University of Guelph

ENGG*2410: Digital Design

Data Path Design “Arithmetic Logic Unit” via VHDL

Group (# 8)

Layal Alzaydi (1041786)

Josiah Varghese (1043468)

Giuliana Lowry (1013417)

Mazen Hassan (0984322)

Problem Statement

The problem is to design a shared operation unit, the arithmetic/logic unit (ALU), that performs data-processing operations with a control unit to determine the sequence of those operations. The group has to design and implement the single stage of the arithmetic unit and a single stage logic unit in VHDL. Using the units, they have to create a one stage ALU and use it as a component to finally create a complete 4-bit ALU in VHDL. To test the results map it on the NEXYS 3 FPGA board and connect a 7-segment decoder to the output of the ALU to show several operations. The results should match the outputs given in the lab.

Assumptions and Constraints

Some of the assumptions and constraints used in this lab are as follows:

- Use only VHDL to implement the design
- Use structural VDL to connect components

System Overview & Justification of Design

The purpose of the lab is to understand and implement the ALU of a simple CPU with the knowledge of modular and hierarchical design. The ALU is used to perform arithmetic and logic operations, in this lab the group has to create the ALU by combining the two modular subsystems that performs a modular task: logic unit outputs the result from an operation (AND, OR, XOR, NOT) of A and B and the arithmetic unit outputs the arithmetic sum G = X + Y + C (carry in). The outputs of the arithmetic and logic units are given to the 2 to 1 MUX and based on the select line to the MUX it will choose either the arithmetic unit or the logic unit. To create a 4-bit ALU, the design uses the one stage ALU as a component to create four single stage ALU.

The 4-bit ALU is designed using only VHDL using a modular, hierarchical approach. The system consists of 4 ALU components, each component is created using modular functions of the logic unit and the arithmetic unit. The system has three select lines S0, S1 are select lines that are both in the logic and arithmetic units to control the operations in the specific unit, S2, which is the select line to the 2 to 1 MUX, controls which output to choose and display and the carry in which is used to determine the arithmetic operations and has no effect on the logic operations. By using the 7-segment decoder it is possible to test the output of the ALU and demonstrate the operation for the ALU. By using a modular approach to create this type of design it breaks down the main system into subsystems which works on certain functionalities which can be easy to maintain and can be reused.

The tables below were used to test different cases for the test bench. Table 1 was used to test for the one bit Arithmetic circuit while table 2 was used to test for the 4 bit ALU. Both test benches are inserted in the appendix of this report.

Select		**Input**	G=A/B/C_{in}	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0's	G = A (transfer)	G=A+1 (increment)
0	1	B	G = A + B (add)	G=A+B+1
1	0	$\bar{B}$	G = A + $\bar{B}$	G=A+$\bar{B}$+1 (subtract)
1	1	all 1's	G = A - 1 (decrement)	G=A (transfer)

Table 1: Function Table for Arithmetic Circuit

S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	G = A	Transfer A
0	0	0	1	G = A+1	Increment A
0	0	1	0	G = A+B	Addition
0	0	1	1	G = A+B+1	Add with carry input of 1
0	1	0	0	$G = A + \bar{B}$	A plus 1's complement of B
0	1	0	1	$G = A + \bar{B} + 1$	Subtraction
0	1	1	0	G = A - 1	Decrement A
0	1	1	1	G = A	Transfer A
1	0	0	X	$G = A \wedge B$	AND
1	0	1	X	$G = A \vee B$	OR
1	1	0	X	$G = A \oplus B$	XOR
1	1	1	X	$G = \bar{A}$	NOT (1's complement)

Table 2: Function Table for ALU

Error Analysis

There were few problems when dealing with the VHDL code for the ALU. Many of the problems could be pointed towards assigning and mapping the variables for the inputs and outputs. This lab used multiple files which caused how the initial values for the overall mapping of the system but this was fixed after going through the components and the giving the proper values to each role. There was also the issue of understanding how the system worked but was resolved by going over the lab and lecture to properly understand how the modular system worked.

