Digital Design CSCE 2114

Lab 6 Report

Zack Fravel

November 20th, 2015

zpfravel@uark.edu

Abstract

The purpose of lab 6 was to get more experience in writing VHDL code in order to implement various combinational circuits, as well as learn more about simulations of circuit designs before implementing.

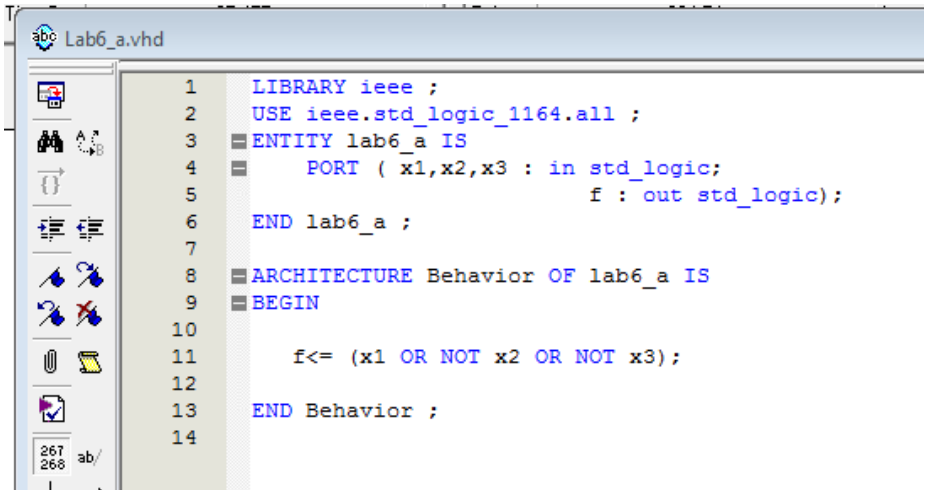
Introduction

The lab consisted of four parts, each dealing with their own specific combinational circuit. We were assigned to implement a regular 3-input function, a flexible 5-bit signed Add/Sub, a decoder, and a 4-1 multiplexer. The main thing we had to keep in mind with writing VHDL in this lab is putting single bits in single quotes and anything more than one bit in double quotes (e.g. '0', '1', "001001", "10011").

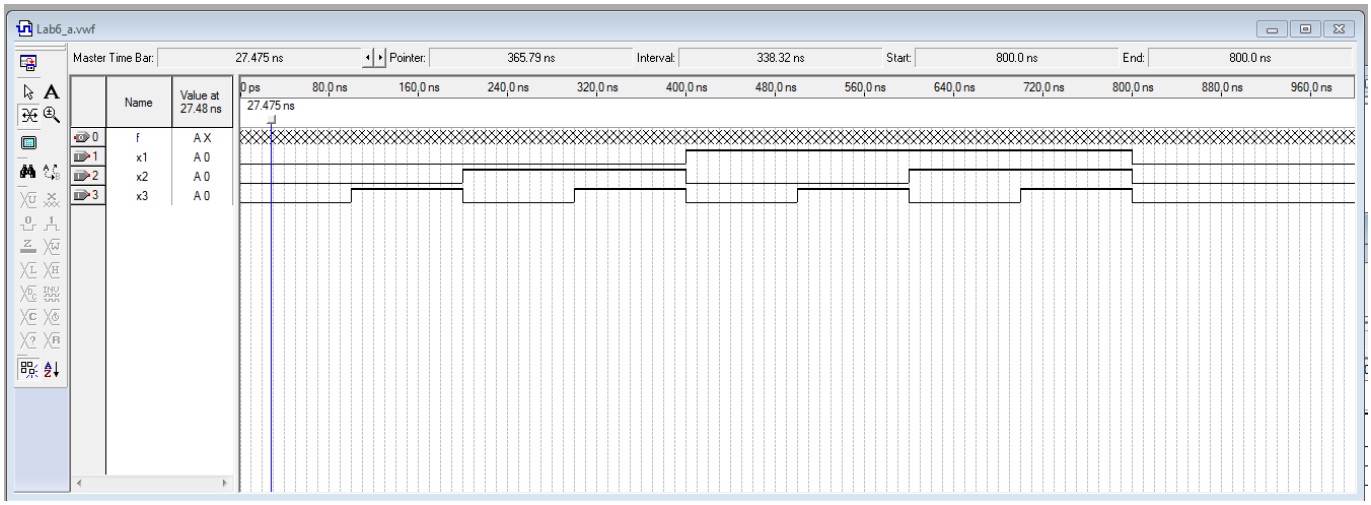
With each part, we start with writing the correct VHDL code to implement the correct circuit function. After writing the code, we set up a waveform file in Quartus so we're able to run a simulation and see different outputs and results.

