

Laboratory Project 3: Adder

MdShahid Bin Emdad

March 6th, 2022

CSC 34200/34300

Table of Contents

Objective:	2
Description of Specifications, and Functionality:	2
Code:	5
Simulation:	30
Conclusion:	36

Objective:

The objective of this assignment is to design and understand the specifications of adders and create waveforms for them. First, we were instructed to design half adder using two processes and then use the half adder as a component to create full adder 1 bit. Then we created 4-bit adder by using the half and full adder as a component. Then, we extended the work of 4bit adder by using a add/sub component. Then, we save our components in a package for future use. After that, we created N-bit design by using behavioral VHDL model and then use OVERFLOW, ZERO, NEGATIVE flags. Then, I used modelsim to verify all my designs, for example, $N = 4$ and $N = 32$. Then, I created N-bit adder/subtractor unit using lpm and compare waveforms with my designs. Finally, I created a testbench VHDL file to test Add_SUB unit for $n=16$ bits.

Description of Specifications, and Functionality:

The digital system I used in this assignment is Quartus Prime 20.1.1 and ModelSimSetup-20.1.1. There two packages needed are cyclonev and cyclonevi (both versions are 20.1.1). In the VHDL editor, I wrote my VHDL code to get the circuit output and then used Modelsim to simulate and run my circuit over time (ns unit).

Specifications:

1. Half Adder Ports:

I used two inputs one for the first operand of the adder and other one is the second operand of the adder. I also used carry which is the bit that is carried in from the next less significant stage.

2. Full Adder Ports:

I used two inputs one for the first operand of the adder and other one is the second operand of the adder. I also used cin which is the bit that is carried in from the next less significant stage. I used cout for the output which is bit that is carried over from an addition when necessary and sum is the last bit of the result.

3. 4-bit Adder Ports:

I used two inputs one for the first operand of the adder and other one is the second operand of the adder. I also used cin which is the bit that is carried in from the next less significant stage. I used cout for the output which is bit that is carried over from an addition when necessary and sum is the last bit of the result.

4. 16-bit Adder Ports:

I used 2 cins, (cin1 and cin2) which are the first and second operand of the adder for the 16 bits. I also had op which is the bit that carried in from the less

significant stage. For the outputs, I had neg which is the carry output that carried in from addition, sum (sum of the two inputs, cin1 and cin2), overflow which checks for overflow errors and finally zero for two input addition.

5. 32-bit Adder Ports:

I used two operands. The first operand of the adder and other one is the second operand of the adder. Then, I used obit which is the bit that carried from the less significant stage. I had carry for the bit output from addition, sum for the inputs addition, overflow which checks for overflow errors and finally zero for two input addition.

Functionality:

1. Half Adder truth table:

		Carry	Sum
x	y	c	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 1: Half Adder truth table

2. Full Adder truth table:

Inputs			Outputs	
A	B	C - IN	Sum	C - OUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 2: Full Adder truth table

3. 4-bit Adder truth table:

Cin	A				B				Sum				Carry
	A3	A2	A1	A0	B3	B2	B1	B0	S3	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	1	1	0	1	1	0	0
0	0	1	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0	1	1	0	1	0	0
0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	1	1	1	0	1	1	1	1	1	1	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	1	0	1	1	0	1
0	1	1	0	0	1	1	0	0	1	0	0	0	1
0	1	1	0	1	1	1	0	1	1	0	1	0	1
0	1	1	1	0	1	1	1	0	1	1	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3: 4-bit Full Adder truth table

4. 4-bit Add/Sub Truth Table:

	B3	B2	B1	B0	M	X3	X2	X1	X0
Addition	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	0	1
	0	0	1	0	0	0	0	1	0
	0	0	1	1	0	0	0	1	1
	0	1	0	0	0	0	1	0	0
	0	1	0	1	0	0	1	0	1
	0	1	1	0	0	0	1	1	0
	0	1	1	1	0	0	1	1	1
	1	0	0	0	0	1	0	0	0
Subtraction	1	0	0	1	0	1	0	0	1
	0	0	0	0	1	1	0	0	1
	0	0	0	1	1	1	0	0	0
	0	0	1	0	1	0	1	1	1
	0	0	1	1	1	0	1	1	0
	0	1	0	0	1	0	1	0	1
	0	1	0	1	1	0	1	0	0
	0	1	1	0	1	0	0	1	1
	0	1	1	1	1	0	0	1	0

Figure 4: 4-bit Add/Sub Truth Table

5. 16-bit Adder truth table:

By combining four 4-cycle CLAs, a 16-bit adder can be made yet extra rationale is required as an LCU. The LCU acknowledges the gathering propagate() and group generate() from every one of the four CLAs. furthermore, have the accompanying articulations for each CLA adder: The LCU then, at that point, creates the convey input for each CLA.

6. 32-bit Adder truth table:

To make a 32-bit adder, the necessity is 4 8-bits full adders. It will result in a 32bit sum and create carry outputs.

Code:

1. Task 1:

In figure 5, we can see the project summary for half adder.

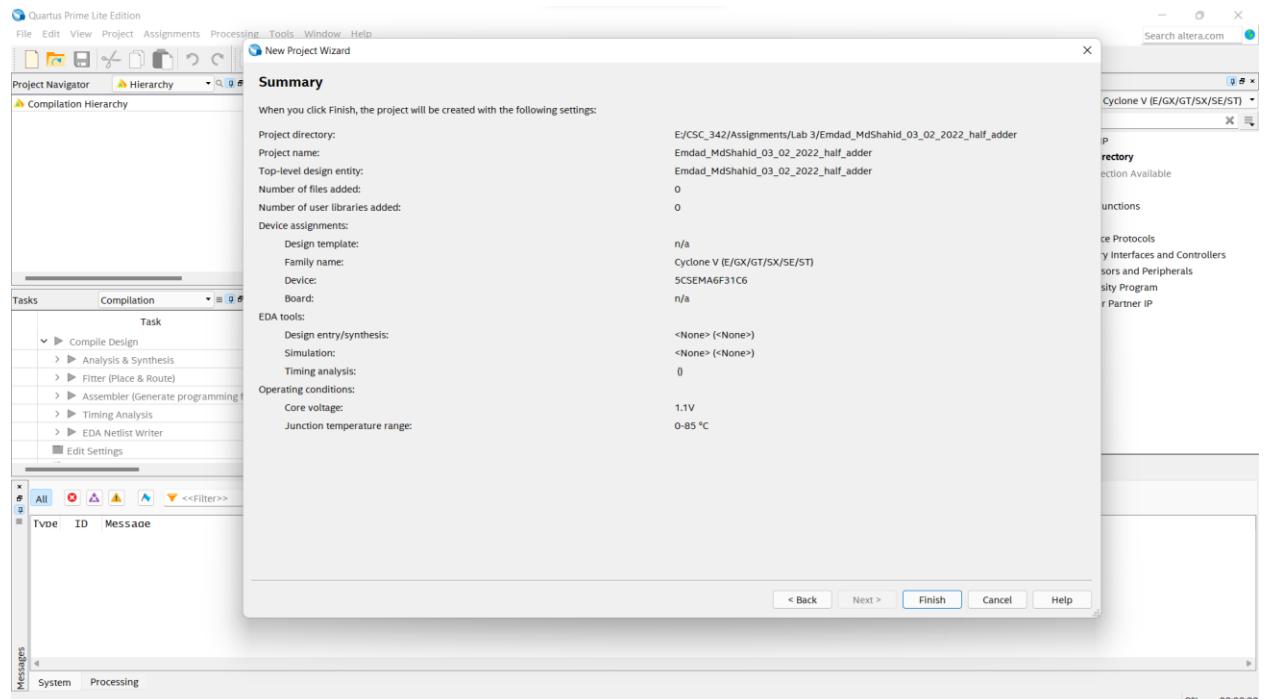


Figure 5: Half Adder Project Summary

In figure 6, we can see the VHDL code for the half adder.

Quartus Prime Lite Edition - E/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_half_adder/Emdad_MdShahid_03_02_2022_half_adder - Emdad_MdShahid_03_02_2022_half_adder

```

library ieee;
use ieee.std_logic_1164.all;
entity Emdad_MdShahid_03_02_2022_half_adder is
    port (Emdad_MdShahid_03_06_2022_A, Emdad_MdShahid_03_06_2022_B : in std_logic;
          Emdad_MdShahid_03_06_2022_Sum, Emdad_MdShahid_03_06_2022_Carry : out std_logic);
end Emdad_MdShahid_03_02_2022_half_adder;

architecture BEHAV_FA of Emdad_MdShahid_03_02_2022_half_adder is
begin
    -- Process P1 that defines the first half adder
    P1: process (Emdad_MdShahid_03_06_2022_A, Emdad_MdShahid_03_06_2022_B)
    begin
        Emdad_MdShahid_03_06_2022_Sum <= Emdad_MdShahid_03_06_2022_A xor Emdad_MdShahid_03_06_2022_B;
    end process;
    -- Process P2 that defines the second half adder and the OR gate
    P2: process (Emdad_MdShahid_03_06_2022_A, Emdad_MdShahid_03_06_2022_B)
    begin
        Emdad_MdShahid_03_06_2022_Carry <= Emdad_MdShahid_03_06_2022_A and Emdad_MdShahid_03_06_2022_B;
    end process;
end BEHAV_FA;

```

Tasks

Task	Time
Compile Design	00:01:29
Analysis & Synthesis	00:00:11
Fitter (Place & Route)	00:00:58
Assembler (Generate ...)	00:00:13
Timing Analysis	00:00:07
EDA Netlist Writer	
Edit Settings	
Program Device (Open Pr...)	

Messages

Type	ID	Message
Info	332154	The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
Info	332140	No Setup paths to report
Info	332140	No Hold paths to report
Info	332140	No Recovery paths to report
Info	332140	No Removal paths to report
Info	332140	No Minimum Pulse Width paths to report
Info	332102	Design is not fully constrained for setup requirements
Info	332102	Design is not fully constrained for hold requirements
Info	332102	Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
Info	293000	Quartus Prime Full Compilation was successful. 0 errors, 14 warnings

Figure 6: Half Adder VHDL Code

In figure 7, we can see the successful compilation report for the half adder.

Quartus Prime Lite Edition - E/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_half_adder/Emdad_MdShahid_03_02_2022_half_adder - Emdad_MdShahid_03_02_2022_half_adder

Flow Summary

Flow Status	Successful - Wed Mar 02 23:08:06 2022
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	Emdad_MdShahid_03_02_2022_half_adder
Top-level Entity Name	Cyclone V
Family	5CSEMA6F31C6
Device	Final
Timing Models	2 / 41,910 (< 1 %)
Logic utilization in ALMs	0
Total registers	
Total pins	4 / 457 (< 1 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCss	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCss	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Tasks

Task	Time
Compile Design	00:01:29
Analysis & Synthesis	00:00:11
Fitter (Place & Route)	00:00:58
Assembler (Generate ...)	00:00:13
Timing Analysis	00:00:07
EDA Netlist Writer	
Edit Settings	
Program Device (Open Pr...)	

Messages

Type	ID	Message
Info	332154	The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
Info	332140	No Setup paths to report
Info	332140	No Hold paths to report
Info	332140	No Recovery paths to report
Info	332140	No Removal paths to report
Info	332140	No Minimum Pulse Width paths to report
Info	332102	Design is not fully constrained for setup requirements
Info	332102	Design is not fully constrained for hold requirements
Info	332102	Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
Info	293000	Quartus Prime Full Compilation was successful. 0 errors, 14 warnings

Figure 7: Half Adder successful compilation report

2. TASK 2:

In figure 8, we can see the full adder project summary.

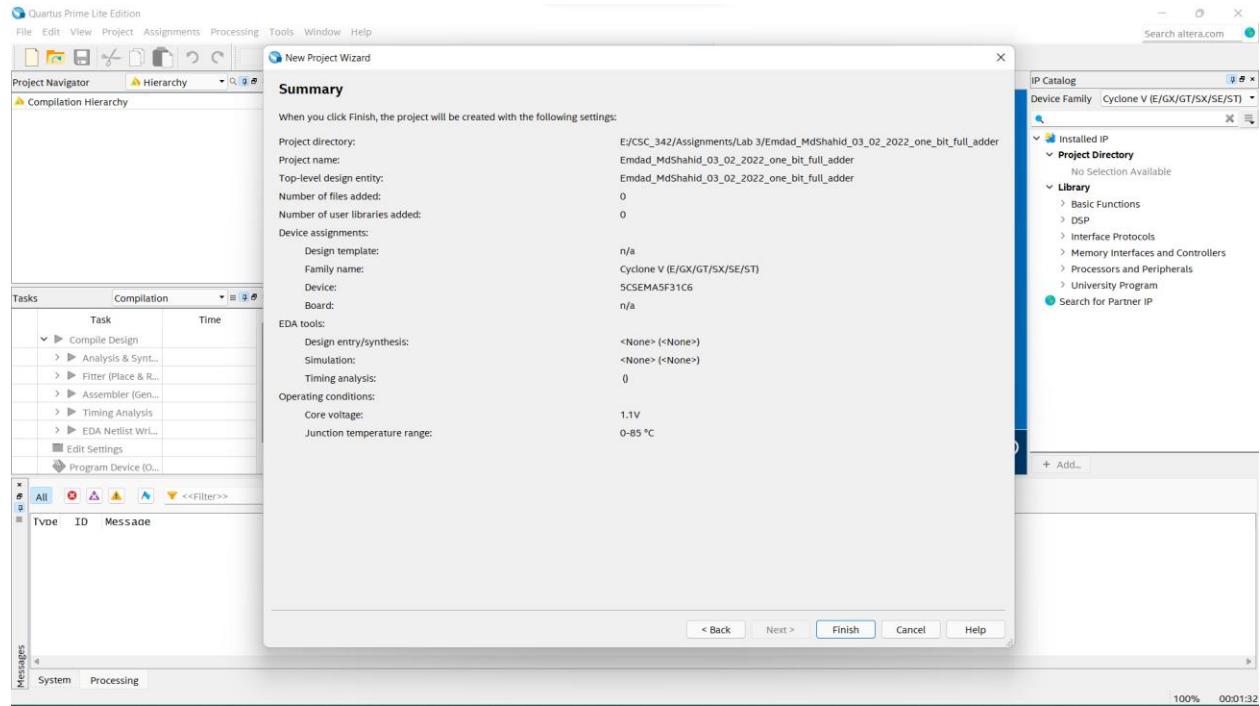


Figure 8: 1-bit Full Adder Project Summary

In figure 9, I used the half adder as a component for the 1-bit full adder.

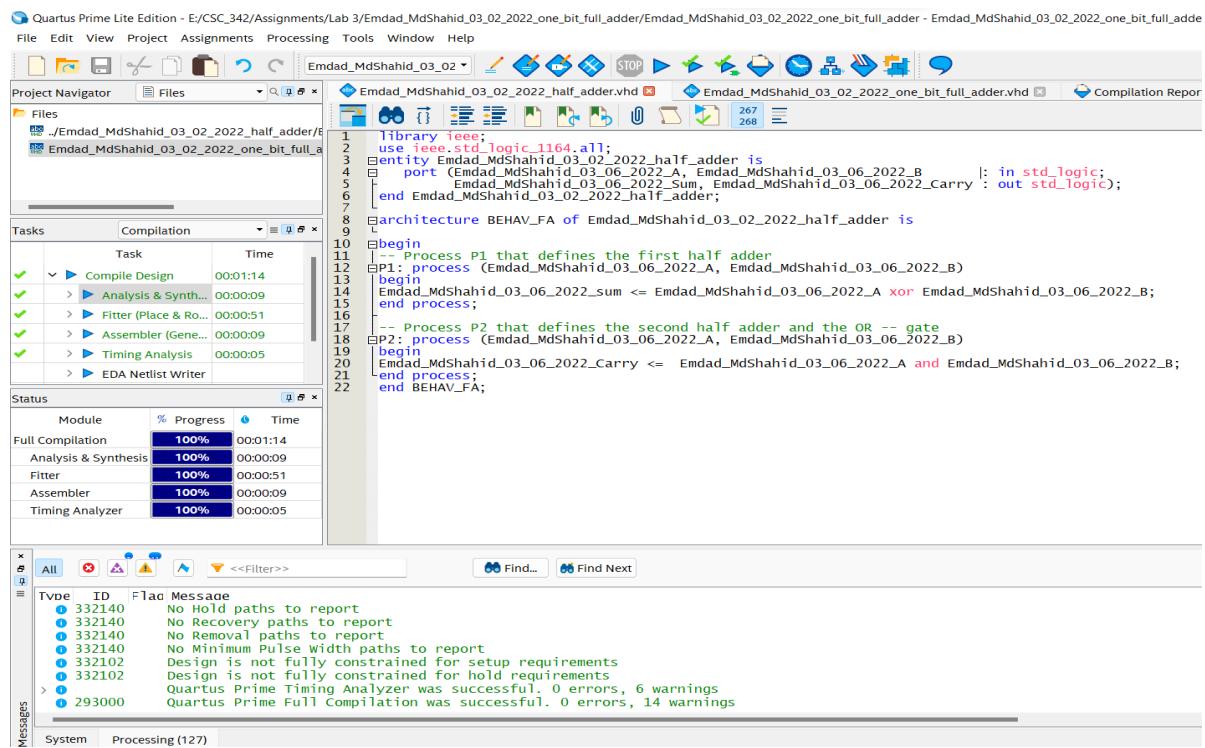


Figure 9: halfAdder VHDL Code as a component

In figure 10, we can see the HDL code for the 1-bit full adder where I used half adder as a component.

