ECE 413: Intro to VLSI

Final Project: 2-Bit ALU

Adam Rawski & Roderick Renwick

December 16, 2019
Fall 2019

Honor Code:

I have neither given nor received unauthorized assistance on this graded report.

X_____Adam_Rawski_&_Roderick_Renwick_____

Table of Contents

Concept

This purpose of this project is to design, implement, and simulate a two-bit ALU via CMOS transistors. The following block diagram belore (figure-1) shows how to construct a two-bit ALU from two one-bit ALUs. The two-bit ALU shall be able to perform four basic operations: addition, AND, NOR, and XOR. The performed operation will be determined via the select lines.

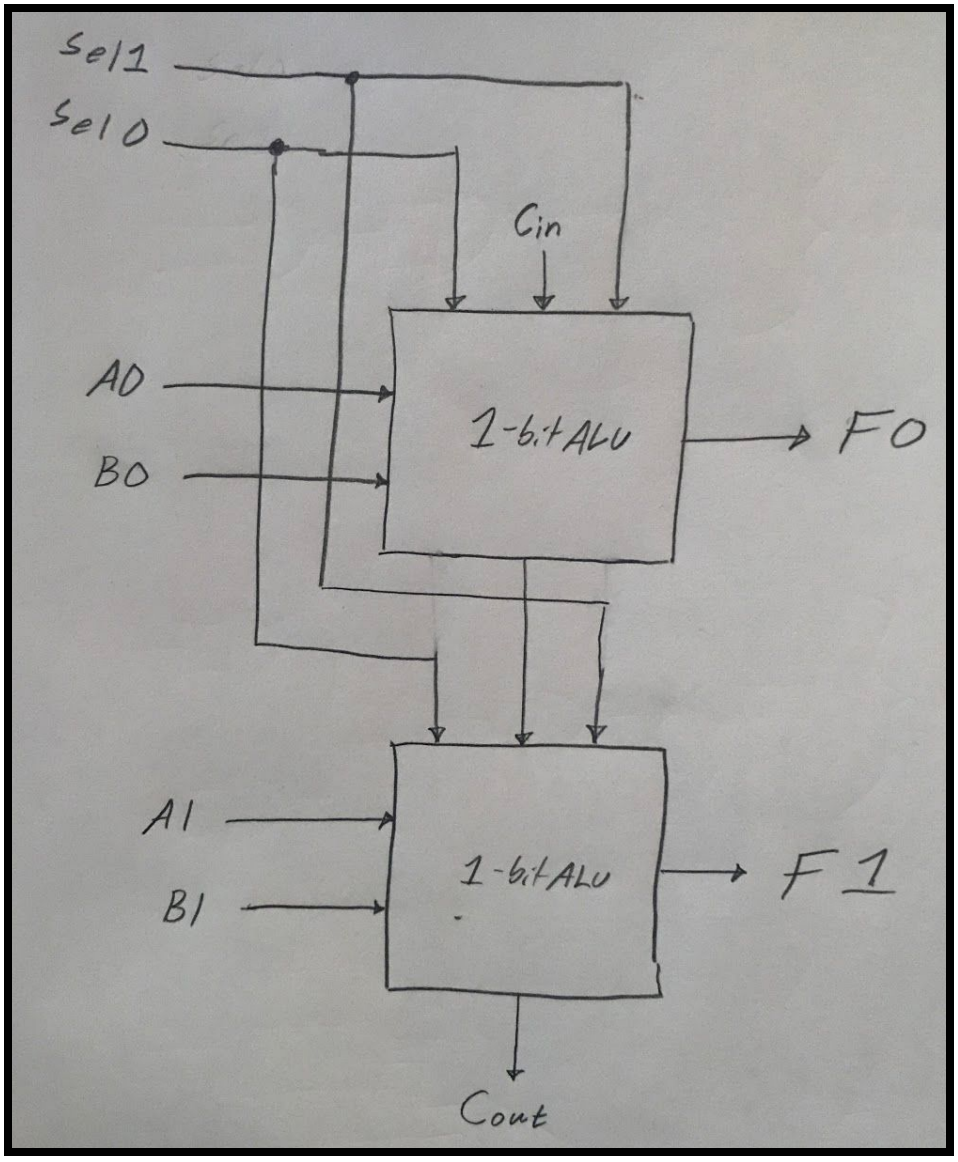


Figure-1: Two-Bit ALU Block Diagram

Design

As stated in the previous section, a two-bit ALU may be constructed from cascading two one-bit ALUs together, therefore, the first step is to design a single one-bit ALU. A one-bit ALU is composed of three units: an arithmetic unit to perform the addition (i.e., a full adder), a logic unit to perform the logical operations (i.e., AND, NOR, and XOR), and a 4:1 multiplexer to provide the output determined by two select lines. The following figure below (figure-2) shows this design of a one-bit ALU.

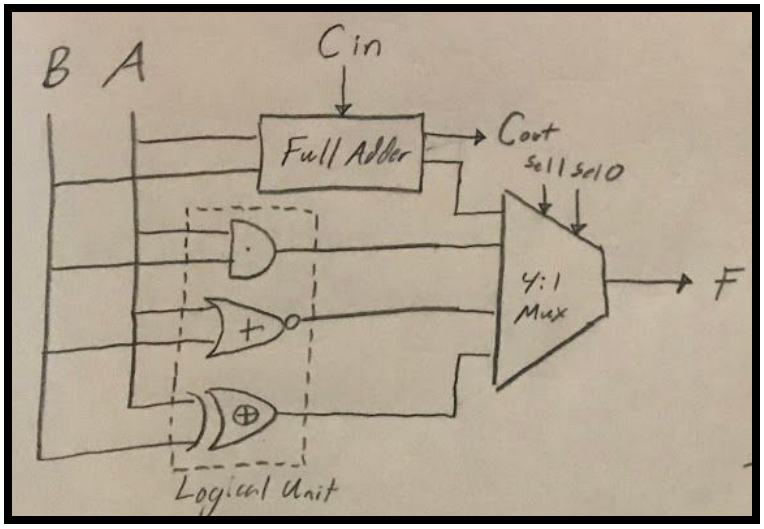


Figure-2: One-Bit ALU Design

The instruction set for this one-bit ALU design is laid out below in table-1.