Design and Implementation

The first part of the lab was to implement a standard 3-input SOP form function. The function was given as $f(x1, x2, x3) = \sum m(0,1,2,4,5,6,7)$. Given this, we are able to draw up a truth table and it becomes relatively straight forward to figure out the correct code.



Once we have the code compile without any errors, we are able to draw up our waveform file in Quartus to select different input values and see the results for different times during the simulation. The following is the waveform file, to the lab specification:

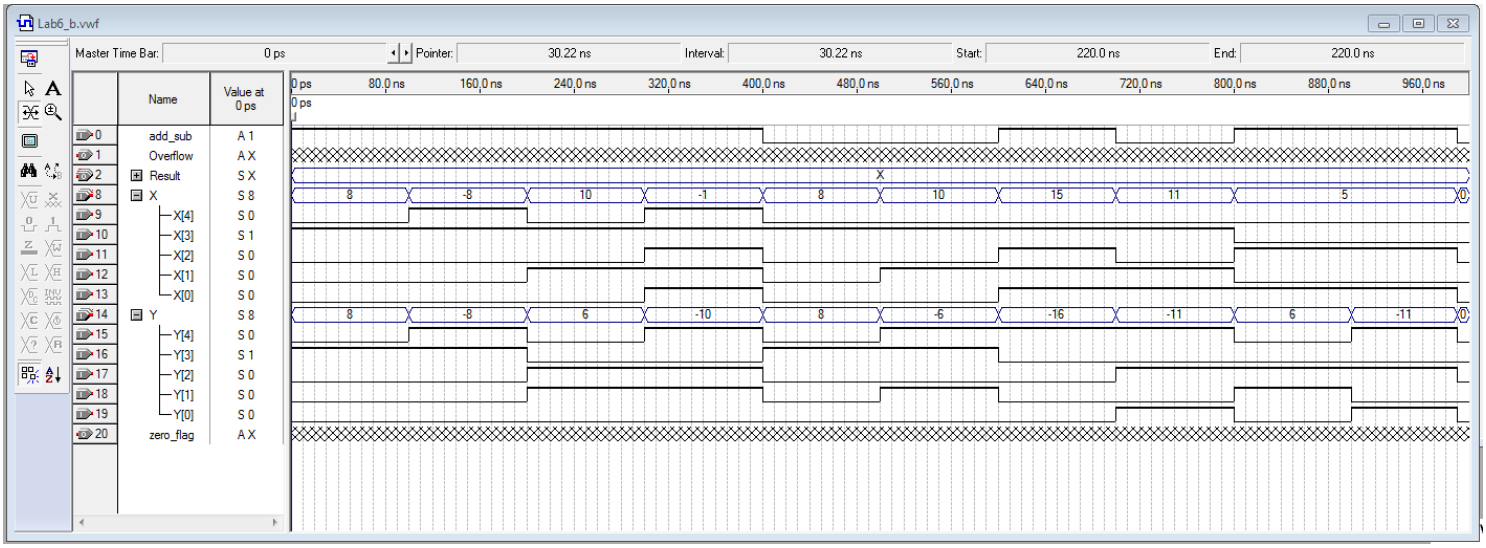


The second part of the lab was designing and implementing a flexible 5-bit signed add/sub with overflow and zero-flag bits. According to the lab, it should be able to both add and subtract and gives the zero-flag and overflow conditions. Zero-flag goes high when the result is zero, overflow goes high when there is a memory overflow to indicate the answer is incorrect. The add_sub bit indicates addition or subtraction.

