Midterm Lab Project

MdShahid Bin Emdad

March 23rd, 2022

CSC 34200/34300

Table of Contents

This is individual test project.

Please hand write and sign statements affirming that you will not cheat:

"I will neither give nor receive unauthorized assistance on this exam.

I will use only one computing device to perform this test"

Please hand write and sign here: 

Objective:

The objective of this assignment is to design and understand the specifications of VHDL array and LPM (library of parameterized modules) modules. In assignment 1, we were instructed to design 32-bit X 16 words data memory and then 32-bit X 32 words instruction memory by using LPM 1 port RAM modules and then use the same idea of instruction memory design 2-port RAM 32-bit X 32 words. In assignment 2, we were instructed to design 32-bit add/subunit from scratch and then design another version using LPM modules and finally design a circuit with flags. Overall, this lab is to learn how data is written and read from RAM.

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).

Assignment 1:

i. Data Memory: we used RAM-1 PORT 32x16.
ii. Instruction Memory: we used RAM-1 PORT 32x32.
iii. Dual Ported Memory Module: we used RAM-2 PORT 32x32

Assignment 2:

i. 32-bit add/sub scratch
ii. 32-bit add/sub LPM module
iii. Circuit Design using add/sub

MIF Data:

In figure 1, first I opened MARS and ran "***Fibonnacci.asm***" file.

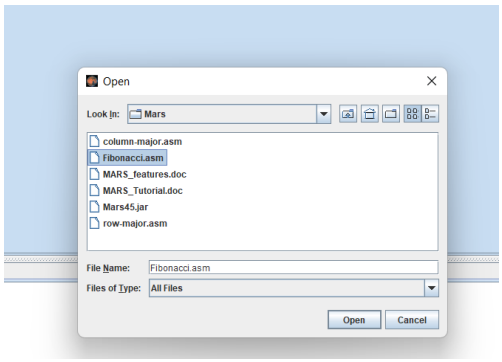

Figure 1: Fibonacci.asm file

In figure 2, we can see the execute result after running the file. From the code column, I collected first 5 inputs to use it in my MIF file.

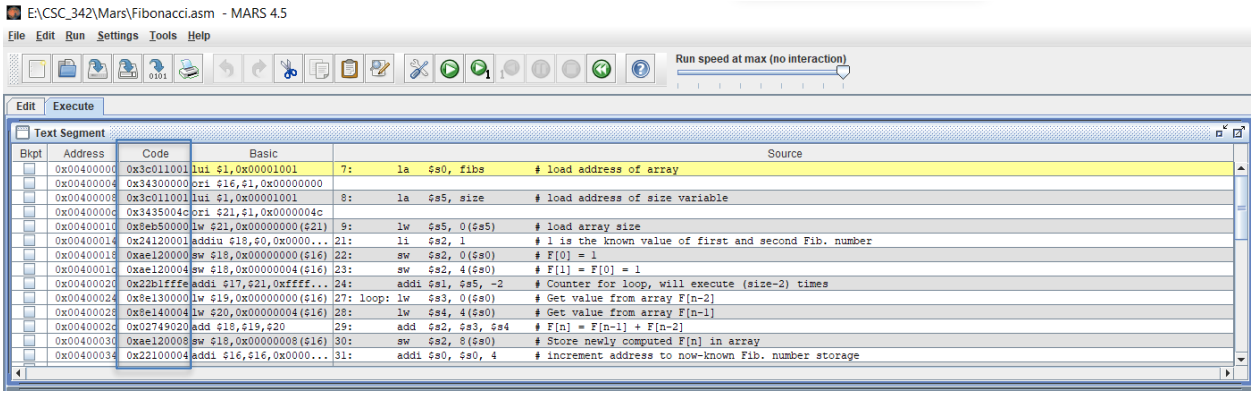

Figure 2: fibonnacci.asm execute output

Assignment 1:

i. Data Memory:

In figure 3, we can see the file directory for the lpm_ram_32x16.

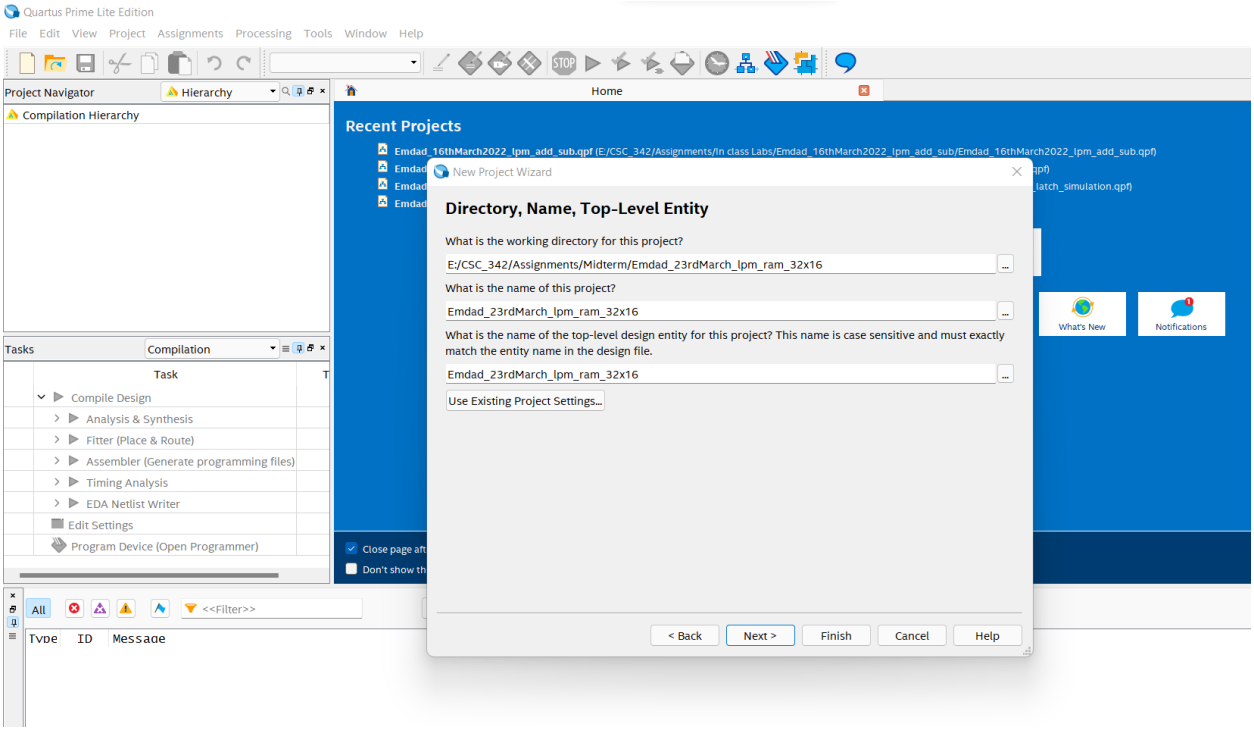

Figure 3: file directory for the lpm_ram_32x16

In figure 4, we can see the project summary for the lpm_ram_32x16.

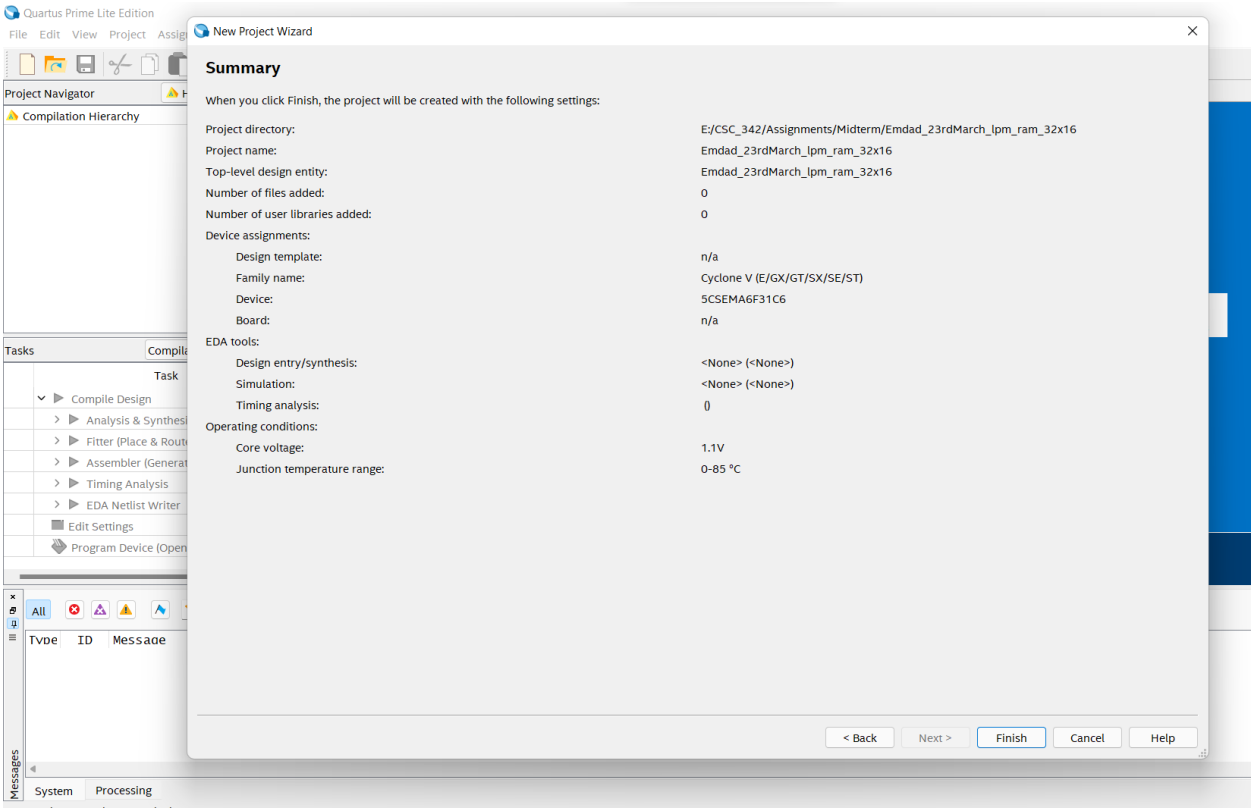

Figure 4: project summary for the lpm_ram_32x16.

In figure 5, we are creating the MIF file and input 16 words and 32 bit.

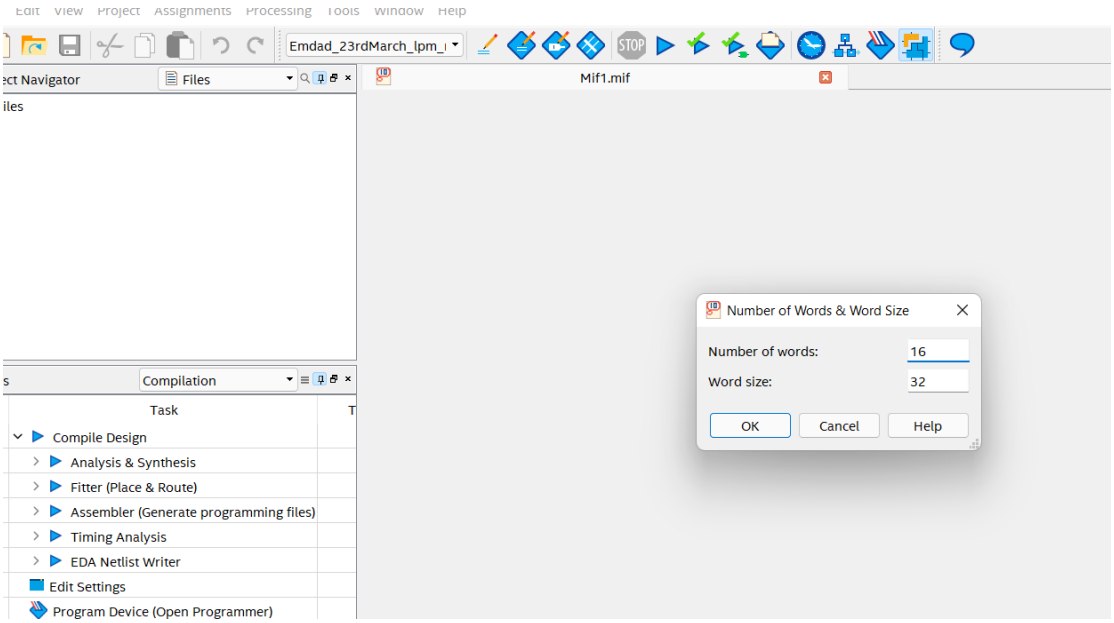

Figure 5: MIF input

In figure 6, I inserted the values for the MIF file to get the read output while I run waveform.

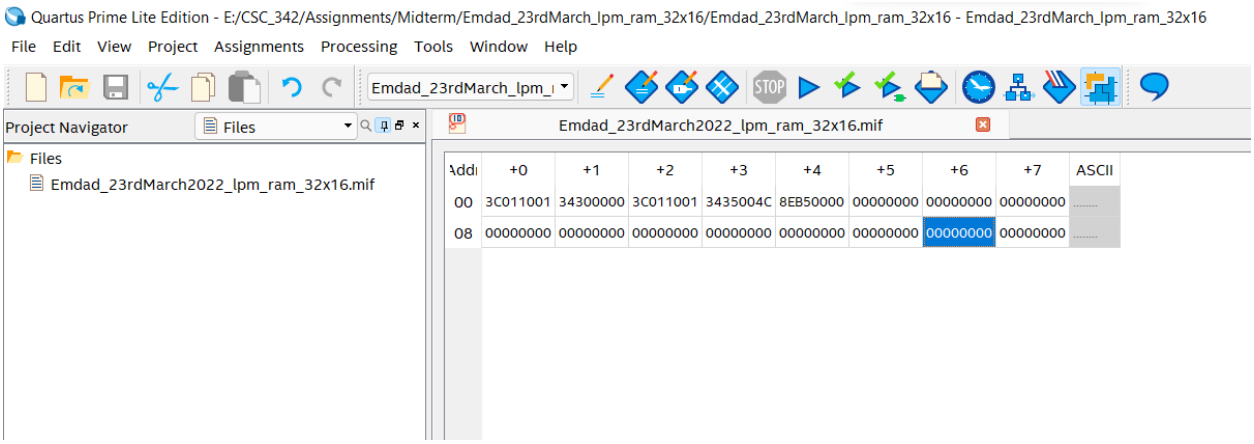

Figure 6: MIF file data

In figure 7, we are IP variation for the modules.

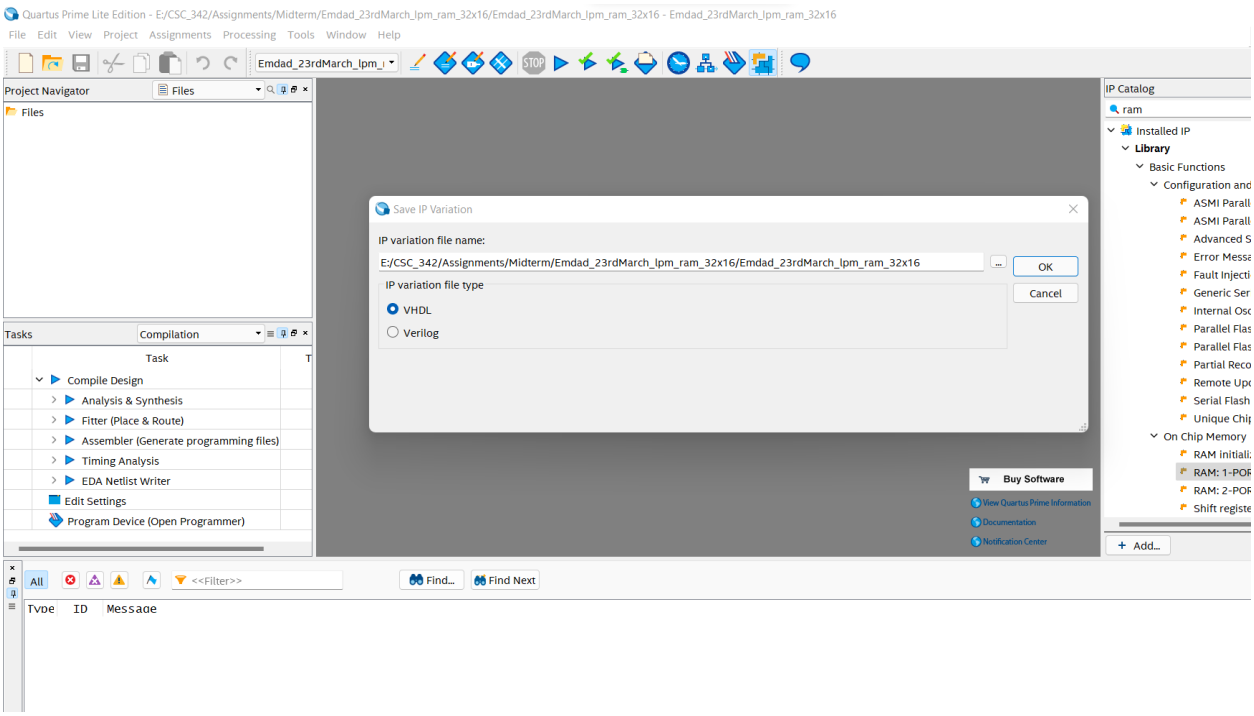

Figure 7: IP variation for the modules.

In figure 8, we selected 32 bits and 16 words and M10K for the memory block and then clicked next.

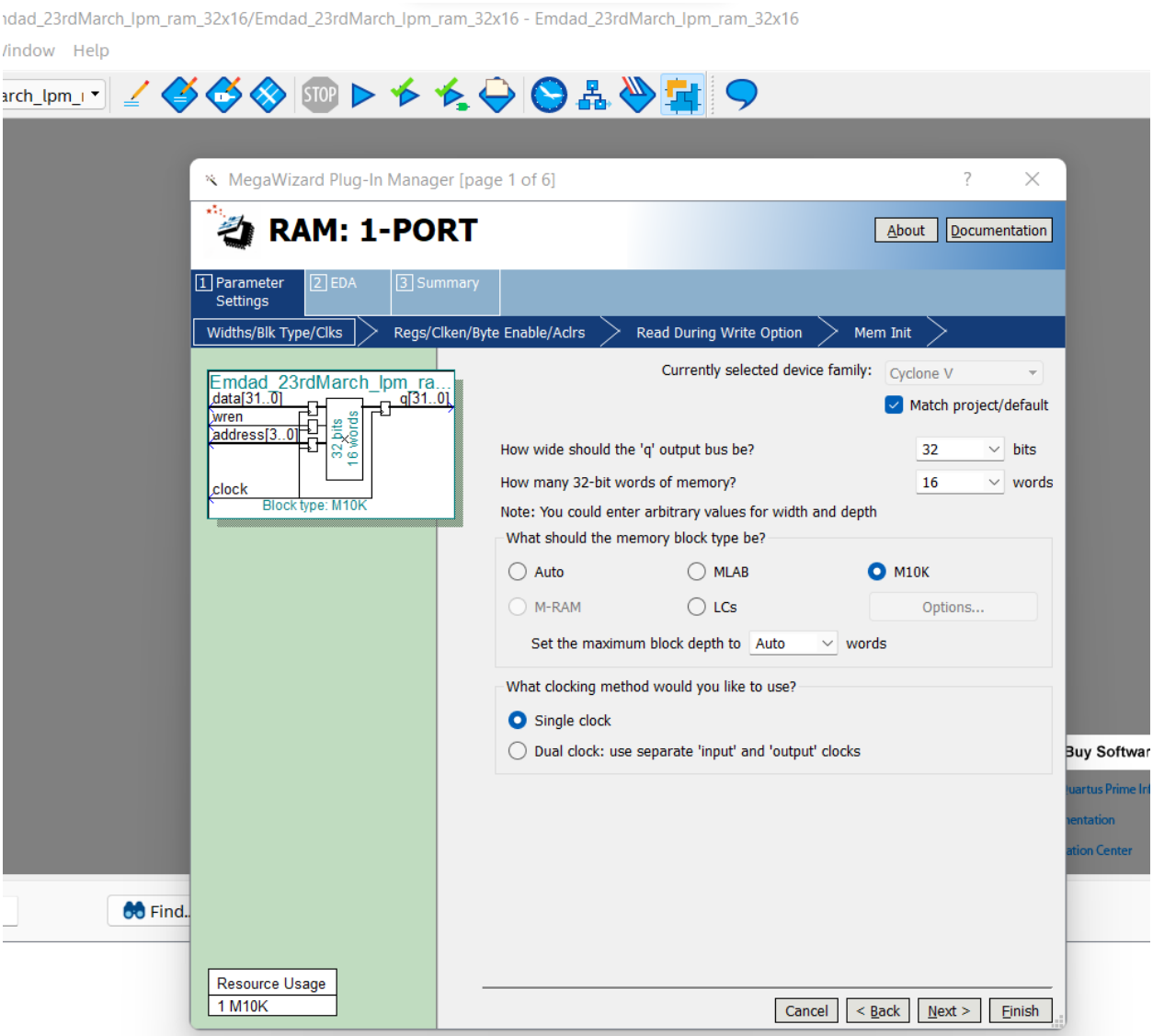

Figure 8: RAM-1 PORT configuration Part 1

In figure 9, I checked out the q output port.

