Lab 3 N BIT ADDER SUBTRACTOR AND COMPARATOR

CSC 343 Fall 2017

September, 24 2017

Jeter Gutierrez

TABLE OF CONTENTS

1. Objective

In this lab we will be implementing a 4 bit, 16 bit and 32 bit comparator that will be capable of determining whether two 4 bit, 16 bit or 32 bit words are equal to each other, will result in an overflow or are negative. We will also be designing 4 bit, 16 bit and 32 bit adders as well as subtractors. After each design we will be using modelsim to test bench the correctness of each design, in some cases we will purposely have en error in our design in order to determine how well modelsim will detect a logical error. The purpose of implementing these circuits is to work better with memory storage and learn how memory works. With these components we would be able to test if our components for storing memory in future labs are storing the appropriate values. We will be designing these circuits in VHDL using Quartus Prime Software. We will be testing the circuits using modelsim waveform simulation. Another method that will be used to test the circuits is testbench with predefined values for each of the n-bit words. After we have completed designing the circuits we will run the circuits in simulation to assert that they are working as expected. We will also design an array of 32 bit words and search for a particular 32 bit word in the array in order to model how memory access works.

The circuits we will be designing in this lab are:

1. 4 BIT ADDER
2. 4 BIT ADDER TEST
3. 4 BIT ADDER SUBTRACTOR

4. 4 BIT ADDER SUBTRACTOR TEST
5. 16 BIT ADDER
6. 16 BIT ADDER TEST
7. 16 BIT ADDER SUBTRACTOR
8. 16 BIT ADDER SUBTRACTOR TEST
9. 32 BIT ADDER
10. 32 BIT ADDER TEST
11. 32 BIT ADDER SUBTRACTOR
12. 32 BIT ADDER SUBTRACTOR TEST
13. 4 BIT COMPARATOR ADVANCED
14. 4 BIT COMPARATOR ADVANCED TEST
15. 16 BIT COMPARATOR ADVANCED
16. 16 BIT COMPARATOR ADVANCED TEST
17. 32 BIT COMPARATOR ADVANCED
18. 32 BIT COMPARATOR ADVANCED TEST
19. ARRAY OF 32 BIT WORDS

2. 4 BIT ADDER

2.1 Functionality and specifications for 4 BIT ADDER

The purpose of the 4 bit adder is to add 2 4 bit words and keep track of the overflow value, the carry output and total sum. The input values will be stored in X and Y which are both 4 bit vectors in vhdl, the sum of the two bits will be stored in sum_prime, the overflow will be stored

in Overflow and the final carry value will be stored in Cout. If a value for the sum of the 2 4 bit words is out of range from the 4 bit word capacity the value of overflow will be 1 or true, this means that the value in sum_prime will be an incorrect value and not useful information. This circuit will be capable of adding 2 4 bit words in signed, unsigned formats correctly keeping track of overflow.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_ADDER_4_BITS is--Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (3 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017
architecture DESIGN of GUTIERREZ_ADDER_4_BITS is --Jeter Gutierrez September 24 ,2017
    component GUTIERREZ_FULL_ADDER_USING_GATES--Jeter Gutierrez September 24 ,2017
        Port ( a,b,cin_prime : in  STD_LOGIC;--Jeter Gutierrez September 24 ,2017
             sum,cout_prime : out  STD_LOGIC);--Jeter Gutierrez September 24 ,2017
    end component;--Jeter Gutierrez September 24 ,2017
    signal C: std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
        FIRST: GUTIERREZ_FULL_ADDER_USING_GATES port map  (cin, X(0),Y(0),sum_prime(0),C(0));--Jeter
        SECOND: Cout<= C(3);--Jeter Gutierrez September 24 ,2017
        THIRD: Overflow<= C(3) xor C(1);--Jeter Gutierrez September 24 ,2017
    FOURTH:--Jeter Gutierrez September 24 ,2017
        for i in 1 to (3) generate--Jeter Gutierrez September 24 ,2017
            FIFTH: GUTIERREZ_FULL_ADDER_USING_GATES port map  (C(i-1), X(i),Y(i),sum_prime(i),C(i));
        end generate;--Jeter Gutierrez September 24 ,2017
end DESIGN;--Jeter Gutierrez September 24 ,2017
```

Figure 1: VHDL code for 4 bit adder.

2.2 Simulation for 4 bit adder

In this simulation X and Y will be given different values, in order to add them together, their result will be sent to sum_prime, we will also keep track of overflow and carry out values, then we will observe the simulation and consider its correctness.

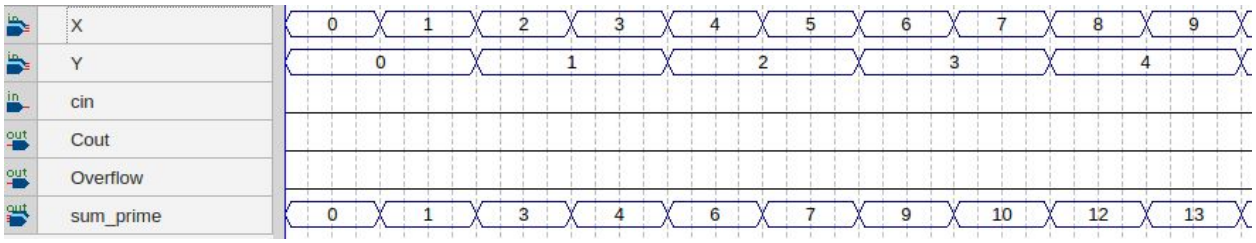

Figure 2: Vector waveform simulation for 4 bit adder. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 4 bit adder.

2.3 Testing incorrect 4 bit adder.

The purpose of this circuit is to perform an error in the structure of the 4 bit adder so that we can see that our testbench code can actually and correctly output an error when we have one based on not having a correct numerical result.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_MISTAKE4_ADDER is--Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (3 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017
--Jeter Gutierrez September 24 ,2017
architecture DESIGN of GUTIERREZ_MISTAKE4_ADDER is --Jeter Gutierrez September 24 ,2017
    component GUTIERREZ_FULL_ADDER_USING_GATES--Jeter Gutierrez September 24 ,2017
        Port ( a,b,cin_prime : in  STD_LOGIC;--Jeter Gutierrez September 24 ,2017
             sum,cout_prime : out  STD_LOGIC);--Jeter Gutierrez September 24 ,2017
    end component;--Jeter Gutierrez September 24 ,2017
    --Jeter Gutierrez September 24 ,2017
    signal C: std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
        FIRST: GUTIERREZ_FULL_ADDER_USING_GATES port map  (cin, X(0),Y(0),sum_prime(0),C(0));--Jeter
        SECOND: Cout<= C(3);--Jeter Gutierrez September 24 ,2017
        THIRD: Overflow<= C(3) xor C(1);--Jeter Gutierrez September 24 ,2017
    FOURTH:--Jeter Gutierrez September 24 ,2017
        for i in 1 to (3) generate--Jeter Gutierrez September 24 ,2017
            FIFTH: GUTIERREZ_FULL_ADDER_USING_GATES port map  (C(i-1), X(i),Y(i),C(i),sum_prime(i));
        end generate;--Jeter Gutierrez September 24 ,2017
end DESIGN;--Jeter Gutierrez September 24 ,2017
```

Figure 3: VHDL code for incorrect 4 bit adder, as we can see the value of sum_prime and the carry values have been switched and are being sent into the wrong ports.

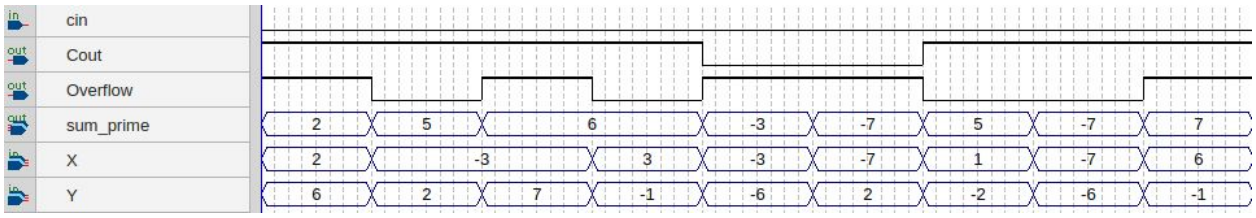

Figure 4: Vector waveform simulation for incorrect 4 bit adder, these sums are not numerically correct but the code still compiles which is why using test bench to test for errors is very useful all of these results are wrong.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_MISTAKE4_ADDER is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_MISTAKE4_ADDER;--Jeter Gutierrez September 24 ,2017
architecture arch_test of GUTIERREZ_TEST_MISTAKE4_ADDER is--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_MISTAKE4_ADDER--Jeter Gutierrez September 24 ,2017
PORT (Cin: IN STD_LOGIC;--Jeter Gutierrez September 24 ,2017
X,Y: IN STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
sum_prime: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
Cout: OUT STD_LOGIC );--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,S :STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT--Jeter Gutierrez September 24 ,2017)
uut: GUTIERREZ_MISTAKE4_ADDER port map (--Jeter Gutierrez September 24 ,2017
Cin => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
sum_prime => S,--Jeter Gutierrez September 24 ,2017
Cout =>Co--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Proces--Jeter Gutierrez September 24 ,2017s
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 4 BIT ADDER WITH MISTAKE";--Jeter Gutierrez September 24 ,2017
A<="0000";--Jeter Gutierrez September 24 ,2017
B<="0000";--Jeter Gutierrez September 24 ,2017
Ci<='0';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 16 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 16 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,20
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez S
assert (S = A+B) report "The sum from 4 bit adder is S= " & integer'image(to_integer(unsigned((S)))) &-
" while the expected A+B = " & integer'image(to_integer(unsigned((A+B)))) severity ERROR;--Jeter Gutier
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"0001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"0001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
---END User Defined Process--Jeter Gutierrez September 24 ,2017
end arch_test;--Jeter Gutierrez September 24 ,2017
```

Figure 5: VHDL code for test bench of incorrect 4 bit adder, this code will be looping through all

possible combinations of 4 bit words and adding them to other 4 bit words but since our code is incorrect we expect to find an error.

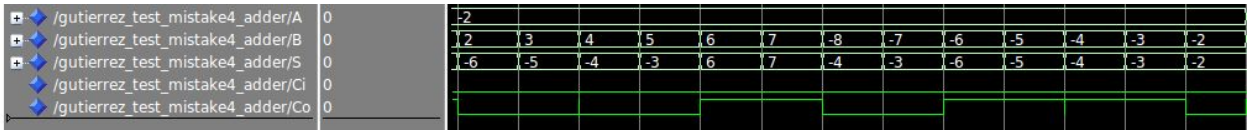

Figure 6: Vector waveform simulation for test bench of incorrect 4 bit adder, we can see we still get results they just are not correct and we did this on purpose to see if modelsim will find the error.

```
# ** Error: The sum from 4 bit adder is S= 8 while the expected A+B = 12
#    Time: 2950 ns  Iteration: 0  Instance: /gutierrez_test_mistake4_adder
# ** Error: The sum from 4 bit adder is S= 9 while the expected A+B = 13
#    Time: 2960 ns  Iteration: 0  Instance: /gutierrez_test_mistake4_adder
# ** Error: The sum from 4 bit adder is S= 4 while the expected A+B = 14
#    Time: 2970 ns  Iteration: 0  Instance: /gutierrez_test_mistake4_adder
# ** Error: The sum from 4 bit adder is S= 5 while the expected A+B = 15
```

Figure 7: Proof of modelsim detecting an error in our incorrect 4 bit adder, this means we were able to successfully prove that modelsim is useful in detecting errors and that we were able to correctly design testbench code for the incorrect 4 bit adder.

3. 4 BIT ADDER TEST

3.1 Functionality and specifications for 4 bit adder test.

The purpose of this circuit is to create a test bench in order to simulate our 4 bit adder. We already simulated our 4 bit adder in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_4_BIT_ADDER is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_4_BIT_ADDER;--Jeter Gutierrez September 24 ,2017
architecture arch_test of GUTIERREZ_TEST_4_BIT_ADDER is--Jeter Gutierrez September 24 ,2017
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADDER_4_BITS--Jeter Gutierrez September 24 ,2017
PORT (Cin: IN STD_LOGIC;--Jeter Gutierrez September 24 ,2017
X,Y: IN STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
sum_prime: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
Cout,Overflow: OUT STD_LOGIC );--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,S :STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT--Jeter Gutierrez September 24 ,2017)
uut: GUTIERREZ_ADDER_4_BITS port map (--Jeter Gutierrez September 24 ,2017
Cin => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
sum_prime => S,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Process--Jeter Gutierrez September 24 ,2017
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 4 BIT ADDER";--Jeter Gutierrez September 24 ,2017
A<="0000";--Jeter Gutierrez September 24 ,2017
B<="0000";--Jeter Gutierrez September 24 ,2017
Ci<='0';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 16 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 16 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,20
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez S
assert (S = A+B) report "The sum from 4 bit adder is S= " & integer'image(to_integer(unsigned((S)))) &-
" while the expected A+B = " & integer'image(to_integer(unsigned((A+B)))) severity ERROR;--Jeter Gutier
assert(Overflow2=Overflow2)report "Overflow is wrong"severity ERROR;--Jeter Gutierrez September 24 ,201
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"0001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"0001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
---END User Defined Process--Jeter Gutierrez September 24 ,2017
end arch_test;--Jeter Gutierrez September 24 ,2017
```

Figure 8: VHDL code for the test bench for testing 4 bit adder. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2 4 bit words by increasing each one by 1 bit and determine whether for every possible case of

adding 2 4 bit words after a certain period of time or after testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 4 bit adder.

3.2 Simulation for 4 bit adder Test.

In this simulation we will be using modelsim to run our test bench file for the 4 bit adder, if we get no errors we know we have designed a 4 bit adder correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

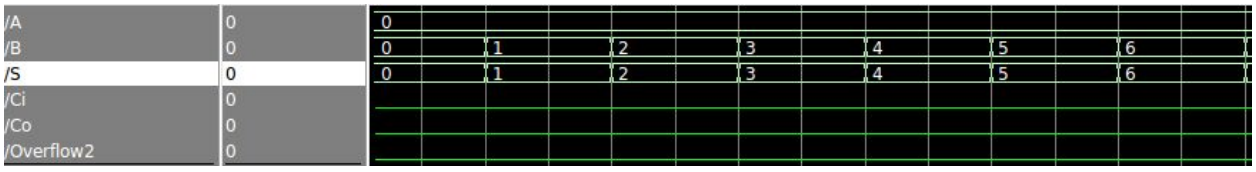

Figure 9: Vector waveform simulation for for test bench of 4 bit adder test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 4 bit adder test. The test bench here was used to test every possible combination of 4-Bit words.