INPUTS		OUTPUTS
SEL_0	SEL_1	Operation
0	**0**	**Addition**
0	**1**	**AND**
1	**0**	**NOR**
1	**1**	**XOR**

Table-1: Two-Bit ALU Instruction Set

One-Bit ALU Arithmetic Unit

Now that we have the one-bit ALU broken down into the three sub-modules (i.e., the arithmetic unit, the logical unit, and the multiplexer), we can start designing a one-bit full adder required for the arithmetic unit. We must start with the truth table for this module so that we may use k-maps to derive the optimized boolean algebra functions needed to layout the logic-gate design. The next figure (figure-3) lays out the truth tables and k-maps of the one-bit ALU.

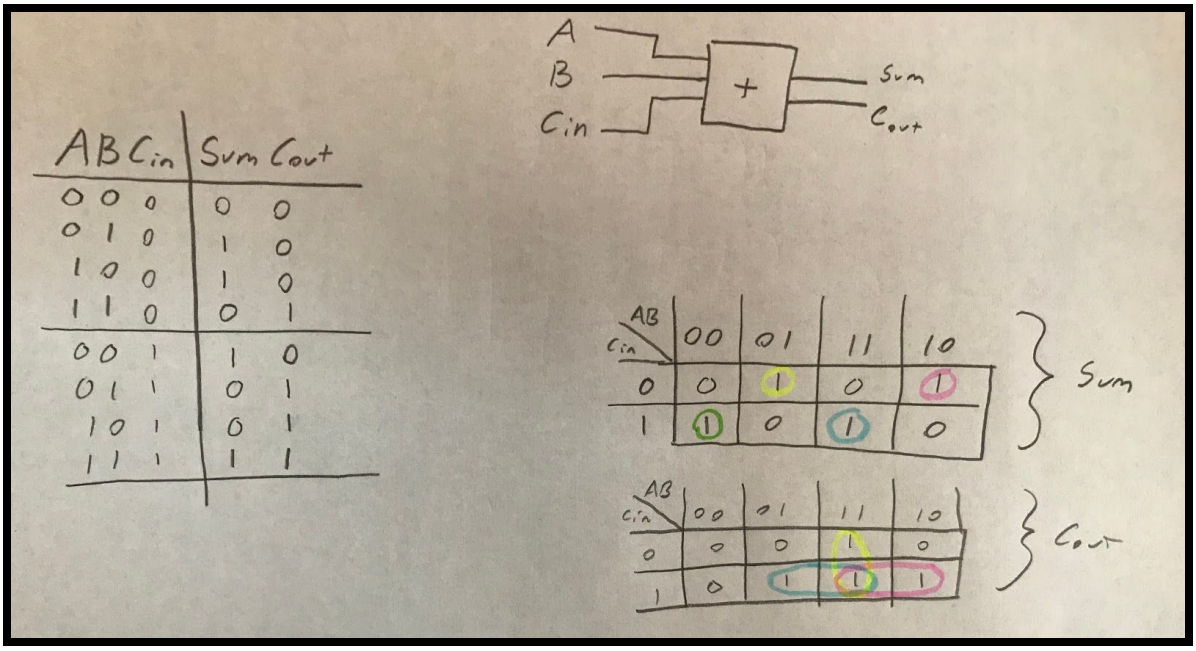


Figure-3: One-Bit Full Adder Truth Tables & K-Maps

Using the K-Maps we can optimize the functions to derive the needed logic gates, which is shown in the following figure (figure-4) on the next page. It details the operations performed to realize the sum and carry-out output variables for the one-bit full adder.

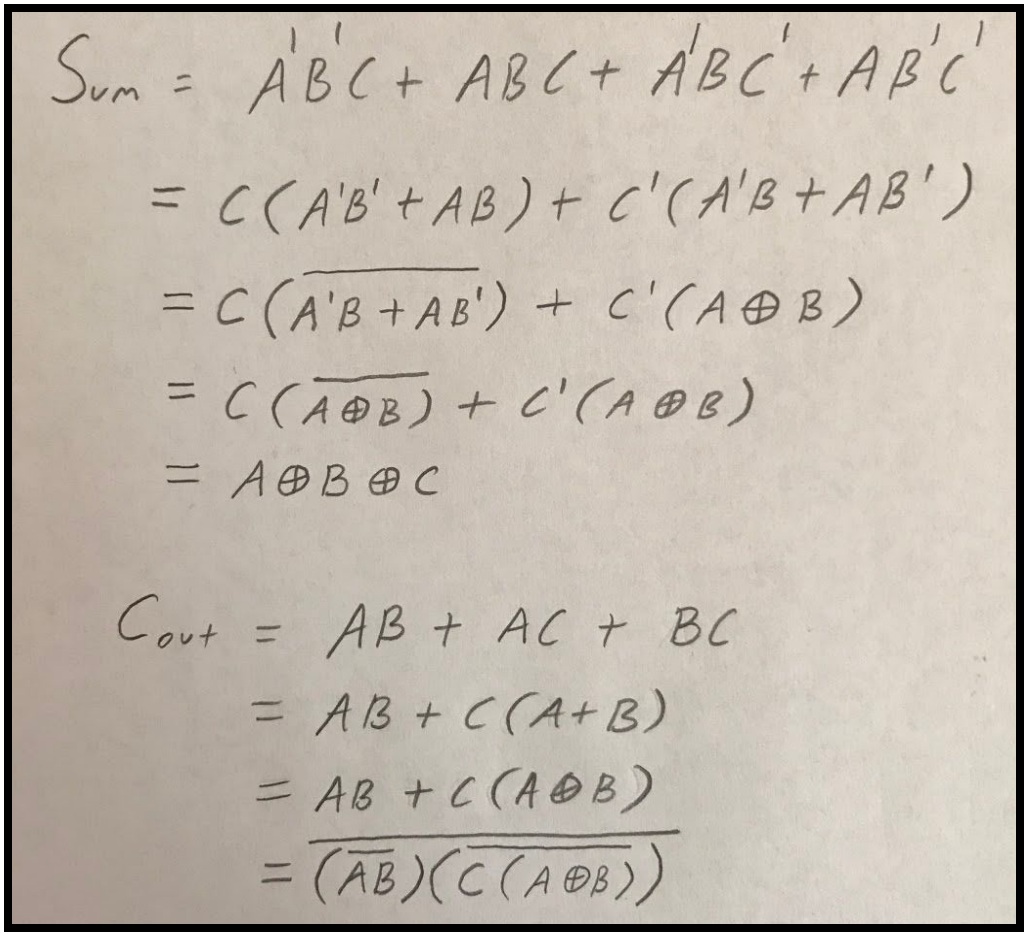


Figure-4: Optimized Boolean Algebra Expressions for One-Bit Full Adder

Now we can finally layout the logic gates to realize a one-bit full adder, as it requires three NAND gates, and two XOR gates. The logic gate design is shown below (figure-5).

