CS 342000 / CS343000
Instructor: Professor Izidor Gertner
Spring 2022
Azwad Shameem, 3/23/2022
Midterm Lab Project

I will neither give nor recieve unathorized assistance on this exam. I will only use one computing device to perform this test

Azwad Shameem

Table of Contents

Objective

The objective of this lab to create SRAM circuit in VHDL and use it in conjunction with a AddSub circuit, so that we can read from a file and use the numbers being read and either add or subtract them and output the result. In addition, with the SRAM circuits we had to make Data Memory, Instruction Memory and a Dual Ported Register File which allowed us to show simulations of read and write from a mif file and instructions from MIPS using Mars.

Description of Specifications and Functionality

Specifications:

1. Shameem_03_23_22_DataMemory:

Shameem_03_23_22_address: Address of the value at the .mif file

Shameem_03_23_22_clock: The clock signal

Shameem_03_23_22_data: 32-bit binary input

Shameem_03_23_22_wren: read, write signals

Shameem_03_23_22_q: Output of circuit

2. Shameem_03_23_22_InstructionMemory:

Shameem_03_23_22_address: Address of the instruction

Shameem_03_23_22_clock: The clock signal

Shameem_03_23_22_data: 32-bit binary input

Shameem_03_23_22_wren: read, write signals

Shameem_03_23_22_q: Output of circuit

3. Shameem_03_23_22_DualPortedRegisterFile:

Shameem_03_23_22_data: 32-bit binary input

Shameem_03_23_22_clock: The clock signal

Shameem_03_23_22_raddress: Address to be read from

Shameem_03_23_22_waddress: Address to be written to

Shameem_03_23_22_wren: read, write signals

Shameem_03_23_22_q: Output of circuit

4. Shameem_03_23_22_AddSubScratchCircuit:

Shameem_03_23_22_AddressA: Address to be read from

Shameem_03_23_22_AddressB: Address to be read from

Shameem_03_23_22_clock: The clock signal

Shameem_03_23_22_reset: Input 1 to reset values to 0 or 0 to do nothing

Shameem_03_23_22_sel: Input 1 to start accumulating

Shameem_03_23_22_addsub: Input 0 to add or input 1 to subtract

Shameem_03_23_22_FinalResult: Output of the addition or subtraction

Shameem_03_23_22_Z: 1 if Shameem_03_23_22_FinalResult is equal to zero

Shameem_03_23_22_N: 1 if Shameem_03_23_22_FinalResult is negative

Shameem_03_23_22_O: overflow output

5. Shameem_03_23_22_AddSubScratch:

Shameem_03_23_22_A: 32-bit binary to be added

Shameem_03_23_22_B: 32-bit binary to be added

Shameem_03_23_22_clock: The clock signal

Shameem_03_23_22_reset: Input 1 to reset values to 0 or 0 to do nothing

Shameem_03_23_22_sel: Input 1 to start accumulating

Shameem_03_23_22_addsub: Input 0 to add or input 1 to subtract

Shameem_03_23_22_Z: Output of circuit

Shameem_03_23_22_O: overflow output

6. Shameem_03_23_22_LPMAddSubCircuit:

Shameem_03_23_22_AddressA: Address to be read from

Shameem_03_23_22_AddressB: Address to be read from

Shameem_03_23_22_clock: The clock signal

Shameem_03_23_22_reset: Input 1 to reset values to 0 or 0 to do nothing

Shameem_03_23_22_sel: Input 1 to start accumulating

Shameem_03_23_22_addsub: Input 1 to add or input 0 to subtract

Shameem_03_23_22_Z: Output of circuit

Shameem_03_23_22_O: overflow output

7. Shameem_03_23_22_LPMAddSub:

Shameem_03_23_22_addsub: Input 1 to add or input 0 to subtract

Shameem_03_23_22_A: 32-bit binary to be added

Shameem_03_23_22_B: 32-bit binary to be added

Shameem_03_23_22_O: overflow output

Shameem_03_23_22_Result: Output of circuit addition or subtraction

Functionality:

Assignment 1:

Design 32-bit word Data Memory module based on LPM tutorial attached. Data memory size 16 words.

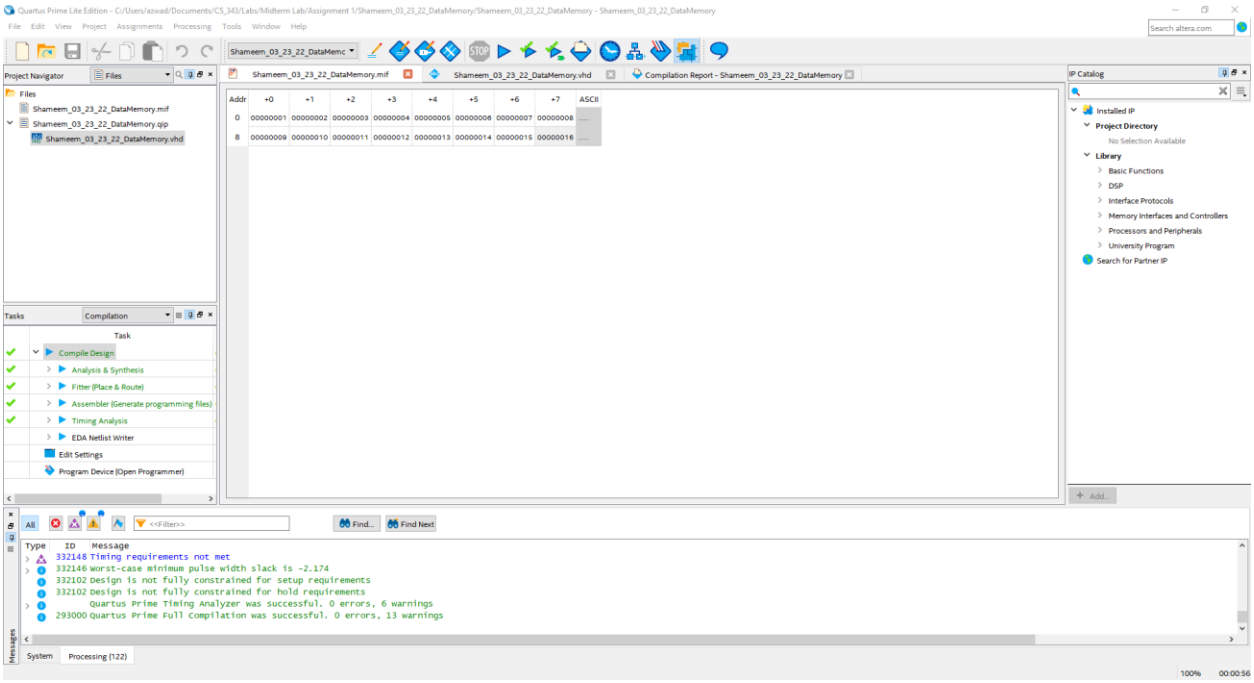


Figure 1: The mif file used in Data Memory module

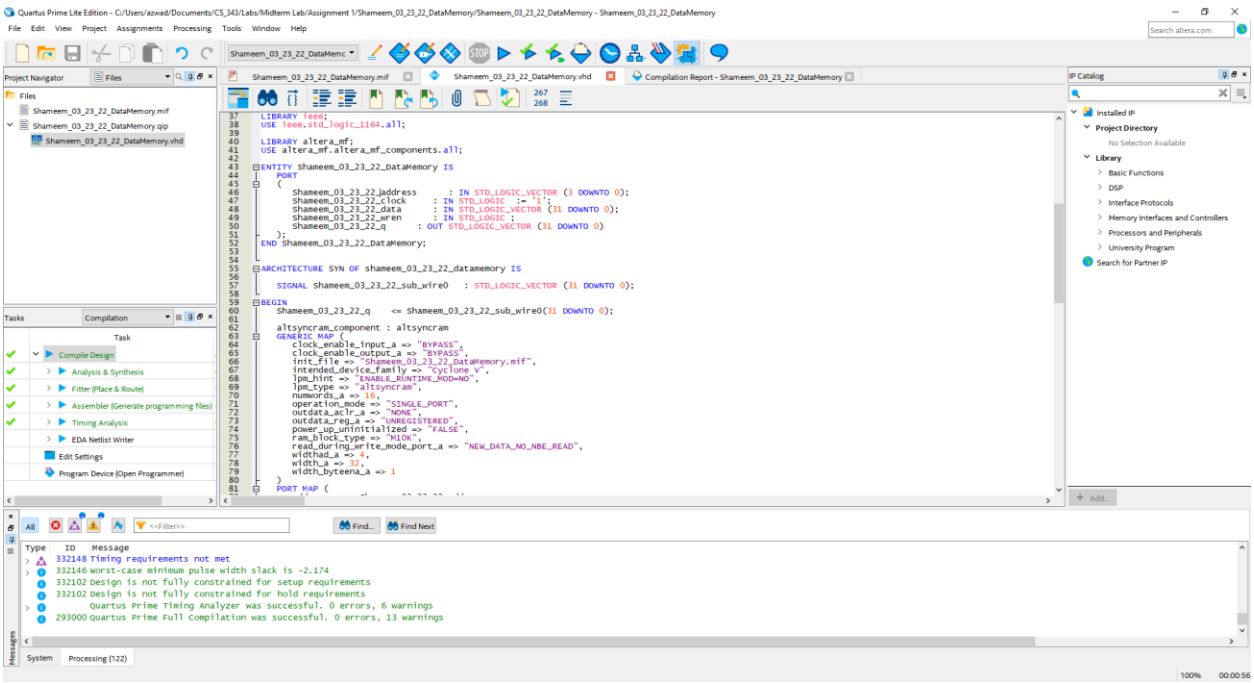


Figure 2: Data Memory module VHDL code generated for RAM 1-port from Quartus IP Catalog

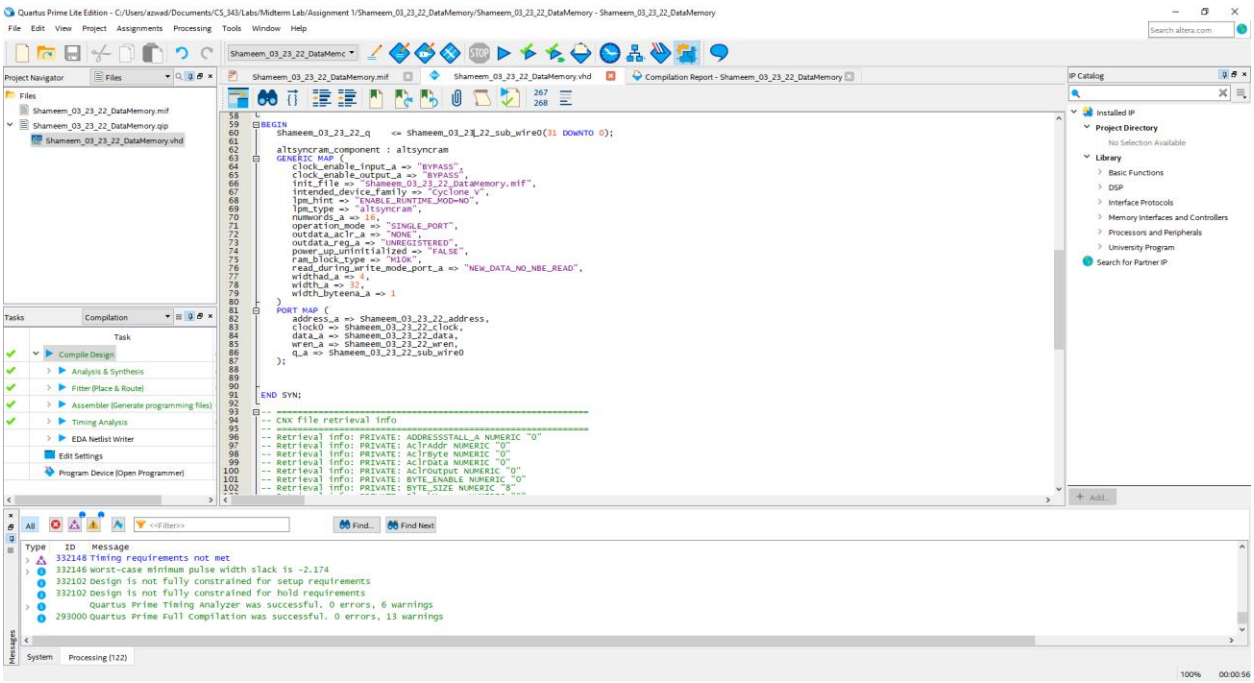


Figure 3: Data Memory module generated VHDL code continued

Explanation:

This code was generated using the Quartus IP catalog and made in accordance with the requirements for the Data Memory module which has words of 32-bit size and 16 words.

Design 32-bit word INSTRUCTION Memory module based on LPM tutorial attached. Instruction memory size 32 words.