The screenshot shows the Quartus Prime Lite Edition interface with the following details:

- Project Navigator:** Shows files: Emdad_MdShahid_03_02_2022_half_adder.vhd and Emdad_MdShahid_03_02_2022_one_bit_full_adder.vhd.
- Tasks:** Compilation tasks completed successfully:
 - Compile Design: 00:01:14
 - Analysis & Synthesis: 00:00:09
 - Fitter (Place & Route): 00:00:51
 - Assembler (Generate): 00:00:09
 - Timing Analysis: 00:00:05
 - EDA Netlist Writer
- Status:** Module compilation progress:

Module	% Progress	Time
Full Compilation	100%	00:01:14
Analysis & Synthesis	100%	00:00:09
Filter	100%	00:00:51
Assembler	100%	00:00:09
Timing Analyzer	100%	00:00:05
- Messages:** A list of messages from the Quartus Prime Timing Analyzer and Full Compilation:

Type	ID	Flag	Message
	332140		No Hold paths to report
	332140		No Recovery paths to report
	332140		No Removal paths to report
	332140		No Minimum Pulse Width paths to report
	332102		Design is not fully constrained for setup requirements
	332102		Design is not fully constrained for hold requirements
>	293000		Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
	293000		Quartus Prime Full Compilation was successful. 0 errors, 14 warnings

Figure 10: 1-bit full Adder VHDL Code

In figure 11, we can see the successful compilation report for 1-bit full adder.

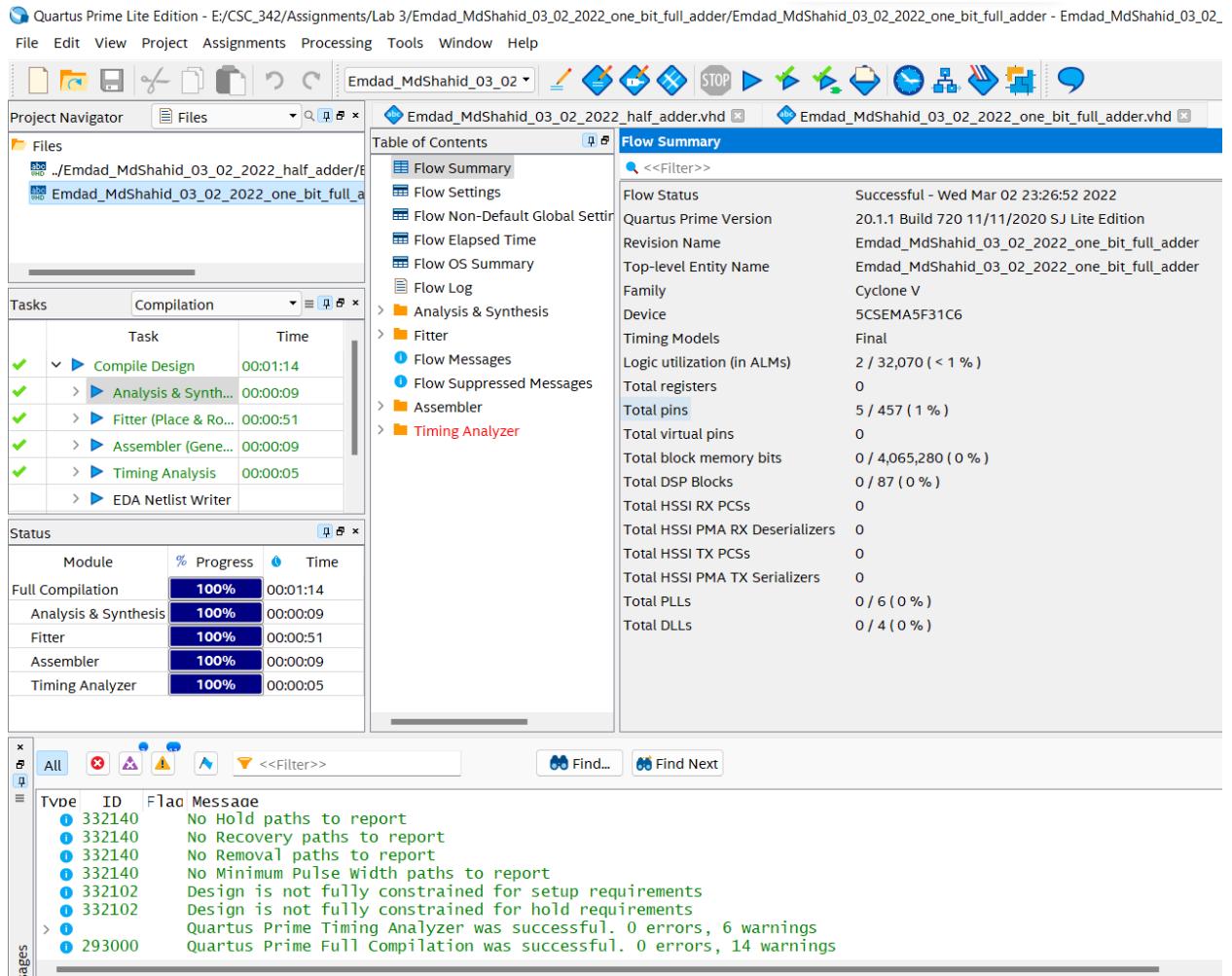


Figure 11: 1-bit full adder successful compilation report

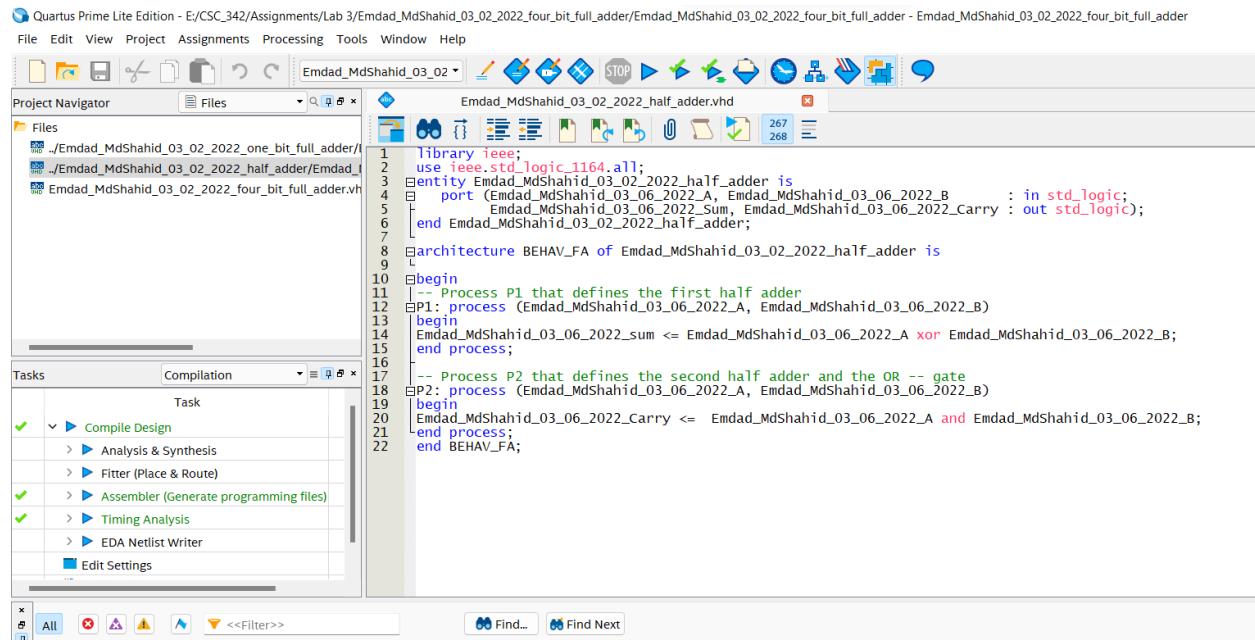
3. TASK 3:

In figure 12, we can see the project summary for 4-bit full adder.



Figure 12: 4-bit full adder Project Summary

In figure 13, we can see I used half adder as a component.



The screenshot shows the Quartus Prime Lite Edition interface with the project "Emdad_MdShahid_03_02" open. The main window displays the VHDL code for a half adder component:

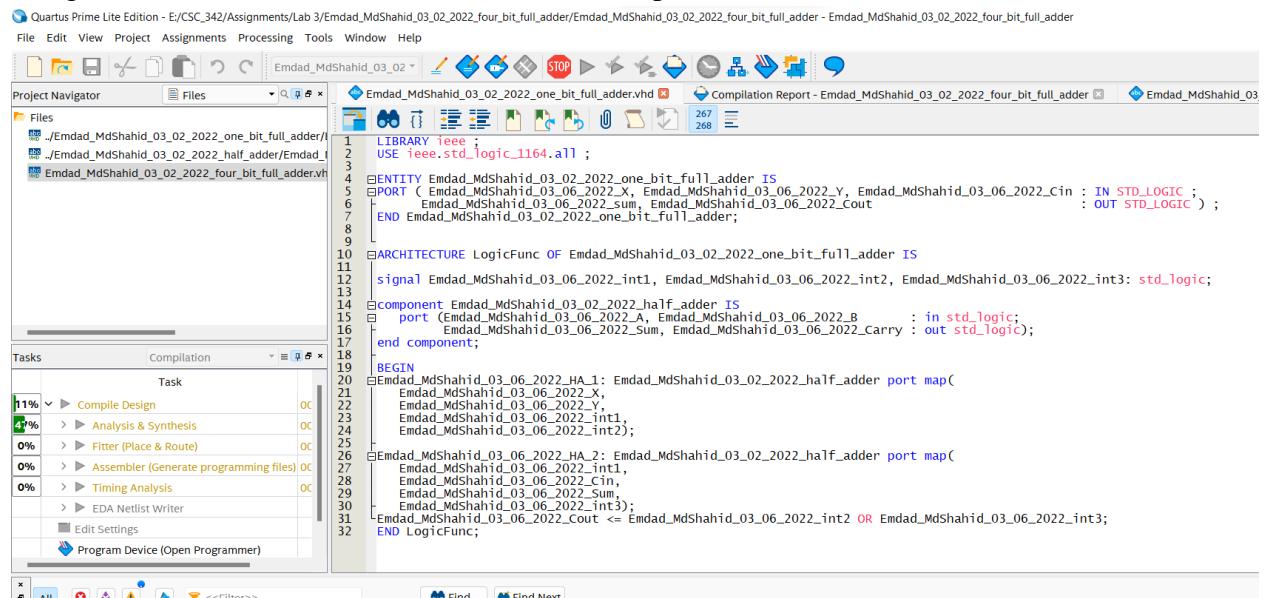
```

1 LIBRARY IEEE;
2 USE ieee.std_logic_1164.all;
3 ENTITY Emdad_MdShahid_03_02_2022_half_adder IS
4   PORT (Emdad_MdShahid_03_06_2022_A, Emdad_MdShahid_03_06_2022_B : IN STD_LOGIC;
5         Emdad_MdShahid_03_06_2022_Sum, Emdad_MdShahid_03_06_2022_Carry : OUT STD_LOGIC);
6 END Emdad_MdShahid_03_02_2022_half_adder;
7
8 ARCHITECTURE BEHAV_FA OF Emdad_MdShahid_03_02_2022_half_adder IS
9
10 BEGIN
11   PROCESS P1 THAT DEFINES THE FIRST HALF ADDER
12   BEGIN
13     Emdad_MdShahid_03_06_2022_Sum <= Emdad_MdShahid_03_06_2022_A XOR Emdad_MdShahid_03_06_2022_B;
14   END PROCESS;
15
16   -- PROCESS P2 THAT DEFINES THE SECOND HALF ADDER AND THE OR -- GATE
17   PROCESS P2 (Emdad_MdShahid_03_06_2022_A, Emdad_MdShahid_03_06_2022_B)
18   BEGIN
19     Emdad_MdShahid_03_06_2022_Carry <= Emdad_MdShahid_03_06_2022_A AND Emdad_MdShahid_03_06_2022_B;
20   END PROCESS;
21 END BEHAV_FA;
22 
```

The Project Navigator shows files like "Emdad_MdShahid_03_02_2022_one_bit_full_adder.vhd" and "Emdad_MdShahid_03_02_2022_half_adder.vhd". The Tasks panel indicates successful compilation of the design.

Figure 13: half adder as a component

In figure 13, we can see I used full adder as a component.



The screenshot shows the Quartus Prime Lite Edition interface with the project "Emdad_MdShahid_03_02" open. The main window displays the VHDL code for a full adder component:

```

1 LIBRARY IEEE;
2 USE ieee.std_logic_1164.all ;
3
4 ENTITY Emdad_MdShahid_03_02_2022_one_bit_full_adder IS
5   PORT (Emdad_MdShahid_03_06_2022_X, Emdad_MdShahid_03_06_2022_Y, Emdad_MdShahid_03_06_2022_Cin : IN STD_LOGIC ;
6         Emdad_MdShahid_03_06_2022_Sum, Emdad_MdShahid_03_06_2022_Cout : OUT STD_LOGIC ) ;
7 END Emdad_MdShahid_03_02_2022_one_bit_full_adder;
8
9
10 ARCHITECTURE LogicFunc OF Emdad_MdShahid_03_02_2022_one_bit_full_adder IS
11
12   SIGNAL Emdad_MdShahid_03_06_2022_int1, Emdad_MdShahid_03_06_2022_int2, Emdad_MdShahid_03_06_2022_int3: STD_LOGIC;
13
14   COMPONENT Emdad_MdShahid_03_02_2022_half_adder IS
15     PORT (Emdad_MdShahid_03_06_2022_A, Emdad_MdShahid_03_06_2022_B : IN STD_LOGIC;
16           Emdad_MdShahid_03_06_2022_Sum, Emdad_MdShahid_03_06_2022_Carry : OUT STD_LOGIC);
17   END COMPONENT;
18
19 BEGIN
20   EMDAD_MDSHAHID_03_06_2022_HA_1: Emdad_MdShahid_03_02_2022_half_adder PORT MAP(
21     Emdad_MdShahid_03_06_2022_X,
22     Emdad_MdShahid_03_06_2022_Y,
23     Emdad_MdShahid_03_06_2022_int1,
24     Emdad_MdShahid_03_06_2022_int2);
25
26   EMDAD_MDSHAHID_03_06_2022_HA_2: Emdad_MdShahid_03_02_2022_half_adder PORT MAP(
27     Emdad_MdShahid_03_06_2022_int1,
28     Emdad_MdShahid_03_06_2022_Cin,
29     Emdad_MdShahid_03_06_2022_Sum,
30     Emdad_MdShahid_03_06_2022_int3);
31   Emdad_MdShahid_03_06_2022_Cout <= Emdad_MdShahid_03_06_2022_int2 OR Emdad_MdShahid_03_06_2022_int3;
32 END LogicFunc;
33 
```

The Project Navigator shows files like "Emdad_MdShahid_03_02_2022_one_bit_full_adder.vhd" and "Emdad_MdShahid_03_02_2022_half_adder.vhd". The Tasks panel shows the status of the compilation process.

Figure 14: full adder as a component

In figure 15, we can see the 4-bit full adder VHDL code.

```

library IEEE;
use ieee.std_logic_1164.all;

entity Emdad_MdShahid_03_02_2022_four_bit_full_adder is
    port (Emdad_MdShahid_03_02_2022_four_bit_full_adder_X : in std_logic_vector(3 downto 0); Emdad_MdShahid_03_02_2022_four_bit_full_adder_Y : in std_logic_vector(3 downto 0); Emdad_MdShahid_03_02_2022_four_bit_full_adder_Cin : in std_logic; Emdad_MdShahid_03_02_2022_four_bit_full_adder_Cout : out std_logic);
end Emdad_MdShahid_03_02_2022_four_bit_full_adder;

architecture fouradder_structure of Emdad_MdShahid_03_02_2022_four_bit_full_adder is
begin
    signal Emdad_MdShahid_03_02_2022_four_bit_full_adder_C : std_logic_vector(2 downto 0);
    component Emdad_MdShahid_03_02_2022_one_bit_full_adder
        PORT (Emdad_MdShahid_03_06_2022_X, Emdad_MdShahid_03_06_2022_Y, Emdad_MdShahid_03_06_2022_Cin : IN STD_LOGIC; Emdad_MdShahid_03_06_2022_Sum, Emdad_MdShahid_03_06_2022_Cout : OUT STD_LOGIC);
    end component;
    begin
        Emdad.MdShahid.03.02.2022.four_bit_full_adder_FA0: Emdad.MdShahid.03.02.2022.one_bit_full_adder
            port map (Emdad.MdShahid.03.02.2022.four_bit_full_adder_X(0), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Y(0), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Cin, Emdad.MdShahid.03.02.2022.four_bit_full_adder_C0(0), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Sum(0));
        Emdad.MdShahid.03.02.2022.four_bit_full_adder_FA1: Emdad.MdShahid.03.02.2022.one_bit_full_adder
            port map (Emdad.MdShahid.03.02.2022.four_bit_full_adder_X(1), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Y(1), Emdad.MdShahid.03.02.2022.four_bit_full_adder_C1(0), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Sum(1));
        Emdad.MdShahid.03.02.2022.four_bit_full_adder_FA2: Emdad.MdShahid.03.02.2022.one_bit_full_adder
            port map (Emdad.MdShahid.03.02.2022.four_bit_full_adder_X(2), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Y(2), Emdad.MdShahid.03.02.2022.four_bit_full_adder_C2(0), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Sum(2));
        Emdad.MdShahid.03.02.2022.four_bit_full_adder_FA3: Emdad.MdShahid.03.02.2022.one_bit_full_adder
            port map (Emdad.MdShahid.03.02.2022.four_bit_full_adder_X(3), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Y(3), Emdad.MdShahid.03.02.2022.four_bit_full_adder_C3(0), Emdad.MdShahid.03.02.2022.four_bit_full_adder_Sum(3));
        end fouradder_structure;
    
```

Figure 15: 4-bit full adder VHDL code.