Appendix

VHDL Code

Seven Segment Display

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity seven_segment is
    Port ( A : in  std_logic_vector(3 downto 0);
           F : out  std_logic_vector(6 downto 0));
end entity seven_segment;
architecture dataflow of seven_segment is
begin
F <= "0000001" when A = "0000" else
          "1001111" when A = "0001" else --1
          "0010010" when A = "0010" else --2
          "0000110" when A = "0011" else --3
          "1001100" when A = "0100" else --4
          "0100100" when A = "0101" else --5
          "0100000" when A = "0110" else --6
          "0001111" when A = "0111" else --7
          "0000000" when A = "1000" else --8
          "0001100" when A = "1001" else --9
          "0001000" when A = "1010" else --A
```

```
        "1100000" when A = "1011" else --b
        "0110001" when A = "1100" else --C
        "1000010" when A = "1101" else --d
        "0110000" when A = "1110" else --E
        "0111000" when A = "1111" else --F
        "XXXXXXX";
end architecture dataflow;
```

4-Bit ALU

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY FourBitTB IS
END FourBitTB;

ARCHITECTURE behavior OF FourBitTB IS
   -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT FourBit_ALU
   PORT(
      S0 : IN  std_logic;
      S1 : IN  std_logic;
      S2 : IN  std_logic;
      A : IN  std_logic_vector(3 downto 0);
      B : IN  std_logic_vector(3 downto 0);
      Cin : IN  std_logic;
```

```
        Cout : OUT  std_logic;
        G : OUT  std_logic_vector(3 downto 0)
       );
    END COMPONENT
   --Inputs
   signal S0 : std_logic := '0';
   signal S1 : std_logic := '0';
   signal S2 : std_logic := '0';
   signal A : std_logic_vector(3 downto 0) := (others => '0');
   signal B : std_logic_vector(3 downto 0) := (others => '0');
   signal Cin : std_logic := '0';

          --Outputs
   signal Cout : std_logic;
   signal G : std_logic_vector(3 downto 0);
BEGIN
          -- Instantiate the Unit Under Test (UUT)
   uut: FourBit_ALU PORT MAP (
        S0 => S0,
        S1 => S1,
        S2 => S2,
        A => A,
        B => B,
        Cin => Cin,
```

```
      Cout => Cout,
      G => G
    );
-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
                S2 <= '0';
                S1 <= '0';
                S0 <= '0';
                Cin <= '0';
  wait for 100 ns;
                S2 <= '0';
                S1 <= '0';
                S0 <= '1';
                Cin <= '0';
  wait for 100 ns;
                A <= "0011";
                B <= "0001";
                S2 <= '0';
                S1 <= '0';
                S0 <= '1';
                Cin <= '0';
  wait for 100 ns;
```

```
                S2 <= '1';
                S1 <= '1';
                S0 <= '0';
                Cin <= 'X';
    wait for 100 ns;
    -- insert stimulus here
                S2 <= '1';
                S1 <= '0';
                S0 <= '0';
                Cin <= 'X';
    wait for 100 ns;
    wait;
  end process;
END;
```

One Bit Arithmetic

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

entity OneBitArithmetic is
    Port ( S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
           B : in  STD_LOGIC;
           A : in  STD_LOGIC;
           Ci : in  STD_LOGIC;
           Y : out  STD_LOGIC;
                              Co: out STD_LOGIC);
end OneBitArithmetic;

architecture Structural of OneBitArithmetic is

           signal Go, B1, Cout : std_logic;

           component BInputLogic is
           Port ( S0 : in STD_LOGIC;
           S1 : in STD_LOGIC;
           B : in STD_LOGIC;
           Y : out STD_LOGIC);
           end component BInputLogic ;

           component FullAdder is
           Port ( A : in  STD_LOGIC;     --A=X
          B : in  STD_LOGIC;   --Y=B
          Ci : in  STD_LOGIC;
          S : out  STD_LOGIC;  --S=G
          Co : out  STD_LOGIC);
           end component FullAdder;

begin
           BLogic0: BInputLogic port map( S0, S1, B, B1);
```

```
        fulladder0: FullAdder  port map( A, B1, Ci, Go, Cout);