4. 4 BIT ADDER SUBTRACTOR

4.1 Functionality and specifications for 4 bit adder subtractor.

The purpose of the 4 bit adder subtractor is to add or subtract 2 4 bit words and keep track of the overflow value, the carry output and total sum or difference. The input values will be stored in X and Y which are both 4 bit vectors in vhdl, the sum of the two bits will be stored in S, the overflow will be stored in Overflow and the final carry value will be stored in Cout. If a value for

the sum or difference of the 2 4 bit words is out of range from the 4 bit word capacity the value of overflow will be 1 or true, this means that the value in S will be an incorrect value and not useful information. This circuit will be capable of adding 2 4 bit words in signed, unsigned formats correctly keeping track of overflow. It will also be capable of subtracting 2 4 bit words when the signal for SUBTRACT is 1, when it is 0 the circuit will add the 2 4 bit words. The idea behind keeping track of overflow is considering that adding 2 positive numbers should ideally result in another positive number, and that subtracting two negative numbers should result in a negative number as well, that is how we keep track of overflow in the design of this 4 bit adder subtractor.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_ADD_SUB_4_BITS is--Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
             S: out std_logic_vector (3 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017
architecture STRUCTURE of GUTIERREZ_ADD_SUB_4_BITS is --Jeter Gutierrez September 24 ,201
component GUTIERREZ_ADDER_4_BITS is --Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (3 downto 0);--Jeter Gutierrez September 24
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
    signal COMPLEMENT: std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
    SECOND:--Jeter Gutierrez September 24 ,2017
        for i in 0 to (3) generate--Jeter Gutierrez September 24 ,2017
            INVERSE: COMPLEMENT(i) <= Y(i) xor SUBTRACT;--Jeter Gutierrez September 24 ,2
        end generate;--Jeter Gutierrez September 24 ,2017
    THIRD : GUTIERREZ_ADDER_4_BITS port map(SUBTRACT, X, COMPLEMENT, S, Cout, Overflow);-
end STRUCTURE;--Jeter Gutierrez September 24 ,2017
```

Figure 10: VHDL code for 4 bit adder subtractor.

4.2 Simulation for 4 bit adder Subtractor

In this simulation X and Y will be given different values, in order to add them together or subtract them from each other based on the value of SUBTRACT, their result will be sent to S,

we will also keep track of overflow and carry out values, then we will observe the simulation and consider its correctness.

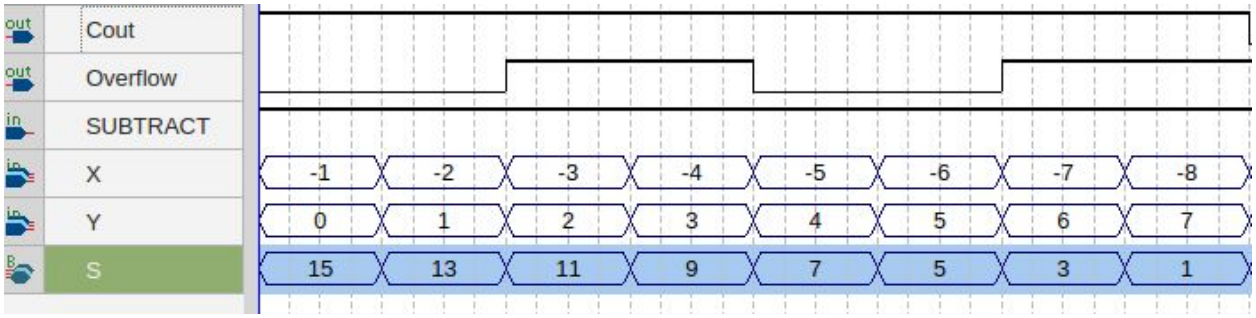

Figure 11: Vector waveform simulation for 4 bit adder subtractor. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 4 bit adder subtractor.

5. 4 BIT ADDER SUBTRACTOR TEST

5.1 Functionality and specifications for 4 bit adder subtractor test.

The purpose of this circuit is to create a test bench in order to simulate our 4 bit adder subtractor. We already simulated our 4 bit adder subtractor in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_4_BIT_ADD_SUB is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_4_BIT_ADD_SUB;--Jeter Gutierrez September 24 ,2017
architecture arch_test of GUTIERREZ_TEST_4_BIT_ADD_SUB is--Jeter Gutierrez September 24 ,2017
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADD_SUB_4_BITS--Jeter Gutierrez September 24 ,2017
PORT (SUBTRACT: IN STD_LOGIC;--Jeter Gutierrez September 24 ,2017
X,Y: IN STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
S: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
Cout,Overflow: OUT STD_LOGIC );--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,Sum_temp :STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT)--Jeter Gutierrez September 24 ,2017
uut: GUTIERREZ_ADD_SUB_4_BITS port map (--Jeter Gutierrez September 24 ,2017
SUBTRACT => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
S => Sum_temp,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Process--Jeter Gutierrez September 24 ,2017
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 4 BIT SUBTRACTOR";--Jeter Gutierrez September 24 ,2017
A<="0000";--Jeter Gutierrez September 24 ,2017
B<="0000";--Jeter Gutierrez September 24 ,2017
Ci<='1';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 16 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 16 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez September 24 ,2017
assert (Sum_temp = A-B) report "The DIFFERENCE from 4 bit SUBTRACTOR is S= " & integer'image(to_integer(signed((Sum_temp)))) &
" while the expected A-B = " & integer'image(to_integer(signed((A-B)))) severity ERROR;--Jeter Gutierrez September 24 ,2017
assert(overflow2=overflow2) report "Overflow is wrong" severity ERROR;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"0001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"0001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
---END User Defined Process--Jeter Gutierrez September 24 ,2017
end arch_test;--Jeter Gutierrez September 24 ,2017
```

Figure 12: VHDL code for the test bench for testing 4 bit adder subtractor. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2 4 bit words by increasing each one by 1 bit and determine whether for every possible case of adding or subtracting 2 4 bit words after a certain period of time or after testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 4 bit adder subtractor.

5.2 Simulation for 4 bit adder subtractor Test.

In this simulation we will be using modelsim to run our test bench file for the 4 bit adder subtractor, if we get no errors we know we have designed a 4 bit adder subtractor correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

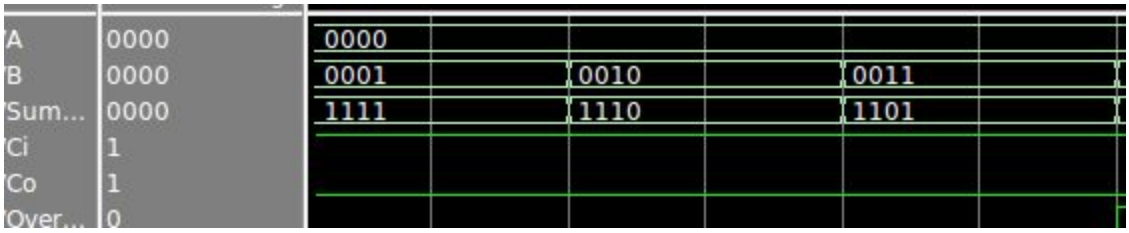

Figure 13: Vector waveform simulation for for test bench of 4 bit adder subtractor test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 4 bit adder subtractor test. The test bench here was used to test every possible combination of 4-Bit words.

6. 16 BIT ADDER

6.1 Functionality and specifications for 16 bit adder.

The purpose of the 16 bit adder is to add 2 16 bit words and keep track of the overflow value, the carry output and total sum. The input values will be stored in X and Y which are both 16 bit vectors in vhdl, the sum of the two bits will be stored in sum_prime, the overflow will be stored in Overflow and the final carry value will be stored in Cout. If a value for the sum of the 2 16 bit words is out of range from the 16 bit word capacity the value of overflow will be 1 or true, this means that the value in sum_prime will be an incorrect value and not useful information. This

circuit will be capable of adding 2 16 bit words in signed, unsigned formats correctly keeping track of overflow.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_ADDER_16_BITS is--Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (15 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017

architecture DESIGN_16 of GUTIERREZ_ADDER_16_BITS is --Jeter Gutierrez September 24 ,2017
    component GUTIERREZ_FULL_ADDER_USING_GATES--Jeter Gutierrez September 24 ,2017
        Port ( a,b,cin_prime : in  STD_LOGIC;--Jeter Gutierrez September 24 ,2017
             sum,cout_prime : out  STD_LOGIC);--Jeter Gutierrez September 24 ,2017
    end component;--Jeter Gutierrez September 24 ,2017

    signal C: std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
    signal sum_temp: std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
        FIRST: GUTIERREZ_FULL_ADDER_USING_GATES port map  (cin, X(0),Y(0),sum_temp(0),C(0));--Jeter G
        SECOND: Cout<= C(15);--Jeter Gutierrez September 24 ,2017
    THIRD:--Jeter Gutierrez September 24 ,2017
        for i in 1 to (15) generate--Jeter Gutierrez September 24 ,2017
            FOURTH: GUTIERREZ_FULL_ADDER_USING_GATES port map  (C(i-1), X(i),Y(i),sum_temp(i),C(i));-
        end generate;--Jeter Gutierrez September 24 ,2017
        FIFTH: Overflow<= '1' WHEN ((X(15)=Y(15) AND sum_temp(15)/=X(15))) ELSE '0';--Jeter Gutierrez
        SIXTH: sum_prime<=sum_temp;--Jeter Gutierrez September 24 ,2017
end DESIGN_16;--Jeter Gutierrez September 24 ,2017
```

Figure 14: VHDL code for 16 bit adder.

6.2 Simulation for 16 bit adder

In this simulation X and Y will be given different values, in order to add them together, their result will be sent to sum_prime, we will also keep track of overflow and carry out values, then we will observe the simulation and consider its correctness.

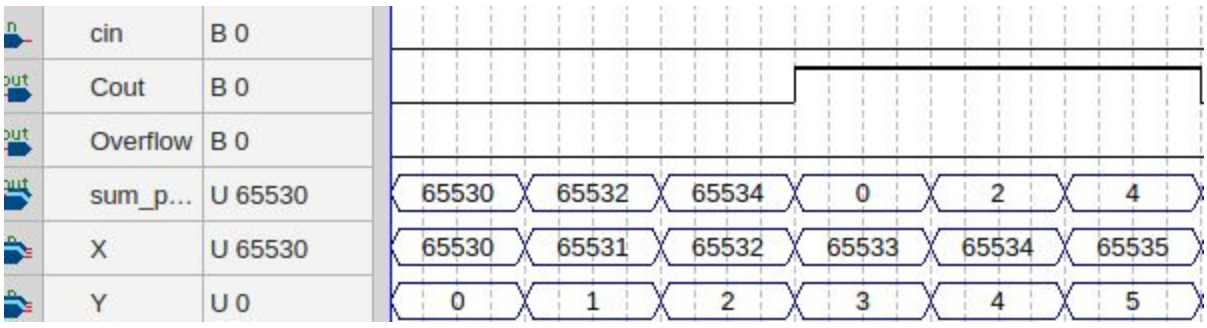

Figure 15: Vector waveform simulation for 16 bit adder. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 16 bit adder.

7. 16 BIT ADDER TEST

7.1 Functionality and specifications for 16 bit adder test.

The purpose of this circuit is to create a test bench in order to simulate our 16 bit adder. We already simulated our 16 bit adder in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_16_BIT_ADDER is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_16_BIT_ADDER;--Jeter Gutierrez September 24 ,2017
architecture arch_test_16 of GUTIERREZ_TEST_16_BIT_ADDER is--Jeter Gutierrez September 24 ,2017
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADDER_16_BITS--Jeter Gutierrez September 24 ,2017
PORT (Cin: IN STD_LOGIC;--Jeter Gutierrez September 24 ,2017
X,Y: IN STD_LOGIC_VECTOR(15 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
sum_prime: OUT STD_LOGIC_VECTOR(15 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
Cout,Overflow: OUT STD_LOGIC );--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,S :STD_LOGIC_VECTOR(15 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT)--Jeter Gutierrez September 24 ,2017
uut: GUTIERREZ_ADDER_16_BITS port map (--Jeter Gutierrez September 24 ,2017
Cin => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
sum_prime => S,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Process--Jeter Gutierrez September 24 ,2017
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 16 BIT ADDER";--Jeter Gutierrez September 24 ,2017
A<="0000000000000000";--Jeter Gutierrez September 24 ,2017
B<="0000000000000000";--Jeter Gutierrez September 24 ,2017
Ci<='0';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,201
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez Se
assert (S = A+B) report "The sum from 16 bit adder is S= " & integer'image(to_integer(unsigned((S)))) &-
" while the expected A+B = " & integer'image(to_integer(unsigned((A+B)))) severity ERROR;--Jeter Gutierr
assert(overflow2=overflow2) report "Overflow is wrong" severity ERROR;--Jeter Gutierrez September 24 ,20
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"0000000000000001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"0000000000000001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
---END User Defined Process--Jeter Gutierrez September 24 ,2017
end arch_test_16;--Jeter Gutierrez September 24 ,2017
```

Figure 16: VHDL code for the test bench for testing 16 bit adder. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2

16 bit words by increasing each one by 1 bit and determine whether for every possible case of adding 2 16 bit words after a certain period of time or after testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 16 bit adder.

7.2 Simulation for 16 bit adder Test.

In this simulation we will be using modelsim to run our test bench file for the 16 bit adder, if we get no errors we know we have designed a 16 bit adder correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

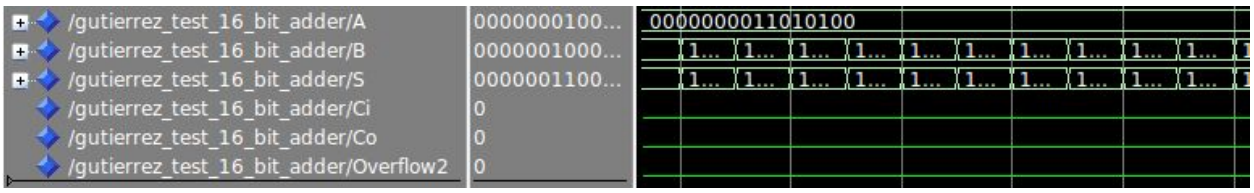

Figure 17: Vector waveform simulation for for test bench of 16 bit adder test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 16 bit adder test. The test bench here was used to test every possible combination of 16-Bit words.

8. 16 BIT ADDER SUBTRACTOR

8.1 Functionality and specifications for 16 bit adder subtractor.

The purpose of the 16 bit adder subtractor is to add or subtract 2 16 bit words and keep track of the overflow value, the carry output and total sum or difference. The input values will be stored in X and Y which are both 16 bit vectors in vhdl, the sum of the two bits will be stored in S, the