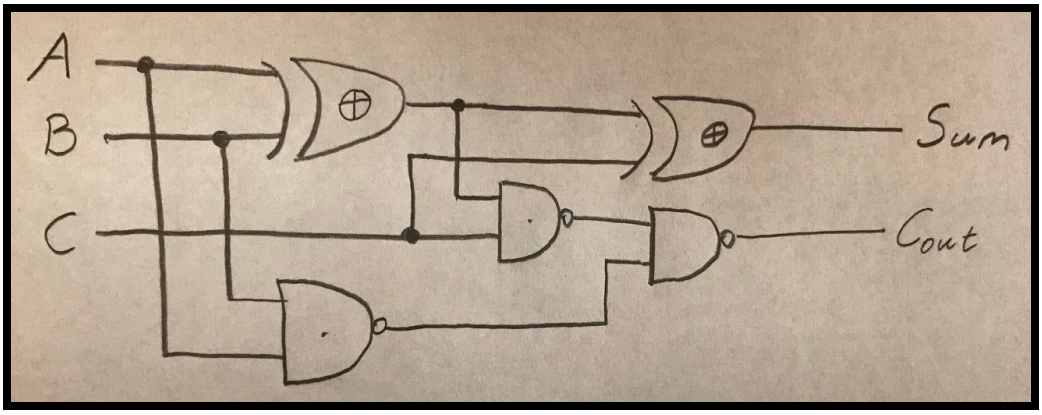


Figure-5: One-Bit Full Adder Logic-Gate Design

One-Bit ALU Multiplexer Unit

Moving onto the second sub-module for the one-bit ALU, we need to design the 4:1 multiplexer. For simplicity, we shall design this using all NAND logic-gates, and break it down into three groups of 2:1 multiplexers, where two select lines are responsible for determining which one of the four input signals will be the output. The breakdown of multiplexer groups are shown below in figure-6, and the logic gate representation follows in figure-7.

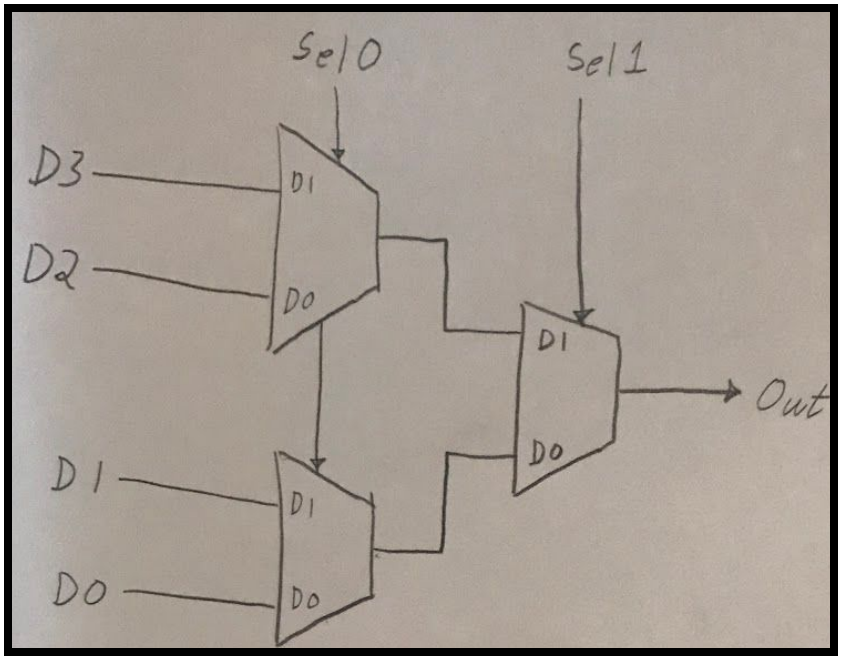


Figure-6: 4:1 Multiplexer Using 2:1 Multiplexers

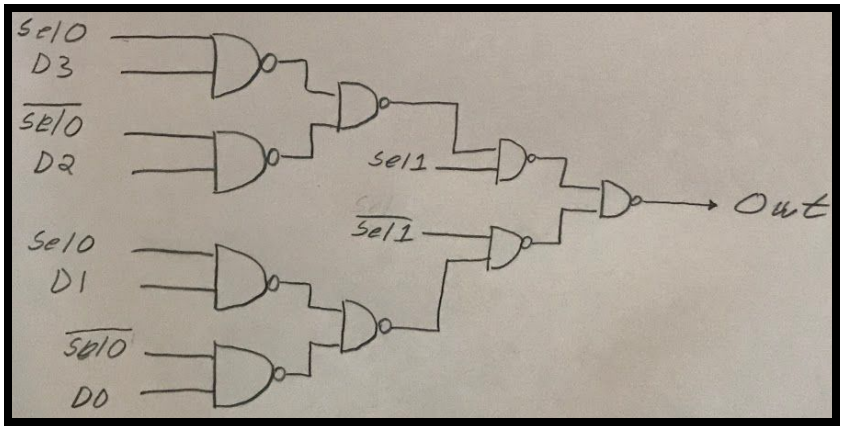


Figure-6: 4:1 Multiplexer Using NAND Logic-Gates

One-Bit ALU Logic Unit

Finally we are onto the third sub-module for the one-bit ALU, where we need to design the AND, NOR, and XOR logic gates. We must start by using the truth tables to describe the function in such a way that we can construct the layout of the design using CMOS transistors.

Firstly, we will design a NAND gate, as it is cheaper to use, we have constructed our multiplexer from them, and we can cascade two NAND gates together to get the functionality of a AND gate. We will also require a XOR gate layout. The truth tables for these logic gates are shown below in tables 2, and 3.