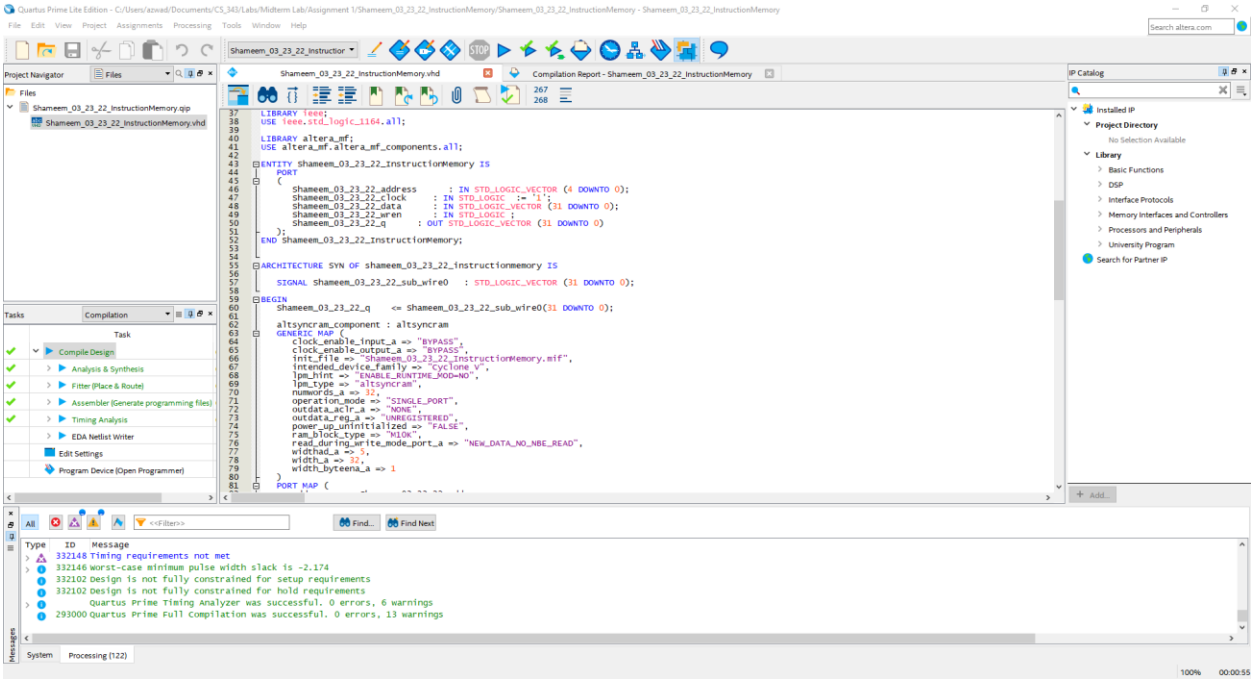


Figure 4: Instruction Memory module generated VHDL code

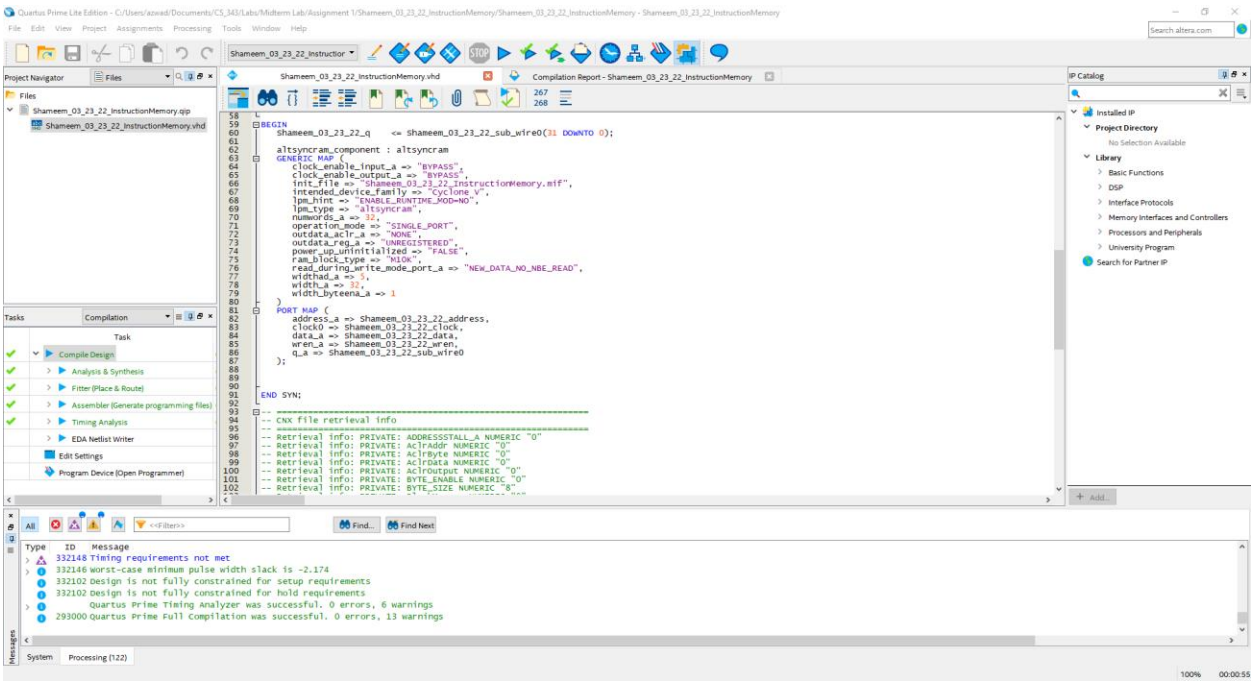


Figure 5: Instruction Memory module generated VHDL code continued

Explanation:

This code was generated using the Quartus IP catalog and made in accordance with the requirements for the Instruction Memory module which has words of 32-bit size and 32 words.

Design 32-bit register DUAL PORTED REGISTER FILE module based on 2-port RAM LPM tutorial attached. EACH register is 32 bits.

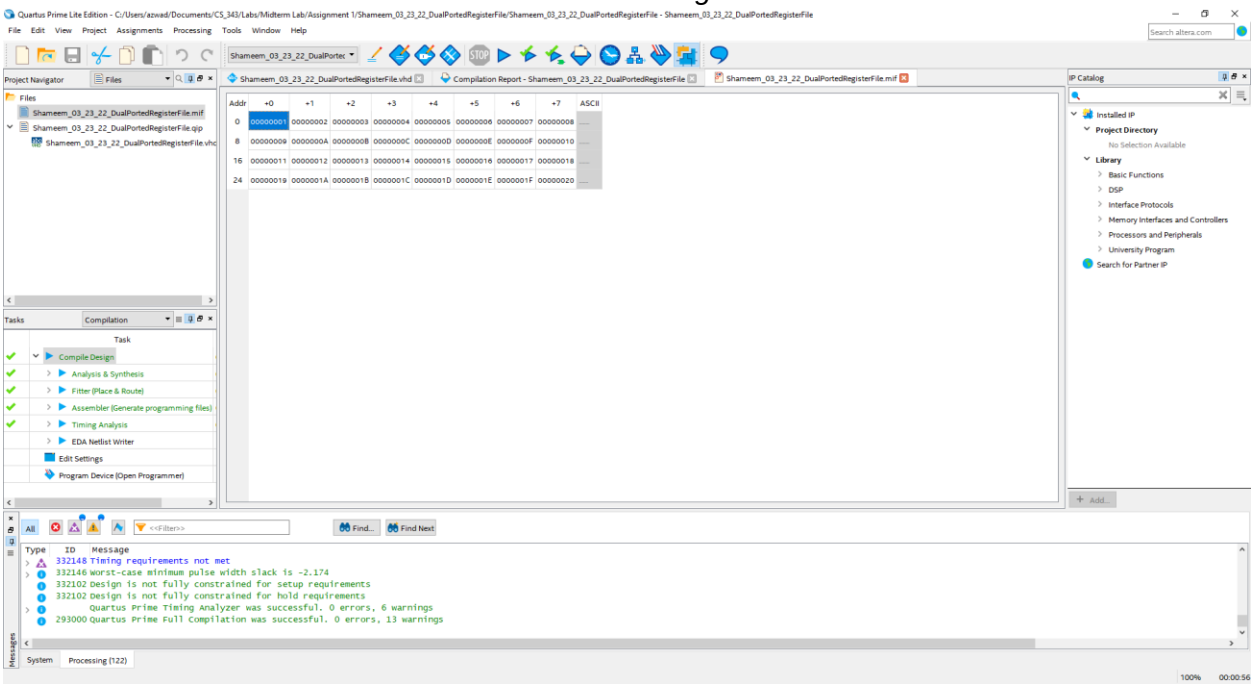


Figure 6: mif file for Dual Ported Register File module

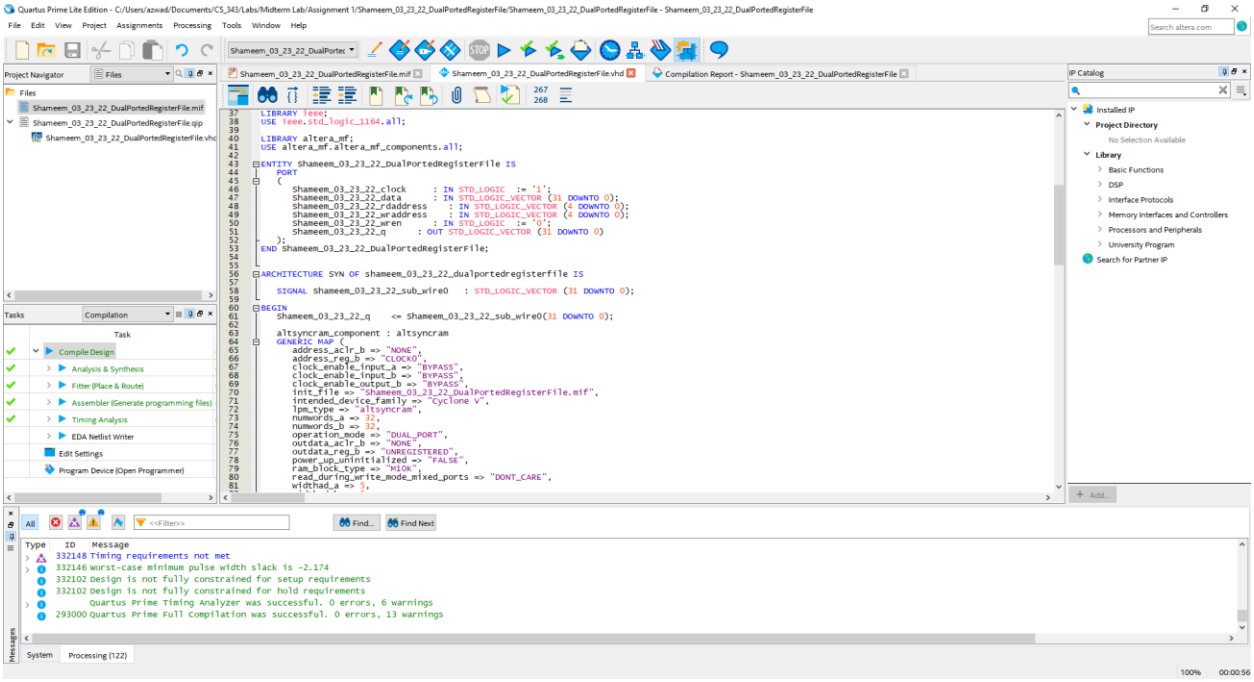


Figure 7: Dual Ported Register File module based on 2-port RAM generated VHDL code from Quartus IP Catalog

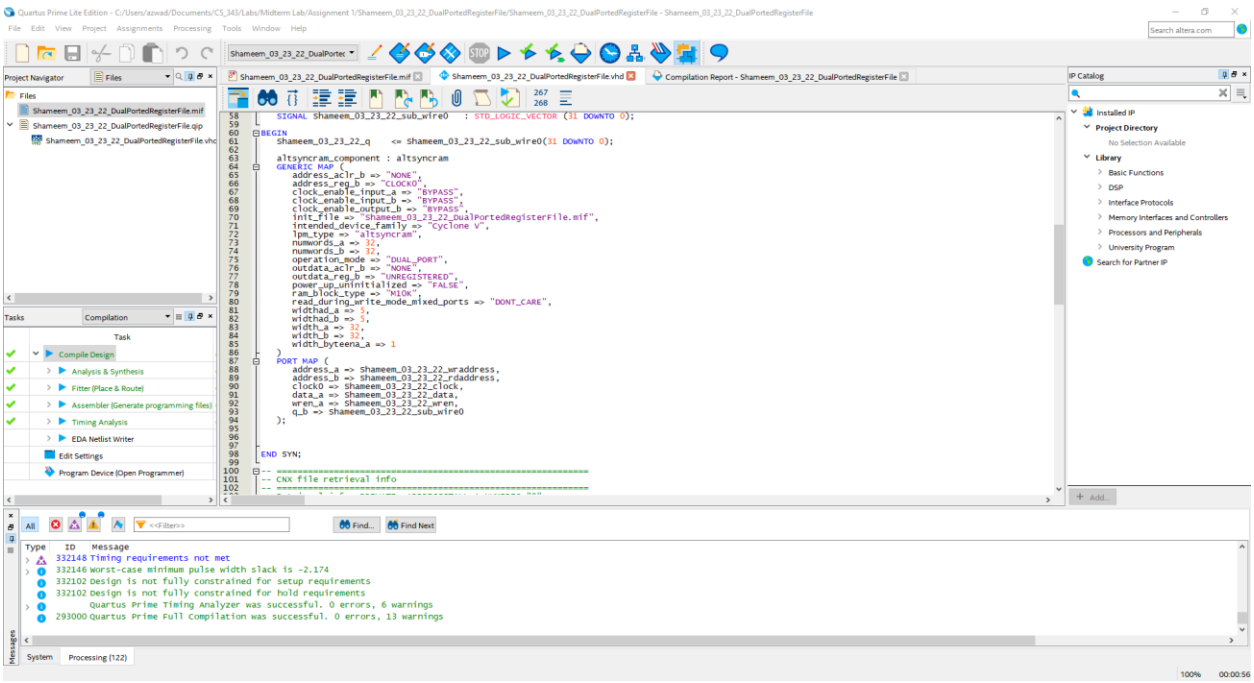


Figure 8: Dual Ported Register File module VHDL code continued

Explanation:

This code was generated using the RAM 2-Port in the Quartus IP catalog and made in accordance with the requirements for the Dual Ported Register File module which has words of 32-bit size and 32 words.

Assignment 2:

Design 32-bit Add/Sub unit as described in the seconds attached tutorial from scratch.

