DCU School of Electronic Engineering Assignment Submission

Student Name(s):	**Michael Lenehan**
Student Number(s):	**15410402**
Programme:	**B.Eng in Electronic and Computer Engineering**
Module Code:	**EE496**
Lecturer:	**X. Wang**
Project Due Date:	**31/10/2018**

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited andacknowledged within the text of my work. I understand thatplagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.

Signed: ***Michael Lenehan***

Assignment 1

Michael Lenehan

Purpose

The purpose of this assignment is to implement and test a number of adder/subtractor designs. The exercises allow the student to become familiar with both the VHDL language, and the Vivado development environment. The first exercise requires a one-bit full adder to be completed, while the second exercise requires this to be expanded to a four-bit adder/subtractor design. The third exercise again expands on the design, adding a number of flags to the adder/subtractor. These flags indicate if a negative result has been achieved, if the result is zero, if a carry has occurred, or if an overflow has occurred.

1.1 One-Bit Full Adder

The purpose of this section of the lab is to implement a One-Bit Full Adder in VHDL, and to verify its implementation using a test bench. The VHDL model created must operate as a one-bit full adder does, giving the following truth table:

Table 1: Truth Table for a One-Bit Full Adder

c_{in}	b	a	c_{out}	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The one-bit full adder is completed using a combination of XOR, AND, and OR operations. The provided diagram indicates the routing of the inputs through the logic gates,

resulting in a correct sum and carry outputs.

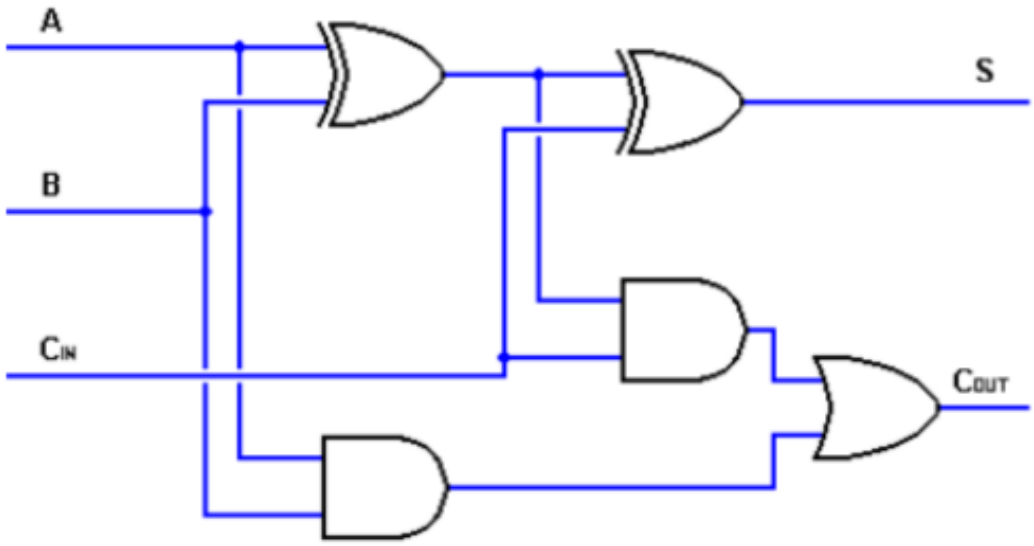

Figure 1: Provided Full Adder Schematic

This operation must be verified using a test bench. The test bench must cycle through all possible input values in such a way that all possible output values are achieved.

1.2 Four-Bit Adder/Subtractor

The purpose of this section is to expand on the one-bit full adder design, allowing for four bit binary numbers to be added, and, using two's compliment, to be subtracted. The range of binary numbers possible to output from this Adder/Subtractor is 0000 to 1111 (which represents 0 to 15 in decimal) for unsigned numbers, and 1000 to 0111 (which represents -8 to 7 in decimal) for signed numbers.

Using the provided diagram, the four-bit adder/subtractor can be created using a number of full adder components, and eight inputs. A "sub" input is required to implement the subtraction functionality. Binary subtraction requires the two's compliment of the signed binary numbers. The two's compliment of the "B" inputs is calculated by XOR'ing the input bits with the "sub" input, and adding the "sub" input as a carry in to the first full adder.