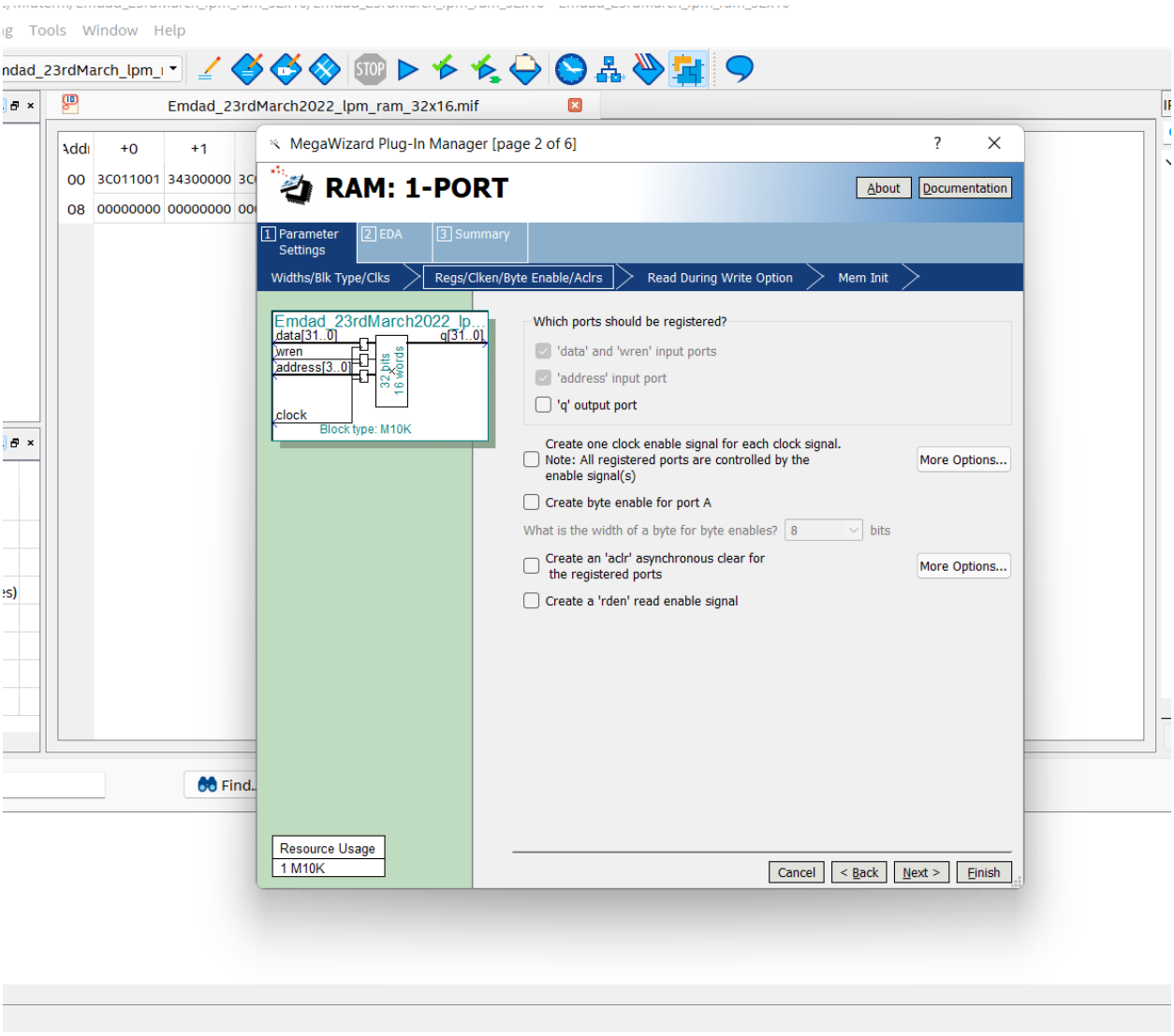

Figure 9: RAM-1 PORT configuration Part 2

In figure 10, we are selecting our created MIF file to read date from there for waveforms.

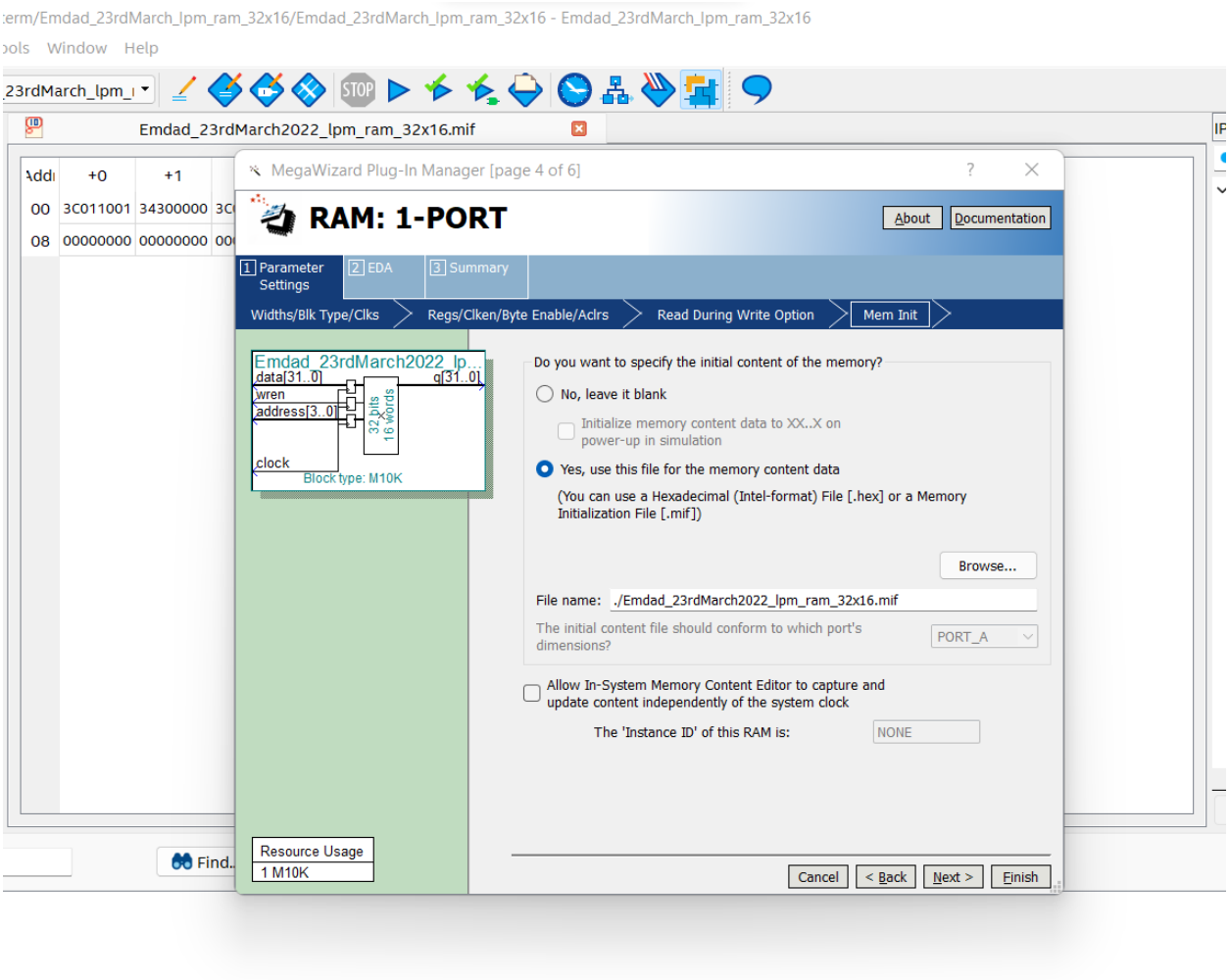

Figure 10: : RAM-1 PORT configuration Part 3

In figure 11, we can see the summary for the RAM-1 PORT.

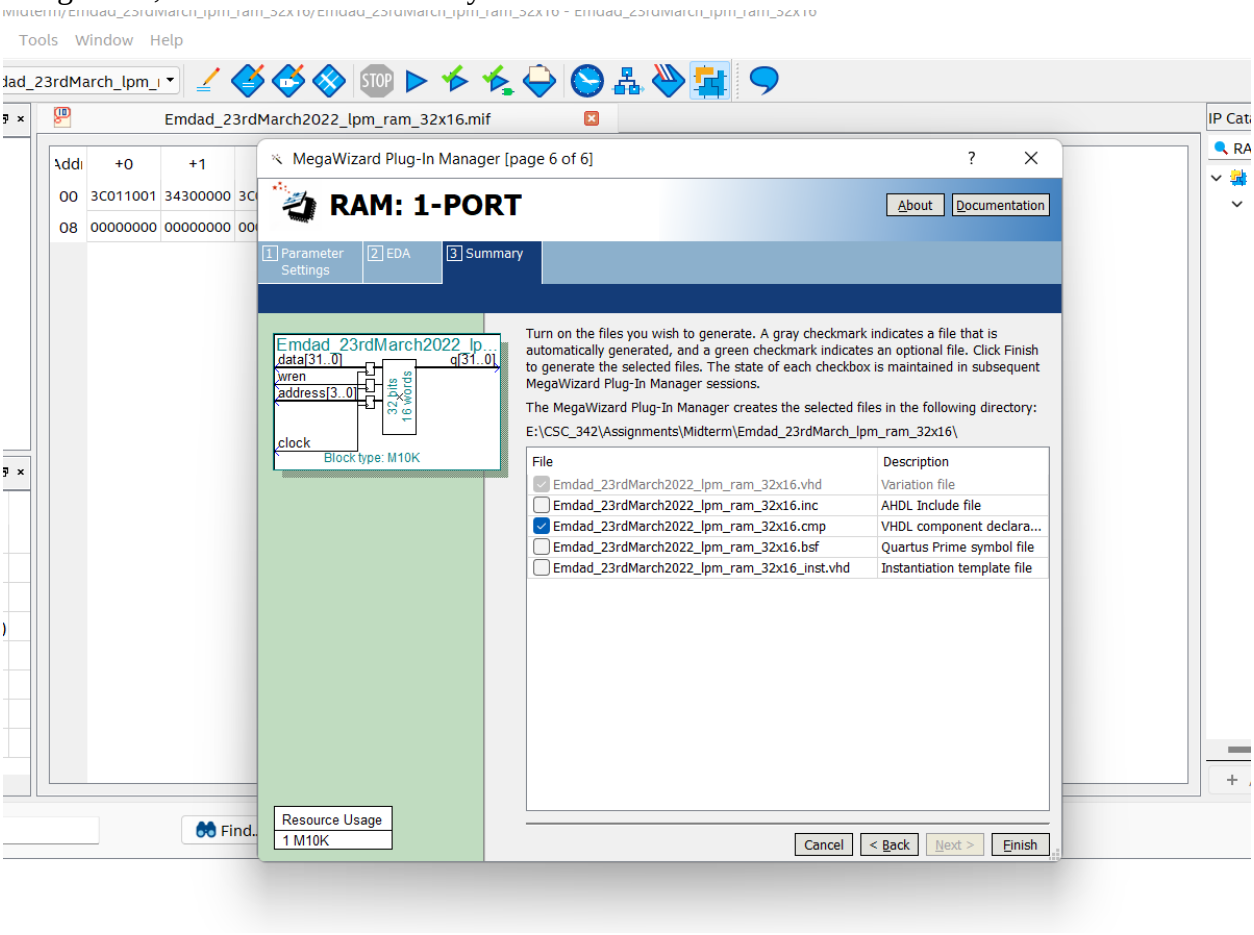

Figure 11: RAM-1 PORT summary

In figure 12, we can see the VHDL code manipulated from the lpm modules.

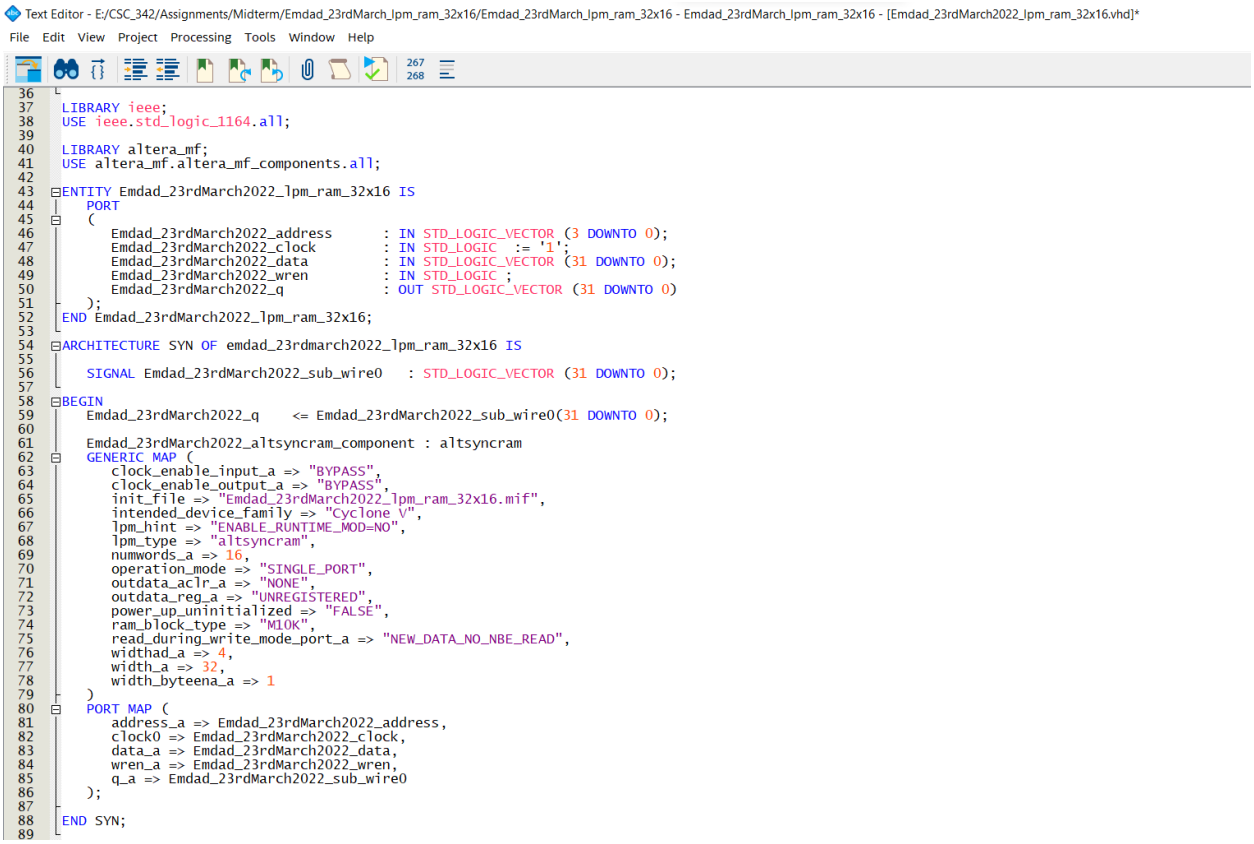

Figure 12: RAM-1 PORT code for 32x16

In figure 13, we can see that it compiled successfully.

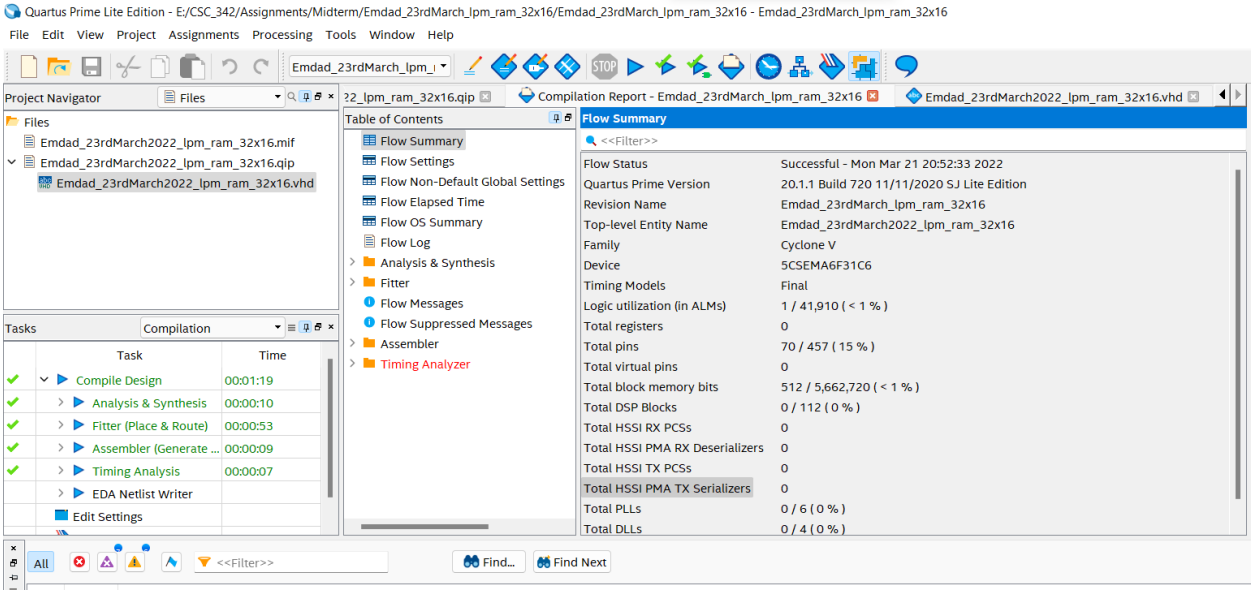

Figure 13: Compilation report

ii. Instruction Memory:

In figure 14, we can see the file directory for lpm RAM-1 PORT 32x32.

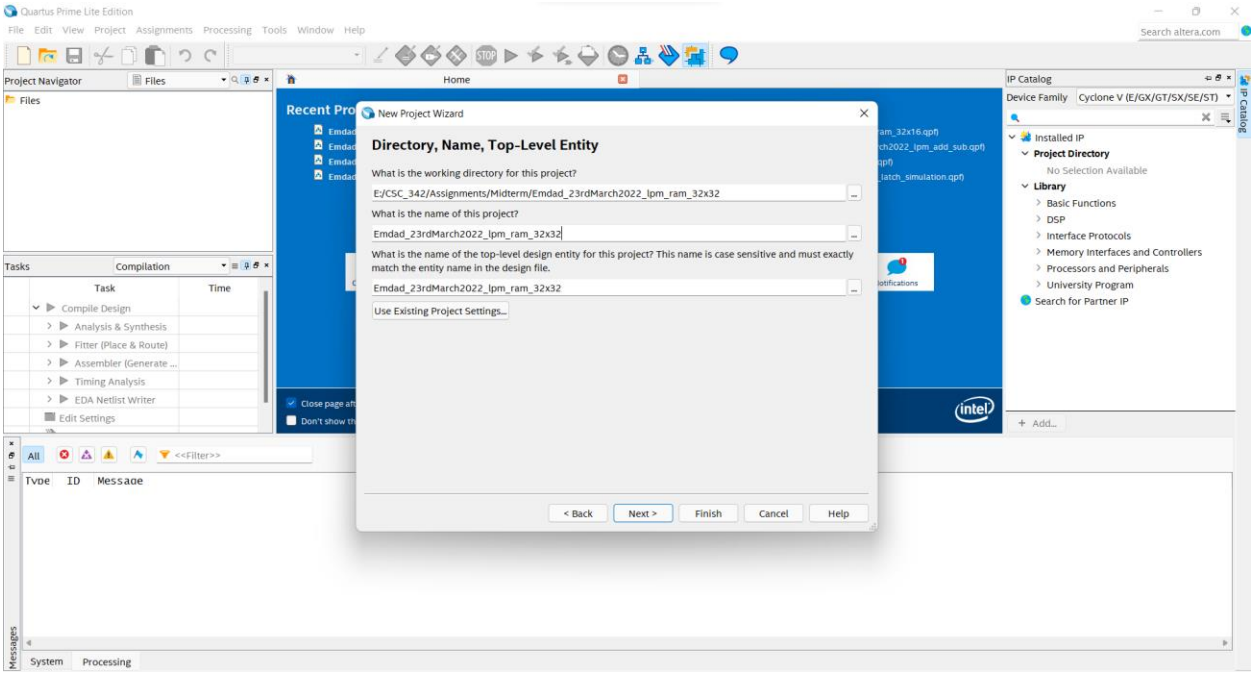

Figure 14: file directory for lpm RAM-1 PORT 32x32.

In figure 15, we can see the project summary for lpm RAM-1 PORT 32x32.

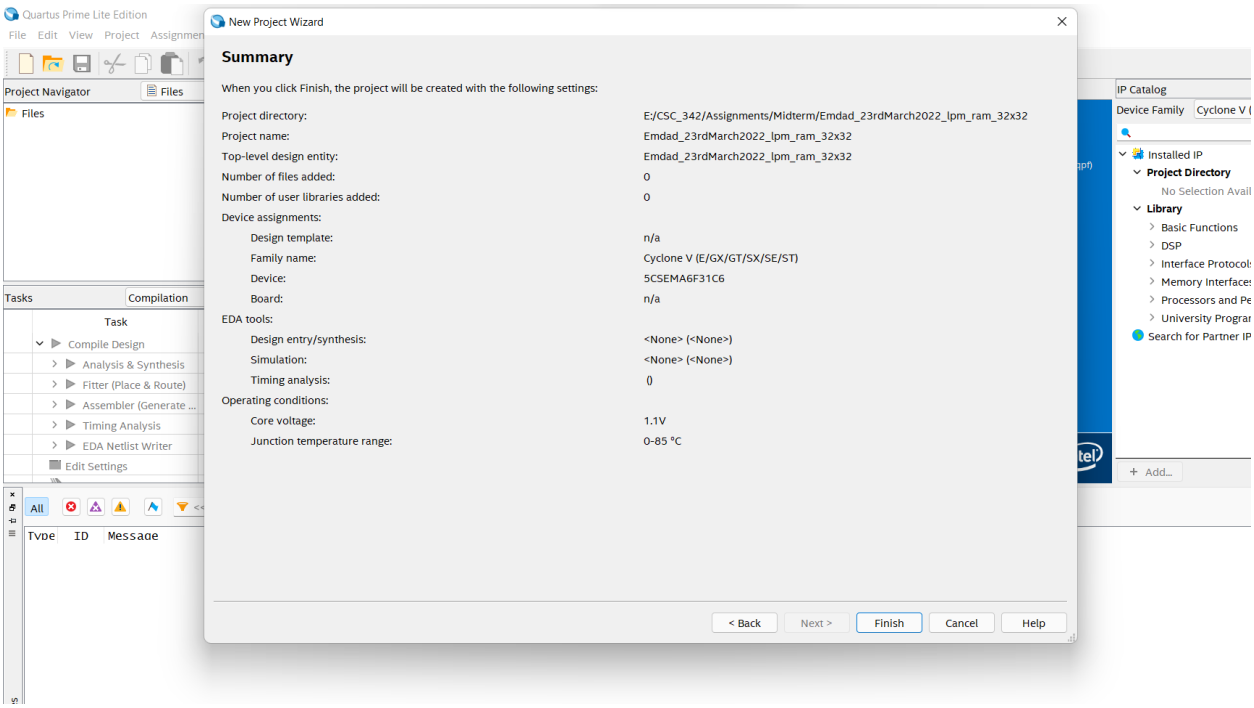

Figure 15: project summary for lpm RAM-1 PORT 32x32.

In figure 16, we are selecting for 32 bit and 32 words for MIF file.

