

Lab : BEQ, BNE, J MIPS OPERATIONS

MdShahid Bin Emdad

April 24th, 2022

CSC 34300

TABLE OF CONTENTS

Objective:	2
Description of Specifications, and Functionality:.....	2
Project Directory:.....	2
Half adder:	3
Full adder:	4
Adder package:	5
32bit add/sub:.....	6
32-bit register:.....	7
Register file.....	8
MIF file:	10
1-bit comparator:.....	11
32-bit comparator:.....	12
Sign extension:.....	14
2_1_MUX:	15
Jump Calculation:	16
Controller:	17
Main Unit:	18
Testbench:	20
Simulation:	22
Conclusion:	25

Objective:

The objective of this assignment is to learn and understand the specifications of comparator and use it with registers and mif file to compute next address. We were instructed to implement MIPS instructions BEQ, BNE and J.

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 (ps unit).

Project Directory:

In figure 1, we can see the project directory for the beq, bne, j mips operations lab.

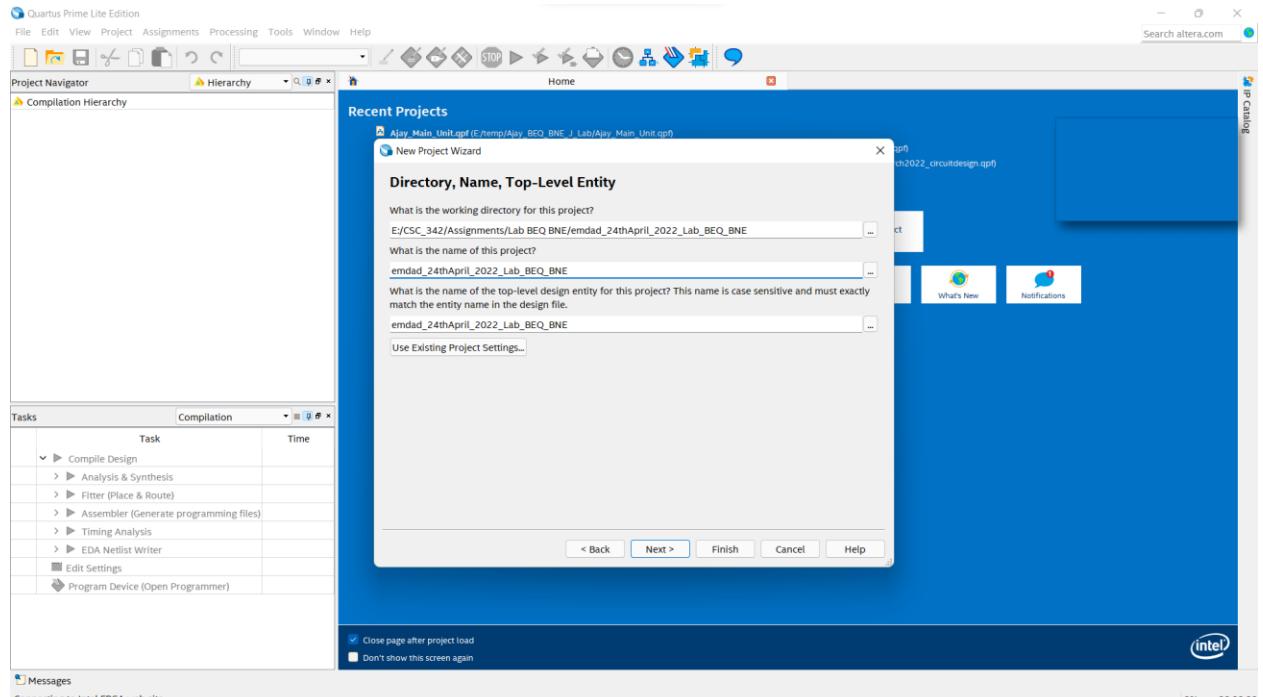


Figure 1: Project directory

In figure 2, we can see the project summary for the lab project.

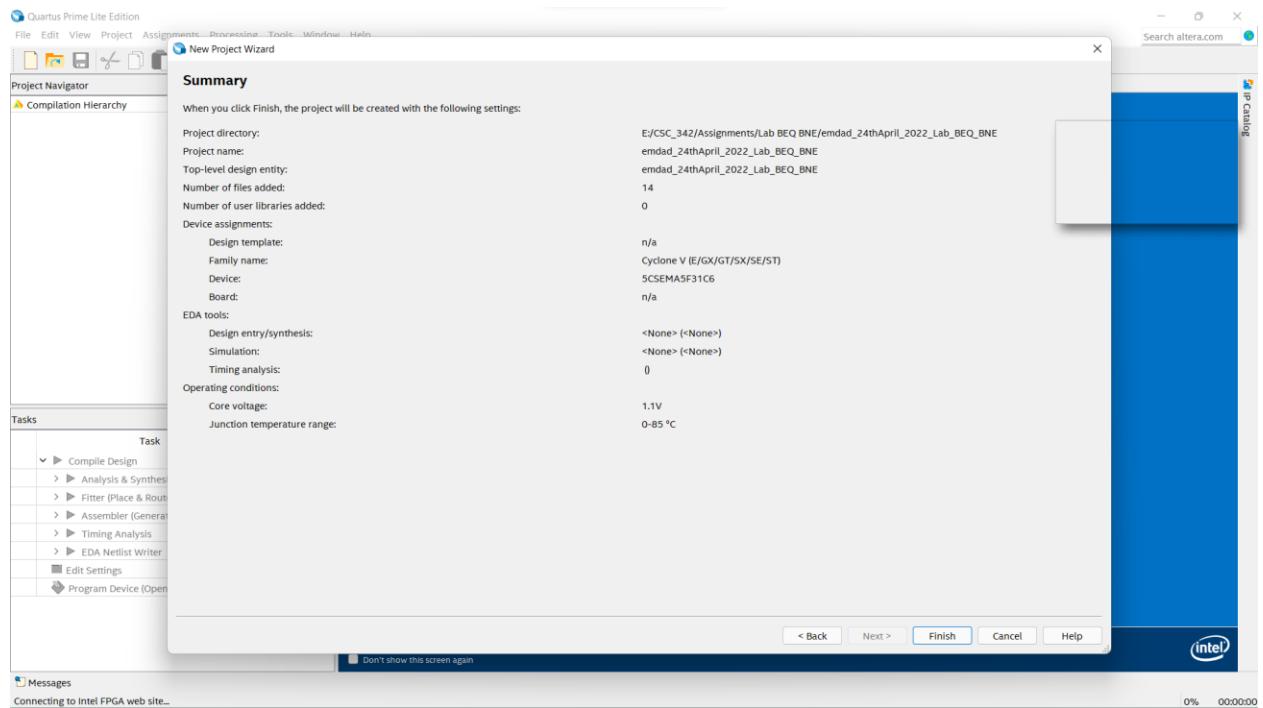


Figure 2: Project Summary

Half adder:

In figure 3, we can see the VHDL code for half adder and will be using this as a component to build full adder.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity emdad_half_adder is
5     port (emdad_x, emdad_y : in std_logic;
6           emdad_sum, emdad_carry : out std_logic);
7 end emdad_half_adder;
8
9 architecture arch of emdad_half_adder is
10 begin
11     emdad_sum <= emdad_x xor emdad_y;
12     emdad_carry <= emdad_x and emdad_y;
13 end arch;

```

Figure 3: VHDL code for half adder

In figure 4, we can see that the VHDL code compiled successfully.

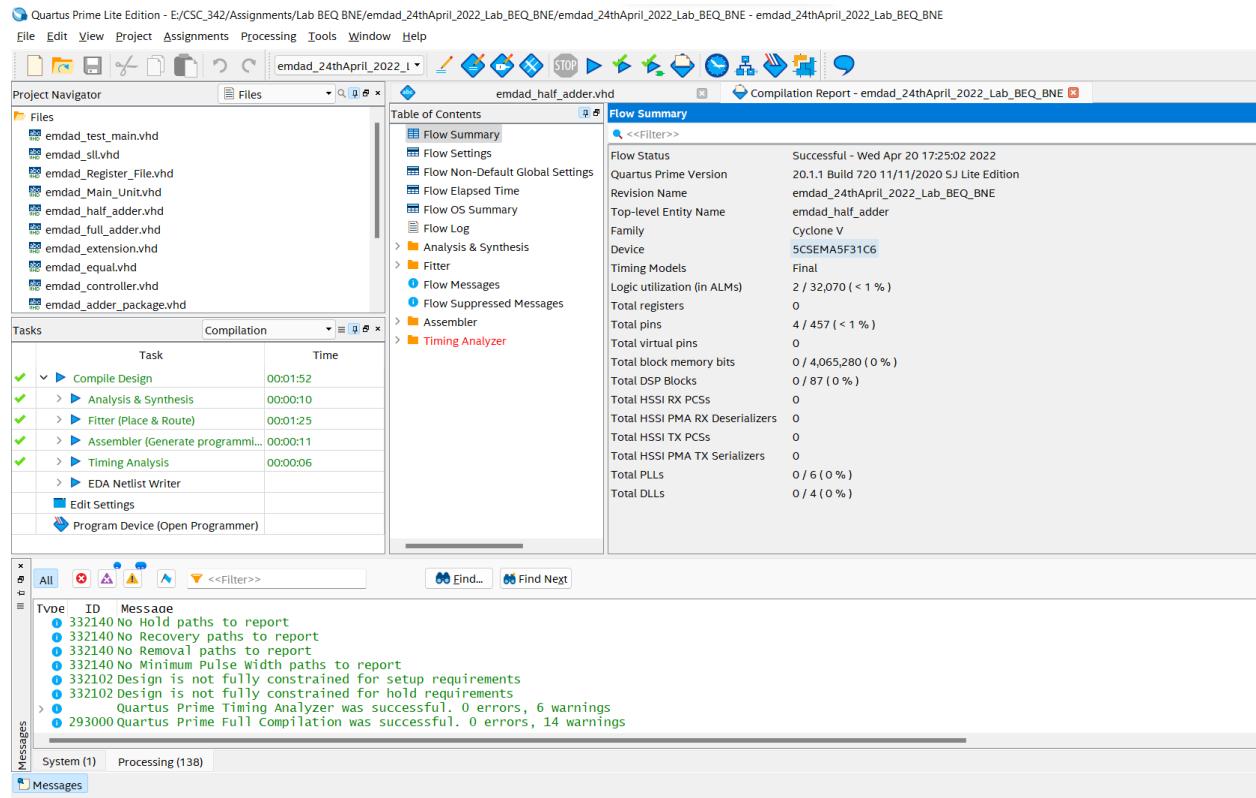


Figure 4: Successful compilation report

Full adder:

In figure 5, we can see the VHDL code for the full adder. I used half adder as a component.

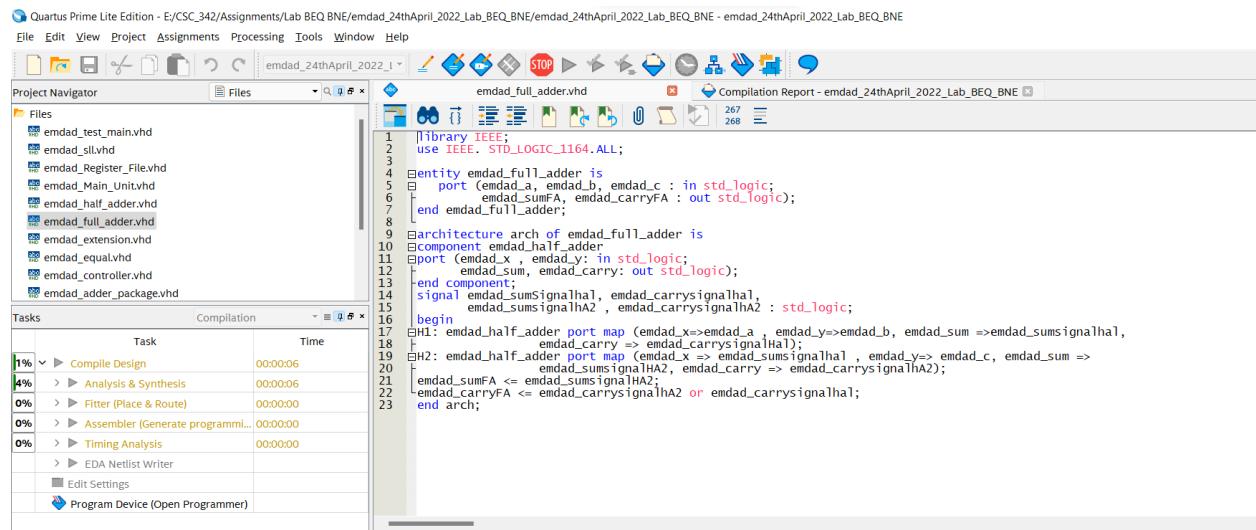


Figure 5: VHDL code for the full adder

In figure 6, we can see that the VHDL code compiled successfully.

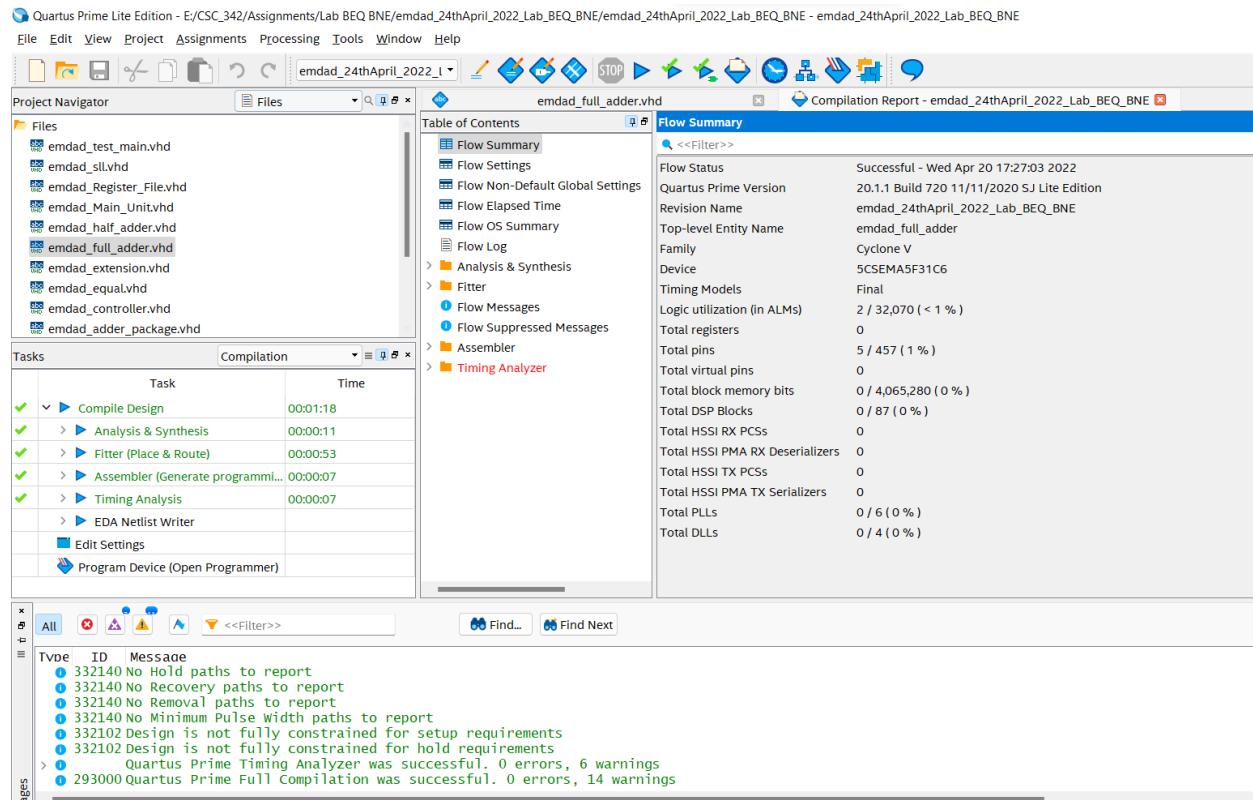


Figure 6: Successful compilation report.

Adder package:

In figure we can see that I am using both half and full adder as a component and creating the adder package to use it later in the project.

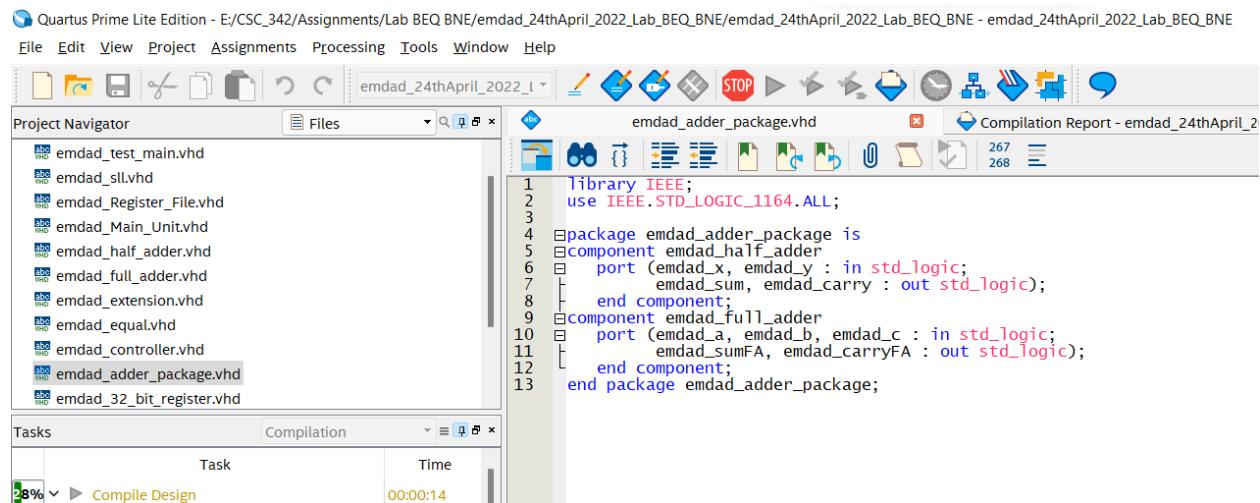


Figure 7: Adder package

In figure 8, we can see that the VHDL code compiled successfully.

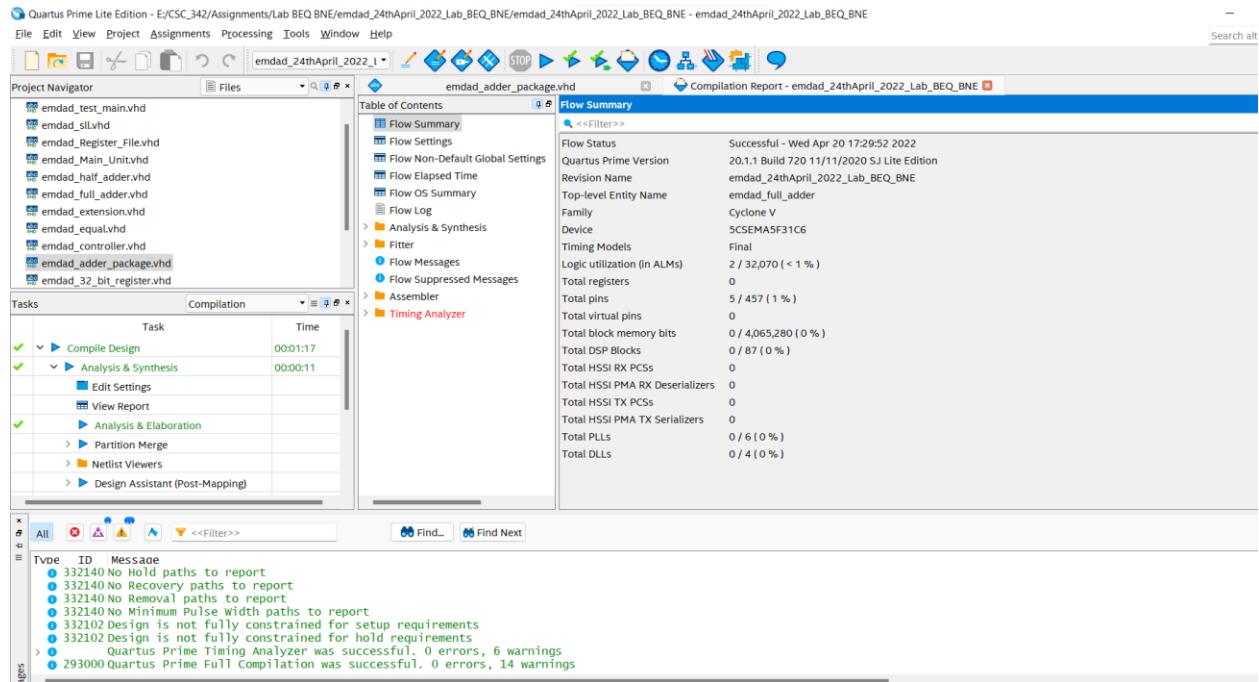


Figure 8: Successful compilation report.

32bit add/sub:

In figure 9, we can see the VHDL code for the 32 bit add/sub.

```

library work;
use work.emdad_adder_package.all;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity emdad_32_bit_adder is
    generic (n : integer :=32);
    port(emdad_x, emdad_y : in std_logic_vector (n-1 downto 0);
          emdad_cin : in std_logic;
          emdad_s : out std_logic_vector (n-1 downto 0);
          emdad_cout : out std_logic);
end emdad_32_bit_adder;

