Toronto Metropolitan University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Computer Organization and Architecture
Course Number:	COE 608
Semester/Year (e.g. F2017)	W2023

Instructor	Demetres Kostas

Assignment/Lab Number:	3
Assignment/Lab Title:	32-bit ALU Design

Submission Date:	Feb 1, 2023
Due Date:	Feb 15, 2023

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Hernandez	Spencer	501033228	11	Spencer H
Dumlao	Carlo	501018239	11	C.D

Table of Contents

1.0 32-Bit ALU

1.1 Description

The main component that is used to perform arithmetic and bitwise operations on two 32-bit numbers.

1.2 VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity alu is
port(
        a      :      in     std_logic_vector(31 downto 0) := (others => '0');
        b      :      in     std_logic_vector(31 downto 0) := (others => '0');
        op     :      in     std_logic_vector(2 downto 0);
        result :      out    std_logic_vector(31 downto 0);
        cout   :      out    std_logic;
        zero   :      out    std_logic);
end alu;

architecture description of alu is
        component rippleAdder32
                port (
                        a            :      in           std_logic_vector(31 downto 0);
                        b            :      in           std_logic_vector(31 downto 0);
                        cin    :      in     std_logic;
                        cout   :      out    std_logic;
                        s            :      out    std_logic_vector(31 downto 0)
                );
        end component;
        component mux2to1
                port (
                        s      :      in     std_logic;
                        w0, w1 :      in     std_logic_vector(31 downto 0);
                        f      :      out    std_logic_vector(31 downto 0)
                );
        end component;
        component mux8to1
                port (
                        s      :      in     std_logic_vector(2 downto 0);
                        w0     :      in     std_logic_vector(31 downto 0);
                        w1 :   in     std_logic_vector(31 downto 0);
                        w2 :   in     std_logic_vector(31 downto 0);
                w3 :  in    std_logic_vector(31 downto 0);
                w4 :  in    std_logic_vector(31 downto 0);
                w5 :  in    std_logic_vector(31 downto 0);
                w6 :  in    std_logic_vector(31 downto 0);
                w7 :  in    std_logic_vector(31 downto 0);
                f     :     out   std_logic_vector(31 downto 0)
            );
    end component;
    component shiftLeft
            port (
                w     :     in    std_logic_vector(31 downto 0);
                f     :     out   std_logic_vector(31 downto 0)
            );
    end component;
    component shiftRight
            port (
                w     :     in    std_logic_vector(31 downto 0);
                f     :     out   std_logic_vector(31 downto 0)
            );
    end component;
    component andOp
            port (
                a     :     in    std_logic_vector(31 downto 0);
                b     :     in    std_logic_vector(31 downto 0);
                f     :     out   std_logic_vector(31 downto 0)
            );
    end component;
    component orOp
            port (
                a     :     in    std_logic_vector(31 downto 0);
                b     :     in    std_logic_vector(31 downto 0);
                f     :     out   std_logic_vector(31 downto 0)
            );
    end component;
    component zeroDetect
            port (
                w     :     in    std_logic_vector(31 downto 0);
                f     :     out   std_logic
            );
    end component;
    component carryDetect
            port (
                      a     :     in    std_logic;
                      b     :     in    std_logic;
                      f     : out std_logic
            );
    end component;

    signal c_out : std_logic;
    signal z_out : std_logic;
    signal add_carry : std_logic;
    signal gnd : std_logic_vector(31 downto 0):= (others => '0');
```