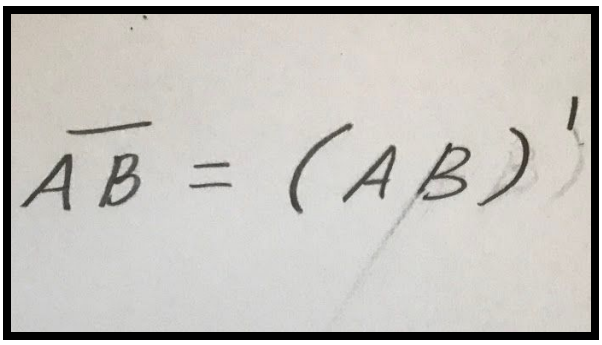
A	B	(AB)’
0	**0**	**1**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

Table-2: NAND Logic-Gate Truth Table

A	B	(A⊕B)
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

Table-3: XOR Logic-Gate Truth Table

Now we may use these tables to derive an expression that allows us to implement the design from CMOS transistors. The following page details these expressions.



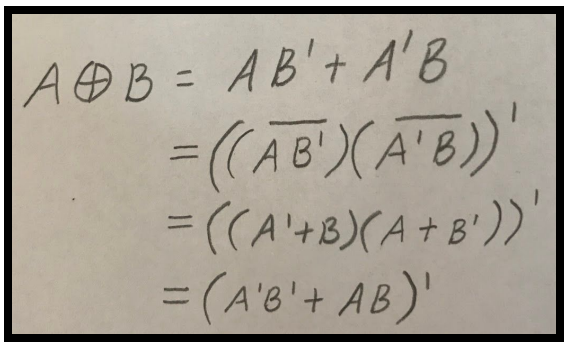


Figure-7: NAND Logic Derive to CMOS Form ***Figure-8: XOR Logic Derive to CMOS Form***

For simplicity, we will use NAND gates throughout and design our wanted NOR functional from the use of multiple NAND gates, as well as our AND gate. The following figure below show an AND gate composed of NAND gates (figure-9).

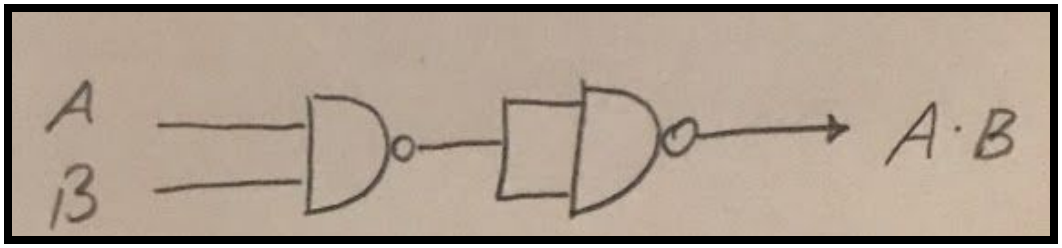


Figure-9: AND Gate Logic via NAND Gates

Likewise, the next figure below shows a NOR gate composed of NAND gates (figure-10).

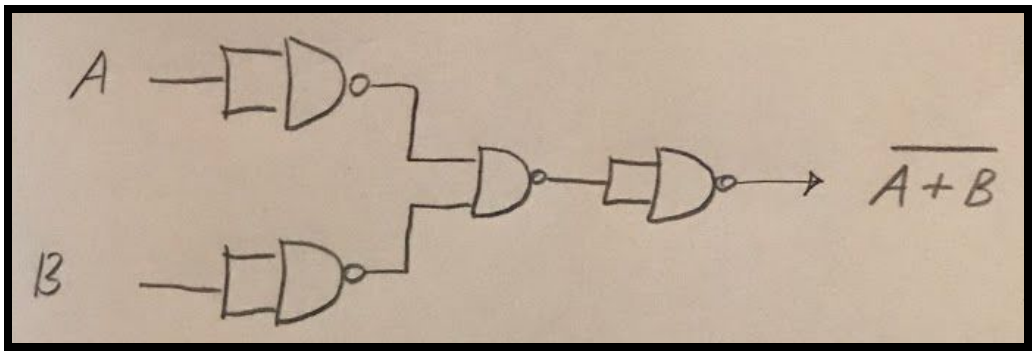


Figure-10: NOR Gate Logic via NAND Gates

Implementation

From the expressions detailed in the design section, we may now implement the CMOS configurations and stick figure diagrams from our needed logic-gates. Note that since we have simplified all of our modules and submodules into XOR and NAND gate designs, we only needed to implement these to gates before we can lay out the CMOS transistors to build our two-bit ALU.

Figures 11, and 12 detail the layout implementation for a NAND gate below.

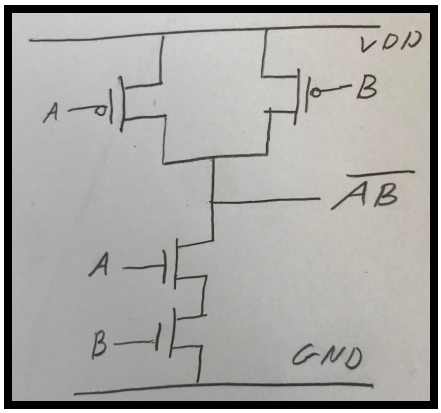


Figure-11: NAND Gate CMOS Implementation

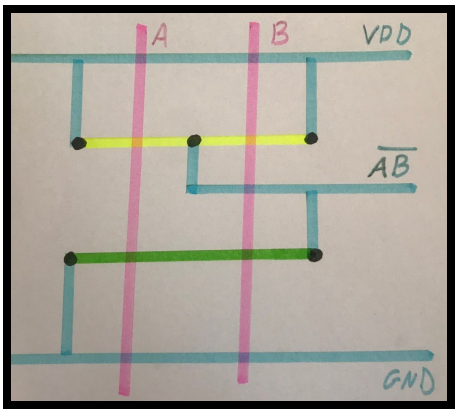


Figure-12: NAND Gate CMOS Stick Diagram

Figures 13, and 14 detail the layout implementation for an XOR gate below.