architecture arch of emdad_32_bit_adder is
    signal emdad_C, emdad_ycin, emdad_sTemp : std_logic_vector (n-1 downto 0);
begin
    FA: for i in 0 to n-1 generate
        emdad_ycin(i) <= emdad_y(i) xor emdad_cin;
        I1: if (i = 0) generate
            F1: emdad_full_adder port map (emdad_a => emdad_x(i), emdad_b => emdad_ycin(i),
                                            emdad_c=>emdad_cin, emdad_sumFA => emdad_sTemp(i),
                                            emdad_carryFA => emdad_c(i));
            emdad_s(i) <= emdad_sTemp(i);
        end generate I1;
        I2: if ((i < n-1) and (i > 0)) generate
            F1: emdad_full_adder port map (emdad_a => emdad_x(i), emdad_b => emdad_ycin(i),
                                            emdad_c=>emdad_c(i-1), emdad_sumFA => emdad_sTemp(i),
                                            emdad_carryFA => emdad_c(i));
            emdad_s(i) <= emdad_sTemp(i);
        end generate I2;
        I3: if (i = n-1) generate
            F2: emdad_full_adder port map (emdad_a => emdad_x(i), emdad_b=>emdad_ycin(i),
                                            emdad_c=>emdad_c(i-1), emdad_sumFA => emdad_sTemp(i),
                                            emdad_carryFA => emdad_c(i));
            emdad_cout <= emdad_c(i);
            emdad_s(i) <= emdad_sTemp(i);
        end generate I3;
    end generate FA;
end arch;

```

Figure 9: VHDL code for the 32 bit add/sub.

In figure 10, we can see that the VHDL code compiled successfully.

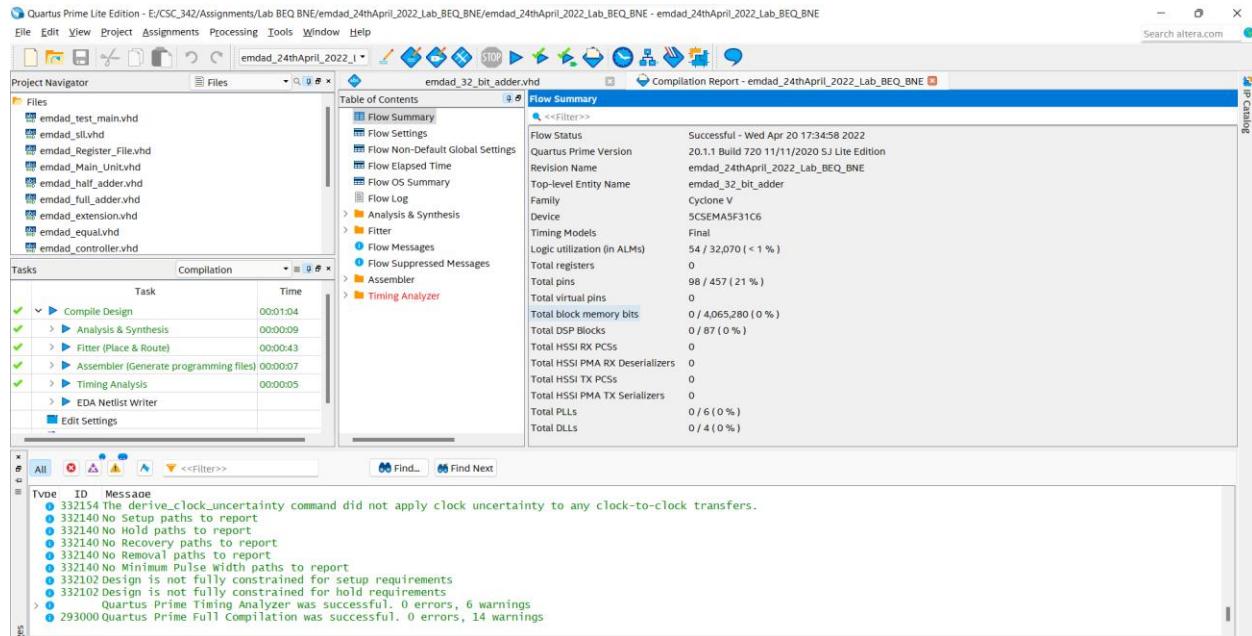


Figure 10: Successful compilation report.

32-bit register:

In figure 11, we can see the VHDL code for the 32-bit register. It will be used for PC register and instruction register.

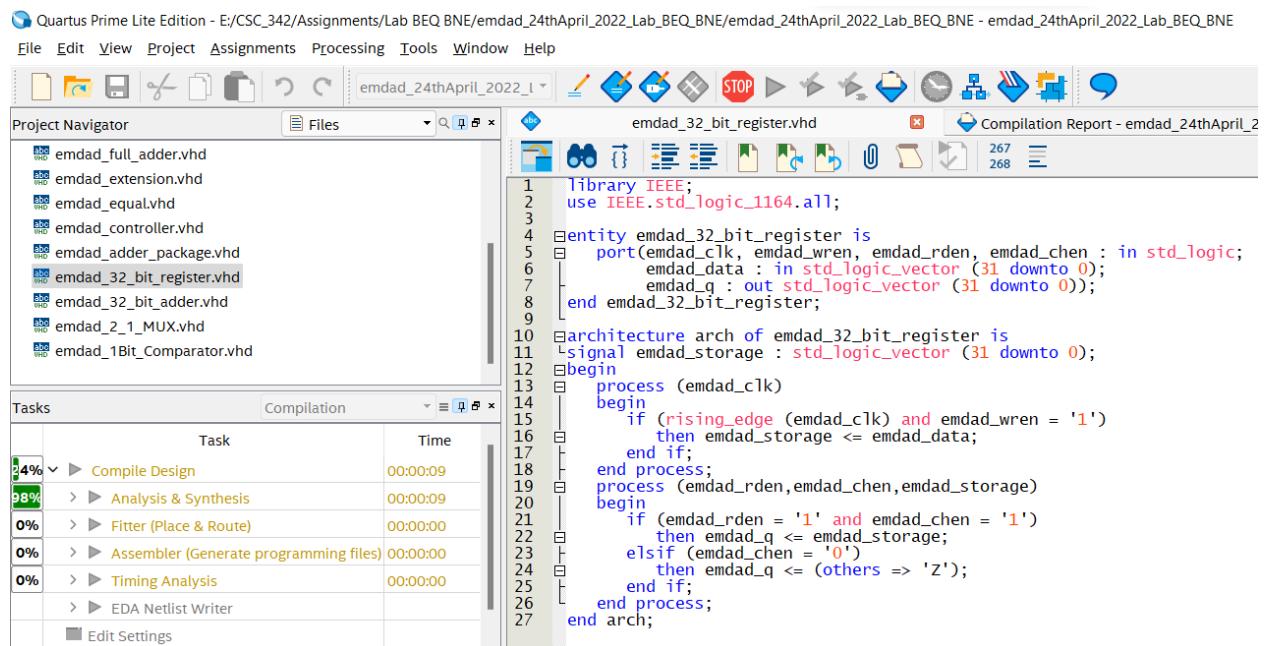


Figure 11: VHDL code for the 32-bit register

In figure 12, we can see that the VHDL code compiled successfully.

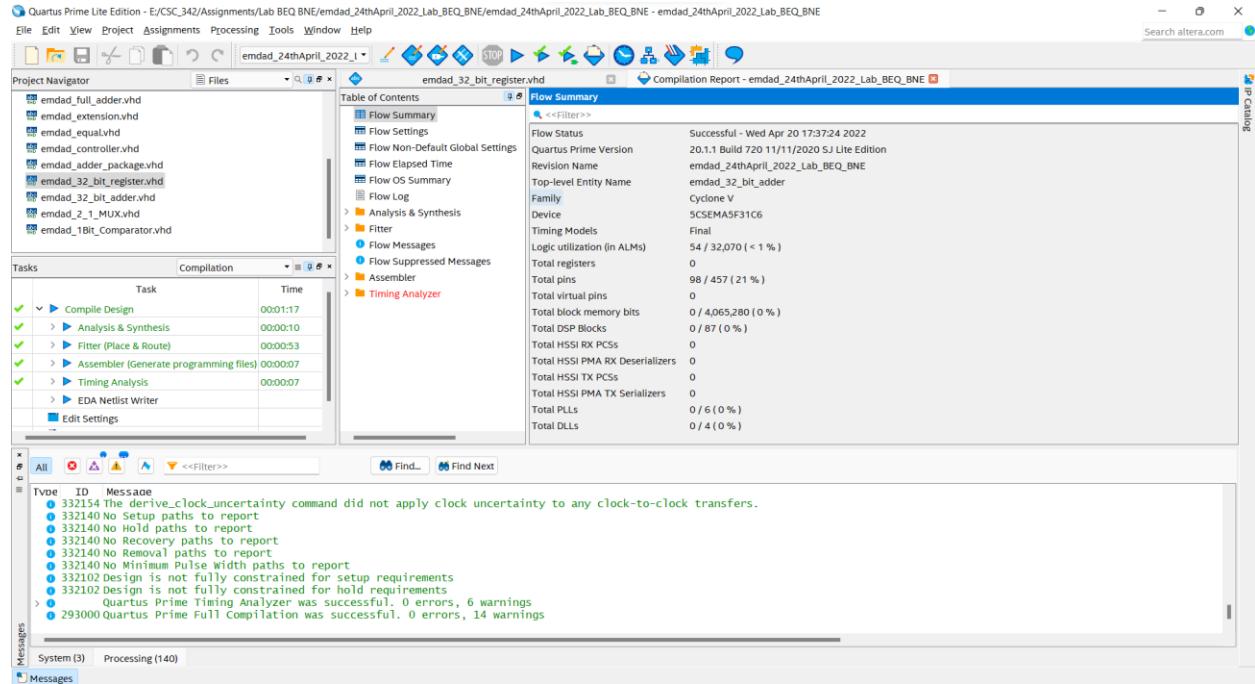


Figure 12: Successful compilation report.

Register file

In figure 13, we can see the VHDL code for register file. It is generated from LPM modules. I removed the extra comments to keep the code clean. I used 32-bit memory size and 32-bit registers. The wraddress is default to 0. We will be using this code to read the MIPS file for the inputs for rdaddress 1 and 2.

Quartus Prime Lite Edition - E/CSC_342/Assignments/Lab BEQ BNE/emdad_24thApril_2022_Lab_BEQ_BNE/emdad_24thApril_2022_Lab_BEQ_BNE - emdad_24thApril_2022_Lab_BEQ_BNE

The screenshot shows the Quartus Prime Lite Edition interface. The top menu includes File, Edit, View, Project, Assignments, Processing, Tools, Window, and Help. The Project Navigator on the left lists files such as emdad_Register_File_Data.mif, emdad_test_main.vhd, emdad_sll.vhd, emdad_Register_File.vhd, emdad_Main_Unit.vhd, emdad_half_adder.vhd, emdad_full_adder.vhd, emdad_extension.vhd, emdad_equal.vhd, emdad_controller.vhd, emdad_adder_package.vhd, emdad_32_bit_register.vhd, emdad_32_bit_adder.vhd, emdad_2_1_MUX.vhd, and emdad_1Bit_Compator.vhd. The main pane displays the VHDL code for emdad_Register_File.vhd, which defines a register file entity and architecture. The architecture uses an altsyncram component with various generic and port map settings. A compilation report on the right shows progress for tasks like Compile Design, Analysis & Synthesis, and Timing Analysis.

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.all;
3
4 LIBRARY altera_mf;
5 USE altera_mf.altera_mf_components.all;
6
7 ENTITY emdad_Register_File IS
8 PORT(
9     emdad_address_a : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
10    emdad_address_b : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
11    emdad_clock : IN STD_LOGIC;
12    emdad_data_a : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
13    emdad_data_b : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
14    emdad_wren_a : IN STD_LOGIC := '0';
15    emdad_wren_b : IN STD_LOGIC := '0';
16    emdad_q_a : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
17    emdad_q_b : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
18 );
19 END emdad_Register_File ;
20
21 ARCHITECTURE SYN OF emdad_Register_File IS
22
23 SIGNAL sub_wire0 : STD_LOGIC_VECTOR (31 DOWNTO 0);
24 SIGNAL sub_wire1 : STD_LOGIC_VECTOR (31 DOWNTO 0);
25
26 BEGIN
27     emdad_q_a <= sub_wire0 (31 DOWNTO 0);
28     emdad_q_b <= sub_wire1 (31 DOWNTO 0);
29     altsyncram_component : altsyncram
30     GENERIC MAP (
31         address_reg_b => "CLOCK0",
32         clock_enable_input_a => "BYPASS",
33         clock_enable_input_b => "BYPASS",
34         clock_enable_output_a => "BYPASS",
35         clock_enable_output_b => "BYPASS",
36         indata_reg_b => "CLOCK0",
37         init_file=>"emdad_Register_File_Data.mif",
38         intended_device_family=>"Cyclone V",
39         lpm_type=>"altsyncram",
40         numwords_a => 32,
41         numwords_b => 32,
42         operation_mode => "BIDIR_DUAL_PORT",
43         outdata_aclr_a => "NONE",
44         outdata_aclr_b => "NONE",
45         outdata_reg_a => "UNREGISTERED",
46         outdata_reg_b => "UNREGISTERED",
47         power_up_uninitialized => "FALSE",
48         read_during_write_mode_mixed_ports => "DONT_CARE",
49         read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
50         read_during_write_mode_port_b => "NEW_DATA_NO_NBE_READ",
51         widthad_a => 5,
52         widthad_b => 5,
53         width_a => 32,
54         width_b => 32,
55         width_bytewidth_a => 1,
56         width_bytewidth_b => 1,
57         wrcontrol_wraddress_reg_b => "CLOCK0"
58     )
59     PORT MAP (
60         address_a => emdad_address_a,
61         address_b => emdad_address_b,
62         clock0 => emdad_clock,
63         data_a => emdad_data_a,
64         data_b => emdad_data_b,
65         wren_a => emdad_wren_a,
66         wren_b => emdad_wren_b,
67         q_a => sub_wire0,
68         q_b => sub_wire1
69     );
70
71 END SYN;

```

Figure 13: VHDL code for register file (Part 1)

The screenshot continues the VHDL code for emdad_Register_File.vhd, showing the completion of the PORT MAP section and the end of the SYNTHESIS architecture. The compilation report on the left shows tasks like Compile Design, Analysis & Synthesis, and Timing Analysis completed successfully.

Figure 14: VHDL code for register file (Part 2)

In figure 15, we can see that the VHDL code compiled successfully.

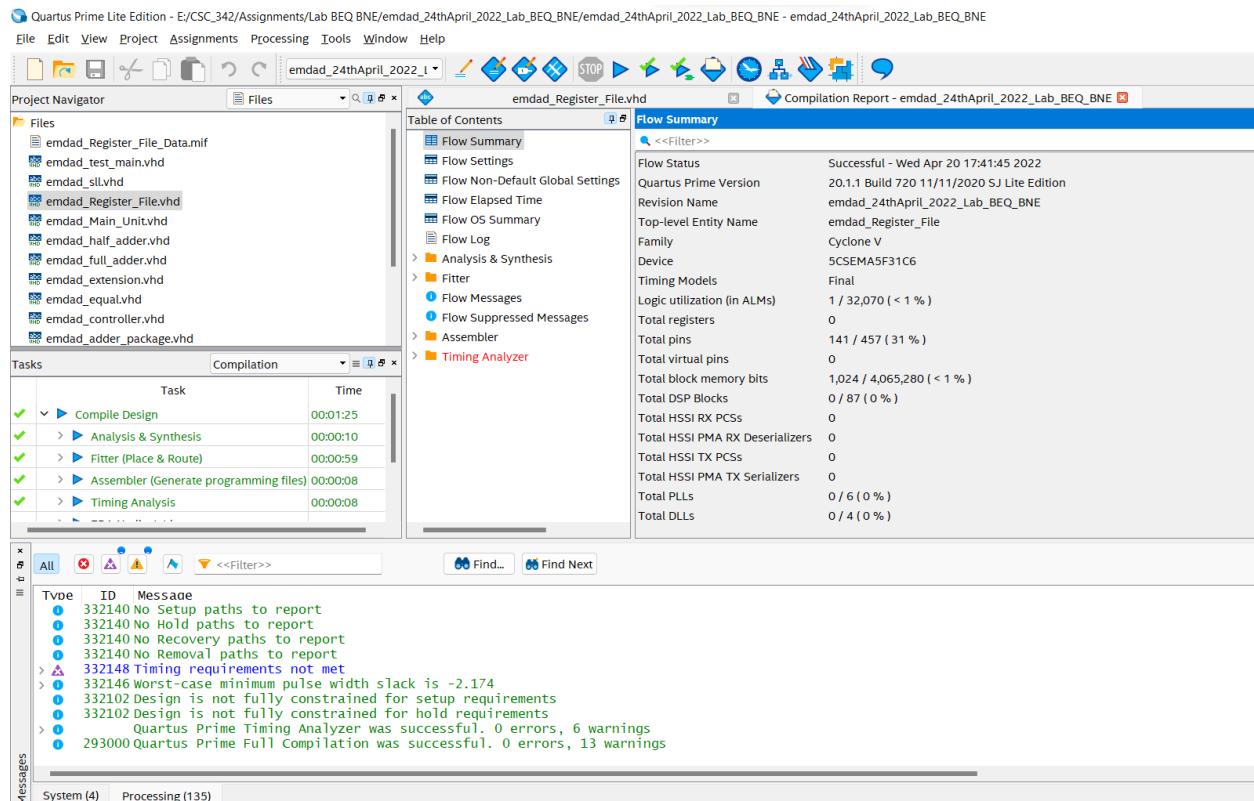


Figure 15: Successful compilation report.

MIF file:

In figure 16, we can see the MIPS file. I randomly entered some hexadecimal inputs for the simulation. I did memory size and register size both 32-bit.