Lab6_b.vhd

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3   USE ieee.std_logic_unsigned.all ;
4
5   ENTITY lab6_b IS
6   PORT (   add_sub : in std_logic;
7            zero_flag : out std_logic;
8            X, Y   : IN  STD_LOGIC_VECTOR(4 DOWNTO 0) ;
9            Result  : OUT  STD_LOGIC_VECTOR(4 DOWNTO 0) ;
10           Overflow : OUT  STD_LOGIC ) ;
11  END lab6_b ;
12
13  ARCHITECTURE Behavior OF lab6_b IS
14      SIGNAL internal_result : STD_LOGIC_VECTOR(5 DOWNTO 0) ;
15  BEGIN
16
17      internal_result      <= ('0'&X+Y)   when add_sub = '1'     else    ('0'&X-Y);
18      Result               <= internal_result(4 downto 0);
19      Overflow             <= X(4) xor Y(4) xor internal_result(4) xor internal_result(5);
20      zero_flag            <= '1' when internal_result(5 downto 0) = "0" else '0';
21  END Behavior ;
22
```

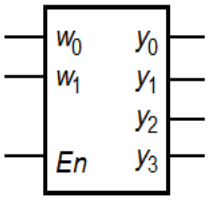
After compiling the code successfully, we were assigned to create a waveform file for this function with the radix being set to signed decimals.



Part C consisted of designing and implementing a 2-4 decoder. The following is the truth table and graphical symbol for a 2-4 decoder.

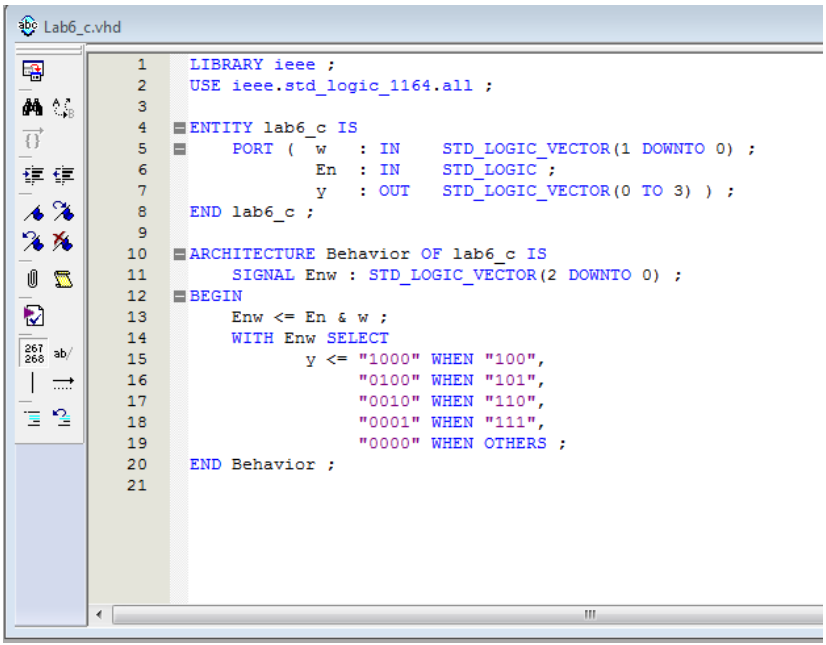
En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table



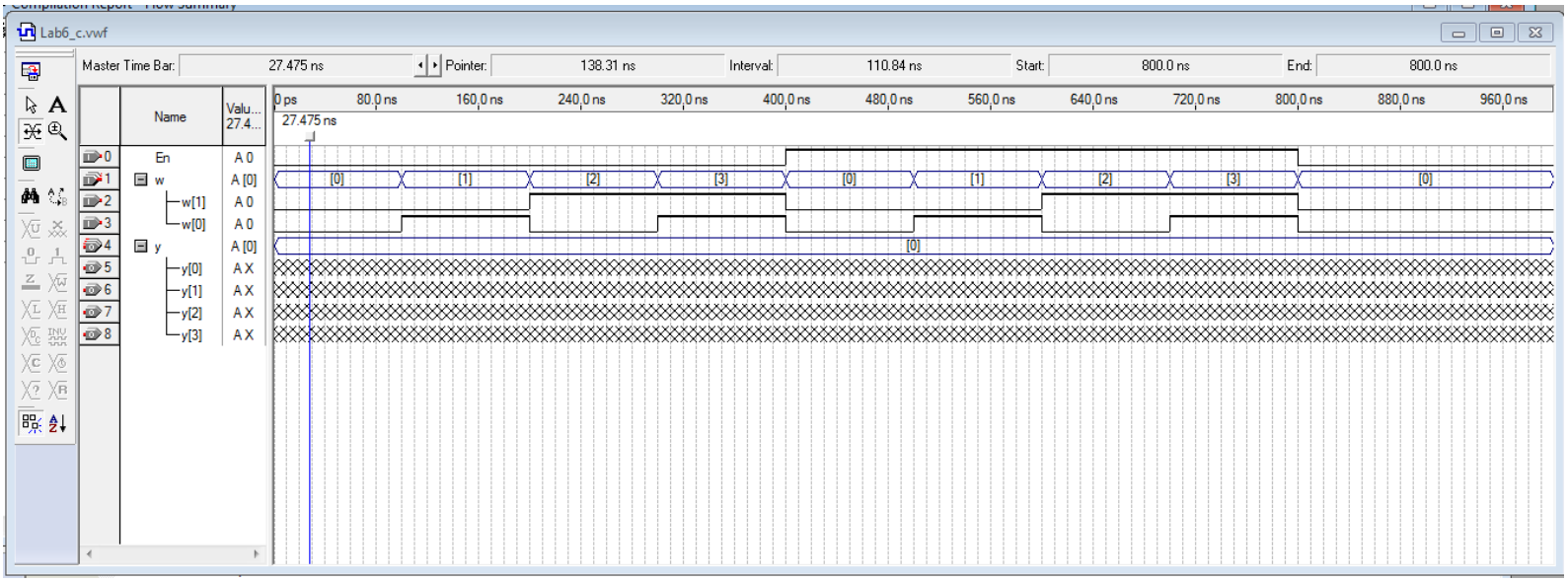
(b) Graphical symbol

With the truth table and partial code given to us, we were able to figure out the complete implementation which is the following:

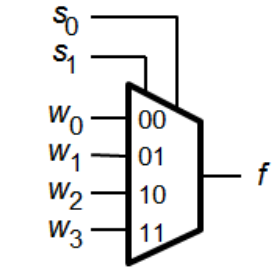