        Y<= Go;
        Co<=Cout;
end Structural;
```

One Bit Logic

```
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 primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity OneStageLogic is
    Port ( S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
           A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           G : out  STD_LOGIC);
end OneStageLogic;

architecture Behavioral of OneStageLogic is

begin
        G <= (A and B) when S1='0' and S0='0' --AND

        else
                (A or B) when S1='0' and S0='1' --OR

        else
                (A xor B) when S1='1' and S0='0' --XOR
```

```
        else
                not(A) when S1='1' and S0='1' --NOT (1's complement)

        else '0';

end Behavioral;
```

B-Input Logic

```
-- Company:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity BInputLogic is
    Port ( S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Y : out  STD_LOGIC);

end BInputLogic;

architecture Behavioral of BInputLogic is

begin
        Y <= '0'
        when ( S1 = '0') and ( S0 = '0') -- 0

        else
        B when ( S1 = '0') and ( S0 = '1') --B

        else
        not(B) when ( S1 = '1') and ( S0 = '0') --not(B)

        else
        '1' when ( S1 = '1') and ( S0 = '1') --1
```

```
        else '0';

end Behavioral;
```

One Bit ALU

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

entity OneBitALU is
    Port ( Ci : in   STD_LOGIC;
           Ai : in   STD_LOGIC;
           Bi : in   STD_LOGIC;
                     S0 : in         STD_LOGIC;
                     S1 : in         STD_LOGIC;
                     S2 : in         STD_LOGIC;
                     Co : out        STD_LOGIC;
                     Gi : out  STD_LOGIC);

end OneBitALU;

architecture Behavioral of OneBitALU  is

signal mux1: std_logic;
signal mux2: std_logic;

component OneStageLogic
    Port ( S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
           A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           G : out  STD_LOGIC);
end component;

component OneBitArithmetic
    Port ( S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
           B : in  STD_LOGIC;
```

```
        A : in  STD_LOGIC;
        Ci : in  STD_LOGIC;
        Y : out  STD_LOGIC;
                        Co: out STD_LOGIC);
end component;

component mux2_1
   Port ( in1 : in  STD_LOGIC;
        in2 : in  STD_LOGIC;
        select0 : in  STD_LOGIC;
        out0 : out  STD_LOGIC);
end component;


begin
         oneBitLogic: OneStageLogic
         port map(S0,S1,Ai,Bi,mux2);
         oneBitAri: OneBitArithmetic
         port map(S0,S1,Bi,Ai,Ci,mux1,Co);
         mux0: mux2_1
         port map(mux1,mux2,S2,Gi);


end behavioral;
```

2:1 Multiplexor

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

entity mux2_1 is
   Port ( in1 : in  STD_LOGIC;
        in2 : in  STD_LOGIC;
        select0 : in  STD_LOGIC;
        out0 : out  STD_LOGIC);
end mux2_1;

architecture Behavioral of mux2_1 is
```

```
begin

        out0 <= in1 when (select0 = '0') else in2;

end Behavioral;
```

4 BIT ALU WITH 7 SEGMENT DISPLAY

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

entity FourBit_ALU is
    Port ( S0 : in  STD_LOGIC;
           S1 : in  STD_LOGIC;
           S2 : in  STD_LOGIC;
           A : in  std_logic_vector(3 downto 0);
           B : in  std_logic_vector(3 downto 0);
           Cin : in  STD_LOGIC;
           Cout : out  STD_LOGIC;
           G : out  std_logic_vector(3 downto 0);
                        F : out  std_logic_vector(6 downto 0));
end FourBit_ALU;

architecture Behavioral of FourBit_ALU is

signal Co: std_logic_vector(3 downto 0);

component OneBitALU is
    Port ( Ci : in   STD_LOGIC;
           Ai : in   STD_LOGIC;
           Bi : in   STD_LOGIC;
                        S0 : in          STD_LOGIC;
                        S1 : in          STD_LOGIC;
                        S2 : in          STD_LOGIC;
                        Co : out         STD_LOGIC;
                        Gi : out  STD_LOGIC);
```

```
end component;

component seven_segment is
   Port ( A : in  std_logic_vector(3 downto 0);
          F : out  std_logic_vector(6 downto 0));
end component;
signal v: std_logic_vector(3 downto 0);
begin

OneBitAlu0 : OneBitALU
port map (Ci => Cin ,Ai => A(0) ,Bi => B(0),S0 => S0,S1 => S1,S2 => S2,Co => Co(0),Gi =>
v(0));

OneBitAlu1 : OneBitALU
port map (Ci => Co(0) ,Ai => A(1) ,Bi => B(1),S0 => S0,S1 => S1,S2 => S2,Co => Co(1),Gi =>
v(1));


OneBitAlu2 : OneBitALU
port map (Ci => Co(1) ,Ai => A(2) ,Bi => B(2),S0 => S0,S1 => S1,S2 => S2,Co => Co(2),Gi =>
v(2));


OneBitAlu3 : OneBitALU
port map (Ci => Co(2) ,Ai => A(3) ,Bi => B(3),S0 => S0,S1 => S1,S2 => S2,Co => Cout,Gi =>
v(3));

seg:seven_segment
port map (A(0) => v(0), A(1) => v(1), A(2) => v(2),  A(3) => v(3), f(0)=>f(0), f(1)=>f(1),
f(2)=>f(2), f(3)=>f(3), f(4)=>f(4), f(5)=>f(5), f(6)=>f(6));
--Cout<= Co(3);

g <= v;

end Behavioral;
```