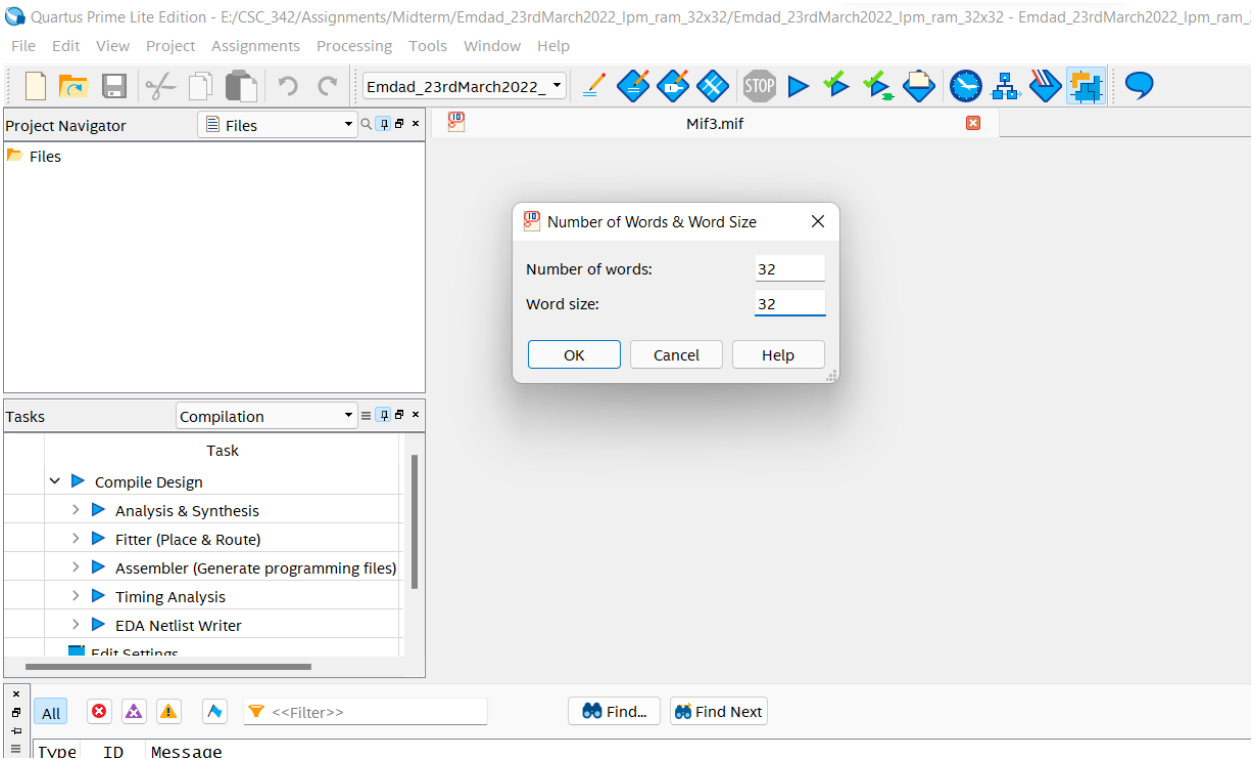

Figure 16: MIF file config

In figure 17, we inserted our data in the MIF file.

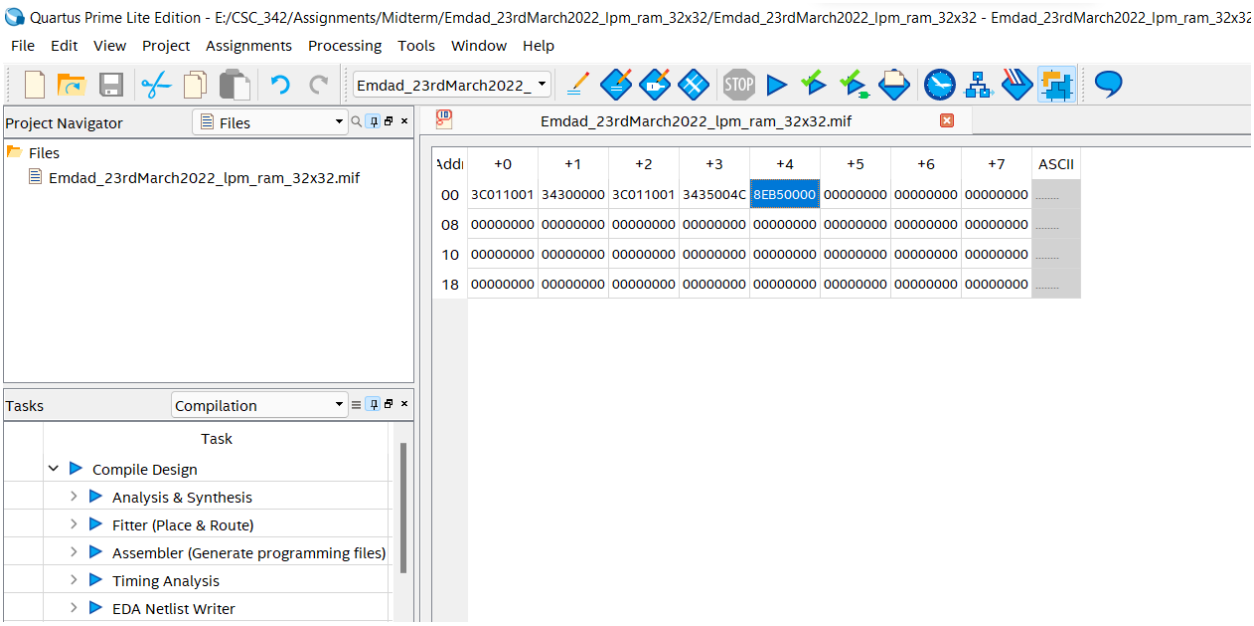

Figure 17: inserted data into MIF file

In figure 18, we are configuring IP variation for lpm RAM-1 PORT 32x32.

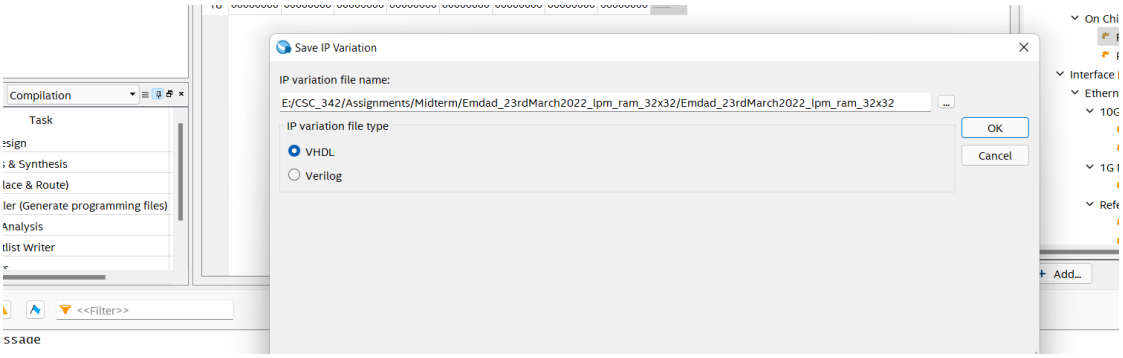

Figure 18: IP variation for lpm RAM-1 PORT 32x32.

In figure 19, we can configure q output bits and words size for the port which is 32bits X 32 words.

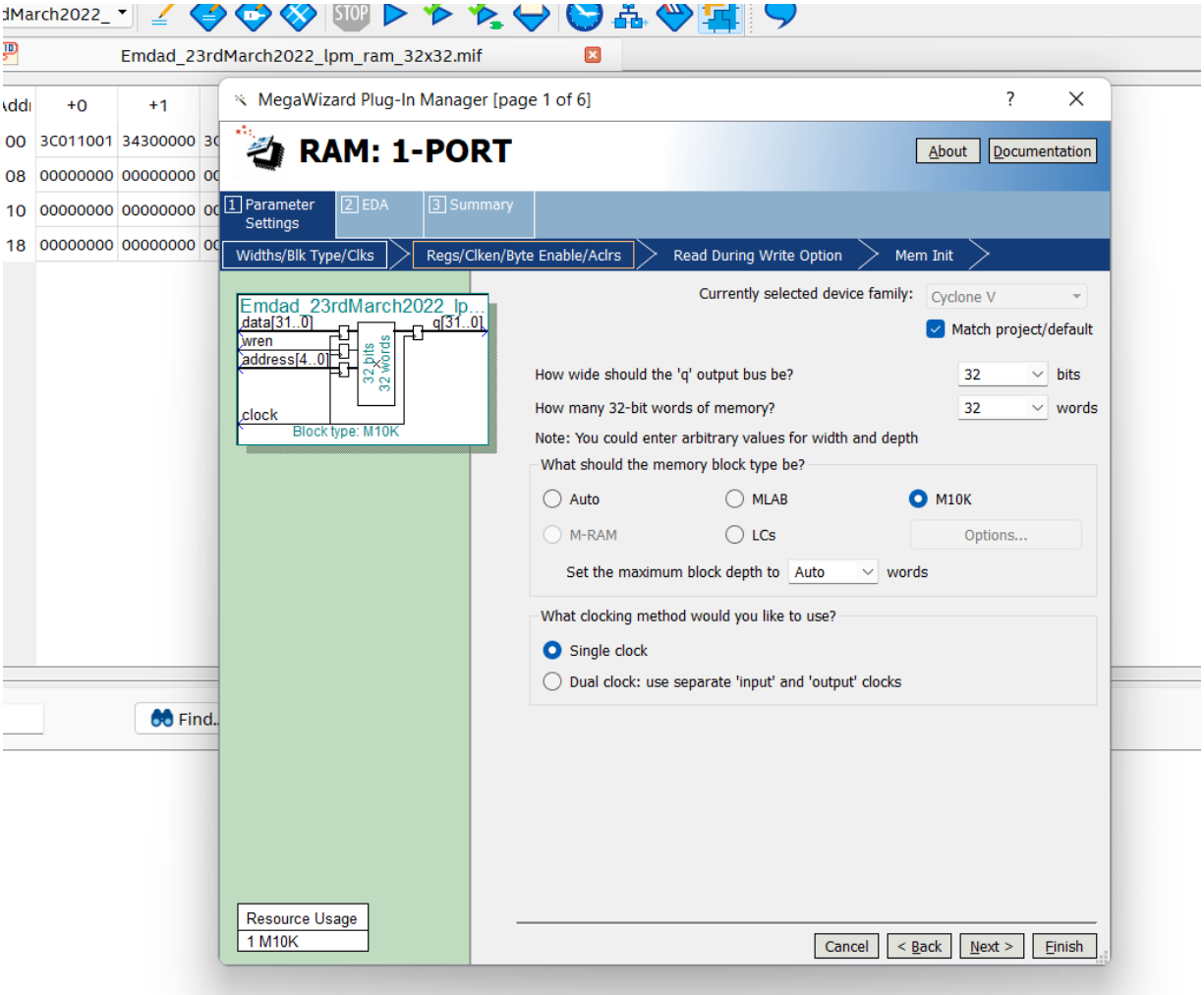

Figure 19: lpm RAM-1 PORT 32x32 config Part 1

In figure 20, we uncheck the “q output port” option and click next.

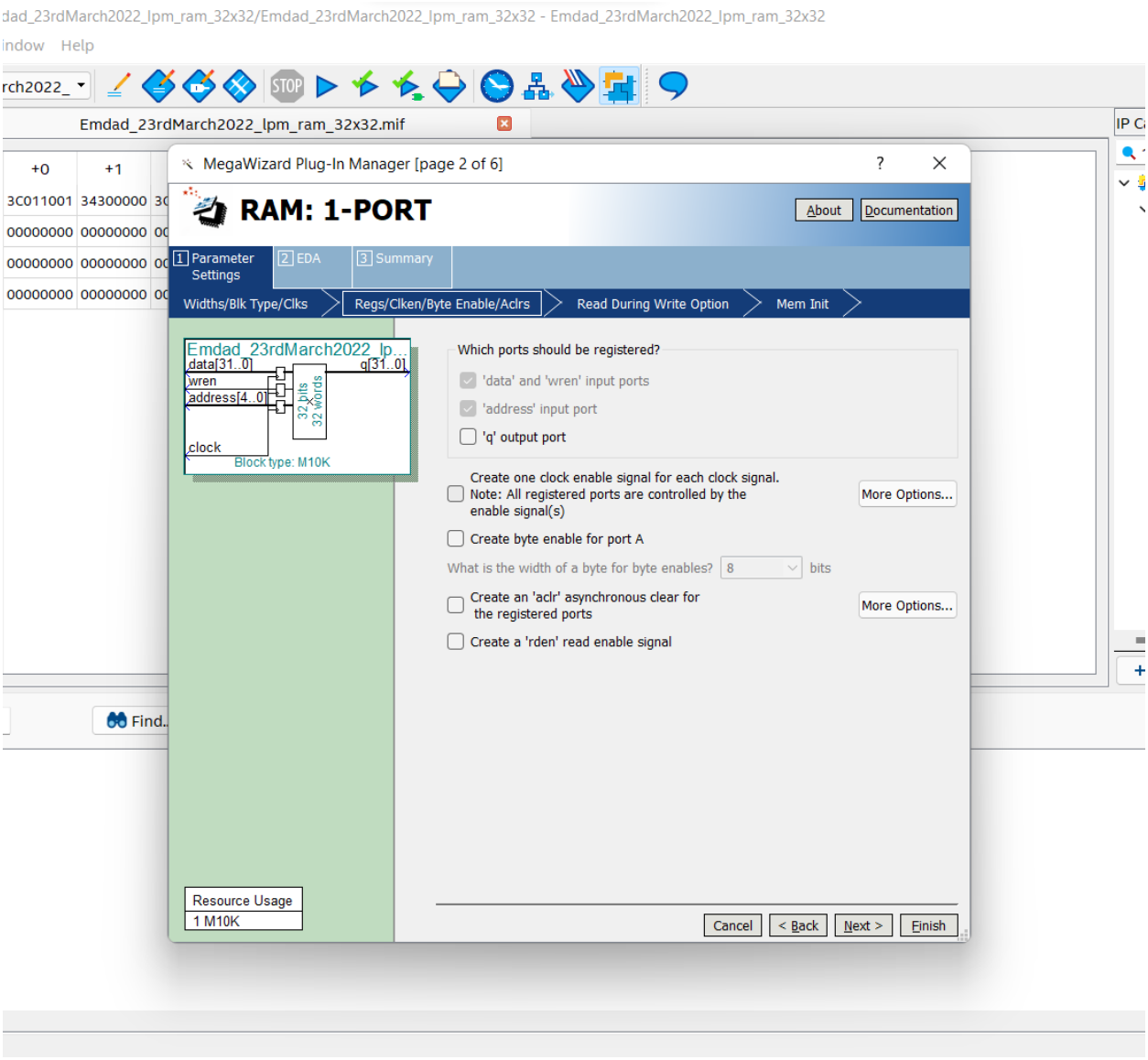

Figure 20: lpm RAM-1 PORT 32x32 config Part 2

In figure 21, we add our created MIF file to read date from it.

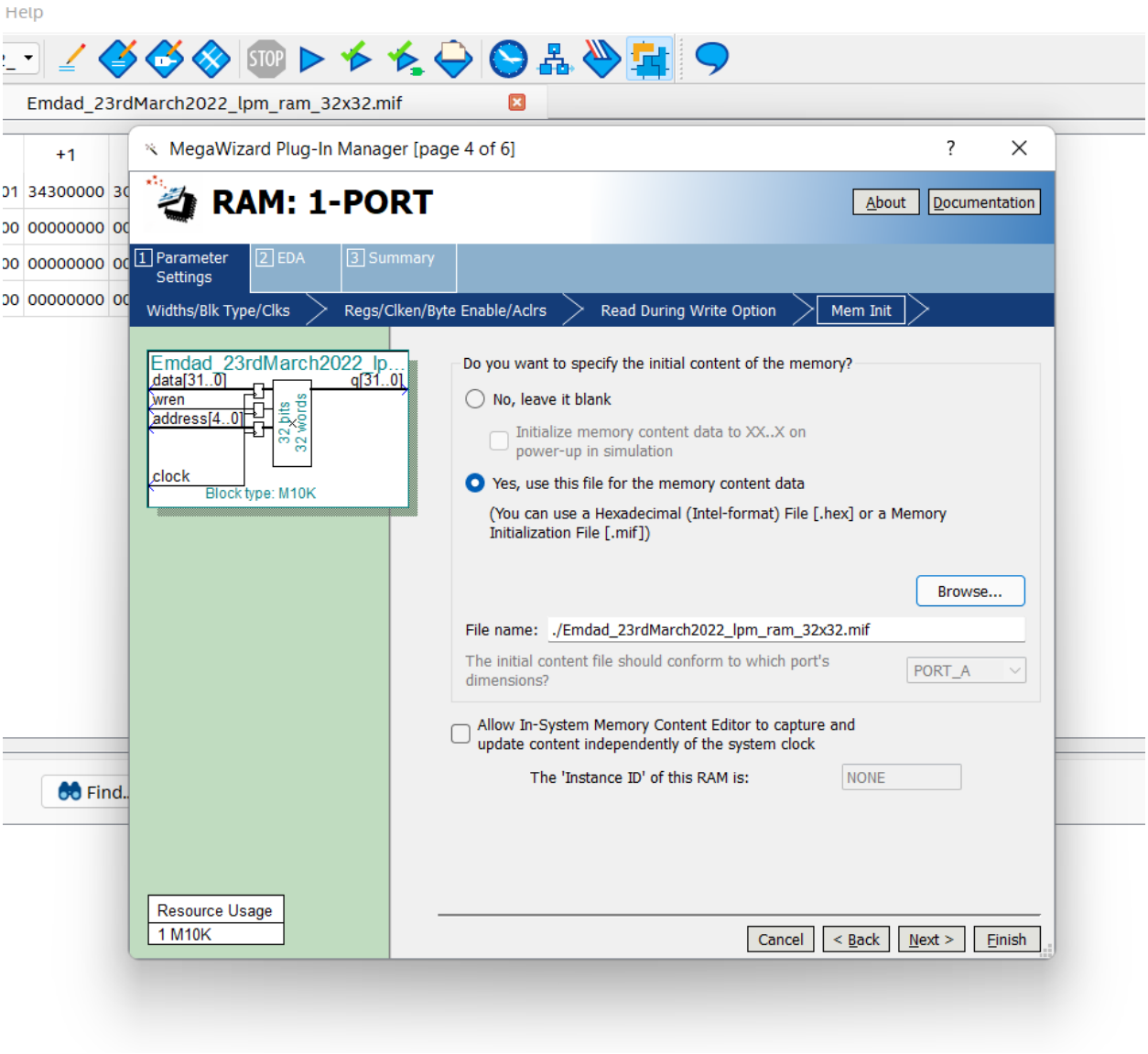

Figure 21: lpm RAM-1 PORT 32x32 config Part 3

In figure 22, we can a summary for the : lpm RAM-1 PORT 32x32.

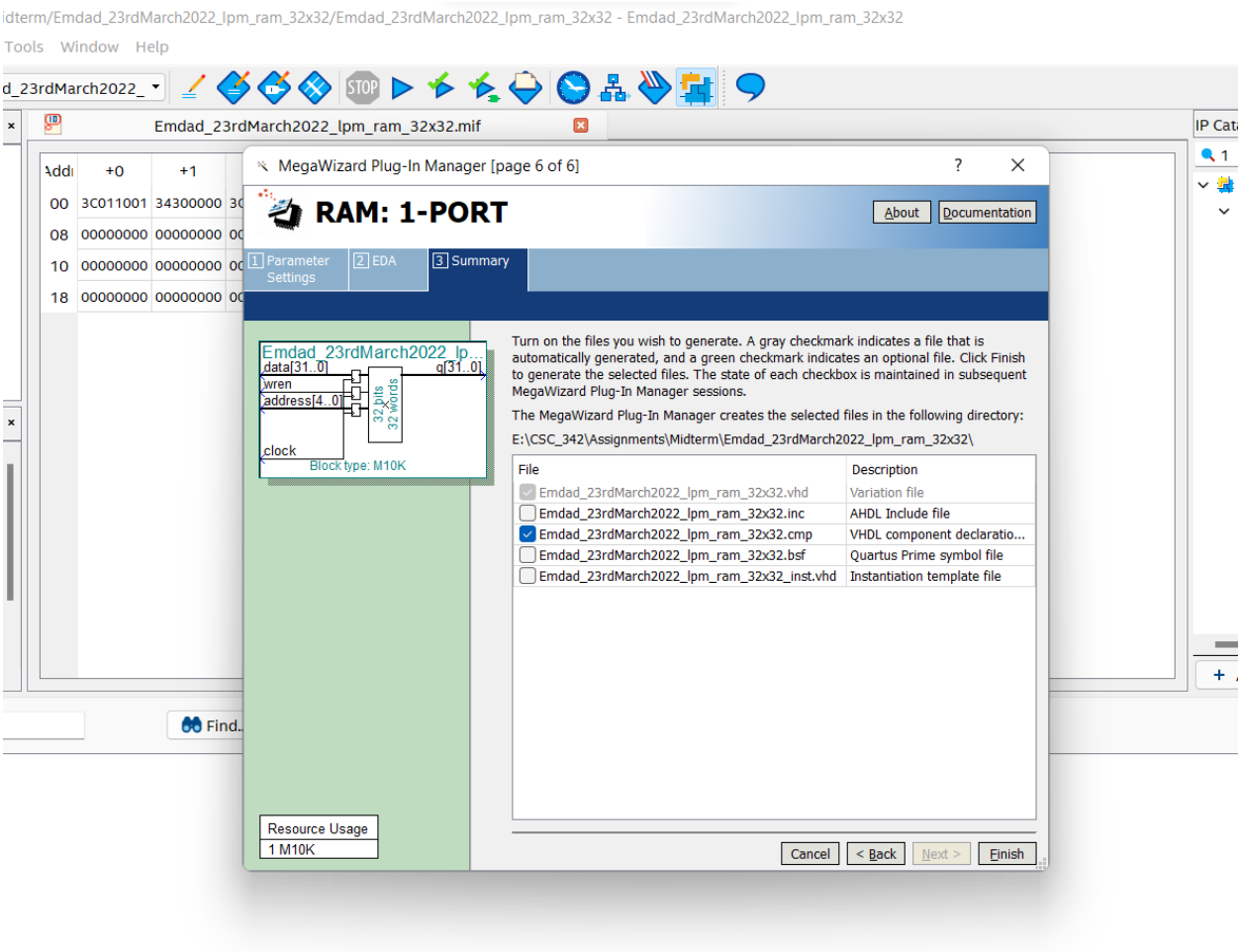

Figure 22: lpm RAM-1 PORT 32x32 config Part 4

In figure 23, we can see the lpm RAM-1 PORT 32x32 VHDL code generated by the modules.

