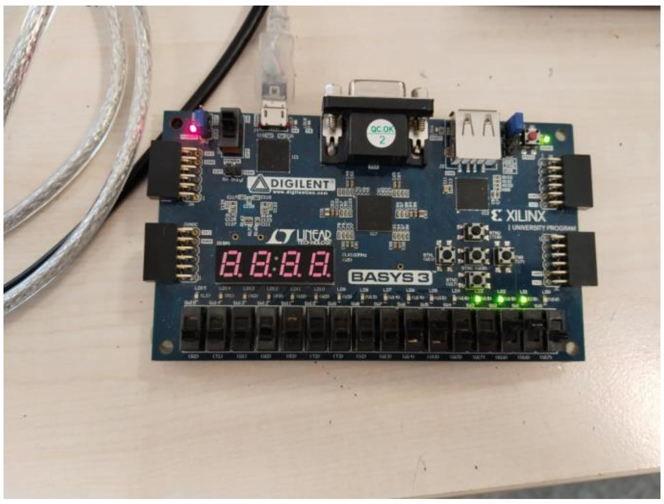

Figure 5: Addition of ‘0110’ and ‘0110’

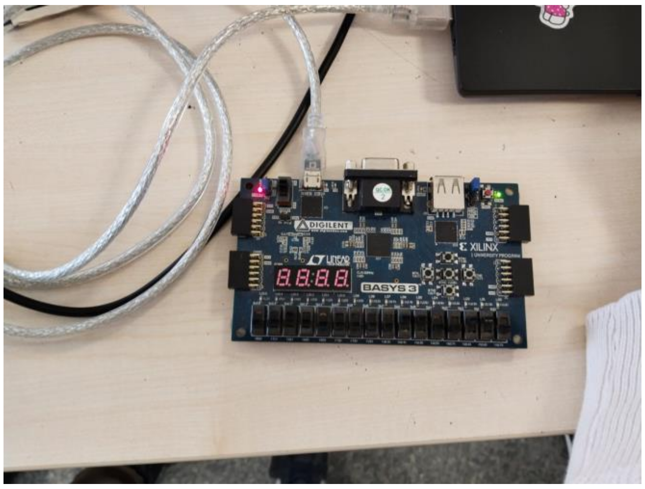

Figure 6: Subtraction of ‘0110’ and ‘0110’

Conclusion:

As a result, I understood BASYS3 mechanism very good and I learned ALU. IN this lab we wrote some functions and it helps us for understanding computer algorithm. So that it was really successfull experiment.

Appendix:

----main module

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

entity mainmodule is
   Port ( in1 : in STD_LOGIC_VECTOR (3 downto 0);
         in2 : in STD_LOGIC_VECTOR (3 downto 0);
         op : in STD_LOGIC_VECTOR (2 downto 0);
         output : out STD_LOGIC_VECTOR (3 downto 0));
end mainmodule;

architecture Behavioral of mainmodule is
   component adder is
      Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
         B : in STD_LOGIC_VECTOR (3 downto 0);
         Sum : out STD_LOGIC_VECTOR (3 downto 0);
         Cout : out STD_LOGIC_VECTOR (3 downto 0));
   end component;
   component subt is
      Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
         B : in STD_LOGIC_VECTOR(3 downto 0);
         diff : out STD_LOGIC_VECTOR(3 downto 0));
   end component;
   component shiftRight is
      Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
         B : in STD_LOGIC_VECTOR(3 downto 0);
         resR : out STD_LOGIC_VECTOR(3 downto 0));
   end component;
   component shiftLeft is
        Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
         B : in STD_LOGIC_VECTOR(3 downto 0);
         resL : out STD_LOGIC_VECTOR(3 downto 0));
    end component;
    component comparison is
      Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
         B : in STD_LOGIC_VECTOR(3 downto 0);
         comp : out STD_LOGIC_VECTOR(3 downto 0));
    end component;
    component Bitwise_Logic is
         Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
         B : in STD_LOGIC_VECTOR(3 downto 0);
         Result : out STD_LOGIC_VECTOR(3 downto 0));
    end component;
    component rotate is
         Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
         B : in STD_LOGIC_VECTOR(3 downto 0);
         resRo : out STD_LOGIC_VECTOR(3 downto 0));
    end component;
    component bitwise_nand is
         Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
         B : in STD_LOGIC_VECTOR(3 downto 0);
         Result_nand : out STD_LOGIC_VECTOR(3 downto 0));
    end component;