In figure 17, we can see the successful compilation report for 4-bit full adder.

Flow Status	Successful - Thu Mar 03 01:18:17 2022
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	Emdad_MdShahid_03_02_2022_four_bit_full_adder
Top-level Entity Name	Emdad_MdShahid_03_02_2022_four_bit_full_adder
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	4 / 32,070 (< 1 %)
Total registers	0
Total pins	14 / 457 (3 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSS	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSS	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Messages

Type	ID	Tac	Message
Info	332154	The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.	
Info	332140	No Setup paths to report	
Info	332140	No Hold paths to report	
Info	332140	No Removal paths to report	
Info	332140	No Removal paths to report	
Info	332240	No Minimum Pulse width paths to report	
Info	332102	Design is not fully constrained for setup requirements	
Info	332102	Design is not fully constrained for hold requirements	
Info	293000	Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings	
Info	293000	Quartus Prime Full Compilation was successful. 0 errors, 14 warnings	

4. TASK 4:

In figure 17, we can see the project summary for 4-bit add/sub.

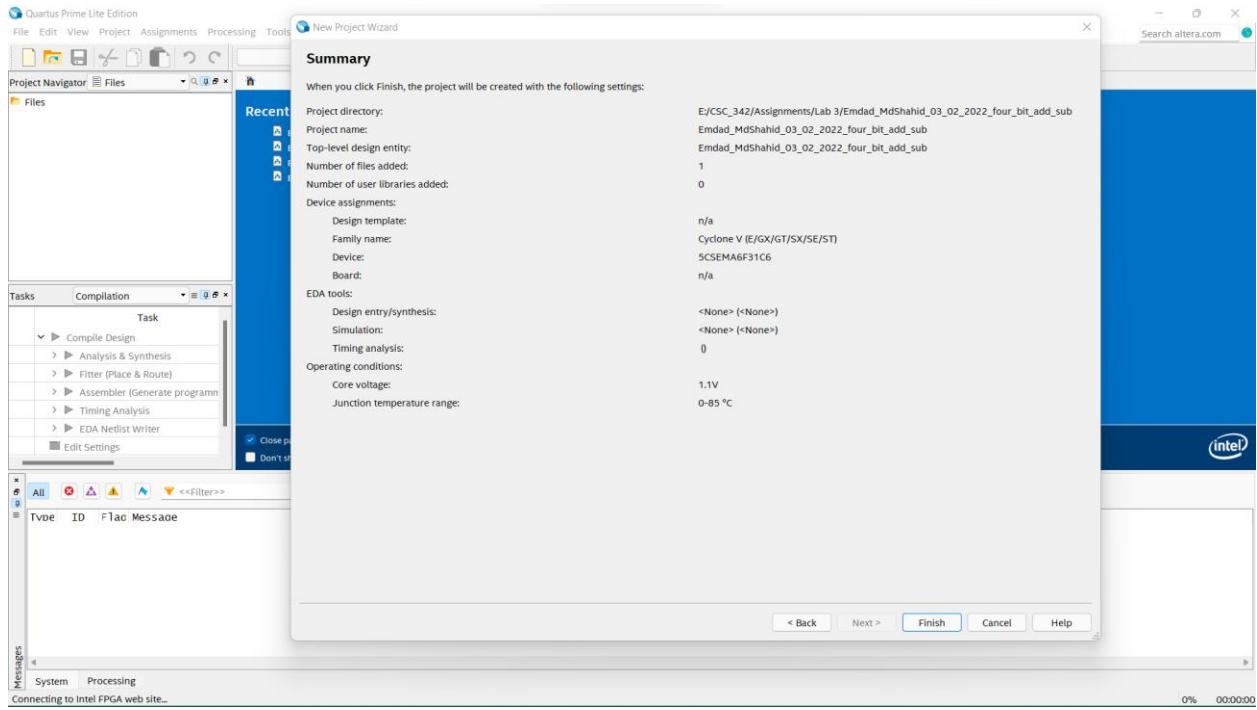


Figure 16: project summary for 4-bit add/sub.

In figure 17, we can see the VHDL code for the 4-bit add/sub.

```

Text Editor - E/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_four_bit_add_sub/Emdad_MdShahid_03_02_2022_four_bit_add_sub - Emdad_MdShahid_03_02_2022_four_bit_add_sub - [Emdad_MdShahid_03_02_2022_four_bit_add_sub.vhd]*

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Emdad_MdShahid_03_02_2022_four_bit_add_sub IS
    PORT (Emdad_MdShahid_03_02_2022_X :in std_logic_vector(3 downto 0);
          Emdad_MdShahid_03_02_2022_Y :in std_logic_vector(3 downto 0);
          Emdad_MdShahid_03_02_2022_Opcode :in std_logic;
          Emdad_MdShahid_03_02_2022_Cin :out std_logic_vector (3 downto 0);
          Emdad_MdShahid_03_02_2022_Cout :out std_logic;
          Emdad_MdShahid_03_06_2022_Overflow :out std_logic);
END Emdad_MdShahid_03_02_2022_four_bit_add_sub;

ARCHITECTURE LogicFunc OF Emdad_MdShahid_03_02_2022_four_bit_add_sub IS
COMPONENT Emdad_MdShahid_03_02_2022_one_bit_full_adder
    PORT (Emdad_MdShahid_03_06_2022_X, Emdad_MdShahid_03_06_2022_Y, Emdad_MdShahid_03_06_2022_Cin : IN STD_LOGIC;
          Emdad_MdShahid_03_06_2022_sum, Emdad_MdShahid_03_06_2022_Cout : OUT STD_LOGIC) ;
END COMPONENT;
signal Emdad_MdShahid_03_06_2022_A, Emdad_MdShahid_03_06_2022_B, Emdad_MdShahid_03_06_2022_C, Emdad_MdShahid_03_06_2022_D: std_logic;
signal Emdad_MdShahid_03_06_2022_E: std_logic_vector (3 downto 0);
BEGIN
    Emdad_MdShahid_03_06_2022_E <= Emdad_MdShahid_03_02_2022_X XOR Emdad_MdShahid_03_02_2022_Y;
    Emdad_MdShahid_03_02_2022_four_bit_add_sub_FA0: Emdad_MdShahid_03_02_2022_one_bit_full_adder
        port map(Emdad_MdShahid_03_06_2022_X(3),
                  Emdad_MdShahid_03_06_2022_E(0),
                  Emdad_MdShahid_03_02_2022_opcode,
                  Emdad_MdShahid_03_02_2022_sum(0),
                  Emdad_MdShahid_03_06_2022_A);
    Emdad_MdShahid_03_02_2022_four_bit_add_sub_FA1: Emdad_MdShahid_03_02_2022_one_bit_full_adder
        port map(Emdad_MdShahid_03_06_2022_X(1),
                  Emdad_MdShahid_03_06_2022_E(1),
                  Emdad_MdShahid_03_02_2022_A,
                  Emdad_MdShahid_03_09_2022_sum(1),
                  Emdad_MdShahid_03_06_2022_B);
    Emdad_MdShahid_03_02_2022_four_bit_add_sub_FA2: Emdad_MdShahid_03_02_2022_one_bit_full_adder
        port map(Emdad_MdShahid_03_02_2022_X(2),
                  Emdad_MdShahid_03_06_2022_E(2),
                  Emdad_MdShahid_03_06_2022_B,
                  Emdad_MdShahid_03_02_2022_sum(2),
                  Emdad_MdShahid_03_06_2022_C);
    Emdad_MdShahid_03_02_2022_four_bit_add_sub_FA3: Emdad_MdShahid_03_02_2022_one_bit_full_adder
        port map(Emdad_MdShahid_03_02_2022_X(3),
                  Emdad_MdShahid_03_06_2022_E(3),
                  Emdad_MdShahid_03_06_2022_C,
                  Emdad_MdShahid_03_02_2022_sum(3),
                  Emdad_MdShahid_03_06_2022_D);
    Emdad_MdShahid_03_06_2022_Overflow <= Emdad_MdShahid_03_06_2022_C XOR Emdad_MdShahid_03_06_2022_D;
    Emdad_MdShahid_03_02_2022_Cout <= Emdad_MdShahid_03_06_2022_D;
END LogicFunc;

```

Figure 17: 4-bit add/sub VHDL code

In figure 18, we can see the 4-bit add/sub VHDL successfully compiled.

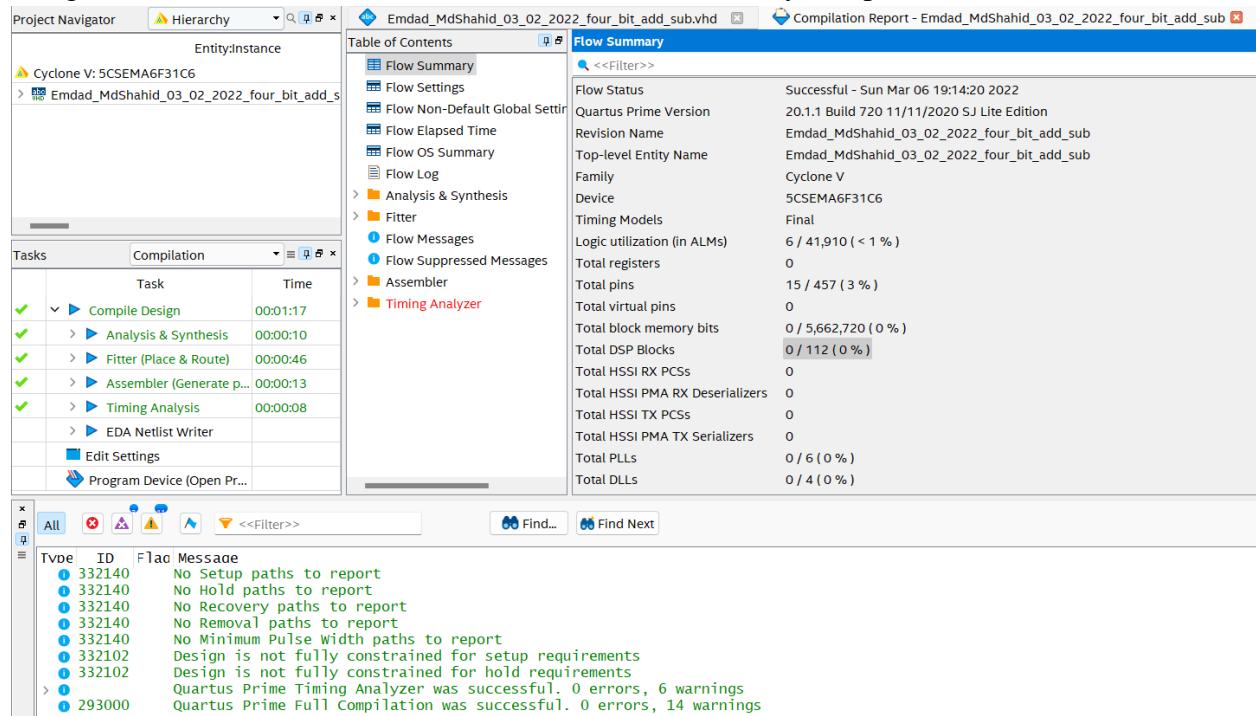


Figure 18: successful compilation report:

5. Task 5:

In figure 19, we can see the package project summary.

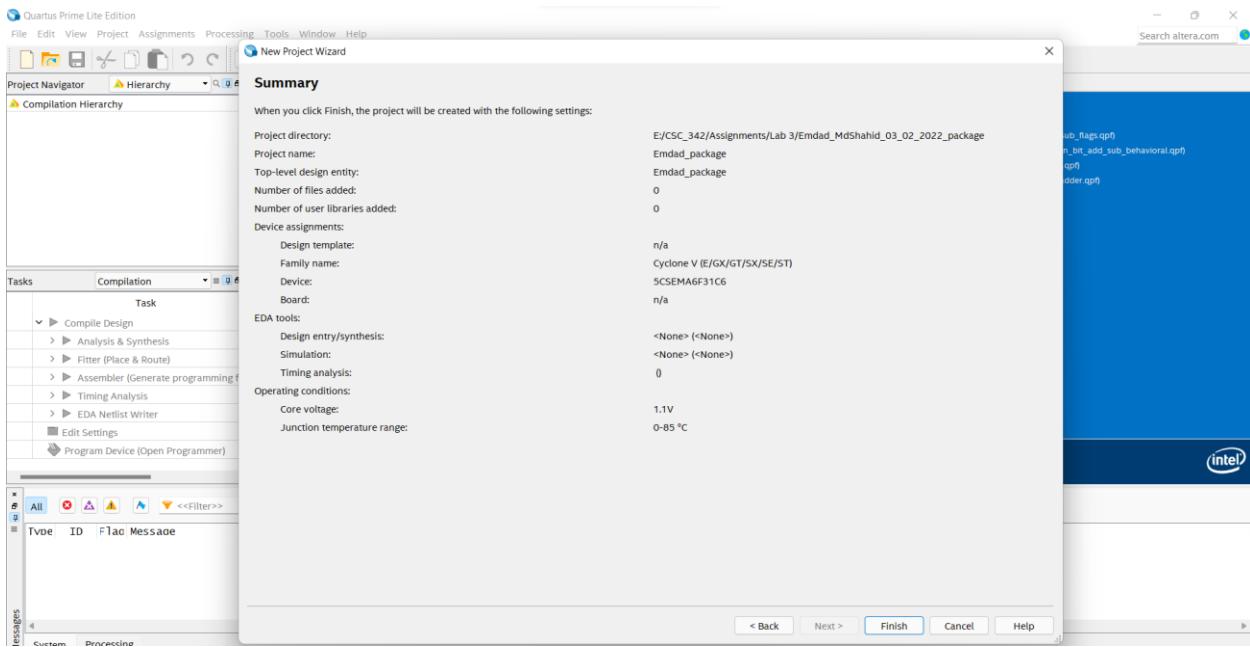


Figure 19: package project summary

In figure 20, I added my components in a package and compiled it and created a package for future use.

```

1 Library ieee;
2 USE ieee.std_logic_vector.all;
3
4 package Emdad_package is
5
6   component Emdad
7     port (Emdad_cin, Emdad_x, Emdad_y : in std_logic;
8           Emdad_s, Emdad_out : out std_logic);
9   end component;
10
11 end Emdad_package;

```

Figure 20: component package

6. TASK 6:

In figure 21, we can see the project summary for n-bit add/sub using behavioral VHDL model.