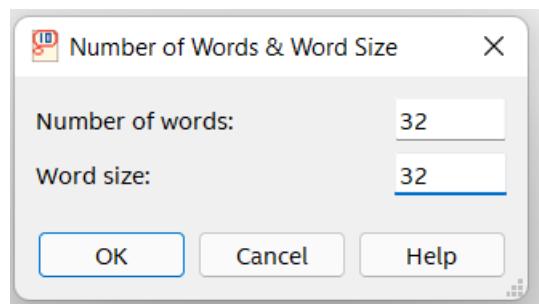


Figure 16: MIFS file specification

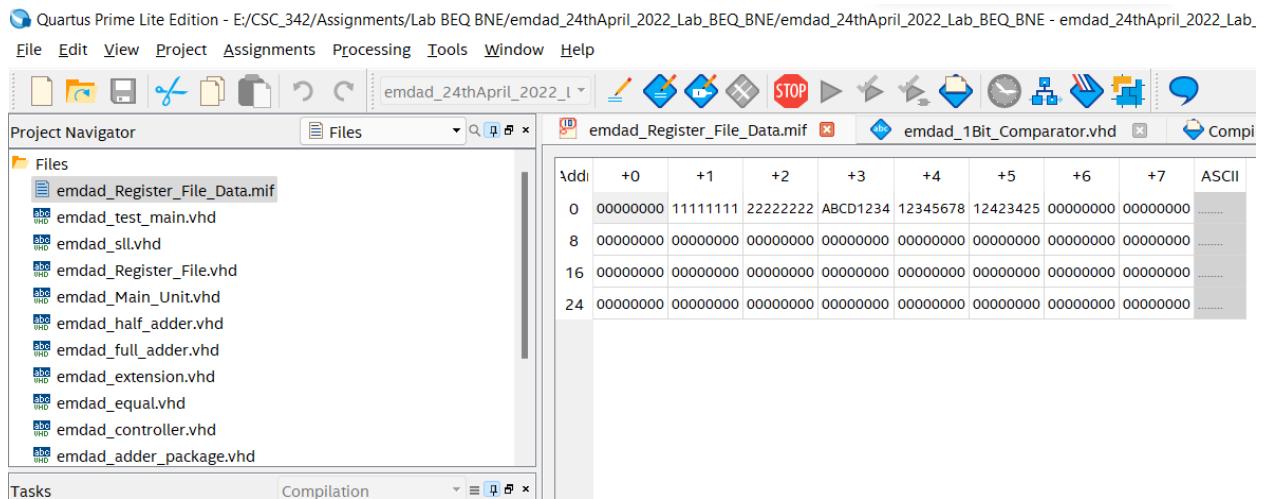


Figure 17: MIPS file for the simulation

1-bit comparator:

In figure 17, we can see the VHDL code for the 1-bit comparator. It will be used to compute the output for BNE and BEQ.

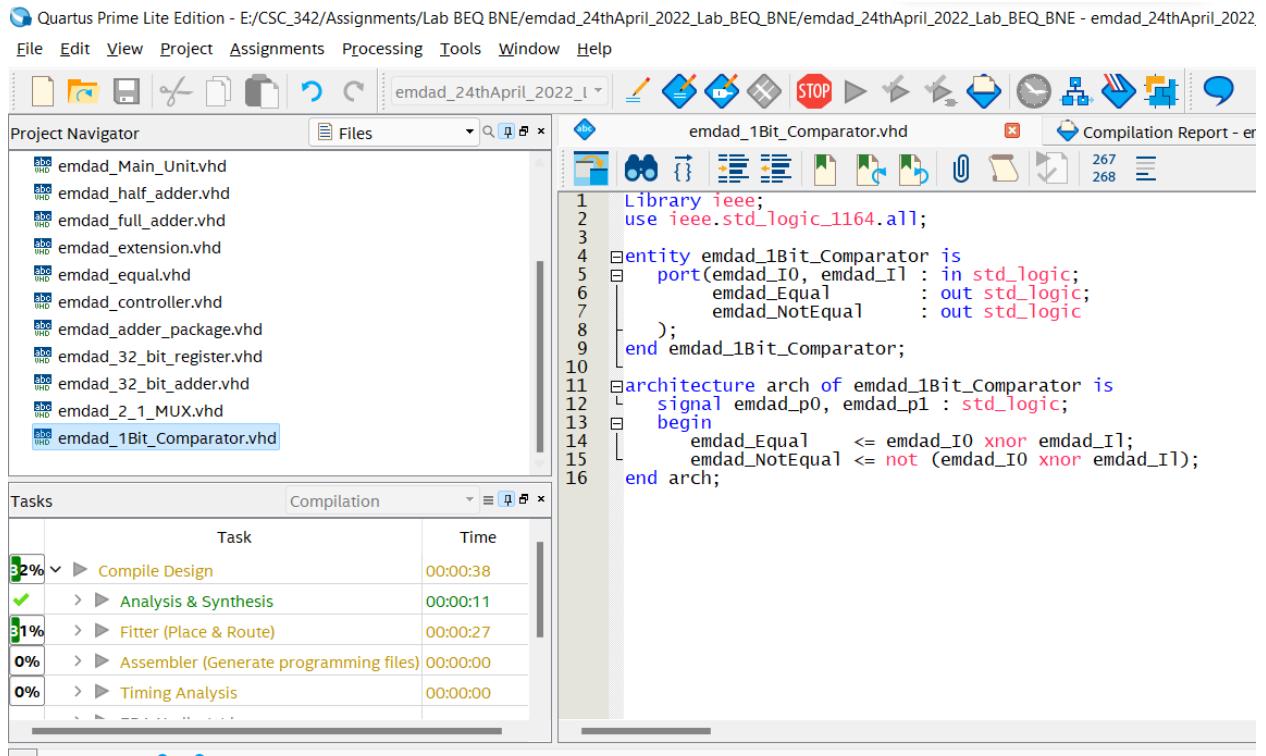


Figure 18: VHDL code for the 1-bit comparator

In figure 19, we can see that the VHDL code compiled successfully.

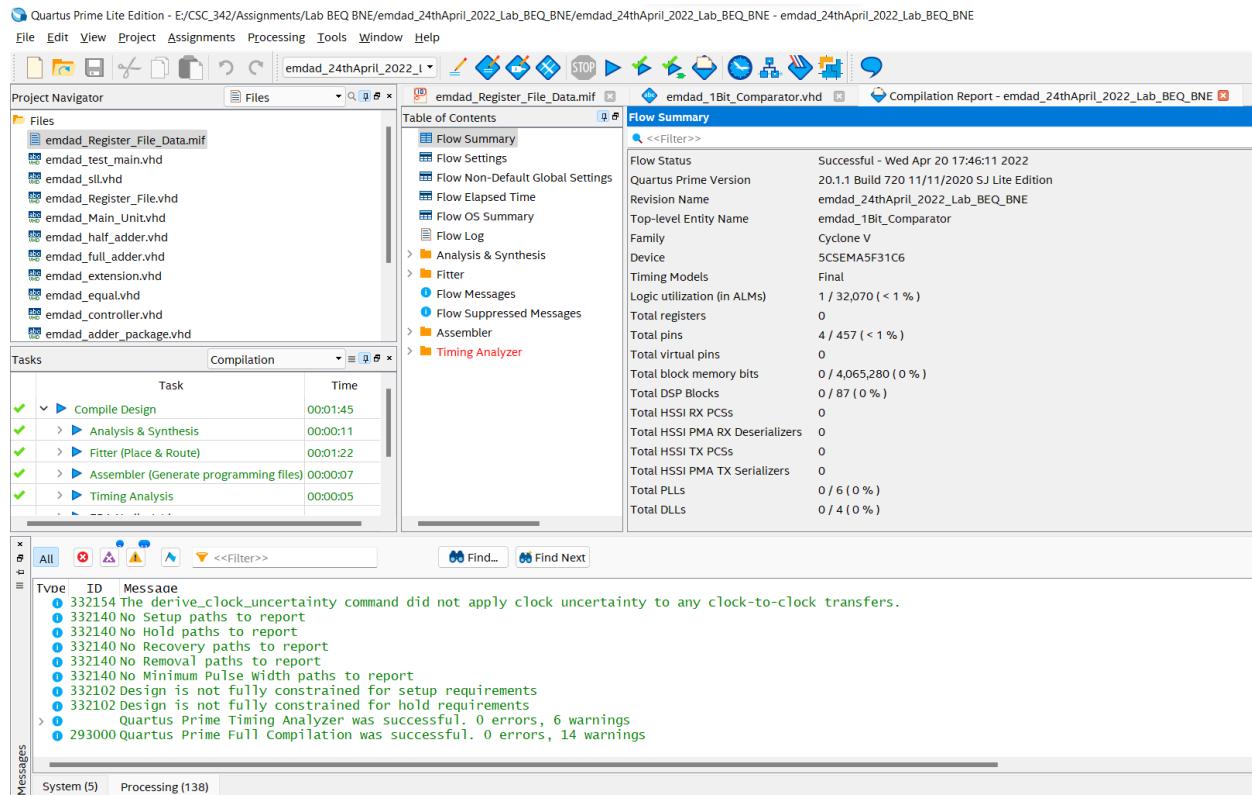


Figure 19: Successful compilation report.

32-bit comparator:

In figure 20, we can see the vhdl code for the 32-bit comparator. The file computes the equal and non-equal for each bit. The when else returns the output on condition for BEQ and BNE.

```

library ieee;
use ieee.std_logic_1164.all;

entity emdad_equal is
  port (emdad_RS, emdad_RT : in std_logic_vector (31 downto 0);
        emdad_OP : out std_logic;
        emdad_Cond : out std_logic);
end emdad_equal;

architecture arch of emdad_equal is
begin
  component emdad_1Bit_comparator port(
    emdad_I0, emdad_I1 : in std_logic;
    emdad_Equal : out std_logic;
    emdad_NotEqual : out std_logic);
  end component;
  signal emdad_EQ, emdad_NEQ : std_logic_vector (31 downto 0);
  begin
    F1: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (0), emdad_I1 => emdad_RT (0),
                                         emdad_Equal => emdad_EQ (0), emdad_NotEqual => emdad_NEQ (0));
    F2: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (1), emdad_I1 => emdad_RT (1),
                                         emdad_Equal => emdad_EQ (1), emdad_NotEqual => emdad_NEQ (1));
    F3: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (2), emdad_I1 => emdad_RT (2),
                                         emdad_Equal => emdad_EQ (2), emdad_NotEqual => emdad_NEQ (2));
    F4: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (3), emdad_I1 => emdad_RT (3),
                                         emdad_Equal => emdad_EQ (3), emdad_NotEqual => emdad_NEQ (3));
    F5: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (4), emdad_I1 => emdad_RT (4),
                                         emdad_Equal => emdad_EQ (4), emdad_NotEqual => emdad_NEQ (4));
    F6: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (5), emdad_I1 => emdad_RT (5),
                                         emdad_Equal => emdad_EQ (5), emdad_NotEqual => emdad_NEQ (5));
    F7: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (6), emdad_I1 => emdad_RT (6),
                                         emdad_Equal => emdad_EQ (6), emdad_NotEqual => emdad_NEQ (6));
    F8: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (7), emdad_I1 => emdad_RT (7),
                                         emdad_Equal => emdad_EQ (7), emdad_NotEqual => emdad_NEQ (7));
    F9: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (8), emdad_I1 => emdad_RT (8),
                                         emdad_Equal => emdad_EQ (8), emdad_NotEqual => emdad_NEQ (8));
    F10: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (9), emdad_I1 => emdad_RT (9),
                                         emdad_Equal => emdad_EQ (9), emdad_NotEqual => emdad_NEQ (9));
    F11: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (10), emdad_I1 => emdad_RT (10),
                                         emdad_Equal => emdad_EQ (10), emdad_NotEqual => emdad_NEQ (10));
    F12: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (11), emdad_I1 => emdad_RT (11),
                                         emdad_Equal => emdad_EQ (11), emdad_NotEqual => emdad_NEQ (11));
    F13: emdad_1Bit_comparator port map (emdad_I0 => emdad_RS (12), emdad_I1 => emdad_RT (12),
                                         emdad_Equal => emdad_EQ (12), emdad_NotEqual => emdad_NEQ (12));
  end;

```

Figure 20: VHDL code for 32-bit comparator (part 1)