overflow will be stored in Overflow and the final carry value will be stored in Cout. If a value for the sum or difference of the 2 16 bit words is out of range from the 16 bit word capacity the value of overflow will be 1 or true, this means that the value in S will be an incorrect value and not useful information. This circuit will be capable of adding 2 16 bit words in signed, unsigned formats correctly keeping track of overflow. It will also be capable of subtracting 2 16 bit words when the signal for SUBTRACT is 1, when it is 0 the circuit will add the 2 16 bit words. The idea behind keeping track of overflow is considering that adding 2 positive numbers should ideally result in another positive number, and that subtracting two negative numbers should result in a negative number as well, that is how we keep track of overflow in the design of this 16 bit adder subtractor.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all; --Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_ADD_SUB_16_BITS--Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
             S: out std_logic_vector (15 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017
architecture STRUCTURE_16 of GUTIERREZ_ADD_SUB_16_BITS--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADDER_16_BITS--Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (15 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
    signal COMPLEMENT: std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
    signal overflow2: std_logic;--Jeter Gutierrez September 24 ,2017
    signal SUM_TEMP: std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
    SECOND:--Jeter Gutierrez September 24 ,2017
        for i in 0 to (15) generate--Jeter Gutierrez September 24 ,2017
            INVERSE: COMPLEMENT(i) <= Y(i) xor SUBTRACT;--Jeter Gutierrez September 24 ,2017
        end generate;--Jeter Gutierrez September 24 ,2017
    THIRD : GUTIERREZ_ADDER_16_BITS port map(SUBTRACT, X, COMPLEMENT, SUM_TEMP, Cout, Overflow2);--Jete
    Overflow<= '1' WHEN (SUBTRACT='0' AND X(15)=COMPLEMENT(15) AND SUM_TEMP(15)/=X(15)) or--Jeter Gutie
                          (SUBTRACT='1' AND X(15)/=COMPLEMENT(15) AND SUM_TEMP(15)/=X(15)) ELSE '0';-
    S<=SUM_TEMP;--Jeter Gutierrez September 24 ,2017
end STRUCTURE_16;--Jeter Gutierrez September 24 ,2017
```

Figure 18: VHDL code for 16 bit adder subtractor.

8.2 Simulation for 16 bit adder Subtractor

In this simulation X and Y will be given different values, in order to add them together or subtract them from each other based on the value of SUBTRACT, their result will be sent to S, we will also keep track of overflow and carry out values, then we will observe the simulation and consider its correctness.

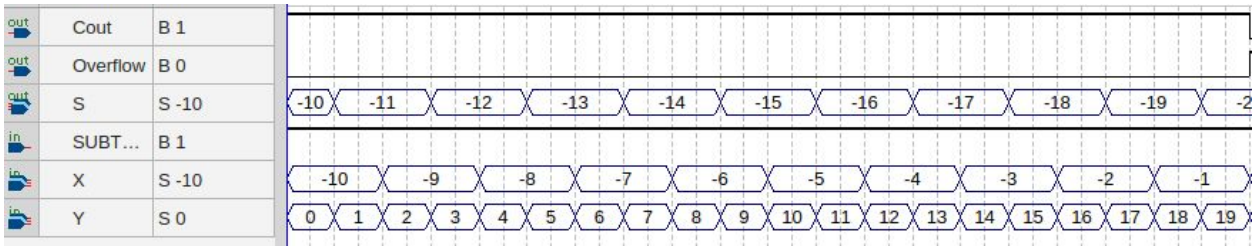

Figure 19: Vector waveform simulation for 16 bit adder subtractor. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 16 bit adder subtractor.

9. 16 BIT ADDER SUBTRACTOR TEST

9.1 Functionality and specifications for 16 bit adder subtractor test.

The purpose of this circuit is to create a test bench in order to simulate our 16 bit adder subtractor. We already simulated our 16 bit adder subtractor in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_16_BIT_ADD_SUB is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_16_BIT_ADD_SUB;--Jeter Gutierrez September 24 ,2017
architecture arch_test of GUTIERREZ_TEST_16_BIT_ADD_SUB is--Jeter Gutierrez September 24 ,2017
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADD_SUB_16_BITS--Jeter Gutierrez September 24 ,2017
PORT (SUBTRACT: IN STD_LOGIC;--Jeter Gutierrez September 24 ,2017
X,Y: IN STD_LOGIC_VECTOR(15 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
S: OUT STD_LOGIC_VECTOR(15 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
Cout,Overflow: OUT STD_LOGIC );--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,Sum_temp :STD_LOGIC_VECTOR(15 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT)--Jeter Gutierrez September 24 ,2017
uut: GUTIERREZ_ADD_SUB_16_BITS port map (--Jeter Gutierrez September 24 ,2017
SUBTRACT => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
S => Sum_temp,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Process--Jeter Gutierrez September 24 ,2017
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 16 BIT SUBTRACTOR";--Jeter Gutierrez September 24 ,2017
A<="0000000000000000";--Jeter Gutierrez September 24 ,2017
B<="0000000000000000";--Jeter Gutierrez September 24 ,2017
Ci<='1';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,2017
---The statement below checks for ALL possible input values if the ouput is correct--Jeter Gutierrez September 24 ,2017.
assert (Sum_temp = A-B) report "The DIFFERENCE BETWEEN A AND B  is S= " & integer'image(to_integer(signed((Sum_temp)))) &
" while the expected A-B = " & integer'image(to_integer(signed((A-B)))) severity ERROR;--Jeter Gutierrez September 24 ,20
assert (overflow2=overflow2) report "Overflow is wrong" severity ERROR;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"0000000000000001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"0000000000000001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
---END User Defined Process--Jeter Gutierrez September 24 ,2017
end arch_test;--Jeter Gutierrez September 24 ,2017
```

Figure 20: VHDL code for the test bench for testing 16 bit adder subtractor. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2 16 bit words by increasing each one by 1 bit and determine whether for every possible case of adding or subtracting 2 16 bit words after a certain period of time or after

testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 16 bit adder subtractor.

9.2 Simulation for 16 bit adder subtractor Test.

In this simulation we will be using modelsim to run our test bench file for the 16 bit adder subtractor, if we get no errors we know we have designed a 16 bit adder subtractor correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

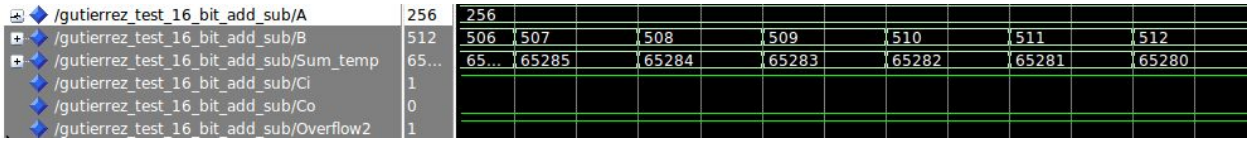

Figure 21: Vector waveform simulation for for test bench of 16 bit adder subtractor test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 16 bit adder subtractor test. The test bench here was used to test every possible combination of 16-Bit words.

10. 32 BIT ADDER

10.1 Functionality and specifications for 32 bit adder.

The purpose of the 32 bit adder is to add 2 32 bit words and keep track of the overflow value, the carry output and total sum. The input values will be stored in X and Y which are both 32 bit vectors in vhdl, the sum of the two bits will be stored in sum_prime, the overflow will be stored in Overflow and the final carry value will be stored in Cout. If a value for the sum of the 2 32 bit words is out of range from the 32 bit word capacity the value of overflow will be 1 or true, this

means that the value in sum_prime will be an incorrect value and not useful information. This circuit will be capable of adding 2 32 bit words in signed, unsigned formats correctly keeping track of overflow.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_ADDER_32_BITS is--Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (31 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017

architecture DESIGN_32 of GUTIERREZ_ADDER_32_BITS is --Jeter Gutierrez September 24 ,2017
    component GUTIERREZ_FULL_ADDER_USING_GATES--Jeter Gutierrez September 24 ,2017
        Port ( a,b,cin_prime : in  STD_LOGIC;--Jeter Gutierrez September 24 ,2017
             sum,cout_prime : out  STD_LOGIC);--Jeter Gutierrez September 24 ,2017
    end component;--Jeter Gutierrez September 24 ,2017

    signal C: std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
    signal sum_temp: std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
        FIRST: GUTIERREZ_FULL_ADDER_USING_GATES port map  (cin, X(0),Y(0),sum_temp(0),C(0));--Jeter (
        SECOND: Cout<= C(31);--Jeter Gutierrez September 24 ,2017
        --Jeter Gutierrez September 24 ,2017
    THIRD:--Jeter Gutierrez September 24 ,2017
        for i in 1 to (31) generate--Jeter Gutierrez September 24 ,2017
            FOURTH: GUTIERREZ_FULL_ADDER_USING_GATES port map  (C(i-1), X(i),Y(i),sum_temp(i),C(i));-
        end generate;--Jeter Gutierrez September 24 ,2017
        FIFTH: Overflow<= '1' WHEN ((X(31)=Y(31) AND sum_temp(31)/=X(31))) ELSE '0';--Jeter Gutierrez
        FINAL: sum_prime<=sum_temp;--Jeter Gutierrez September 24 ,2017
end DESIGN_32;--Jeter Gutierrez September 24 ,2017
```

Figure 22: VHDL code for 32 bit adder.

10.2 Simulation for 32 bit adder

In this simulation X and Y will be given different values, in order to add them together, their result will be sent to sum_prime, we will also keep track of overflow and carry out values, then we will observe the simulation and consider its correctness.

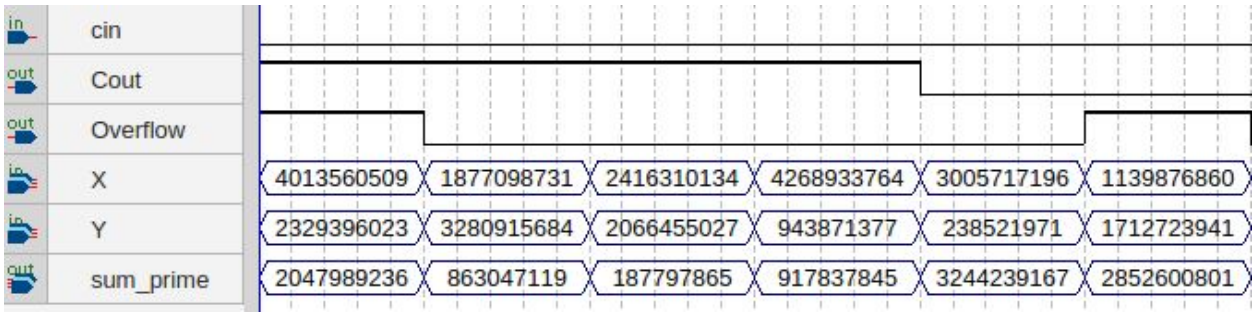

Figure 23: Vector waveform simulation for 32 bit adder. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 32 bit adder.

11. 32 BIT ADDER TEST

11.1 Functionality and specifications for 32 bit adder test.

The purpose of this circuit is to create a test bench in order to simulate our 32 bit adder. We already simulated our 32 bit adder in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_32_BIT_ADDER is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_32_BIT_ADDER;--Jeter Gutierrez September 24 ,2017
architecture arch_test_32 of GUTIERREZ_TEST_32_BIT_ADDER is--Jeter Gutierrez September 24 ,2017
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADDER_32_BITS--Jeter Gutierrez September 24 ,2017
PORT (Cin: IN STD_LOGIC;--Jeter Gutierrez September 24 ,2017
X,Y: IN STD_LOGIC_VECTOR(31 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
sum_prime: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
Cout,Overflow: OUT STD_LOGIC );--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,S :STD_LOGIC_VECTOR(31 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT--Jeter Gutierrez September 24 ,2017)
uut: GUTIERREZ_ADDER_32_BITS port map (--Jeter Gutierrez September 24 ,2017
Cin => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
sum_prime => S,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Proces--Jeter Gutierrez September 24 ,2017s
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 32 BIT ADDER";--Jeter Gutierrez September 24 ,2017
A<="00000000000000000000000000000000";--Jeter Gutierrez September 24 ,2017
B<="00000000000000000000000000000000";--Jeter Gutierrez September 24 ,2017
Ci<='0';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,2017
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez Sep
assert (S = A+B) report "The sum from 32 bit adder is S= " & integer'image(to_integer(unsigned((S)))) &--
" while the expected A+B = " & integer'image(to_integer(unsigned((A+B)))) severity ERROR;--Jeter Gutierre
assert(overflow2=overflow2) report "Overflow is wrong" severity ERROR;--Jeter Gutierrez September 24 ,201
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"00000000000000000000000000000001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"00000000000000000000000000000001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
---END User Defined Process--Jeter Gutierrez September 24 ,2017
end arch_test_32;--Jeter Gutierrez September 24 ,2017
```

Figure 24: VHDL code for the test bench for testing 32 bit adder. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2

32 bit words by increasing each one by 1 bit and determine whether for every possible case of adding 2 32 bit words after a certain period of time or after testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 32 bit adder.

11.2 Simulation for 32 bit adder Test.

In this simulation we will be using modelsim to run our test bench file for the 32 bit adder, if we get no errors we know we have designed a 32 bit adder correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

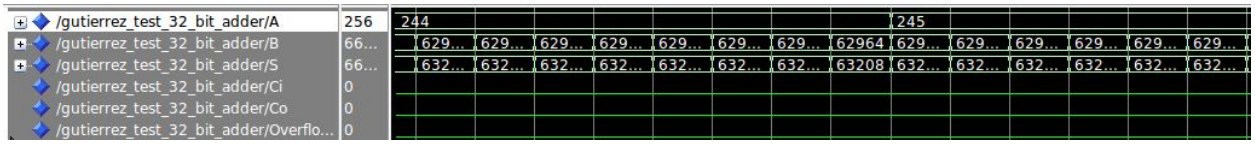

Figure 25: Vector waveform simulation for for test bench of 32 bit adder test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 32 bit adder test. The test bench here was used to test every possible combination of 32-Bit words.

12. 32 BIT ADDER SUBTRACTOR

12.1 Functionality and specifications for 32 bit adder subtractor.

The purpose of the 32 bit adder subtractor is to add or subtract 2 32 bit words and keep track of the overflow value, the carry output and total sum or difference. The input values will be stored in X and Y which are both 32 bit vectors in vhdl, the sum of the two bits will be stored in S, the overflow will be stored in Overflow and the final carry value will be stored in Cout. If a value for

the sum or difference of the 2 32 bit words is out of range from the 32 bit word capacity the value of overflow will be 1 or true, this means that the value in S will be an incorrect value and not useful information. This circuit will be capable of adding 2 32 bit words in signed, unsigned formats correctly keeping track of overflow. It will also be capable of subtracting 2 32 bit words when the signal for SUBTRACT is 1, when it is 0 the circuit will add the 2 32 bit words. The idea behind keeping track of overflow is considering that adding 2 positive numbers should ideally result in another positive number, and that subtracting two negative numbers should result in a negative number as well, that is how we keep track of overflow in the design of this 32 bit adder subtractor just like we did with the 4 bit adder subtractor and the 16 bit adder subtractor.

```
library IEEE;--Jeter Gutierrez September 24 ,2017
use IEEE.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use IEEE.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use IEEE.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_ADD_SUB_32_BITS is--Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
             S: out std_logic_vector (31 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017
architecture STRUCTURE_32 of GUTIERREZ_ADD_SUB_32_BITS is --Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADDER_32_BITS is --Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (31 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
    signal COMPLEMENT: std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
    signal overflow2: std_logic;--Jeter Gutierrez September 24 ,2017
    signal SUM_TEMP: std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
    FIRST:--Jeter Gutierrez September 24 ,2017
        for i in 0 to (31) generate--Jeter Gutierrez September 24 ,2017
            INVERSE: COMPLEMENT(i) <= Y(i) xor SUBTRACT;--Jeter Gutierrez September 24 ,2017
        end generate;--Jeter Gutierrez September 24 ,2017
    SECOND : GUTIERREZ_ADDER_32_BITS port map(SUBTRACT, X, COMPLEMENT, SUM_TEMP, Cout, Overflow2);--Jet
    Overflow<= '1' WHEN (SUBTRACT='0' AND X(31)=COMPLEMENT(31) AND SUM_TEMP(31)/=X(31)) or--Jeter Gutie
                        (SUBTRACT='1' AND X(31)/=COMPLEMENT(31) AND SUM_TEMP(31)/=X(31)) ELSE '0';-
    S<=SUM_TEMP;--Jeter Gutierrez September 24 ,2017
end STRUCTURE_32;--Jeter Gutierrez September 24 ,2017
```

Figure 26: VHDL code for 32 bit adder subtractor.

12.2 Simulation for 32 bit adder Subtractor

In this simulation X and Y will be given different values, in order to add them together or subtract them from each other based on the value of SUBTRACT, their result will be sent to S, we will also keep track of overflow and carry out values, then we will observe the simulation and consider its correctness.

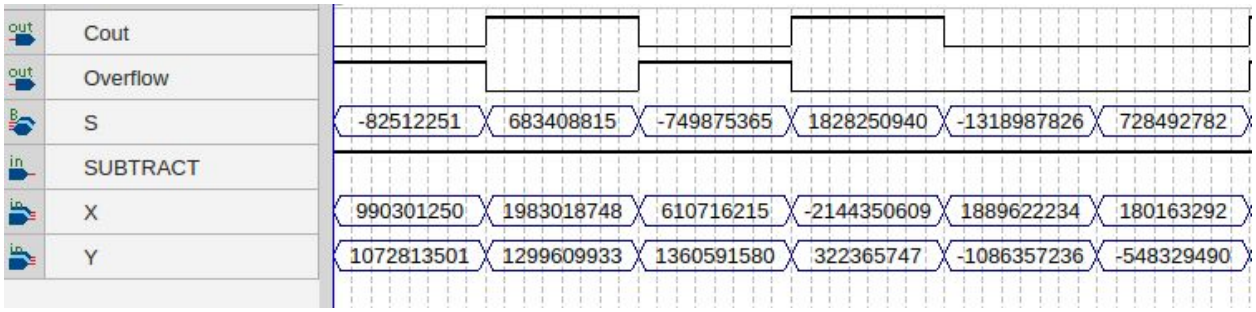

Figure 27: Vector waveform simulation for 32 bit adder subtractor. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 32 bit adder subtractor.