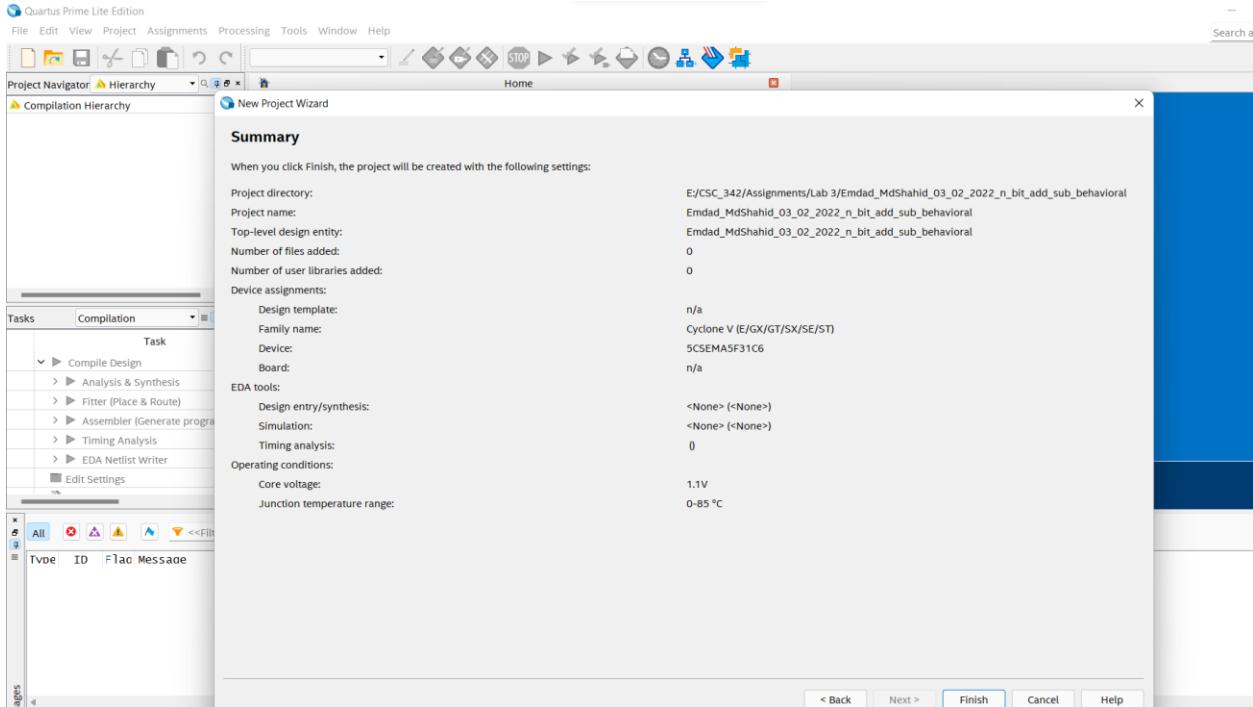


Figure 21: project summary for n-bit add/sub using behavioral VHDL model

In figure 22, we can see the n-bit add/sub using behavioral VHDL code.

```

library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

ENTITY Emdad_MdShahid_03_02_2022_n_bit_add_sub_behavioral IS
generic(n : integer := 4);
PORT(
    op      : in STD_LOGIC;
    Emdad_MdShahid_03_02_2022_x : in STD_LOGIC_VECTOR(n-1 downto 0);
    Emdad_MdShahid_03_02_2022_y : in STD_LOGIC_VECTOR(n-1 downto 0);
    Emdad_MdShahid_03_02_2022_output : out STD_LOGIC_VECTOR(n-1 downto 0);
    Emdad_MdShahid_03_02_2022_cout : out STD_LOGIC);
END Emdad_MdShahid_03_02_2022_n_bit_add_sub_behavioral;

ARCHITECTURE nbitaddsub OF Emdad_MdShahid_03_02_2022_n_bit_add_sub_behavioral IS
SIGNAL Emdad_MdShahid_03_02_2022_p : STD_LOGIC_VECTOR(n downto 0);
BEGIN
    PROCESS(Emdad_MdShahid_03_02_2022_x, Emdad_MdShahid_03_02_2022_y, Emdad_MdShahid_03_02_2022_op)
    BEGIN
        if Emdad_MdShahid_03_02_2022_op = '0' then
            Emdad_MdShahid_03_02_2022_p <= ('0' & Emdad_MdShahid_03_02_2022_x) + ('0' & Emdad_MdShahid_03_02_2022_y);
        end if;
        if Emdad_MdShahid_03_02_2022_op = '1' then
            Emdad_MdShahid_03_02_2022_p <= ('0' & Emdad_MdShahid_03_02_2022_x) - ('0' & Emdad_MdShahid_03_02_2022_y);
        end if;
    END PROCESS;
    Emdad_MdShahid_03_02_2022_output <= Emdad_MdShahid_03_02_2022_p(n-1 downto 0);
    Emdad_MdShahid_03_02_2022_cout <= Emdad_MdShahid_03_02_2022_p(n);
END nbitaddsub;

```

Figure 22: n-bit add/sub using behavioral VHDL code

In figure 23, we can see successful compilation report for the n-bit add/sub using behavioral VHDL model.

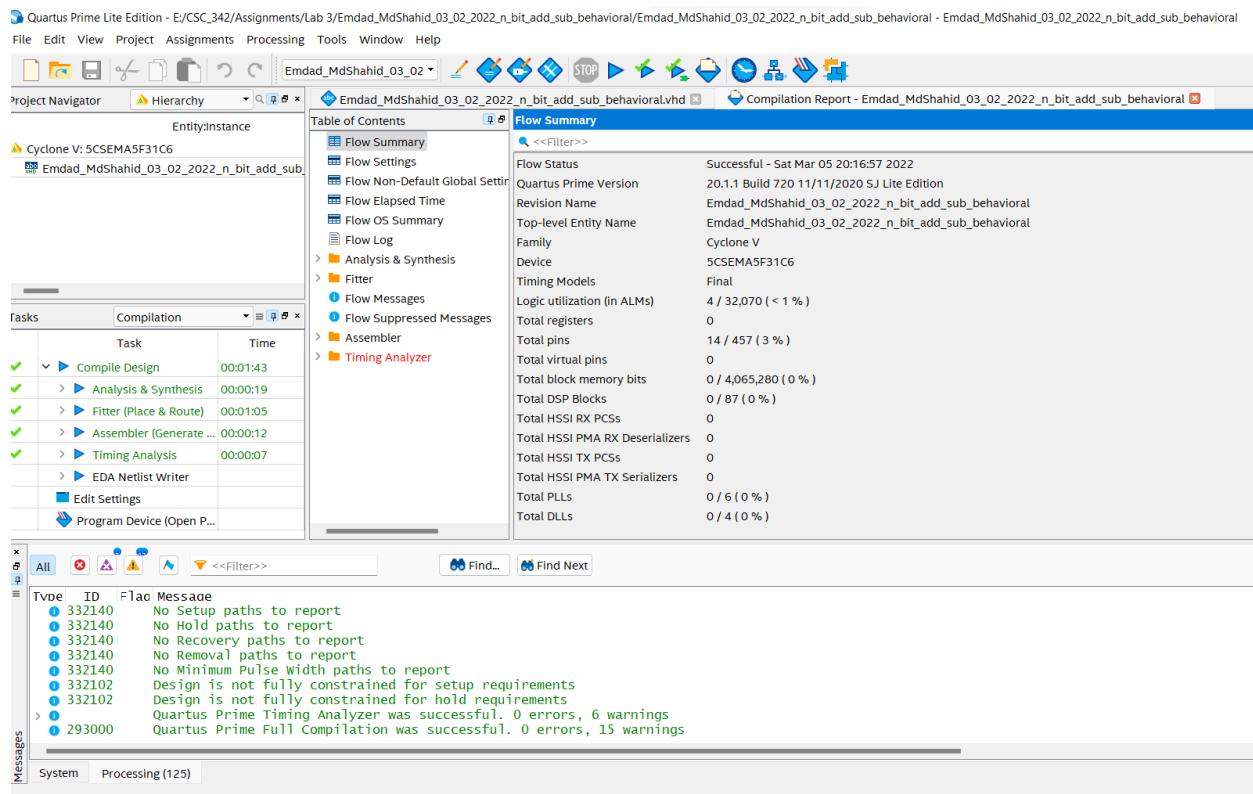


Figure 23: successful compilation report

7. LAB 7:

In figure 24, we can see the project summary for n-bit add/sub flags.

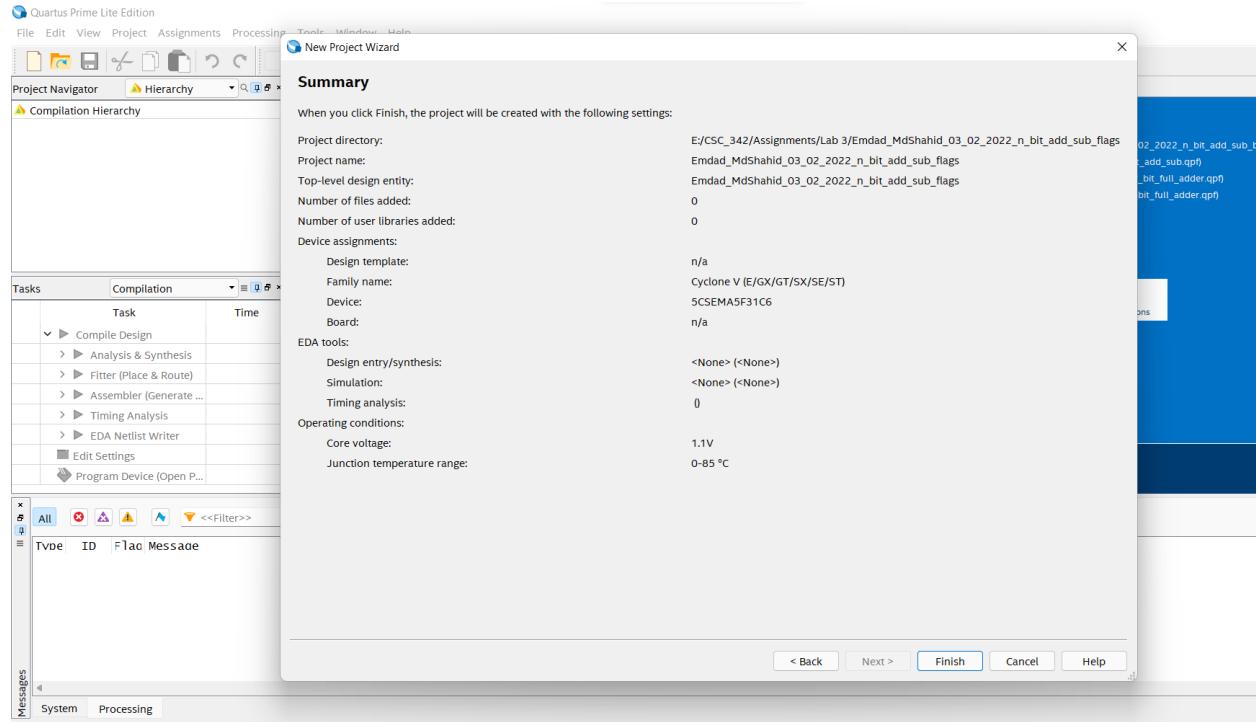


Figure 24: project summary for n-bit add/sub flags

In figure 25, we can see the VHDL code for n-bit add/sub.

```

1  LIBRARY ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6  ENTITY Emdad_MdShahid_03_02_2022_n_bit_add_sub_flags IS
7  generic (n : integer := 32);
8
9  PORT ( Emdad_MdShahid_03_02_2022_op : in STD_LOGIC;
10         Emdad_MdShahid_03_02_2022_x : in STD_LOGIC_VECTOR(n-1 downto 0);
11         Emdad_MdShahid_03_02_2022_y : in STD_LOGIC_VECTOR(n-1 downto 0);
12         Emdad_MdShahid_03_02_2022_output : out STD_LOGIC_VECTOR(n-1 downto 0);
13         Emdad_MdShahid_03_02_2022_overflow : out STD_LOGIC;
14         Emdad_MdShahid_03_02_2022_zero : out STD_LOGIC;
15         Emdad_MdShahid_03_02_2022_negative : out STD_LOGIC);
16
17 END Emdad_MdShahid_03_02_2022_n_bit_add_sub_flags;
18
19 ARCHITECTURE nbitaddsubflags OF Emdad_MdShahid_03_02_2022_n_bit_add_sub_flags IS
20
21 SIGNAL Emdad_MdShahid_03_02_2022_p : STD_LOGIC_VECTOR(n downto 0);
22
23 BEGIN
24 PROCESS (Emdad_MdShahid_03_02_2022_x, Emdad_MdShahid_03_02_2022_y, Emdad_MdShahid_03_02_2022_op)
25 BEGIN
26 IF Emdad_MdShahid_03_02_2022_op = '0' then
27     Emdad_MdShahid_03_02_2022_p <= ('0' & Emdad_MdShahid_03_02_2022_x) + ('0' & Emdad_MdShahid_03_02_2022_y);
28     Emdad_MdShahid_03_02_2022_negative <= '0';
29 end if;
30 IF Emdad_MdShahid_03_02_2022_op = '1' then
31     IF Emdad_MdShahid_03_02_2022_x < Emdad_MdShahid_03_02_2022_y then
32         Emdad_MdShahid_03_02_2022_negative <= '1';
33     ELSE
34         Emdad_MdShahid_03_02_2022_negative <= '0';
35     end if;
36     Emdad_MdShahid_03_02_2022_p <= ('0' & Emdad_MdShahid_03_02_2022_x) - ('0' & Emdad_MdShahid_03_02_2022_y);
37 end if;
38 END PROCESS;
39
40 Emdad_MdShahid_03_02_2022_output <= Emdad_MdShahid_03_02_2022_p(n-1 downto 0);
41 Emdad_MdShahid_03_02_2022_overflow <= Emdad_MdShahid_03_02_2022_p(n);
42 Emdad_MdShahid_03_02_2022_zero <= '1' WHEN Emdad_MdShahid_03_02_2022_p(n-1 downto 0) = 0 ELSE '0';
43
44 END nbitaddsubflags;

```

Figure 25: VHDL code for n-bit add/sub.

In figure 26, we can see the successful compilation report.

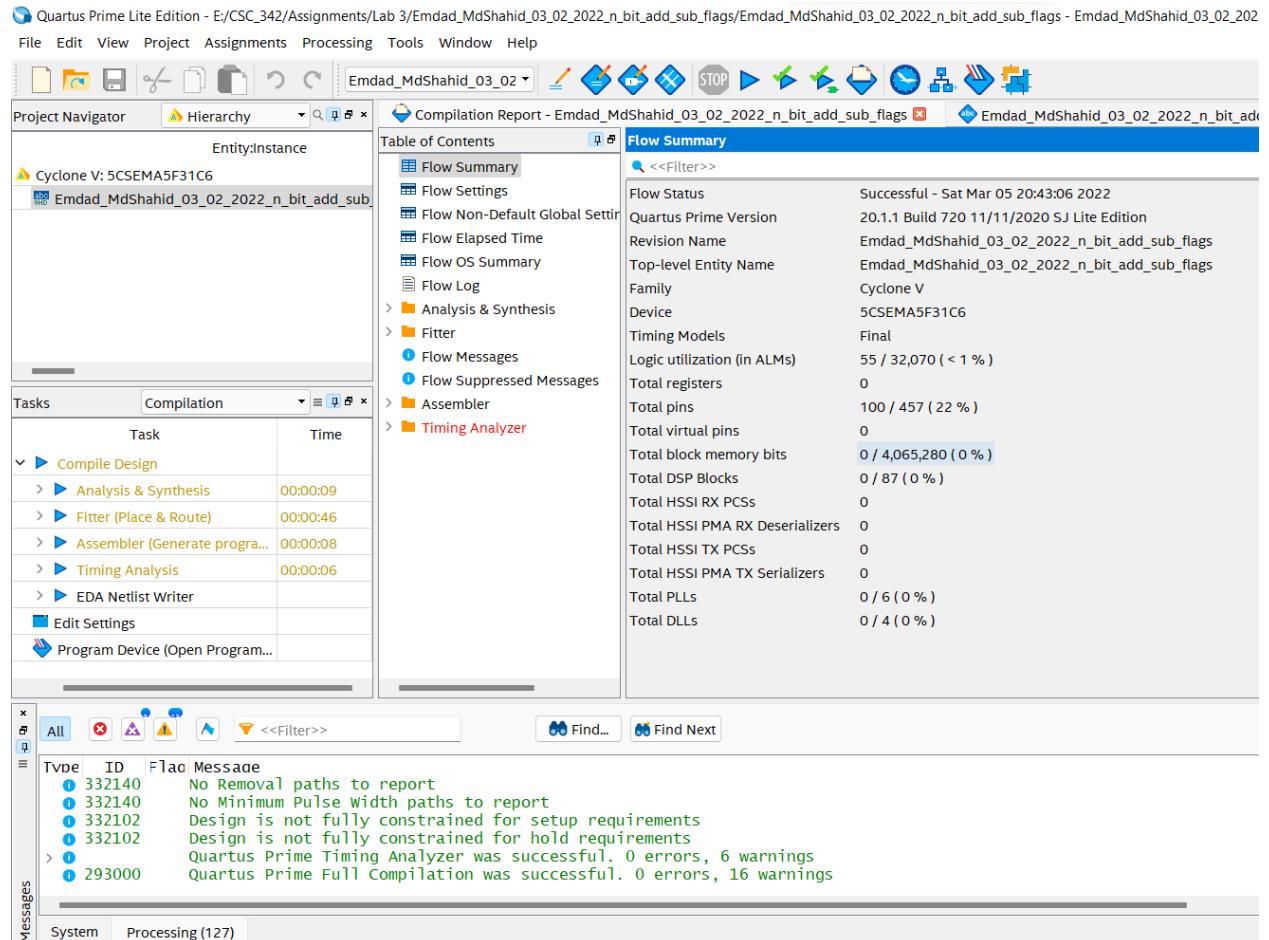


Figure 26: successful compilation report.

8. TASK 8:

Task 8 was divided into 2 parts. First, we worked on the 4-bit design in Modelsim and to verify that I created a testbench to get a waveform out which was provided in the simulation section.

In figure 27, we can see the project summary for N=4 and N =32-bit testbench compilation.