The screenshot shows the Quartus Prime Lite interface with the project 'emdad_24thApril_2022' open. The 'Files' tab is selected, displaying various VHDL files including 'emdad_equal.vhd'. The main pane shows the VHDL code for a 32-bit comparator, which includes multiple 1-bit comparator ports (F14 to F26) and a main assignment block. The code uses various comparison operators like >, <, =, and >=.

```

F14: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (3), emdad_I1 => emdad_RT (3),
47 emdad_Equal => emdad_eq (1), emdad_NotEqual => emdad_neq (1));
48 F15: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (4), emdad_I1 => emdad_RT (4),
49 emdad_Equal => emdad_eq (4), emdad_NotEqual => emdad_neq (4));
50 F16: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (5), emdad_I1 => emdad_RT (5),
51 emdad_Equal => emdad_eq (5), emdad_NotEqual => emdad_neq (5));
52 F17: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (6), emdad_I1 => emdad_RT (6),
53 emdad_Equal => emdad_eq (6), emdad_NotEqual => emdad_neq (6));
54 F18: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (7), emdad_I1 => emdad_RT (7),
55 emdad_Equal => emdad_eq (7), emdad_NotEqual => emdad_neq (7));
56 F20: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (8), emdad_I1 => emdad_RT (8),
57 emdad_Equal => emdad_eq (8), emdad_NotEqual => emdad_neq (8));
58 F19: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (9), emdad_I1 => emdad_RT (9),
59 emdad_Equal => emdad_eq (9), emdad_NotEqual => emdad_neq (9));
60 F21: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (10), emdad_I1 => emdad_RT (10),
61 emdad_Equal => emdad_eq (10), emdad_NotEqual => emdad_neq (10));
62 F22: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (11), emdad_I1 => emdad_RT (11),
63 emdad_Equal => emdad_eq (11), emdad_NotEqual => emdad_neq (11));
64 F23: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (12), emdad_I1 => emdad_RT (12),
65 emdad_Equal => emdad_eq (12), emdad_NotEqual => emdad_neq (12));
66 F24: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (13), emdad_I1 => emdad_RT (13),
67 emdad_Equal => emdad_eq (13), emdad_NotEqual => emdad_neq (13));
68 F25: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (14), emdad_I1 => emdad_RT (14),
69 emdad_Equal => emdad_eq (14), emdad_NotEqual => emdad_neq (14));
70 F26: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (15), emdad_I1 => emdad_RT (15),
71 emdad_Equal => emdad_eq (15), emdad_NotEqual => emdad_neq (15));
72 F27: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (16), emdad_I1 => emdad_RT (16),
73 emdad_Equal => emdad_eq (16), emdad_NotEqual => emdad_neq (16));
74 F28: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (17), emdad_I1 => emdad_RT (17),
75 emdad_Equal => emdad_eq (17), emdad_NotEqual => emdad_neq (17));
76 F29: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (18), emdad_I1 => emdad_RT (18),
77 emdad_Equal => emdad_eq (18), emdad_NotEqual => emdad_neq (18));
78 F30: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (19), emdad_I1 => emdad_RT (19),
79 emdad_Equal => emdad_eq (19), emdad_NotEqual => emdad_neq (19));
80 F31: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (20), emdad_I1 => emdad_RT (20),
81 emdad_Equal => emdad_eq (20), emdad_NotEqual => emdad_neq (20));
82 F32: emdad_1bit_comparator port map (emdad_I0 => emdad_RS (21), emdad_I1 => emdad_RT (21),
83 emdad_Equal => emdad_eq (21), emdad_NotEqual => emdad_neq (21));
84 emdad_cons <= (emdad_eq (0) and emdad_eq (1) and emdad_eq (2) and emdad_eq (3) and emdad_eq (4)
85 and emdad_eq (5) and emdad_eq (6) and emdad_eq (7) and emdad_eq (8) and emdad_eq (9) and emdad_eq (10)
86 and emdad_eq (11) and emdad_eq (12) and emdad_eq (13) and emdad_eq (14) and emdad_eq (15) and emdad_eq (16)
87 and emdad_eq (17) and emdad_eq (18) and emdad_eq (19) and emdad_eq (20) and emdad_eq (21) and emdad_eq (22)
88 and emdad_eq (23) and emdad_eq (24) and emdad_eq (25) and emdad_eq (26) and emdad_eq (27) and emdad_eq (28)
89 and emdad_eq (29) and emdad_eq (30) and emdad_eq (31)) WHEN emdad_OP = "000100"
90 or emdad_neq (0) or emdad_neq (1) or emdad_neq (2) or emdad_neq (3) or emdad_neq (4) or emdad_neq (5)
91 or emdad_neq (6) or emdad_neq (7) or emdad_neq (8) or emdad_neq (9) or emdad_neq (10) or emdad_neq (11)
92 or emdad_neq (12) or emdad_neq (13) or emdad_neq (14) or emdad_neq (15) or emdad_neq (16)
93 or emdad_neq (17) or emdad_neq (18) or emdad_neq (19) or emdad_neq (20) or emdad_neq (21)
94 or emdad_neq (22) or emdad_neq (23) or emdad_neq (24) or emdad_neq (25) or emdad_neq (26)
95 or emdad_neq (27) or emdad_neq (28) or emdad_neq (29) or emdad_neq (30)
96 or emdad_neq (31)) WHEN emdad_OP = "000101";
97 end arch;
98

```

Figure 21: VHDL code for 32-bit comparator (part 2)

This screenshot shows the continuation of the VHDL code from Figure 21. It includes lines 93 through 98, which define the logic for the 'emdad_cons' signal based on the 'emdad_OP' value. The code uses 'or' and 'and' operators to combine various equality and inequality conditions.

```

93 or emdad_neq (12) or emdad_neq (13) or emdad_neq (14) or emdad_neq (15) or emdad_neq (16)
94 or emdad_neq (17) or emdad_neq (18) or emdad_neq (19) or emdad_neq (20) or emdad_neq (21)
95 or emdad_neq (22) or emdad_neq (23) or emdad_neq (24) or emdad_neq (25) or emdad_neq (26)
96 or emdad_neq (27) or emdad_neq (28) or emdad_neq (29) or emdad_neq (30)
97 or emdad_neq (31)) WHEN emdad_OP = "000101";
98 end arch;
99

```

Figure 22: VHDL code for 32-bit comparator (part 3)

In figure 23, we can see that the VHDL code compiled successfully.

The screenshot shows the Quartus Prime Lite interface with the project 'emdad_24thApril_2022' open. The 'Compilation' tab is selected, showing a successful compilation status. The 'Flow Summary' table provides detailed information about the compilation process, including flow status, revision name, and timing models. The 'Messages' section at the bottom shows no errors or warnings.

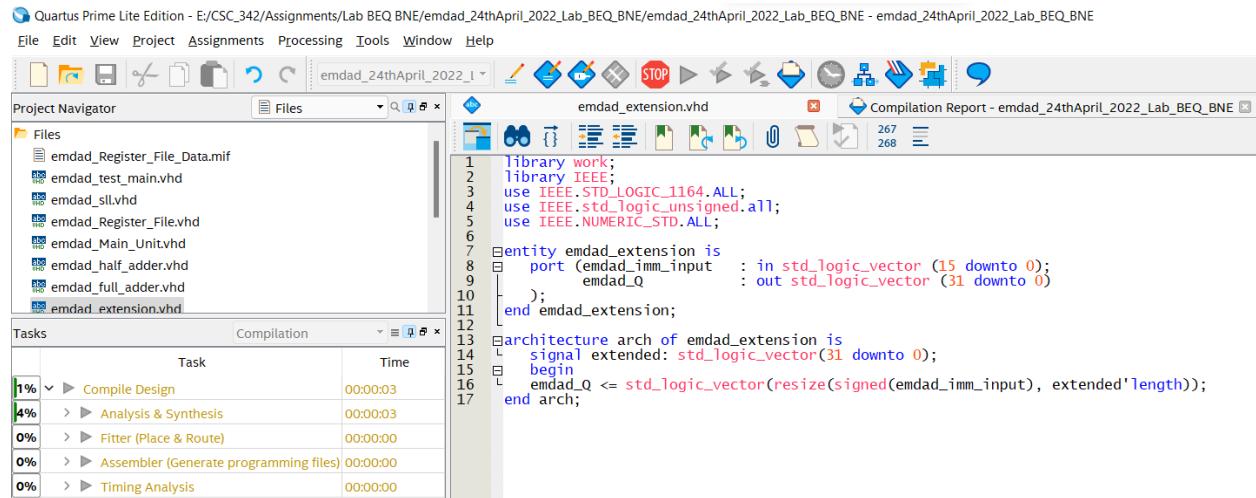
Flow	Summary
Flow Status	Successful - Wed Apr 20 17:49:11 2022
Quartus Prime Version	20.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	emdad_24thApril_2022_Lab_BEQ_BNE
Top-level Entity Name	emdad_1bit_comparator
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Fina
Logic utilization (in ALMs)	1 / 32,070 (< 1 %)
Total registers	0
Total pins	4 / 457 (< 1 %)
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 %)

Trove ID Message
332154 No derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
332240 No Setup paths to report
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
Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 14 warnings

Figure 23: Successful compilation report.

Sign extension:

In figure 24, we vhdl code for sign extension. I am using this to extend the most significant bit for IMM. It is used to extend the 16bit register data to 32bit register so I can use the 32-bit add/sub



```

Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab BEQ BNE/emdad_24thApril_2022_Lab_BEQ_BNE/emdad_24thApril_2022_Lab_BEQ_BNE - emdad_24thApril_2022_Lab_BEQ_BNE

File Edit View Project Assignments Processing Tools Window Help

Project Navigator Files emdad_extension.vhd Compilation Report - emdad_24thApril_2022_Lab_BEQ_BNE
emdad_Register_File_Data.mif
emdad_test_main.vhd
emdad_sll.vhd
emdad_Register_File.vhd
emdad_Main_Unit.vhd
emdad_half_adder.vhd
emdad_full_adder.vhd
emdad_extension.vhd

Tasks Compilation
Task Time
1% ▶ Compile Design 00:00:03
4% > Analysis & Synthesis 00:00:03
0% > Fitter (Place & Route) 00:00:00
0% > Assembler (Generate programming files) 00:00:00
0% > Timing Analysis 00:00:00

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

library work;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;

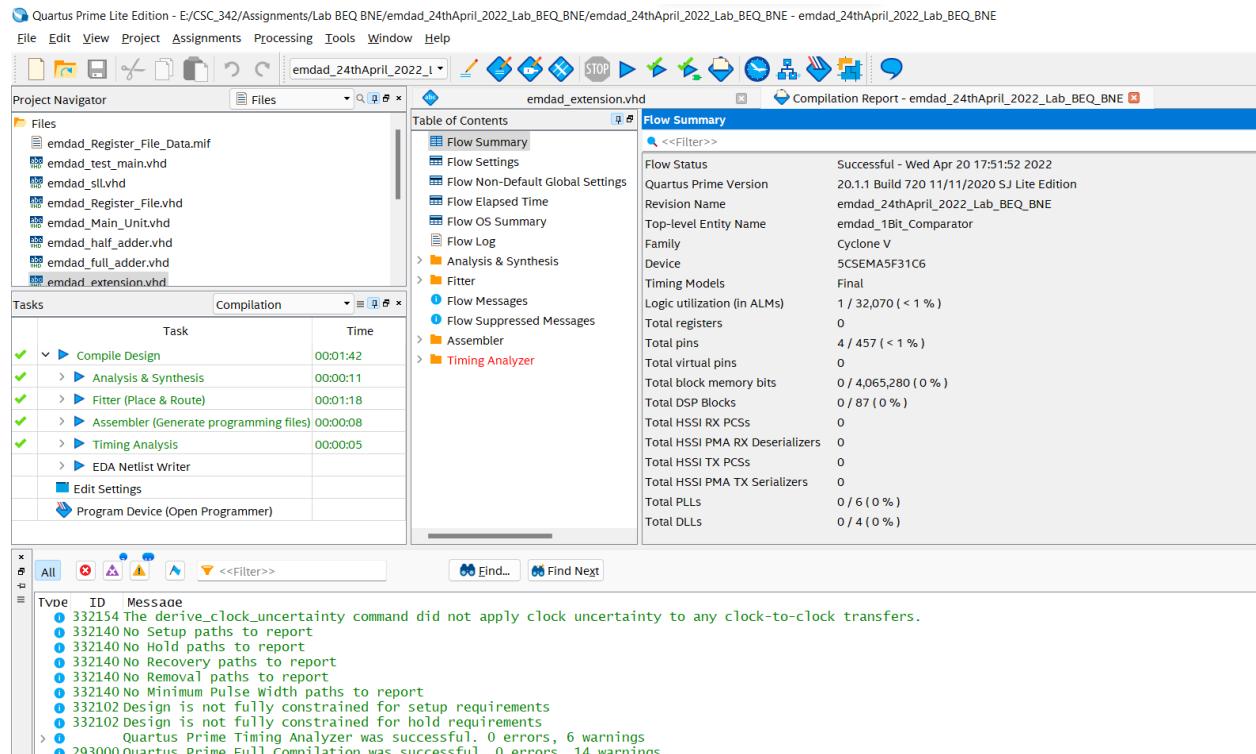
entity emdad_extension is
    port (emdad_imm_input : in std_logic_vector (15 downto 0);
          emdad_Q : out std_logic_vector (31 downto 0));
end emdad_extension;

architecture arch of emdad_extension is
begin
    signal extended: std_logic_vector(31 downto 0);
    begin
        emdad_Q <= std_logic_vector(resize(signed(emdad_imm_input), extended'length));
    end arch;

```

Figure 24: vhdl code for sign extension

In figure 25, we can see that the VHDL code compiled successfully.



Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab BEQ BNE/emdad_24thApril_2022_Lab_BEQ_BNE/emdad_24thApril_2022_Lab_BEQ_BNE - emdad_24thApril_2022_Lab_BEQ_BNE

File Edit View Project Assignments Processing Tools Window Help

Project Navigator Files emdad_extension.vhd Compilation Report - emdad_24thApril_2022_Lab_BEQ_BNE

Table of Contents Flow Summary

Flow Status Successful - Wed Apr 20 17:51:52 2022

Quartus Prime Version 20.1 Build 720 11/11/2020 SJ Lite Edition

Revision Name emdad_1Bit_Comparator

Top-level Entity Name Cyclone V

Family 5CSEMAF31C6

Device Final

Timing Models

Logic utilization (in ALMs) 1 / 32,070 (< 1 %)

Total registers 0

Total pins 4 / 457 (< 1 %)

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 %)

Tasks Compilation

Task Time

✓ ▶ Compile Design 00:01:42

✓ > Analysis & Synthesis 00:00:11

✓ > Fitter (Place & Route) 00:01:18

✓ > Assembler (Generate programming files) 00:00:08

✓ > Timing Analysis 00:00:05

> EDA Netlist Writer

Edit Settings

Program Device (Open Programmer)

Message Log

Type ID Message

- 332154 The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
- 332140 No Setup paths to report
- 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
- Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
- 293000 Quartus Prime Full Compilation was successful. 0 errors, 14 warnings

Figure 25: Successful compilation report.