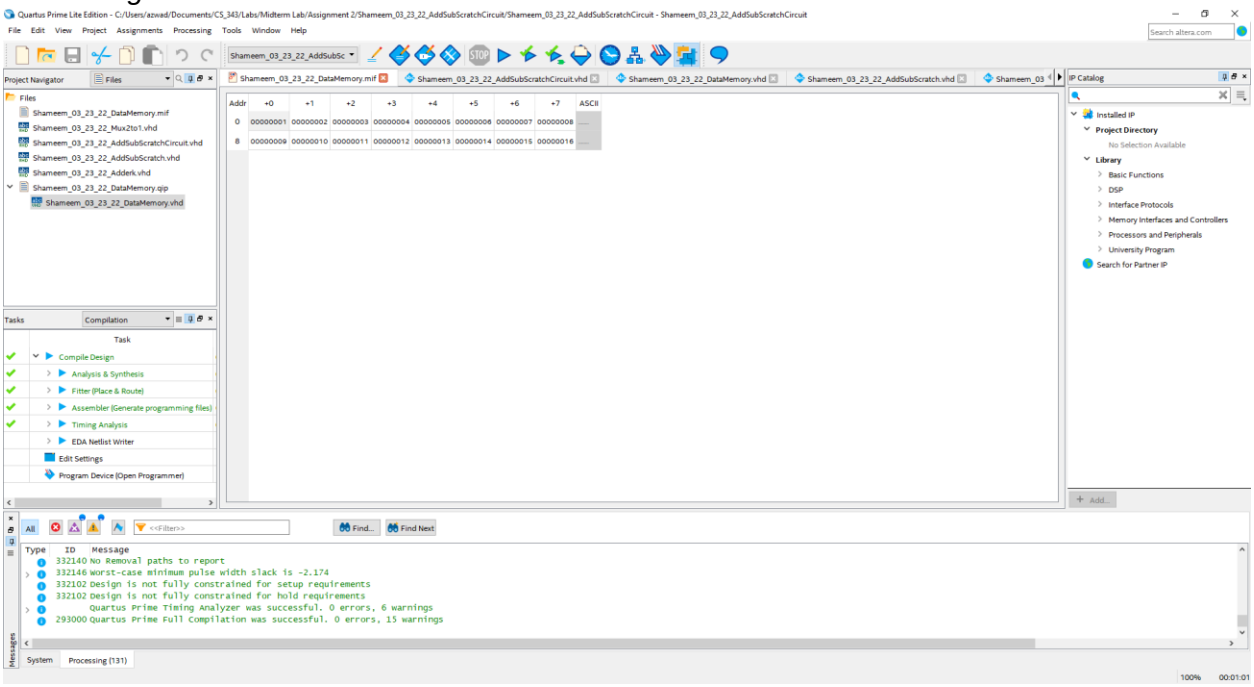


Figure 9: mif file used for the Data Memory module

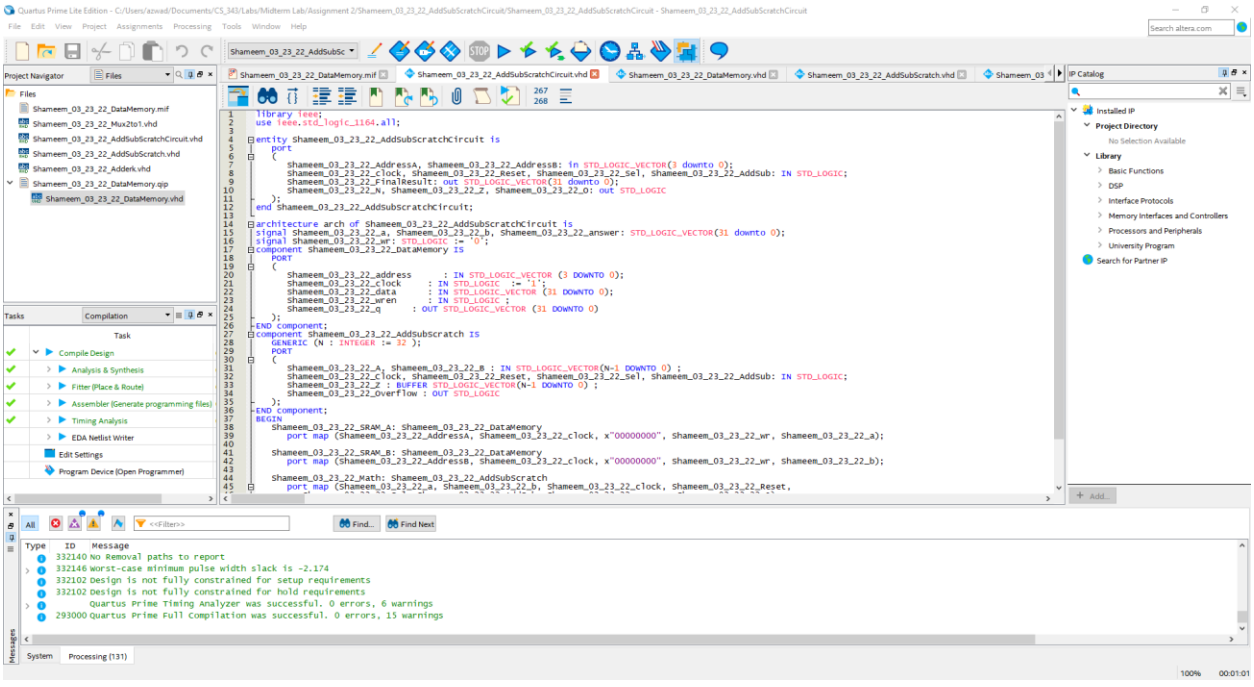


Figure 10: ScratchAddSubCircuit VHDL code. This circuit utilizes the Data Memory module and Add/Sub unit as components to load data to memory from MIF file and then use that data in order to do computation on the Add/Sub unit.

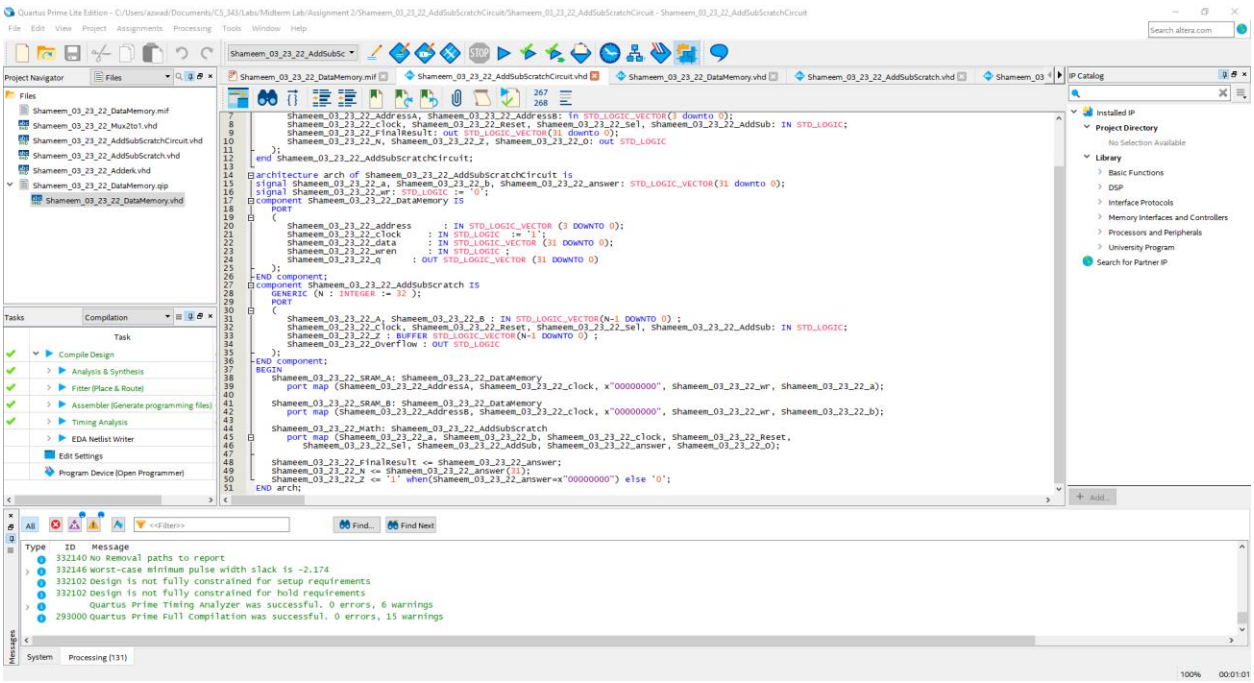


Figure 11: ScratchAddSubCircuit VHDL code continued

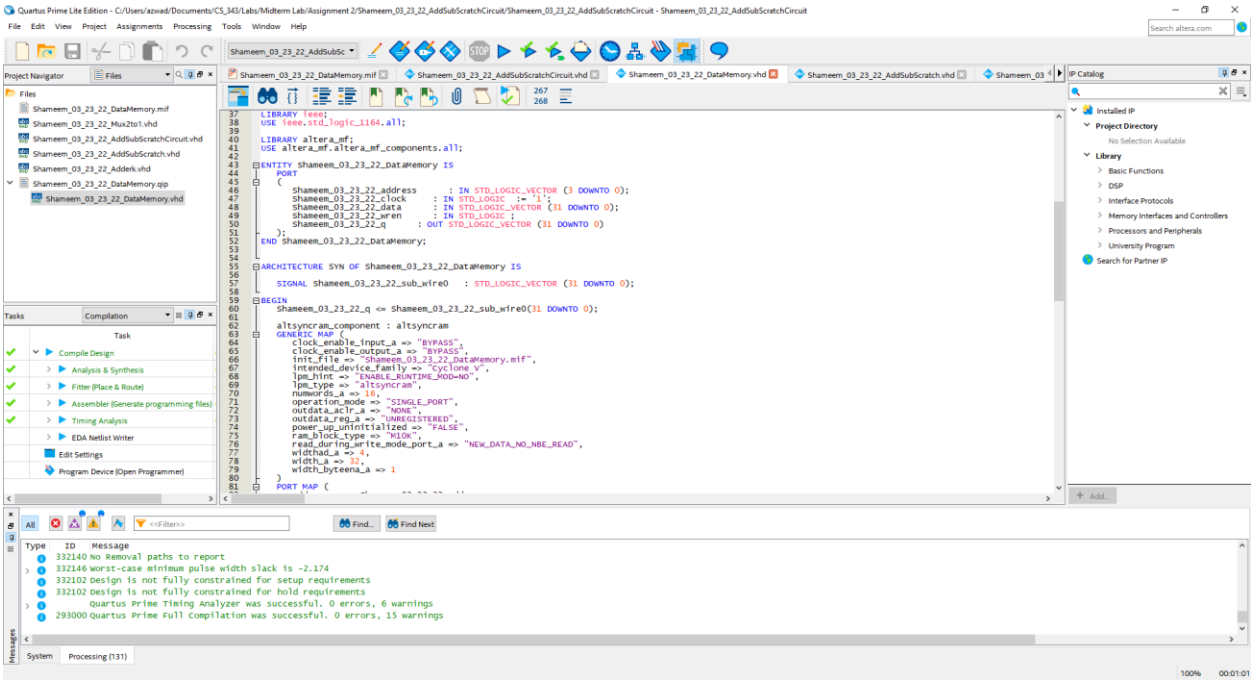


Figure 12: Data Memory module VHDL code. Data Memory is used as a component to load data into memory from MIF file.

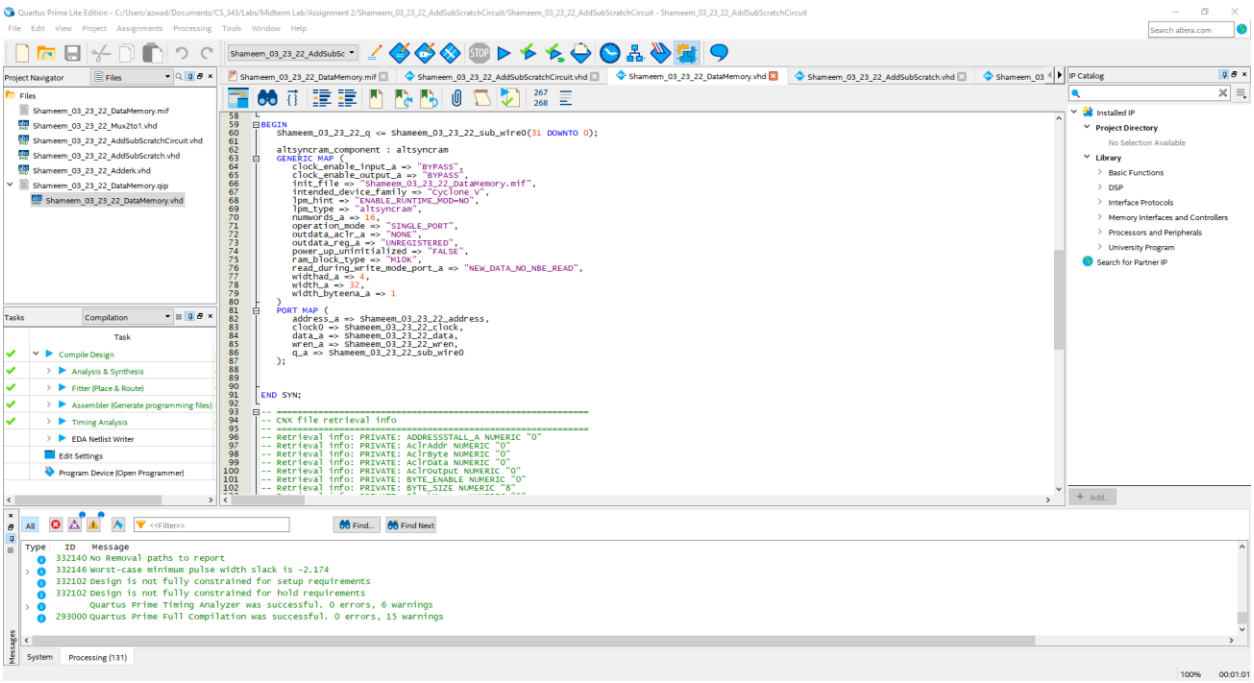


Figure 13: Data Memory module VHDL code continued

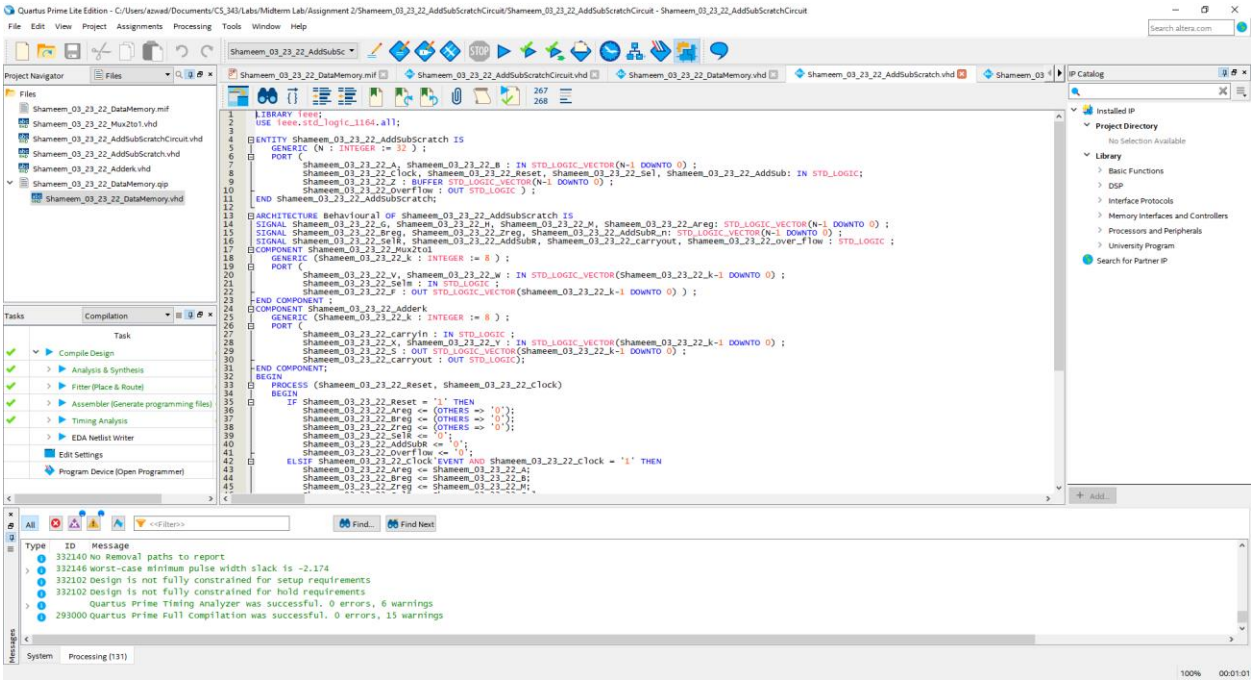


Figure 14: Add/Sub unit VHDL code. Note: This is the rewritten code from INTEL's AP note.