```
1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.all ;
3
4  ENTITY lab6_c IS
5      PORT ( w   : IN    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
6             En  : IN    STD_LOGIC ;
7             y   : OUT   STD_LOGIC_VECTOR(0 TO 3) ) ;
8  END lab6_c ;
9
10 ARCHITECTURE Behavior OF lab6_c IS
11     SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
12 BEGIN
13     Enw <= En & w ;
14     WITH Enw SELECT
15         y <= "1000" WHEN "100",
16              "0100" WHEN "101",
17              "0010" WHEN "110",
18              "0001" WHEN "111",
19              "0000" WHEN OTHERS ;
20 END Behavior ;
21
```

Once again, upon compiling it was time to create a waveform file to output various results.



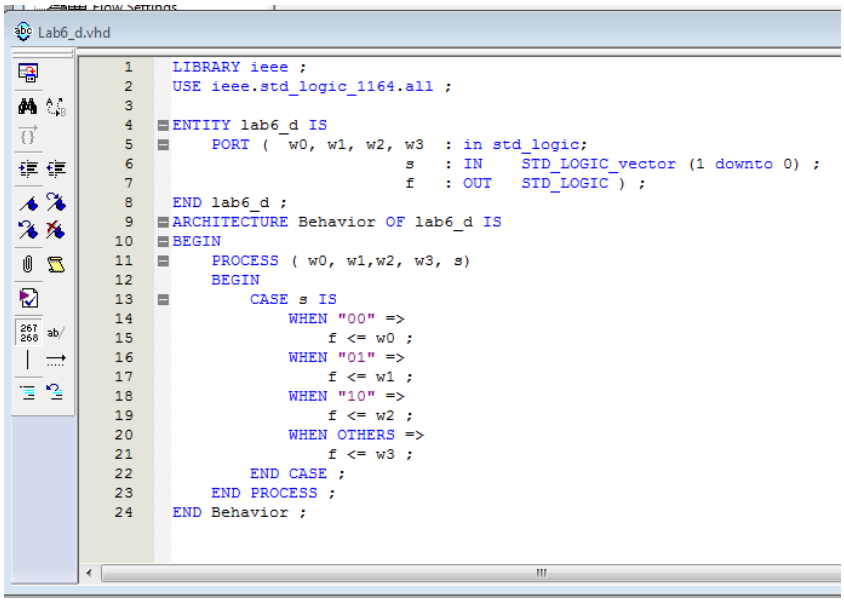
The final part of the lab was the design and implementation of a 4-1 multiplexer. The following is the representation for a 4-1 mux. Using this, it was relatively straight forward to gather all the information needed to write the VHDL code. The following is the VHDL implementation as well as the



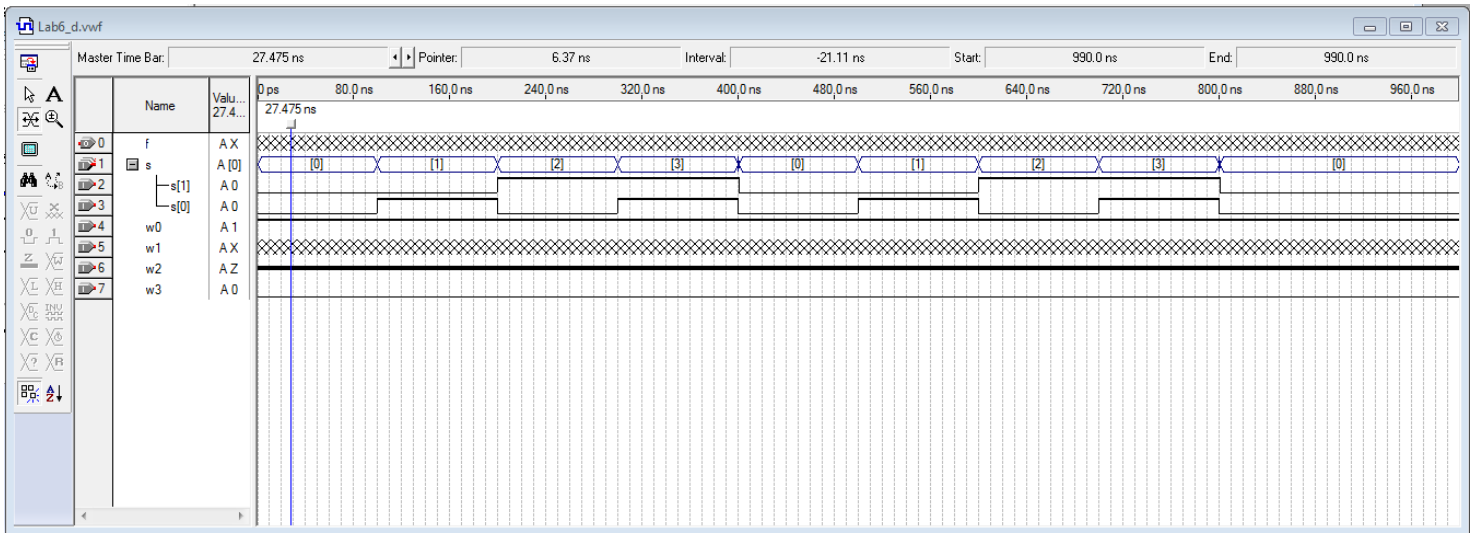
(a) Graphic symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

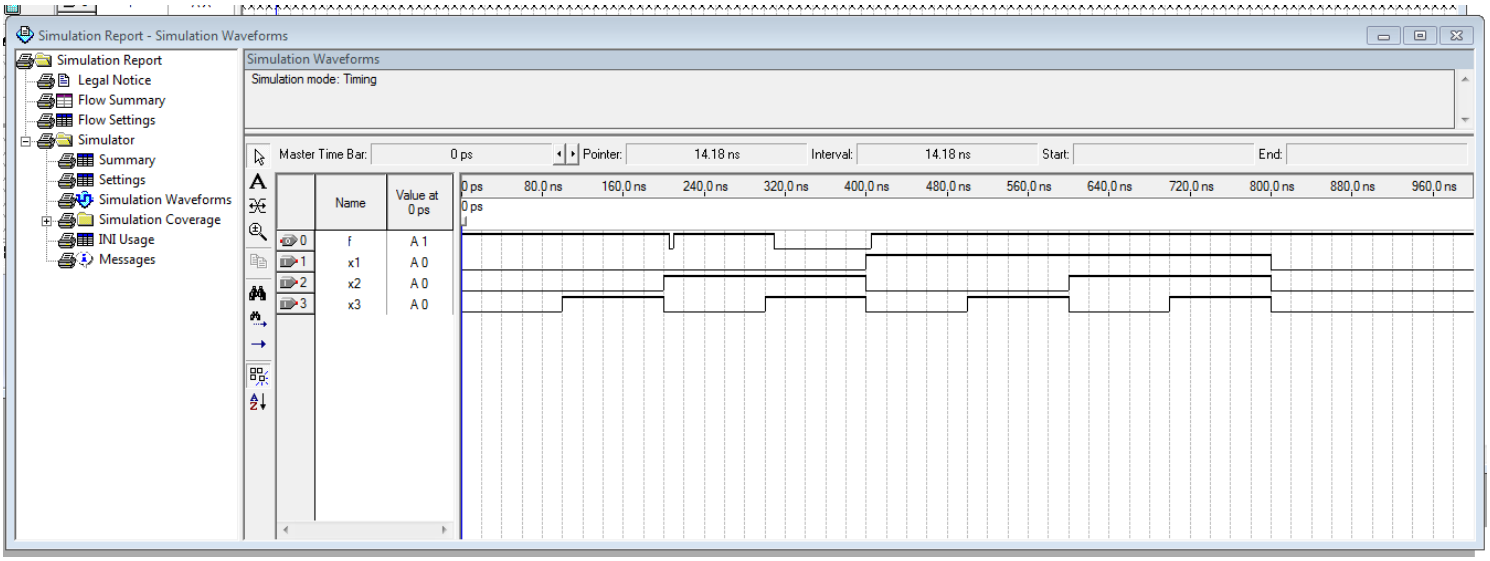