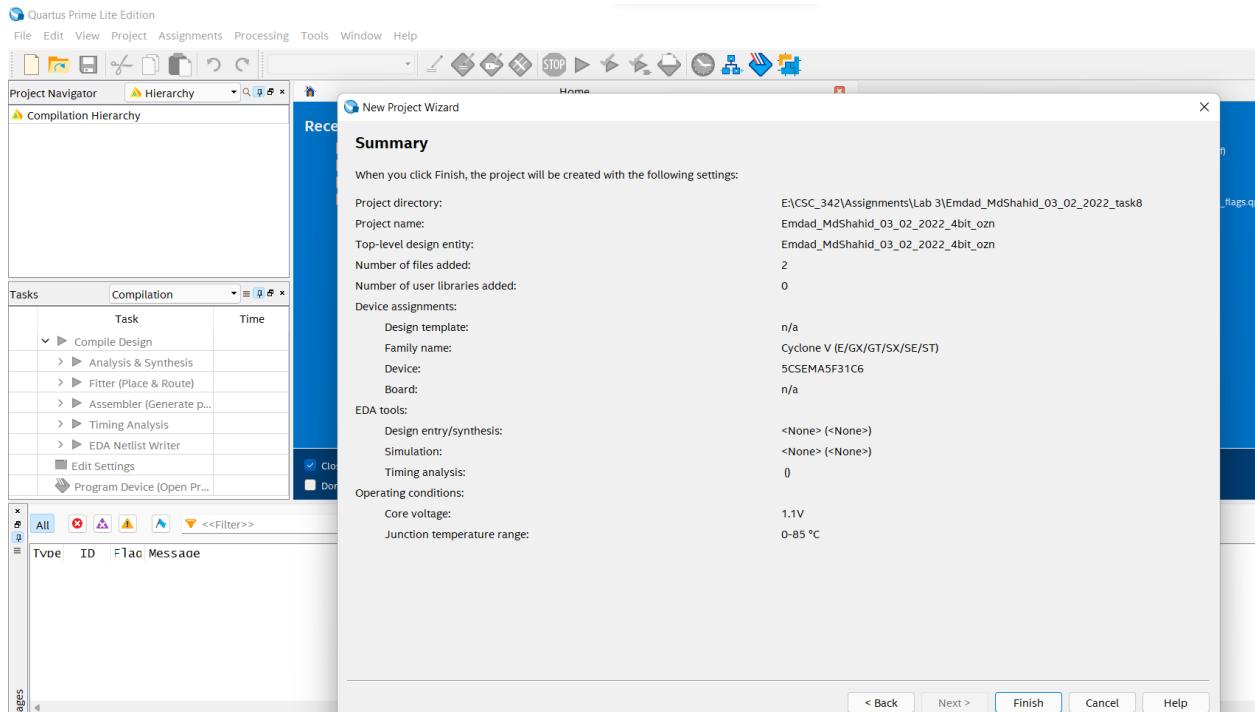


Figure 27: task 8 project summary

In figure 28, we can see the code written for 4-bit. I also added the flags.

```

library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity Emdad_MdShahid_03_02_2022_4bit_ozn is
generic (n : integer := 4);
port (x: STD_LOGIC_VECTOR(n-1 downto 0);
      y: STD_LOGIC_VECTOR(n-1 downto 0);
      sum: STD_LOGIC_VECTOR(n-1 downto 0);
      cbit: STD_LOGIC;
      carry: STD_LOGIC);
end Emdad_MdShahid_03_02_2022_4bit_ozn;
architecture nbittaddsub of Emdad_MdShahid_03_02_2022_4bit_ozn is
begin
  sum <= x xor y;
  cbit <= (x and y) or ((not x) and (not y));
  carry <= (x and y) or ((not x) and y) or (x and (not y));
end nbittaddsub;
  
```

Figure 28: 4bit add/sub with flags

In figure 29, we can see the testbench for 4-bit add/sub flags waveform check.

Quartus Prime Lite Edition - E/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_task8/Emdad_MdShahid_03_02_2022_4bit_ozn - Emdad_MdShahid_03_02_2022_4bit_ozn

```

1 Library ieee;
2 use ieee.std_logic_1164.all;
3
4 Entity Emdad_MdShahid_03_02_2022_4bit_ozn_tb is
5 End Emdad_MdShahid_03_02_2022_4bit_ozn_tb;
6
7 Architecture signalized_4_bit_test of Emdad_MdShahid_03_02_2022_4bit_ozn_tb is
8 Component Emdad_MdShahid_03_02_2022_n_bit_add_sub
9 Generic (n: integer := 4);
10 Port (
11     Emdad_MdShahid_03_02_2022_op: in std_logic;
12     Emdad_MdShahid_03_02_2022_a: in std_logic_vector(n-1 downto 0);
13     Emdad_MdShahid_03_02_2022_b: in std_logic_vector(n-1 downto 0);
14     Emdad_MdShahid_03_02_2022_overflow: out std_logic_vector(n-1 downto 0);
15     Emdad_MdShahid_03_02_2022_zero, Emdad_MdShahid_03_02_2022_zero;
16     Emdad_MdShahid_03_02_2022_negative: out std_logic);
17 End Component;
18 Signal p0: std_logic;
19 Signal p1, p2: std_logic_vector(4-1 downto 0);
20 Signal psum: std_logic_vector(4-1 downto 0);
21 Signal pzero: std_logic;
22 Signal pneg: std_logic;
23 Signal error: std_logic := '0';
24 Begin
25     uut: Emdad_MdShahid_03_02_2022_n_bit_add_sub port map(
26         Emdad_MdShahid_03_02_2022_op => p0, Emdad_MdShahid_03_02_2022_a => p1,
27         Emdad_MdShahid_03_02_2022_b => p2, Emdad_MdShahid_03_02_2022_output => psum,
28         Emdad_MdShahid_03_02_2022_overflow => pover, Emdad_MdShahid_03_02_2022_zero => pzero,
29         Emdad_MdShahid_03_02_2022_negative => pneg);
30     Process
31     begin
32         --most positive plus 1
33         p0 <= '1';
34         p1 <= "111";
35         p2 <= "000";
36         wait for 1 ns;
37         if (pover = '0' or pzero = '0' or pneg = '1') then
38             error <= '1';
39         end if;
40         wait for 200 ns;

```

Figure 29: 4-bit testbench (Part 1)

Quartus Prime Lite Edition - E/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_task8/Emdad_MdShahid_03_02_2022_4bit_ozn - Emdad_MdShahid_03_02_2022_4bit_ozn

```

41     --most positive minus 1
42     p0 <= '1';
43     p1 <= "111";
44     p2 <= "001";
45     wait for 1 ns;
46     if (pover = '1' or pzero = '1' or pneg = '1') then
47         error <= '1';
48     end if;
49     wait for 200 ns;
50     --most negative plus 1
51     p0 <= '0';
52     p1 <= "100";
53     p2 <= "001";
54     wait for 1 ns;
55     if (pover = '1' or pzero = '1' or pneg = '1') then
56         error <= '1';
57     end if;
58     wait for 200 ns;
59     --most negative minus 1
60     p0 <= '1';
61     p1 <= "111";
62     p2 <= "001";
63     wait for 1 ns;
64     if (pover = '1' or pzero = '1' or pneg = '1') then
65         error <= '1';
66     end if;
67     wait for 200 ns;
68     --most positive minus most negative
69     p0 <= '1';
70     p1 <= "111";
71     p2 <= "100";
72     wait for 1 ns;
73     if (pover = '1' or pzero = '1' or pneg = '1') then
74         error <= '1';
75     end if;
76     wait for 200 ns;
77     --most positive plus most negative
78     p0 <= '0';
79     p1 <= "111";

```

Figure 30: 4-bit testbench (Part 2)

Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_task8/Emdad_MdShahid_03_02_2022_4bit_ozn - Emdad_MdShahid_03_02_2022_4bit_ozn

File Edit View Project Assignments Processing Tools Window Help

Project Navigator Files Emdad_MdShahid_03_02_2022_4bit_ozn Emdad_MdShahid_4bit_ozn_tb.vhd

Compilation Report - Emdad_MdShahid_03_02_2022_4bit_ozn Emdad_MdShahid_4bit_ozn_tb.vhd

```

68      --most positive minus most negative
69      p0 <= '1';
70      p1 <= "1111";
71      p2 <= "1000";
72      wait for 1 ns;
73      if (pover = '1' or pzero = '1' or pneg = '1') then
74          error <= '1';
75      end if;
76      wait for 200 ns;
77      --most positive plus most negative
78      p0 <= '0';
79      p1 <= "1111";
80      p2 <= "1000";
81      wait for 1 ns;
82      if (pover = '0' or pzero = '1' or pneg = '1') then
83          error <= '1';
84      end if;
85      wait for 200 ns;
86      --most positive minus most positive
87      p0 <= '1';
88      p1 <= "1111";
89      p2 <= "1111";
90      wait for 1 ns;
91      if (pover = '1' or pzero = '0' or pneg = '1') then
92          error <= '1';
93      end if;
94      wait for 200 ns;
95
96      if (error = '0') then
97          report "No errors detected. Simulation successful" severity
98          failure;
99      else
100         report "Error detected" severity failure;
101     end if;
102   end process;
103 end signalized_4_bit_test;
104
105
106
107

```

All Find... Find Next

Type TD Flag Message

Figure 31: 4-bit testbench (Part 3)

In figure 32, we can see the project summary for 32 bits.

Quartus Prime Lite Edition

New Project Wizard

Summary

When you click Finish, the project will be created with the following settings:

Project directory:	E:\CSC_342\Assignments\Lab 3\Emdad_03_02_2022_32_bit
Project name:	Emdad_03_02_2022_32_bit
Top-level design entity:	Emdad_03_02_2022_32_bit
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Design template:	n/a
Family name:	Cyclone V (E/GX/GT/SX/SE/ST)
Device:	5CSEMA6F31C6
Board:	n/a
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	<None> (<None>)
Timing analysis:	0
Operating conditions:	
Core voltage:	1.1V
Junction temperature range:	0-85 °C

Tasks Compilation

Task

- ✓ ▶ Compile Design 00:01:45
- ✓ ▶ Analysis & Synthesis 00:00:09
- ✓ ▶ Fitter (Place & Route) 00:01:23
- ✓ ▶ Assembler (Generate programming files) 00:00:08
- ✓ ▶ Timing Analysis 00:00:05
- ▶ EDA Netlist Writer
- Edit Settings
- Program Device (Open Programmer)

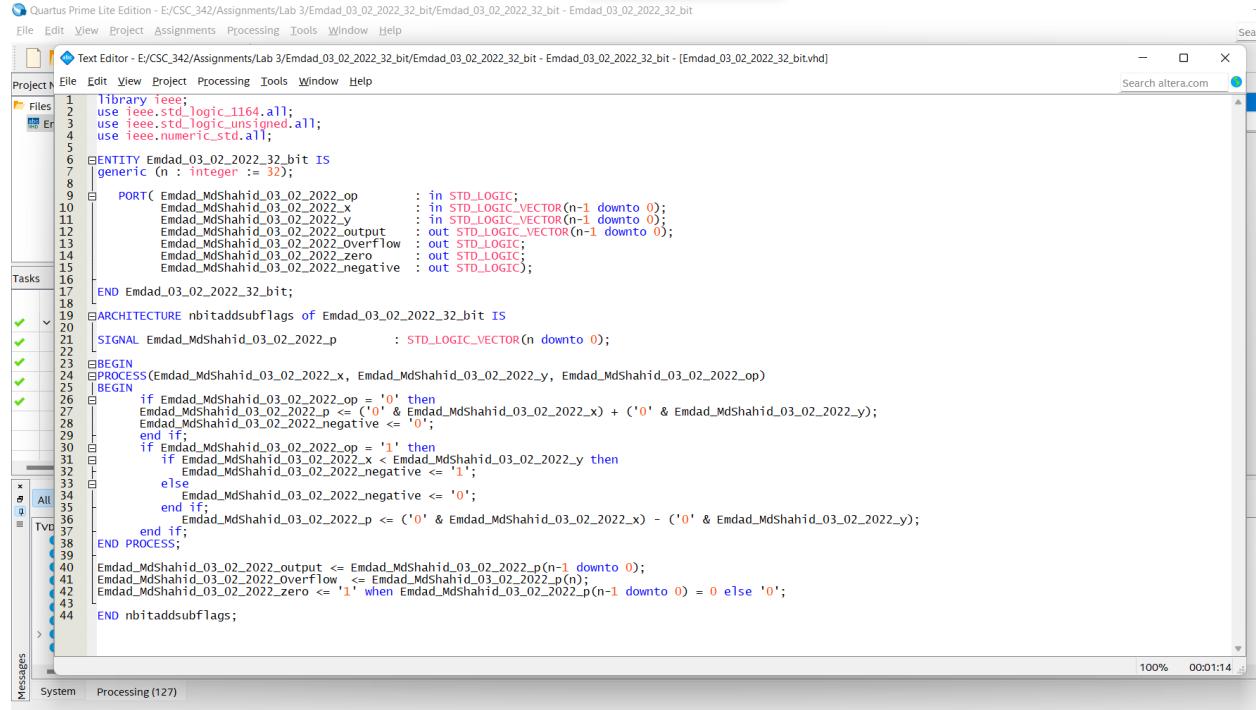
All Find... Find Next

Type TD Flag Message

Messages System Processing

Figure 32: 32 bits project summary

In figure 33, we can see the VHDL code for the 32-bits with flags.



The screenshot shows the Quartus Prime Lite Text Editor interface with the following details:

- Title Bar:** Text Editor - E:/CSC_342/Assignments/Lab 3/Emdad_03_02_2022_32_bit/Emdad_03_02_2022_32_bit - Emdad_03_02_2022_32_bit.vhd
- Menu Bar:** File Edit View Project Assignments Processing Tools Window Help
- Search Bar:** Search altera.com
- Project Tree:** Shows a single file named "Emdad_03_02_2022_32_bit.vhd".
- Code Area:**

```

1 library IEEE;
2 use IEEE.STD_logic_1164.all;
3 use IEEE.STD_logic_unsigned.all;
4 use IEEE.numeric_std.all;
5
6 ENTITY Emdad_03_02_2022_32_bit IS
7   generic (n : integer := 32);
8
9   PORT (Emdad_MdShahid_03_02_2022_op : in STD_LOGIC;
10        Emdad_MdShahid_03_02_2022_x : in STD_LOGIC_VECTOR(n-1 downto 0);
11        Emdad_MdShahid_03_02_2022_y : in STD_LOGIC_VECTOR(n-1 downto 0);
12        Emdad_MdShahid_03_02_2022_output : out STD_LOGIC_VECTOR(n-1 downto 0);
13        Emdad_MdShahid_03_02_2022_overflow : out STD_LOGIC;
14        Emdad_MdShahid_03_02_2022_zero : out STD_LOGIC;
15        Emdad_MdShahid_03_02_2022_negative : out STD_LOGIC);
16
17 END Emdad_03_02_2022_32_bit;
18
19 ARCHITECTURE nbitaddsubflags OF Emdad_03_02_2022_32_bit IS
20
21   SIGNAL Emdad_MdShahid_03_02_2022_p : STD_LOGIC_VECTOR(n downto 0);
22
23   BEGIN
24     PROCESS(Emdad_MdShahid_03_02_2022_x, Emdad_MdShahid_03_02_2022_y, Emdad_MdShahid_03_02_2022_op)
25     BEGIN
26       if Emdad_MdShahid_03_02_2022_op = '0' then
27         Emdad_MdShahid_03_02_2022_p <= ('0' & Emdad_MdShahid_03_02_2022_x) + ('0' & Emdad_MdShahid_03_02_2022_y);
28         Emdad_MdShahid_03_02_2022_negative <= '0';
29       end if;
30       if Emdad_MdShahid_03_02_2022_op = '1' then
31         if Emdad_MdShahid_03_02_2022_x < Emdad_MdShahid_03_02_2022_y then
32           Emdad_MdShahid_03_02_2022_negative <= '1';
33         else
34           Emdad_MdShahid_03_02_2022_negative <= '0';
35         end if;
36         Emdad_MdShahid_03_02_2022_p <= ('0' & Emdad_MdShahid_03_02_2022_x) - ('0' & Emdad_MdShahid_03_02_2022_y);
37       end if;
38     END PROCESS;
39
40   Emdad_MdShahid_03_02_2022_output <= Emdad_MdShahid_03_02_2022_p(n-1 downto 0);
41   Emdad_MdShahid_03_02_2022_Overflow <= Emdad_MdShahid_03_02_2022_p(n);
42   Emdad_MdShahid_03_02_2022_zero <= '1' when Emdad_MdShahid_03_02_2022_p(n-1 downto 0) = 0 else '0';
43
44 END nbitaddsubflags;

```
- Task List:** Shows several tasks marked with green checkmarks.
- Messages:** Shows "All" messages.
- Status Bar:** 100% 00:01:14

Figure 33: VHDL code for the 32-bits with flags.

In figure 34, we can see it compiled successfully.