```
        signal b_not : std_logic_vector(31 downto 0);
        signal add_out : std_logic_vector(31 downto 0);
        signal and_out : std_logic_vector(31 downto 0);
        signal or_out : std_logic_vector(31 downto 0);
        signal rol_out : std_logic_vector(31 downto 0);
        signal ror_out : std_logic_vector(31 downto 0);
        signal mux2to1_out : std_logic_vector(31 downto 0);
        signal mux8to1_out : std_logic_vector(31 downto 0);
begin
        and32 : andOp port map(a, b, and_out);
        or32 : orOp port map(a, b, or_out);

        b_not <= not b;
        smallMux : mux2to1 port map(op(2), b, b_not, mux2to1_out);
        adder32 : rippleAdder32 port map(a, mux2to1_out, op(2), add_carry, add_out);

        logicLeft32 : shiftLeft port map(a, rol_out);
        logicRight32 : shiftRight port map(a, ror_out);

        largeMux : mux8to1 port map(op, and_out, or_out, add_out, gnd, rol_out, ror_out, add_out, gnd, mux8to1_out);
        zeroSystem : zeroDetect port map(mux8to1_out, z_out);
        carrySystem : carryDetect port map(add_carry, op(1), c_out);

        result <= mux8to1_out;
        zero <= z_out;
        cout <= c_out;

    end description;
```

1.3 Waveform

The functional and timing waveform of the 32-bit ALU are shown in Figure 1.1 and Figure 1.2, respectively. The correctness of its output waveform is briefly discussed below :

- **AND**: The 1st operation that the ALU performed (i.e. [0, 100 ns]). As expected, the zero flag is set to high during this time, as a result of the signal ‘a’ that has a current value of 0x0.
- **OR**: The 2nd operation that the ALU performed (i.e. [100ns, 200 ns]). For this time interval, the signal ‘a’ and ‘b’ have a value of 0x1 and 0x0, respectively; hence, the output of the ALU is immediately set to 0x1 forcing the zero flag to be low.
- **ADD**: The 3rd operation that the ALU performed (i.e. [200ns, 300 ns]). In this time interval, its output is 0x7, which is to be expected since the current operands are 0x4 and 0x3 (0x04+0x03 = 0x07).
- **SUB**: The 4th operation that the ALU performed (i.e. [300ns, 400 ns]). During this time interval, its output has a value of 0x1 due to the current operands of 0x4 and 0x3

(0x04-0x03 = 0x01). Notice that the operation and these operands have set the carry out of the ALU to be high.

- **ROL**: The 4th operation that the ALU performed (i.e. [400ns, 500 ns]). During this operation, the ‘1’ bit at the least significant bit of signal ‘a’ has been shifted to the left, resulting in an output value of 0x2.
- **ROR**:The last operation that the ALU performed (i.e. [500ns, 600 ns]). In this time interval, the value 0x2 from the input signal has been shifted to the right, resulting in an output value of 0x1.

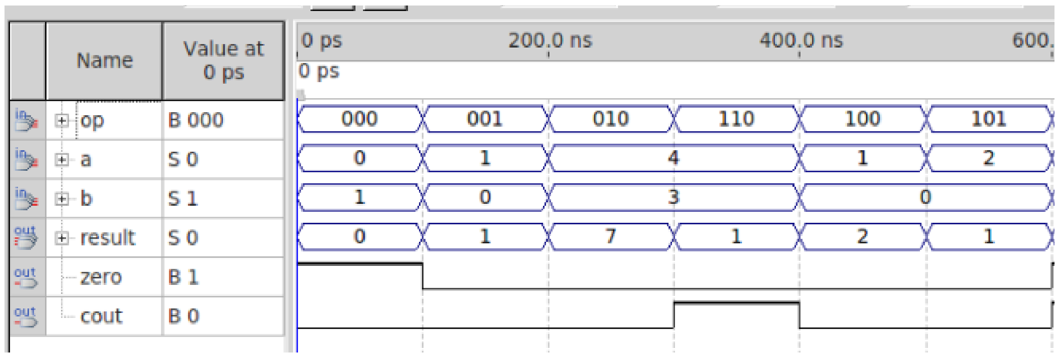


Figure 1.1: Functional waveform of the 32-bit ALU.

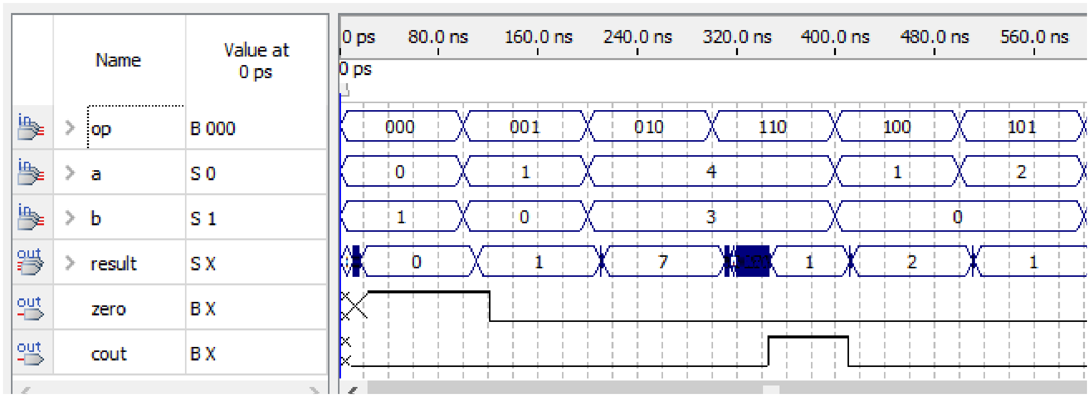


Figure 1.2: Timing waveform of the 32-bit ALU.

2.0 Full Adder

2.1 Description

Performs the addition of three binary inputs, which is an essential component of the ripple adder.

2.2 VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity fullAdder is
port(
        a           :     in          std_logic;
        b           :     in          std_logic;
        cin    :     in     std_logic;
        cout   :     out    std_logic;
        s           :     out    std_logic
        );
end fullAdder;

architecture description of fullAdder is
begin
        s <= a xor b xor cin;
        cout <= ((a xor b) and cin) or (a and b);
end description;
```