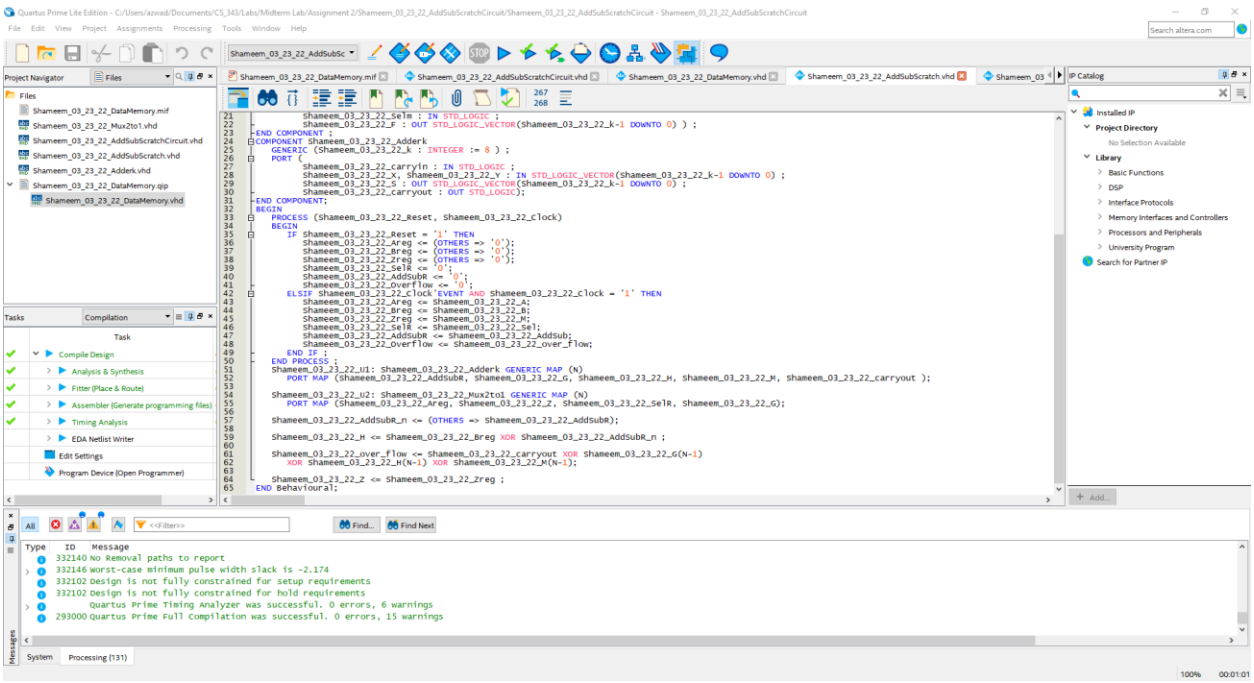


Figure 15: Add/Sub unit VHDL code continued. Note: This is the rewritten code from INTEL's AP note.

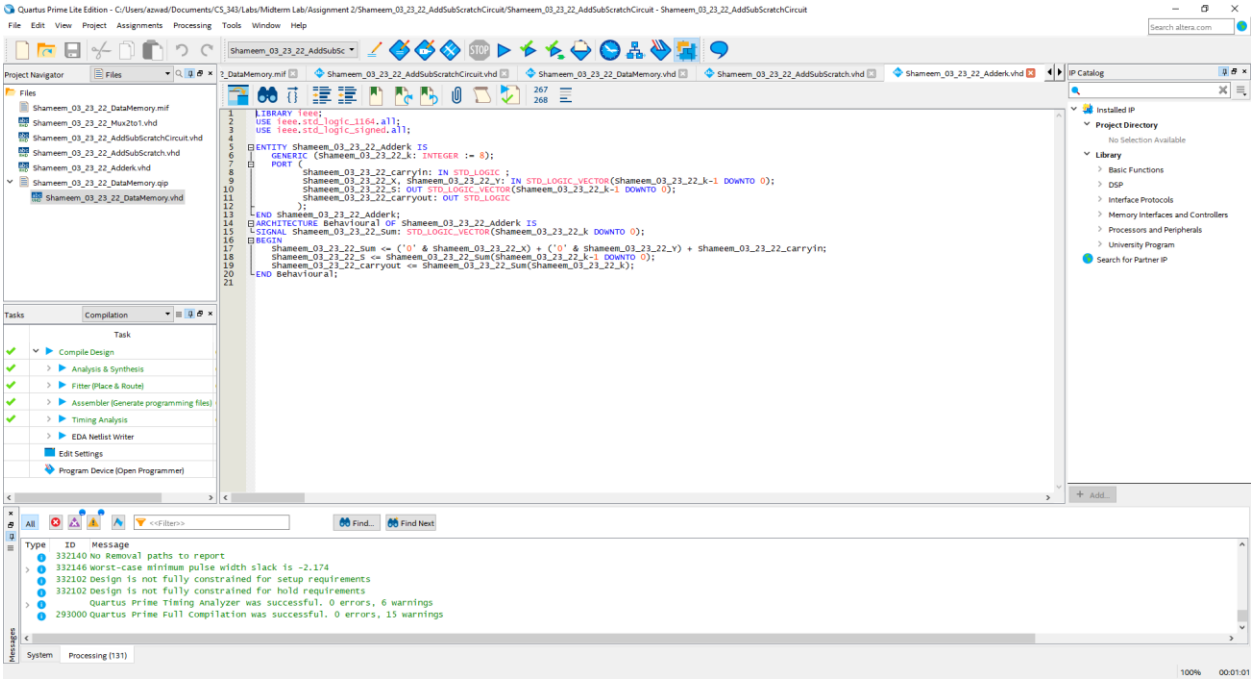


Figure 16: Adderk VHDL code utilized in Scratch Add/Sub unit.

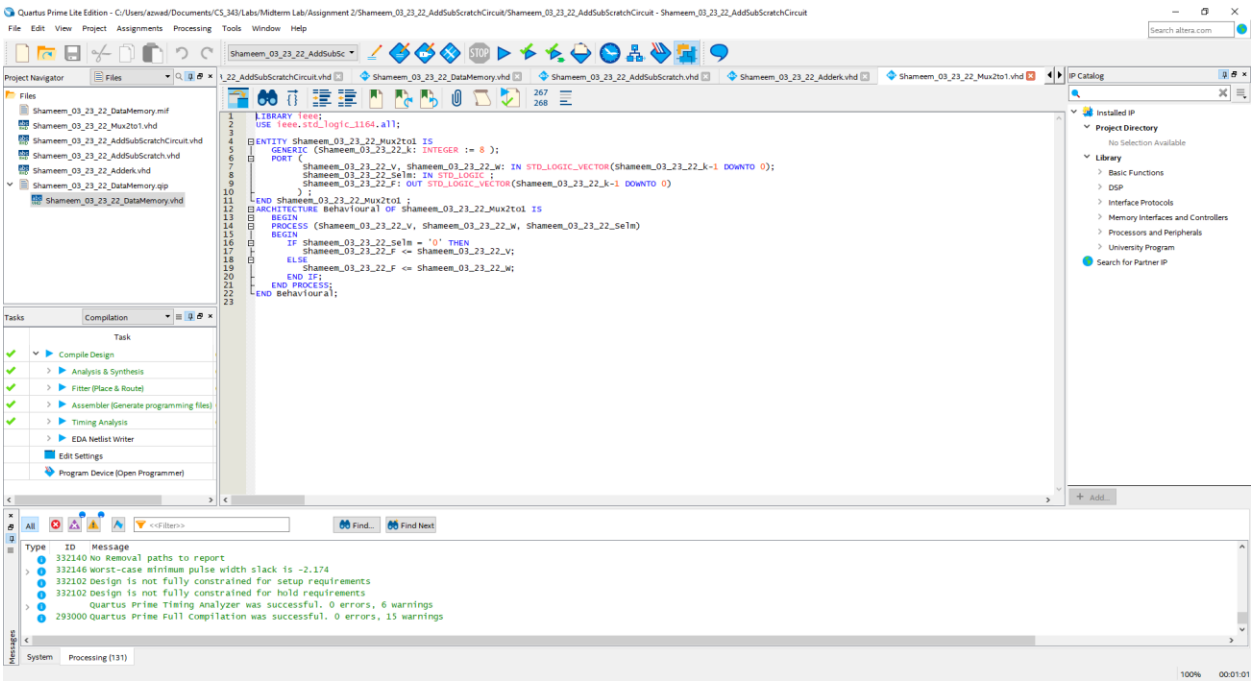


Figure 17: Mux 2:1 VHDL Code. This unit is utilized in scratch Add/Sub.

Explanation:

The ScratchAddSubCircuit is a circuit that utilizes the scratch Add/Sub unit and the Data Memory module. The ScratchAddSubCircuit first utilizes the Data Memory modules in order to load data to memory from the MIF file. Then the ScratchAddSubCircuit takes the data from memory and puts them into the scratch Add/Sub unit which either adds or subtracts and outputs the result of the operation. In addition, the scratch Add/Sub unit is the rewritten VHDL code from Intel AP's note which includes a Mux2:1 which actually gives it an option to act as an accumulator as well. The ScratchAddSubCircuit utilizes these units as components in order to output the result of the operation and negative, zero and overflow flags.

Design 32-bit Add/Sub unit as described in the seconds attached tutorial using LPM.

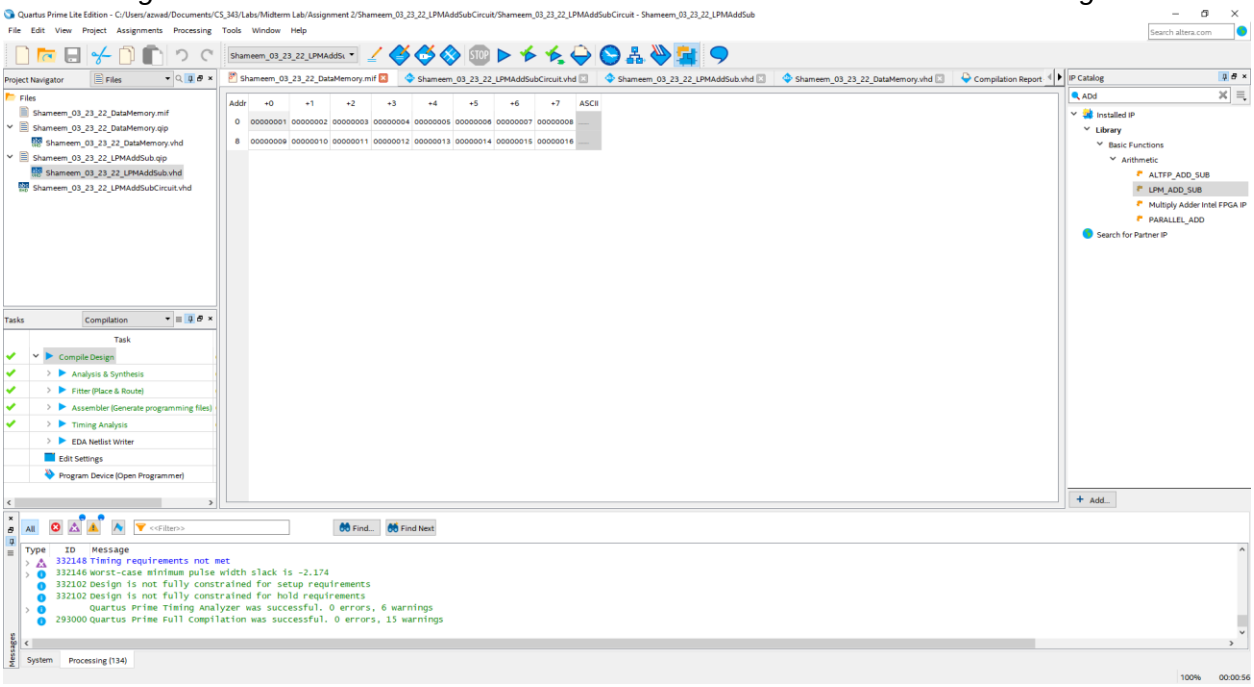


Figure 18: MIF file for the Data Memory

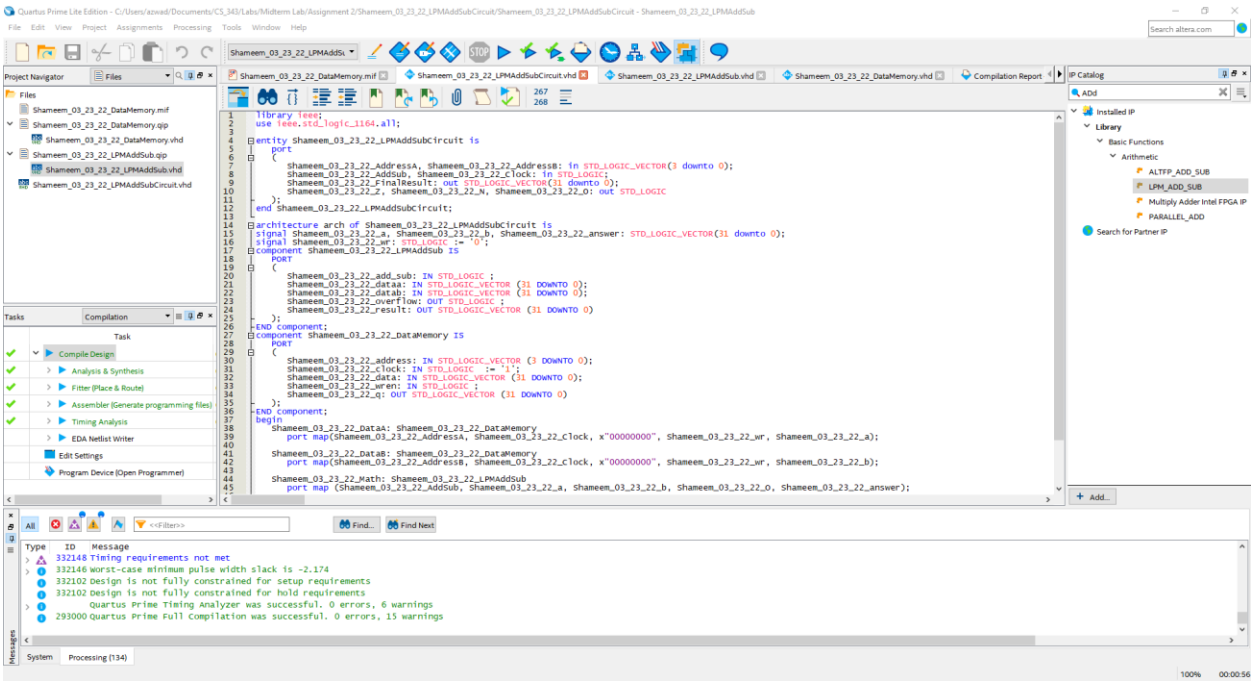


Figure 19: VHDL code for the LPMAddSubCircuit. This circuit utilizes LPMAddSub and Data Memory as components to load data to memory and then add or subtract the data to output results.