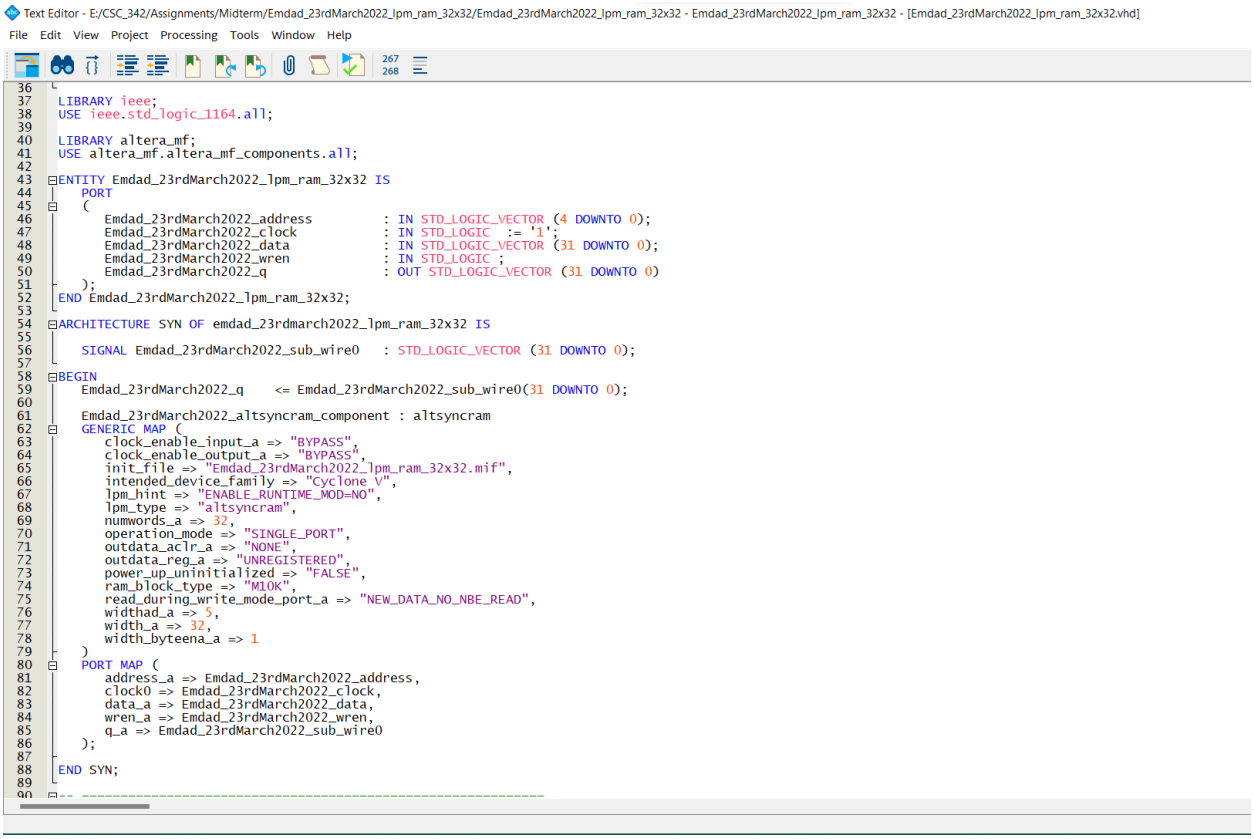

Figure 23: VHDL code lpm RAM-1 PORT 32x32

In figure 24, we can see it compiled successfully.

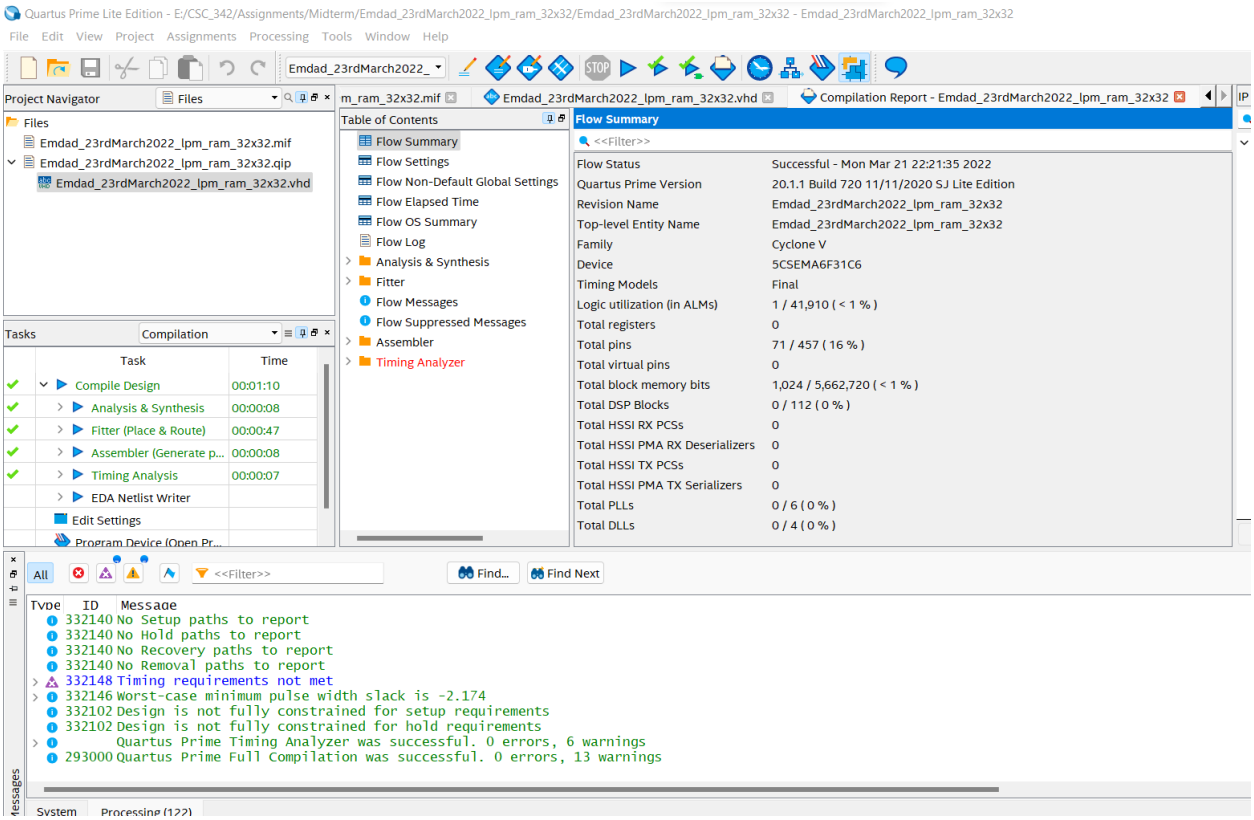

Figure 24: lpm RAM-1 PORT 32x32 compiled successfully

iii. Dual Ported Memory Module:

In figure 25, we can see the file directory for Dual Ported Memory Module.

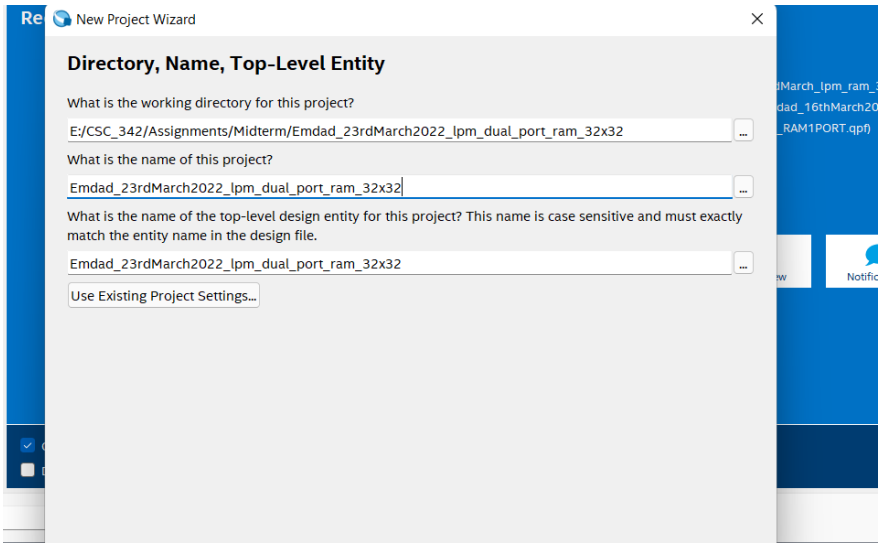

Figure 25: file directory for Dual Ported Memory Module.

In figure 26, we can see the project summary for Dual Ported Memory Module.

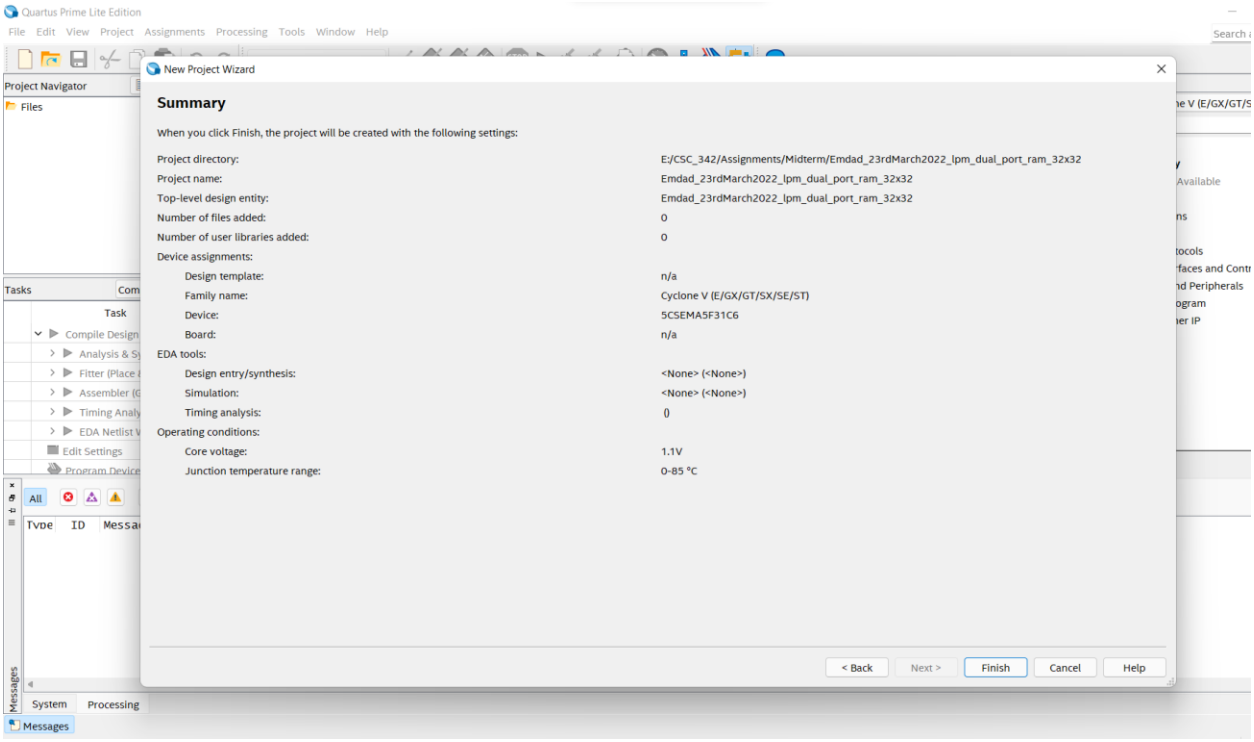

Figure 26: project summary for Dual Ported Memory Module

In figure 27, we are configure input size for MIF file.

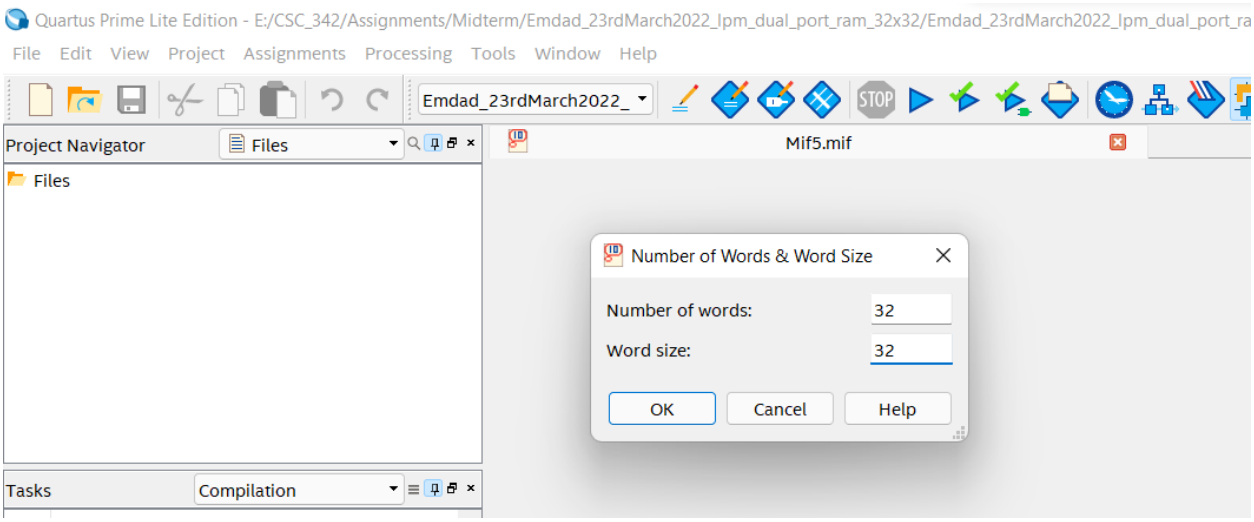

Figure 27: MIF file input config

In figure 28, we are inserting data in MIF file.

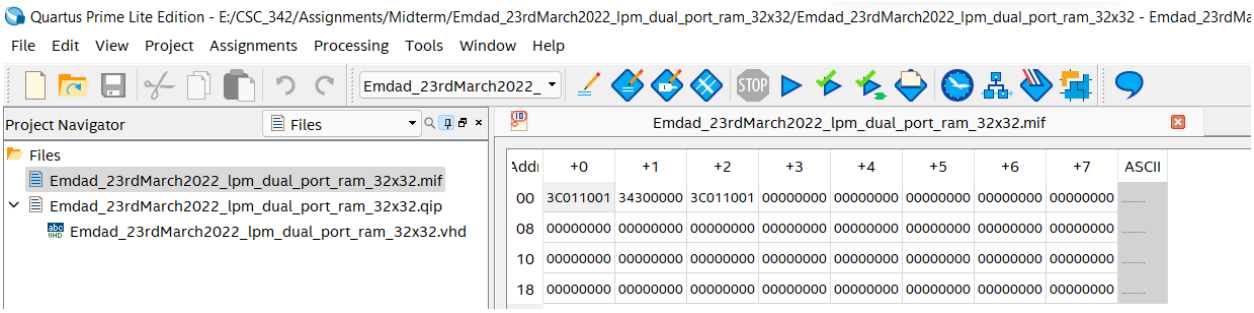

Figure 28: insert data into MIF file

In figure 29, we can see that we chose 32 bits and then M10K for memory block.

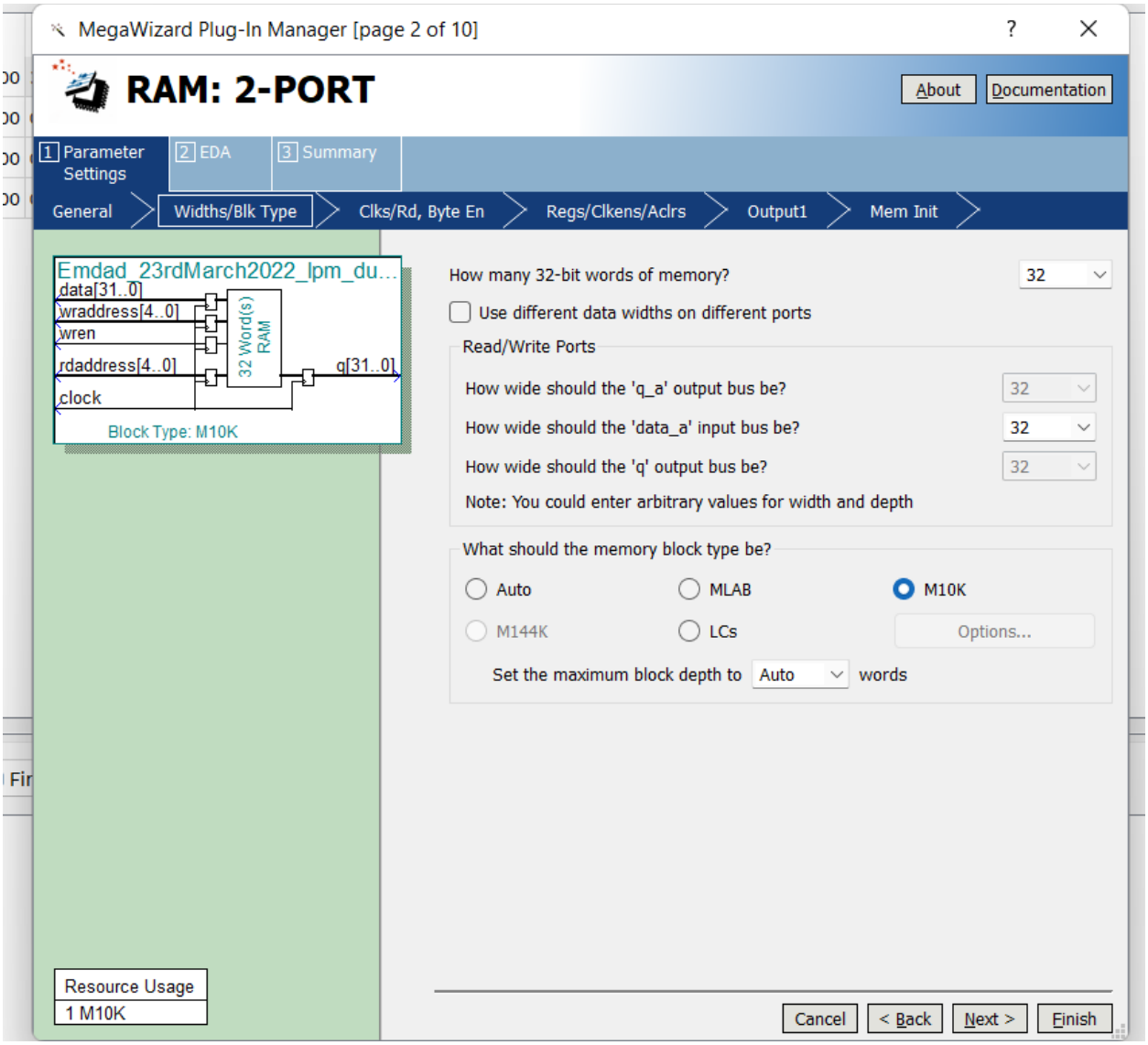

Figure 29: lpm RAM-2 PORT 32x32 config Part 1

In figure 30, we are adding out created MIF in the modules.

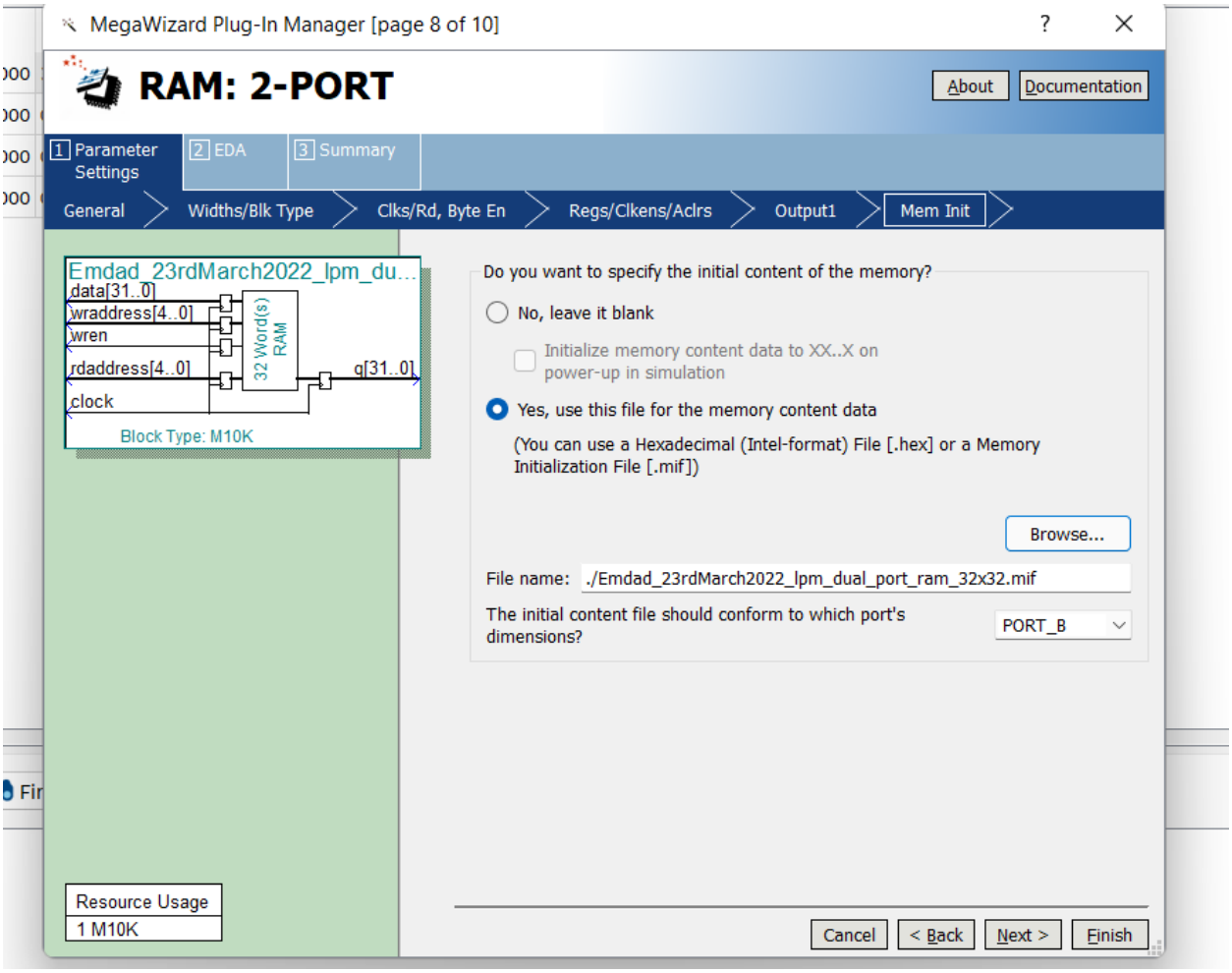

Figure 30: lpm RAM-2 PORT 32x32 config Part 2

In figure 31, we can see the generated VHDL code of lpm RAM-2 PORT 32x32.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY Emdad_23rdMarch2022_lpm_dual_port_ram_32x32 IS
    PORT
    (
        Emdad_23rdMarch2022_clock                       : IN STD_LOGIC  := '1';
        Emdad_23rdMarch2022_data                        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        Emdad_23rdMarch2022_rdaddress                   : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        Emdad_23rdMarch2022_wraddress                   : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        Emdad_23rdMarch2022_wren                        : IN STD_LOGIC  := '0';
        Emdad_23rdMarch2022_q                           : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END Emdad_23rdMarch2022_lpm_dual_port_ram_32x32;