waveform file used to measure different input/output combinations.



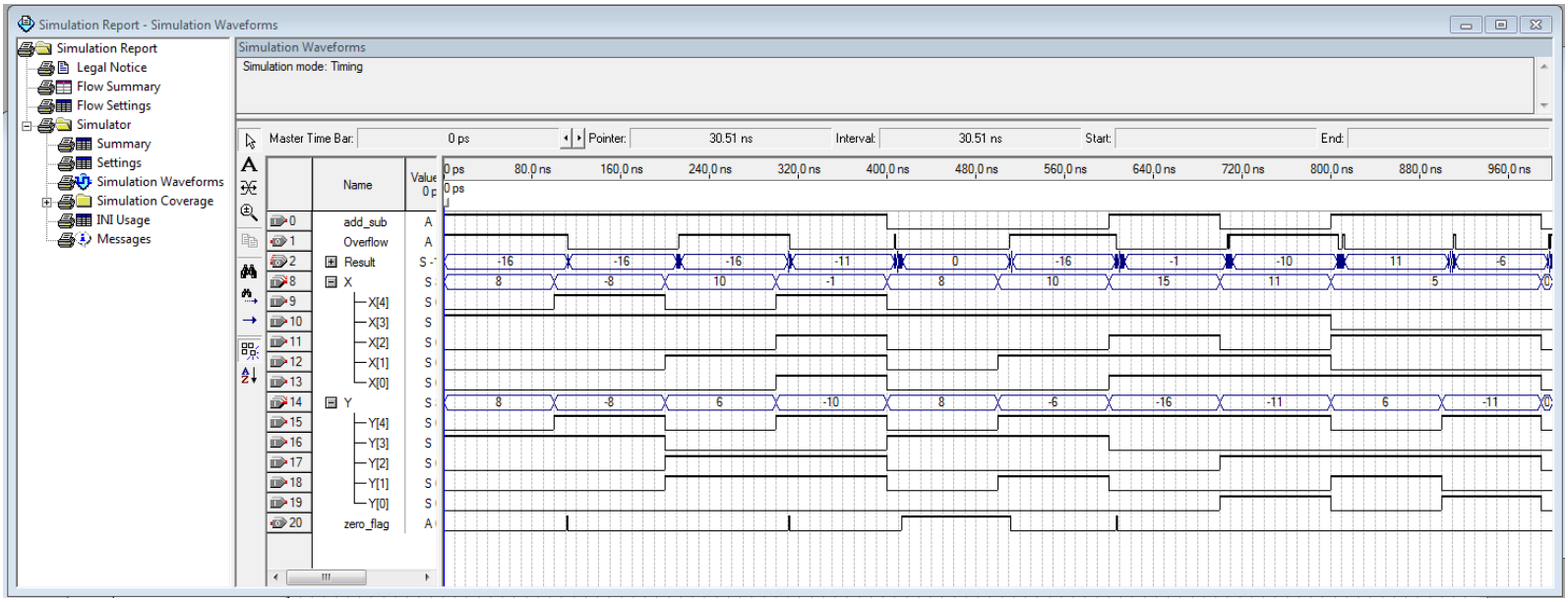
Results

The following are the results of the function from part A.



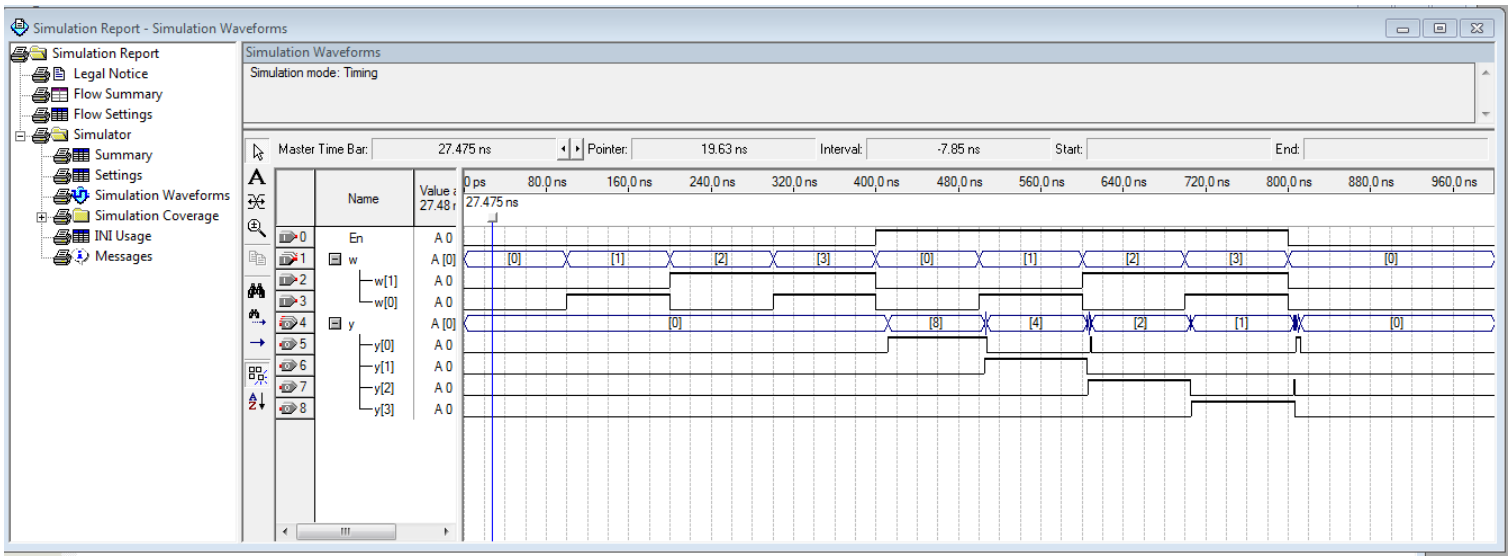
As indicated in the truth table for f, the function is always ‘1’ except when x1’+x2+x3.

Part B was a little more complex, being a 5-bit signed add/sub function. There are separate bits indicating overflow, add/sub, as well as if zero is the output.

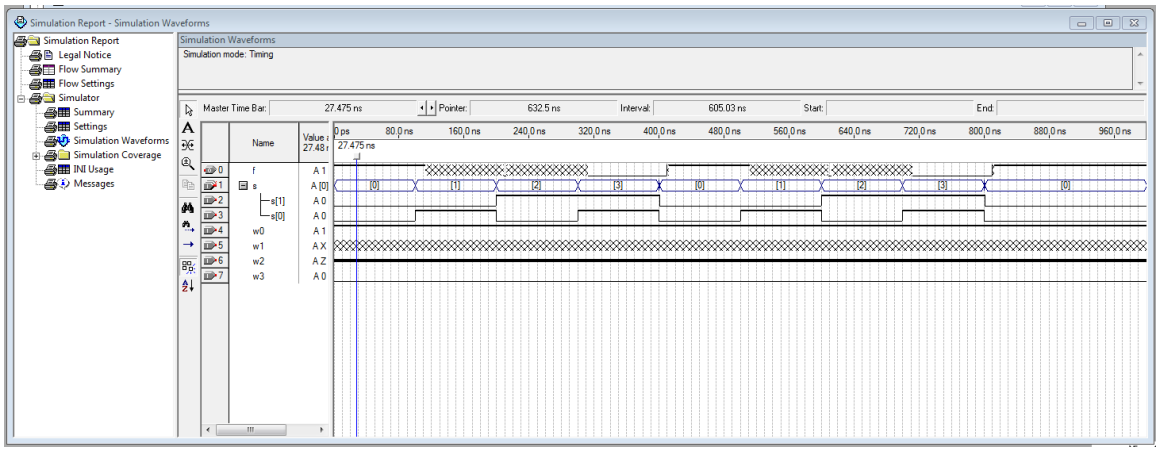


This picture makes it relatively clear to see how the implemented circuit actually works to add or subtract numbers. Part C was implementing a 2-4 decoder, here are the results.

Finally, the last part of the lab was to implement a 4-1 multiplexer. We were given the truth table, from there it was as simple as writing the VHDL and running a simulation.



Here are the results for the 4-1 mux.



Conclusion

This lab made Quartus much more approachable as a piece of software. After implementing these various circuits I feel much more comfortable using the software than a few weeks ago. All the circuits once you have the information you need are relatively straight forward to implement, set inputs for, and simulate.