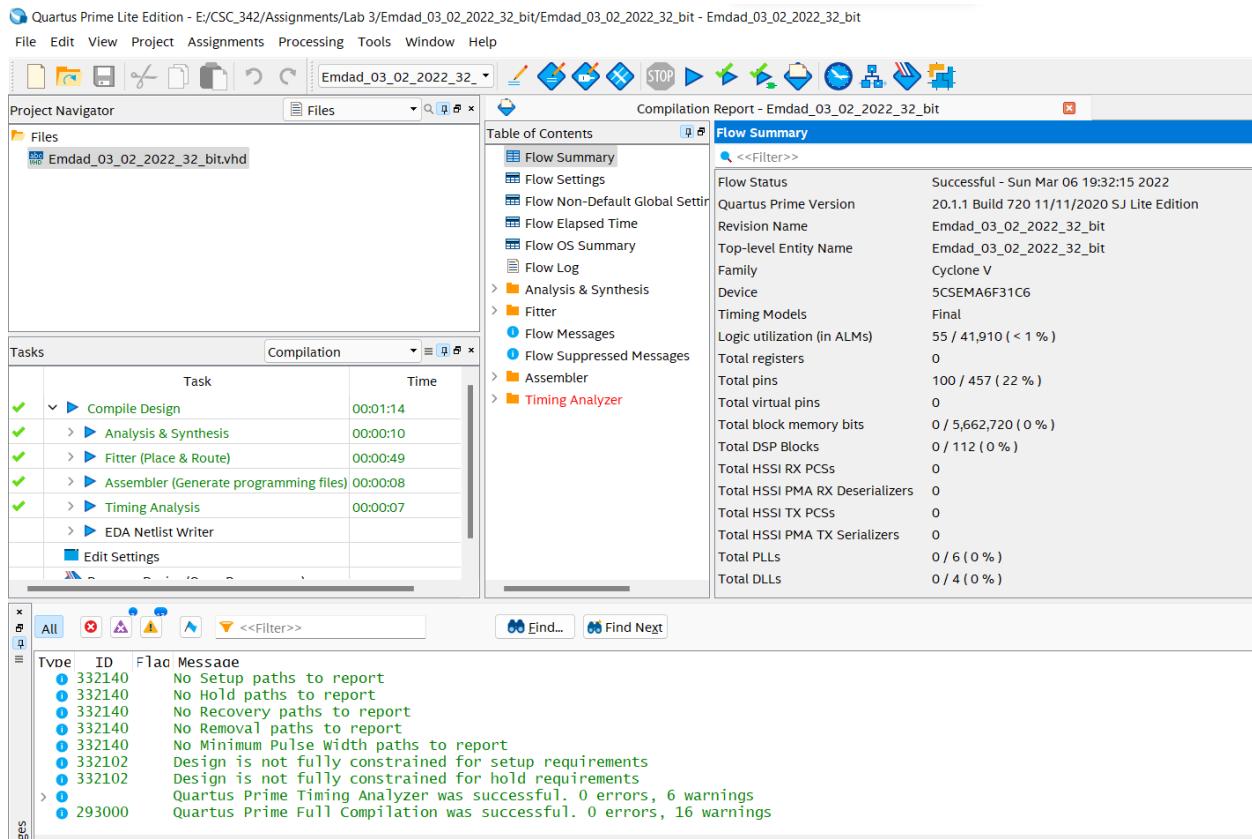


Figure 34: successful compilation report

9. TASK 9:

In figure 35, we can see n-bit LMP add/sub project summary.

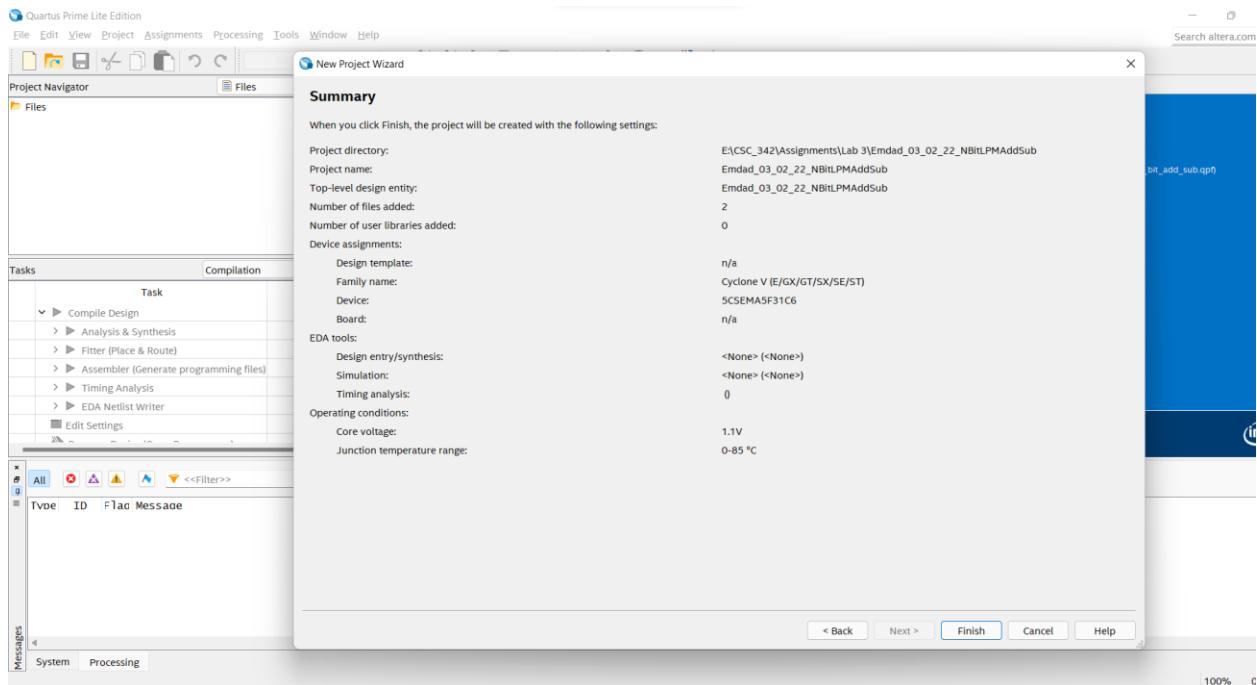


Figure 35:n-bit LMP add/sub

In figure 36, we can see the VHDL code.

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3 LIBRARY lpm;
4 USE lpm.lpm_components.all;
5
6 ENTITY Emdad_03_02_22_NBitLPMAddSub IS
7 PORT ( add_sub : IN STD_LOGIC ;
8 cin : IN STD_LOGIC ;
9 dataa : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
10 datab : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
11 cout : OUT STD_LOGIC ;
12 overflow : OUT STD_LOGIC ;
13 result : OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
14 END Emdad_03_02_22_NBitLPMAddSub;
15
16 ARCHITECTURE SYN OF Emdad_03_02_22_NBitLPMAddSub IS
17
18 SIGNAL sub_wire0 : STD_LOGIC ;
19 SIGNAL sub_wire1 : STD_LOGIC ;
20 SIGNAL sub_wire2 : STD_LOGIC_VECTOR (15 DOWNTO 0);
21
22 COMPONENT lpm_add_sub
23 GENERIC (
24 lpm_direction : STRING;
25 lpm_hint : STRING;
26 lpm_representation : STRING;
27 lpm_type : STRING;
28 lpm_width : NATURAL);
29
30 PORT ( add_sub : IN STD_LOGIC ;
31 cin : IN STD_LOGIC ;
32

```

Figure 36: VHDL code for n-bit lmp add/sub(Part 1)

In figure 37, we can see the rest of the code for n-bit lmp add/sub.

Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab 3/Emdad_03_02_22_NBitLPMAddSub/Emdad_03_02_22_NBitLPMAddSub - Emdad_03_02_22_NBitLPMAddSub

File Edit View Project Assignments Processing Tools Window Help

Project Navigator Files Compilation Report - Emdad_03_02_22_NBitLPMAddSub Emdad_03_02_22_NBitLPMAddSub.vhd

```

33      dataaa : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
34      datab : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
35      cout : OUT STD_LOGIC;
36      overflow : OUT STD_LOGIC;
37      result : OUT STD_LOGIC_VECTOR (15 DOWNTO 0);
38  END COMPONENT;
39
40 BEGIN
41     cout <= sub_wire0;
42     overflow <= sub_wire1;
43     result <= sub_wire2(15 DOWNTO 0);
44
45     lpm_add_sub_component : lpm_add_sub
46         GENERIC MAP (
47             lpm_direction => "UNUSED",
48             lpm_hint => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
49             lpm_representation => "UNSIGNED",
50             lpm_type => "LPM_ADD_SUB",
51             lpm_width => 16)
52     PORT MAP (
53         add_sub => add_sub,
54         cin => cin,
55         dataaa => dataaa,
56         datab => datab,
57         cout => sub_wire0,
58         overflow => sub_wire1,
59         result => sub_wire2 );
60
61     END SYN;
62
63

```

Tasks

Task	Time
Compile Design	00:01:01
Analysis & Synthesis	00:00:09
Fitter (Place & Route)	00:00:39
Assembler (Generate programming files)	00:00:08
Timing Analysis	00:00:05
EDA Netlist Writer	
Edit Settings	
Program Device (Open Programmer)	

Find... Find Next

Type ID Flag Message

Type	ID	Flag	Message
332140	332140		No Setup paths to report
332140	332140		No Hold paths to report
332140	332140		No Recovery paths to report
332140	332140		No Removal paths to report
332140	332140		No Minimum Pulse Width paths to report
332102	332102		Design is not fully constrained for setup requirements
332102	332102		Design is not fully constrained for hold requirements
293000	293000		Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
293000	293000		Quartus Prime Full Compilation was successful. 0 errors, 14 warnings

Figure 37: VHDL code for n-bit lmp add/sub (Part 2)

In figure 38, we can see the testbench for the n-bitlmpadd/sub to verify our waveform.

Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab 3/Emdad_03_02_22_NBitLPMAddSub/Emdad_03_02_22_NBitLPMAddSub - Emdad_03_02_22_NBitLPMAddSub

File Edit View Project Assignments Processing Tools Window Help

Project Navigator Files Compilation Report - Emdad_03_02_22_NBitLPMAddSub Emdad_03_02_22_NBitLPMAddSub.vhd Emdad_MdShahid_NBitLPMAddSub_TestBench.vhd

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity Emdad_03_06_22_NBitLPMAddSubTestBench is
7 end Emdad_03_06_22_NBitLPMAddSubTestBench;
8
9 architecture arch_test of Emdad_03_06_22_NBitLPMAddSubTestBench is
10 component Emdad_03_02_22_NBitLPMAddSub is
11     generic (N: integer := 16);
12     PORT (add_sub : 
13         cin : IN STD_LOGIC;
14         datab : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
15         datab : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
16         cout : OUT STD_LOGIC;
17         overflow : OUT STD_LOGIC;
18         result : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0));
19     END component;
20     signal s_add_sub, s_cin, s_cout, s_overflow: STD_LOGIC;
21     signal s_dataaa, s_datab, s_result: STD_LOGIC_VECTOR(16-1 DOWNTO 0);
22
23 begin
24     U1: Emdad_03_06_22_NBitLPMAddSub PORT MAP(s_add_sub, s_cin, s_dataaa, s_datab, s_cout, s_overflow, s_result);
25     vectors: PROCESS
26     BEGIN
27         s_cin <='0';
28         -- a. Most Positive N bit integer + 1
29         s_add_sub <='0';
30         s_dataaa <="0000000000000001";
31         s_datab <="0000000000000001";
32         WAIT FOR 100 ns;

```

Tasks

Task	Time
Compile Design	00:01:01
Analysis & Synthesis	00:00:09
Fitter (Place & Route)	00:00:39
Assembler (Generate programming files)	00:00:08
Timing Analysis	00:00:05
EDA Netlist Writer	
Edit Settings	
Program Device (Open Programmer)	

Find... Find Next

Type ID Flag Message

Type	ID	Flag	Message
332140	332140		No Setup paths to report
332140	332140		No Hold paths to report
332140	332140		No Recovery paths to report
332140	332140		No Removal paths to report
332140	332140		No Minimum Pulse Width paths to report
332102	332102		Design is not fully constrained for setup requirements
332102	332102		Design is not fully constrained for hold requirements
293000	293000		Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
293000	293000		Quartus Prime Full Compilation was successful. 0 errors, 14 warnings

Figure 38: nbitlmpaddsub testbench (Part 1)

The screenshot shows the Quartus Prime Lite Edition interface. The top menu bar includes File, Edit, View, Project, Assignments, Processing, Tools, Window, and Help. The Project Navigator shows files like Emdad_03_02_22_NBitLPMAddSub.vhd and Emdad_MdShahid_NBitLPMAddSub_Emdad_03_02_22_NBitLPMAddSub_TestBench.vhd. The Compilation Report displays VHDL code for a testbench. The Tasks table lists build steps with their times: Compile Design (00:01:01), Analysis & Synthesis (00:00:09), Fitter (Place & Route) (00:00:39), Assembler (Generate programming files) (00:00:08), Timing Analysis (00:00:05), and others. The Messages table shows two warnings related to setup paths.

```

28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
-- a. Most Positive N bit integer + 1
s_add_sub <= "0";
s_dataa <= "0111111111111111";
s_datab <= "0000000000000001";
WAIT FOR 100 ns;
-- b. Most Positive N bit integer - 1
s_add_sub <= "1";
s_dataa <= "0111111111111111";
s_datab <= "0000000000000001";
WAIT FOR 100 ns;
-- c. Most Negative N bit integer + 1
s_add_sub <= "0";
s_dataa <= "1000000000000000";
s_datab <= "0000000000000001";
WAIT FOR 100 ns;
-- d. Most Negative N bit integer ? 1
s_add_sub <= "1";
s_dataa <= "1000000000000000";
s_datab <= "0000000000000001";
WAIT FOR 100 ns;
-- e. Most Positive N bit integer - Most Negative N bit integer
s_add_sub <= "1";
s_dataa <= "0111111111111111";
s_datab <= "1000000000000000";
WAIT FOR 100 ns;
-- f. Most Positive N bit integer + Most Negative N bit integer
s_add_sub <= "0";
s_dataa <= "0111111111111111";
s_datab <= "1000000000000000";
WAIT FOR 100 ns;
-- g. Most Positive N bit integer - Most Positive N bit integer
s_add_sub <= "1";
s_dataa <= "0111111111111111";
s_datab <= "0111111111111111";
WAIT FOR 100 ns;
WAIT;
END PROCESS;
end arch_test;

```

Figure 39: nbitlmpaddsub testbench (Part 2)

10. TASK 10:

In figure 40, we can see the project summary for 16 bit.

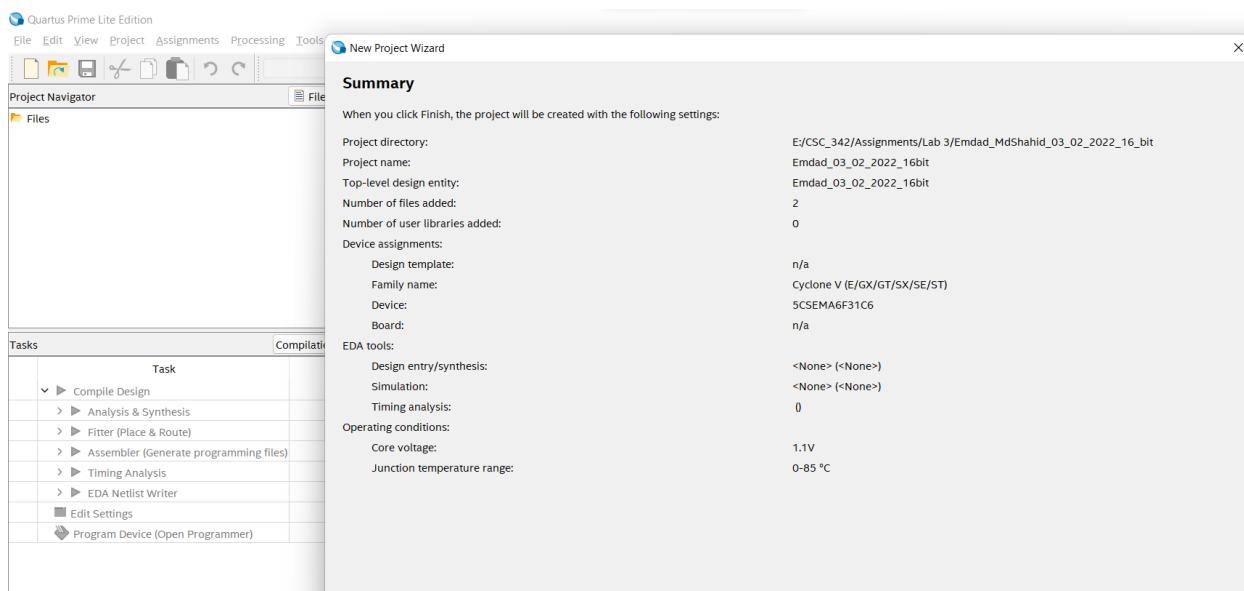


Figure 40: project summary for 16bits

In figure 41, we can see the 16 bits VHDL code.

```

Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_16bit/Emdad_03_02_2022_16bit - Emdad_03_02_2022_16bit

File Edit View Project Assignments Processing Tools Window Help

Project Navigator Files Emdad_03_02_2022_16bit.vhd Emdad_03_02_2022_16bit_tb.vhd Compilation Report - Emdad_03_02_2022_16bit

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 entity Emdad_03_02_2022_16bit is
7 port(A,B:in std_logic_vector(15 downto 0); --two 16 bit inputs
8      X:out std_logic_vector(15 downto 0); --output
9      C:out std_logic; --carry output
10     S:in std_logic); --Select Bit to define Addition or subtraction
11 end Emdad_03_02_2022_16bit;
12
13 architecture Behavioral of Emdad_03_02_2022_16bit is
14 signal Y:std_logic_vector(16 downto 0); -- Define internal signal Y
15 begin
16 process (A,B,S) --Process with process variables A, B, and S
17 begin
18
19 case S is
20 when '0' => --S=0 defines the subtraction operation
21 Y<=(not A)-(not B);
22 when others => --S=1 defines the Addition operation
23 Y<=(A)+(B);
24 end case;
25 end process;
26 C<=Y(16); -- Assignment of the 17th bit as carry out
27 X<=Y(15 downto 0); -- Assignment of the 1 to 16th bit as 16-bit sum or difference
28
29 end Behavioral;

```

Tasks

Task	Time
Compile Design	00:01:14
> Analysis & Synthesis	00:00:09
> Fitter (Place & Route)	00:00:50
> Assembler (Generate programming files)	00:00:09
> Timing Analysis	00:00:06
> EDA Netlist Writer	
Edit Settings	
Program Device (Open Programmer)	

Messages

Type	ID	Flag	Message
332140		No Hold paths to report	
332140		No Recovery paths to report	
332140		No Removal paths to report	
332140		No Minimum Pulse Width paths to report	
332102		Design is not fully constrained for setup requirements	
332102		Design is not fully constrained for hold requirements	
293000		Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings	
293000		Quartus Prime Full Compilation was successful. 0 errors, 14 warnings	

Figure 41: 16-bit VHDL code

In figure 42, we can see the testbench for the 16bit VHDL code to get the waveform.

Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_16_bit/Emdad_03_02_2022_16bit

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3
4 ENTITY Emdad_03_02_2022_16bit_tb IS
5 END Emdad_03_02_2022_16bit_tb;
6
7 ARCHITECTURE behavior OF Emdad_03_02_2022_16bit_tb IS
8
9
10 COMPONENT Emdad_03_02_2022_16bit
11 PORT(
12   A : IN std_logic_vector(15 downto 0); --16 bit input A
13   B : IN std_logic_vector(15 downto 0); --16 bit input B
14   X : OUT std_logic_vector(15 downto 0); --Output
15   C : OUT std_logic; --carry out
16   S : IN std_logic --Select Bit
17 );
18 END COMPONENT;
19
20
21 --Define Inputs as internal signal
22 signal A : std_logic_vector(15 downto 0) := (others => '0');
23 signal B : std_logic_vector(15 downto 0) := (others => '0');
24 signal S : std_logic := '0';
25
26 --Define Outputs as internal signal
27 signal X : std_logic_vector(15 downto 0);
28 signal C : std_logic;
29
30 BEGIN
31
32 Buut: Emdad_03_02_2022_16bit PORT MAP (
33   |A => A, -- Definition of the Unit Under Test
34

```

Figure 42: testbench code for 16bit Part 1

Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab 3/Emdad_MdShahid_03_02_2022_16_bit/Emdad_03_02_2022_16bit