3.0 32-bit Ripple Adder

3.1 Description

A complex component composed of multiple full adders that is capable of adding and subtracting two 32-bit signals. This ripple adder is always used whenever the ALU wants to perform an ADD or SUB operation.

3.2 VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity rippleAdder32 is
port(
        a           :     in          std_logic_vector(31 downto 0);
        b           :     in          std_logic_vector(31 downto 0);
        cin    :     in     std_logic;
        cout   :     out    std_logic;
        s           :     out    std_logic_vector(31 downto 0)
        );
end rippleAdder32;

architecture description of rippleAdder32 is
        component fullAdder
              port (
                    a           :     in          std_logic;
```

```
                        b           :       in            std_logic;
                        cin     :       in      std_logic;
                        cout    :       out     std_logic;
                        s           :       out     std_logic
                );
        end component;
        signal c : std_logic_vector(31 downto 0);
begin
        add0 : fullAdder port map(a(0), b(0), cin, c(0), s(0));
        add1 : fullAdder port map(a(1), b(1), c(0), c(1), s(1));
        add2 : fullAdder port map(a(2), b(2), c(1), c(2), s(2));
        add3 : fullAdder port map(a(3), b(3), c(2), c(3), s(3));
        add4 : fullAdder port map(a(4), b(4), c(3), c(4), s(4));
        add5 : fullAdder port map(a(5), b(5), c(4), c(5), s(5));
        add6 : fullAdder port map(a(6), b(6), c(5), c(6), s(6));
        add7 : fullAdder port map(a(7), b(7), c(6), c(7), s(7));
        add8 : fullAdder port map(a(8), b(8), c(7), c(8), s(8));
        add9 : fullAdder port map(a(9), b(9), c(8), c(9), s(9));
        add10 : fullAdder port map(a(10), b(10), c(9), c(10), s(10));
        add11 : fullAdder port map(a(11), b(11), c(10), c(11), s(11));
        add12 : fullAdder port map(a(12), b(12), c(11), c(12), s(12));
        add13 : fullAdder port map(a(13), b(13), c(12), c(13), s(13));
        add14 : fullAdder port map(a(14), b(14), c(13), c(14), s(14));
        add15 : fullAdder port map(a(15), b(15), c(14), c(15), s(15));
        add16 : fullAdder port map(a(16), b(16), c(15), c(16), s(16));
        add17 : fullAdder port map(a(17), b(17), c(16), c(17), s(17));
        add18 : fullAdder port map(a(18), b(18), c(17), c(18), s(18));
        add19 : fullAdder port map(a(19), b(19), c(18), c(19), s(19));
        add20 : fullAdder port map(a(20), b(20), c(19), c(20), s(20));
        add21 : fullAdder port map(a(21), b(21), c(20), c(21), s(21));
        add22 : fullAdder port map(a(22), b(22), c(21), c(22), s(22));
        add23 : fullAdder port map(a(23), b(23), c(22), c(23), s(23));
        add24 : fullAdder port map(a(24), b(24), c(23), c(24), s(24));
        add25 : fullAdder port map(a(25), b(25), c(24), c(25), s(25));
        add26 : fullAdder port map(a(26), b(26), c(25), c(26), s(26));
        add27 : fullAdder port map(a(27), b(27), c(26), c(27), s(27));
        add28 : fullAdder port map(a(28), b(28), c(27), c(28), s(28));
        add29 : fullAdder port map(a(29), b(29), c(28), c(29), s(29));
        add30 : fullAdder port map(a(30), b(30), c(29), c(30), s(30));
        add31 : fullAdder port map(a(31), b(31), c(30), c(31), s(31));
        cout <= c(31);
end description;
```

4.0 Multiplexer 2:1

4.1 Description

A small multiplexer that is used to invert a 32-bit signal when the ripple adder is operating in a subtraction mode.

4.2 VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2to1 is
port   (
        s      :      in     std_logic;
        w0, w1 :      in     std_logic_vector(31 downto 0);
        f      :      out    std_logic_vector(31 downto 0)
        );
end mux2to1;

architecture Behavior of mux2to1 is
begin
        with s select
                f <= w0 when '0',
                     w1 when others;
end Behavior;
```

5.0 Multiplexer 8:1

5.1 Description

A big multiplexer that chooses the operation to be executed by the ALU, based on the 3-bit signal ALU-Op.

5.2 VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity mux8to1 is

port   (
        s      :      in     std_logic_vector(2 downto 0);
        w0     :      in     std_logic_vector(31 downto 0);
        w1 :   in     std_logic_vector(31 downto 0);
        w2 :   in     std_logic_vector(31 downto 0);
        w3 :   in     std_logic_vector(31 downto 0);
        w4 :   in     std_logic_vector(31 downto 0);
        w5 :   in     std_logic_vector(31 downto 0);
        w6 :   in     std_logic_vector(31 downto 0);
        w7 :   in     std_logic_vector(31 downto 0);
        f      :      out    std_logic_vector(31 downto 0)
        );
end mux8to1;

```

```
architecture Behavior of mux8to1 is
begin
        with s select
                f <= w0 when "000",
                      w1 when "001",
                          w2 when "010",
                          w3 when "011",
                          w4 when "100",
                          w5 when "101",
                          w6 when "110",
                          w7 when "111";
end Behavior;
```

6.0 Shift Left Logical

6.1 Description

Moves a 32-bit data to the left and replaces its least significant bit by a zero bit. This is used as a ROL operation for the ALU.

6.2 VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity shiftLeft is
port   (
        w      :      in     std_logic_vector(31 downto 0);
        f      :      out    std_logic_vector(31 downto 0)
        );
end shiftLeft;

architecture Behavior of shiftLeft is
begin
        f <= w(30 downto 0) & '0';
end Behavior;
```

7.0 Shift Right Logical

7.1 Description

Moves a 32-bit data to the right and replaces its most significant bit by a zero bit. This is used as a ROR operation for the ALU.