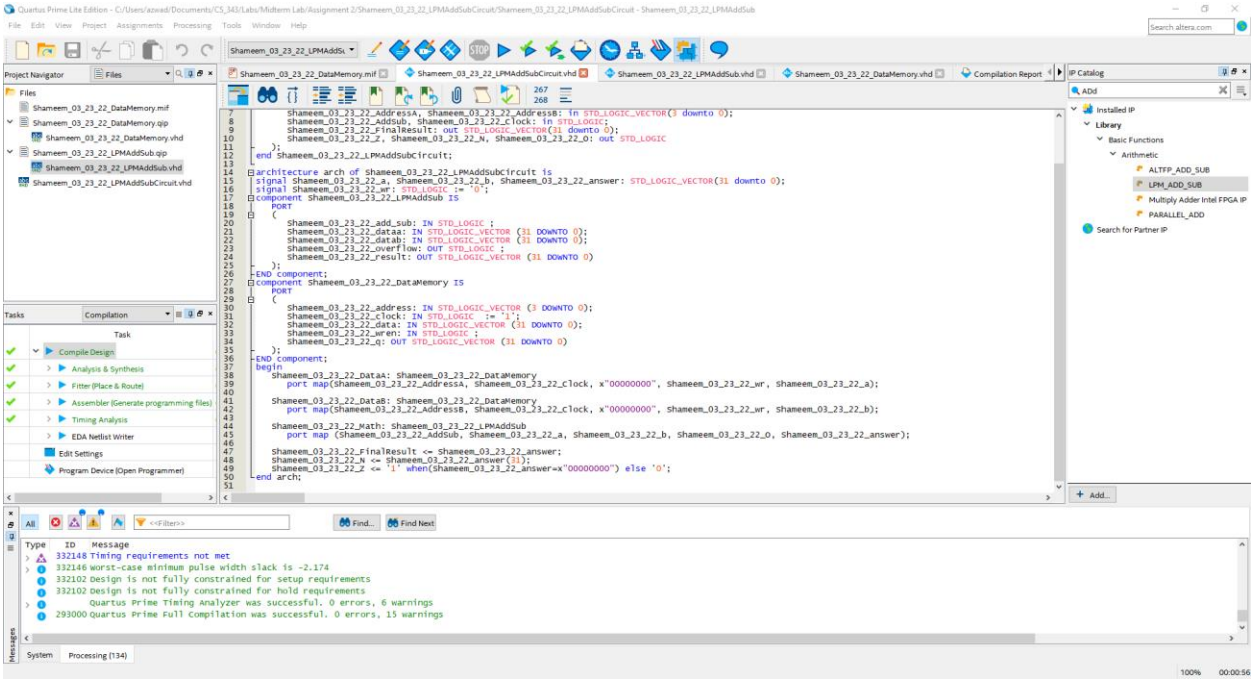


Figure 20: VHDL code for LPMAddSubCircuit continued.

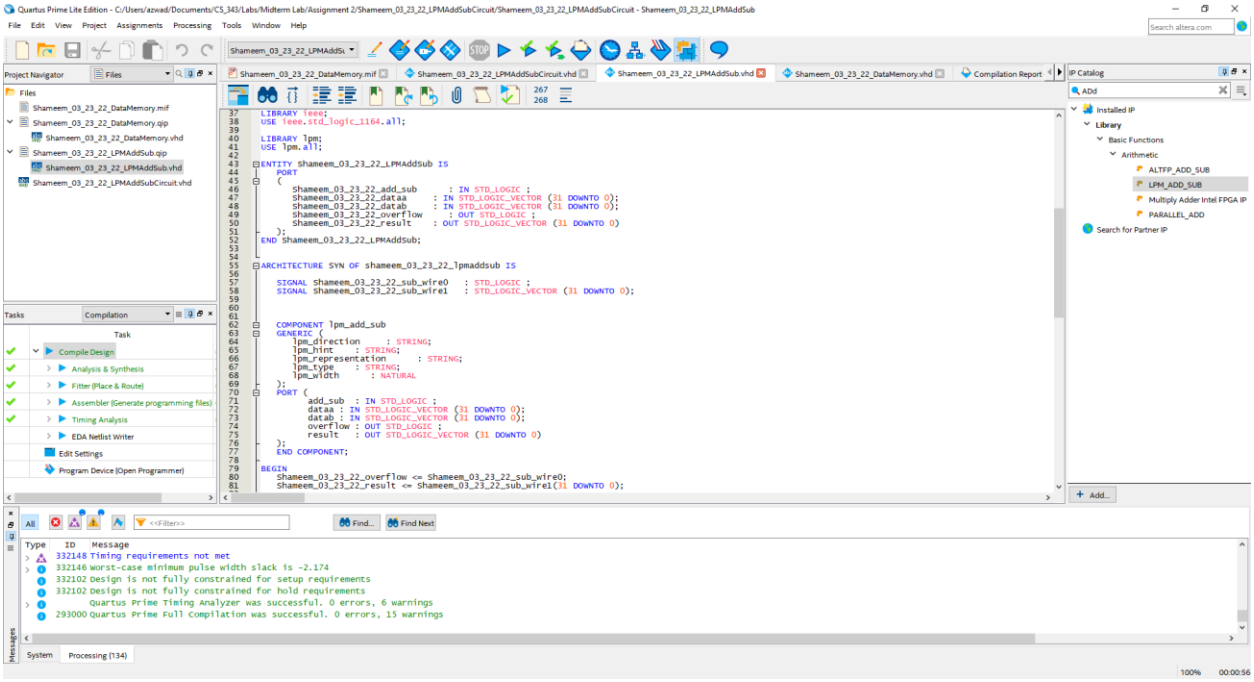


Figure 21: LPMAddSub VHDL code. This code is generated by LPM_ADD_SUB in Quartus IP Catalog.

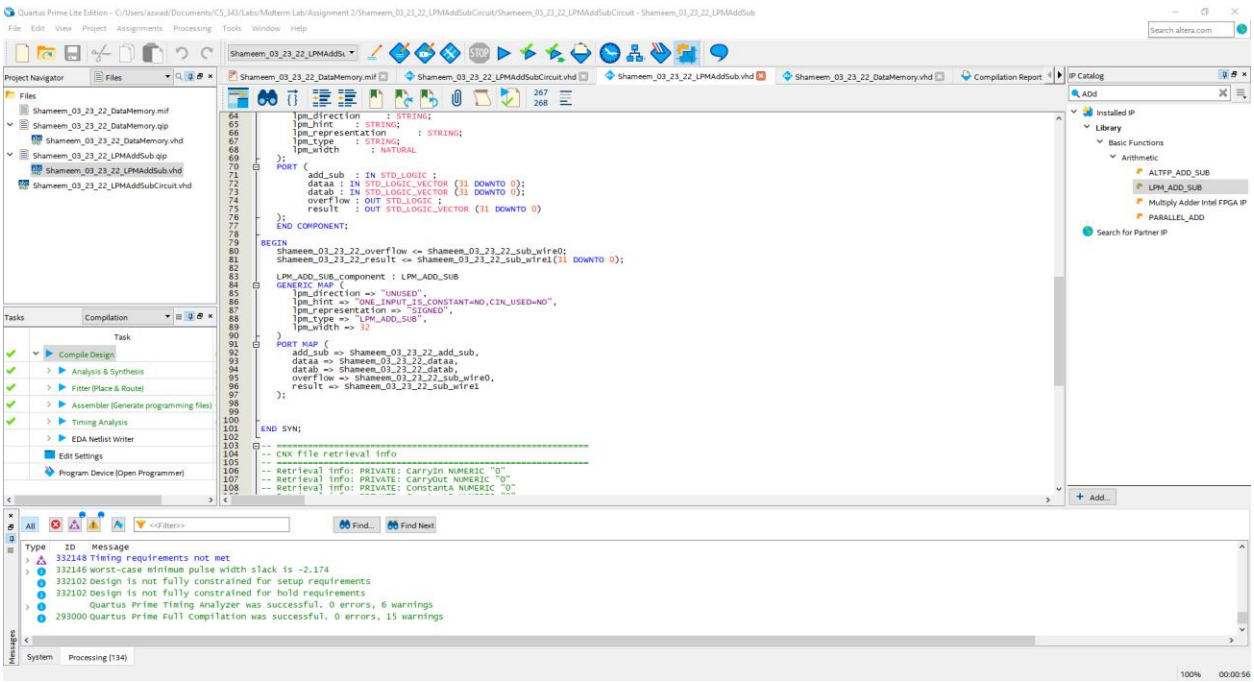


Figure 22: LPMAddSub VHDL code continued.

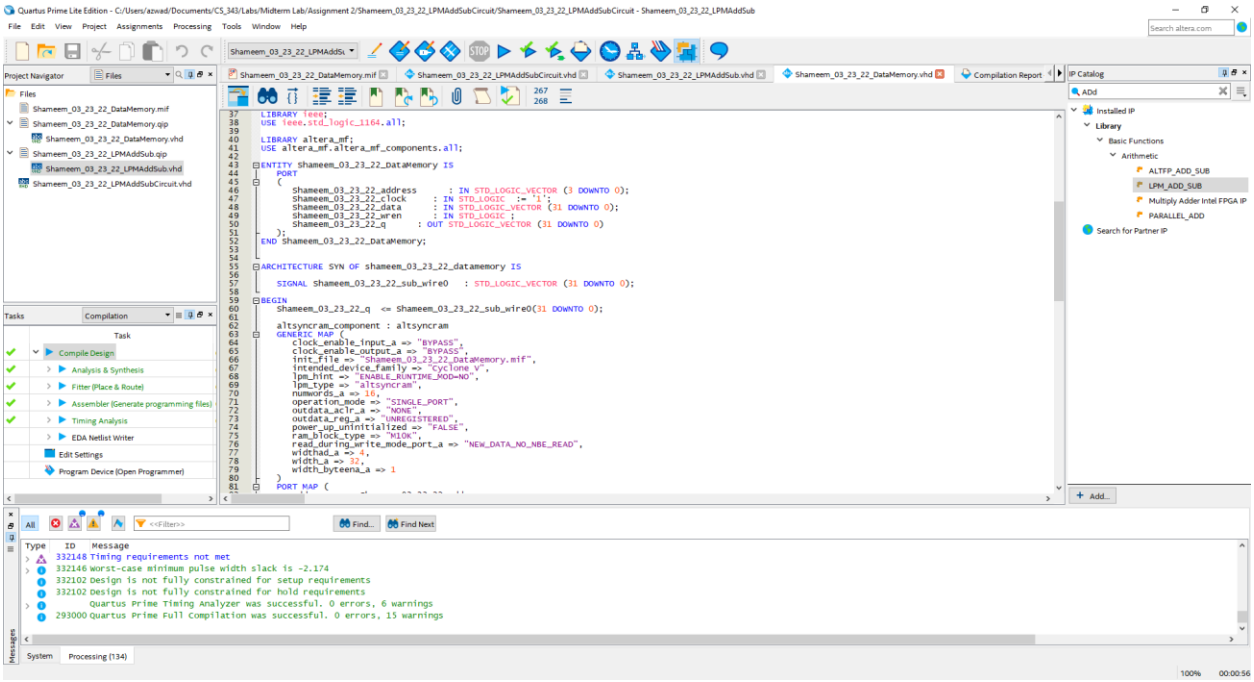


Figure 23: VHDL code for Data Memory module.

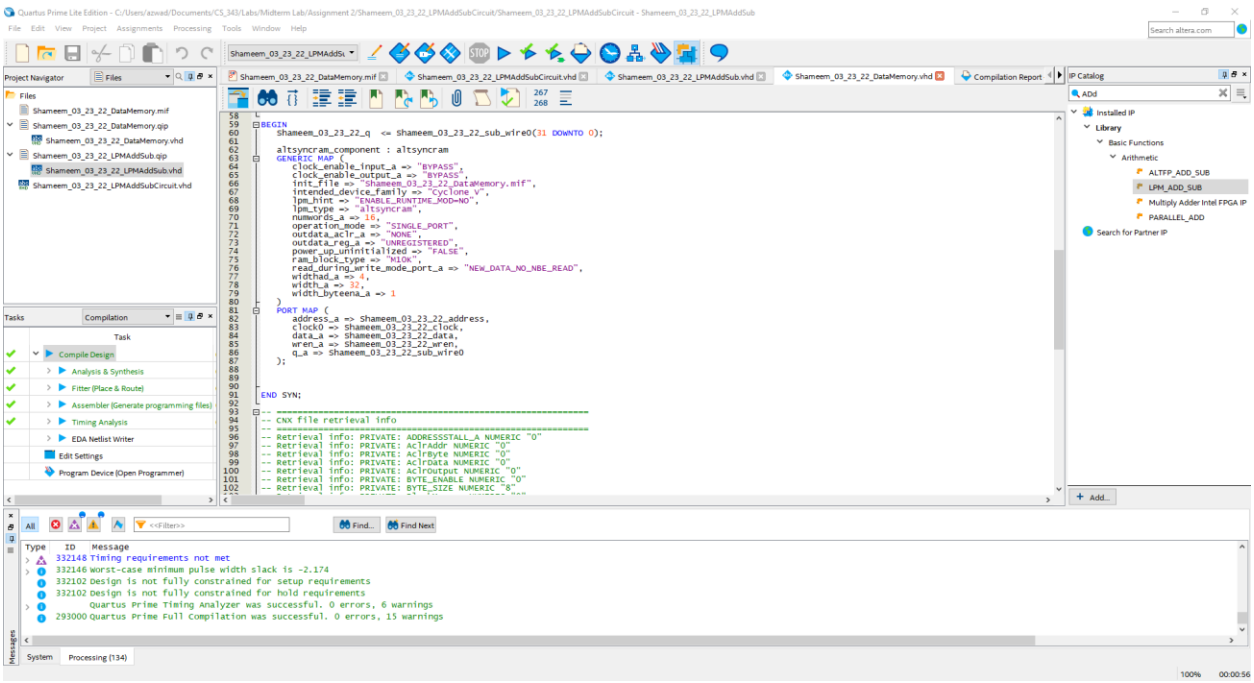


Figure 24: VHDL code continued for Data Memory module.

Explanation:

The LPMAddSubCircuit is a circuit that utilizes the LPM Add/Sub unit and the Data Memory module. The Data Memory module loads the data to memory so that it can be used by the next component which is the LPM Add/Sub unit. The LPM Add/Sub takes the data loaded by the Data Memory module and then computes addition or subtraction and gives the outputs of the computation and the overflow flag. Furthermore, the LPMAddSubCircuit takes the output of the LPM Add/Sub unit and outputs overflow, negative and zero flags depending on the result of the computation of the LPM Add/Sub unit.

Simulation

Assignment 1:

Design 32-bit word Data Memory module based on LPM tutorial attached. Data memory size 16 words.