13. 32 BIT ADDER SUBTRACTOR TEST

13.1 Functionality and specifications for 32 bit adder subtractor test.

The purpose of this circuit is to create a test bench in order to simulate our 32 bit adder subtractor. We already simulated our 32 bit adder subtractor in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_32_BIT_ADD_SUB is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_32_BIT_ADD_SUB;--Jeter Gutierrez September 24 ,2017
architecture arch_test of GUTIERREZ_TEST_32_BIT_ADD_SUB is--Jeter Gutierrez September 24 ,2017
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADD_SUB_32_BITS--Jeter Gutierrez September 24 ,2017
PORT (SUBTRACT: IN STD_LOGIC;--Jeter Gutierrez September 24 ,2017
X,Y: IN STD_LOGIC_VECTOR(31 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
S: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
Cout,Overflow: OUT STD_LOGIC );--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,Sum_temp :STD_LOGIC_VECTOR(31 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT--Jeter Gutierrez September 24 ,2017)
uut: GUTIERREZ_ADD_SUB_32_BITS port map (--Jeter Gutierrez September 24 ,2017
SUBTRACT => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
S => Sum_temp,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Process--Jeter Gutierrez September 24 ,2017
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 32 BIT SUBTRACTOR";--Jeter Gutierrez September 24 ,2017
A<="00000000000000000000000000000000";--Jeter Gutierrez September 24 ,2017
B<="00000000000000000000000000000000";--Jeter Gutierrez September 24 ,2017
Ci<='1';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,2017
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez September 24 ,2017
assert (Sum_temp = A-B) report "The DIFFERENCE FROM THE 32 BIT SUBTRACTOR IS S= " & integer'image(to_integer(signed((Sum_temp)))) &-
" while the expected A-B = " & integer'image(to_integer(signed((A-B)))) severity ERROR;--Jeter Gutierrez September 24 ,2017
assert (overfow2=overflow2) report "Overflow is wrong" severity ERROR;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"00000000000000000000000000000001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"00000000000000000000000000000001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
---END User Defined Process--Jeter Gutierrez September 24 ,2017
end arch_test;--Jeter Gutierrez September 24 ,2017
```

Figure 28: VHDL code for the test bench for testing 32 bit adder subtractor. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2 32 bit words by increasing each one by 1 bit and determine whether for every possible case of adding or subtracting 2 32 bit words after a certain period of time or after testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 32 bit adder subtractor.

13.2 Simulation for 32 bit adder subtractor Test.

In this simulation we will be using modelsim to run our test bench file for the 32 bit adder subtractor, if we get no errors we know we have designed a 32 bit adder subtractor correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

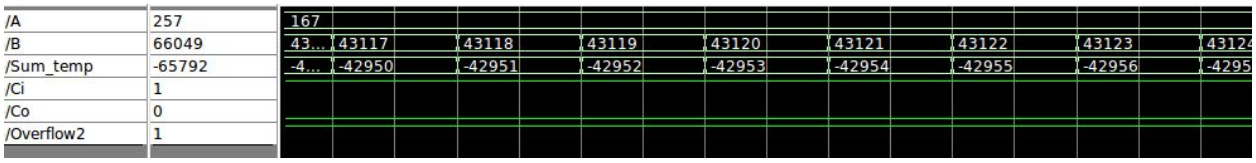

Figure 29: Vector waveform simulation for for test bench of 32 bit adder subtractor test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 32 bit adder subtractor test. The test bench here was used to test every possible combination of 32-Bit words.

14. 4 BIT COMPARATOR ADVANCED

14.1 Functionality and specifications for 4 bit comparator advanced.

The purpose of this 4 bit comparator advanced is to compare 2 4 bit words. The comparison will be done by subtracting the 2 words from each other instead of comparing the words one bit of information at a time, this way we can much more rapidly and more easily keep track of whether these 2 4 bit words are equal to each other. We will also be keeping track of their overflow values and whether the result of the difference is negative which would imply that the second 4 bit word is greater in magnitude than the first 4 bit word. The inputs will be stored in X and Y which are both 4 bit words, there will be a value of SUBTRACT that will always be 1 in order to

correctly compare the 2 4 bit words and the value of whether they are equal will be stored in Z, also keeping track of a carry out and the Overflow.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_COMPARE_4_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
             S: out std_logic_vector (3 downto 0);--Jeter Gutierrez September 24 ,2017
             NEGATIVE: out std_logic;--Jeter Gutierrez September 24 ,2017
             Z: out std_logic;--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017
architecture IDEA of GUTIERREZ_COMPARE_4_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADDER_4_BITS is --Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (3 downto 0);--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
    signal COMPLIMENT: std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
    signal SUM_TEMP: std_logic_vector (3 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
    FIRST:--Jeter Gutierrez September 24 ,2017
        for i in 0 to (3) generate--Jeter Gutierrez September 24 ,2017
            INVERSE: COMPLIMENT(i) <= Y(i) xor SUBTRACT;--Jeter Gutierrez September 24 ,2017
        end generate;--Jeter Gutierrez September 24 ,2017
    SECOND : GUTIERREZ_ADDER_4_BITS port map(SUBTRACT, X, COMPLIMENT, SUM_TEMP, Cout, Overflow);
    THIRD:--Jeter Gutierrez September 24 ,2017
    NEGATIVE<=SUM_TEMP(3);--Jeter Gutierrez September 24 ,2017
        S<=SUM_TEMP;--Jeter Gutierrez September 24 ,2017
        Z<=(((SUM_TEMP(0) nor SUM_TEMP(1)) and (SUM_TEMP(2) nor SUM_TEMP(3))));     --Jeter Guti
end IDEA;--Jeter Gutierrez September 24 ,2017
```

Figure 30: VHDL code for 4 bit comparator advanced.

14.2 Simulation for 4 bit comparator advanced.

In this simulation X and Y will be given different values, the sum or difference of the two words will be sent to S based on the value of SUBTRACT, we will be storing the most significant bit in NEGATIVE which when it is equal to 1 will represent that Y is greater than X in magnitude, we will also keep track of overflow and final carry out value.

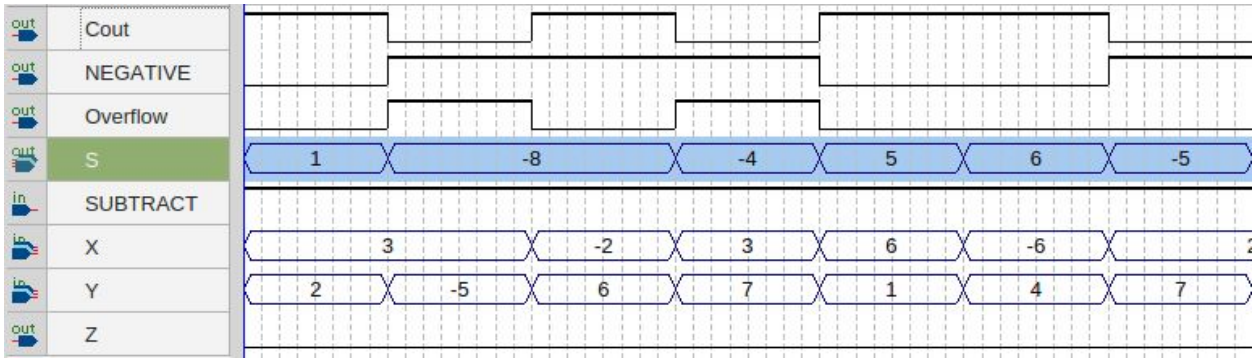

Figure 31: Vector waveform simulation for 4 bit comparator advanced. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 4 bit comparator advanced.

15. 4 BIT COMPARATOR ADVANCED TEST

15.1 Functionality and specifications for 4 bit comparator advanced test.

The purpose of this circuit is to create a test bench in order to simulate our 4 comparator advanced. We already simulated our 4 bit comparator advanced in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_COMPARE_4_BITS_ADVANCE is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_COMPARE_4_BITS_ADVANCE;--Jeter Gutierrez September 24 ,2017
architecture arch_test_4_COMPARE of GUTIERREZ_TEST_COMPARE_4_BITS_ADVANCE is--Jeter Gutierrez September 24 ,2017
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_COMPARE_4_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
           X, Y: in std_logic_vector(3 downto 0);--Jeter Gutierrez September 24 ,2017
           S: out std_logic_vector (3 downto 0);--Jeter Gutierrez September 24 ,2017
           NEGATIVE: out std_logic;--Jeter Gutierrez September 24 ,2017
           Z: out std_logic;--Jeter Gutierrez September 24 ,2017
           Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,SUM :STD_LOGIC_VECTOR(3 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,NEGATIVE2,Z1,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT)--Jeter Gutierrez September 24 ,2017
uut: GUTIERREZ_COMPARE_4_BITS_ADVANCE port map (--Jeter Gutierrez September 24 ,2017
SUBTRACT => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
S => SUM,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
NEGATIVE => NEGATIVE2,--Jeter Gutierrez September 24 ,2017
Z => Z1,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Process--Jeter Gutierrez September 24 ,2017
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 4 BIT COMPARATOR ADVANCED";--Jeter Gutierrez September 24 ,2017
A<="0000";--Jeter Gutierrez September 24 ,2017
B<="0000";--Jeter Gutierrez September 24 ,2017
Ci<='1';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 16 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 16 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,2017
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez September
assert (Sum = A-B) report "The Comparison between A and Be IS S= " & integer'image(to_integer(signed((SUM)))) &-
" while the expected A compared to B = " & integer'image(to_integer(signed((A-B)))) severity ERROR;--Jeter Gutie
if A=B then--Jeter Gutierrez September 24 ,2017
assert (Z1='1') report "Same Values are being reported as different" severity ERROR;--Jeter Gutierrez September
else assert(Z1='0')report "Different Values are being  reported as teh same" severity ERROR;--Jeter Gutierrez Se
end if;--Jeter Gutierrez September 24 ,2017
assert(NEGATIVE2=Sum(3))report "Sign is incorrect." severity ERROR;--Jeter Gutierrez September 24 ,2017
assert(overflow2=overflow2)report "Overflow is wrong" severity ERROR;--Jeter Gutierrez September 24 ,2017
--Jeter Gutierrez September 24 ,2017
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"0001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"0001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
```

Figure 32: VHDL code for the test bench for testing 4 bit comparator advanced. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2 4 bit words by increasing each one by 1 bit and determine whether for every

possible case of comparing 2 4 bit words after a certain period of time or after testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 4 bit comparator advanced.

15.2 Simulation for 4 bit comparator advanced Test.

In this simulation we will be using modelsim to run our test bench file for the 4 bit comparator advanced, if we get no errors we know we have designed a 4 bit comparator advanced correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

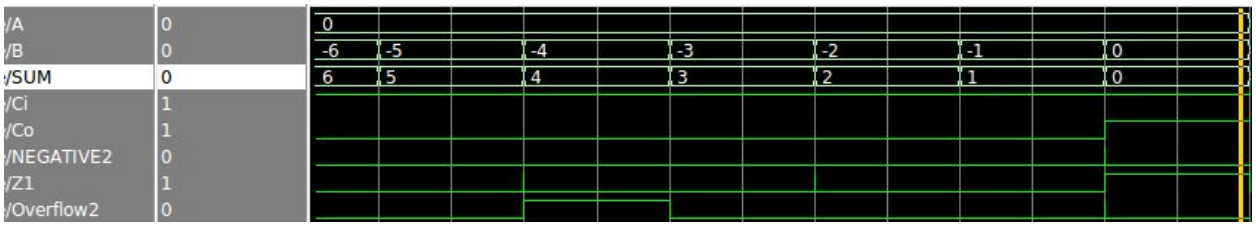

Figure 33: Vector waveform simulation for for test bench of 4 bit comparator advanced test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 4 bit comparator advanced test. The test bench here was used to test every possible combination of 4-Bit words.

16. 16 BIT COMPARATOR ADVANCED

16.1 Functionality and specifications for 16 bit comparator advanced.

The purpose of this 16 bit comparator advanced is to compare 2 16 bit words. The comparison will be done by subtracting the 2 words from each other instead of comparing the words one bit of information at a time, this way we can much more rapidly and more easily keep track of

whether these 2 16 bit words are equal to each other. We will also be keeping track of their overflow values and whether the result of the difference is negative which would imply that the second 16 bit word is greater in magnitude than the first 16 bit word. The inputs will be stored in X and Y which are both 16 bit words, there will be a value of SUBTRACT that will always be 1 in order to correctly compare the 2 16 bit words and the value of whether they are equal will be stored in Z, also keeping track of a carry out and the Overflow.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_COMPARE_16_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
            X, Y: in std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
            S: out std_logic_vector (15 downto 0);--Jeter Gutierrez September 24 ,2017
            NEGATIVE: out std_logic;--Jeter Gutierrez September 24 ,2017
            Z: out std_logic;--Jeter Gutierrez September 24 ,2017
            Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017

architecture IDEA_16 of GUTIERREZ_COMPARE_16_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADDER_16_BITS is --Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
            X, Y: in std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
            sum_prime: out std_logic_vector (15 downto 0);--Jeter Gutierrez September 24 ,2017
            Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
    signal COMPLIMENT: std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
    signal SUM_TEMP: std_logic_vector (15 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
    FIRST:--Jeter Gutierrez September 24 ,2017
        for i in 0 to (15) generate--Jeter Gutierrez September 24 ,2017
            INVERSE: COMPLIMENT(i) <= Y(i) xor SUBTRACT;--Jeter Gutierrez September 24 ,2017
        end generate;--Jeter Gutierrez September 24 ,2017
    SECOND : GUTIERREZ_ADDER_16_BITS port map(SUBTRACT, X, COMPLIMENT, SUM_TEMP, Cout, Overflow);-
    THIRD:--Jeter Gutierrez September 24 ,2017
    NEGATIVE<=SUM_TEMP(15);--Jeter Gutierrez September 24 ,2017
        S<=SUM_TEMP;--Jeter Gutierrez September 24 ,2017
        Z<=((SUM_TEMP(0) nor SUM_TEMP(1)) and (SUM_TEMP(2) nor SUM_TEMP(3))and(SUM_TEMP(4) nor SUM
(15)));
end IDEA_16;--Jeter Gutierrez September 24 ,2017
```

Figure 34: VHDL code for 16 bit comparator advanced.

16.2 Simulation for 16 bit comparator advanced.

In this simulation X and Y will be given different values, the sum or difference of the two words will be sent to S based on the value of SUBTRACT, we will be storing the most significant bit in

NEGATIVE which when it is equal to 1 will represent that Y is greater than X in magnitude, we will also keep track of overflow and final carry out value.

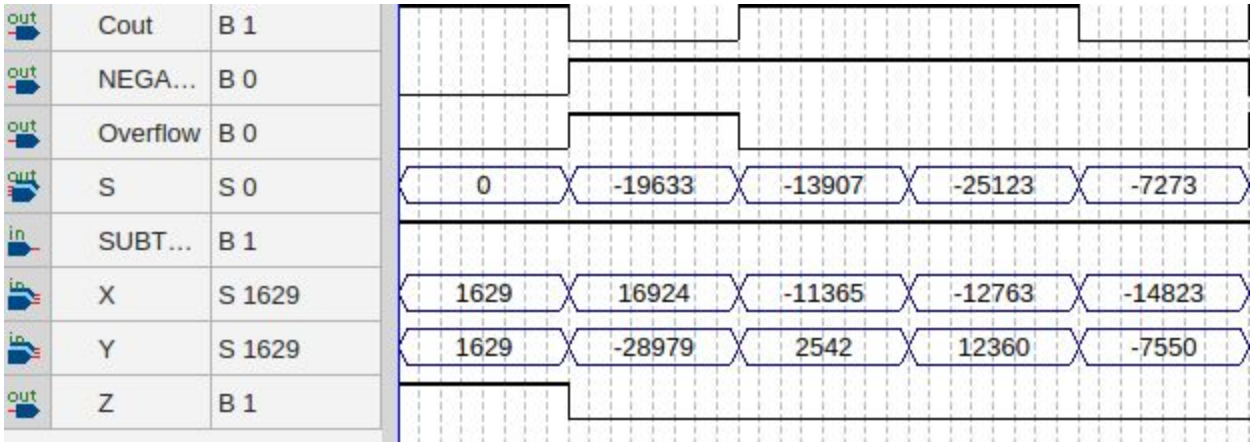

Figure 35: Vector waveform simulation for 16 bit comparator advanced. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 16 bit comparator advanced.