ARCHITECTURE SYN OF emdad_23rdmarch2022_lpm_dual_port_ram_32x32 IS

    SIGNAL Emdad_23rdMarch2022_sub_wire0    : STD_LOGIC_VECTOR (31 DOWNTO 0);

BEGIN
    Emdad_23rdMarch2022_q    <= Emdad_23rdMarch2022_sub_wire0(31 DOWNTO 0);
```

Figure 31: RAM-2 PORT 32x32 VHDL code

```

    Emdad_23rdMarch2022_altsyncram_component : altsyncram
    GENERIC MAP (
        address_aclr_b => "NONE",
        address_reg_b => "CLOCK0",
        clock_enable_input_a => "BYPASS",
        clock_enable_input_b => "BYPASS",
        clock_enable_output_b => "BYPASS",
        init_file => "Emdad_23rdMarch2022_lpm_dual_port_ram_32x32.mif",
        intended_device_family => "Cyclone V",
        lpm_type => "altsyncram",
        numwords_a => 32,
        numwords_b => 32,
        operation_mode => "DUAL_PORT",
        outdata_aclr_b => "NONE",
        outdata_reg_b => "CLOCK0",
        power_up_uninitialized => "FALSE",
        ram_block_type => "M10K",
        read_during_write_mode_mixed_ports => "DONT_CARE",
        widthad_a => 5,
        widthad_b => 5,
        width_a => 32,
        width_b => 32,
        width_byteena_a => 1
    )
    PORT MAP (
        address_a => Emdad_23rdMarch2022_wraddress,
        address_b => Emdad_23rdMarch2022_rdaddress,
        clock0 => Emdad_23rdMarch2022_clock,
        data_a => Emdad_23rdMarch2022_data,
        wren_a => Emdad_23rdMarch2022_wren,
        q_b => Emdad_23rdMarch2022_sub_wire0
    );