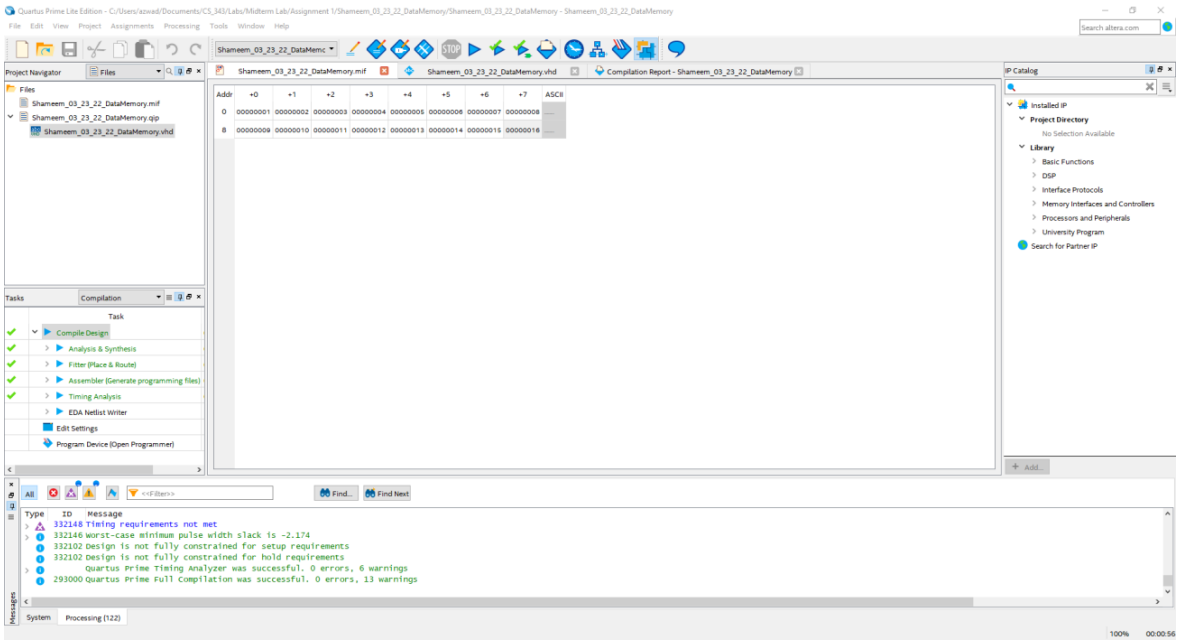


Figure 25: MIF file for Data Memory module.

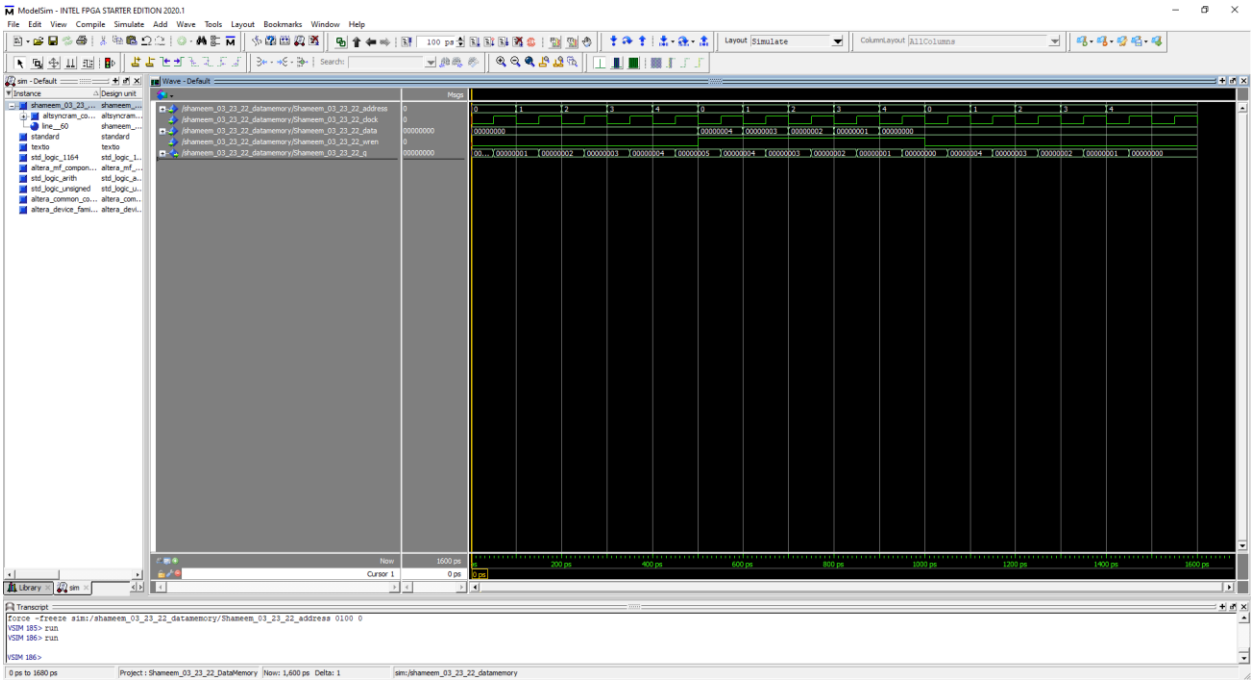


Figure 26: The waveform from the simulation of the Data Memory module using the MIF file in figure 25. Note: The simulation's waveforms are shown in ***Hexadecimal***

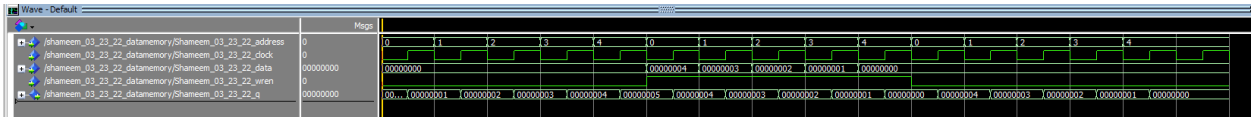


Figure 27: Waveform Simulation from figure 26 zoomed in.
Note: The simulation's waveforms are shown in ***Hexadecimal***

Explanation:

The simulation above is for the Data Memory module and it goes in the order of reading 5 addresses, then writing to 5 addresses and then finally reading the 5 written to addresses to confirm they are correct. From 0 ps to 500 ps the Data Memory module has wren=0. This means the Data Memory module is reading from addresses 0, 1, 2, 3, 4 and outputting the value at that address. The Data memory module correctly reads the addresses 0, 1, 2, 3, 4 because the value returned from the addresses match the MIF file. From 500 ps to 1000 ps the Data Memory module is writing instead because wren=1. This means that the Data Memory module is writing to addresses 0, 1, 2, 3, 4 in between 500 ps to 1000 ps. From 1000 ps to 1600 ps, wren=0 so the Data Memory module is reading the addresses 0, 1, 2, 3, 4. In 1000 ps to 1600 ps, we confirm that the Data Memory module had correctly written in 500 to 1000 ps because the memory being read in 1000 ps to 1600 ps is exactly the same as written data from 500 ps to 1000 ps.

Design 32-bit word INSTRUCTION Memory module based on LPM tutorial attached. Instruction memory size 32 words.

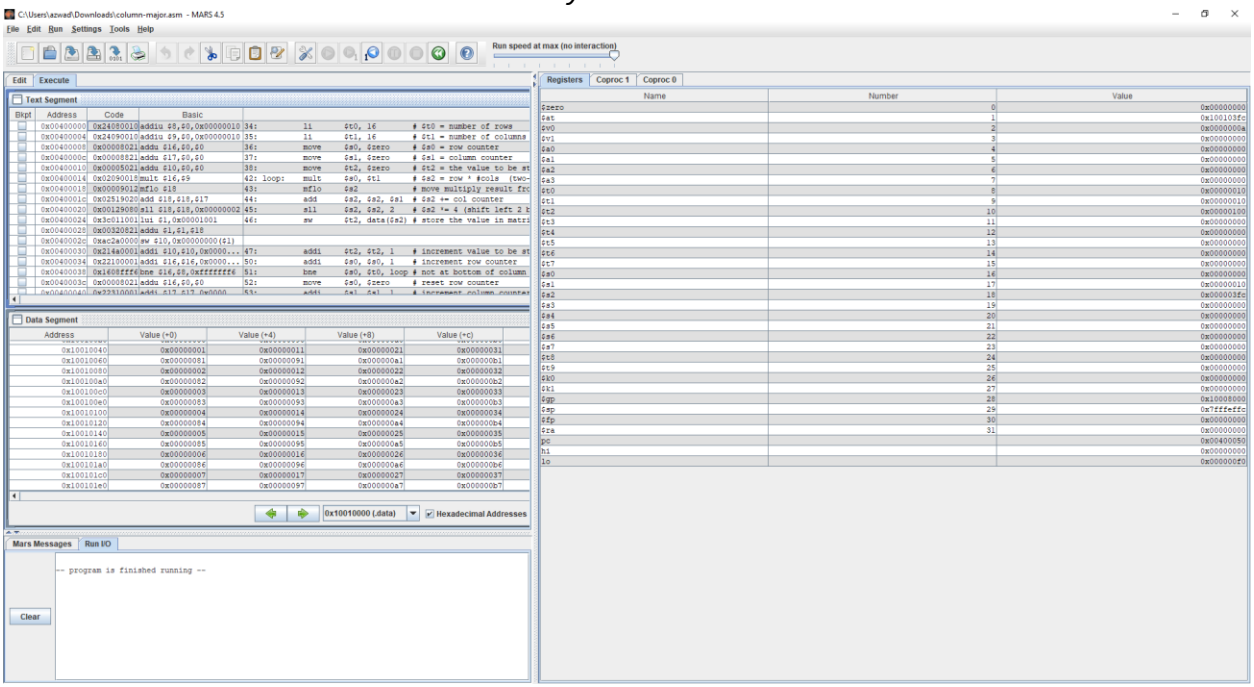
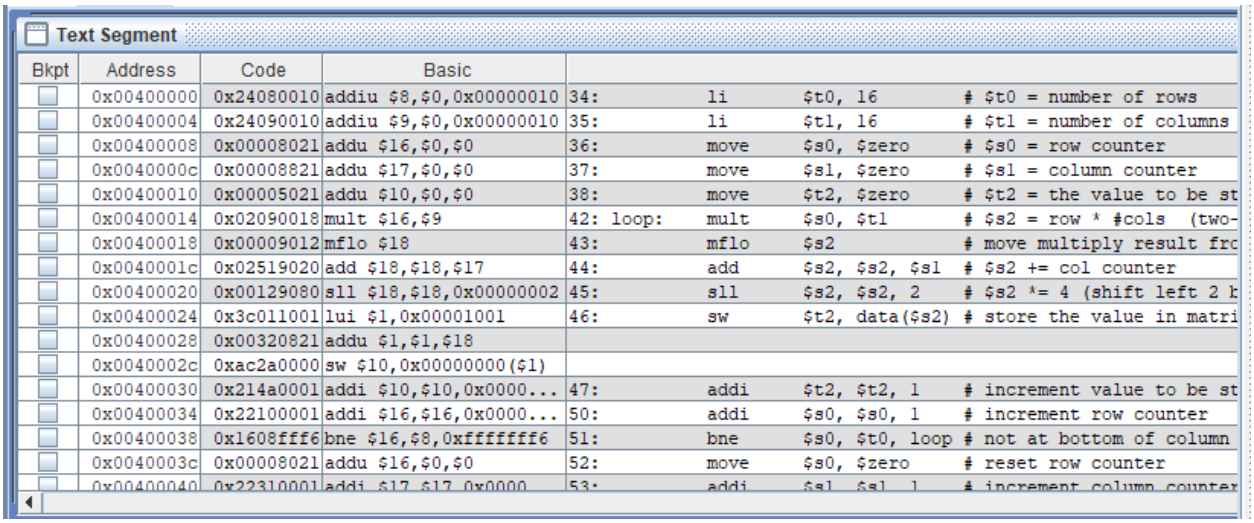


Figure 28: Mars output screen after fully running column-major.asm



Text Segment

Bkpt	Address	Code	Basic				
	0x00400000	0x24080010	addiu $8,$0,0x00000010	34:	li	$t0, 16	# $t0 = number of rows
	0x00400004	0x24090010	addiu $9,$0,0x00000010	35:	li	$t1, 16	# $t1 = number of columns
	0x00400008	0x00008021	addu $16,$0,$0	36:	move	$s0, $zero	# $s0 = row counter
	0x0040000c	0x00008821	addu $17,$0,$0	37:	move	$s1, $zero	# $s1 = column counter
	0x00400010	0x00005021	addu $10,$0,$0	38:	move	$t2, $zero	# $t2 = the value to be st
	0x00400014	0x02090018	mult $16,$9	42: loop:	mult	$s0, $t1	# $s2 = row * #cols (two-
	0x00400018	0x00009012	mflo $18	43:	mflo	$s2	# move multiply result frc
	0x0040001c	0x02519020	add $18,$18,$17	44:	add	$s2, $s2, $s1	# $s2 += col counter
	0x00400020	0x00129080	sll $18,$18,0x00000002	45:	sll	$s2, $s2, 2	# $s2 *= 4 (shift left 2 b
	0x00400024	0x3c011001	lui $1,0x00001001	46:	sw	$t2, data($s2)	# store the value in matri
	0x00400028	0x00320821	addu $1,$1,$18				
	0x0040002c	0xac2a0000	sw $10,0x00000000($1)				
	0x00400030	0x214a0001	addi $10,$10,0x0000...	47:	addi	$t2, $t2, 1	# increment value to be st
	0x00400034	0x22100001	addi $16,$16,0x0000...	50:	addi	$s0, $s0, 1	# increment row counter
	0x00400038	0x1608fff6	bne $16,$8,0xfffffff6	51:	bne	$s0, $t0, loop	# not at bottom of column
	0x0040003c	0x00008021	addu $16,$0,$0	52:	move	$s0, $zero	# reset row counter
	0x00400040	0x22310001	addi $17,$17,0x0000	53:	addi	$s1, $s1, 1	# increment column counter

Figure 29: Zoomed in Mars text segment.