7.2 VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity shiftRight is
port  (
        w     :     in    std_logic_vector(31 downto 0);
        f     :     out   std_logic_vector(31 downto 0)
        );
end shiftRight;

architecture Behavior of shiftRight is
begin
        f <= '0' & w(31 downto 1);
end Behavior;
```

8.0 32-Bit AND Operator

8.1 Description

Performs an AND operation between two 32-bit signals for the ALU.

8.2 VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity andOp is
port  (
        a     :     in    std_logic_vector(31 downto 0);
        b     :     in    std_logic_vector(31 downto 0);
        f     :     out   std_logic_vector(31 downto 0)
        );
end andOp;

architecture Behavior of andOp is
begin
        f <= a and b;
end Behavior;
```

9.0 32-Bit OR Operator

9.1 Description

Performs an OR operation between two 32-bit signals for the ALU.

9.2 VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity orOp is
port   (
        a      :      in     std_logic_vector(31 downto 0);
        b      :      in     std_logic_vector(31 downto 0);
        f      :      out    std_logic_vector(31 downto 0)
        );
end orOp;

architecture Behavior of orOp is
begin
        f <= a or b;
end Behavior;
```

10.0 Zero Detector

10.1 Description

A component that updates the zero flag based on the output of the 32-bit ALU. It is set to low if the output is a non-zero value; otherwise, the flag is set to high.

10.2 VHDL Code

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity zeroDetect is
port   (
        w      :      in     std_logic_vector(31 downto 0);
        f      :      out    std_logic
        );
end zeroDetect;

architecture Behavior of zeroDetect is
begin
        f <= not(w(0) or w(1) or w(2) or w(3) or w(4) or w(5)
                    or w(6) or w(7) or w(8) or w(9) or w(10)
                    or w(11) or w(12) or w(13) or w(14) or w(15)
                    or w(16) or w(17) or w(18) or w(19) or w(20)
                    or w(21) or w(22) or w(23) or w(24) or w(25)
                    or w(26) or w(27) or w(28) or w(29) or w(30)
                    or w(31));
end Behavior;
```

11.0 Carry Detector

11.1 Description

A component that updates the carry flag depending on the output carry of the ripple adder, as well as the current operation of the ALU. Specifically, when the ALU is adding or subtracting, then the flag only depends on the output carry of the adder; otherwise, the flag is always set to low.

11.2 VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity carryDetect is
port   (
        a      :      in     std_logic;
        b      :      in     std_logic;
        f      :      out std_logic
        );
end carryDetect;

architecture Behavior of carryDetect is
begin
        f <= a and b;
end Behavior;
```

12.0 Decoder 4:7

12.1 Description

A decoder that is used to control a single 7-segment display, based on the incoming 4-bit data of the ALU.

12.2 VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder is
port   (
        s      :      in     std_logic_vector(3 downto 0);
        f      :      out    std_logic_vector(6 downto 0)
        );
end decoder;
```

```

architecture Behavior of decoder is
        signal f_out : std_logic_vector(6 downto 0);
begin
        with s select
                f_out <= "1111110" when "0000",
                      "0110000" when "0001",
                          "1101101" when "0010",
                          "1111001" when "0011",
                          "0110011" when "0100",
                          "1011011" when "0101",
                          "1011111" when "0110",
                          "1110000" when "0111",
                          "1111111" when "1000",
                          "1111011" when "1001",
                          "1110111" when "1010",
                          "0011111" when "1011",
                          "1001110" when "1100",
                          "0111101" when "1101",
                          "1001111" when "1110",
                          "1000111" when "1111";
                f <= not f_out;
end Behavior;
```

13.0 32-Bit ALU + Decoder 4:7

13.1 Description

The complete system that is used to display the 32-bit output of the ALU in the 8 digit 7-segment LED display.