UCF FILE

```
//Slide switches
 NET A<0> LOC = T10;
 NET A<1> LOC = T9;
 NET A<2> LOC = V9;
 NET A<3> LOC = M8;
 NET B<0> LOC = N8;
 NET B<1> LOC = U8;
 NET B<2> LOC = V8;
 NET B<3> LOC = T5;
// Pushbutton switches
 NET S0 LOC = D9;
 NET S1 LOC = B8;
 NET S2 LOC = C4;
 NET Cin LOC = C9;
 NET Cout LOC = U16;
// 7seg digit segments
 NET f<6> LOC = T17;
 NET f<5> LOC = T18;
 NET f<4> LOC = U17;
 NET f<3> LOC = U18;
 NET f<2> LOC = M14;
 NET f<1> LOC = N14;
 NET f<0> LOC = L14;
 // NET seg<0> LOC = M13;
```

Test Benches

One Bit Arithmetic Test Bench

```
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;

ENTITY oneBitAriTB IS
END oneBitAriTB;

ARCHITECTURE behavior OF oneBitAriTB IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT OneBitArithmetic
   PORT(
        S0 : IN  std_logic;
        S1 : IN  std_logic;
        B : IN  std_logic;
        A : IN  std_logic;
        Ci : IN  std_logic;
        Y : OUT  std_logic;
        Co : OUT  std_logic
       );
   END COMPONENT;


  --Inputs
  signal S0 : std_logic := '0';
  signal S1 : std_logic := '0';
  signal B : std_logic := '0';
  signal A : std_logic := '0';
  signal Ci : std_logic := '0';

        --Outputs
  signal Y : std_logic;
  signal Co : std_logic;
  -- No clocks detected in port list. Replace <clock> below with
  -- appropriate port name


BEGIN

        -- Instantiate the Unit Under Test (UUT)
```

```
uut: OneBitArithmetic PORT MAP (
      S0 => S0,
      S1 => S1,
      B => B,
      A => A,
      Ci => Ci,
      Y => Y,
      Co => Co
    );


-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.

                S0 <= '0';
                S1 <= '0';
                Ci <= '0';


          wait for 100 ns;

                S0 <= '0';
                S1 <= '1';
                Ci <= '1';


                 wait for 100 ns;

                S0 <= '1';
                S1 <= '0';
                Ci <= '1';


                 wait for 100 ns;

                S0 <= '0';
                S1 <= '0';
```

```
                Ci <= '1';

    wait;
  end process;

END;
```