17. 16 BIT COMPARATOR ADVANCED TEST

17.1 Functionality and specifications for 16 bit comparator advanced test.

The purpose of this circuit is to create a test bench in order to simulate our 16 comparator advanced. We already simulated our 16 bit comparator advanced in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_COMPARE_16_BITS_ADVANCE is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_COMPARE_16_BITS_ADVANCE;--Jeter Gutierrez September 24 ,2017
architecture arch_test_16_COMPARE of GUTIERREZ_TEST_COMPARE_16_BITS_ADVANCE is--Jeter Gutierrez September 24 ,2(
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_COMPARE_16_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
           X, Y: in std_logic_vector(15 downto 0);--Jeter Gutierrez September 24 ,2017
           S: out std_logic_vector (15 downto 0);--Jeter Gutierrez September 24 ,2017
           NEGATIVE: out std_logic;--Jeter Gutierrez September 24 ,2017
           Z: out std_logic;--Jeter Gutierrez September 24 ,2017
           Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,SUM :STD_LOGIC_VECTOR(15 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,NEGATIVE2,Z1,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT)--Jeter Gutierrez September 24 ,2017
uut: GUTIERREZ_COMPARE_16_BITS_ADVANCE port map (--Jeter Gutierrez September 24 ,2017
SUBTRACT => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
S => SUM,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
NEGATIVE => NEGATIVE2,--Jeter Gutierrez September 24 ,2017
Z => Z1,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Process--Jeter Gutierrez September 24 ,2017
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING COMPARATOR 16 BITS ADVANCED";--Jeter Gutierrez September 24 ,2017
A<="0000000000000000";--Jeter Gutierrez September 24 ,2017
B<="0000000000000000";--Jeter Gutierrez September 24 ,2017
Ci<='1';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,2017
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez September
assert (Sum = A-B) report "The Comparison between A and B are S= " & integer'image(to_integer(signed((SUM)))) &
" while the expected A compared to B = " & integer'image(to_integer(signed((A-B)))) severity ERROR;--Jeter Gutie
if A=B then--Jeter Gutierrez September 24 ,2017
assert (Z1='1') report "Same Values are being reported as different" severity ERROR;--Jeter Gutierrez September
else assert(Z1='0')report "Different Values are being  reported as teh same" severity ERROR;--Jeter Gutierrez Se
end if;--Jeter Gutierrez September 24 ,2017
assert(NEGATIVE2=Sum(15))report "Sign is incorrect." severity ERROR;--Jeter Gutierrez September 24 ,2017
assert(Overflow2=Overflow2) report "Overflow is wrong" severity ERROR;--Jeter Gutierrez September 24 ,2017
--Jeter Gutierrez September 24 ,2017
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"0000000000000001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"0000000000000001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
```

Figure 36: VHDL code for the test bench for testing 16 bit comparator advanced. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2 16 bit words by increasing each one by 1 bit and determine whether for

every possible case of comparing 2 16 bit words after a certain period of time or after testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 16 bit comparator advanced.

17.2 Simulation for 16 bit comparator advanced Test.

In this simulation we will be using modelsim to run our test bench file for the 16 bit comparator advanced, if we get no errors we know we have designed a 16 bit comparator advanced correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

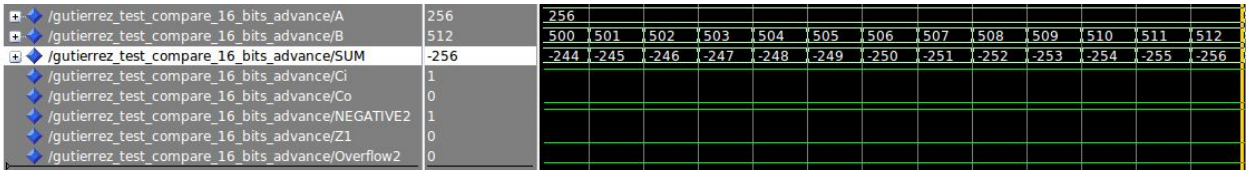

Figure 37: Vector waveform simulation for for test bench of 16 bit comparator advanced test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 16 bit comparator advanced test. The test bench here was used to test every possible combination of 16-Bit words.

18. 32 BIT COMPARATOR ADVANCED

18.1 Functionality and specifications for 32 bit comparator advanced.

The purpose of this 32 bit comparator advanced is to compare 2 32 bit words. The comparison will be done by subtracting the 2 words from each other instead of comparing the words one bit of information at a time, this way we can much more rapidly and more easily keep track of

whether these 2 32 bit words are equal to each other. We will also be keeping track of their overflow values and whether the result of the difference is negative which would imply that the second 32 bit word is greater in magnitude than the first 32 bit word. The inputs will be stored in X and Y which are both 32 bit words, there will be a value of SUBTRACT that will always be 1 in order to correctly compare the 2 32 bit words and the value of whether they are equal will be stored in Z, also keeping track of a carry out and the Overflow.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_COMPARE_32_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
             S: out std_logic_vector (31 downto 0);--Jeter Gutierrez September 24 ,2017
             NEGATIVE: out std_logic;--Jeter Gutierrez September 24 ,2017
             Z: out std_logic;--Jeter Gutierrez September 24 ,2017
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end entity;--Jeter Gutierrez September 24 ,2017
--Jeter Gutierrez September 24 ,2017
architecture IDEA_32 of GUTIERREZ_COMPARE_32_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
component GUTIERREZ_ADDER_32_BITS is --Jeter Gutierrez September 24 ,2017
    port ( cin: in std_logic;--Jeter Gutierrez September 24 ,2017
             X, Y: in std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
             sum_prime: out std_logic_vector (31 downto 0);
             Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
    signal COMPLIMENT: std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
    signal SUM_TEMP: std_logic_vector (31 downto 0);--Jeter Gutierrez September 24 ,2017
    begin--Jeter Gutierrez September 24 ,2017
    FIRST:--Jeter Gutierrez September 24 ,2017
        for i in 0 to (31) generate--Jeter Gutierrez September 24 ,2017
            INVERSE: COMPLIMENT(i) <= Y(i) xor SUBTRACT;--Jeter Gutierrez September 24 ,2017
        end generate;--Jeter Gutierrez September 24 ,2017
    SECOND : GUTIERREZ_ADDER_32_BITS port map(SUBTRACT, X, COMPLIMENT, SUM_TEMP, Cout, Overflow);-
    THIRD:--Jeter Gutierrez September 24 ,2017
    NEGATIVE<=SUM_TEMP(31);--Jeter Gutierrez September 24 ,2017
        S<=SUM_TEMP;--Jeter Gutierrez September 24 ,2017
        Z<=((SUM_TEMP(0) nor SUM_TEMP(1)) and (SUM_TEMP(2) nor SUM_TEMP(3))and(SUM_TEMP(4) nor SUM
(13))and(SUM_TEMP(14) nor SUM_TEMP(15))and(SUM_TEMP(16) nor SUM_TEMP(17)) and (SUM_TEMP(18) nor SU
SUM_TEMP(27))and(SUM_TEMP(28) nor SUM_TEMP(29))and(SUM_TEMP(30) nor SUM_TEMP(31)));
end IDEA_32;--Jeter Gutierrez September 24 ,2017
```

Figure 38: VHDL code for 32 bit comparator advanced.

18.2 Simulation for 32 bit comparator advanced.

In this simulation X and Y will be given different values, the sum or difference of the two words will be sent to S based on the value of SUBTRACT, we will be storing the most significant bit in

NEGATIVE which when it is equal to 1 will represent that Y is greater than X in magnitude, we will also keep track of overflow and final carry out value.

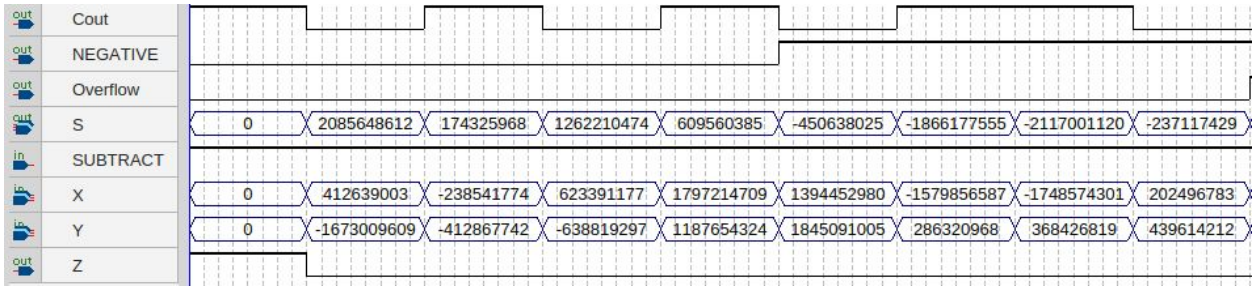

Figure 39: Vector waveform simulation for 32 bit comparator advanced. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 32 bit comparator advanced.

19. 32 BIT COMPARATOR ADVANCED TEST

19.1 Functionality and specifications for 32 bit comparator advanced test.

The purpose of this circuit is to create a test bench in order to simulate our 32 comparator advanced. We already simulated our 32 bit comparator advanced in Vector waveform but it is more professional and exact to use a test bench to test the correctness of our circuits because we have more control over the simulation this way. The test bench imitates a physical lab bench making our simulation cleaner and more organized.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.numeric_std.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_unsigned.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
--USE work.fulladd_package.all ;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_TEST_COMPARE_32_BITS_ADVANCE is--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_TEST_COMPARE_32_BITS_ADVANCE;--Jeter Gutierrez September 24 ,2017
architecture arch_test_32_COMPARE of GUTIERREZ_TEST_COMPARE_32_BITS_ADVANCE is--Jeter Gutierrez September 24 ,2
--componengt declaration for the Unit Under Test--Jeter Gutierrez September 24 ,2017
component GUTIERREZ_COMPARE_32_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
           X, Y: in std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
           S: out std_logic_vector (31 downto 0);--Jeter Gutierrez September 24 ,2017
           NEGATIVE: out std_logic;--Jeter Gutierrez September 24 ,2017
           Z: out std_logic;--Jeter Gutierrez September 24 ,2017
           Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal A,B,SUM :STD_LOGIC_VECTOR(31 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal Ci,Co,NEGATIVE2,Z1,Overflow2 :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
-----Instantiate the Unit Under Test (UUT)--Jeter Gutierrez September 24 ,2017
uut: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (--Jeter Gutierrez September 24 ,2017
SUBTRACT => Ci,--Jeter Gutierrez September 24 ,2017
X => A,--Jeter Gutierrez September 24 ,2017
Y => B,--Jeter Gutierrez September 24 ,2017
S => SUM,--Jeter Gutierrez September 24 ,2017
Cout =>Co,--Jeter Gutierrez September 24 ,2017
NEGATIVE => NEGATIVE2,--Jeter Gutierrez September 24 ,2017
Z => Z1,--Jeter Gutierrez September 24 ,2017
Overflow => Overflow2--Jeter Gutierrez September 24 ,2017
);--Jeter Gutierrez September 24 ,2017
---- Test Bench ---User Defined Process--Jeter Gutierrez September 24 ,2017
tb : process--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
--Hold reset state for 100 ns--Jeter Gutierrez September 24 ,2017
wait for 100 ns;--Jeter Gutierrez September 24 ,2017
report "TESTING 32 BIT COMPARATOR ADVANCED";--Jeter Gutierrez September 24 ,2017
A<="00000000000000000000000000000000";--Jeter Gutierrez September 24 ,2017
B<="00000000000000000000000000000000";--Jeter Gutierrez September 24 ,2017
Ci<='1';--Jeter Gutierrez September 24 ,2017
--Loop over all values of A--Jeter Gutierrez September 24 ,2017
for I in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Loop over all values of B--Jeter Gutierrez September 24 ,2017
for J in 0 to 256 loop--Jeter Gutierrez September 24 ,2017
--Wait for outputto update--Jeter Gutierrez September 24 ,2017
wait for 10 ns;--Jeter Gutierrez September 24 ,2017
---report " the A+B = " & integer'image(to_integer(unsigned((A+B))));--Jeter Gutierrez September 24 ,2017
---The statement below checks for ALL possible input values if the ouput is correct.--Jeter Gutierrez September
assert (Sum = A-B) report "The Comparison between A and B are S= " & integer'image(to_integer(signed((SUM)))) &
" while the expected A compared to B = " & integer'image(to_integer(signed((A-B)))) severity ERROR;--Jeter Guti
if A=B then--Jeter Gutierrez September 24 ,2017
assert (Z1='1') report "Same Values are being reported as different" severity ERROR;--Jeter Gutierrez September
else assert(Z1='0')report "Different Values are being  reported as teh same" severity ERROR;--Jeter Gutierrez S
end if;--Jeter Gutierrez September 24 ,2017
assert(NEGATIVE2=Sum(31))report "Sign is incorrect." severity ERROR;--Jeter Gutierrez September 24 ,2017
assert (overflow2=overflow2) report "Overflow is wrong" severity ERROR;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of B--Jeter Gutierrez September 24 ,2017
B<=B+"00000000000000000000000000000001";--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
--Increment to the next value of A--Jeter Gutierrez September 24 ,2017
A<=A+"00000000000000000000000000000001";--Jeter Gutierrez September 24 ,2017
--Echo to users test is finished--Jeter Gutierrez September 24 ,2017
end loop;--Jeter Gutierrez September 24 ,2017
report "Test completed";--Jeter Gutierrez September 24 ,2017
wait; -- will wait for ever--Jeter Gutierrez September 24 ,2017
end process;--Jeter Gutierrez September 24 ,2017
---END User Defined Process--Jeter Gutierrez September 24 ,2017
```

Figure 40: VHDL code for the test bench for testing 32 bit comparator advanced. In this design modelsim will be used in order to simulate what is written in the testing code. It will be used to test the values for 2 32 bit words by increasing each one by 1 bit and determine whether for

every possible case of comparing 2 32 bit words after a certain period of time or after testing another state it will continue to be correct. The reason we do this is to have more control over our testing for correctness of our 32 bit comparator advanced.

19.2 Simulation for 32 bit comparator advanced Test.

In this simulation we will be using modelsim to run our test bench file for the 32 bit comparator advanced, if we get no errors we know we have designed a 32 bit comparator advanced correctly. The difference in this state is that we already wrote what values we want to test for and modelsim will create those values for us instead of us having to use the cursor to select the values using waveform. Using a test bench is more efficient than manually inputting values to test, this way we test every possible value.

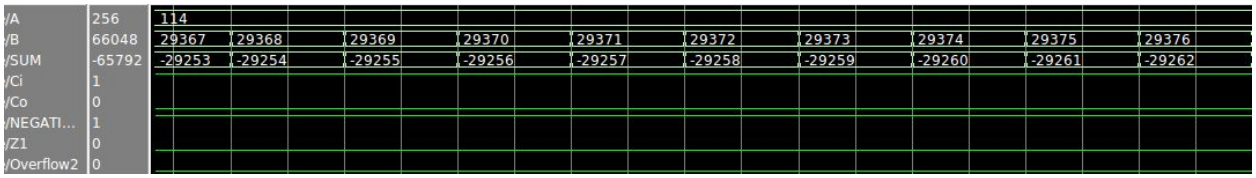

Figure 41: Vector waveform simulation for for test bench of 32 bit comparator advanced test. As we can see we have no errors in our simulation or design, that means that our design was correct and that we did not make any mistakes, we were able to successfully design a 32 bit comparator advanced test. The test bench here was used to test every possible combination of 32-Bit words.

20. ARRAY OF 32 BIT WORDS

20.1 Functionality and specifications of 32 bit words.

The purpose of this circuit is to design an array of 32 bit words, in this case we will be using 10 different words and then use the 32 bit comparator advanced in order to search through the array and find the location for the 32 bit word we are looking for. The reason we are doing this is to

model how memory works and better understand how to locate addresses in memory the way a computer does.