END SYN;
```

Figure 32: lpm RAM-2 PORT 32x32 VHDL code

In figure 33, we can see it compiled successfully.

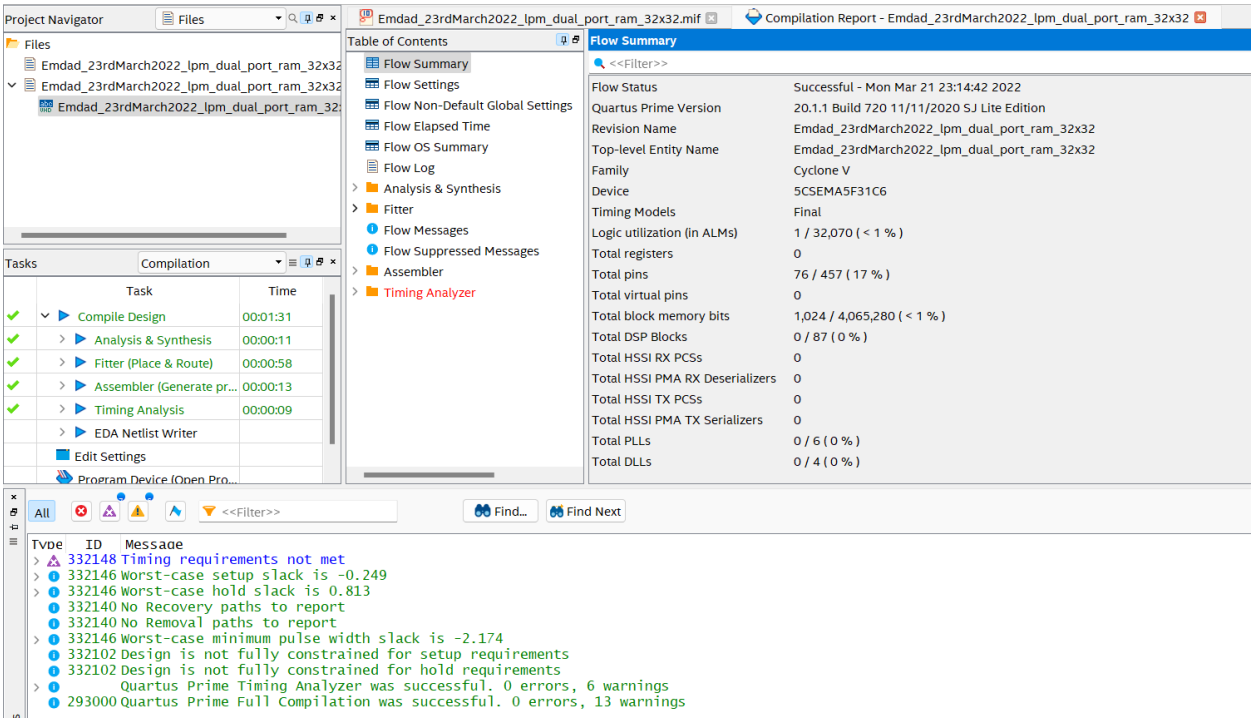

Figure 33: lpm RAM-2 PORT 32x32 compilation report

ASSIGNMENT 2:

i. 32-bit add/sub scratch

In figure 34, we are creating file directory for 32-bit add/sub scratch.

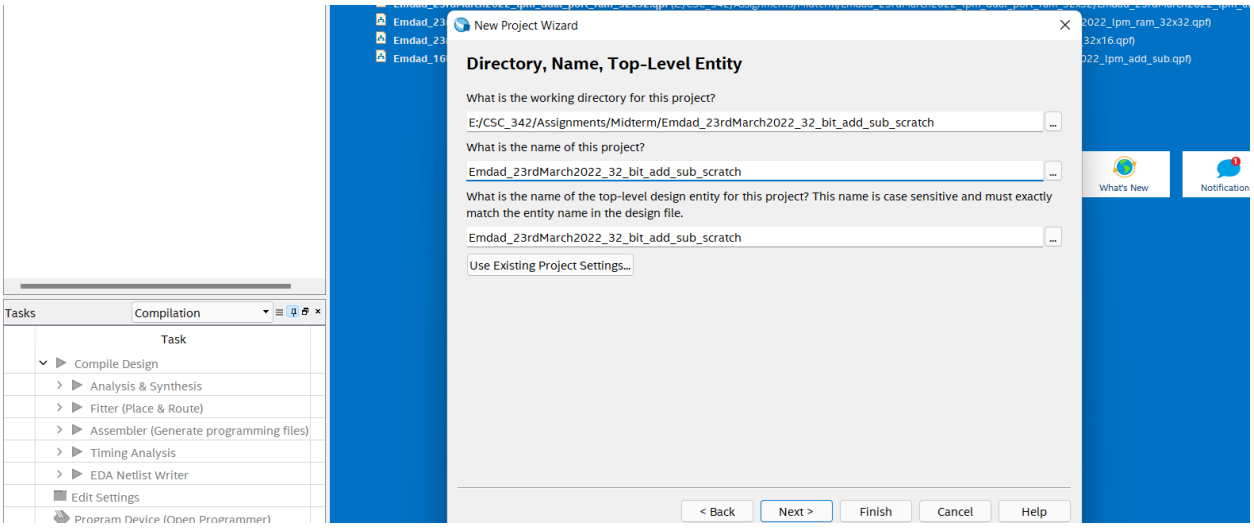

Figure 34: 32-bit add/sub scratch file directory

In figure 35, we can see the project summary for 32-bit add/sub scratch.

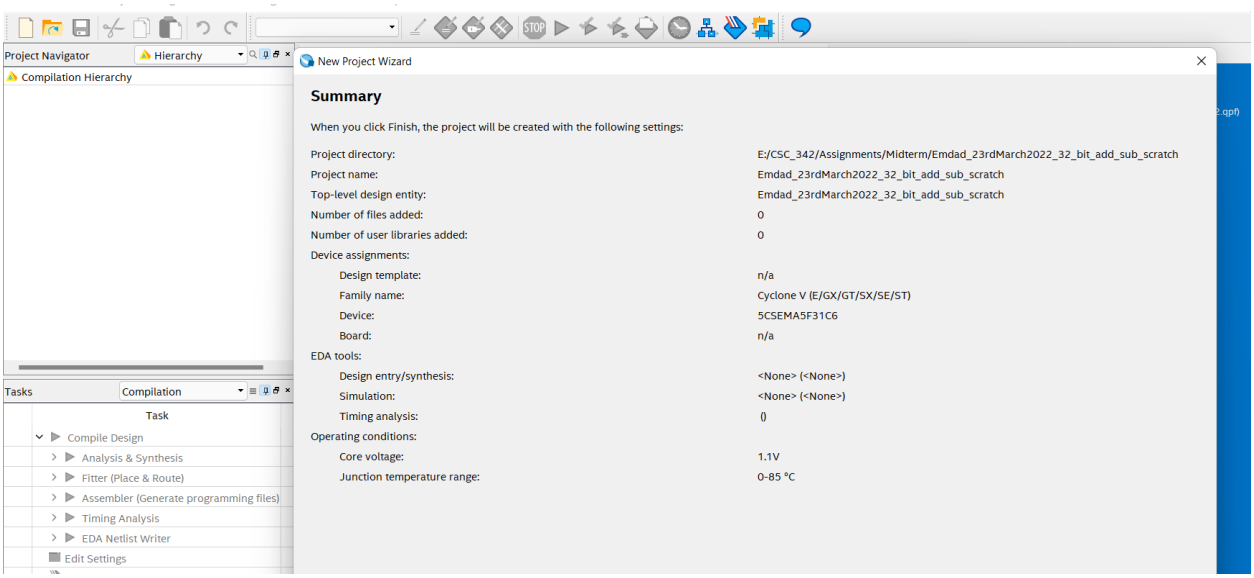

Figure 35: project summary for 32-bit add/sub scratch

In figure 36, we can see the VHDL code for adderk.

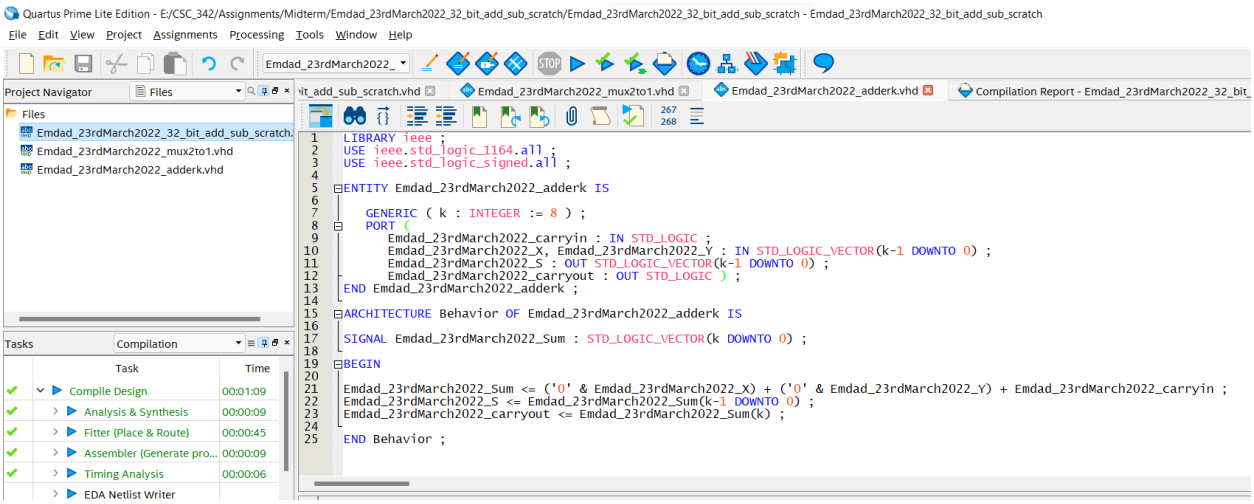

Figure 36: VHDL code for adderk

In figure 37, we can see the VHDL code for mux2to1.

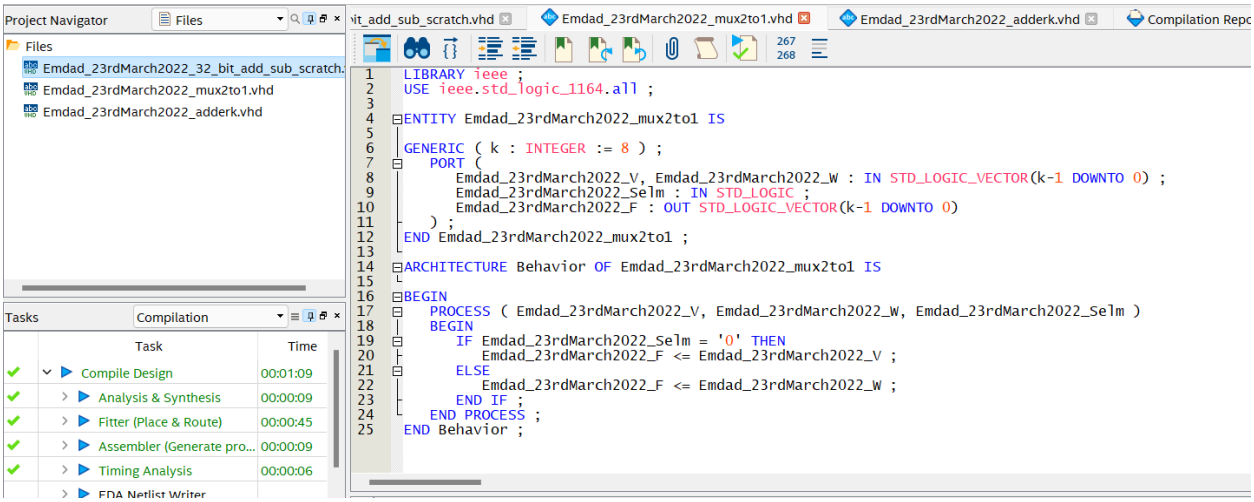

Figure 37: VHDL code for mux2to1

In figure 38, we can see the VHDL code for add_sub from intel pdf.

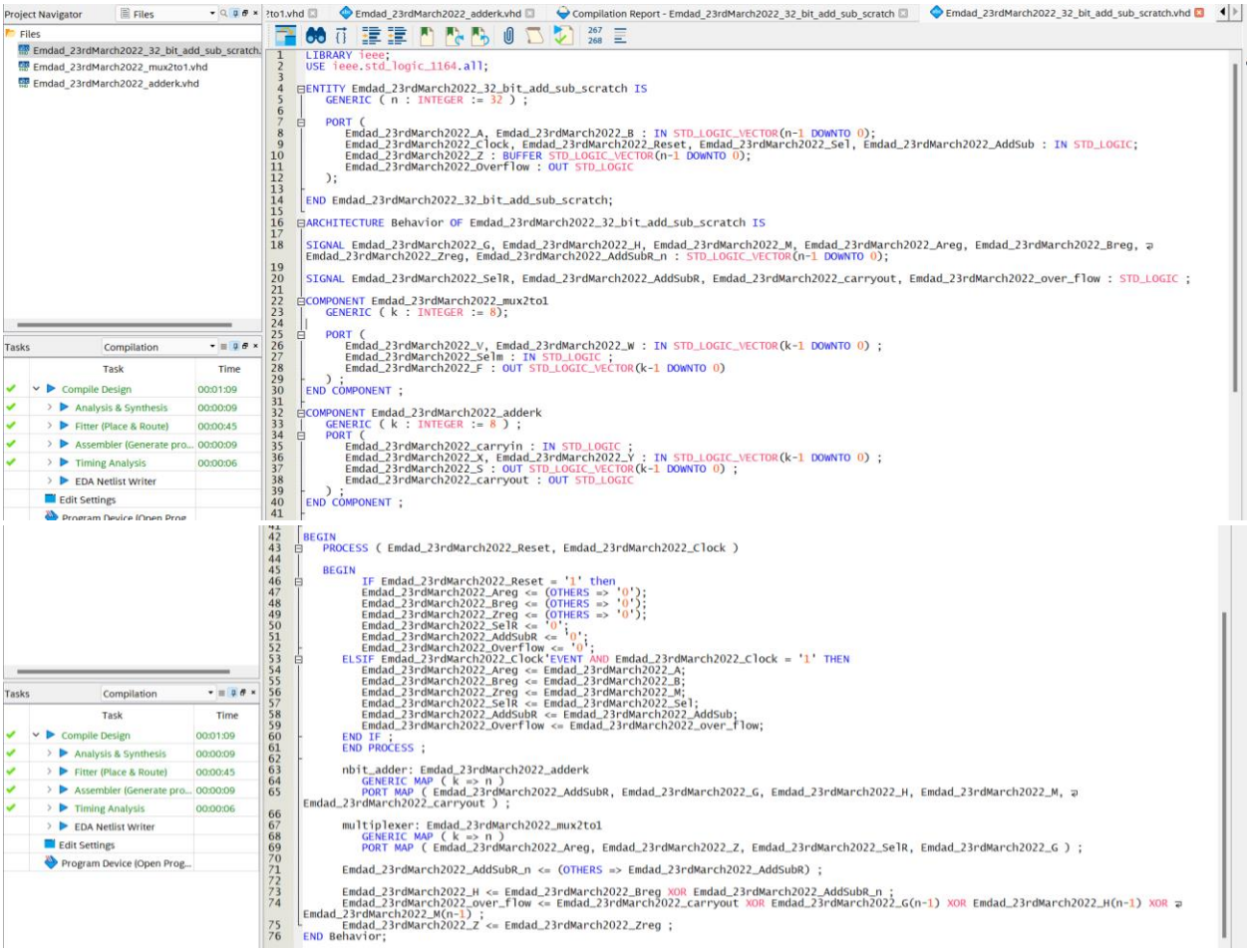

Figure 38: VHDL code for add_sub

In figure 39, we can see that all the three files compiled successfully.

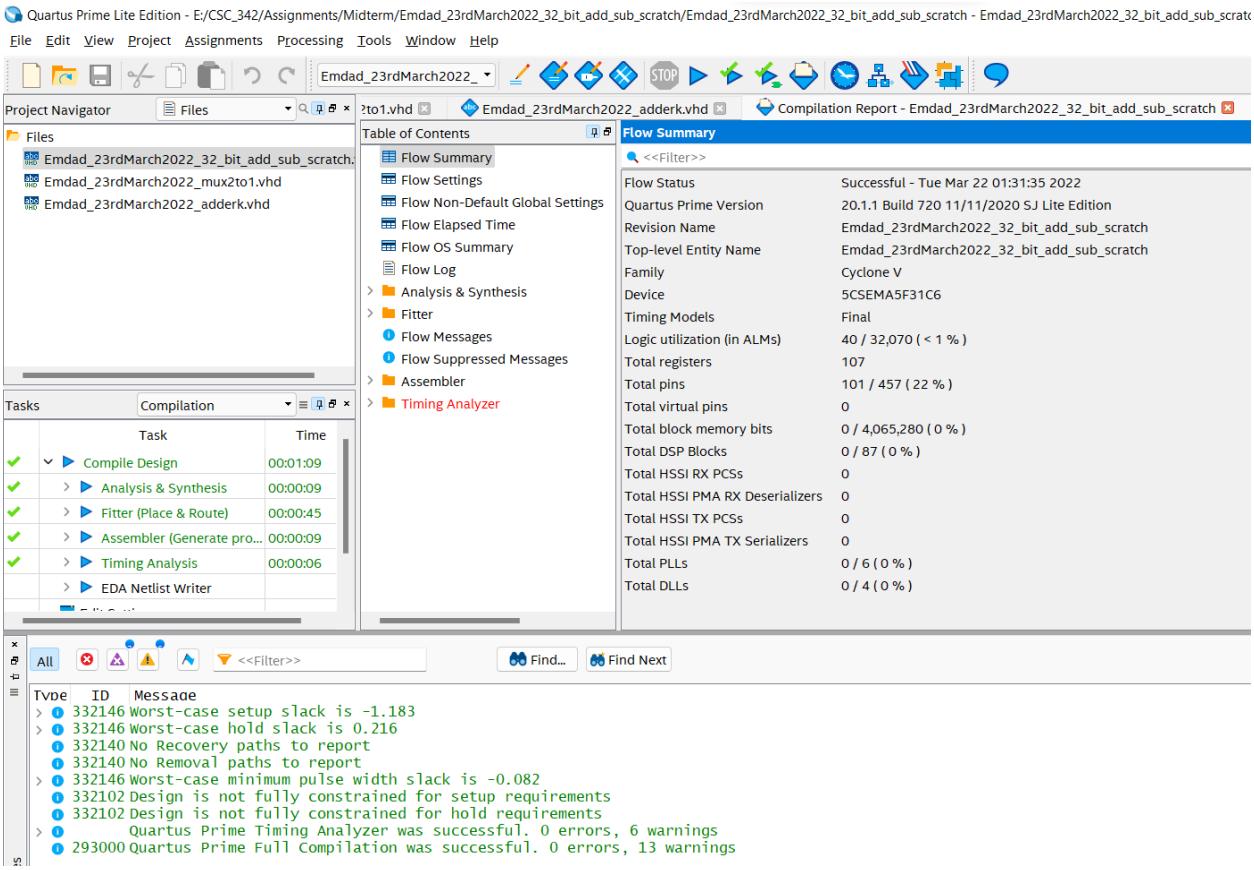

Figure 39: compilation report for add_sub

ii. 32-bit add/sub LPM module:

In figure 40, we can see the project directory for 32-bit add/sub LPM module.

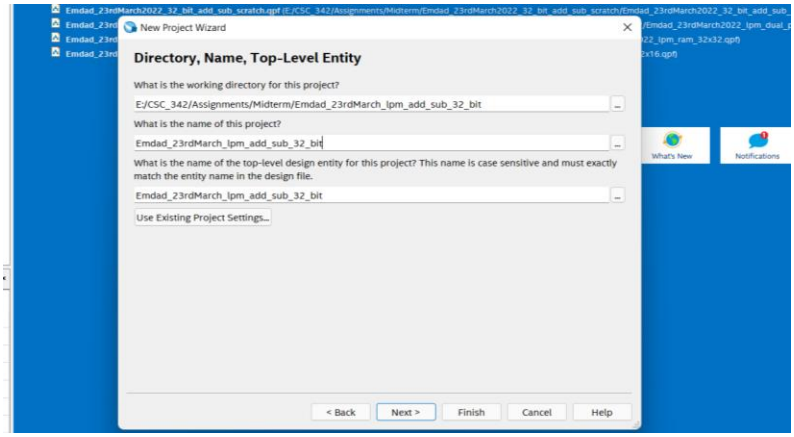

Figure 40: project directory for 32-bit add/sub LPM module

In figure 41, we can see the project summary for 32-bit add/sub LPM module.

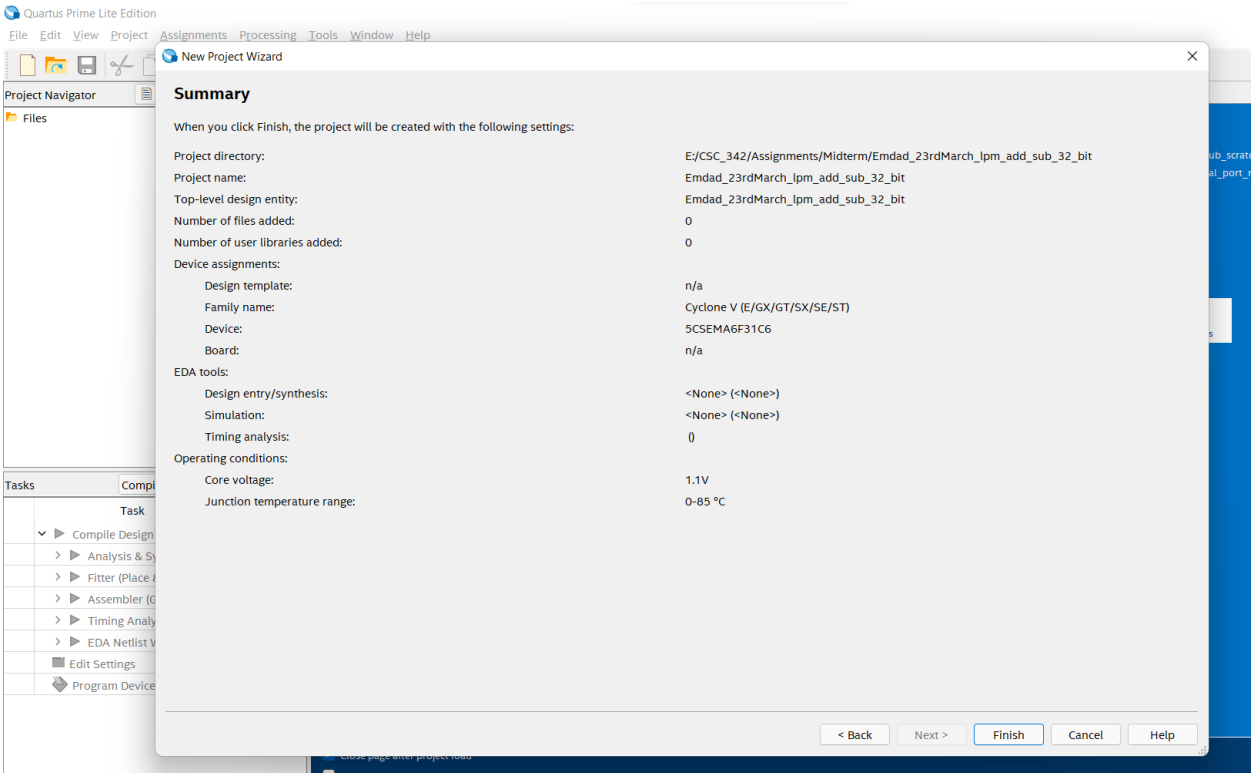

Figure 41: project summary for 32-bit add/sub LPM module

In figure 42, we are doing IP varitation for 32-bit add/sub LPM module.

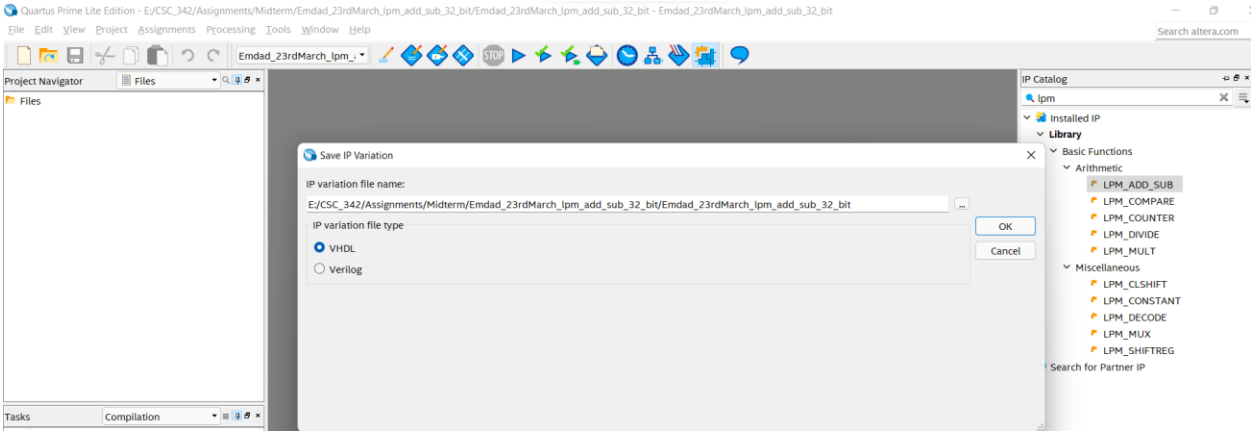

Figure 42: IP variation for 32-bit add/sub LPM module

In figure 43, we are selecting 32 bits and creating add_sub.

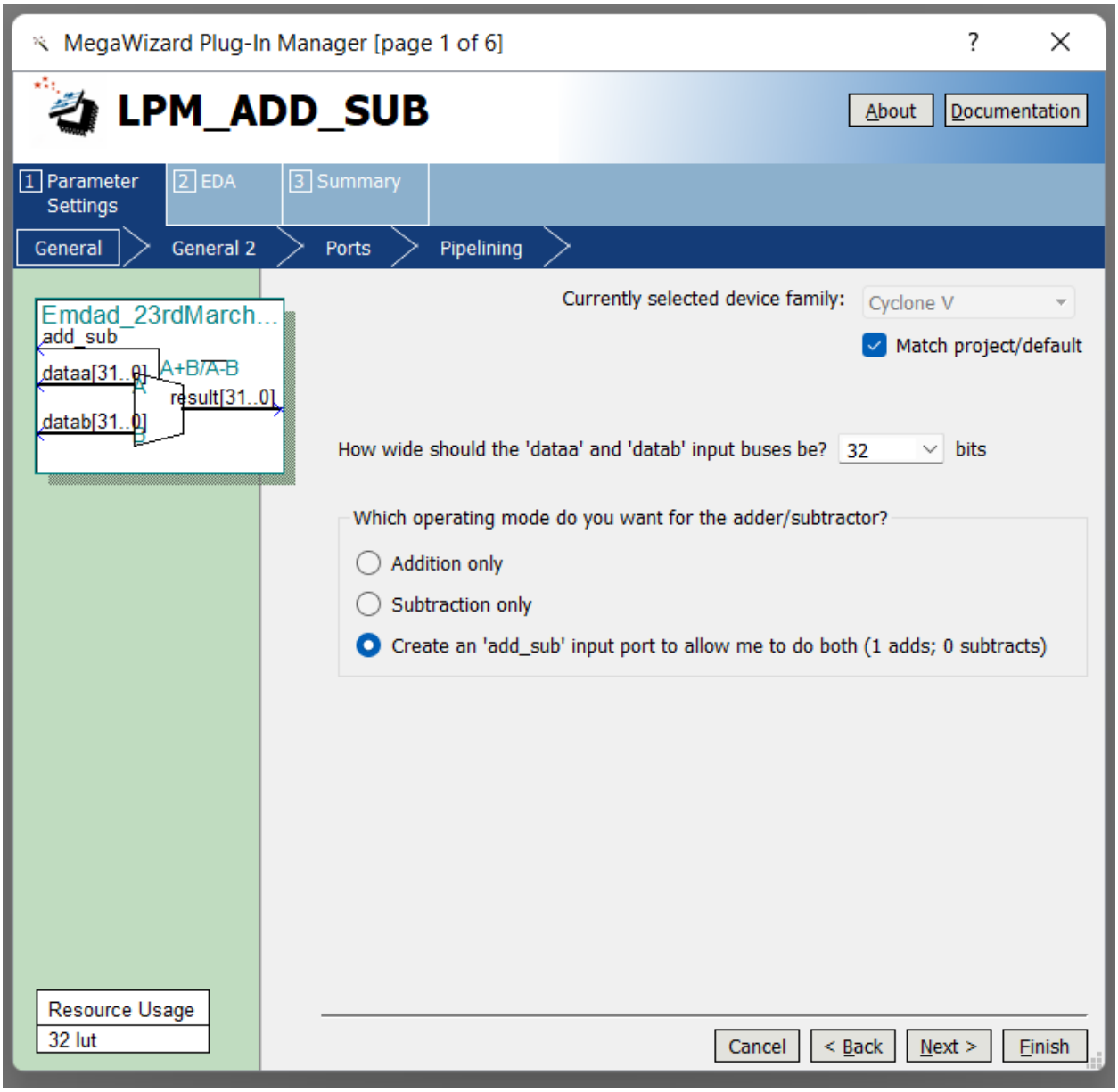

Figure 43:32-bit add/sub LPM module config Part 1

In figure 44, we can change the addition/substraction to signed.

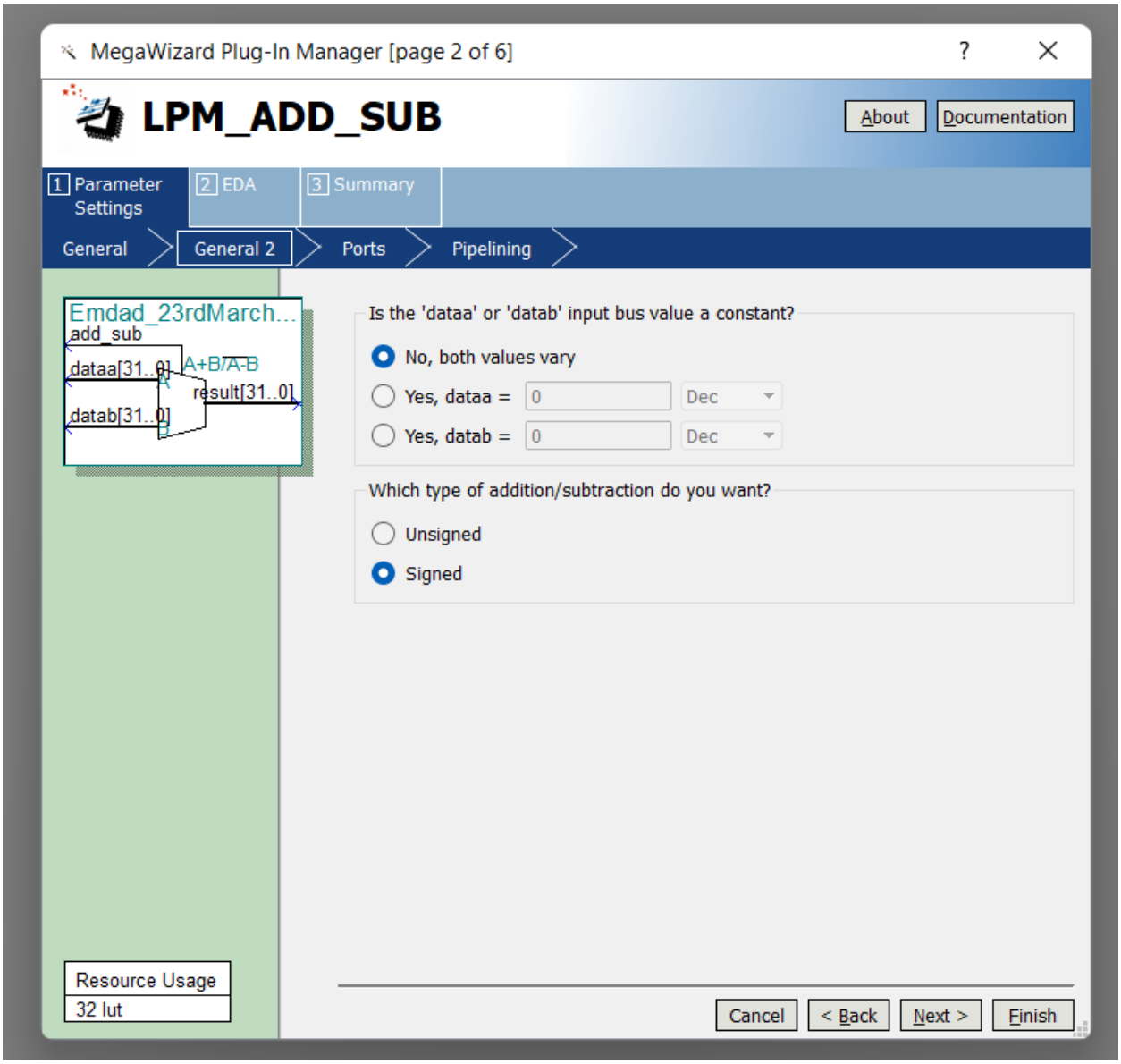

Figure 44:32-bit add/sub LPM module config Part 2

In figure 45, we care creating an overflow output.

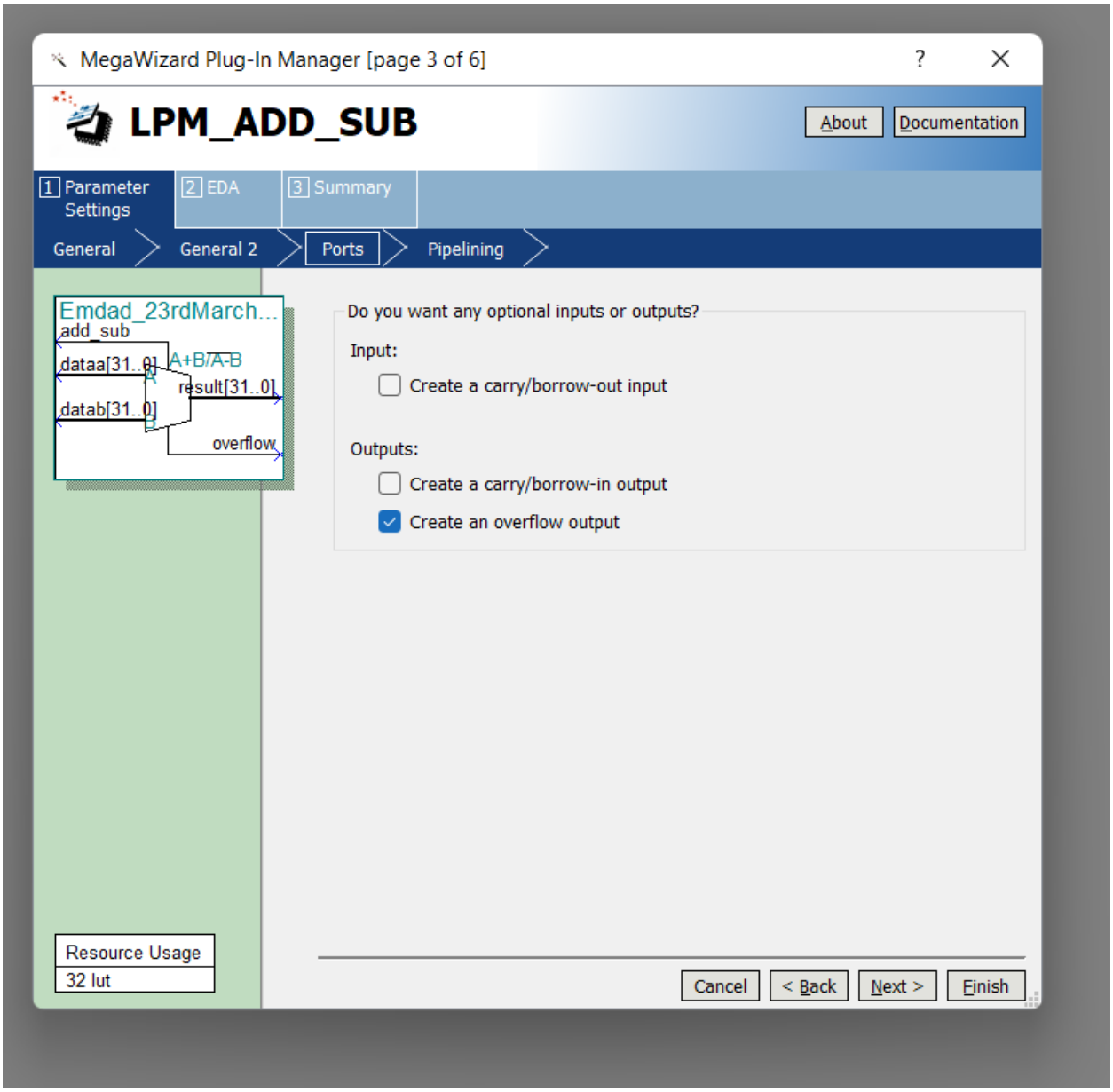

Figure 45:32-bit add/sub LPM module config Part 3

In figure 46, we can see the 32-bit add/sub LPM module config summary.

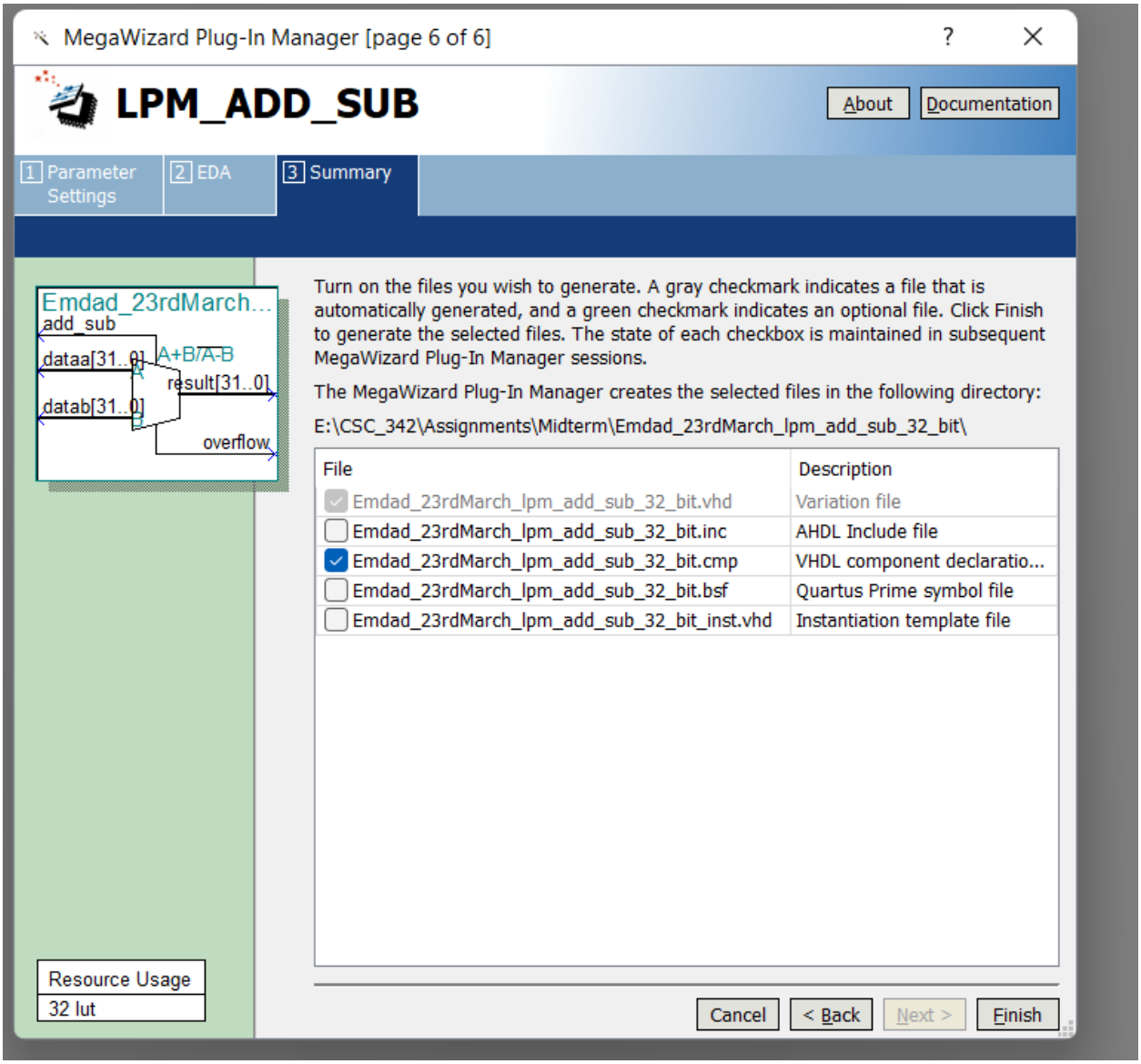

Figure 46:32-bit add/sub LPM module config summary

In figure 47, we can see the generated VHDL code from 32-bit add/sub LPM module.

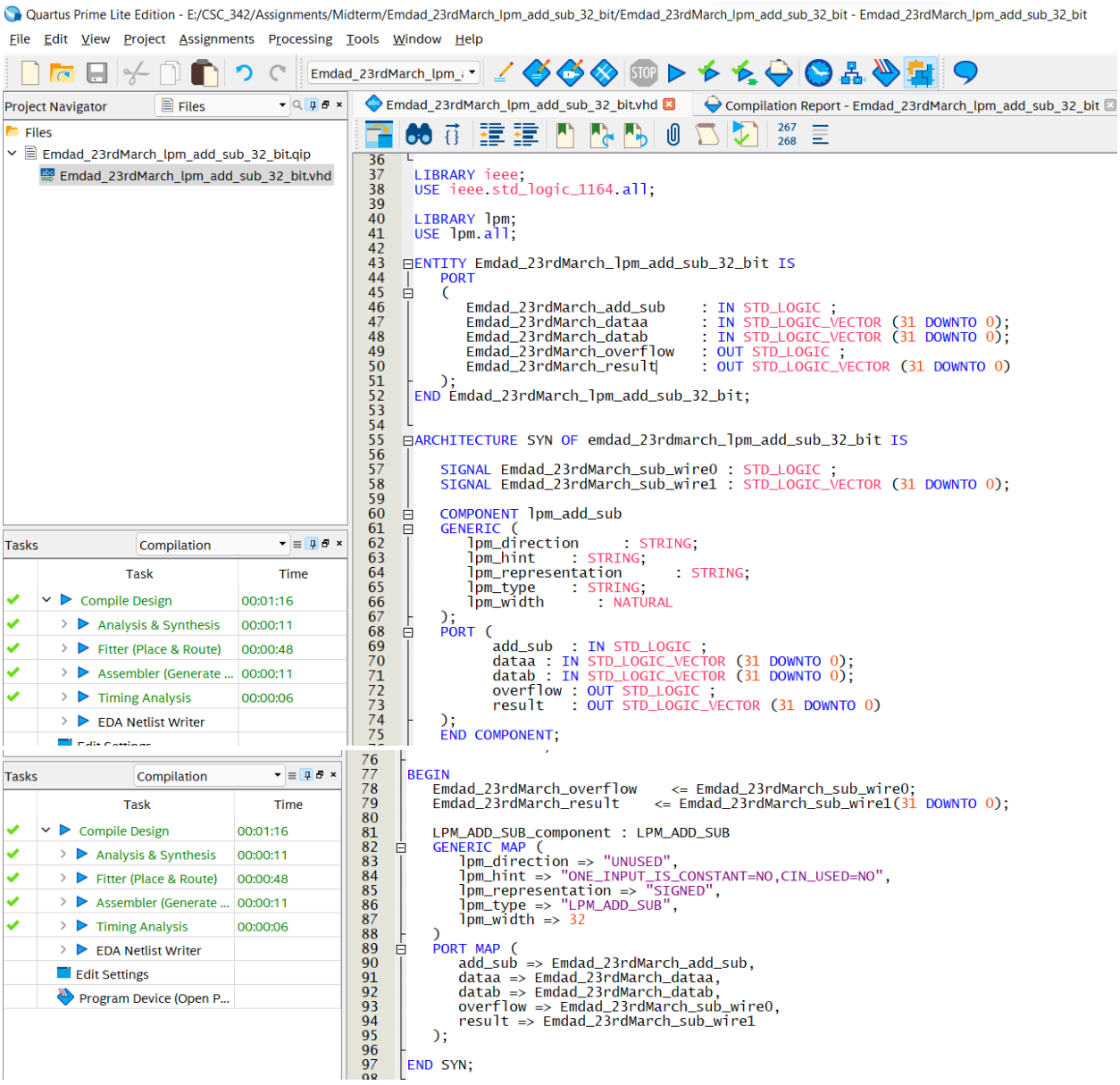

Figure 47: 32-bit add/sub LPM module VHDL code

iii. Circuit Design using add/sub:

In figure 48, we can see file directory for circuit design.

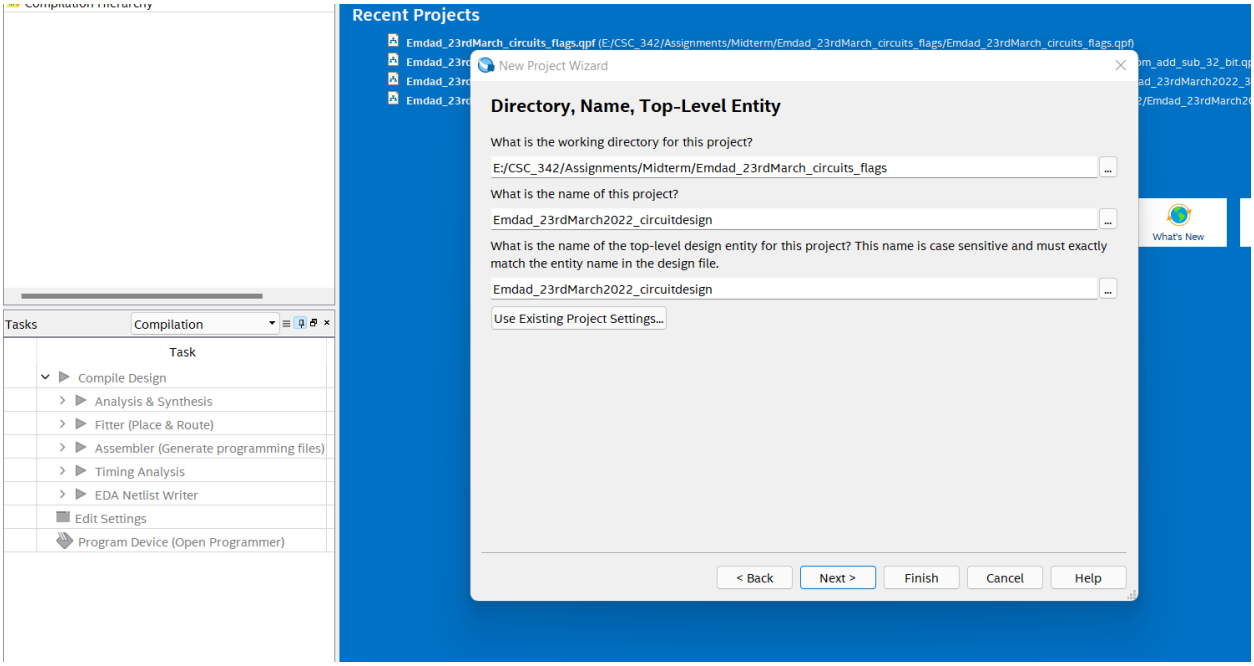

Figure 48: file directory for circuit design.