4-Bit ALU Test Bench

```
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;

ENTITY FourBitTB IS
END FourBitTB;

ARCHITECTURE behavior OF FourBitTB IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT FourBit_ALU
   PORT(
        S0 : IN  std_logic;
        S1 : IN  std_logic;
        S2 : IN  std_logic;
        A : IN  std_logic_vector(3 downto 0);
        B : IN  std_logic_vector(3 downto 0);
        Cin : IN  std_logic;
        Cout : OUT  std_logic;
        G : OUT  std_logic_vector(3 downto 0)
       );
   END COMPONENT;


  --Inputs
```

```
  signal S0 : std_logic := '0';
  signal S1 : std_logic := '0';
  signal S2 : std_logic := '0';
  signal A : std_logic_vector(3 downto 0) := "0011";
  signal B : std_logic_vector(3 downto 0) := "0100";
  signal Cin : std_logic := '0';

        --Outputs
  signal Cout : std_logic;
  signal G : std_logic_vector(3 downto 0);


BEGIN

        -- Instantiate the Unit Under Test (UUT)
  uut: FourBit_ALU PORT MAP (
       S0 => S0,
       S1 => S1,
       S2 => S2,
       A => A,
       B => B,
       Cin => Cin,
       Cout => Cout,
       G => G
     );

  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
                  S2 <= '0';
                  S1 <= '0';
                  S0 <= '0';
                  Cin <= '0';

    wait for 80 ns;

                  S2 <= '0';
                  S1 <= '0';
```

```
        S0 <= '0';
        Cin <= '1';

wait for 80 ns;

        S2 <= '0';
        S1 <= '0';
        S0 <= '1';
        Cin <= '0';

wait for 80 ns;

        S2 <= '0';
        S1 <= '0';
        S0 <= '1';
        Cin <= '1';

wait for 80 ns;

        S2 <= '0';
        S1 <= '1';
        S0 <= '0';
        Cin <= '0';

wait for 80 ns;

        S2 <= '0';
        S1 <= '1';
        S0 <= '0';
        Cin <= '1';

wait for 80 ns;

        S2 <= '0';
        S1 <= '1';
        S0 <= '1';
        Cin <= '0';

wait for 80 ns;
```

```
                S2 <= '0';
                S1 <= '1';
                S0 <= '1';
                Cin <= '1';

    wait for 80 ns;

                S2 <= '1';
                S1 <= '0';
                S0 <= '0';
                Cin <= '1';

    wait for 80 ns;

                S2 <= '1';
                S1 <= '0';
                S0 <= '1';
                Cin <= '1';

    wait for 80 ns;

                S2 <= '1';
                S1 <= '1';
                S0 <= '0';
                Cin <= '1';

    wait for 80 ns;

                S2 <= '1';
                S1 <= '1';
                S0 <= '1';
                Cin <= '1';

    wait for 80 ns;

    --wait;
  end process;
END;
```

Circuit Diagram

The 4-bit ALU is a system that provides eight arithmetic and four logic operations. This ALU is used to perform integer operations except for division which could result in fractions.