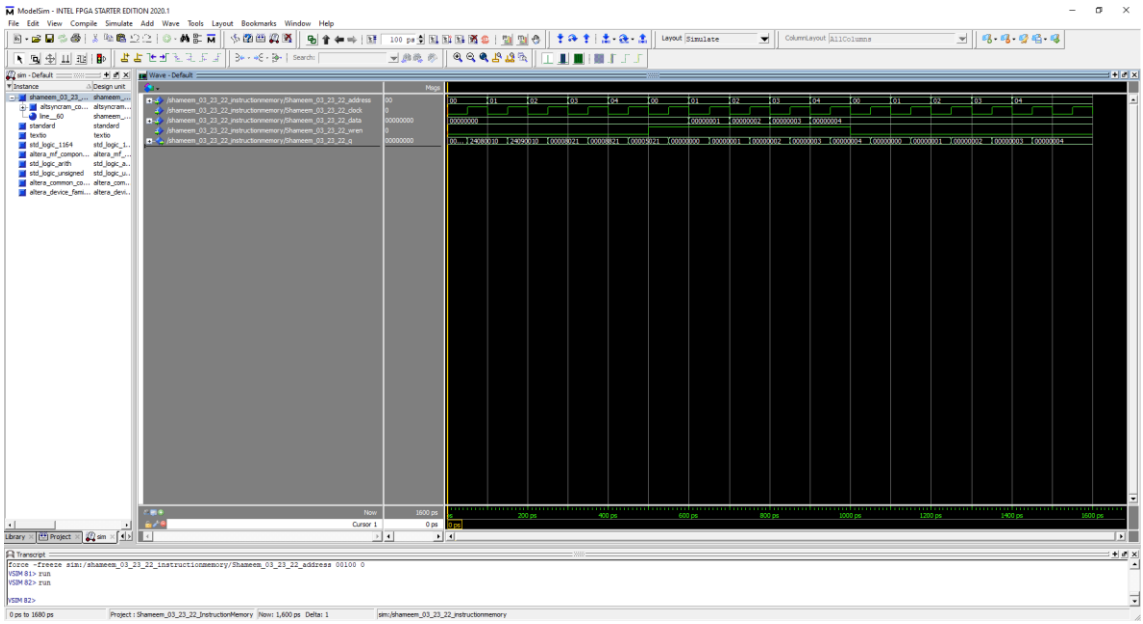


Figure 30: Instruction Memory waveforms from simulation.
Note: The simulation's waveforms are shown in ***Hexadecimal***

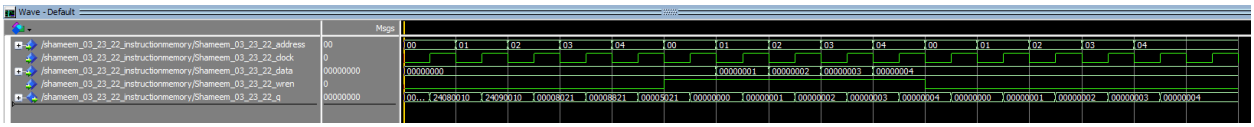


Figure 31: Instruction Memory waveforms simulation zoomed in.
Note: The simulation's waveforms are shown in ***Hexadecimal***

Explanation:

The simulation above is for the Instruction Memory module and it goes in the order of reading 5 addresses, then writing to 5 addresses and then finally reading the 5 written to addresses to confirm they are correct. From 0 ps to 500 ps the Instruction Memory module has wren=0. This means the Instruction Memory module is reading the addresses and outputting the value of the instructions. The Instruction Memory module is reading correctly because the outputted results match the value of the instruction code values from the Mars text segment. From 500 ps to 1000 ps the Instruction Memory module is writing instead because wren=1. This means that the Instruction Memory module is writing to the addresses in between 500 ps to 1000 ps. From 1000 ps to 1600 ps, wren=0 so the Instruction Memory module is reading the addresses. In 1000 ps to 1600 ps, we confirm that the Instruction Memory module had correctly written in 500 to 1000 ps because the memory being read in 1000 ps to 1600 ps is exactly the same as written data from 500 ps to 1000 ps.

Design 32-bit register DUAL PORTED REGISTER FILE module based on 2-port RAM LPM tutorial attached. EACH register is 32 bits

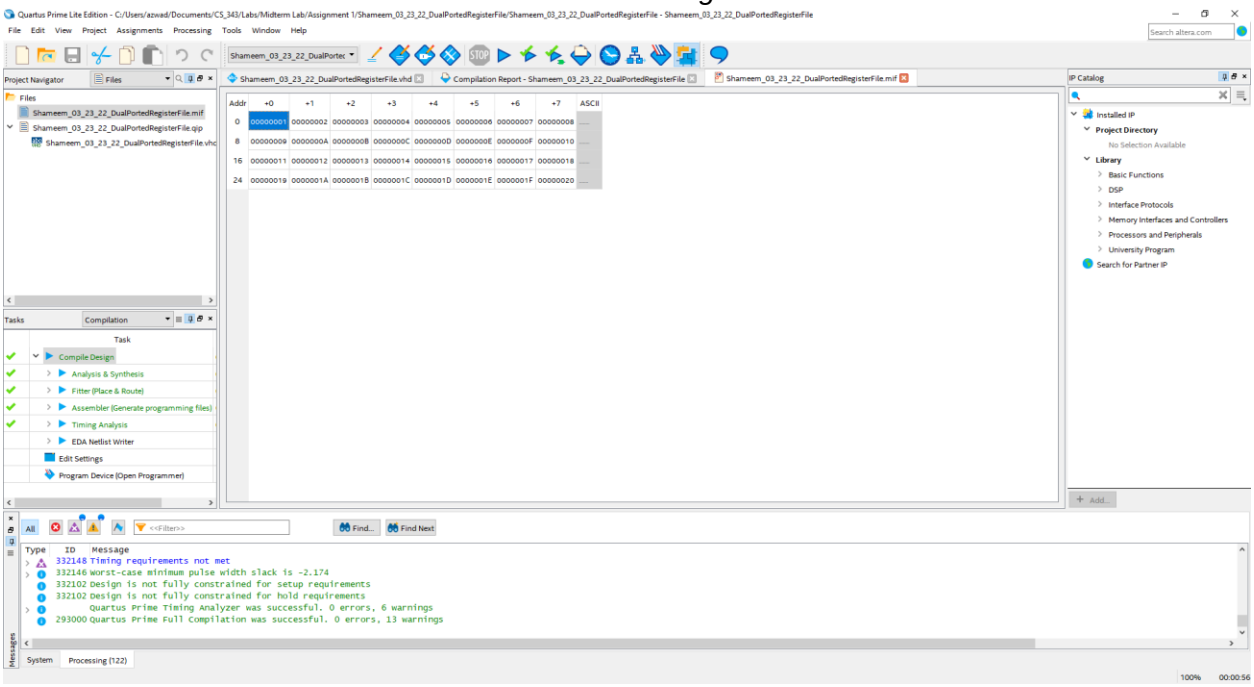


Figure 32: MIF file for Dual Ported Register File

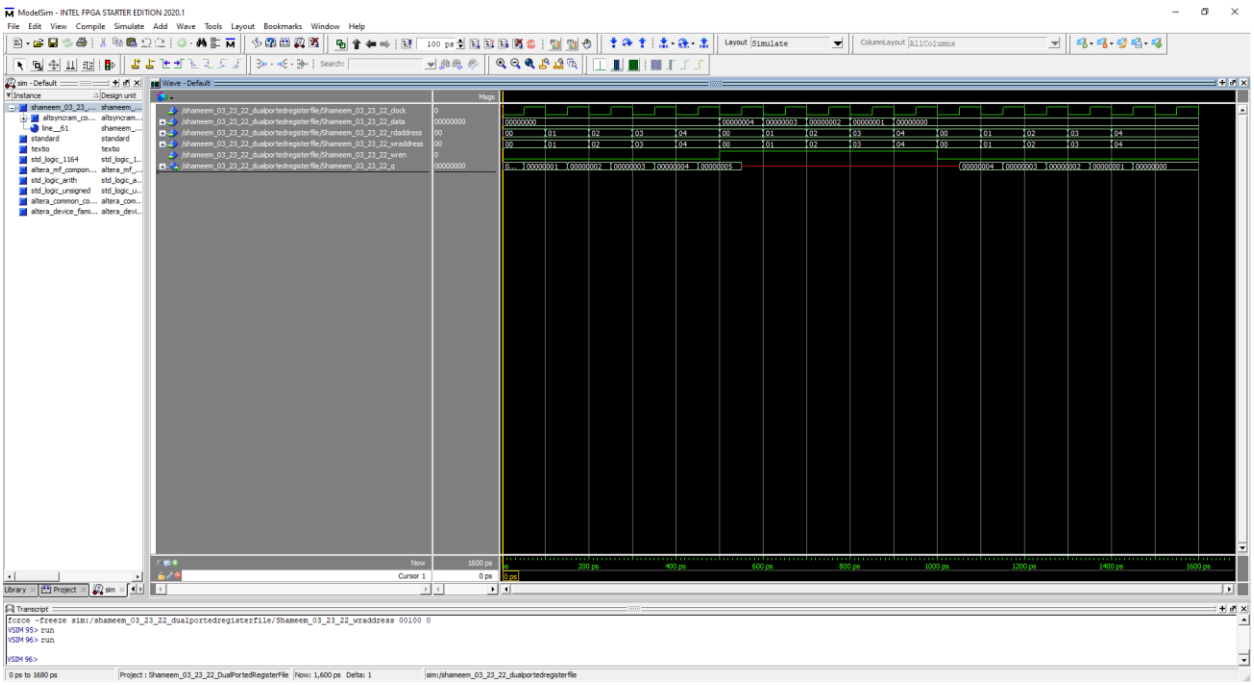


Figure 33: Simulation waveform for the Dual Ported Register File using the MIF file in figure 32. Note: The simulation's waveforms are shown in ***Hexadecimal***

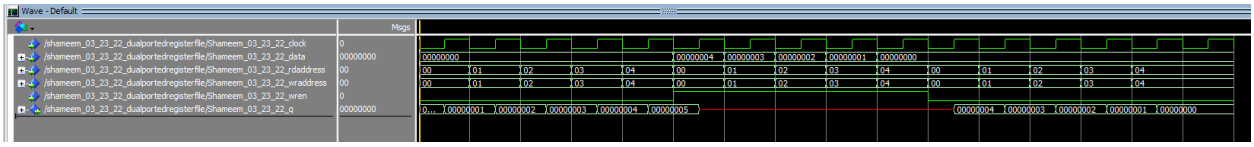


Figure 34: Simulation waveform for the Dual Ported Register File zoomed in.
Note: The simulation's waveforms are shown in ***Hexadecimal***

Explanation:

The simulation above is for the Dual Ported Register File and it goes in the order of reading 5 addresses, then writing to 5 addresses and then finally reading the 5 written to addresses to confirm they are correct. From 0 ps to 500 ps the Dual Ported Register File has wren=0. This means the Dual Ported Register File is reading the addresses and outputting the value at the addresses. The Dual Ported Register File module is reading correctly because the outputted results match the value put into the same addresses in the MIF file. From 500 ps to 1000 ps the Dual Ported Register File is writing instead because wren=1. As a result of writing, the Dual Ported Register File shows that the output q's waveform is red because it is writing and therefore is not showing any results for q. This means that the Dual Ported Register File is writing correctly to the addresses in between 500 ps to 1000 ps. From 1000 ps to 1600 ps, wren=0 so the Dual Ported Register File is reading the addresses that was written in. In 1000 ps to 1600 ps, we confirm that the Dual Ported Register File had correctly written in 500 to 1000 ps because the addresses being read in 1000 ps to 1600 ps is exactly the same as written data from 500 ps to 1000 ps.

Assignment 2:

Design 32-bit Add/Sub unit as described in the seconds attached tutorial from scratch.

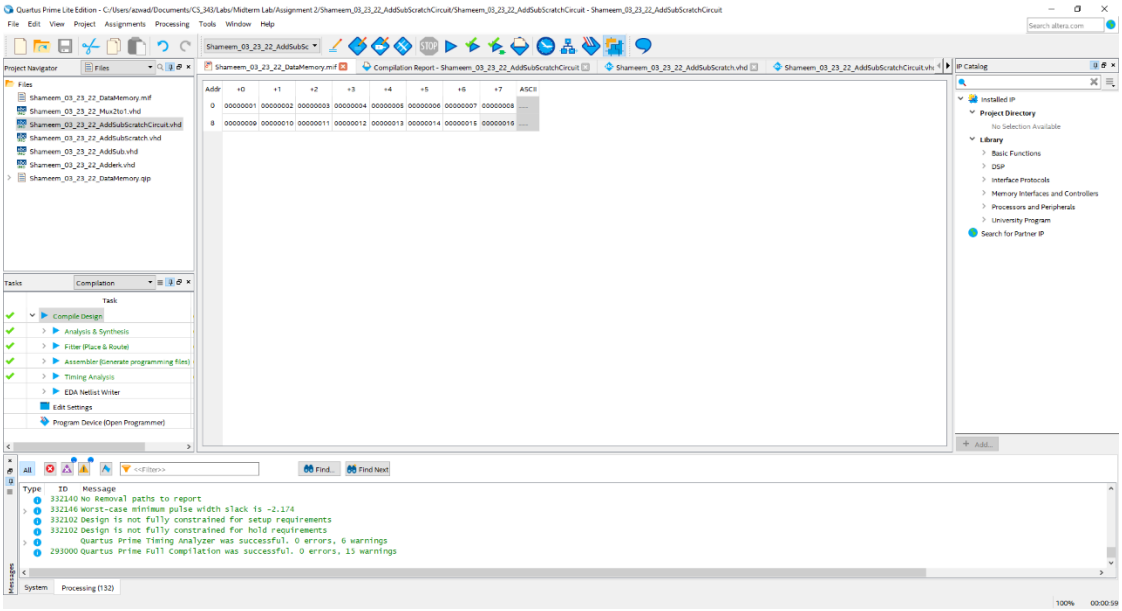


Figure 35: MIF file for the Add/Sub circuit from scratch

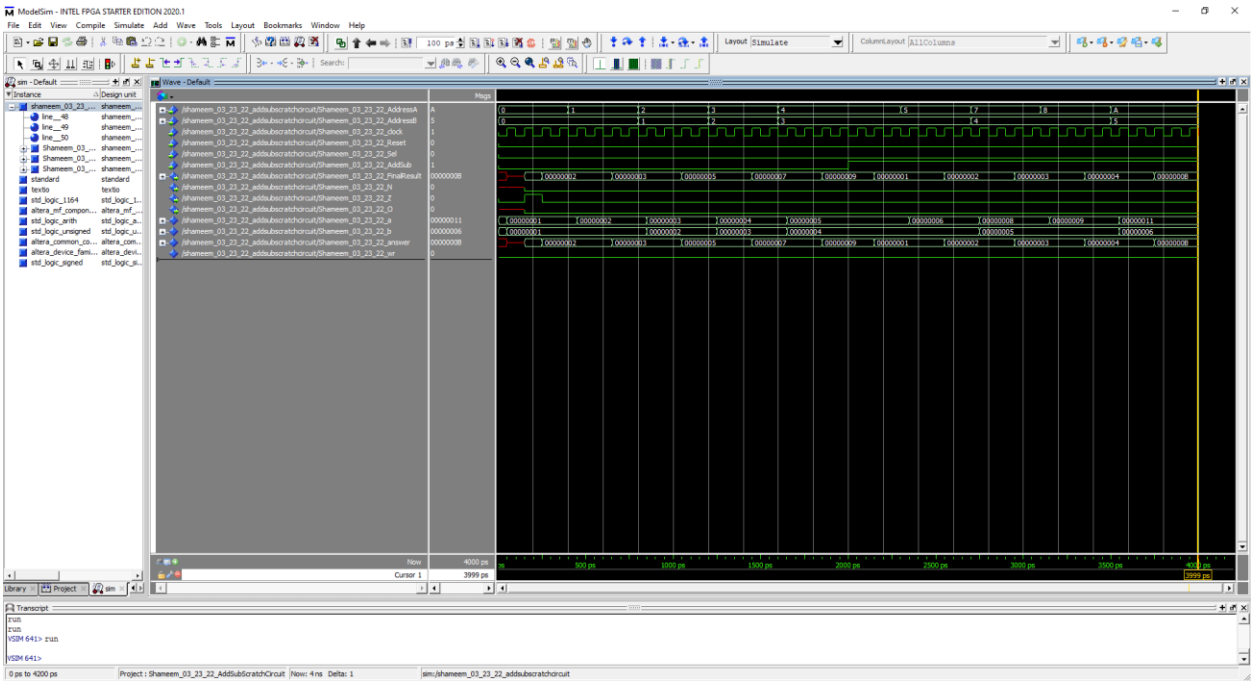


Figure 36: Simulation of ScratchAddSubCircuit using MIF file from figure 35

Note: The simulation's waveforms are shown in ***Hexadecimal***

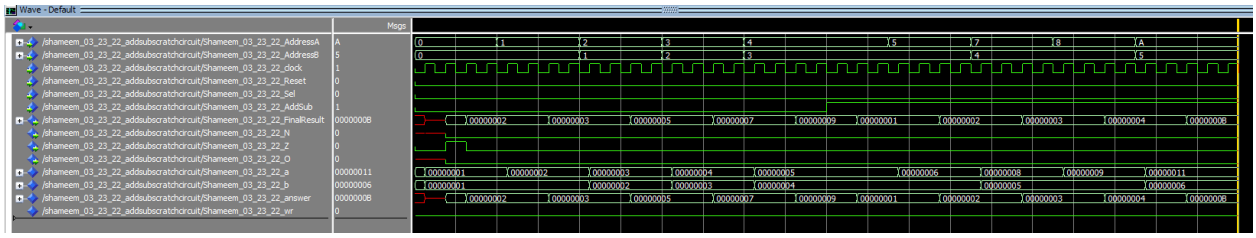


Figure 37: Zoomed in waveform simulation from figure 36
Note: The simulation's waveforms are shown in ***Hexadecimal***

Explanation:

The simulation waveform shown in figures 36 and 37 show the simulation of the ScratchSubAddCircuit. This circuit utilizes the scratch Sub/Add unit and Data Memory module to read values from the addresses in the MIF file and use those values in the scratch Sub/Add unit to add or subtract. From 0 ps to 2000 ps the ScratchSubAddCircuit shows a total of 5 different additions, each with their own values from different addresses. From 2000 ps to 4000 ps the ScratchSubAddCircuit shows a total of 5 different subtractions, each with their own values from different addresses.