2.1 MUX:

In figure 17, we can see the VHDL code for the 2:1 mux. This code decides for output to return PC + 4 or PC + +SIGN EXTENDEC IMM.

```

Quartus Prime Lite Edition - E:/CSC_342/Assignments/Lab BEQ BNE/emdad_24thApril_2022_Lab_BEQ_BNE/emdad_24thApril_2022_Lab_BEQ_BNE - emdad_24thApril_2022_Lab_BEQ_BNE
File Edit View Project Assignments Processing Tools Window Help
Project Navigator Files emdad_2_1_MUX.vhd* Compilation Report - emdad_24thApril_2022_Lab_BEQ_BNE
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
1 ENTITY emdad_2_1_MUX IS
2 PORT(
3     emdad_4, emdad_4_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
4     emdad_cond : IN STD_LOGIC;
5     emdad_out : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
6 );
7 END emdad_2_1_MUX;
8
9 ARCHITECTURE arch OF emdad_2_1_MUX IS
10 BEGIN
11     emdad_out <= emdad_4 WHEN emdad_cond = '0' ELSE emdad_4_IMM;
12 END arch;

```

Figure 26: VHDL code for the 2:1 mux

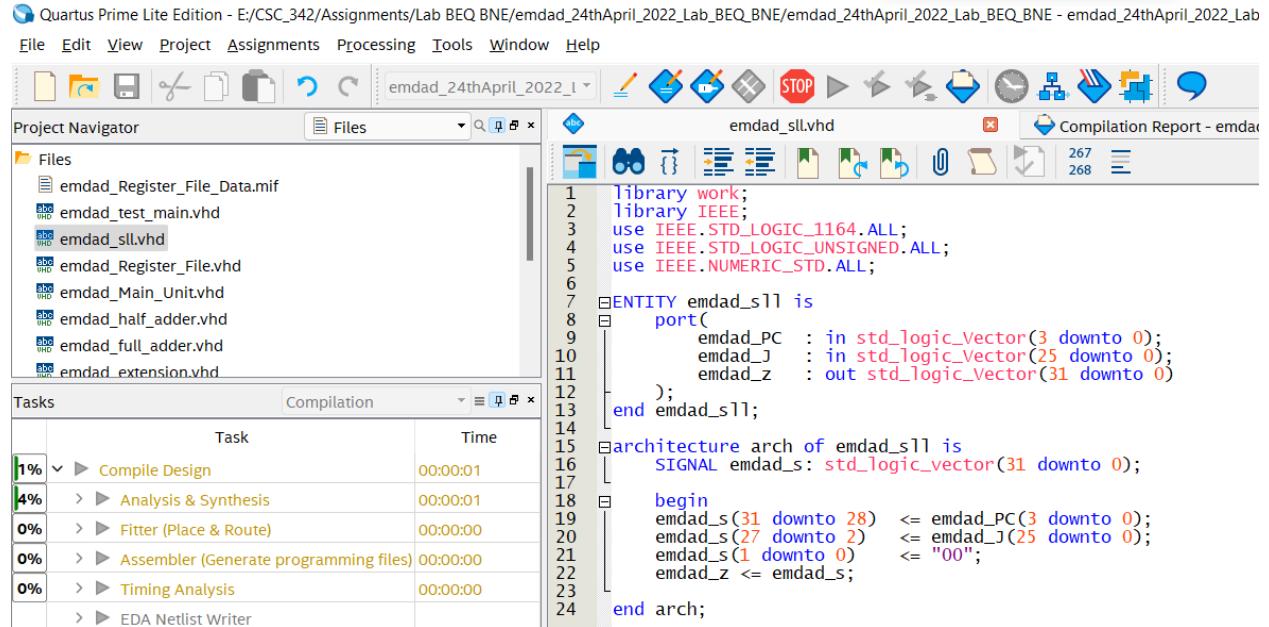
In figure 27, we can see that the VHDL code compiled successfully.

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

Figure 27: Successful compilation report.

Jump Calculation:

In figure 28, we can see the VHDL code for jump calculation. This code requires the input of the 4 most significant bits of PC and 26 least significant bits.



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 toolbar contains various icons for file operations and design tools. The Project Navigator on the left lists files such as emdad_Register_File_Data.mif, emdad_test_main.vhd, emdad_sll.vhd, emdad_Register_File.vhd, emdad_Main_Unit.vhd, emdad_half_adder.vhd, emdad_full_adder.vhd, and emdad_extension.vhd. The main workspace displays the VHDL code for the entity emdad_s11:

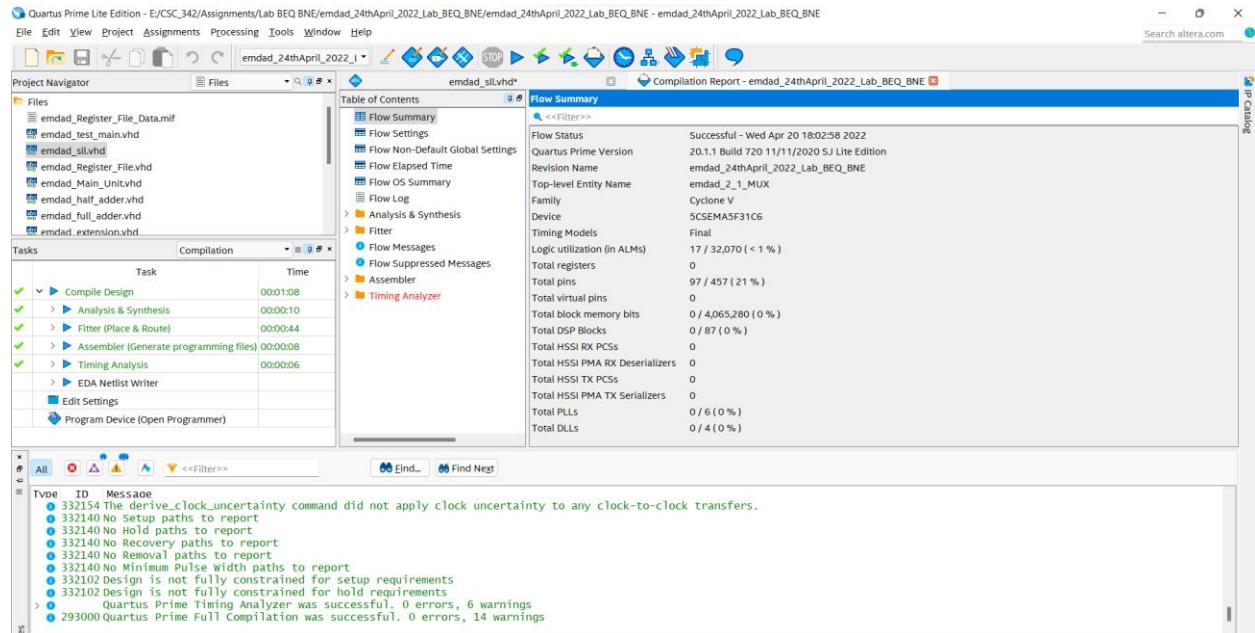
```

1 library work;
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5 use IEEE.NUMERIC_STD.ALL;
6
7 ENTITY emdad_s11 IS
8 PORT(
9   emdad_PC : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
10  emdad_J : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
11  emdad_z : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
12 );
13 END emdad_s11;
14
15 ARCHITECTURE arch OF emdad_s11 IS
16 SIGNAL emdad_s : STD_LOGIC_VECTOR(31 DOWNTO 0);
17
18 BEGIN
19   emdad_s(31 DOWNTO 28) <= emdad_PC(3 DOWNTO 0);
20   emdad_s(27 DOWNTO 2) <= emdad_J(25 DOWNTO 0);
21   emdad_s(1 DOWNTO 0) <= "00";
22   emdad_z <= emdad_s;
23
24 END arch;

```

Figure 28: VHDL code for jump calculation

In figure 29, we can see that the VHDL code compiled successfully.



The screenshot shows the Quartus Prime Lite Edition interface after compilation. The top menu bar and toolbar are identical to Figure 28. The Project Navigator and main workspace are also similar. The compilation report is displayed in the center-right area. The 'Flow Summary' section shows the following details:

Flow Status	Successful - Wed Apr 20 18:02:58 2022
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	emdad_24thApril_2022_Lab_BEQ_BNE
Top-level Entity Name	emdad_2_1_MUX
Family	Cyclone V
Device	5CSEMAF31C6
Timing Model	Final
Logic utilization (in ALMs)	17 / 32,070 (< 1 %)
Total registers	0
Total pins	97 / 457 (21 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCGs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCGs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

The 'Messages' section at the bottom lists several warnings, including:

- 332154 The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
- 332140 No Setup paths to report
- 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 29: Successful compilation report.

Controller:

In figure 30, we can see the VHDL code for the controller. It designates the input date for the 32bit PC register, it selects the right output based on the OP code from instruction register.

```

library work;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

ENTITY emdad_controller IS
    PORT(
        emdad_J : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        emdad_BEQBNE : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        emdad_OP : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
        emdad_nextInstruction : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
END ENTITY emdad_controller;

ARCHITECTURE arch OF emdad_controller IS
    SIGNAL emdad_s : STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
    BEGIN
        emdad_s <= emdad_J WHEN emdad_OP = "000010";
        ELSE
            emdad_BEQBNE WHEN emdad_OP = "000100";
            ELSE
                emdad_BEQBNE WHEN emdad_OP = "000101";
                emdad_nextInstruction <= emdad_s;
    END;
END arch;

```

Figure 30: VHDL code for the controller

In figure 31, we can see that the VHDL code compiled successfully.

Type	ID	Message
332154	The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.	
332240	No setup paths to report	
332240	No hold paths to report	
332240	No recovery paths to report	
332240	No removal paths to report	
332240	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 31: Successful compilation report.

Main Unit:

In figure 32, we can see the VHDL code for main unit, has other VHDL codes as component.

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 ENTITY emdad_Main_unit IS
6     PORT(
7         emdad_instruction : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
8         emdad_clk : IN STD_LOGIC;
9         emdad_rden : IN STD_LOGIC;
10        emdad_rden : IN STD_LOGIC;
11        emdad_pc_chen : IN STD_LOGIC;
12        emdad_nextInstruction : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
13    );
14 end emdad_Main_unit;
15
16
17 ARCHITECTURE arch OF emdad_Main_unit IS
18     COMPONENT emdad_32_bit_register
19     PORT(
20         emdad_clk, emdad_wren, emdad_rden, emdad_chen : IN STD_LOGIC;
21         emdad_data : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
22         emdad_q : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
23     );
24     END COMPONENT;
25
26     COMPONENT emdad_Register_File
27     PORT(
28         emdad_address_a : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
29         emdad_address_b : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
30         emdad_wren_a : IN STD_LOGIC;
31         emdad_data_a : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
32         emdad_wren_b : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
33         emdad_wren_a : IN STD_LOGIC := '0';
34         emdad_wren_b : IN STD_LOGIC := '0';
35         emdad_q_a : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
36         emdad_q_b : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
37     );
38     END COMPONENT;
39
40     COMPONENT emdad_equal
41     PORT(emdad_RS, emdad_RT: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
42         emdad_OP: IN STD_LOGIC_VECTOR(5 DOWNTO 0);
43         emdad_Cond : OUT STD_LOGIC);
44     END COMPONENT;
45
46     COMPONENT emdad_controller
47     PORT(emdad_RS, emdad_RT: IN STD_LOGIC_VECTOR(31 DOWNTO 0);
48         emdad_OP: IN STD_LOGIC_VECTOR(5 DOWNTO 0);
49         emdad_Cond : OUT STD_LOGIC);
50     END COMPONENT;
51

```

Figure 32: VHDL code for main unit (part 1)

```

52
53     COMPONENT emdad_s11
54     PORT(
55         emdad_PC : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
56         emdad_J : IN STD_LOGIC_VECTOR(25 DOWNTO 0);
57         emdad_Z : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
58     );
59     END COMPONENT;
60
61     COMPONENT emdad_controller
62     PORT(
63         emdad_J : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
64         emdad_BEQBNE : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
65         emdad_OP : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
66         emdad_nextInstruction : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
67     );
68     END COMPONENT;
69
70     COMPONENT emdad_32_bit_adder
71     GENERIC(N: INTEGER := 32);
72     PORT(emdad_X, emdad_Y : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
73         emdad_Cin : IN STD_LOGIC;
74         emdad_S : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0);
75         emdad_Cout : OUT STD_LOGIC);
76     END COMPONENT;
77
78     COMPONENT emdad_2_1_MUX
79     PORT(
80         emdad_4, emdad_4_IMM : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
81         emdad_cond : IN STD_LOGIC;
82         emdad_out : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
83     );
84     END COMPONENT;
85
86     -- PC DATA
87     SIGNAL emdad_PCINOUT : STD_LOGIC_VECTOR(31 DOWNTO 0);
88     SIGNAL emdad_PC_MUX_output : STD_LOGIC_VECTOR(31 DOWNTO 0);
89
90
91     -- OPCODE RT RS REGISTER DATA
92     SIGNAL emdad_instructionRegister_out : STD_LOGIC_VECTOR(31 DOWNTO 0);
93     SIGNAL emdad_OPCODE : STD_LOGIC_VECTOR(5 DOWNTO 0);
94     SIGNAL emdad_RSADDRESS : STD_LOGIC_VECTOR(4 DOWNTO 0);
95     SIGNAL emdad_RTADDRESS : STD_LOGIC_VECTOR(4 DOWNTO 0);
96     SIGNAL emdad_RDDATA : STD_LOGIC_VECTOR(31 DOWNTO 0);
97     SIGNAL emdad_PC_HIGHER_ORDER : STD_LOGIC_VECTOR(25 DOWNTO 0);
98     SIGNAL emdad_JUMP : STD_LOGIC_VECTOR(25 DOWNTO 0);
99     SIGNAL emdad_IMM_EXTENDED : STD_LOGIC_VECTOR(31 DOWNTO 0);
100    SIGNAL emdad_RSREGISTERFILE_OUT : STD_LOGIC_VECTOR(31 DOWNTO 0);
101    SIGNAL emdad_RTREGISTERFILE_OUT : STD_LOGIC_VECTOR(31 DOWNTO 0);
102
103    -- ADDING LOGIC SIGNALS
104    SIGNAL emdad_ADD_4 : STD_LOGIC_VECTOR(31 DOWNTO 0);
105    SIGNAL emdad_ADD_IMM : STD_LOGIC_VECTOR(31 DOWNTO 0);

```

Figure 33: VHDL code for main unit (part 2)