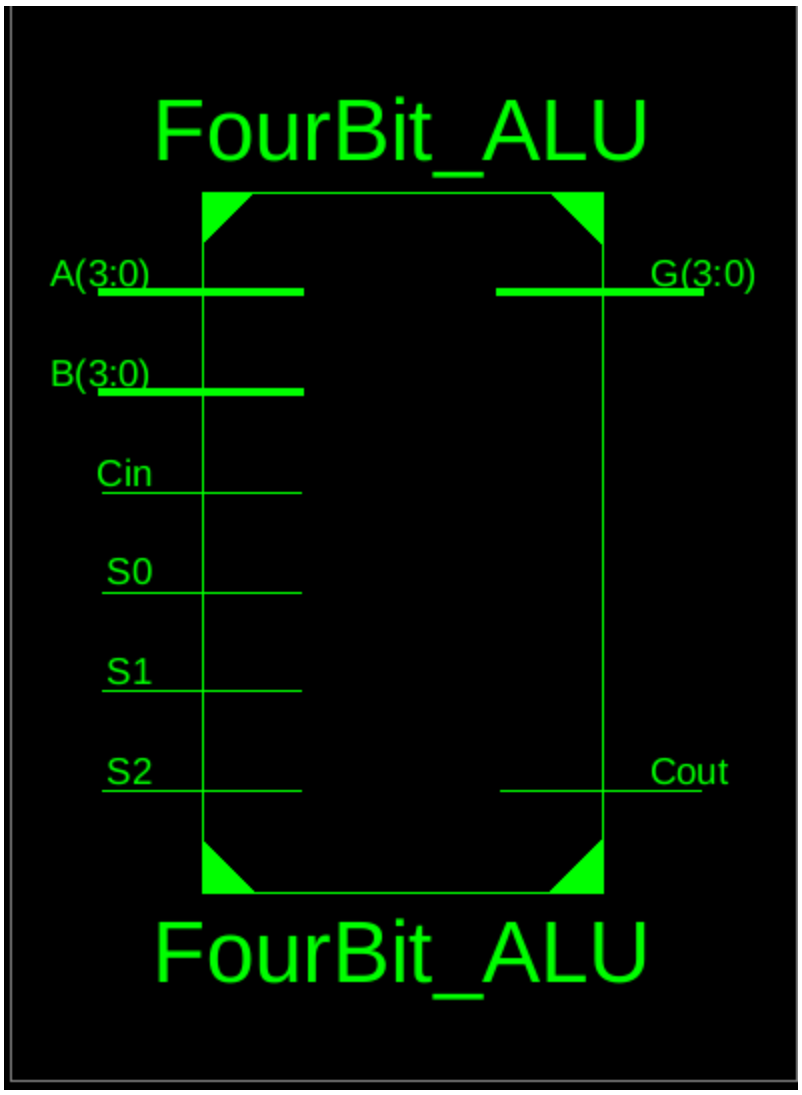

Figure 1.1 The 4-Bit ALU

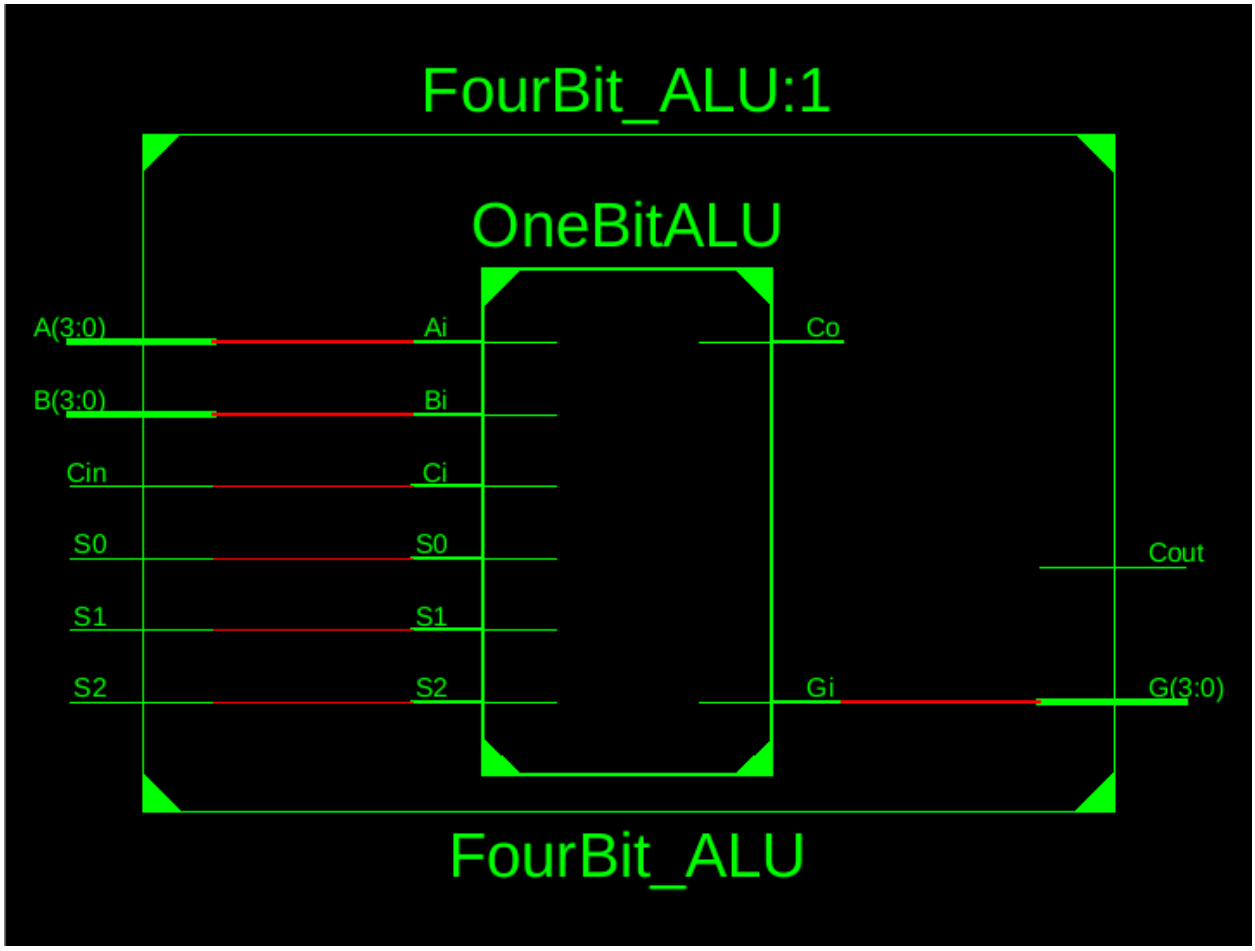

Figure 1.2 Schematic of one-bit ALU