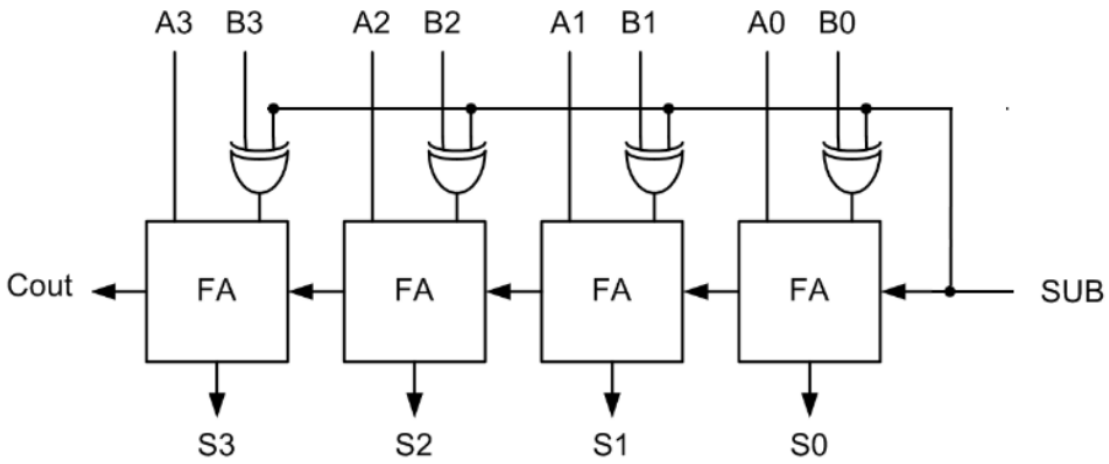

Figure 2: Provided Four-Bit Adder/Subtracter Schematic

The test bench developed must cycle through all combinations of four-bit binary inputs, and perform either an addition or subtraction operation, as specified by the "sub" select value. In doing this, all possible output values for the adder/subtractor are achieved.

1.3 Status Flags

The purpose of this section is to add a number of status flags to the four-bit adder /subtractor. The flags must indicate when a certain condition has occurred. The flags used are as follows:

Table 2: Description of the implemented Status Flags

Flag	*Symbol*	*Purpose*
Negative	N	Indicates a negative result
Zero	Z	Indicates all resulting bits are '0'
Carry	C	Indicates a carry out (overflow for signed values)
Overflow	V	Indicated an overflow has occurred, i.e. result is out of range

The developed test bench for this section must expand on the four-bit adder/subtractor test bench. When any of the flag conditions are satisfied, the flag value must be set. This is achieved in the following way for each of the flags:

1.3.1 Negative Flag:

The flag value is assigned the value of the four-bit sum outputs most significant bit, if the "sub" input is set high. If not, the flag is set low.

1.3.2 Zero Flag:

An OR operation is performed on each bit of the sum output. A conditional assignment is used to set the zero flag value to 1 if there is a 1 present in the sum, otherwise it is set to 0.

1.3.3 Carry Flag:

The carry flag is assigned the value of the final carry out from the four-bit adder/subtracter.

1.3.4 Overflow Flag:

The overflow flag is assigned the output of an XOR operation on the third, and the final carry out values of the four-bit adder/subtracter. If the final carry out value is set high, then overflow has occurred, as the output value cannot be represented in the available four-bits. If the third carry out is set high, the sign bit, i.e. sum bit four has been modified, and the output value does not represent the actual summation value.

Results

The following results were obtained from the completion of the assignment exercises.

2.1 One-Bit Full Adder

Once complete, the one-bit full adder generates the following output schematic:

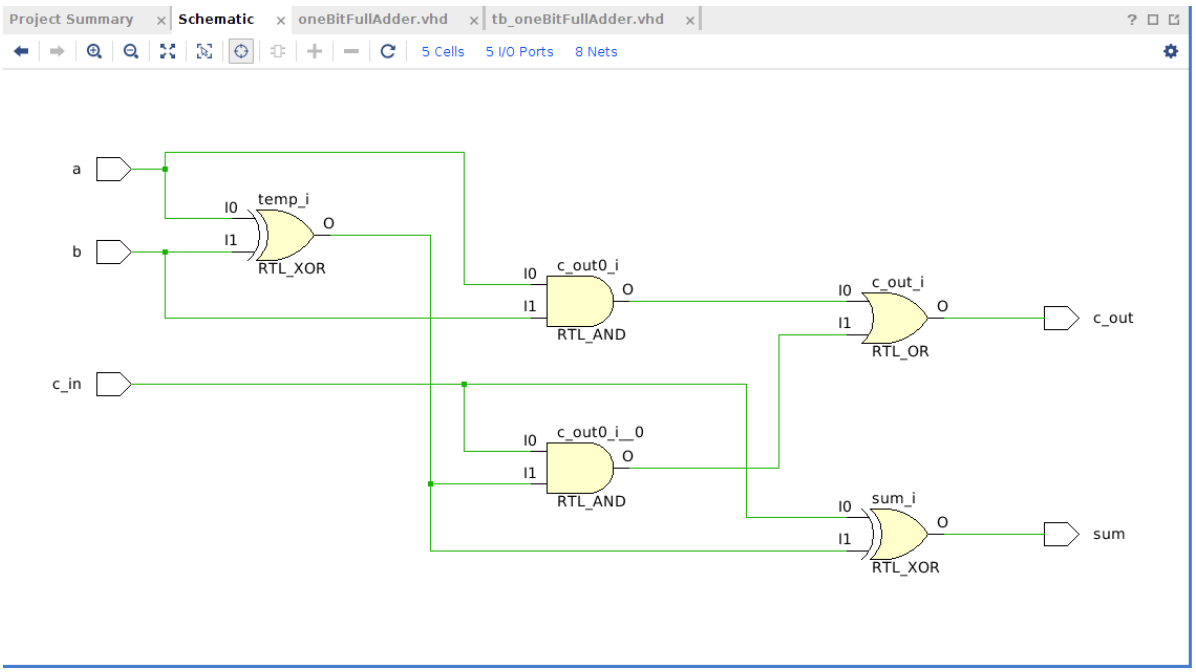

Figure 3: Generated One-Bit Full Adder Schematic

The sum output of the full adder is determined using an XOR operation on the A and B inputs, followed by an XOR of the output of this operation and the C_{in} input. The C_{out} output is determined using an OR operation on the output of two AND operations. The first AND operation is performed on the A and B inputs, with the second performed on the XOR'd A and B inputs, and the C_{in} input.

The one-bit full adder is tested using the developed test bench, which cycles through the input values for A, B, and C_{in}. The scope output, as displayed below in Figure 4, shows all possible outputs, given by all possible combinations of inputs.

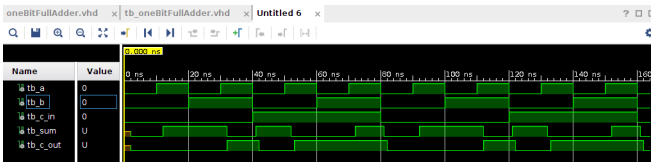

Figure 4: One-Bit Full Adder Test Bench Scope Output

2.2 Four-Bit Adder/Subtractor

Once complete, the four-bit adder/subtractor generates the following output schematic:

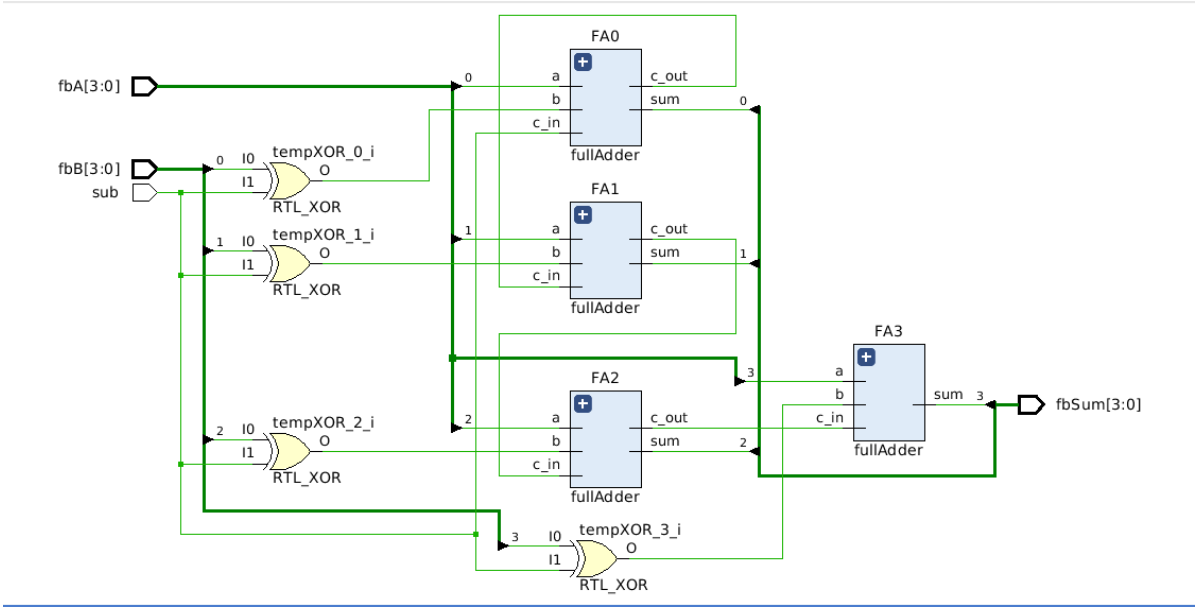

Figure 5: Generated Four-Bit Adder/Subtractor Schematic

The four-bit adder consists of eight inputs, four A inputs, and four B inputs, and four full adders. In order to implement the subtraction functionality, four XOR operations are performed between the sub input, and each of the B input bits. The sub input acts as the C_{in} input for the first full adder. The C_{out} of each full adder acts as the C_{in} for the next full adder.