```

32 Buut: Emdad_03_02_2022_16bit PORT MAP (
33   |A => A, -- Definition of the Unit Under Test
34   B => B,
35   X => X,
36   C => C,
37   S => S
38 );
39
40 Estim_proc: process
41 begin
42
43   wait for 100 ns;
44
45   -- Define Test inputs:
46   S<='1';
47   A<="000000000100000";
48   B<="111111111111111";
49   wait for 10 ns; --Adding delay of 10 nanoseconds for convinience
50   S<='0';
51
52   --Add several test inputs
53   A<="0000000000100000";
54   B<="011111111111111";
55   wait for 10 ns;
56   S<='0';
57   A<="111111111111111";
58   B<="0000000000100000";
59
60   -- Here including subtraction operation
61   A<="0000000000100000";
62   B<="000011111111111";
63   wait;
64 end process;
65
66

```

Type	ID	Flag	Message
1	332140	No Hold paths to report	
1	332140	No Recovery paths to report	
1	332140	No Removal paths to report	
1	332140	No Minimum Pulse Width paths to report	
1	332102	Design is not fully constrained for setup requirements	
1	332102	Design is not fully constrained for hold requirements	

Figure 43: testbench code for 16bit Part 2

Simulation:

4-bit simulation:

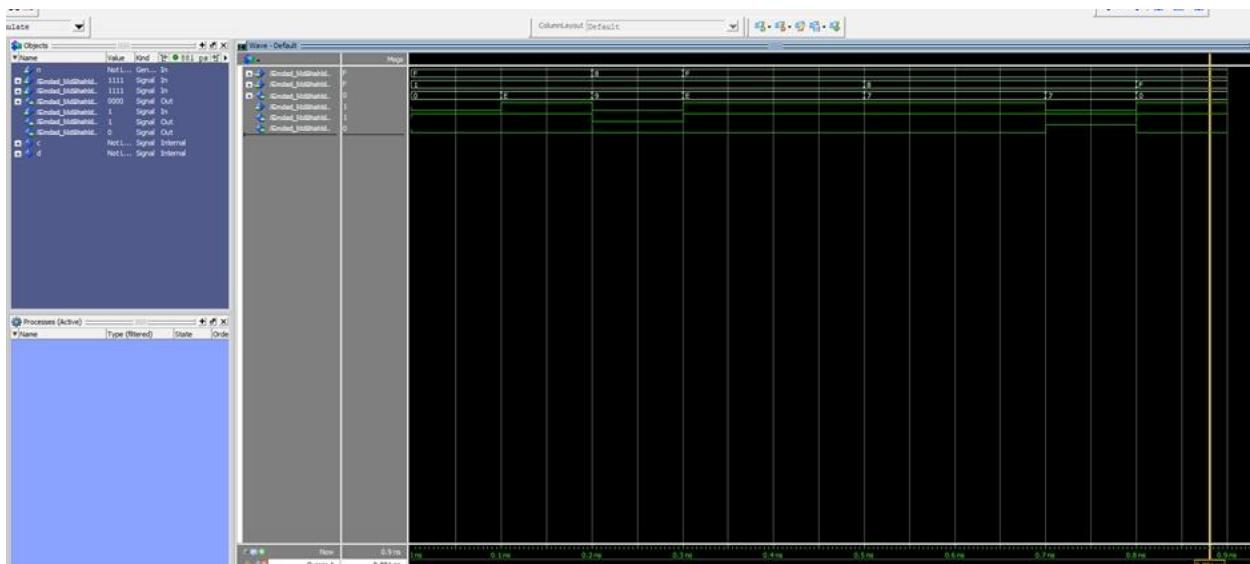


Figure 44: 4-bit simulation

Most negative 4-bit integer: -7 (binary: 0000)

- a. Most Positive N bit integer + 1: obit: 0, a = 1111, b = 0001



Figure 45: case a

- b. Most Positive N bit integer - 1: obit = 1, a = 1111, b = 0001

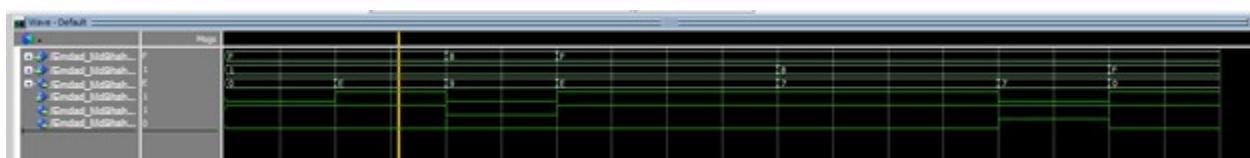


Figure 46: case b

- c. Most Negative N bit integer + 1: Obit = 0, a = 1000, b = 0001

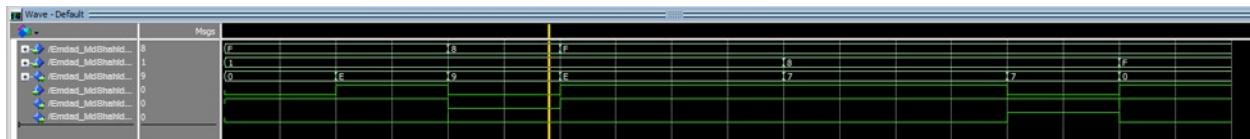


Figure 47: case c

d. Most Negative N bit integer – 1: Obit = 1, a = 1111, b = 0001

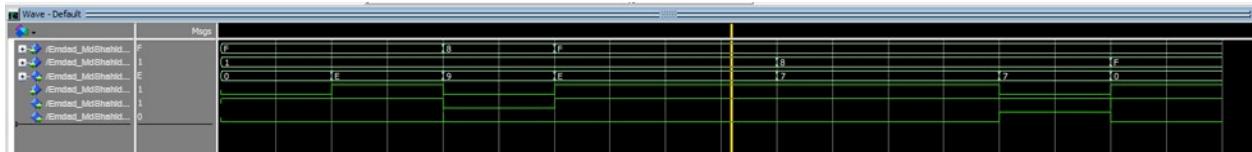


Figure 48: case d

e. Most Positive N bit integer-Most Negative N bit integer: Obit = 1, a = 1111, b = 1000

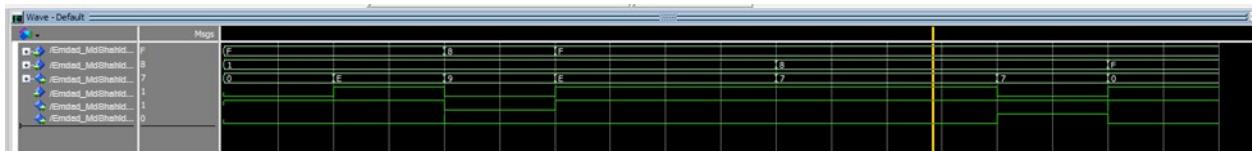


Figure 49: case e

f. Most Positive N bit integer+ Most Negative N bit integer: Obit = 0, a = 1111, b = 1000

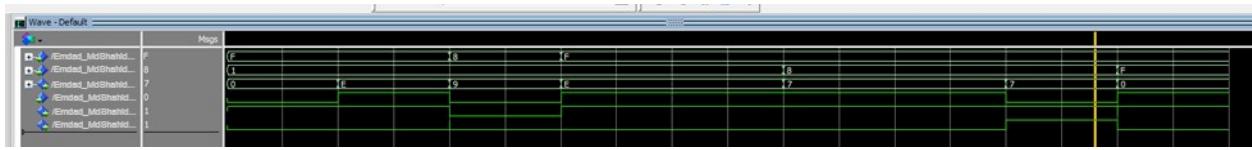


Figure 50: case f

g. Most Positive N bit integer-Most Positive N bit integer: Obit = 1, a = 1111, b = 1111

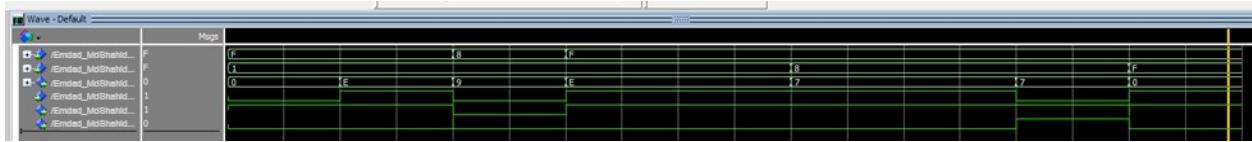


Figure 51: case g

32-bit simulation:

Most 32-bit positive integer: 2,147,483,647 (binary: 011111111111111111111111)

Most 32-bit negative integer: -2,147,483,648 (binary: 10000000000000000000000000000000)

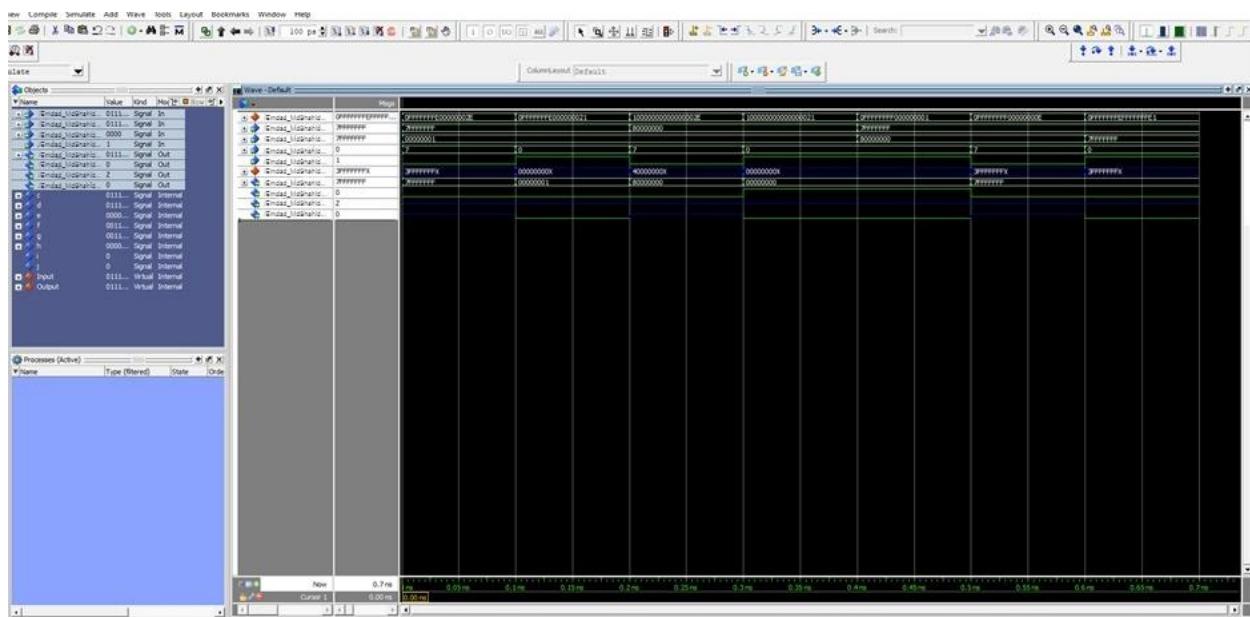


Figure 52: 32-bit simulation

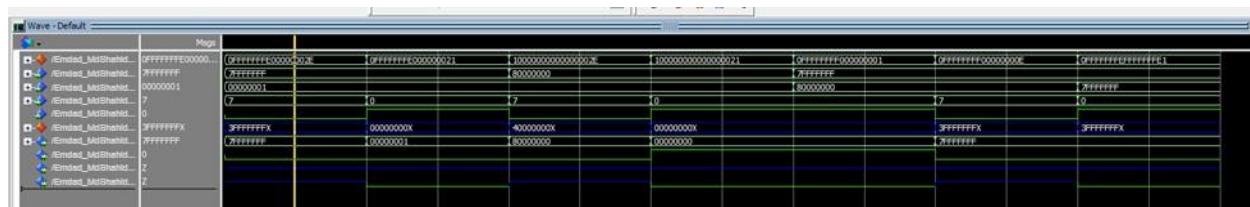


Figure 53: case a

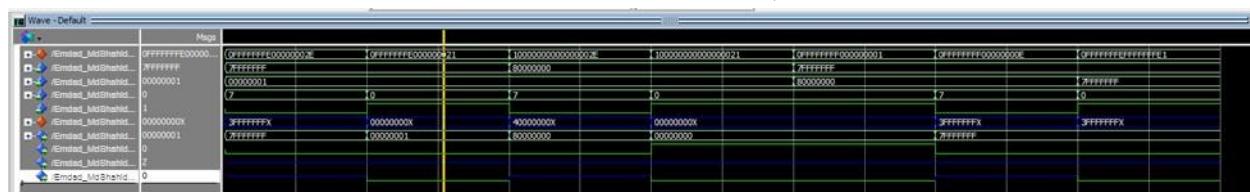


Figure 54: case b

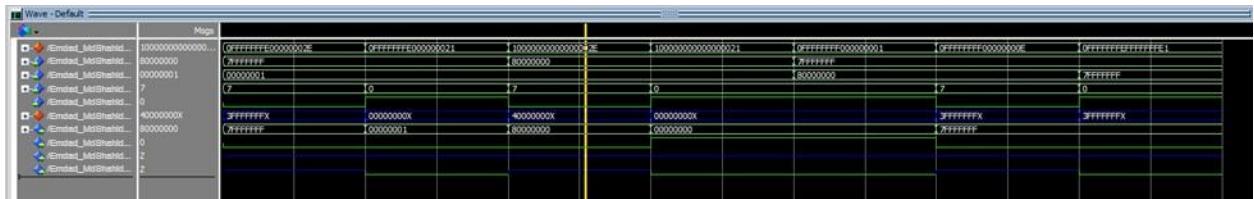


Figure 55: case c

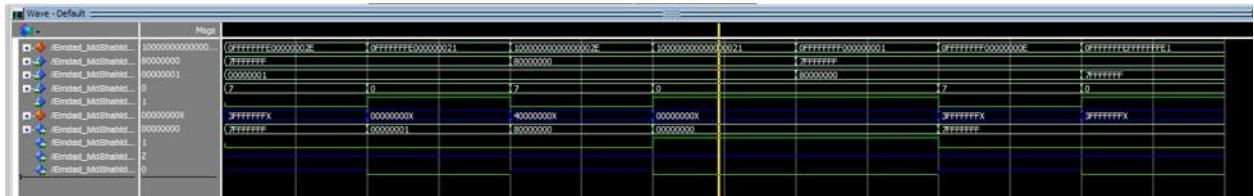


Figure 56: case d

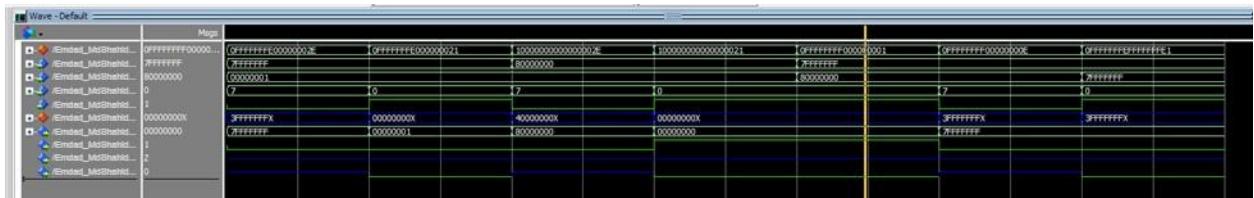


Figure 57: case e

- f. Most Positive N bit integer+ Most Negative N bit integer: a =
 $011111111111111111111111111111$, b = $10000000000000000000000000000000$ 4bit
 $= 0111$, obit = 0

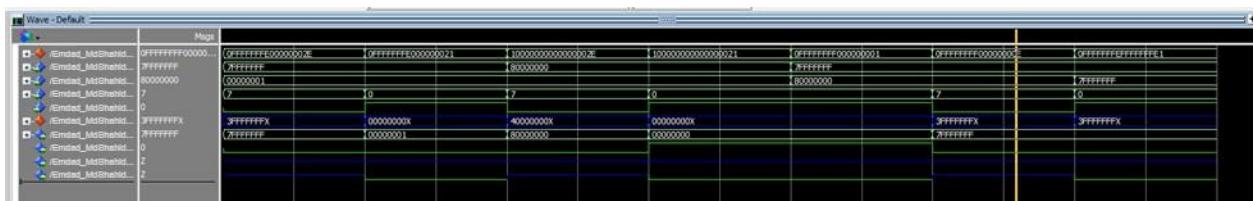


Figure 58: case f

- g. Most Positive N bit integer-Most Positive N bit integer: a =
 $01111111111111111111111111111111$, b = $01111111111111111111111111111111$ 4bit
 $\equiv 0000$, obit = 1

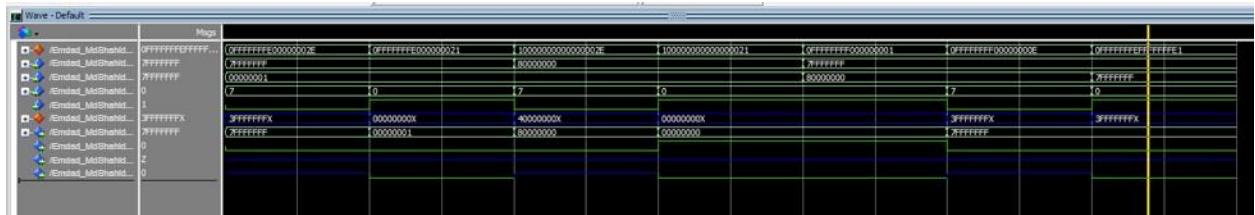


Figure 59: case g

TASK 9:

Simulation with LPM:

 ModelSim - INTEL FPGA STARTER EDITION 2020.1

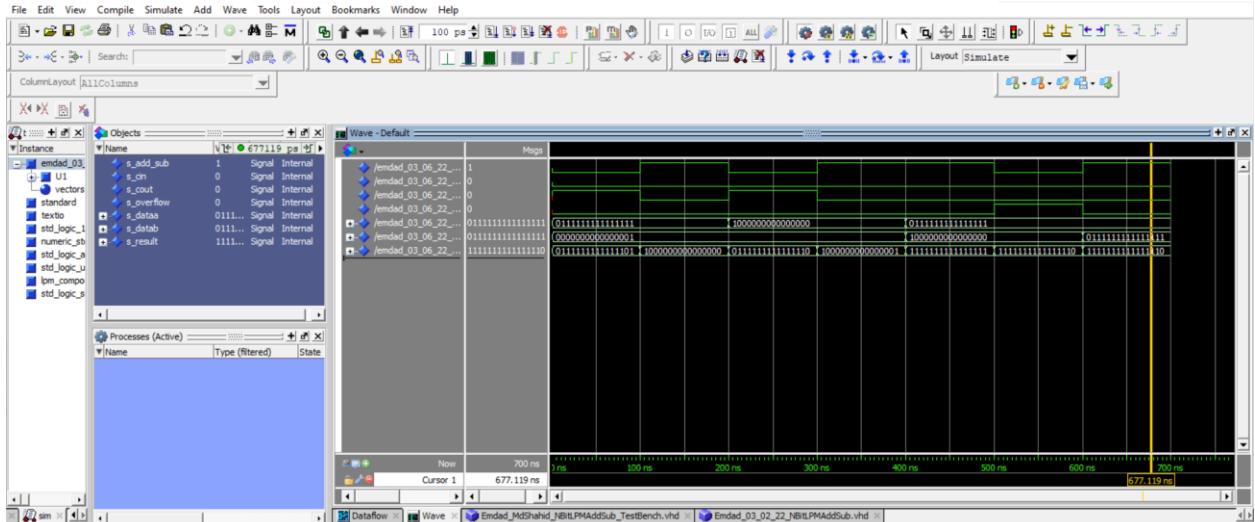


Figure 60: N-bit adder/subtractor unit using lpm simulation

- a. Most Positive N bit integer + 1: Obit = 0, a = 1111, b = 0001

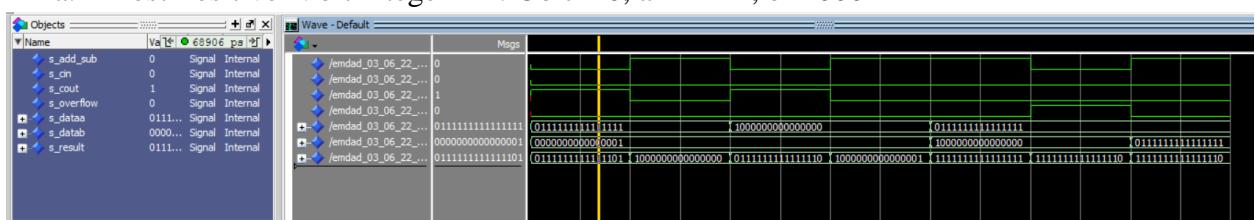


Figure 61: case a

- b. Most Positive N bit integer - 1: Obit = 1, a = 1111, b = 0001

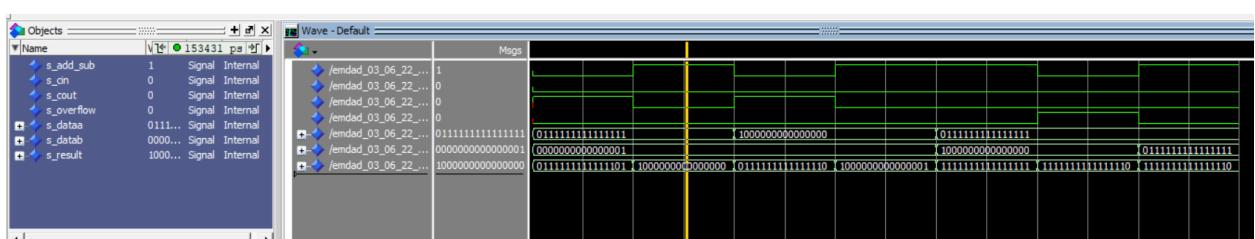


Figure 62: case b

c. Most Negative N bit integer + 1: Obit = 0, a = 1000, b = 0001

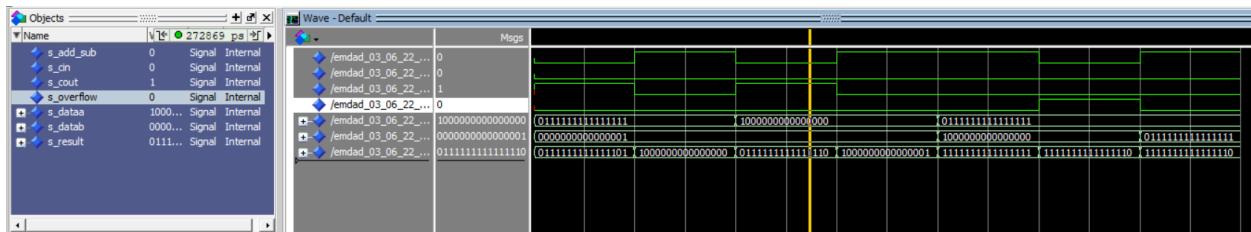


Figure 63: case c

d. Most Negative N bit integer - 1: Obit = 1, a = 1111, b = 0001

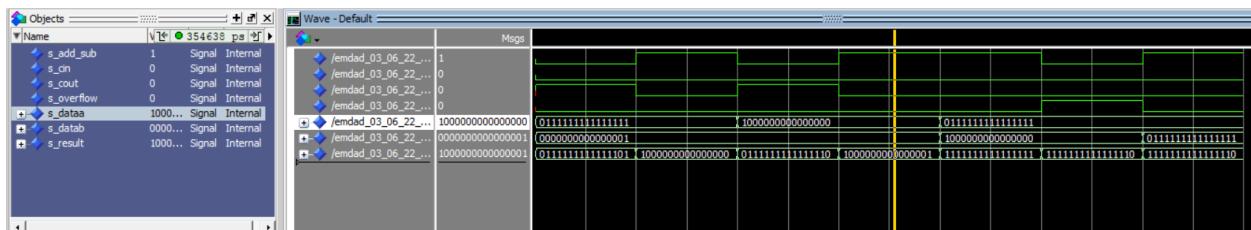


Figure 64: case d

e. Most Positive N bit integer-Most Negative N bit integer: Obit = 1, a = 1111, b = 1000

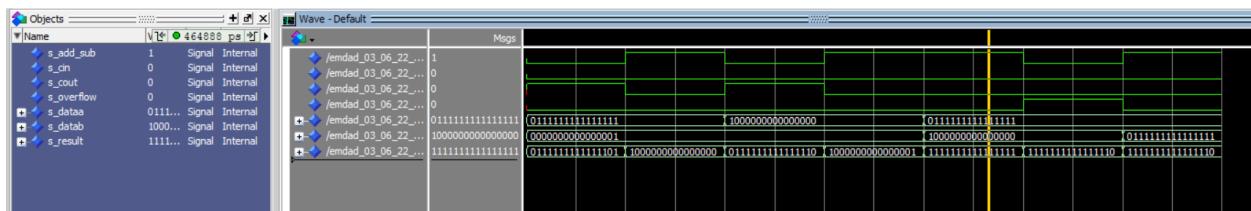


Figure 65: case e

f. Most Positive N bit integer+ Most Negative N bit integer: Obit = 0, a = 1111, b = 1000

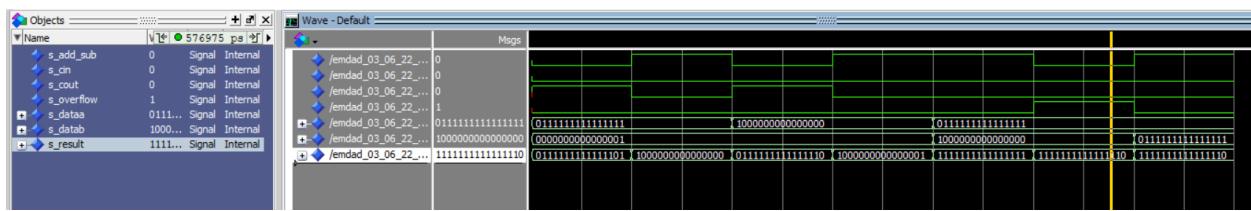


Figure 66: case f

g. Most Positive N bit integer-Most Positive N bit integer: Obit = 1, a = 1111, b = 1111

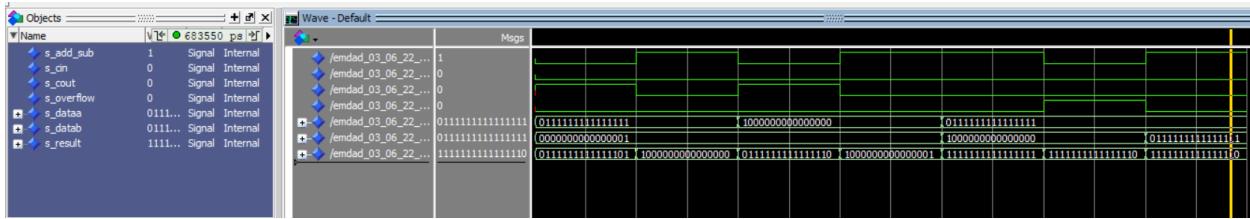


Figure 67: case g

10. TASK 10:

16 bits simulation

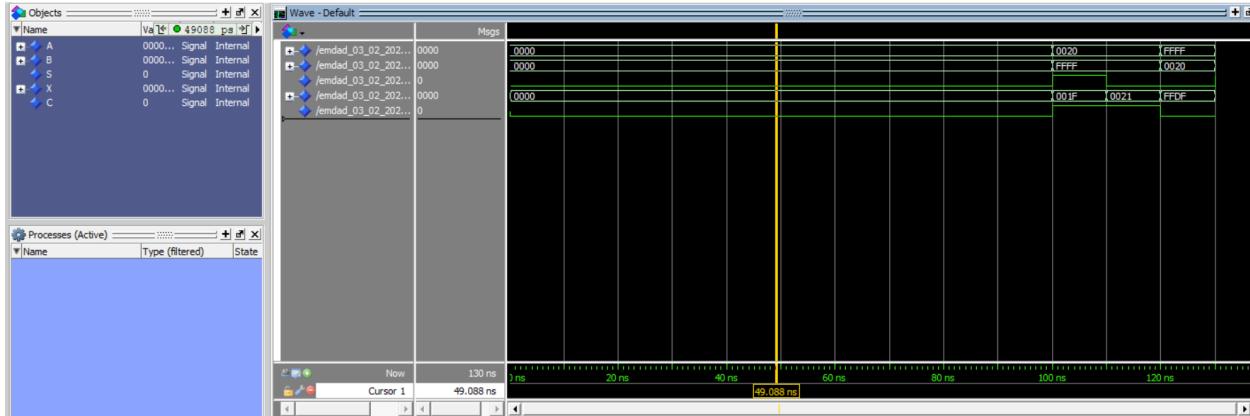


Figure 68: 16bit simulation by testbench without error

In figure 69, the hexadecimal values are different here. Therefore, it shows the error in the simulation.

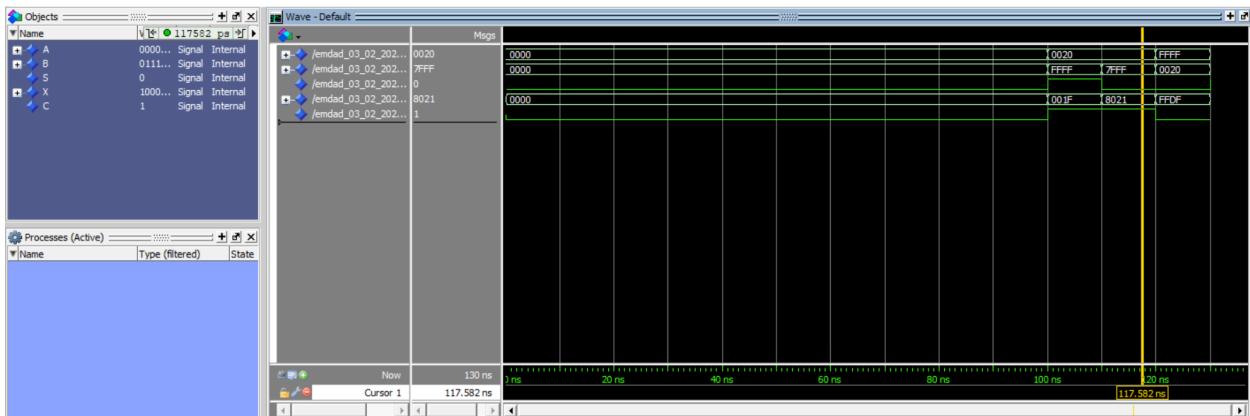


Figure 69: 16bit simulation with error

Conclusion:

In this assignment I learned about the concepts of adders and different features of Quartus and modelsim. I also learned deeply on testbench files, components, port mapping. I also learned to

create my own package. It will help me next time from coding from scratch. This assignment was helpful to learn as it was an introduction to VHDL, design Comparator, and how to test code. It was a good practice on Modelsim for circuit simulation and learn how to simulate comparator in Modelsim. The operations and condition logics were helpful to learn easily. I relearned and reviewed the concept again and which will help me in the course further. Overall, I learned a lot through this lab about adders and VHDL.