signal temp_result,out1,out2,out3,out4,out5,out6,out7,out8 : STD_LOGIC_VECTOR(3 downto 0);

begin
u1: adder
port map(A => in1, B =>in2,  Sum => out1);
```

```
u2: subt
port map(A => in1, B =>in2,  diff => out2);
u3: bitwise_nand
port map(A => in1, B =>in2,  Result_nand => out3);
u4: comparison
port map(A => in1, B =>in2,  comp => out4);
u5: Bitwise_Logic
port map(A => in1, B =>in2,  Result => out5);
u6: rotate
port map(A => in1, B =>in2,  resRo => out6);
u7: shiftLeft
port map(A => in1, B =>in2,  resL => out7);
u8: shiftRight
port map(A => in1, B =>in2,  resR => out8);
 process(op, out1, out2, out3, out4, out5, out6, out7, out8)
   begin
      if op = "000" then
         output <= out1;
      elsif op = "001" then
         output <= out2;
      elsif op = "010" then
         output <= out3;
      elsif op = "011" then
         output <= out4;
      elsif op = "100" then
         output <= out5;
      elsif op = "101" then
         output <= out6;
      elsif op = "110" then
         output <= out7;
      else
```

```
            output <= out8;
        end if;
    end process;
end Behavioral;
```

------full adder

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

entity adder is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Sum : out STD_LOGIC_VECTOR (3 downto 0);
           Cout : out STD_LOGIC_VECTOR (3 downto 0));
end adder;

architecture Behavioral of adder is
    signal sumHA1, sumHA2,carryHA1,carryHA2,sumHA3,carryHA3,sumHA4,carryHA4 :
STD_LOGIC_VECTOR (3 downto 0);
    component halfadder is
        Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
               B : in STD_LOGIC_VECTOR (3 downto 0);
               sum : out STD_LOGIC_VECTOR (3 downto 0);
               carry : out STD_LOGIC_VECTOR (3 downto 0));
    end component;
begin
    -- First half adder
    HA1: halfadder port map (A => A, B => B, sum => sumHA1, carry => carryHA1);
    HA2: halfadder port map (A => sumHA1, B => carryHA1, sum => sumHA2, carry => carryHA2);
    HA3: halfadder port map (A => sumHA2, B => carryHA2, sum => sumHA3, carry => carryHA3);
    HA4: halfadder port map (A => sumHA3, B => carryHA3, sum => sumHA4, carry => carryHA4);

    -- Final sum
    Sum <= sumHA4;

end Behavioral;
```

-----half adder

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

entity halfadder is
    Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
           B : in STD_LOGIC_VECTOR(3 downto 0);
           sum : out STD_LOGIC_VECTOR(3 downto 0);
           carry : out STD_LOGIC_VECTOR(3 downto 0));
end halfadder;

architecture Behavioral of halfadder is
begin
    process(A, B)
    begin
        Sum <= (A xor B);
        carry(0) <= (A(0) and B(0));
        carry(1) <= (A(1) and B(1)) or (A(0) and B(0));
        carry(2) <= (A(2) and B(2)) or (A(1) and B(1)) or (A(0) and B(0));
        carry(3) <= (A(3) and B(3)) or (A(2) and B(2)) or (A(1) and B(1)) or (A(0) and B(0));
    end process;
end Behavioral;
```

-----subtraction

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

entity subt is
   Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
          B : in STD_LOGIC_VECTOR(3 downto 0);
          diff : out STD_LOGIC_VECTOR(3 downto 0));
end subt;

architecture Behavioral of subt is
signal B_not: STD_LOGIC_VECTOR(3 downto 0);
begin
   B_not <= not B + 0001;
   diff <= A + B_not;
end Behavioral;
```

----nand gate

-- Bitwise Logic Module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity bitwise_nand is
   Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
          B : in STD_LOGIC_VECTOR(3 downto 0);
          Result_nand : out STD_LOGIC_VECTOR(3 downto 0));
end bitwise_nand;

architecture Behavioral of bitwise_nand is
```

```
begin
   process (A, B)
   begin
   Result_nand <= A nand B;
   end process;
end Behavioral;

----comparison
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comparison is
   Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
         B : in STD_LOGIC_VECTOR(3 downto 0);
         comp : out STD_LOGIC_VECTOR(3 downto 0));
end comparison;

architecture Behavioral of comparison is
begin
   process(A, B)
   begin
      if A > B then
         comp <= A;
      else
         comp <= B;
      end if;
   end process;
end Behavioral;

