Computer Organization & Architecture Lab (PCCCS492) - 2023



PROJECT REPORT ON

**IMPLEMENTATION OF 8-BIT ALU**

Submitted By

| NAME | SECTION | ROLL NO. | ENROLLMENT NO. |
| --- | --- | --- | --- |
| Himanshu Singh | CSE - 2A - C1 | 30 | 12021002002080 |
| Dibyansh Gupta | CSE - 2A - C1 | 12 | 12021002002023 |
| Akashdip Mazumder | CSE - 2A - C1 | 10 | 12021002002019 |

Table of Contents

| Title | 1 |
| --- | --- |
| Software Specification | 1 |
| Theory | 1 |
| Functions | 1 |
| Diagram | 2 |
| Working Principle | 3 |
| VHDL Code | 3 |
| Simulation Result | 11 |
| Conclusion | 11 |

TITLE:

Implementation of 8-bit ALU in VHDL using structural model.

SOFTWARE SPECIFICATION:

| Property Name | Value |
| --- | --- |
| Device Family | Spartan 3 |
| Device | XC3S50 |
| Package | PQ208 |
| Speed | -5 |
| Top-level source type | HDL |
| Synthesis Tool | XST (VHDL/Verilog) |
| Simulator | ISim (VHDL/Verilog) |
| Preferred Language | VHDL |

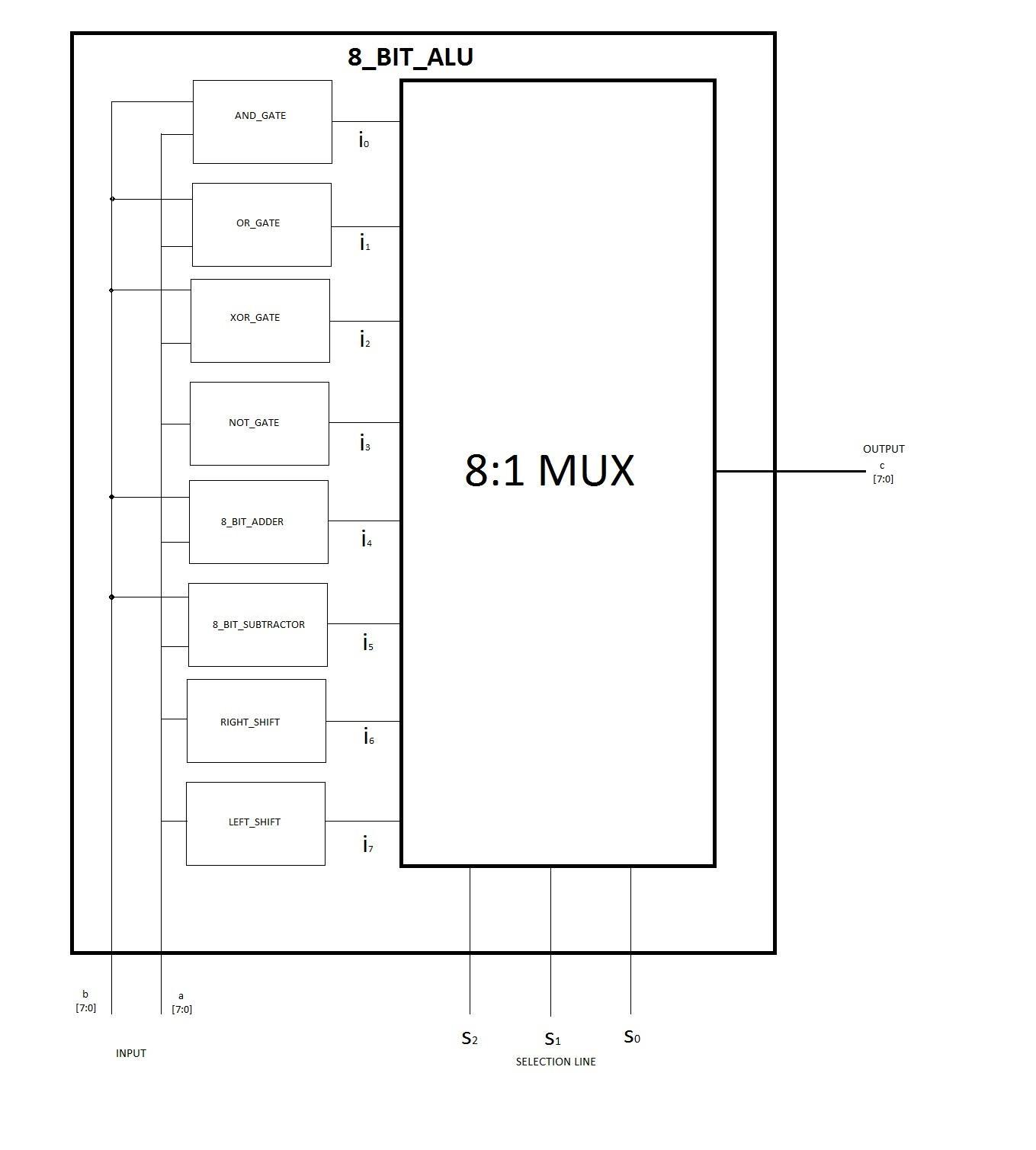
THEORY:

The ALU (arithmetic logic unit) contains elements for performing arithmetic and logical computations on data values, based on the input received from the control unit. It typically contains an adder, multiplier, and has units to compute logical bitwise operations. It has three parallel data buses consisting of two input operands (A and B) and a result output (C). Each data bus is a group of signals that conveys one binary integer number. Typically, the A, B and C bus widths (the number of signals comprising each bus) are identical and match the native word size of the external circuitry (e.g., the encapsulating CPU or other processor).

FUNCTIONS:

1. Bitwise AND: It outputs a binary number that has a 1 in each bit position for which the corresponding bits of both operands are 1, and a 0 elsewhere.
2. Bitwise OR: It outputs a binary number that has a 1 in each bit position for which the corresponding bits of any of the operands is 1, and a 0 elsewhere.
3. Bitwise XOR: It outputs a binary number that has a 1 in each bit position for which the corresponding bits of exactly one of the operands is 1, and a 0 elsewhere.
4. Bitwise NOT: It outputs a binary number that has a 1 in each bit position for which the corresponding bit of the single operand is 0, and a 0 elsewhere.
5. Adder: It outputs the sum of the two signed binary operands, discarding the carry out bit to maintain the word size. Alternatively, it can be stored in the overflow flags register.
6. Subtractor: It outputs the difference of the two signed binary operands.
7. Right Shift: It outputs the binary number obtained by shifting each input bit one place to the right, discarding the least significant bit and copying the input sign bit to the most significant bit.
8. Left Shift: It outputs the binary number obtained by shifting each input bit one place to the left, discarding the most significant bit and writing a 0 to the least significant bit.

DIAGRAM:



WORKING PRINCIPLE:

The two input vectors A and B (or only A in case of unary operations) are fed to each of the eight components implementing the arithmetic and logic operations. The output of these components are fed to the data lines of a 8:1 multiplexer which selects one of these based on the input vector S fed to its select line. The output of the multiplexer corresponds to the final output of the ALU, namely C.

VHDL CODE:

1. ALU.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity ALU is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  S: in std\_logic\_vector (2 downto 0);  C: out std\_logic\_vector (7 downto 0));  end ALU;  architecture Structural of ALU is  signal D,E,F,G,H,I,J,K: std\_logic\_vector (7 downto 0);  component and\_gate is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end component;  component or\_gate is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end component;  component xor\_gate is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end component;  component not\_gate is  port(  A: in std\_logic\_vector (7 downto 0);  B: out std\_logic\_vector (7 downto 0));  end component;  component adder is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end component;  component subtractor is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end component;  component right\_shift is  port(  A: in std\_logic\_vector (7 downto 0);  B: out std\_logic\_vector (7 downto 0));  end component;  component left\_shift is  port(  A: in std\_logic\_vector (7 downto 0);  B: out std\_logic\_vector (7 downto 0));  end component;  component MUX is  port(  D0: in std\_logic\_vector (7 downto 0);  D1: in std\_logic\_vector (7 downto 0);  D2: in std\_logic\_vector (7 downto 0);  D3: in std\_logic\_vector (7 downto 0);  D4: in std\_logic\_vector (7 downto 0);  D5: in std\_logic\_vector (7 downto 0);  D6: in std\_logic\_vector (7 downto 0);  D7: in std\_logic\_vector (7 downto 0);  S: in std\_logic\_vector (2 downto 0);  Q: out std\_logic\_vector (7 downto 0));  end component;  begin  v0: and\_gate port map(A,B,D);  v1: or\_gate port map(A,B,E);  v2: xor\_gate port map(A,B,F);  v3: not\_gate port map(A,G);  v4: adder port map(A,B,H);  v5: subtractor port map(A,B,I);  v6: right\_shift port map(A,J);  v7: left\_shift port map(A,K);  v8: MUX port map(D,E,F,G,H,I,J,K,S,C);  end Structural; |
| --- |