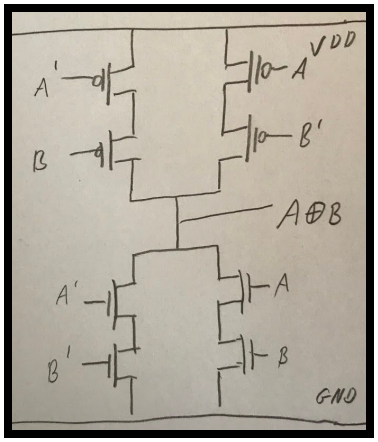


Figure-13: XOR Gate CMOS Implementation

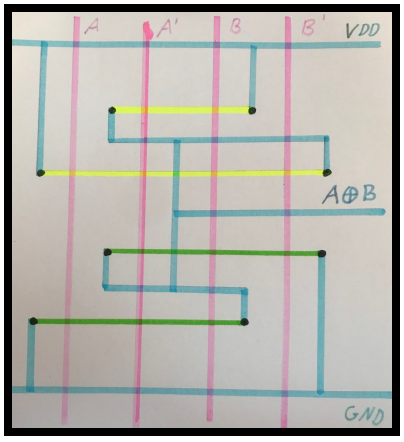


Figure-14: XOR Gate CMOS Stick Diagram

One-Bit ALU in L-Edit

We are now able to implement the transistor layouts in L-Edit to realize the full design. Starting with the one-bit ALU, you can see it has been broken up into two rows, and the left and right sides are divided by the select lines (figure-15).

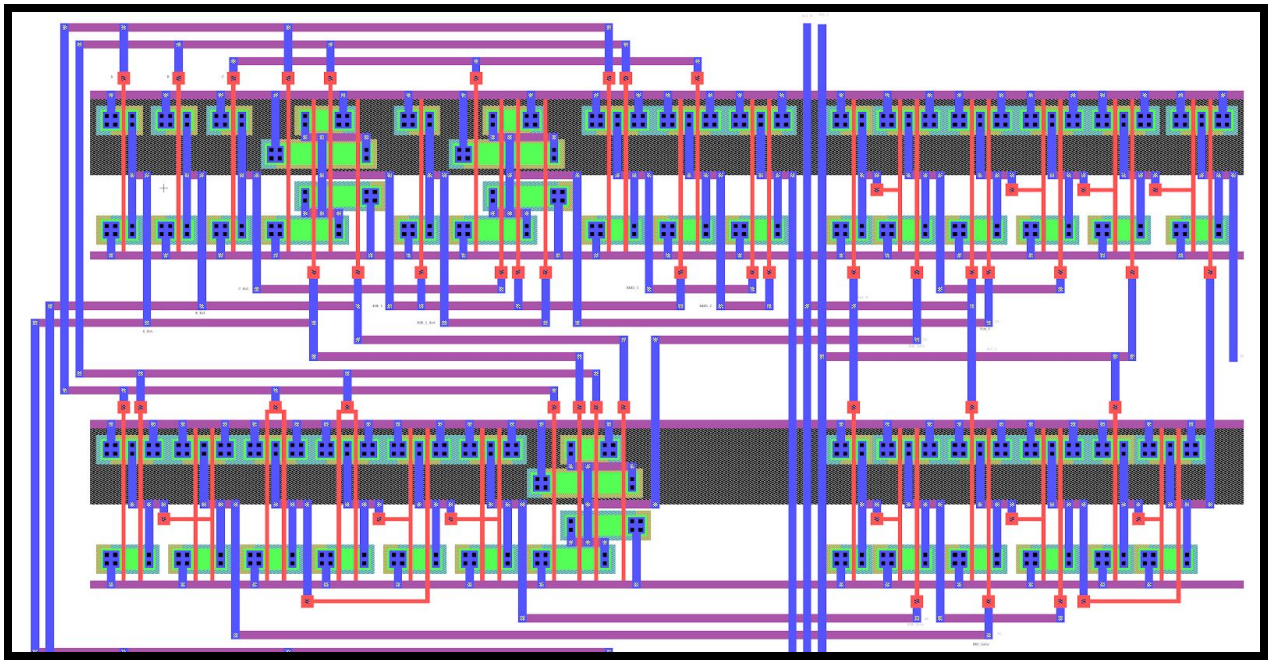


Figure-15: One-Bit ALU Layout in L-Edit

The left side of the top row is the one-bit full adder, and the left of the bottom row is the AND, NOR, and XOR gates laid out correspondingly. The right sides of the one-bit ALU show the 4:1 multiplexer implementation, as the outputs of the logic and arithmetic units are fed in as inputs, and the blue select lines in the middle dictate the operation given for the output.

The last step is to cascade two ALUs to complete our design and initial concept.

Two-Bit ALU in L-Edit

Finally, we arrive at the last layout that illustrates our two-bit ALU from the initial concept.