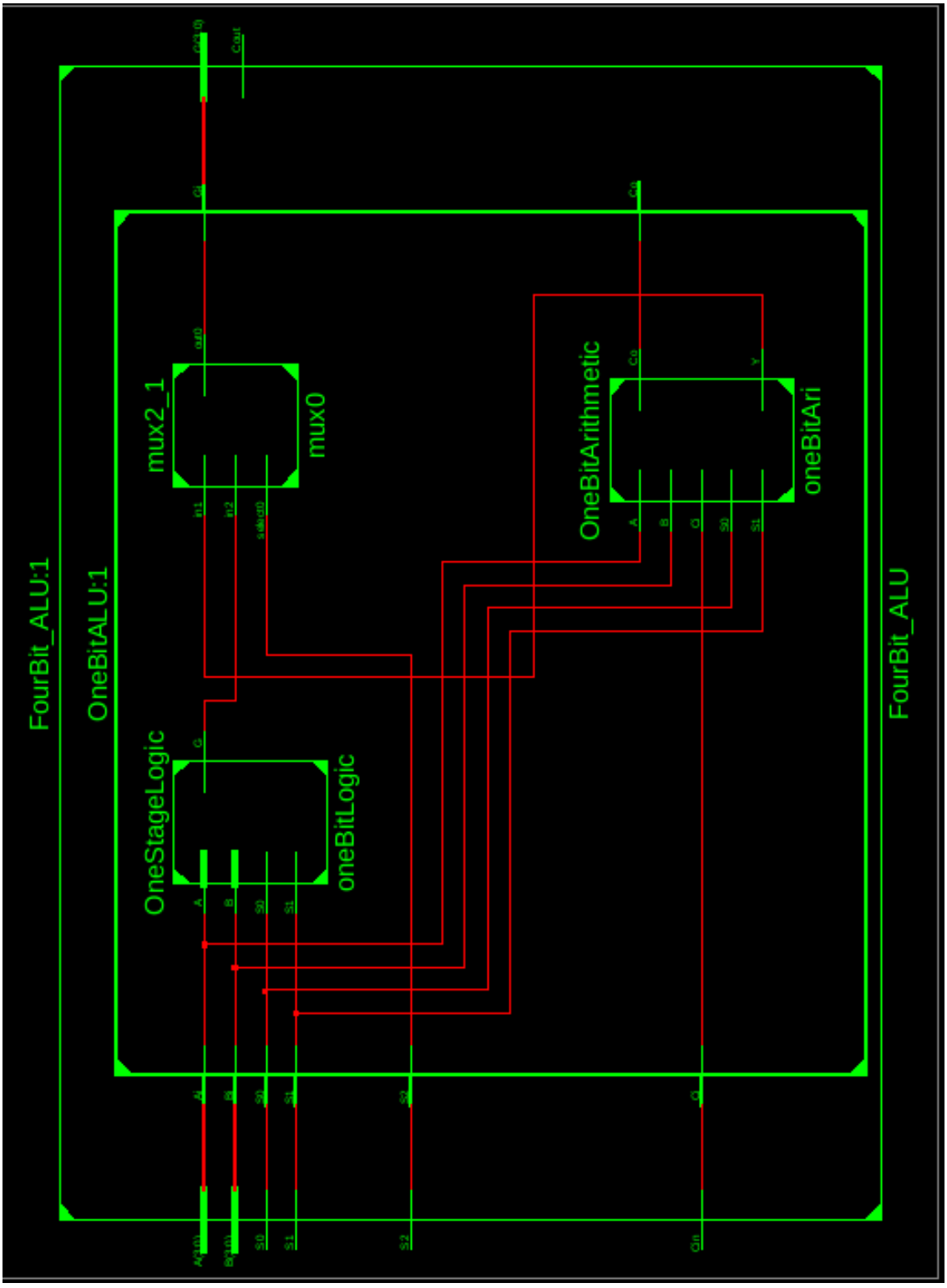

Figure 1.3 Schematic of 4-bit ALU

Simulations

Figure 2.1 Arithmetic Unit simulation

Figure 2.2a ALU simulation

Figure 2.2b ALU simulation