2. testbench.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;    entity testbench is  end testbench;  architecture tb of testbench is  component ALU is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  S: in std\_logic\_vector (2 downto 0);  C: out std\_logic\_vector (7 downto 0));  end component;  signal A\_in, B\_in: std\_logic\_vector (7 downto 0);  signal S\_in: std\_logic\_vector (2 downto 0);  signal C\_out: std\_logic\_vector (7 downto 0);  begin  DUT: ALU port map(A\_in, B\_in, S\_in, C\_out);  process  begin  A\_in <= "00101100";  B\_in <= "01001001";  S\_in <= "000";  wait for 1 ns;  assert(C\_out = "00001000") report "Fail 000" severity error;  A\_in <= "00101100";  B\_in <= "01001001";  S\_in <= "001";  wait for 1 ns;  assert(C\_out = "01101101") report "Fail 001" severity error;    A\_in <= "00101100";  B\_in <= "01001001";  S\_in <= "010";  wait for 1 ns;  assert(C\_out = "01100101") report "Fail 010" severity error;    A\_in <= "00101100";  B\_in <= "01001001";  S\_in <= "011";  wait for 1 ns;  assert(C\_out = "11010011") report "Fail 011" severity error;    A\_in <= "00101100";  B\_in <= "01001001";  S\_in <= "100";  wait for 1 ns;  assert(C\_out = "01110101") report "Fail 100" severity error;    A\_in <= "00101100";  B\_in <= "01001001";  S\_in <= "101";  wait for 1 ns;  assert(C\_out = "11100011") report "Fail 101" severity error;    A\_in <= "00101100";  B\_in <= "01001001";  S\_in <= "110";  wait for 1 ns;  assert(C\_out = "00010110") report "Fail 110" severity error;    A\_in <= "00101100";  B\_in <= "01001001";  S\_in <= "111";  wait for 1 ns;  assert(C\_out = "01011000") report "Fail 111" severity error;    A\_in <= "00000000";  B\_in <= "00000000";  S\_in <= "000";  wait;  end process;  end tb; |
| --- |

3. mux.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity MUX is  port(  D0: in std\_logic\_vector (7 downto 0);  D1: in std\_logic\_vector (7 downto 0);  D2: in std\_logic\_vector (7 downto 0);  D3: in std\_logic\_vector (7 downto 0);  D4: in std\_logic\_vector (7 downto 0);  D5: in std\_logic\_vector (7 downto 0);  D6: in std\_logic\_vector (7 downto 0);  D7: in std\_logic\_vector (7 downto 0);  S: in std\_logic\_vector (2 downto 0);  Q: out std\_logic\_vector (7 downto 0));  end MUX;  architecture Behavioral of MUX is  begin  process(D0,D1,D2,D3,D4,D5,D6,D7,S)  begin  if (S(2) = '0') then  if (S(1) = '0') then  if (S(0) = '0') then  Q <= D0;  else  Q <= D1;  end if;  else  if (S(0) = '0') then  Q <= D2;  else  Q <= D3;  end if;  end if;  else  if (S(1) = '0') then  if (S(0) = '0') then  Q <= D4;  else  Q <= D5;  end if;  else  if (S(0) = '0') then  Q <= D6;  else  Q <= D7;  end if;  end if;  end if;  end process;  end Behavioral; |
| --- |