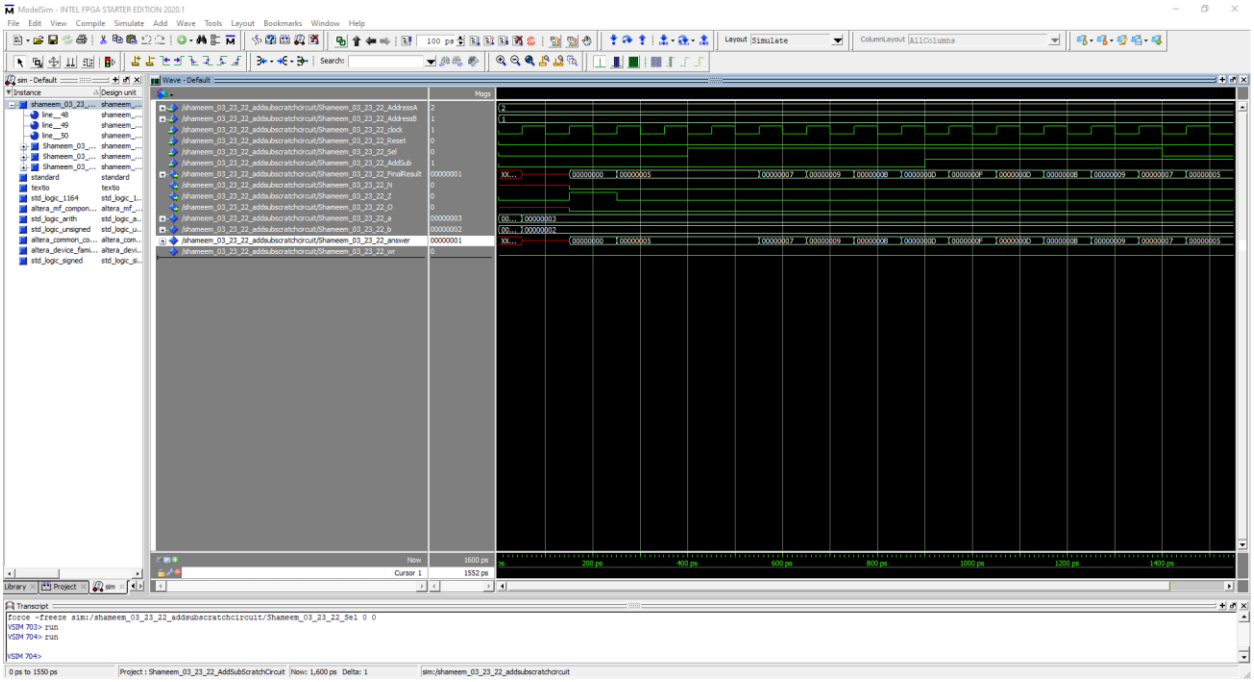


Figure 38: Simulation of ScratchAddSubCircuit using MIF file from figure 35
Note: The simulation's waveforms are shown in ***Hexadecimal***

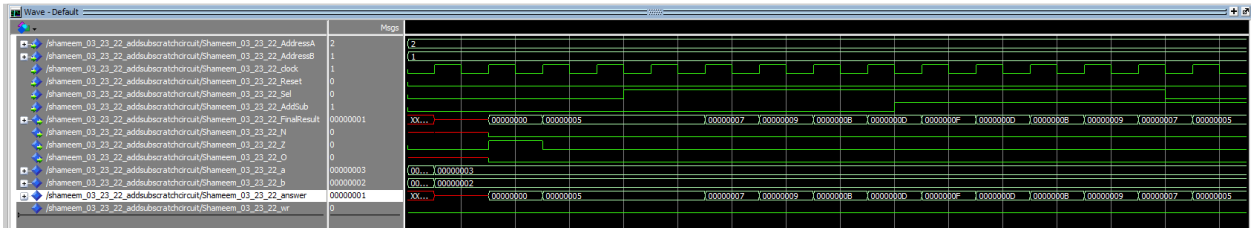


Figure 39: Zoomed in waveforms from simulation in figure 38
Note: The simulation's waveforms are shown in ***Hexadecimal***

Explanations:

From 0 ps to 400 ps, in the ScratchAddSubCircuit the first addition happens. From 400 ps to 900 ps, sel=1 and addsub=0 meaning it is now positively accumulating. As it is shown in the waveform from 400 to 900 ps accumulation happens by starting from 5 and increase to 7 then 9 then B (B is in hex) which is 11 and increase again to D and F and so forth. From 900 ps to 1400 ps the circuit has sel=1 and addsub=1 which means it is now negatively accumulating. This is shown in the waveform where F (hex) decrease to D and then B and then to 9 and so forth. This shows the accumulator part of the circuit is working properly.

Design 32-bit Add/Sub unit as described in the seconds attached tutorial using LPM.

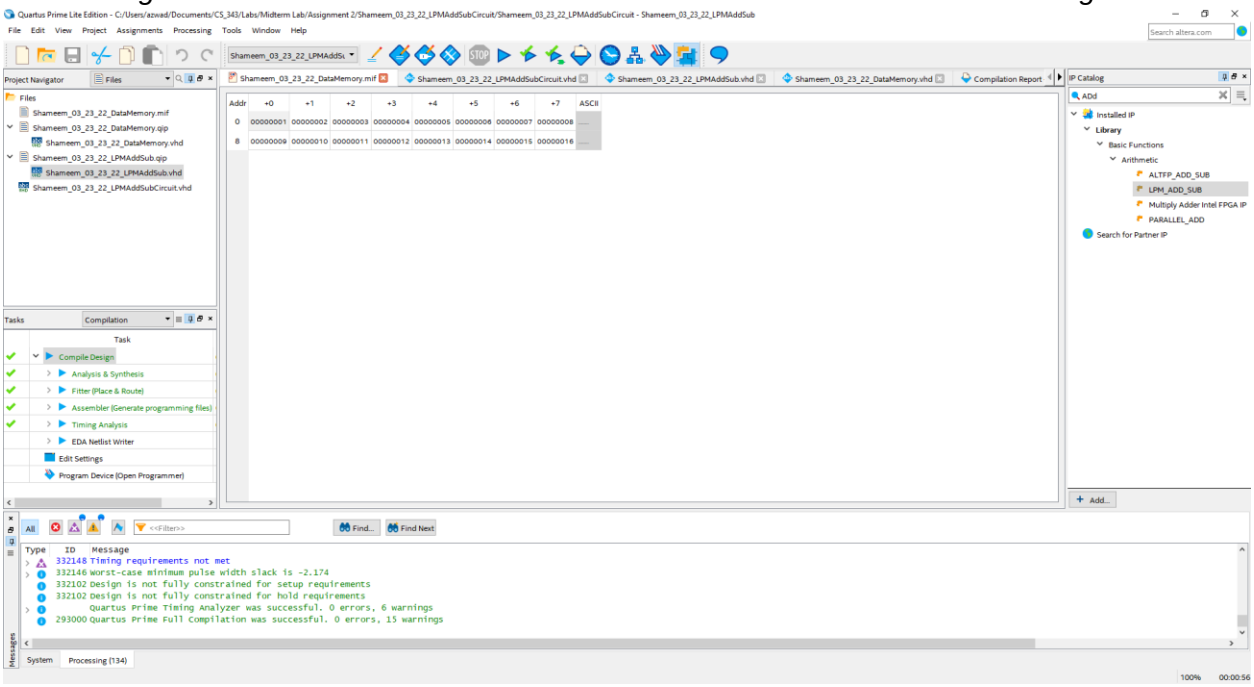


Figure 40: MIF file for the LPMAddSubCircuit

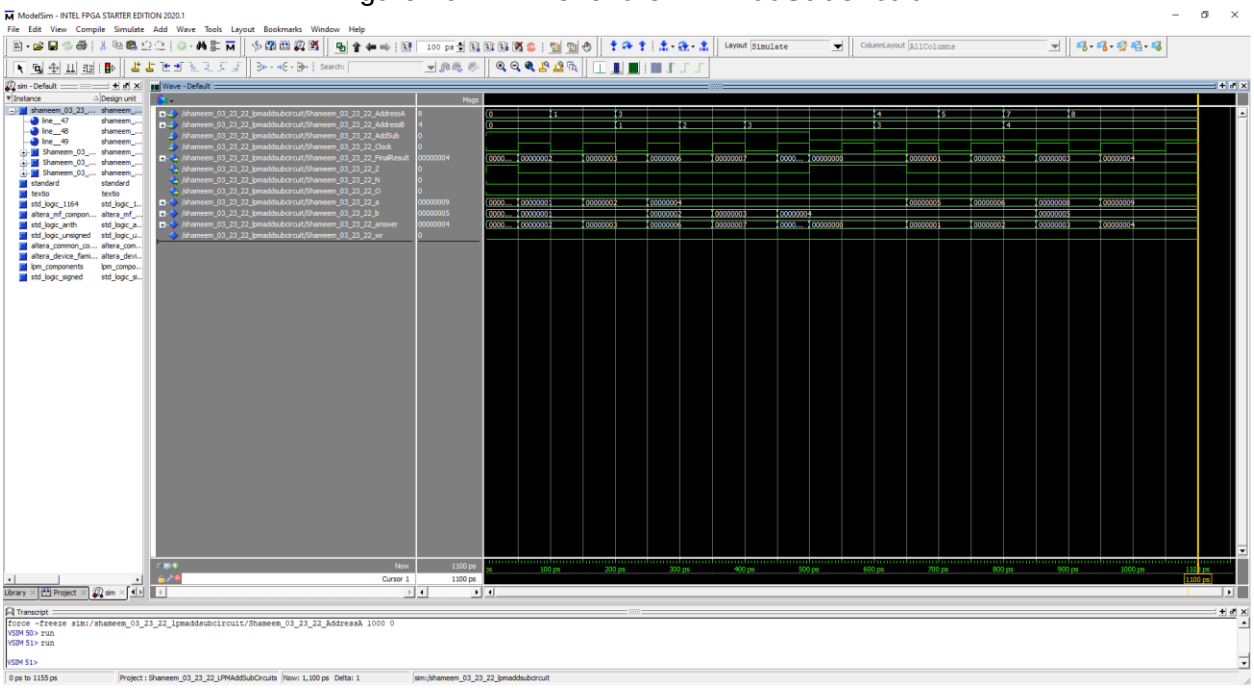


Figure 41: LPMAddSubCircuit simulation waveforms

Note: The simulation's waveforms are shown in ***Hexadecimal***

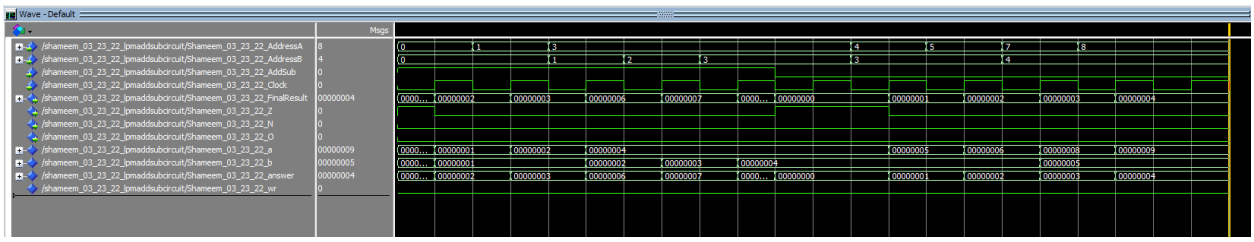


Figure 39: Zoomed in waveforms from simulation in figure 38
Note: The simulation's waveforms are shown in ***Hexadecimal***

Explanations:

From 0 ps to 500 ps the LPMAddSubCircuit shows addsub=1 which means we are doing addition. In fact, in the ranges 0 ps to 500 ps the circuit reads different addresses 5 times and computes addition correctly 5 times. The circuit can be analyzed as correct for addition by making sure the signals shameem_03_23_22_a, shameem_03_23_22_b signals which show the value of the integer read at the address and the value at finalResult output or the answer signal. From 500 ps to 1100 ps the LPMAddSubCircuit shows addsub=0 which means we are doing subtraction. In the ranges 500 ps to 1100 ps the circuit reads different addresses 5 times and computes subtraction correctly 5 times. It is also easy to make sure the circuit is computing subtraction correctly by checking the integers read from the addresses and the finalresult output.

Conclusion

The midterm lab was a great way to experiment with Quartus's IP catalog and create RAM 1-port and RAM 2-port and become familiar with their use. Furthermore, we utilized the ram in order to read the data from the MIF file and load that data into a Add/Sub unit so that we can do computation with the data. Utilizing Add/Sub and RAM in conjunction is a great way to learn VHDL and is a great way to start to understand how a circuit may do addition or subtraction with data in our computers. Lastly this exercise gave us good practice with ModelSim as well because we had to simulate to prove our circuit was working.