The four-bit adder/subtractor is tested using the developed test bench, which cycles through all possible combinations of A, B, and sub inputs. The scope output in Figure 6 shows the addition operation, and the sum output achieved. Within the scope output, a hexadecimal representation of the binary inputs can be seen. While useful for this example, this can be difficult to correctly interpret when dealing with negative numbers. For this reason, decimal representations will be used throughout the results. A number of working addition operations can be seen in Figure 6, e.g. $_{dec}1 +_{dec} 6 =_{dec} 7$.

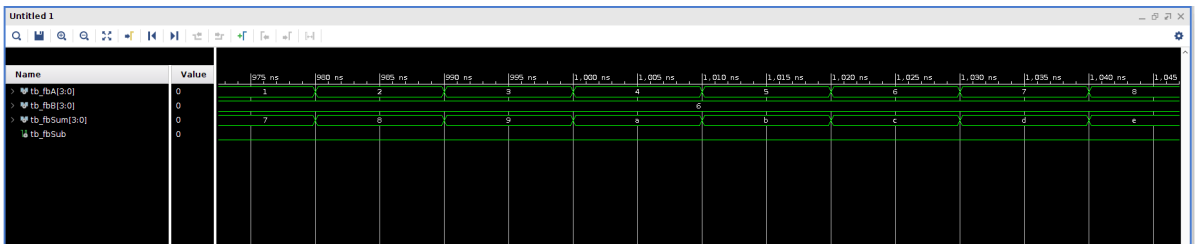

Figure 6: Four-Bit Adder/Subtractor Addition Operation Scope Output

The addition shown below, in Figure 7, shows the result of an addition operation on two binary inputs out of the range of a four-bit adder. The scope output shows the hexidecimal and binary representations of all input and outputs, and demonstrates the addition of two large unsigned binary numbers, e.g. $_{dec}14 +_{dec} 14 =_{dec} 28$. As can be seen from the scope output, the most significant bit is not included in the sum output, resulting in an output of $_{dec}12$. This may also be interpreted as the addition of two negative numbers, i.e. $-_{dec}6 -_{dec} 6 = -_{dec}4$. Again, the carry is out of range, resulting in the incorrect output.

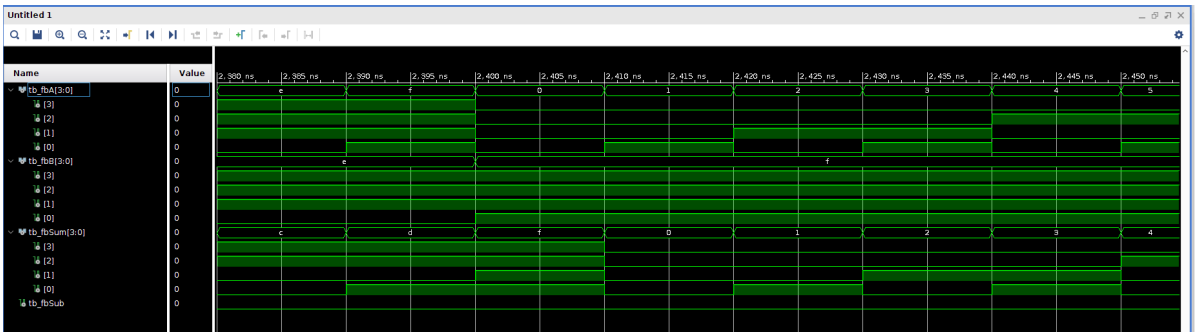

Figure 7: Four-Bit Adder/Subtractor Out of Range Addition of Two Unsigned Positives/Negatives Scope Output

Figure 8 shows the results obtained from the subtraction of two binary inputs within range of the adder/subtractor. The scope output shows the hexadecimal and binary representations of all inputs and outputs, and demonstrates the output of the operation, e.g. $_{dec}4 -_{dec} 3 =_{dec} 1$. Subtractions resulting in a negative output can also be seen to operate correctly e.g. $_{dec}1 -_{dec} 3 =_{dec} -2$

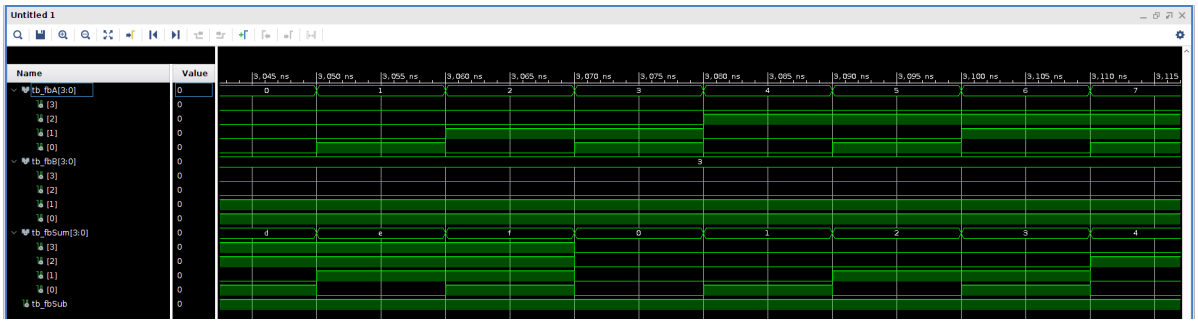

Figure 8: Four-Bit Adder/Subtractor Subtraction Operation Scope Output

The subtraction shown below, Figure 9, demonstrates the subtraction operation resulting in an incorrect output due to overflow. For example, $_{dec}2 - -_{dec}6 = -_{dec}8$. The most