----xor gate
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Bitwise_Logic is
   Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
          B : in STD_LOGIC_VECTOR(3 downto 0);
          Result : out STD_LOGIC_VECTOR(3 downto 0));
end Bitwise_Logic;

architecture Behavioral of Bitwise_Logic is
begin
   process (A, B)
   begin
   Result <= A xor B;
   end process;
end Behavioral;
```

-----rotation

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity rotate is
   Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
          B : in STD_LOGIC_VECTOR(3 downto 0);
          resRo : out STD_LOGIC_VECTOR(3 downto 0));
end rotate;

architecture Behavioral of rotate is
signal new_res: STD_LOGIC_VECTOR(3 downto 0);
begin

new_res(0) <= A(3);
new_res(1) <= A(2);
new_res(2) <= A(1);
new_res(3)<= A(0);
resRo <= new_res;
end Behavioral;

-----shifting Left
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity shiftLeft is
   Port ( A : in STD_LOGIC_VECTOR(3 downto 0);
          B : in STD_LOGIC_VECTOR(3 downto 0);
          resL : out STD_LOGIC_VECTOR(3 downto 0));
end shiftLeft;

architecture Behavioral of shiftLeft is
signal new_res: STD_LOGIC_VECTOR(3 downto 0);
begin
```

```
new_res(0) <= A(1);

new_res(1) <= A(2);

new_res(2) <= A(3);

new_res(3)<= '1';

resL <= new_res;

end Behavioral;
```

------shift Right

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

 --use UNISIM.VComponents.all;


entity shiftRight is

   Port ( A : in STD_LOGIC_VECTOR(3 downto 0);

          B : in STD_LOGIC_VECTOR(3 downto 0);

          resR : out STD_LOGIC_VECTOR(3 downto 0));

end shiftRight;


architecture Behavioral of shiftRight is

signal new_res: STD_LOGIC_VECTOR(3 downto 0);

begin


new_res(0) <= A(1);

new_res(1) <= A(2);

new_res(2) <= A(3);

new_res(3)<= '1';

resR <= new_res;

end Behavioral;
```

----simulation sources

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

entity sim is
end sim;

architecture Behavioral of sim is
  signal in1_test : std_logic_vector(3 downto 0) := (others => '0');
  signal in2_test : std_logic_vector(3 downto 0) := (others => '0');
  signal output_test : std_logic_vector(3 downto 0) := (others => '0');
  signal op_test : std_logic_vector(2 downto 0) := (others => '0');
begin
  DUT: entity work.mainmodule
    port map (
      in1 => in1_test,
      in2 => in2_test,
      output => output_test,
      op => op_test
    );

  process
  begin
    for i in 0 to 7 loop
      op_test <= std_logic_vector(to_unsigned(i, op_test'length));
      for a in 0 to 15 loop
        in1_test <= std_logic_vector(to_unsigned(a, in1_test'length));
        for b in 0 to 15 loop
          in2_test <= std_logic_vector(to_unsigned(b, in2_test'length));
          wait for 0.5ns;
        end loop;
```

```
            wait for 0.5ns;
         end loop;
         wait for 0.5ns;
      end loop;
   end process;
end Behavioral;
```

----constraint

```
set_property PACKAGE_PIN V17 [get_ports {in2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in2[0]}]
set_property PACKAGE_PIN V16 [get_ports {in2[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in2[1]}]
set_property PACKAGE_PIN W16 [get_ports {in2[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in2[2]}]
set_property PACKAGE_PIN W17 [get_ports {in2[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in2[3]}]
set_property PACKAGE_PIN W15 [get_ports {in1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in1[0]}]
set_property PACKAGE_PIN V15 [get_ports {in1[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in1[1]}]
set_property PACKAGE_PIN W14 [get_ports {in1[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in1[2]}]
set_property PACKAGE_PIN W13 [get_ports {in1[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {in1[3]}]
set_property PACKAGE_PIN R2 [get_ports {op[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {op[2]}]
set_property PACKAGE_PIN T1 [get_ports {op[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {op[1]}]
set_property PACKAGE_PIN U1 [get_ports {op[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {op[0]}]
```

```
# LEDs
set_property PACKAGE_PIN U16 [get_ports {output[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {output[0]}]
set_property PACKAGE_PIN E19 [get_ports {output[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {output[1]}]
set_property PACKAGE_PIN U19 [get_ports {output[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {output[2]}]
set_property PACKAGE_PIN V19 [get_ports {output[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {output[3]}]
```