```
library ieee;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_1164.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_arith.all;--Jeter Gutierrez September 24 ,2017
use ieee.std_logic_signed.all;--Jeter Gutierrez September 24 ,2017
use work.GUTIERREZ_ADDER_PACKAGE_COMPARE.all;--Jeter Gutierrez September 24 ,2017
entity GUTIERREZ_32_BIT_SEARCH is--Jeter Gutierrez September 24 ,2017
    port(X :in std_logic_vector (31 downto 0);
            Position: out std_logic_vector(9 downto 0));--Jeter Gutierrez September 24 ,2017
end GUTIERREZ_32_BIT_SEARCH;--Jeter Gutierrez September 24 ,2017
architecture DESIGN_SEARCH of GUTIERREZ_32_BIT_SEARCH is --Jeter Gutierrez September 24 ,2017
type MEMORY is array (0 to 9) of std_logic_vector(31 downto 0);
component GUTIERREZ_COMPARE_32_BITS_ADVANCE is --Jeter Gutierrez September 24 ,2017
    port ( SUBTRACT: in std_logic;--Jeter Gutierrez September 24 ,2017
            X, Y: in std_logic_vector(31 downto 0);--Jeter Gutierrez September 24 ,2017
            S: out std_logic_vector (31 downto 0);--Jeter Gutierrez September 24 ,2017
            NEGATIVE: out std_logic;--Jeter Gutierrez September 24 ,2017
            Z: out std_logic;--Jeter Gutierrez September 24 ,2017
            Cout, Overflow: out std_logic);--Jeter Gutierrez September 24 ,2017
end component;--Jeter Gutierrez September 24 ,2017
signal STORED_WORDS : MEMORY :=--Jeter Gutierrez September 24 ,2017
("00000000000000000000000000000000",--<= STORED_WORDS(0);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000000001",--<= STORED_WORDS(1);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000000010",--<= STORED_WORDS(2);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000000011",--<= STORED_WORDS(3);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000000100",--<= STORED_WORDS(4);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000000101",--<= STORED_WORDS(5);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000000110",--<= STORED_WORDS(6);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000000111",--<= STORED_WORDS(7);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000001000",--<= STORED_WORDS(8);--Jeter Gutierrez September 24 ,2017
"00000000000000000000000000001001");--<= STORED_WORDS(9);--Jeter Gutierrez September 24 ,2017
signal S_TEMP :STD_LOGIC_VECTOR(31 DOWNTO 0);--Jeter Gutierrez September 24 ,2017
signal SUBTRACT_TEMP,NEGATIVE_TEMP,Z_TEMP,COUT_TEMP,OVERFLOW_TEMP :STD_LOGIC;--Jeter Gutierrez September 24 ,2017
begin--Jeter Gutierrez September 24 ,2017
SUBTRACT_TEMP<= '1';--Jeter Gutierrez September 24 ,2017
FIRST: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(0),S_TEMP,NEGATIVE_TEMP,POSITION(0),Overflow_TEMP);--
SECOND: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(1),S_TEMP,NEGATIVE_TEMP,POSITION(1),Overflow_TEMP);-
THIRD: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(2),S_TEMP,NEGATIVE_TEMP,POSITION(2),Overflow_TEMP);--
FOURTH: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(3),S_TEMP,NEGATIVE_TEMP,POSITION(3),Overflow_TEMP);-
FIFTH: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(4),S_TEMP,NEGATIVE_TEMP,POSITION(4),Overflow_TEMP);--
SIXTH: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(5),S_TEMP,NEGATIVE_TEMP,POSITION(5),Overflow_TEMP);--
SEVENTH: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(6),S_TEMP,NEGATIVE_TEMP,POSITION(6),Overflow_TEMP);
EIGHT: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(7),S_TEMP,NEGATIVE_TEMP,POSITION(7),Overflow_TEMP);--
NINTH: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(8),S_TEMP,NEGATIVE_TEMP,POSITION(8),Overflow_TEMP);--
TENTH: GUTIERREZ_COMPARE_32_BITS_ADVANCE port map (SUBTRACT_TEMP,X,STORED_WORDS(9),S_TEMP,NEGATIVE_TEMP,POSITION(9),Overflow_TEMP);--
end DESIGN_SEARCH;--Jeter Gutierrez September 24 ,2017
```

Figure 42: VHDL code for array of 32 bit words. I initialized some of the words already in the array in order to search later in simulation to show that we are working with arrays in VHDL correctly.

20.2 Simulation for array of 32 bit words.

In this simulation we will run the code in order to determine whether we were able to locate the 32 bit word within the array. If the position is reported correctly it will have meant that we were able to successfully detect and search for a 32 bit word in an array of 32 bit words.