significant, signed bit has been lost, resulting in a negative 8 output, rather than a positive 8.

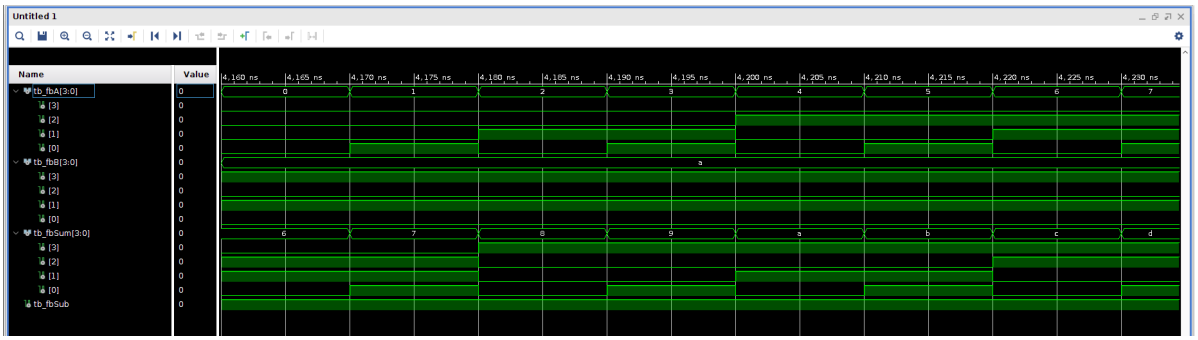

Figure 9: Four-Bit Adder/Subtractor Subtraction Operation Output Overflow Scope Output

2.3 Status Flags

The completed test bench for the Status Flags section generates the following schematic:

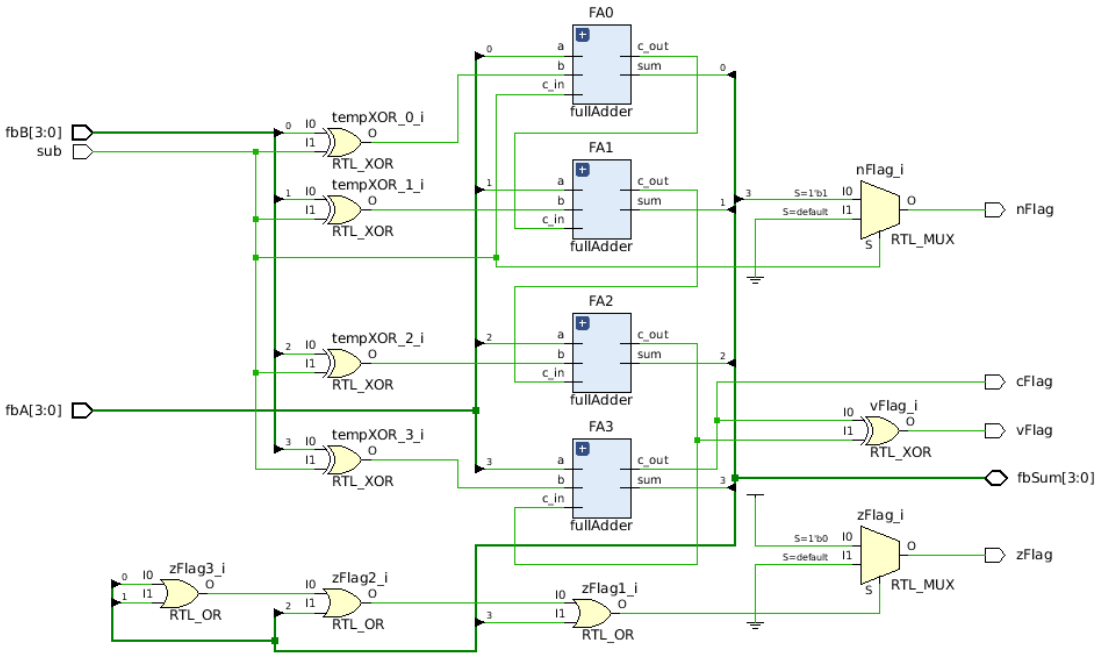

Figure 10: Generated Four-Bit Adder/Subtractor with Flags Schematic

The status flag functionality is added to the four-bit adder/subtractor using a number of logic gates and multiplexers.