13.2 VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity alu_7seg is
port(
        a       :       in      std_logic_vector(31 downto 0);
        b       :       in      std_logic_vector(31 downto 0);
        op      :       in      std_logic_vector(2 downto 0);
        r0      :       out     std_logic_vector(6 downto 0);
        r1      :       out     std_logic_vector(6 downto 0);
        r2      :       out     std_logic_vector(6 downto 0);
        r3      :       out     std_logic_vector(6 downto 0);
        r4      :       out     std_logic_vector(6 downto 0);
        r5 :  out     std_logic_vector(6 downto 0);
```

```
      r6 :  out    std_logic_vector(6 downto 0);
      r7 :  out    std_logic_vector(6 downto 0);
      cout   :     out    std_logic;
      zero   :     out    std_logic);
end alu_7seg;

architecture description of alu_7seg is
      component    alu
             port (
             a     :      in     std_logic_vector(31 downto 0) := (others => '0');
             b     :      in     std_logic_vector(31 downto 0) := (others => '0');
             op    :      in     std_logic_vector(2 downto 0);
             result:      out    std_logic_vector(31 downto 0);
             cout  :      out    std_logic;
             zero  :      out    std_logic
             );
      end component;

      component    decoder
             port (
                   s      :      in     std_logic_vector(3 downto 0);
                   f      :      out    std_logic_vector(6 downto 0)
             );
      end component;

      signal result : std_logic_vector(31 downto 0);
begin
      alu32 : alu port map(a, b, op, result, cout, zero);
      d0 : decoder port map(result(3 downto 0), r0);
      d1 : decoder port map(result(7 downto 4), r1);
      d2 : decoder port map(result(11 downto 8), r2);
      d3 : decoder port map(result(15 downto 12), r3);
      d4 : decoder port map(result(19 downto 16), r4);
      d5 : decoder port map(result(23 downto 20), r5);
      d6 : decoder port map(result(27 downto 24), r6);
      d7 : decoder port map(result(31 downto 28), r7);