4. and\_gate.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity and\_gate is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end and\_gate;  architecture Dataflow of and\_gate is  begin  C <= A and B;  end Dataflow; |
| --- |

5. or\_gate.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity or\_gate is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end or\_gate;  architecture Dataflow of or\_gate is  begin  C <= A or B;  end Dataflow; |
| --- |

6. xor\_gate.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity xor\_gate is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end xor\_gate;  architecture Dataflow of xor\_gate is  begin  C <= A xor B;  end Dataflow; |
| --- |

7. not\_gate.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity not\_gate is  port(  A: in std\_logic\_vector (7 downto 0);  B: out std\_logic\_vector (7 downto 0));  end not\_gate;  architecture Dataflow of not\_gate is  begin  B <= not A;  end Dataflow; |
| --- |

8. adder.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity adder is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end adder;  architecture Behavioral of adder is  begin  process(A,B)  variable u: std\_logic\_vector (8 downto 0);  begin  u(0) := '0';  for j in 0 to 7 loop  u(j+1) := (A(j) and B(j)) or (A(j) and u(j)) or (B(j) and u(j));  C(j) <= A(j) xor B(j) xor u(j);  end loop;  end process;  end Behavioral; |
| --- |

9. subtractor.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity subtractor is  port(  A: in std\_logic\_vector (7 downto 0);  B: in std\_logic\_vector (7 downto 0);  C: out std\_logic\_vector (7 downto 0));  end subtractor;  architecture Behavioral of subtractor is  begin  process(A,B)  variable u: std\_logic\_vector (8 downto 0);  begin  u(0) := '1';  for j in 0 to 7 loop  u(j+1) := (A(j) and (not B(j))) or (A(j) and u(j)) or ((not B(j)) and u(j));  C(j) <= A(j) xor (not B(j)) xor u(j);  end loop;  end process;  end Behavioral; |
| --- |

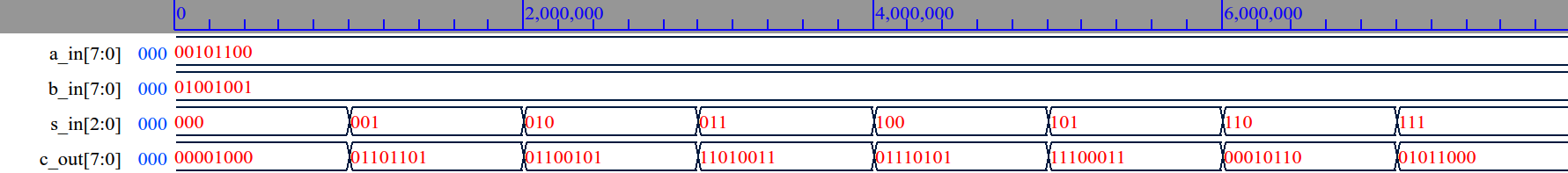
10. right\_shift.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity right\_shift is  port(  A: in std\_logic\_vector (7 downto 0);  B: out std\_logic\_vector (7 downto 0));  end right\_shift;  architecture Behavioral of right\_shift is  begin  process(A)  begin  for j in 0 to 6 loop  B(j) <= A(j+1);  end loop;  B(7) <= A(7);  end process;  end Behavioral; |
| --- |

11. left\_shift.vhd

| library IEEE;  use IEEE.std\_logic\_1164.all;  entity left\_shift is  port(  A: in std\_logic\_vector (7 downto 0);  B: out std\_logic\_vector (7 downto 0));  end left\_shift;  architecture Behavioral of left\_shift is  begin  process(A)  begin  for j in 6 downto 0 loop  B(j+1) <= A(j);  end loop;  B(0) <= '0';  end process;  end Behavioral; |
| --- |

SIMULATION RESULT:



CONCLUSION:

* The ALU can be made more sophisticated by adding compare and division instructions.
* The carry out signals of adder and subtractor can be stored separately in the register file, and used to check overflow.
* We additionally learned how to execute VHDL code in terminal, with only a text editor, GHDL compiler and a waveform viewer. We observed that the process can be broken down into three stages: analysis (generation of object file), elaboration (creation of executable) and finally run to generate the required waveform.