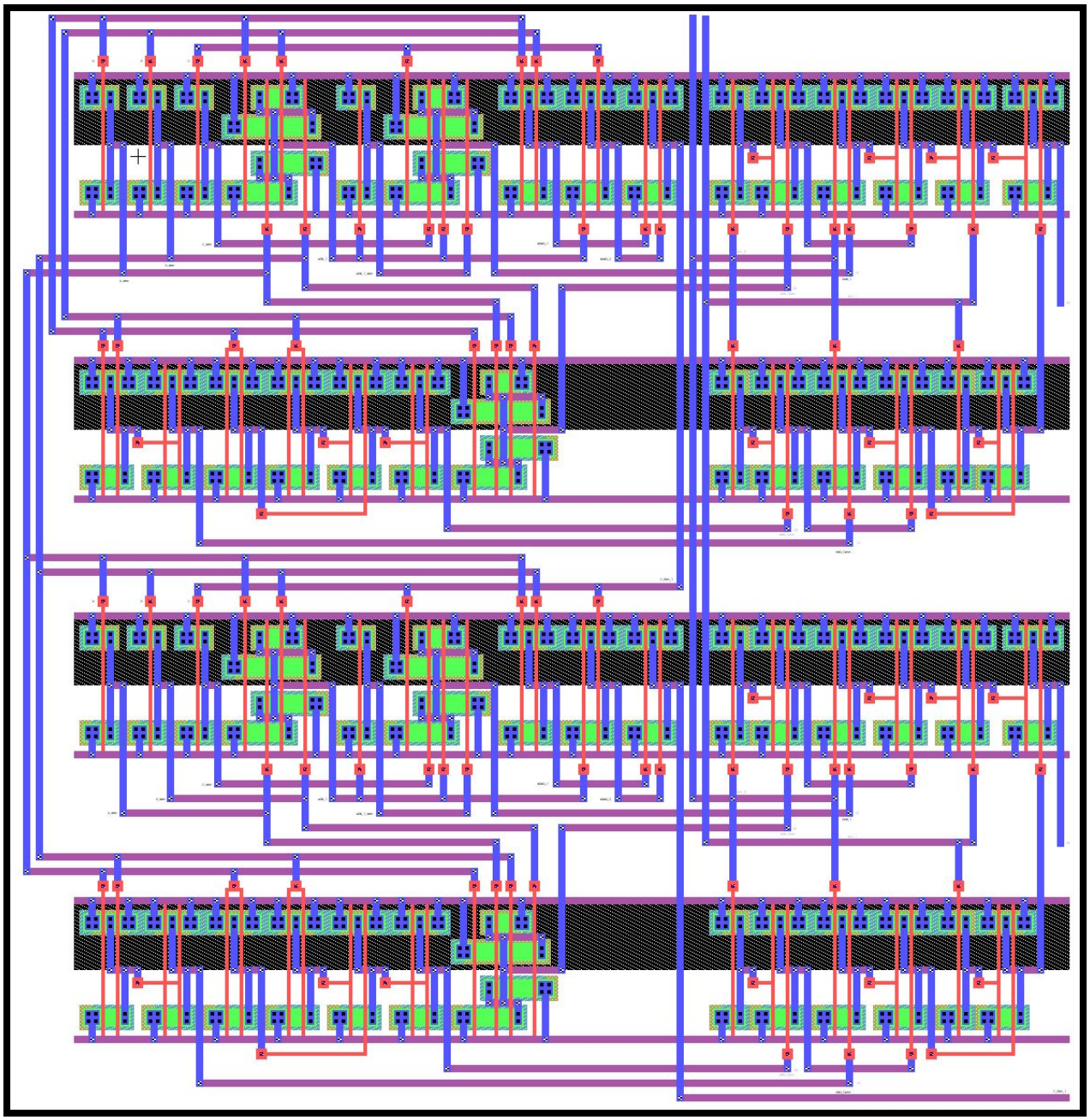


Figure-16: Two-Bit ALU Layout in L-Edit

Testing

The DRC check below (figures 17, and 18) shows that the two-bit ALU is verified to have been implemented corrected using the design rules for the transistor layouts.

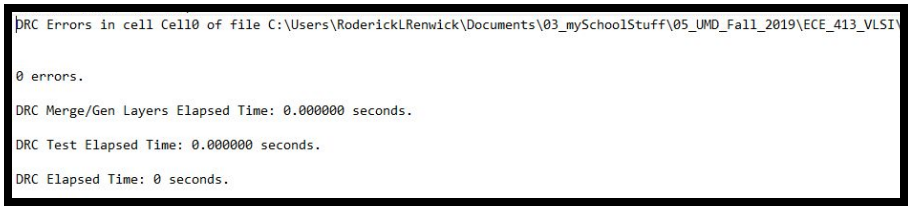


Figure-17: DRC Error File

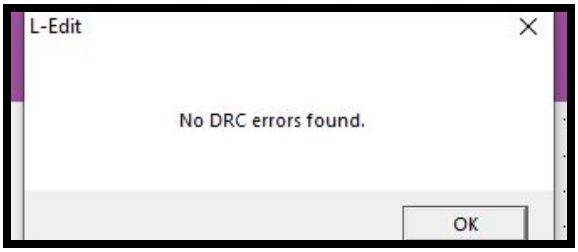


Figure-18: DRC Error Notification

The simulated waveforms below (figure-19) validates our two-bit ALU design as it was implemented in Valvado to run the circuit simulation. The code for the two-bit ALU written in Valvado is located in the following documentation section, detailing the test bench for the simulation as well.