The negative flag is assigned the value of the most significant bit of the output sum. If the output is negative, the most significant bit is high, and the flag is set high. If the output is positive, the most significant bit is low, and the flag is set low. A conditional assignment, implemented using a multiplexer, is used to set the negative flag, only if a subtraction has occurred, allowing unsigned four-bit additions to be completed. As in Figure 11, the negative flag is set high when a subtraction results in a negative sum output.

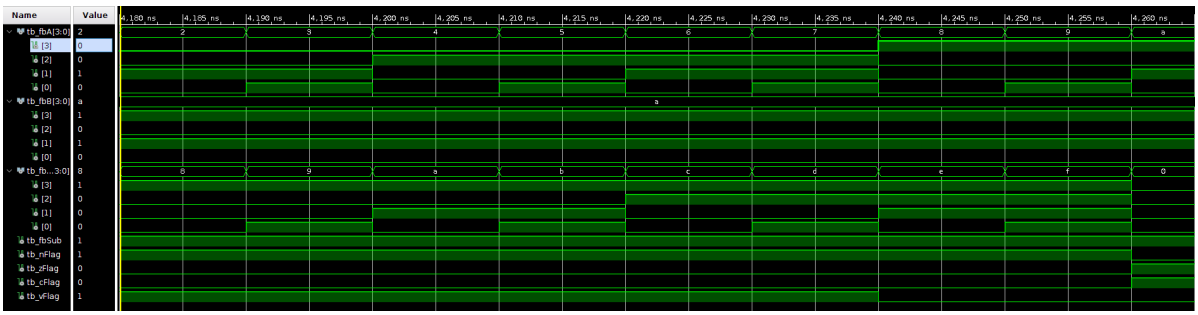

Figure 11: Four-Bit Adder/Subtractor Scope Output with Negative Output and Flag

The zero flag requires each bit of the sum to be OR'd, and, using a conditional assignment, which is implemented using a multiplexer, the flag is set if no bits are high. As in Figure 12, the zero flag is set high when the sum output is zero.

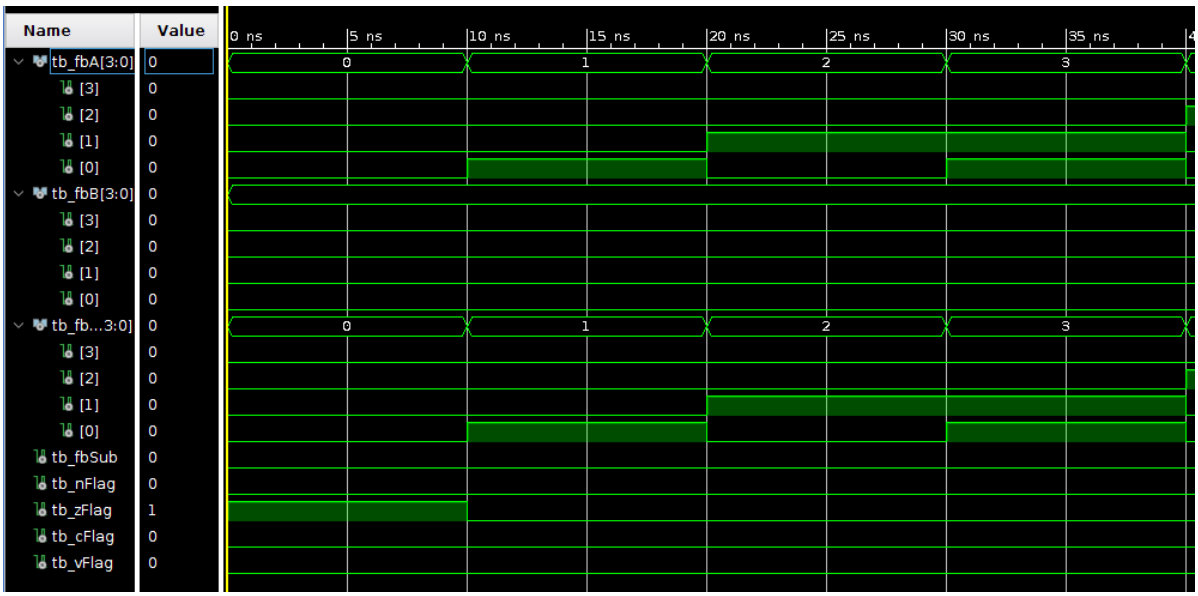

Figure 12: Four-Bit Adder/Subtractor Scope Output with Zero Output and Flag

The carry flag is assigned the value of the fourth full adders carry output. As in Figure 13, the carry value is set when a carry occurs in the final full adder output, e.g. $_{dec}14 +_{dec} 14 =_{dec} 28$. The carry flag is set representing the missing $_{dec}16$ of the output.