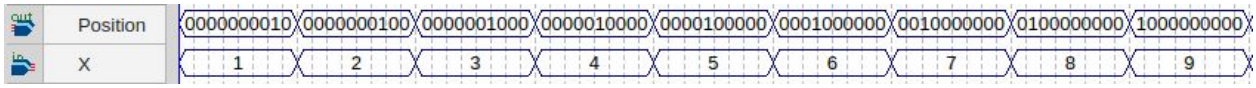

Figure 43: Vector waveform simulation for array of 32 bit words. As we can see the address of the 32 bit word we were looking for was located properly which means that we were able to correctly design a model for searching in memory using arrays and 32 bit words and advanced 32 bit word comparators. We saw in the implementation that 32 bit words stored in the array were stored in the location where the position is outputting a value of 1 which means it is searching through the array correctly.

21.1 Demonstration of 4 Bit adder subtractor on DE2-115 Board.

The inputs and outputs assigned to the DE2-115 board are:

X[0] is assigned to PIN_AB28

X[1] is assigned to PIN_AC28

X[2] is assigned to PIN_AC27

X[3] is assigned to PIN_AD27

Y[0] is assigned to PIN_AB27

Y[1] is assigned to PIN_AC26

Y[2] is assigned to PIN_AD26

Y[3]is assigned to PIN_AB26

S[0] is assigned to PIN_G19

S[1] is assigned to PIN_F19

S[2] is assigned to PIN_E19

S[3] is assigned to PIN_F21

Cout is assigned to PIN_F18

Overflow is assigned to PIN_E18

SUBTRACT is assigned to PIN_Y23

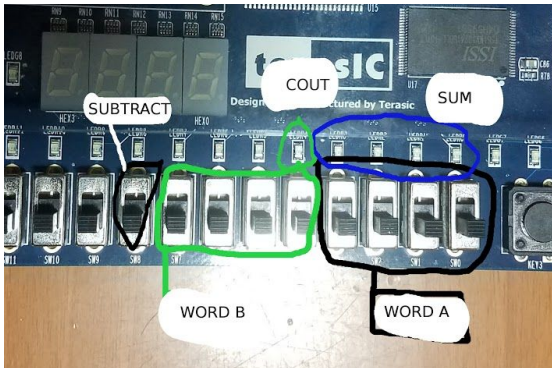

Figure 44: Digital circuit of 4 bit adder subtractor on DE2-115 Board. The reason we are testing the 4 bit adder subtractor instead of 8 or 16 or higher is because we are not using SRAM on this lab report and so we do not have enough pins to test out the circuit on the DE2-115 board but implementation is the same so it will work correctly.

21.2 Demonstration of 8 Bit comparator advanced on DE2-115 Board.

The inputs and outputs assigned to the DE2-115 board are:

SUBTRACT is assigned to PIN_M23

X[0] is assigned to PIN_AB28

X[1] is assigned to PIN_AC28

X[2] is assigned to PIN_AC27

X[3] is assigned to PIN_AD27

X[4] is assigned to PIN_AB27

X[5] is assigned to PIN_AC26

X[6] is assigned to PIN_AD26

X[7] is assigned to PIN_AB26

Y[0] is assigned to PIN_AC25

Y[1] is assigned to PIN_AB25

Y[2] is assigned to PIN_AC24

Y[3] is assigned to PIN_AB24

Y[4] is assigned to PIN_AB23

Y[5] is assigned to PIN_AA24

Y[6] is assigned to PIN_AA23

Y[7] is assigned to PIN_AA22

NEGATIVE is assigned to PIN_G19

Z is assigned to PIN_F19

Cout is assigned to PIN_E19

Overflow is assigned to PIN_F21

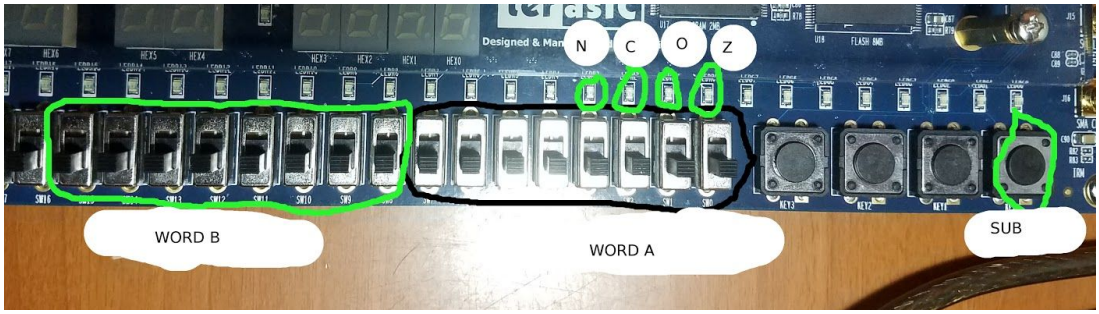

Figure 45: Digital circuit design of 8 bit comparator on DE2-115 Board,the reason we are testing an 8 bit comparator advanced instead of 16 or 32 is because we are not using sram and so we do not have enough switches.

21.3 Demonstration of 8 Bit array on DE2-115 Board.

The inputs and outputs assigned to the DE2-115 board are:

X[0] is assigned to PIN_AB28

X[1] is assigned to PIN_AC28

X[2] is assigned to PIN_AC27

X[3] is assigned to PIN_AD27

X[4] is assigned to PIN_AB27

X[5] is assigned to PIN_AC26

X[6] is assigned to PIN_AD26

X[7] is assigned to PIN_AB26

Position[0] is assigned to PIN_G19

Position[1] is assigned to PIN_F19

Position[2] is assigned to PIN_E19

Position[3] is assigned to PIN_F21

Position[4] is assigned to PIN_F18

Position[5] is assigned to PIN_E18

Position[6] is assigned to PIN_J19

Position[7] is assigned to PIN_H19

Position[8] is assigned to PIN_J17

Position[9] is assigned to PIN_G17

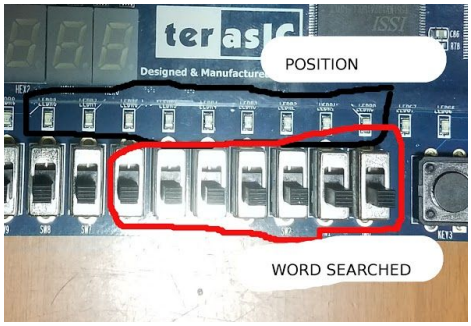

Figure 46: Digital Circuit of 8 bit array search the reason we are using 8 bits is because we do not have enough switches for 32 bits.

22. Conclusion.

As it turns out designing n-bit adders, subtractors and comparators is very useful. It is a much more efficient way to compare n-bit words with each other instead of comparing them one bit at a time. This also helps me understand how memory works better because instead of iterating through elements it takes a difference from what it is looking for to immediately point to the location that it is looking for, not immediately but as quickly as possible and that's fantastic. I learned how to work with arrays in VHDL which I was already expecting to be a useful tool and it is. Detecting overflow is something else that I learned how to do in more detail instead of just for 4 bits, now i understand how to correctly detect overflow for signed integers of any length of bits. I also have learned to value using a test bench, it saves a great deal of time and it also is a more precise way to test for all possible values of a program, this way it can detect errors for values that otherwise might not have been tested in the first place.