In figure 49, we can see the project summary for circuit design.

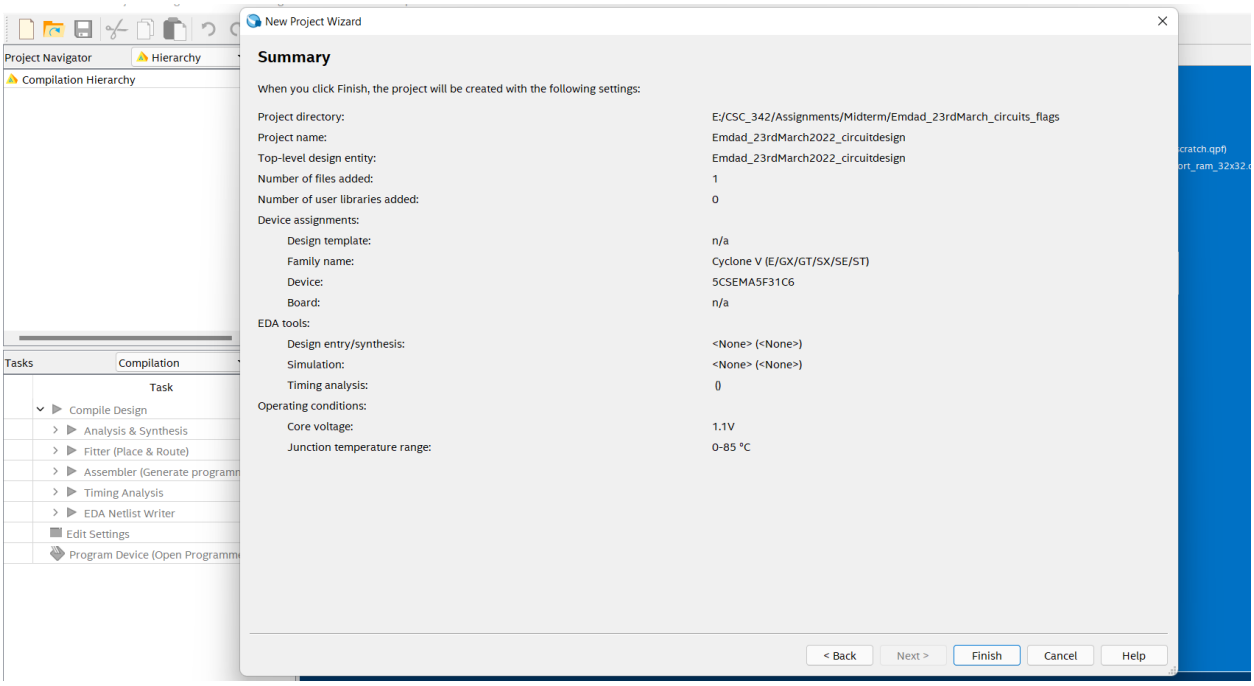

Figure 49: project summary for circuit design.

In figure 50, we can see the MIF data inserted.

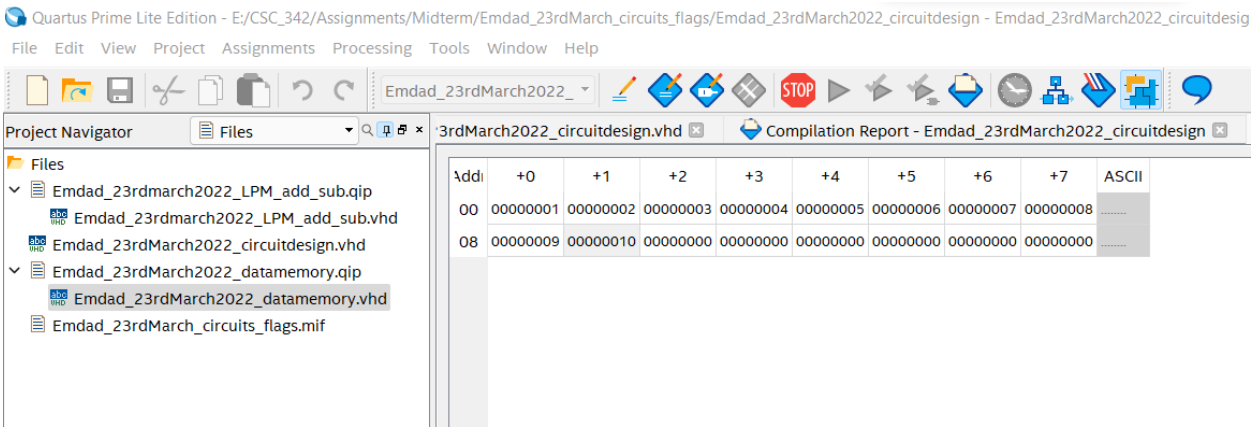

Figure 50: data inserted in MIF file

In figure 51, we can see the code data memory.

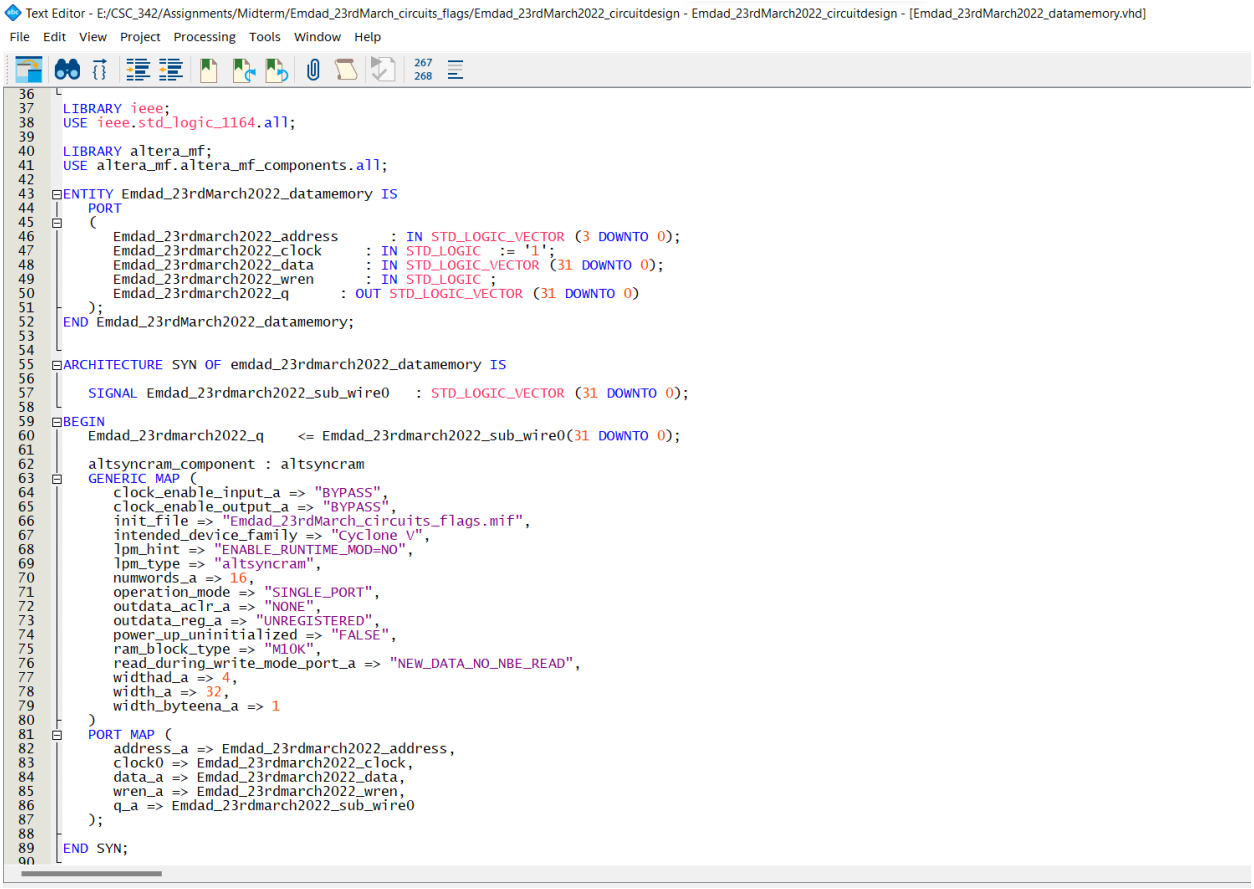

Figure 51: data memory VHDL code

In figure 52, we can see the VHDL code for LPM add/sub modules.

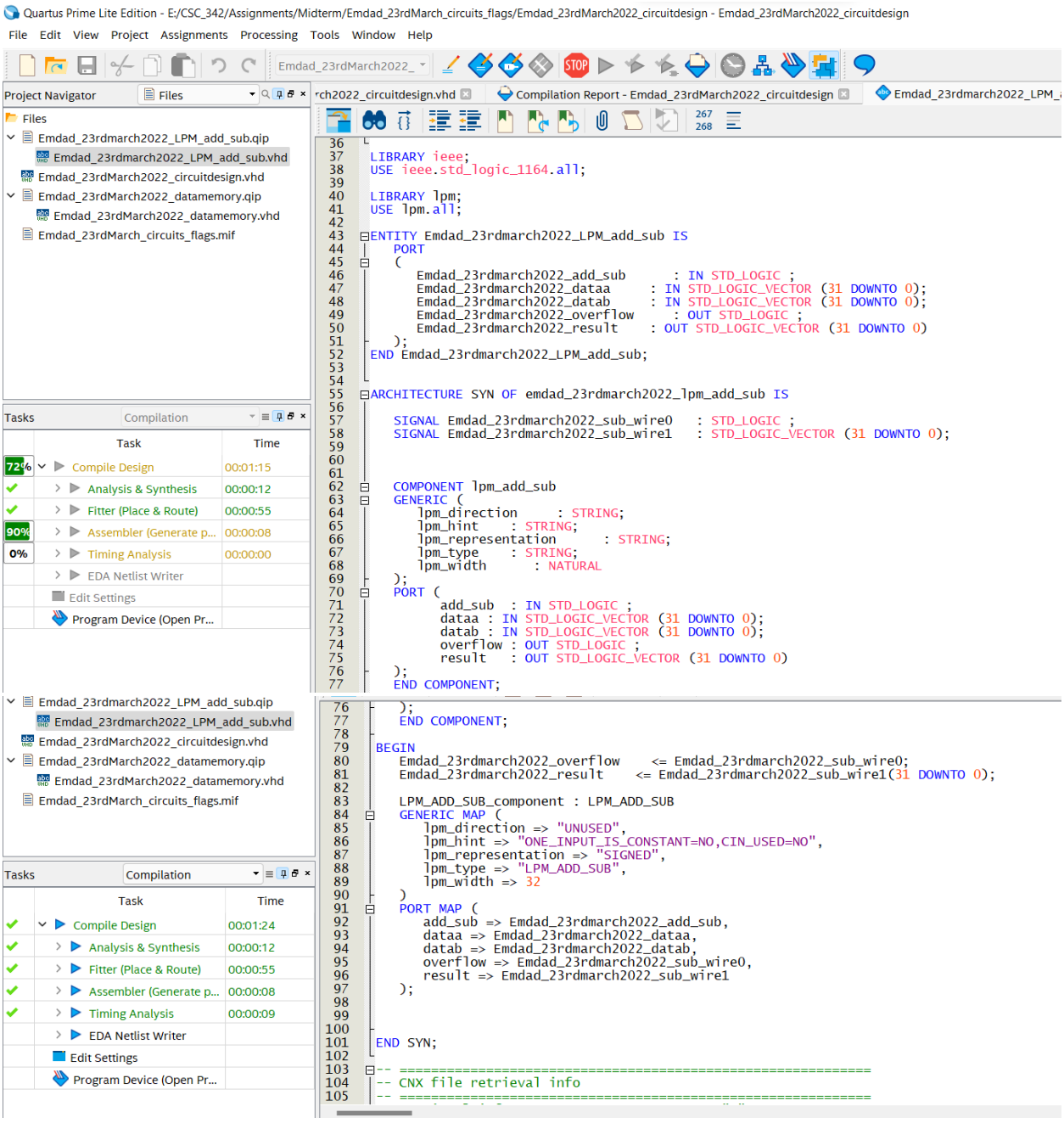

Figure 52: lpm add/sub VHDL code

In figure 53, we can see the VHDL code for circuit design.

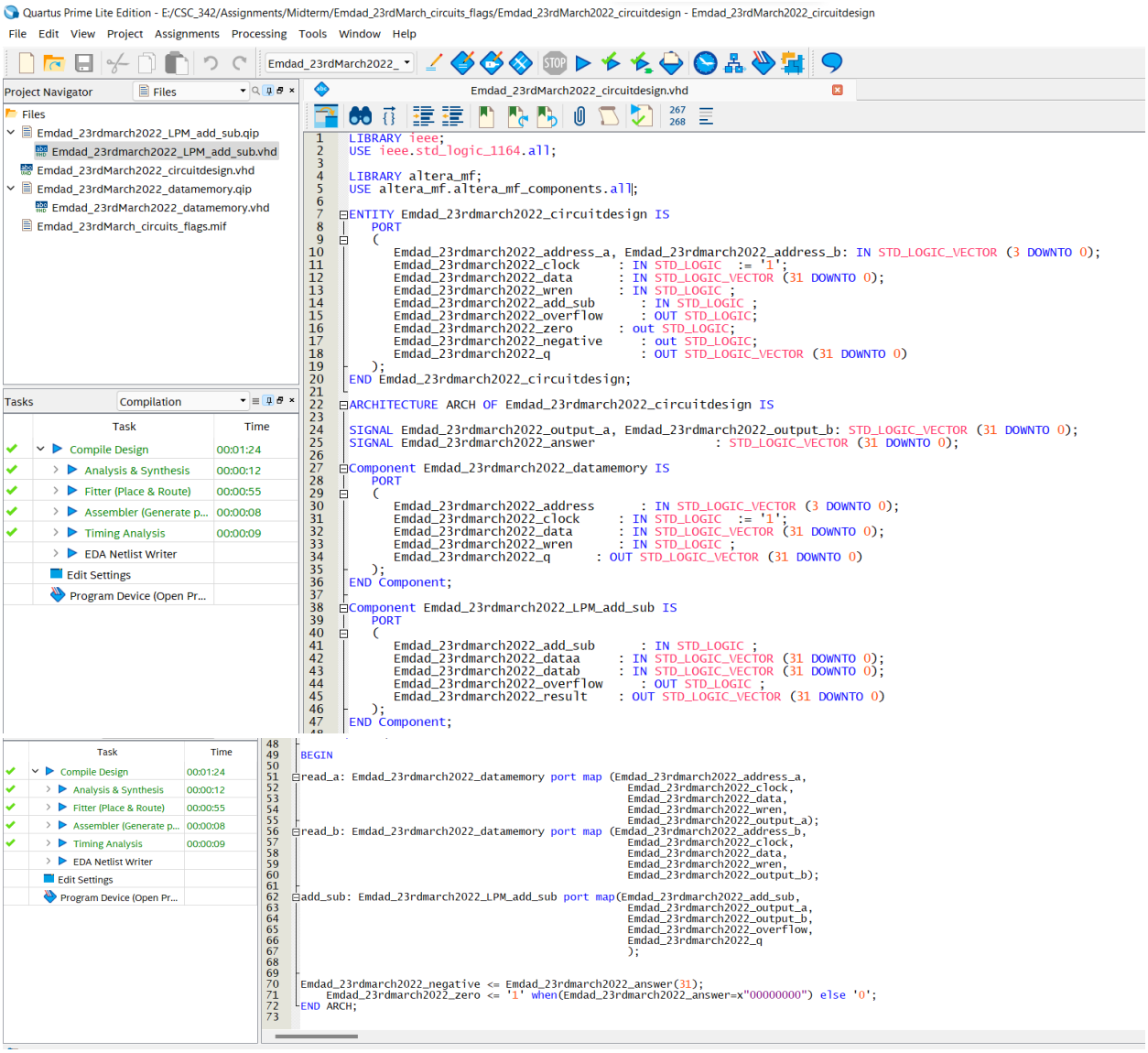

Figure 53: VHDL code for circuit design

In figure 54, we can see it compiled successfully.

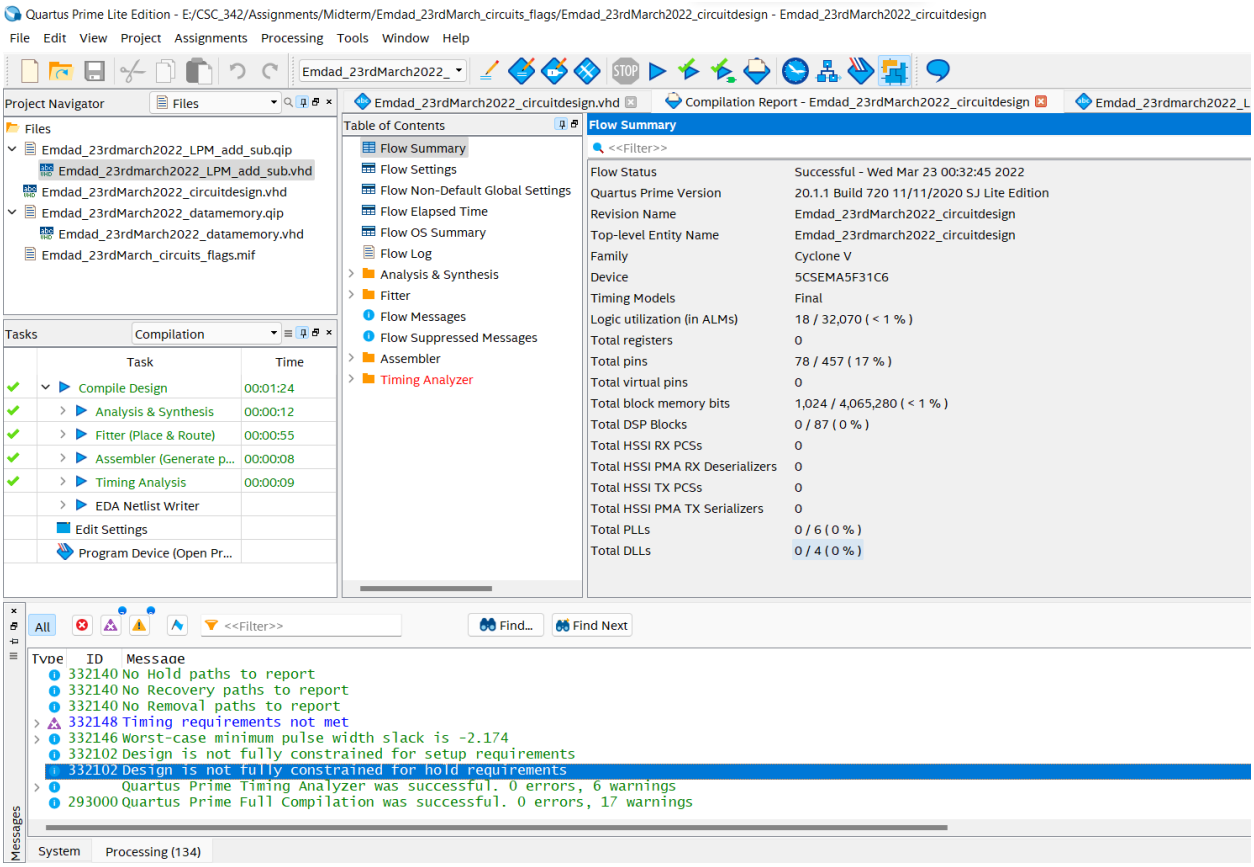

Figure 54: compilation report for circuit design

Simulation:

Assignment 1:

i. Data Memory:

In figure 55, we can see waveform created from the lpm RAM-1 port 32x16 modules.

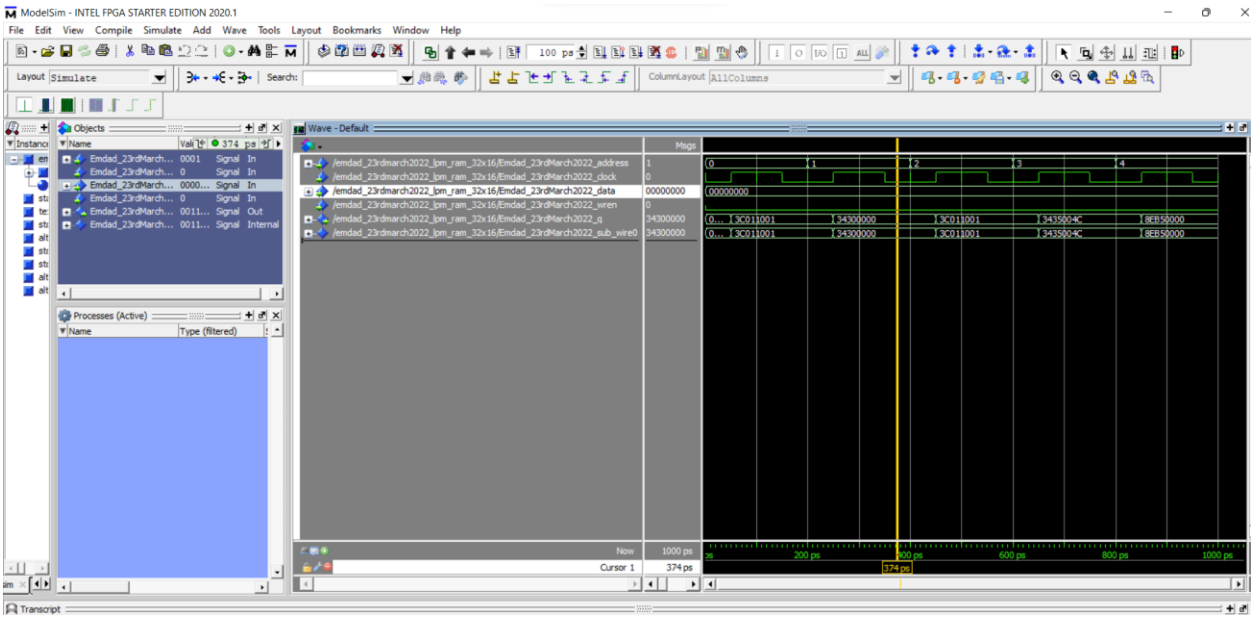

Figure 55: Simulation for Data Memory

The data memory waveform is for single port RAM. For this part of the assignment, we were instructed to input 16 words and 32 bits word size. We used MIF files to see if the code can read the data from the MIF file and display in the simulation. In q, each block represents unique address value from the MIF file. I ran the code every 100ps and data value was set to 0. Every 200ps, I changed the values of address 0,1,2,3, 4 respectively. Overall, we can see the simulation shows 5 32-bits word which was read from the MIF file.

ii. Instruction Memory:

In figure 56, we can see the waveform generated from the lpm RAM-1 port 32x32 modules.

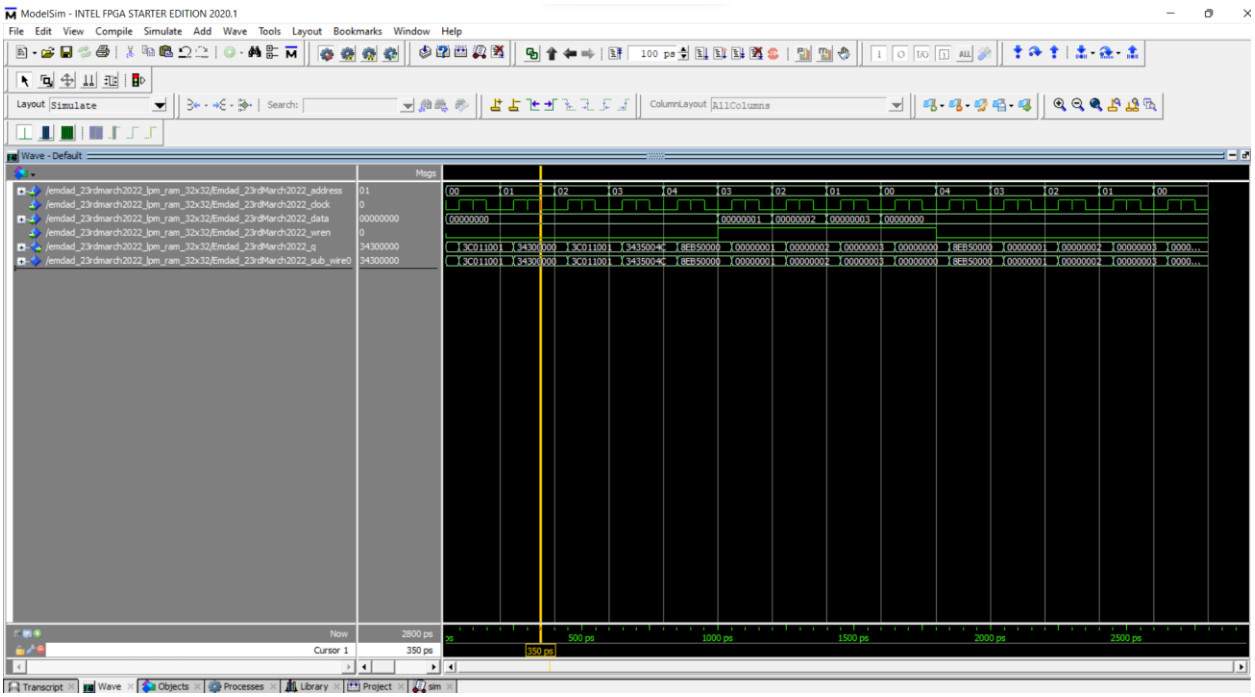

Figure 56: Simulation for Instruction Memory

The Instruction Memory waveform is for single port RAM. For this part of the assignment, we were instructed to input 32 words and 32 bits word size. We used MIF files to see if the code can read the data from the MIF file and display in the simulation. In q, each block represents unique address value from the MIF file. I ran the code every 100ps. We enabled the address by inputting 1 and it shows the data was overwritten.

iii. Dual Ported Memory Module:

In figure 57, we can see the waveform generated from the lpm RAM-2 port 32x32 modules.

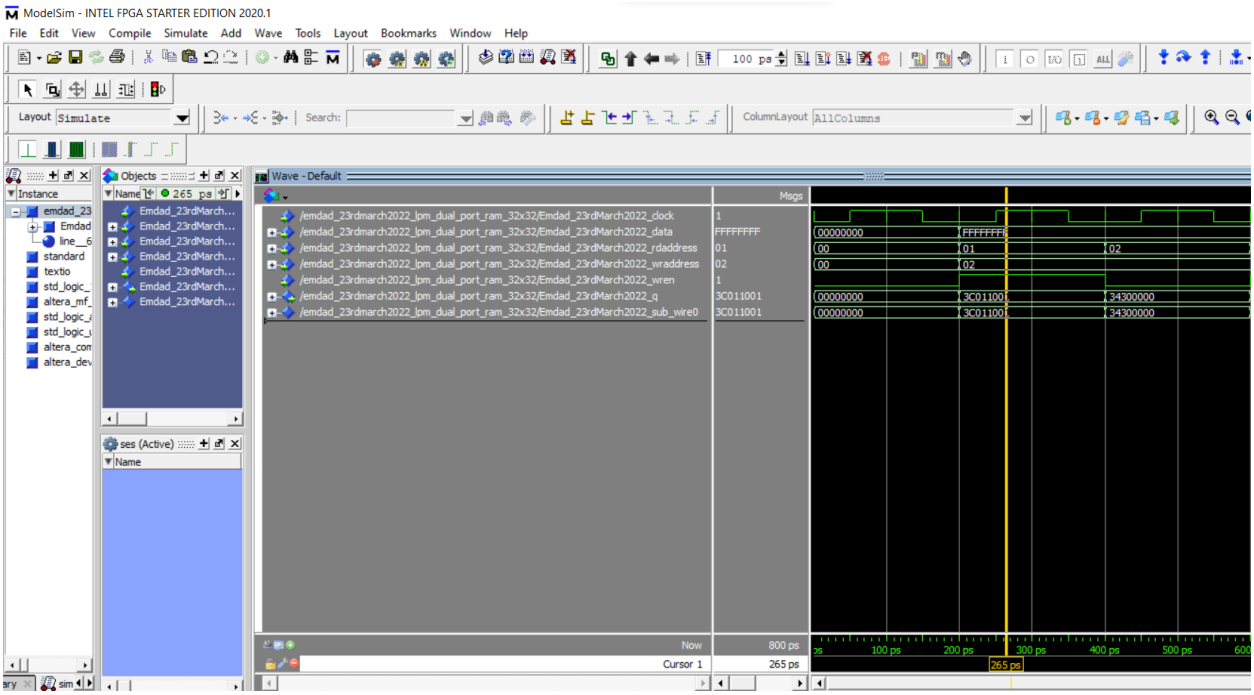

Figure 57: Simulation of Dual Ported Register File

The Dual Ported waveform is for dual port RAM. For this part of the assignment, we were instructed to input 32 words and 32 bits word size. We used MIF files to see if the code can read the data from the MIF file and display in the simulation. In q, each block represents unique address value from the MIF file. I ran the code every 100ps. We could read from address and enable writing at the same time to another address.