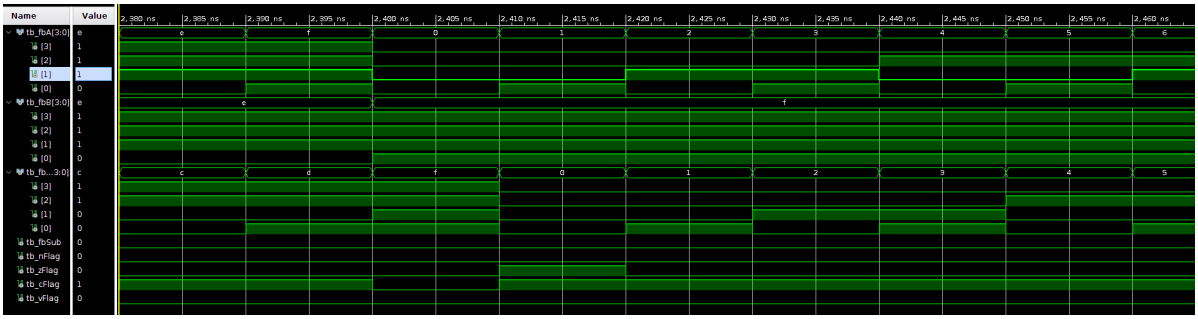

Figure 13: Four-Bit Adder/Subtractor Scope Output with Carry Output and Flag

The overflow flag is set according to the output of an XOR operation on the third and fourth carry bits. This accounts for both a carry changing the fourth sum bit, making it either positive or negative incorrectly, or the carry out from the fourth full adder. As in Figure 14, the overflow flag is set when a subtraction results in a carry in either the third or fourth carry bits.

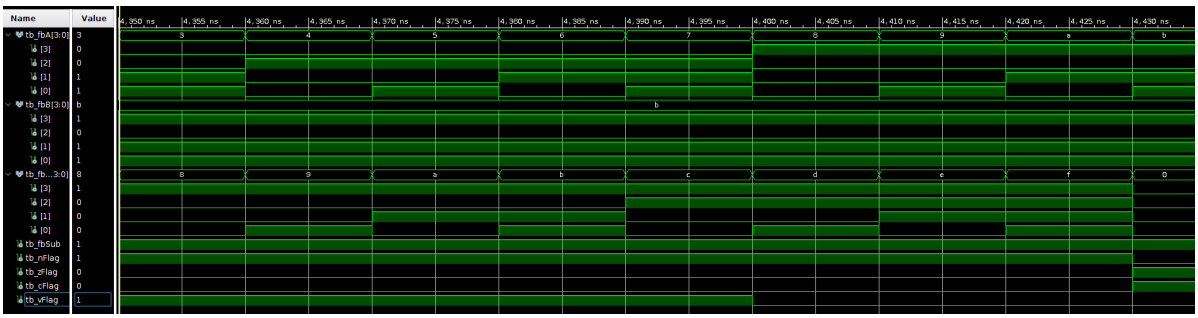

Figure 14: Four-Bit Adder/Subtractor Scope Output with Overflow Output and Flag

Conclusion

The full-adder, four-bit adder/subtracter, and status flags passed all tests, as indicated in the included scope figures. A total of approximately six hours was spent completing the exercises of this assignment. A majority of this time was spent correcting errors, and learning to use features of Vivado, and the VHDL language. Many of the issues which were encountered were due to inexperience with VHDL, and misunderstandings of the operation of the language. One issue which occurred while writing the report was in confusion caused by the hexadecimal outputs of the scope, and how they correspond with the signed binary numbers, which caused doubt in the correctness of the four-bit adder/subtractor outputs.

Once these issues were corrected, the full adder and four-bit adder/subtractor were tested, and determined to be operating as intended. This assignment allowed for the opportunity to improve understanding in VHDL and the concepts of the language. Using the basic concepts of the full adder, and the adder/subtractor, the student can become more familiar with the workings of the language, the assignment of input and output ports, use of signals, and logical operations such as XOR, AND, and OR. It also allows for becoming more familiar with the helpful tools that the Vivado development environment provides, such as the ability to generate schematics, and to simulate test cases and view the outputs on a scope.