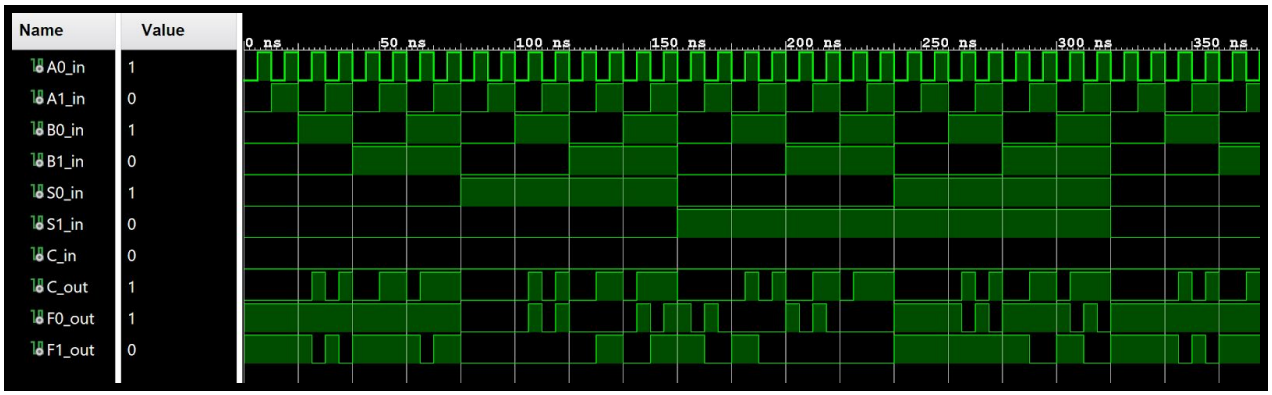


Figure-19: Resulting Waveforms of Two-Bit ALU Simulation

Documentation

The following figures show the implementation of the two-bit ALU through VHDL, simulated with Valvado. The breakdown follows the design steps, and the test bench is provided at the end.

```
1  ----------------------------------------------------------------------------
2  ----------------------------------------------------------------------------
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  library UNISIM;
7  use UNISIM.VComponents.all;
8
9  entity nor2 is                          -- NAND_GATE ENTITY DECLARATION
10     port (
11             i0, i1 : in std_logic;
12             o : out std_logic
13           );
14 end nor2;
15
16 architecture behaviour of nor2 is      -- NAND_GATE ARCHITECTURE DECLARATION
17 begin
18     o <= not ( i0 or i1 );               -- NAND_GATE BEHAVIORAL STATEMENT
19 end behaviour;
20 ----------------------------------------------------------------------------
21 ----------------------------------------------------------------------------
```

```
20 ----------------------------------------------------------------------------
21 ----------------------------------------------------------------------------
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 library UNISIM;
26 use UNISIM.VComponents.all;
27
28 entity nand2 is                          -- NAND_GATE ENTITY DECLARATION
29     port (
30             i0, i1 : in std_logic;
31             o : out std_logic
32           );
33 end nand2;
34
35 architecture behaviour of nand2 is      -- NAND_GATE ARCHITECTURE DECLARATION
36 begin
37     o <= not ( i0 and i1 );               -- NAND_GATE BEHAVIORAL STATEMENT
38 end behaviour;
39 ----------------------------------------------------------------------------
40 ----------------------------------------------------------------------------
```

```
39 ----------------------------------------------------------------------------
40 ----------------------------------------------------------------------------
41 library IEEE;
42 use IEEE.STD_LOGIC_1164.ALL;
43
44 library UNISIM;
45 use UNISIM.VComponents.all;
46
47 entity and2 is                          -- NAND_GATE ENTITY DECLARATION
48     port (
49             i0, i1 : in std_logic;
50             o : out std_logic
51           );
52 end and2;
53
54 architecture behaviour of and2 is      -- NAND_GATE ARCHITECTURE DECLARATION
55 begin
56     o <= ( i0 and i1 );                -- NAND_GATE BEHAVIORAL STATEMENT
57 end behaviour;
58 ----------------------------------------------------------------------------
59 ----------------------------------------------------------------------------
```

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

library UNISIM;
use UNISIM.VComponents.all;

entity xor2 is                          -- XOR_GATE ENTITY DECLARATION
    port (
           i0, i1 : in std_logic;
           o : out std_logic
         );
end xor2;