end description;
```

13.3 Diagram

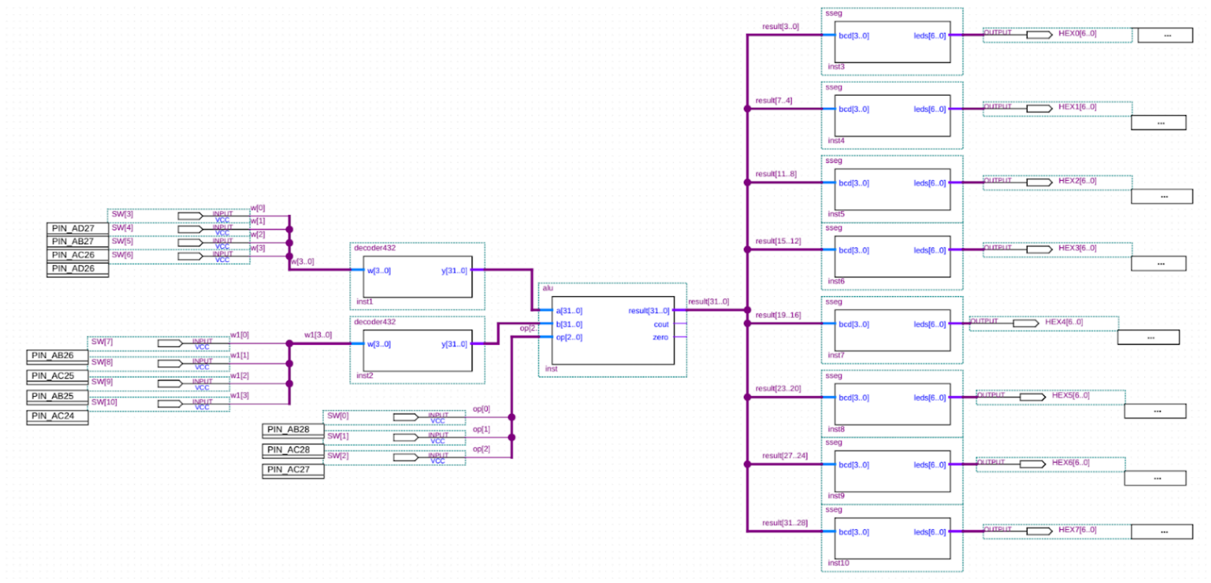
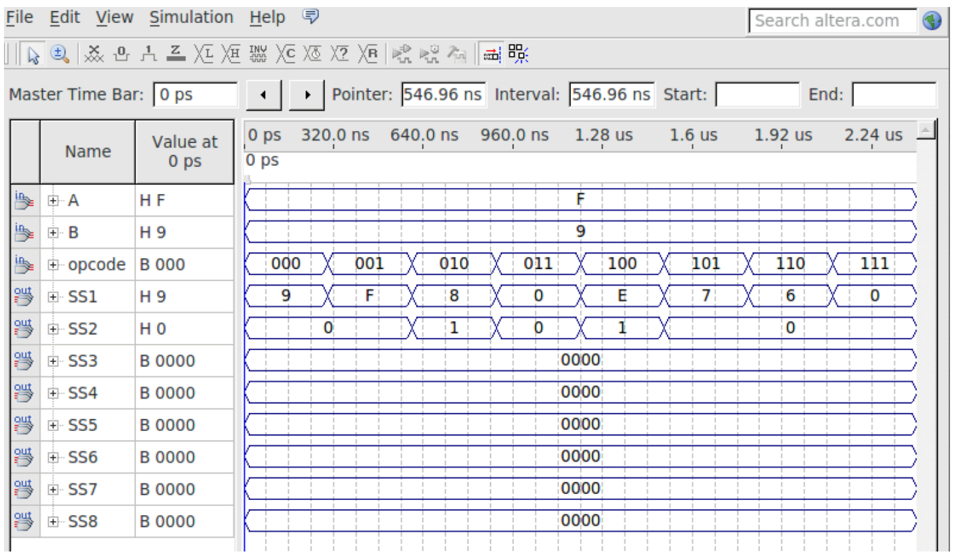


Figure 13.1: Schematic of the 32-bit ALU, along with the 4 to 7 decoders.

13.4 Waveform

` Its functional and timing waveform are shown in Figure 13.2 and Figure 13.3, respectively. Notice that each decoder is able to receive a unique 4-bit data from the 32-bit ALU. When the ALU output is 0x00000018 (i.e. during the time interval [600, 900ns]), it is evident that the decoder ‘S0’ holds the first 4-bit (0x8) while ‘S1’ holds the second 4-bit (0x1); and since the next bits of the output are all zero, the rest of the decoders are also set to a value of 0x0.



Master Time Bar: 0 ps Pointer: 546.96 ns Interval: 546.96 ns Start: End:

Name	Value at 0 ps
A	H F
B	H 9
opcode	B 000
SS1	H 9
SS2	H 0
SS3	B 0000
SS4	B 0000
SS5	B 0000
SS6	B 0000
SS7	B 0000
SS8	B 0000

Figure 13.2: Functional waveform of the 32-Bit ALU + Decoder 4:7.

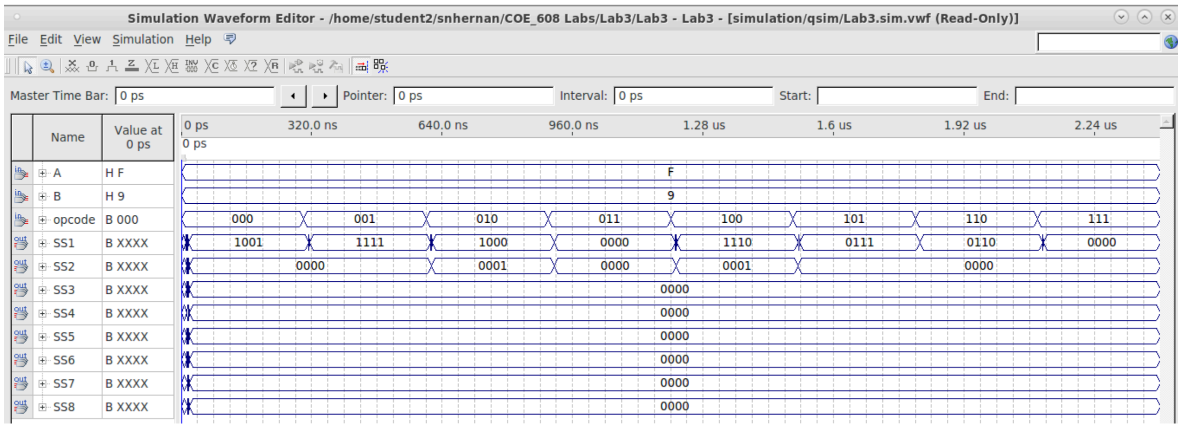