Assignment 2:

i. 32-bit add/sub scratch:

In figure 58, we can see the waveform from the 32-bit add/sub scratch code.

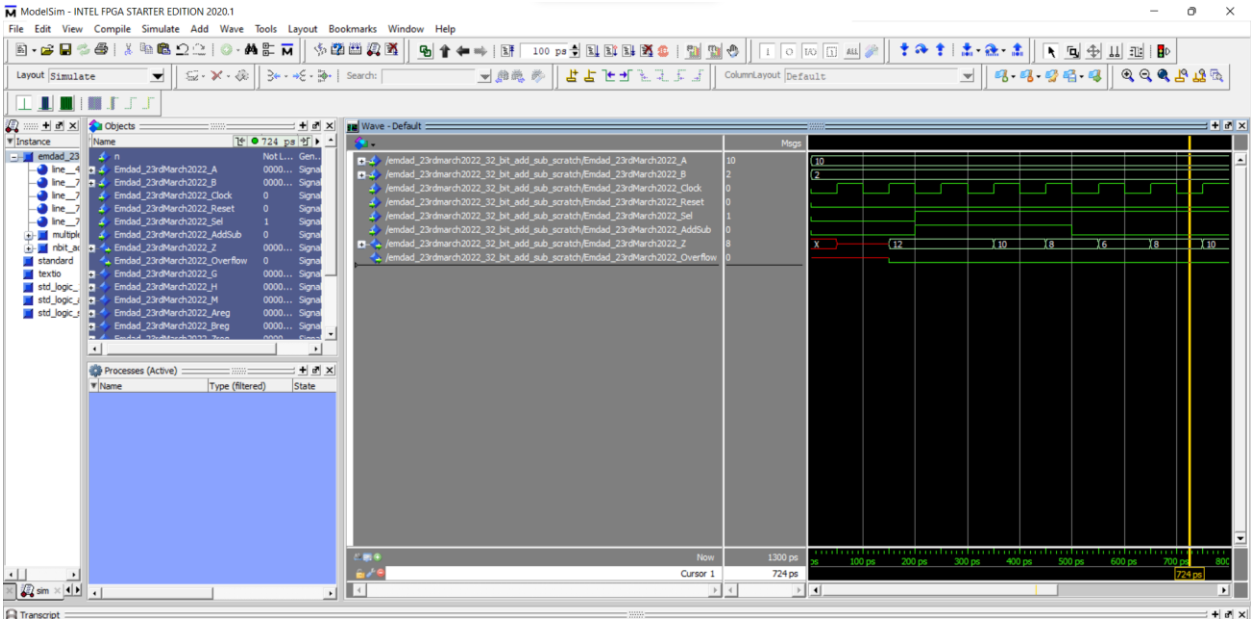

Figure 58: Simulation of 32-bit add/sub scratch code

In above waveform, A and B are the address. I ran the code every 100ps. The clock changes when its 1 and rising.

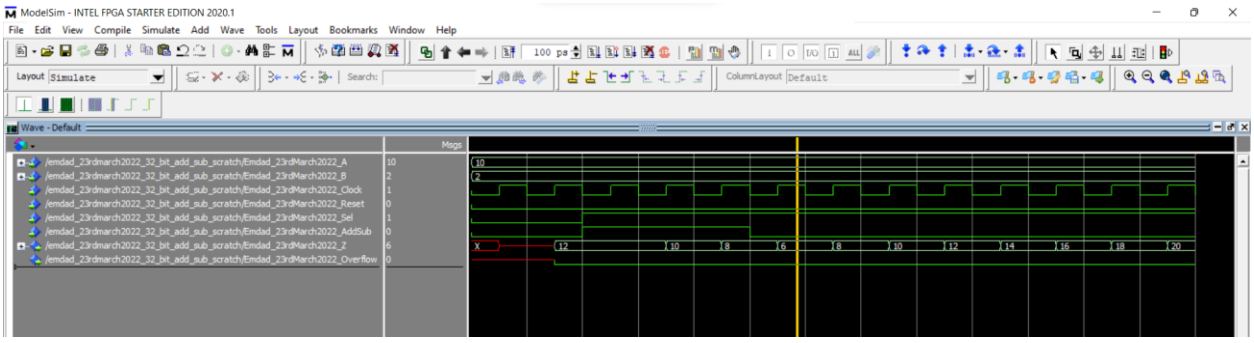

Figure 59: Simulation of 32-bit add/sub scratch code(cases)

In the above picture, we can see how the value of Z went down to up again. When we set the value 1 to sel, our add/sub will act as an accumulator and triggered. When add/sub is 1 it does subtract and when its 0 it adds both addresses.

ii. 32-bit add/sub LPM module:

In figure 60, we can see the waveform generated from the 32-bit add/sub LPM module.

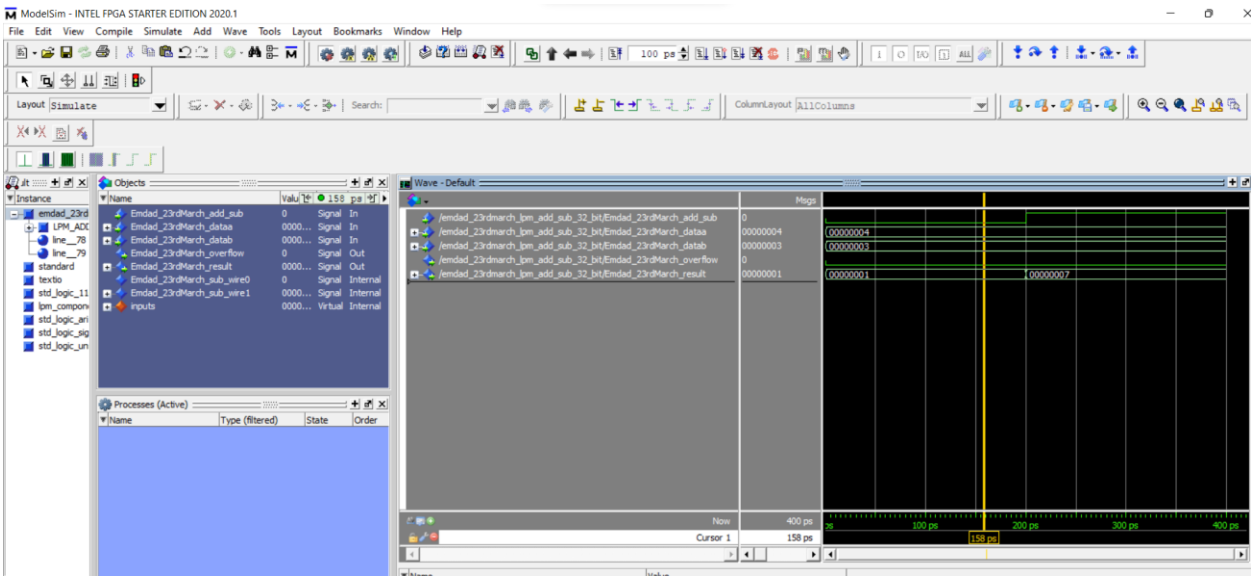

Figure 60: Simulation of 32-bit add/sub LPM module.

In the above simulation we can see that we have two inputs for data entry. I manually input the date in simulator. When add/sub trigger is 0, it subtracts the two inputs and if it at 1, it adds the values. We can see when I did add/sub to 1, it added the inputs and results 7. Overflow will only occur if the result gets out-of-range.

iii. Circuit Design using add/sub:

In figure 61, we can see waveform from the circuit design.

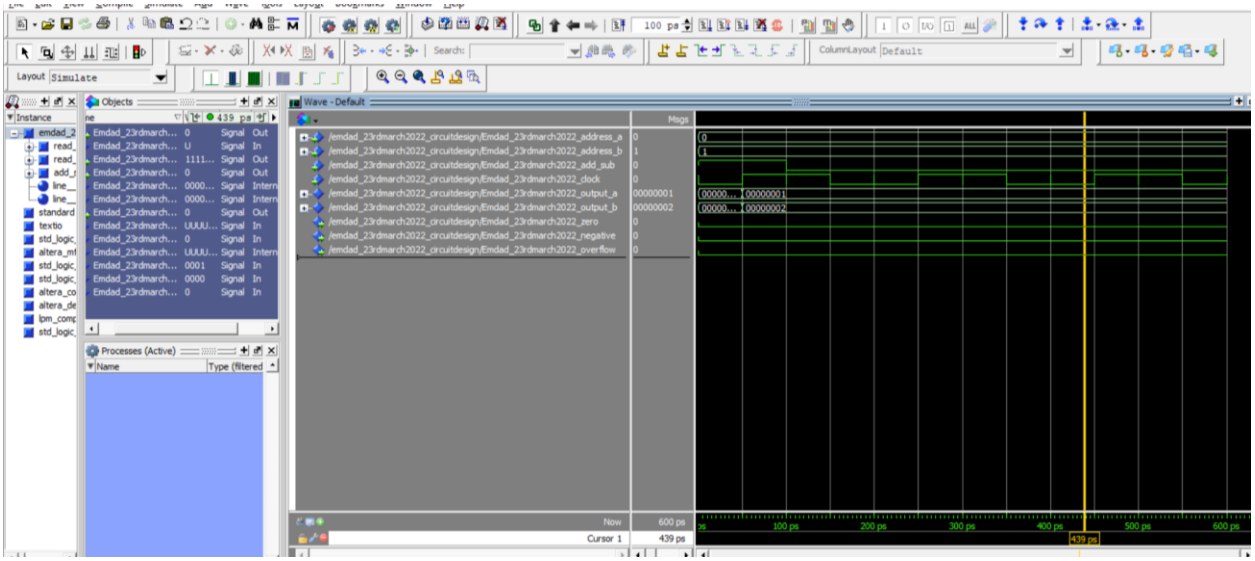

Figure 61: Simulation of circuit design

In this circuit design project, we used both RAM-1 PORT and LPM add/sub modules. We added them in our circuit file as a component. The lpm add/sub can not read data from MIF file so we had to use data memory to read data from the MIF file. We have two inputs, address_a and address_b. We keep them as for the whole timeline. We use them to do add/sub and give the output to answer. The clock has a falling/rising every 100ps. The zero flag means if the answer is 0. When the answer is negative, it raises the negative flag and overflow flag is when the answer is out-of-range.

Conclusion:

In this assignment I learned how to design and understand the specifications of VHDL array and LPM (library of parameterized modules) modules using Quartus and Modelsim. It was a good practice on Modelsim for circuit simulation. This lab is to learn how data is written and read from RAM and using MIF file. This helped me understand add/sub. I can also use LPM and direct VHDL code to help you understand the logics and how the VHDL language works. 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.