architecture behaviour of xor2 is       -- XOR_GATE ARCHITECTURE DECLARATION
begin
    o <= not ( ( not ( ( not ( i0 and i1 ) ) and i0 ) ) and ( not ( not ( i0 and i1 ) ) and i1 ) );              -- XOR_GATE BEHAVIORAL
end behaviour;
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
```

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

library UNISIM;
use UNISIM.VComponents.all;

entity one_bit_full_adder is                        -- XOR_GATE ENTITY DECLARATION
    Port (
           X_in : in STD_LOGIC;
           Y_in : in STD_LOGIC;
           C_in : in STD_LOGIC;
           S_out : out STD_LOGIC;
           C_out : out STD_LOGIC
         );
end one_bit_full_adder;

architecture struct of one_bit_full_adder is      -- XOR_GATE ARCHITECTURE DECLARATION

    component xor2
    port ( i0, i1 : in std_logic; o : out std_logic );
    end component;

    component nand2
    port ( i0, i1 : in std_logic; o : out std_logic );
    end component;
```

```

    -- Internal Signals Declarations
    signal xor0  : std_logic;
    signal nand0 : std_logic;
    signal nand1 : std_logic;

begin

    -- Component Instantiations Statements
    u0  : nand2 port map ( i0 => X_in, i1 => Y_in, o => nand0 );
    u1  : nand2 port map ( i0 => xor0, i1 => C_in, o => nand1 );
    u2  : nand2 port map ( i0 => nand0, i1 =>nand1, o => C_out );
    u3  :  xor2 port map ( i0 => X_in, i1 => Y_in, o => xor0 );
    u4  :  xor2 port map ( i0 => xor0, i1 => C_in, o => S_out );

end struct;
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
```

```
20 ----------------------------------------------------------------------------
21 ----------------------------------------------------------------------------
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  library UNISIM;
26  use UNISIM.VComponents.all;
27
28 entity four_one_mux is                          -- XOR_GATE ENTITY DECLARATION
29      Port (
30             D0 : in STD_LOGIC;
31             D1 : in STD_LOGIC;
32             D2 : in STD_LOGIC;
33             D3 : in STD_LOGIC;
34             S0 : in STD_LOGIC;
35             S1 : in STD_LOGIC;
36             M0 : out STD_LOGIC
37           );
38 end four_one_mux;
39
40 architecture struct of four_one_mux is      -- XOR_GATE ARCHITECTURE DECLARATION
41
42      component nand2
43      port ( i0, i1 : in std_logic; o : out std_logic );
44      end component;
45
```

```
46      -- Internal Signals Declarations
47      signal not_S0 : std_logic;
48      signal not_S1 : std_logic;
49
50      signal nand_0 : std_logic;
51      signal nand_1 : std_logic;
52      signal nand_2 : std_logic;
53      signal nand_3 : std_logic;
54      signal nand_4 : std_logic;
55      signal nand_5 : std_logic;
56      signal nand_6 : std_logic;
57      signal nand_7 : std_logic;
```

```
59  begin
60
61      not_S0 <= not S0;
62      not_S1 <= not S1;
63
64      -- Component Instantiations Statements
65      u0  : nand2 port map ( i0 =>     S0, i1 => D3, o => nand_0 );
66      u1  : nand2 port map ( i0 => not_S0, i1 => D2, o => nand_1 );
67      u2  : nand2 port map ( i0 =>     S0, i1 => D1, o => nand_2 );
68      u3  : nand2 port map ( i0 => not_S0, i1 => D0, o => nand_3 );
69
70      u4  : nand2 port map ( i0 => nand_0, i1 => nand_1, o => nand_4 );
71      u5  : nand2 port map ( i0 => nand_2, i1 => nand_3, o => nand_5 );
72
73      u6  : nand2 port map ( i0 => nand_4, i1 =>     S1, o => nand_6 );
74      u7  : nand2 port map ( i0 => nand_5, i1 => not_S1, o => nand_7 );
75
76      u8  : nand2 port map ( i0 => nand_6, i1 => nand_7, o => M0 );
77
78 end struct;
79 ----------------------------------------------------------------------------
80 ----------------------------------------------------------------------------
```

```
179 ----------------------------------------------------------------------------
180 ----------------------------------------------------------------------------
181 library IEEE;
182 use IEEE.STD_LOGIC_1164.ALL;
183
184 library UNISIM;
185 use UNISIM.VComponents.all;
186
187 entity one_bit_alu is                -- 2-bit adder ENTITY DECLARATION
188     Port (
189           X0_in  : in STD_LOGIC;
190           Y0_in  : in STD_LOGIC;
191           C_in   : in STD_LOGIC;
192           C_out  : out STD_LOGIC;
193           S0_in  : in STD_LOGIC;
194           S1_in  : in STD_LOGIC;
195           S_out  : out STD_LOGIC
196         );
197 end one_bit_alu;
198
199 architecture struct of one_bit_alu is-- 2-bit adder ARCHITECTURE DECLARATION
200
201     component one_bit_full_adder                  -- NAND_GATE Component Declaration
202     port (
203            X_in, Y_in, C_in: in std_logic;
204            C_out, S_out : out std_logic
205          );
206     end component;
207
```

```
207
208     component xor2
209     port ( i0, i1 : in std_logic; o : out std_logic );
210     end component;
211
212     component nor2
213     port ( i0, i1 : in std_logic; o : out std_logic );
214     end component;
215
216     component and2
217     port ( i0, i1 : in std_logic; o : out std_logic );
218     end component;
219
220     component four_one_mux
221     port ( D0, D1, D2, D3, S0, S1 : in std_logic; M0 : out std_logic );
222     end component;
223
224     -- Internal Signals Declarations
225     signal D0 : std_logic;
226     signal D1 : std_logic;
227     signal D2 : std_logic;
228     signal D3 : std_logic;
```

```
230 begin
231
232     -- Component Instantiations Statements
233     u0  : one_bit_full_adder port map ( X_in => X0_in, Y_in => Y0_in, C_in => C_in, C_out => C_out, S_out => D0 );
234     u1  : and2 port map ( i0 => X0_in, i1 => Y0_in, o => D1 );
235     u2  : nor2 port map ( i0 => X0_in, i1 => Y0_in, o => D2 );
236     u3  : xor2 port map ( i0 => X0_in, i1 => Y0_in, o => D3 );
237     u4  : four_one_mux port map ( D0 => D0, D1 => D1, D2 => D2, D3 => D3, S0 => S0_in, S1 => S1_in, M0 => S_out);
238
239 end struct;
240 ----------------------------------------------------------------------------
241 ----------------------------------------------------------------------------
```

```
240 ----------------------------------------------------------------------------
241 ----------------------------------------------------------------------------
242 library IEEE;
243 use IEEE.STD_LOGIC_1164.ALL;
244
245 library UNISIM;
246 use UNISIM.VComponents.all;
247
248 entity two_bit_alu is                -- 2-bit adder ENTITY DECLARATION
249     Port (
250           A0_in  : in STD_LOGIC;
251           B0_in  : in STD_LOGIC;
252           A1_in  : in STD_LOGIC;
253           B1_in  : in STD_LOGIC;
254           S0_in  : in STD_LOGIC;
255           S1_in  : in STD_LOGIC;
256           C_in   : in STD_LOGIC;
257           C_out  : out STD_LOGIC;
258           F0_out : out STD_LOGIC;
259           F1_out : out STD_LOGIC
260         );
261 end two_bit_alu;
```

```
263 architecture struct of two_bit_alu is-- 2-bit adder ARCHITECTURE DECLARATION
264
265     component one_bit_alu
266     port (
267           X0_in  : in STD_LOGIC;
268           Y0_in  : in STD_LOGIC;
269           C_in   : in STD_LOGIC;
270           C_out  : out STD_LOGIC;
271           S0_in  : in STD_LOGIC;
272           S1_in  : in STD_LOGIC;
273           S_out  : out STD_LOGIC
274          );
275     end component;
276
277     -- Internal Signals Declarations
278     signal c_out_signal : std_logic;
279
280 begin
281
282     -- Component Instantiations Statements
283     u0  : one_bit_alu port map ( X0_in => A0_in, Y0_in => B0_in, C_in => C_in, C_out => c_out_signal, S0_in => S0_in, S1_in => S1_in, S
284     u1  : one_bit_alu port map ( X0_in => A1_in, Y0_in => B1_in, C_in => c_out_signal, C_out => C_out, S0_in => S0_in, S1_in => S1_in, S
285
286 end struct;
287 ----------------------------------------------------------------------------
288 ----------------------------------------------------------------------------
```

The last two figures displayed below are of the test bench file used for the simulation.

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

library UNISIM;
use UNISIM.VComponents.all;

entity two_bit_alu_tb is
end two_bit_alu_tb;

architecture two_bit_alu_tb of two_bit_alu_tb is

    component two_bit_alu is
        Port (
               A0_in  : in STD_LOGIC;
               B0_in  : in STD_LOGIC;
               A1_in  : in STD_LOGIC;
               B1_in  : in STD_LOGIC;
               S0_in  : in STD_LOGIC;
               S1_in  : in STD_LOGIC;
               C_in   : in STD_LOGIC;
               C_out  : out STD_LOGIC;
               F0_out : out STD_LOGIC;
               F1_out : out STD_LOGIC
             );
    end component;
```

```
    signal A0_in  : std_logic := '0';
    signal A1_in  : std_logic := '0';
    signal B0_in  : std_logic := '0';
    signal B1_in  : std_logic := '0';
    signal S0_in  : std_logic := '0';
    signal S1_in  : std_logic := '0';
    signal C_in   : std_logic := '0';
    signal C_out  : std_logic;
    signal F0_out : std_logic;
    signal F1_out : std_logic;

begin

    UUT : two_bit_alu
    port map (
               A0_in => A0_in, A1_in => A1_in, B0_in => B0_in, B1_in => B1_in, C_in => C_in, S0_in => S0_in, S1_in => S1_in,
               C_out => C_out, F0_out => F0_out, F1_out => F1_out
             );

    A0_in <= NOT A0_in after 5 ns;
    A1_in <= NOT A1_in after 10 ns;
    B0_in <= NOT B0_in after 20 ns;
    B1_in <= NOT B1_in after 40 ns;
    S0_in <= NOT S0_in after 80 ns;
    S1_in <= NOT S1_in after 160 ns;

end two_bit_alu_tb;
```