```

107 -- COMPARATOR SIGNALS
108 SIGNAL emdad_CONDITIONAL : STD_logic;
109
110 -- JUMP SIGNAL
111 SIGNAL emdad_JUMP_INSTRUCTION : STD_LOGIC_VECTOR (31 DOWNTO 0);
112
113 -- CONTROLLER_OUTPUT
114 SIGNAL emdad_Controller_out : STD_LOGIC_VECTOR (31 DOWNTO 0);
115 SIGNAL emdad_temp_pcinout : STD_LOGIC_VECTOR (31 DOWNTO 0);
116
117 BEGIN
118
119     emdad_temp_pcinout <= x"00000000" WHEN emdad_pc_chen = '0' ELSE emdad_pcinout;
120
121     emdad_OPCODE <= emdad_InstructionRegister_out (31 downto 26);
122     emdad_RSADDRESS <= emdad_InstructionRegister_out (25 downto 21);
123     emdad_RTADDRESS <= emdad_InstructionRegister_out (20 downto 16);
124     emdad_data <= emdad_InstructionRegister_out (15 downto 0);
125     emdad_JUMP <= emdad_InstructionRegister_out (25 downto 0);
126     emdad_PC_HIGHER_ORDER <= "0000" when emdad_pc_chen='0' ELSE emdad_pcinout(31 downto 28);
127
128     F1: emdad_32_bit_register port map (
129         emdad_c1k => emdad_c1k,
130         emdad_wren => emdad_wren,
131         emdad_rden => emdad_rden,
132         emdad_chen => emdad_chen,
133         emdad_data => emdad_instruction,
134         emdad_q => emdad_InstructionRegister_out
135     );
136
137     F2: emdad_Register_File port map(
138         emdad_clock => emdad_c1k,
139         emdad_address_a => emdad_RSADDRESS,
140         emdad_address_b => emdad_RTADDRESS,
141         emdad_data_a => x"00000000",
142         emdad_data_b => x"00000000",
143         emdad_wren_a => '0',
144         emdad_wren_b => '0',
145         emdad_q_a => emdad_RSREGISTERFILE_OUT,
146         emdad_q_b => emdad_RTREGISTERFILE_OUT
147     );
148
149     F3: emdad_equal port map(
150         emdad_RS => emdad_RSREGISTERFILE_OUT,
151         emdad_RT => emdad_RTREGISTERFILE_OUT,
152         emdad_OP => emdad_OPCODE,
153         emdad_Cond => emdad_CONDITIONAL
154     );
155
156     F4: emdad_extension port map(
157         emdad_IMM_input=> emdad_IMM_DATA,
158         emdad_Q => emdad_IMM_EXTENDED
159     );

```

Figure 34: VHDL code for main unit (part 3)

```

160
161     F5: emdad_32_bit_adder port map(
162         emdad_x => emdad_temp_pcinout,
163         emdad_y => x"00000004",
164         emdad_cin => '0',
165         emdad_s => emdad_ADD_4
166     );
167
168     F6: emdad_sll port map(
169         emdad_PC => emdad_PC_HIGHER_ORDER,
170         emdad_J => emdad_JUMP,
171         emdad_Z => emdad_JUMP_INSTRUCTION
172     );
173
174     F7: emdad_32_bit_adder port map(
175         emdad_x => emdad_ADD_4,
176         emdad_y => emdad_IMM_EXTENDED,
177         emdad_cin => '0',
178         emdad_s => emdad_ADD_IMM
179     );
180
181     F8: emdad_2_1_MUX port map(
182         emdad_4 => emdad_ADD_4,
183         emdad_4_IMM => emdad_ADD_IMM,
184         emdad_cond => emdad_CONDITIONAL,
185         emdad_out => emdad_PC_MUX_output
186     );
187
188     F9: emdad_controller port map(
189         emdad_in => emdad_INSTRUCTION,
190         emdad_BNQ => emdad_PC_MUX_output,
191         emdad_OP => emdad_OPCODE,
192         emdad_nextInstruction => emdad_Controller_out
193     );
194
195     F10: emdad_32_bit_register port map(
196         emdad_c1k => emdad_c1k,
197         emdad_wren => emdad_wren,
198         emdad_rden => emdad_rden,
199         emdad_chen => emdad_chen,
200         emdad_data => emdad_controller_out,
201         emdad_q => emdad_PCInout
202     );
203
204     emdad_nextInstruction <= emdad_PCInout;
205
206 end arch;

```

Figure 35: VHDL code for main unit (part 4)

F1 is 32-bit register, stores the instructions from input and output is instructionRegister_Out. F2 is register file component. The RSADDRESS and RTADDRESS receive address date from 32-bit instruction register and output 32bit values. The address data loaded from .mif file. F3 is comparator component output the condition based on the OPCODE BEQ or BNE. F4 is the extension component to compute the 32-bit extension of IMM field to store from the instructions register. F5 is the 32bit add/sub and add the PC register output with 4. F6 is jump calculation and it takes higher order bits and 26 least significant bits. F7 is takes the output from F5 and extend it from F4. F9 is controller to decide which input to select for 32-bit register. F10 is 32-bit register.

In figure 36, we can see that the VHDL code compiled successfully.

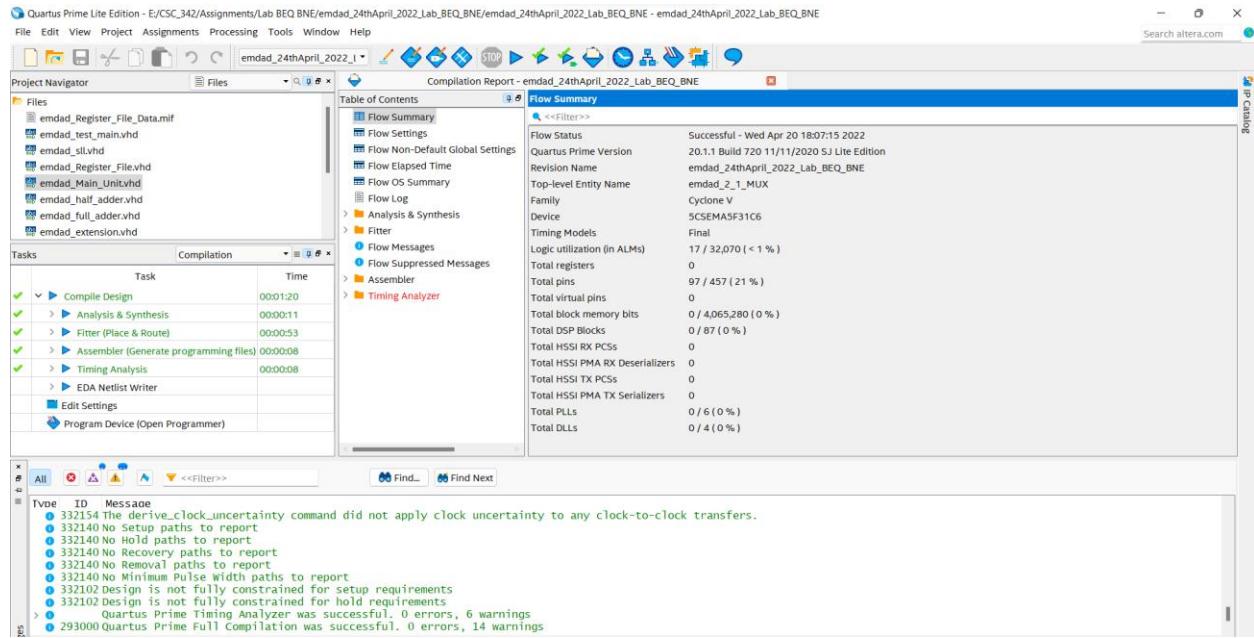


Figure 36: Successful compilation report.

Testbench:

In figure 37, we can see the VHDL code for testbench. It is to get waves on ModelSim and test my code validation.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY emdad_testbench IS
5  END emdad_testbench;
6
7  ARCHITECTURE arch OF emdad_testbench IS
8
9  BEGIN
10    component emdad_Main_unit is
11      port(
12        emdad_instruction : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
13        emdad_clk : IN STD_LOGIC;
14        emdad_wren : IN STD_LOGIC;
15        emdad_rden : IN STD_LOGIC;
16        emdad_pc_chen : IN STD_LOGIC;
17        emdad_nextInstruction : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
18      );
19    END component;
20
21    SIGNAL emdad_instruction : STD_LOGIC_VECTOR(31 DOWNTO 0);
22    SIGNAL emdad_clk : STD_LOGIC;
23    SIGNAL emdad_wren : STD_LOGIC;
24    SIGNAL emdad_rden : STD_LOGIC;
25    SIGNAL emdad_pc_chen : STD_LOGIC;
26    SIGNAL emdad_nextInstruction : STD_LOGIC_VECTOR(31 DOWNTO 0);
27
28    begin
29      process
30      begin
31        emdad_instruction <= "00010010001101000000000011111111";
32        emdad_clk <= '1';
33        wait for 50 ps;
34        emdad_clk <= '0';
35        wait for 50 ps;
36      end process;
37
38      process
39      begin
40        emdad_wren <= '0';
41        wait for 100 ps;
42        emdad_wren <= '1';
43        wait for 100 ps;
44      end process;
45
46      process
47      begin
48        emdad_pc_chen <= '0';
49        wait for 600 ps;
50        emdad_pc_chen <= '1';
51        wait;
52      end process;

```

Figure 37: VHDL code for testbench (Part 1)

```

53
54 process
55 begin
56   emdad_rden <= '0';
57   wait for 200 ps;
58   emdad_rden <= '1';
59   wait for 200 ps;
60 end process;
61
62 F1: emdad_Main_Unit port map(
63   emdad_instruction => emdad_instruction,
64   emdad_Clk => emdad_Clk,
65   emdad_wren => emdad_wren,
66   emdad_rden => emdad_rden,
67   emdad_pc_chen => emdad_pc_chen,
68   emdad_nextInstruction => emdad_nextInstruction
69 );
70 end arch;

```

Figure 38: VHDL code for testbench (Part 2)

In figure 39, we can see that the VHDL code compiled successfully.

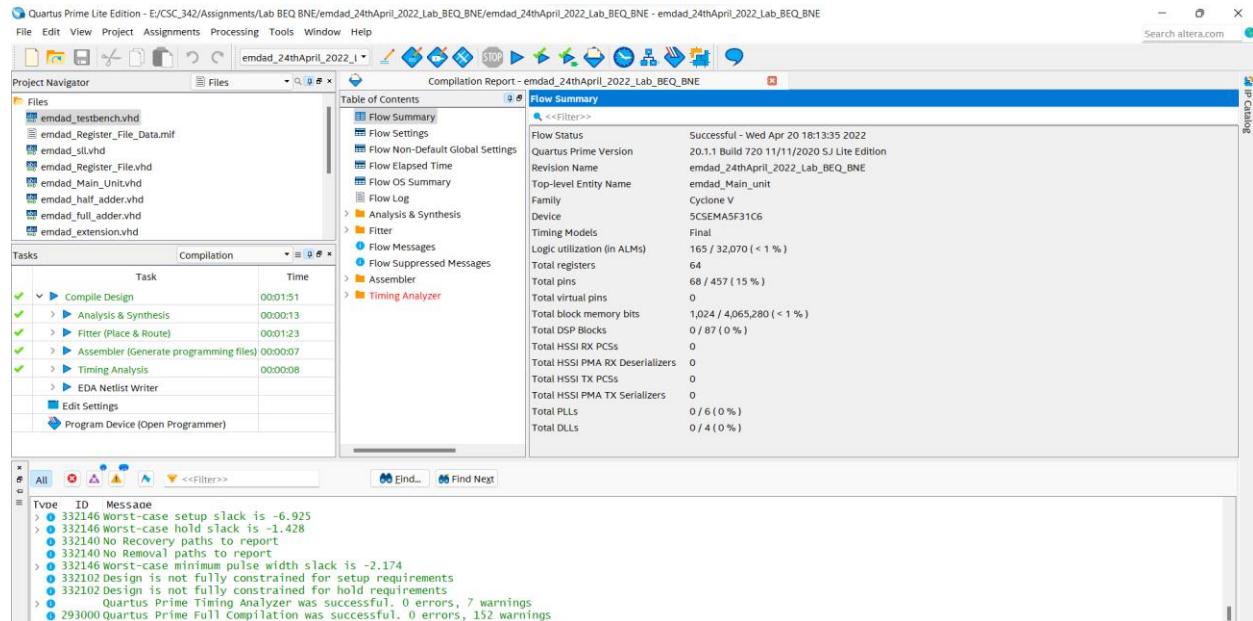


Figure 39: Successful compilation report.

Simulation:

In figure 40, we can see the ModelSim project directory for lab project and VHDL codes compiled successfully.

The screenshot shows the ModelSim interface. The top window is titled "ModelSim - INTEL FPGA STARTER EDITION 2020.1". The menu bar includes File, Edit, View, Compile, Simulate, Add, Project, Tools, Layout, Bookmarks, Window, and Help. Below the menu is a toolbar with various icons. The main area displays a table of files in the "Project" window, and the bottom area shows the "Transcript" window with compilation logs.