Appendix

4.1 One-Bit Full Adder Code

```
-- Company: DCU ECE4
-- Engineer: Michael Lenehan
--
-- Create Date: 08.10.2018 14:35:59
-- Design Name: One Bit Full Adder
-- Module Name: oneBitFullAdder - Behavioral
-- Project Name: EE496 Assignment 1
-- Target Devices: NA
-- Tool Versions: NA
-- Description: The design of a One Bit Full Adder
--              Inputs: a, b, carry in
--              Outputs: sum, carry out
--
-- Dependencies: NA
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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 oneBitFullAdder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           c_in : in STD_LOGIC;
           sum : out STD_LOGIC;
           c_out : out STD_LOGIC);
end oneBitFullAdder;

architecture Behavioral of oneBitFullAdder is
    signal temp: std_logic;
```

```
begin

    temp <= a xor b after 1 ns;
    c_out <= (a and b) or (c_in and temp) after 2 ns;
    sum <= c_in xor temp after 1 ns;

end Behavioral;
```

4.2 Four-Bit Adder/Subtractor Code

```
----------------------------------------------------------------------------------
-- Company: DCU
-- Engineer: Michael Lenehan
--
-- Create Date: 08.10.2018 18:25:35
-- Design Name: Four-Bit Adder/Subtractor
-- Module Name: fourBitAddSub - Behavioral
-- Project Name: Assignment 1
-- Target Devices:
-- Tool Versions:
-- Description: Four-Bit Adder Subtractor
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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 fourBitAddSub is
    port ( fbA, fbB: in std_logic_vector(3 downto 0);
           fbSum: out std_logic_vector(3 downto 0);
           sub : in std_logic);
end fourBitAddSub;

architecture Behavioral of fourBitAddSub is
    signal tempCarry: std_logic := '0';
    signal fbCin, Cout, tempXOR : std_logic_vector(3 downto 0) := (others => '0');
    component fullAdder
        port ( a: in std_logic;
               b: in std_logic;
               c_in: in std_logic;
               sum: out std_logic;
               c_out: out std_logic);
    end component;
begin

tempXOR(0) <= fbB(0) xor sub;
tempXOR(1) <= fbB(1) xor sub;
tempXOR(2) <= fbB(2) xor sub;
tempXOR(3) <= fbB(3) xor sub;
FA0: fullAdder port map(a => fbA(0),
            b => tempXOR(0),
            c_in => sub,
            sum => fbSum(0),
            c_out => Cout(0));
FA1: fullAdder port map(a => fbA(1),
            b => tempXOR(1),
            c_in => Cout(0),
            sum => fbSum(1),
            c_out => Cout(1));
FA2: fullAdder port map(a => fbA(2),
            b => tempXOR(2),
            c_in => Cout(1),
            sum => fbSum(2),
            c_out => Cout(2));
FA3: fullAdder port map(a => fbA(3),
            b => tempXOR(3),
            c_in => Cout(2),
            sum => fbSum(3),
            c_out => Cout(3));
end Behavioral;
```

4.3 Four-Bit Adder/Subtractor with Status Flags Code

```
----------------------------------------------------------------------------------
-- Company: DCU
-- Engineer: Michael Lenehan
--
-- Create Date: 08.10.2018 18:25:35
-- Design Name: Four-Bit Adder/Subtractor with Status Flags
-- Module Name: fourBitAddSub - Behavioral
-- Project Name: Assignment 1
-- Target Devices:
-- Tool Versions:
-- Description: Four-Bit Adder Subtractor with Status Flags
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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 fourBitAddSub is
    port ( fbA, fbB: in std_logic_vector(3 downto 0);
           fbSum: inout std_logic_vector(3 downto 0);
           sub : in std_logic;
           nFlag, zFlag, cFlag, vFlag : out std_logic);
end fourBitAddSub;

architecture Behavioral of fourBitAddSub is
```

```
    signal tempCarry: std_logic := '0';
    signal fbCin, Cout, tempXOR : std_logic_vector(3 downto 0) := (others => '0');
    component fullAdder
        port ( a: in std_logic;
               b: in std_logic;
               c_in: in std_logic;
               sum: out std_logic;
               c_out: out std_logic);
    end component;
begin

tempXOR(0) <= fbB(0) xor sub;
tempXOR(1) <= fbB(1) xor sub;
tempXOR(2) <= fbB(2) xor sub;
tempXOR(3) <= fbB(3) xor sub;
FA0: fullAdder port map(a => fbA(0),
            b => tempXOR(0),
            c_in => sub,
            sum => fbSum(0),
            c_out => Cout(0));
FA1: fullAdder port map(a => fbA(1),
            b => tempXOR(1),
            c_in => Cout(0),
            sum => fbSum(1),
            c_out => Cout(1));
FA2: fullAdder port map(a => fbA(2),
            b => tempXOR(2),
            c_in => Cout(1),
            sum => fbSum(2),
            c_out => Cout(2));
FA3: fullAdder port map(a => fbA(3),
            b => tempXOR(3),
            c_in => Cout(2),
            sum => fbSum(3),
            c_out => Cout(3));
nFlag <= fbSum(3) when sub = '1' else
         '0';
zFlag <= '1' when (fbSum(0) or fbSum(1) or fbSum(2) or fbSum(3)) = '0' else
         '0';
cFlag <= Cout(3);
vFlag <= Cout(3) xor Cout(2);
end Behavioral;
```