22. Appendix

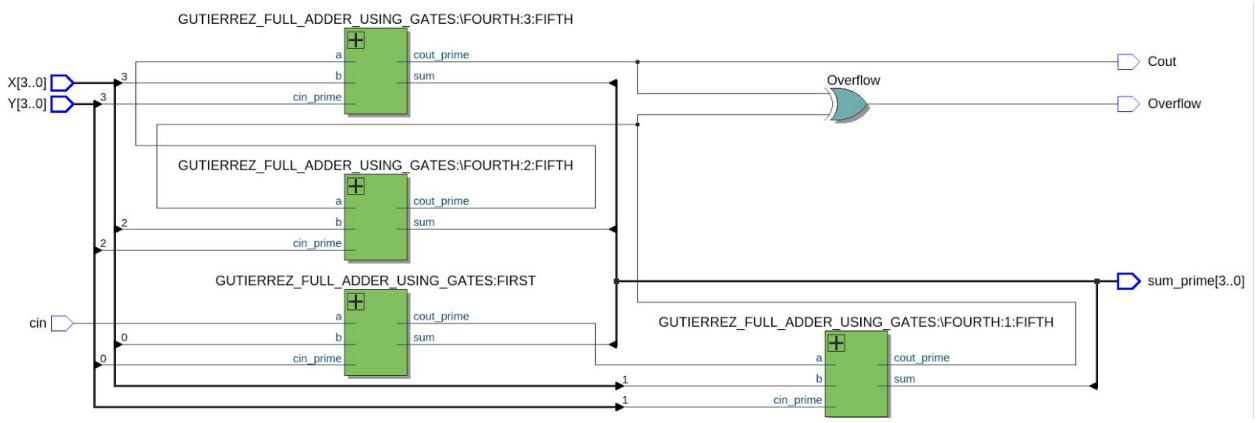

Figure 47: Block diagram for 4 bit adder.

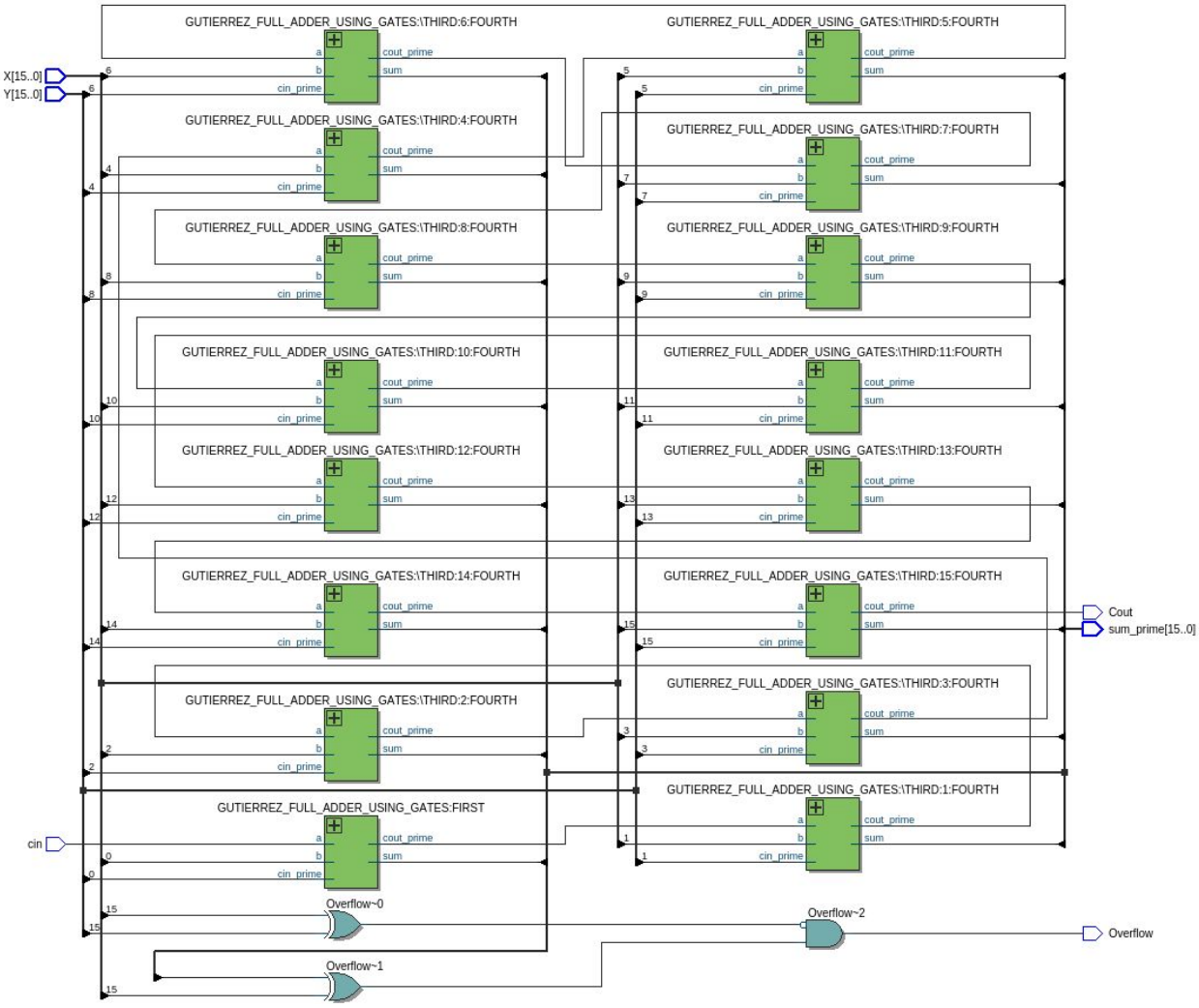

Figure 48: Block diagram for 16 bit adder.

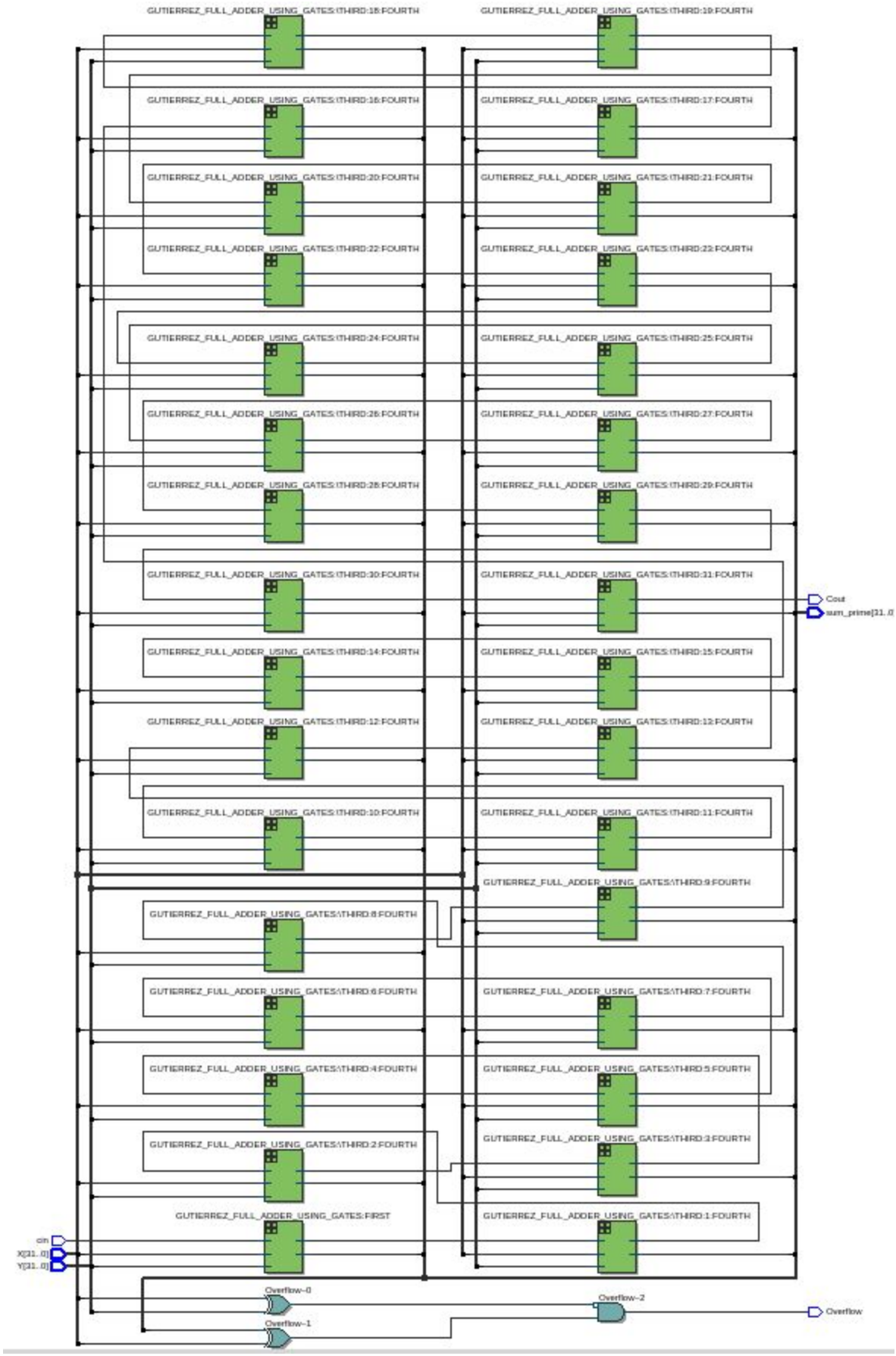
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:18:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:19:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:16:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:17:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:20:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:21:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:22:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:23:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:24:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:25:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:26:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:27:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:28:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:29:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:30:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:31:FOURTH
Cout
sum_prime[31..0]
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:14:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:15:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:12:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:13:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:10:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:11:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:9:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:8:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:6:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:7:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:4:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:5:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:2:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:3:FOURTH
GUTIERREZ_FULL_ADDER_USING_GATES:FIRST
GUTIERREZ_FULL_ADDER_USING_GATES:ITHIRD:1:FOURTH
cin
X[31..0]
Y[31..0]
Overflow~0
Overflow~2
Overflow
Overflow~1

Figure 49: Block diagram for 32 bit adder.

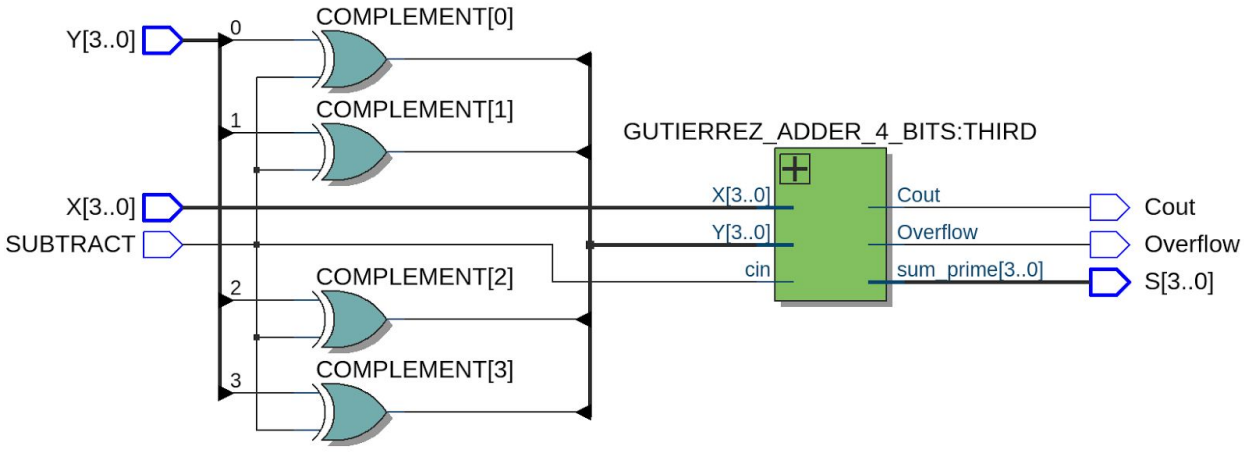

Figure 50: Block diagram for 4 bit adder subtractor.

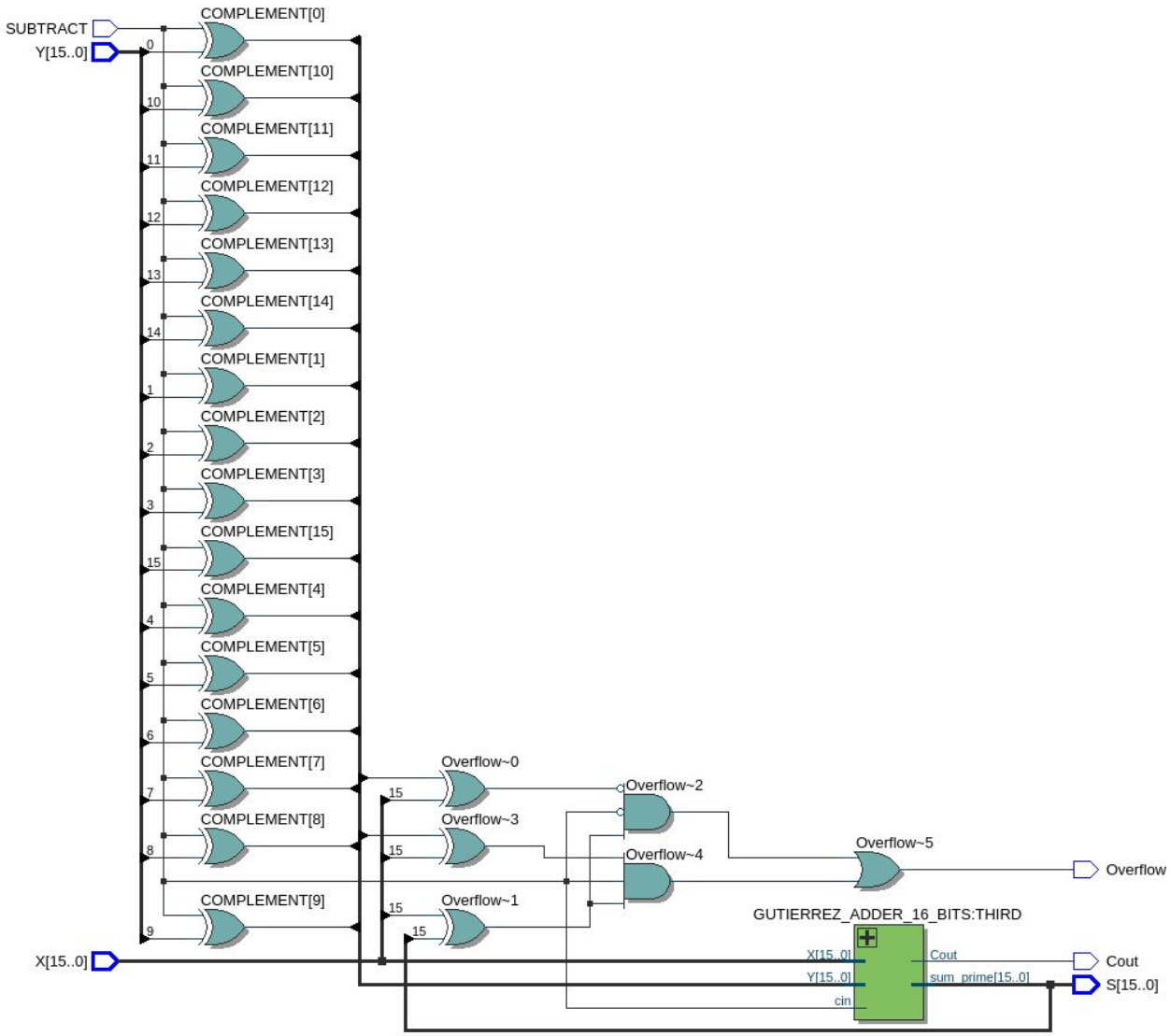

Figure 51: Block diagram for 16 bit adder subtractor.

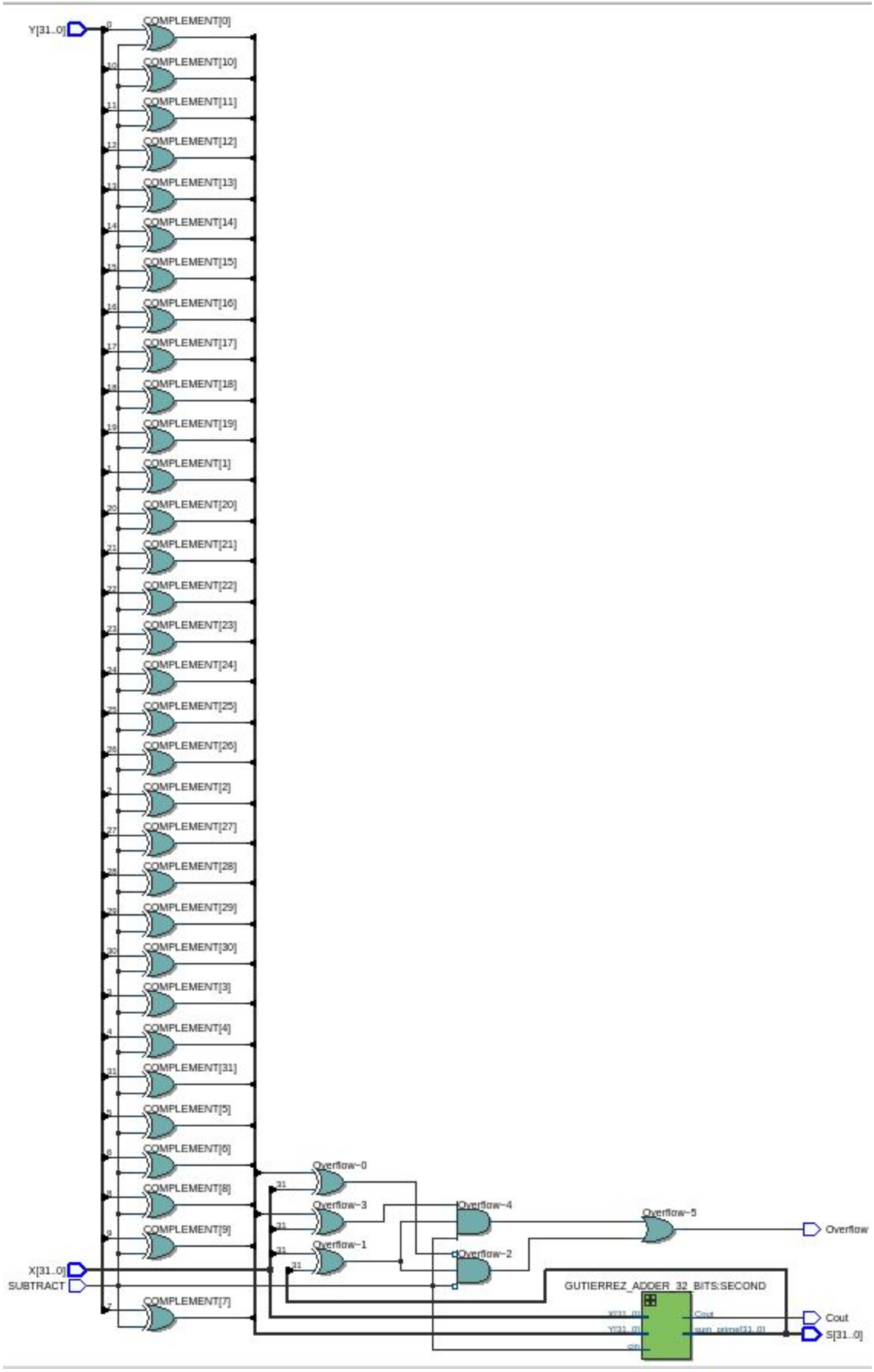
Y[31..0]
COMPLEMENT[0]
COMPLEMENT[10]
COMPLEMENT[11]
COMPLEMENT[12]
COMPLEMENT[13]
COMPLEMENT[14]
COMPLEMENT[15]
COMPLEMENT[16]
COMPLEMENT[17]
COMPLEMENT[18]
COMPLEMENT[19]
COMPLEMENT[1]
COMPLEMENT[20]
COMPLEMENT[21]
COMPLEMENT[22]
COMPLEMENT[23]
COMPLEMENT[24]
COMPLEMENT[25]
COMPLEMENT[26]
COMPLEMENT[2]
COMPLEMENT[27]
COMPLEMENT[28]
COMPLEMENT[29]
COMPLEMENT[30]
COMPLEMENT[3]
COMPLEMENT[4]
COMPLEMENT[31]
COMPLEMENT[5]
COMPLEMENT[6]
COMPLEMENT[8]
COMPLEMENT[9]
COMPLEMENT[7]
X[31..0]
SUBTRACT
Overflow~0
Overflow~3
Overflow~1
Overflow~4
Overflow~2
Overflow~5
Overflow
GUTIERREZ_ADDER_32_BITS:SECOND
Cout
S[31..0]

Figure 52: Block diagram for 32 bit adder subtractor.

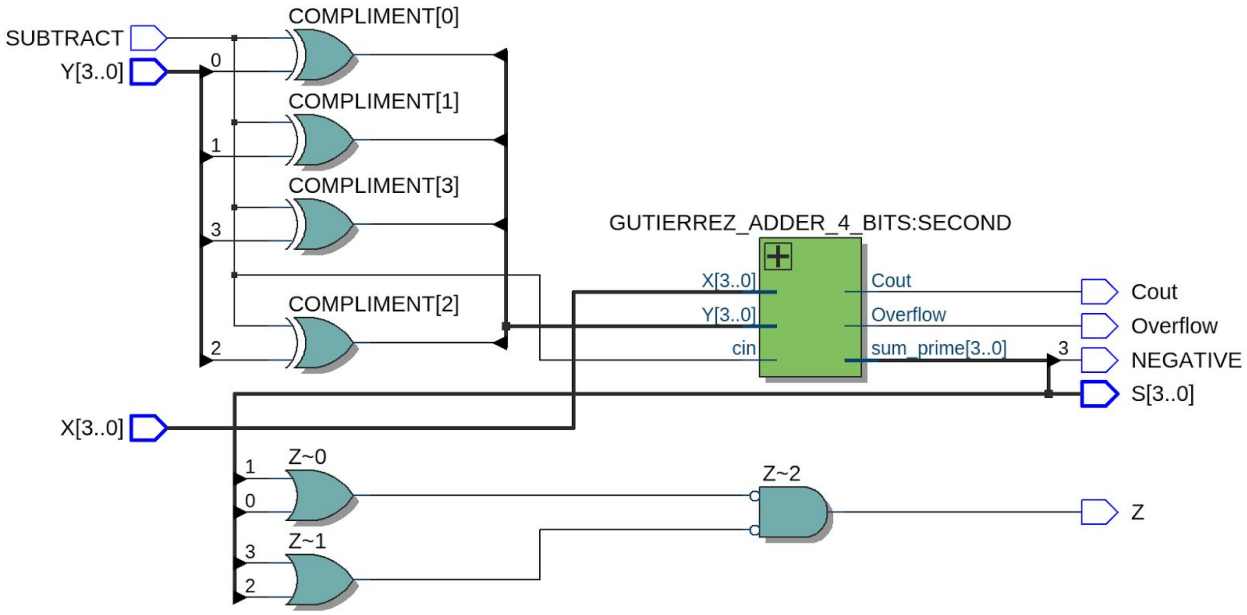

Figure 53: Block diagram for 4 bit comparator advanced.

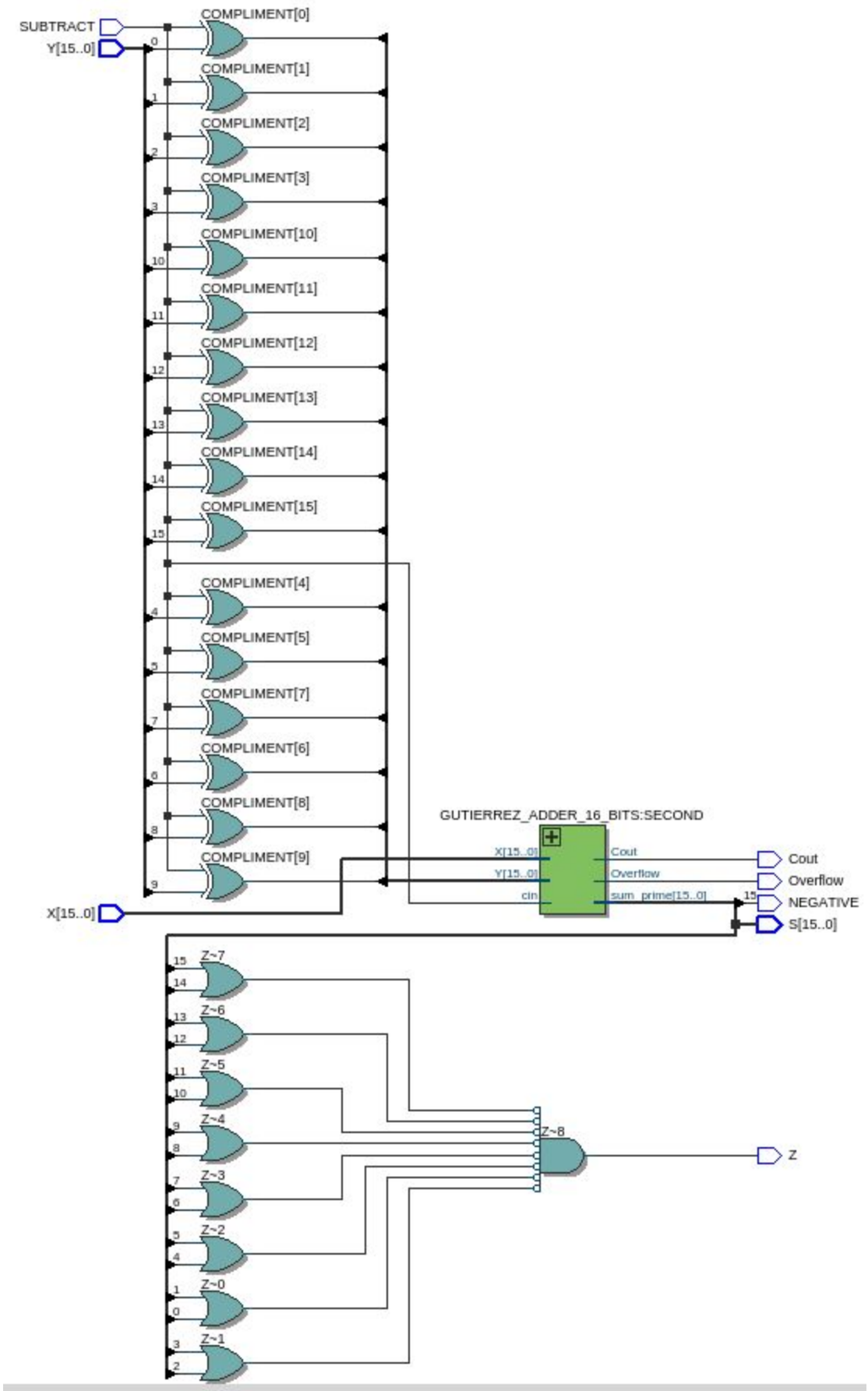
SUBTRACT
Y[15..0]
COMPLIMENT[0]
COMPLIMENT[1]
COMPLIMENT[2]
COMPLIMENT[3]
COMPLIMENT[10]
COMPLIMENT[11]
COMPLIMENT[12]
COMPLIMENT[13]
COMPLIMENT[14]
COMPLIMENT[15]
COMPLIMENT[4]
COMPLIMENT[5]
COMPLIMENT[7]
COMPLIMENT[6]
COMPLIMENT[8]
COMPLIMENT[9]
X[15..0]
GUTIERREZ_ADDER_16_BITS:SECOND
X[15..0]
Y[15..0]
cin
Cout
Overflow
sum_prime[15..0]
Cout
Overflow
NEGATIVE
S[15..0]
Z~7
Z~6
Z~5
Z~4
Z~3
Z~2
Z~0
Z~1
Z~8
Z

Figure 54: Block diagram for 16 bit comparator advanced.

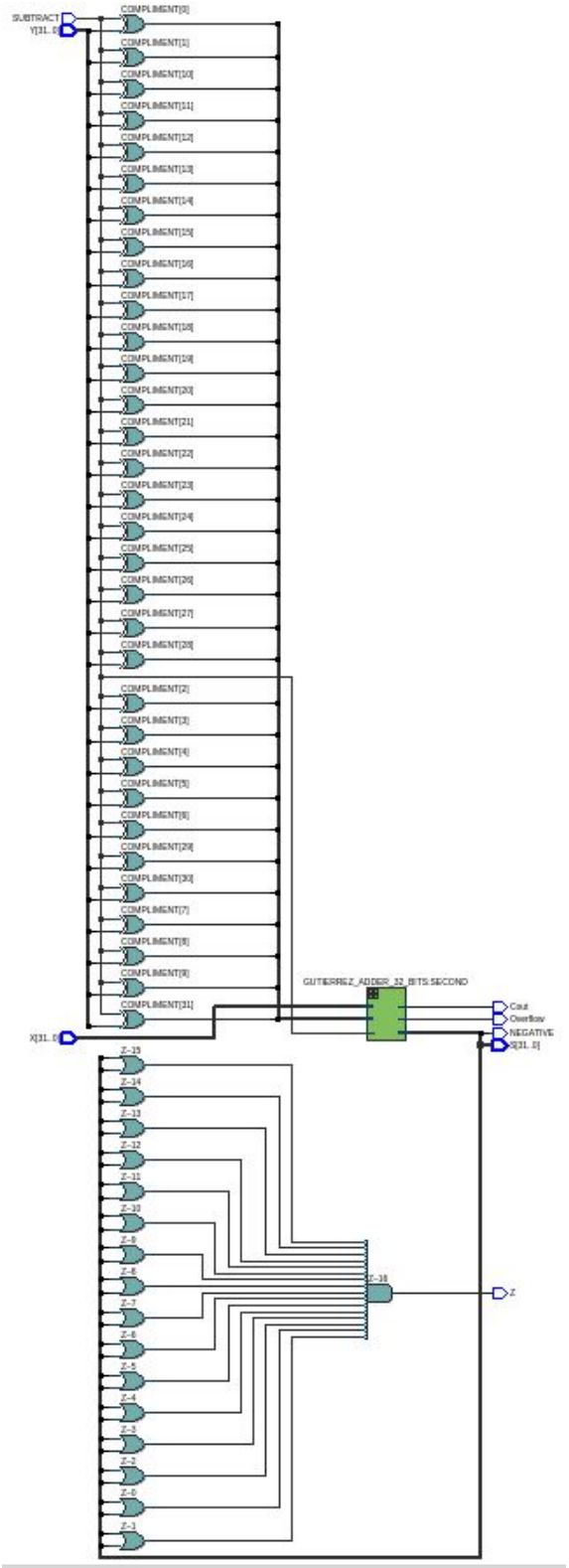

Figure 55: Block diagram for 32 bit comparator advanced.

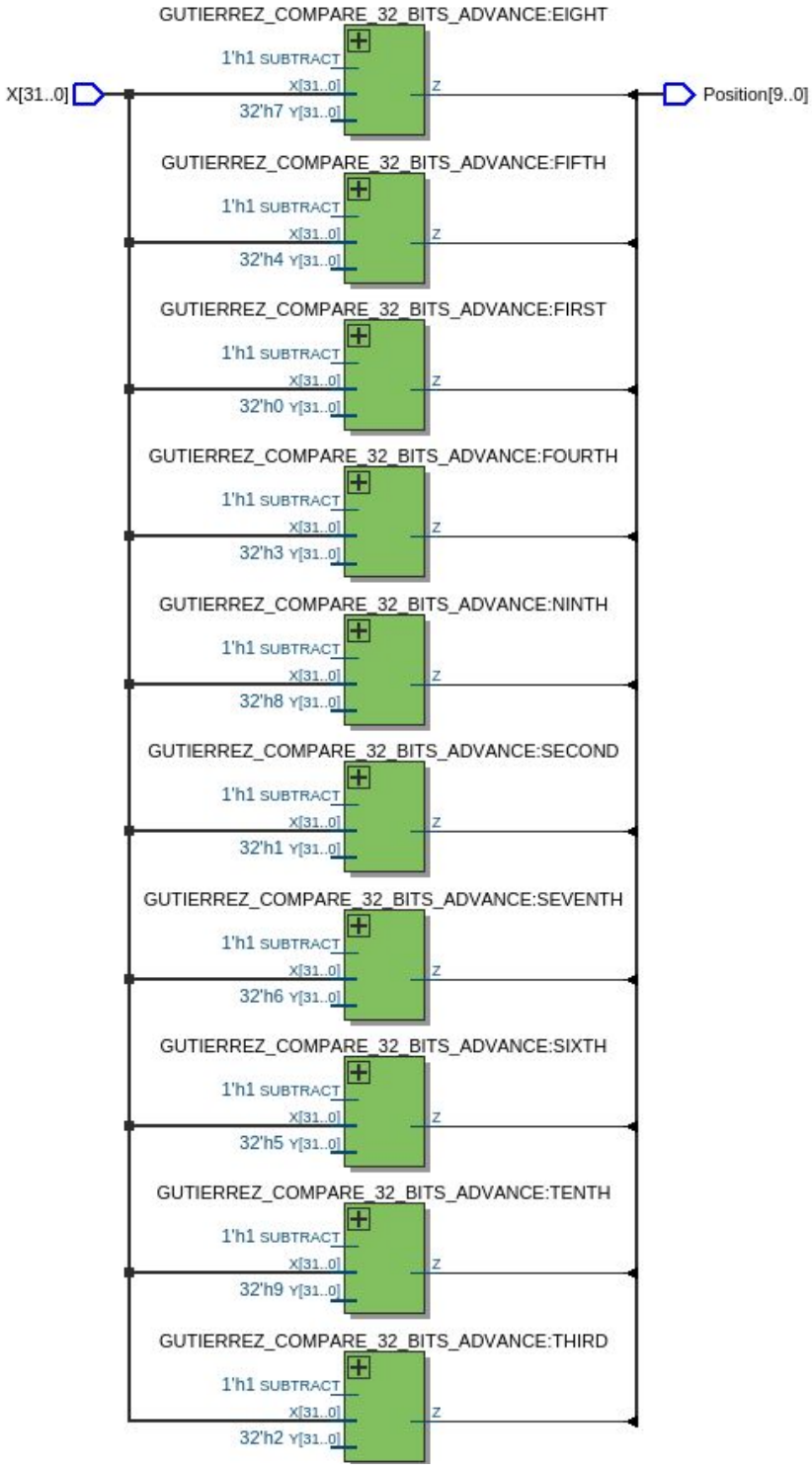

Figure 56: Block diagram for array of 32 bit words.