Name	Status	Type	Or	Modified
emdad_test_main.vhd	✓	VHDL	13	04/20/2022 04:58:16 ...
emdad_Main_Unit.vhd	✓	VHDL	12	04/20/2022 04:58:16 ...
emdad_sll.vhd	✓	VHDL	11	04/20/2022 04:58:16 ...
emdad_controller.vhd	✓	VHDL	10	04/20/2022 04:58:16 ...
emdad_2_1_MUX.vhd	✓	VHDL	9	04/20/2022 04:58:16 ...
emdad_extension.vhd	✓	VHDL	8	04/20/2022 04:58:16 ...
emdad_equal.vhd	✓	VHDL	7	04/20/2022 04:58:16 ...
emdad_1Bit_Comparator.vhd	✓	VHDL	6	04/20/2022 04:58:16 ...
emdad_Register_File.vhd	✓	VHDL	5	04/20/2022 04:58:16 ...
emdad_32_bit_register.vhd	✓	VHDL	4	04/20/2022 04:58:16 ...
emdad_32_bit_adder.vhd	✓	VHDL	3	04/20/2022 04:58:16 ...
emdad_adder_package.vhd	✓	VHDL	2	04/20/2022 04:58:16 ...
emdad_full_adder.vhd	✓	VHDL	1	04/20/2022 04:58:16 ...
emdad_half_adder.vhd	✓	VHDL	0	04/20/2022 04:58:16 ...
emdad_Register_File_Data.mif	.mif ...	-	-	04/20/2022 04:58:16 ...

Transcript window content:

```
# Loading project emdad_24thApril_2022_Lab_BEQ_BNE
# Compile of emdad_half_adder.vhd was successful.
# Compile of emdad_full_adder.vhd was successful.
# Compile of emdad_adder_package.vhd was successful.
# Compile of emdad_32_bit_adder.vhd was successful.
# Compile of emdad_32_bit_register.vhd was successful.
# Compile of emdad_Register_File.vhd was successful.
# Compile of emdad_1Bit_Comparator.vhd was successful.
# Compile of emdad_equal.vhd was successful.
# Compile of emdad_extension.vhd was successful.
# Compile of emdad_2_1_MUX.vhd was successful.
# Compile of emdad_controller.vhd was successful.
# Compile of emdad_sll.vhd was successful.
# Compile of emdad_Main_Unit.vhd was successful.
# Compile of emdad_test_main.vhd was successful.
# 14 compiles, 0 failed with no errors.
```

Figure 40: ModelSim project directory

In figure 41, we can see the input entered, instruction is 12340FF in hexadecimal. I did add 4 in the add_4 and kept pc higher order 0. The pcinout is 0.

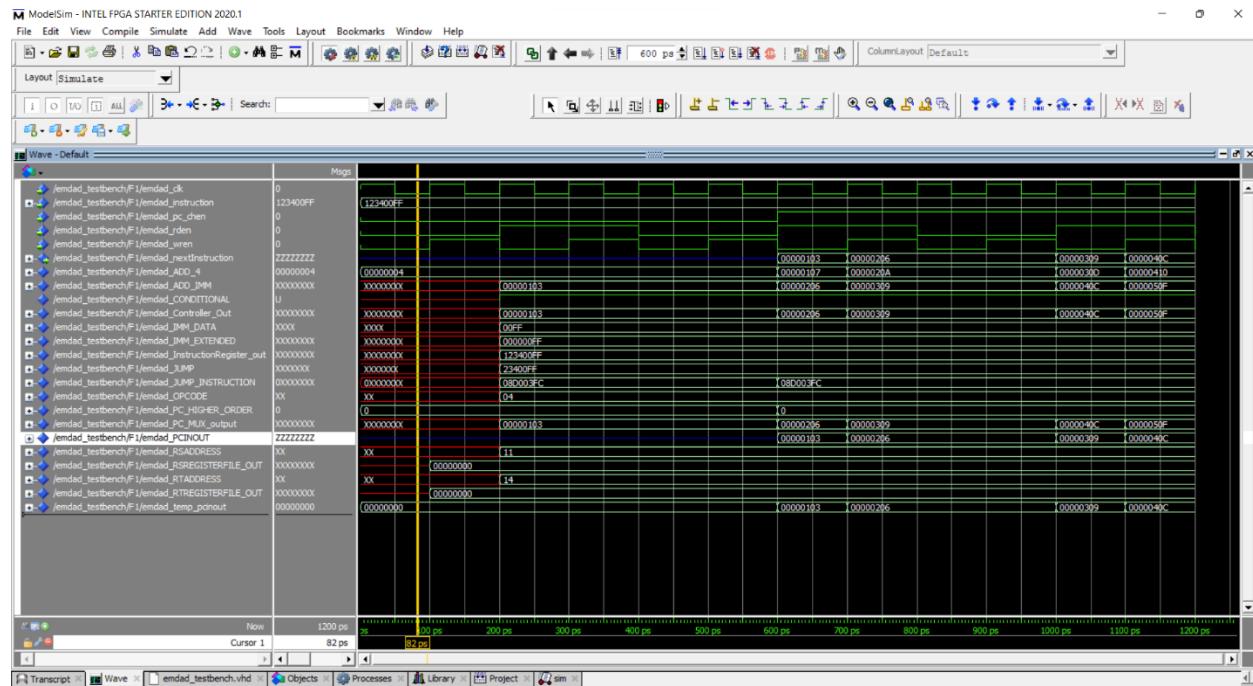


Figure 41: Simulation for inputs

In figure 42, we can see that everything stayed same, but RS and RT got the address input and giving output.

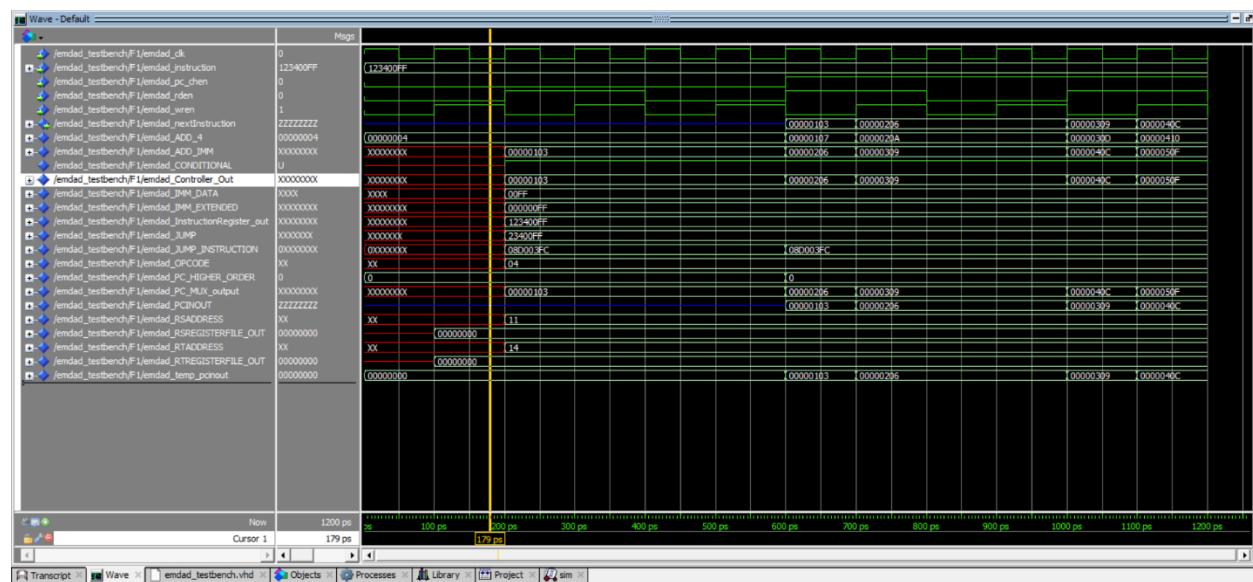


Figure 42: Simulation for RS and RT address output

In figure 43, we can see after the rising clock, both the add_imm address and controller out changes to 103, it is also same for PC_MUX_output. The RSADDRESS and RTADDRESS gave output of 11 and 14. The opcode is 04.

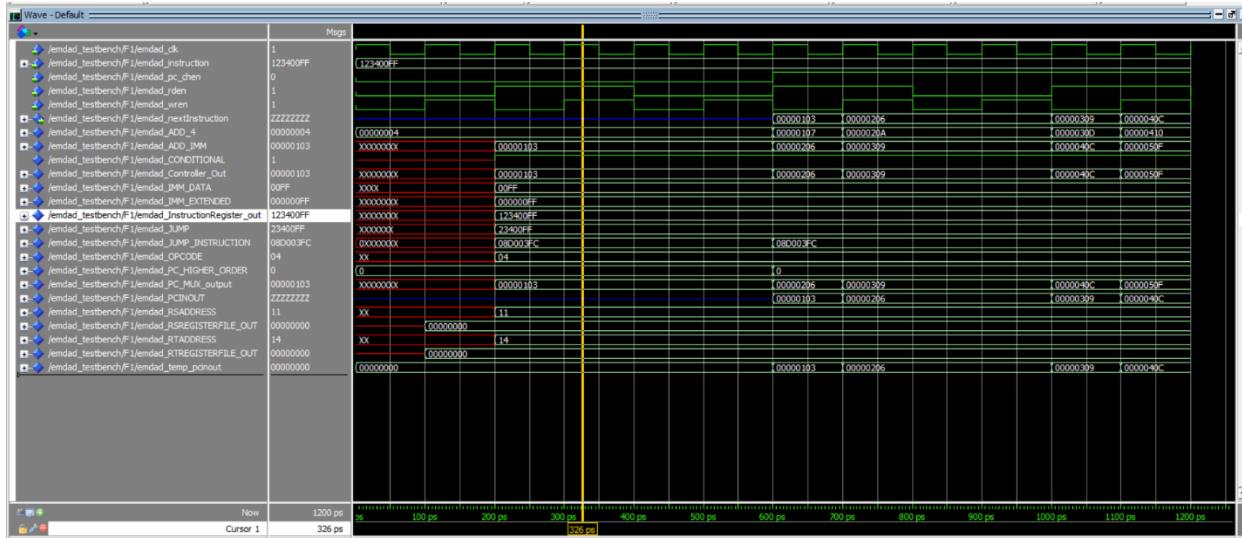


Figure 43: Simulation for RSADDRESS and RTADDRESS output

In figure 44, we can see that after rising clock, we can see the add/sub gave the output to 107 which produced from $103+4=107$. The ADD_IMM got double as well by extended.

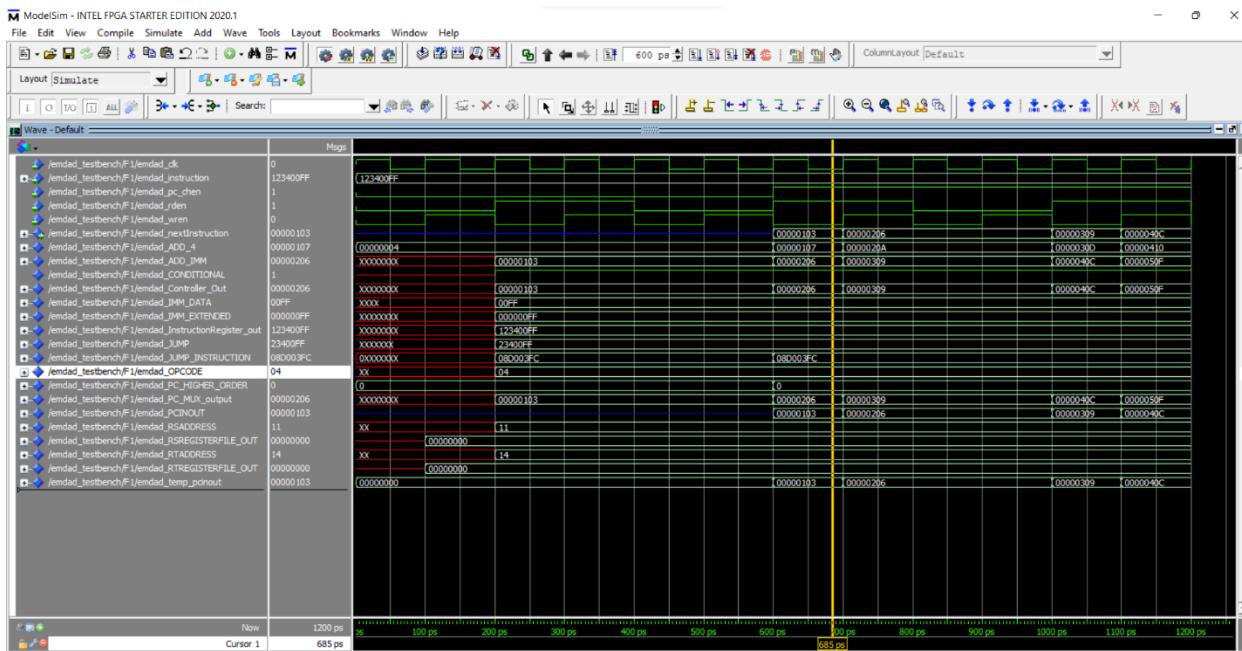


Figure 44: Simulation for add/sub

Conclusion:

In this assignment, I learned how to design and understand the specifications of BNE, BEW and J instruction. I was also able to implement it. It was a code writing and using the components right way was hard. The experiment taught me how to implement MIPS instructions in VHDL further. I relearned and reviewed the concept again and which will help me in the course further.