Simulation Waveform Editor - /home/student2/snhernan/COE_608 Labs/Lab3/Lab3 - Lab3 - [simulation/qsim/Lab3.sim.vwf (Read-Only)]
File Edit View Simulation Help
Master Time Bar: 0 ps
Pointer: 0 ps
Interval: 0 ps
Start:
End:
Name
Value at 0 ps
0 ps
320.0 ns
640.0 ns
960.0 ns
1.28 us
1.6 us
1.92 us
2.24 us
A H F F
B H 9 9
opcode B 000 000 001 010 011 100 101 110 111
SS1 B XXXX 1001 1111 1000 0000 1110 0111 0110 0000
SS2 B XXXX 0000 0001 0000 0001 0000
SS3 B XXXX 0000
SS4 B XXXX 0000
SS5 B XXXX 0000
SS6 B XXXX 0000
SS7 B XXXX 0000
SS8 B XXXX 0000

Figure 13.3: Timing waveform of the 32-Bit ALU + Decoder 4:7.

14.0 Discussion

Depending on the different opcodes the ALU is capable of performing bitwise and arithmetic operations as observed from the functional diagram. Upon looking at the timing diagram Figure 1.2 there appears to be a bigger dealer upon the addition function and the subtraction function that it does for bitwise operations such as the AND and OR function. This stems from the fact that it was implemented using a ripple carry adder. In contrast to the look ahead adder, the ripple carry adder carries more propagation delay proportional to the size of the number in bits. Due to a finite amount of time to perform each carry the propagation delay increases as the number of time it takes to carry occurs consecutively instead of instantaneously.

In Figure 14.1 below, notice that the time delay depends on the value of the operands – the larger the binary, the greater instability the ALU will experience. Taking this into consideration, the worst case path T_{WC} of the designed ALU can be approximated by setting all bits of signal ‘a’ to one bit and allowing signal ‘b’ to have a value of 0x1. This will ensure that all of the full adders within the ripple adder are triggered by a carry, when the ALU is performing an ADD operation. By inspection, the T_{WC} is around 50ns or $f_{WC} \approx 20MHz$. The CPU frequency should not go below this frequency; otherwise, the ALU will likely output unwanted data.

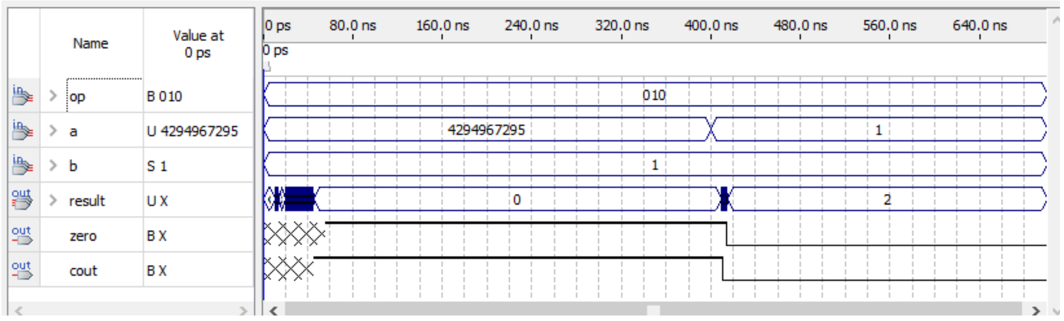


Figure 14.1: Timing waveform showing the effect of adding a large number.