

CS 342000 / CS343000

Instructor: Professor Izidor Gertner

Spring 2022

Azwad Shameem, 3/23/2022

Midterm Lab Project

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

Azwad Shameem

Table of Contents

| | |
|--------------------------------------------------------------|----|
| Objective | 3 |
| Description of Specifications and Functionality | 3 |
| Specifications: | 3 |
| 1. Shameem_03_23_22_DataMemory: | 3 |
| 2. Shameem_03_23_22_InstructionMemory: | 3 |
| 3. Shameem_03_23_22_DualPortedRegisterFile:..... | 3 |
| 4. Shameem_03_23_22_AddSubScratchCircuit:..... | 4 |
| 5. Shameem_03_23_22_AddSubScratch: | 4 |
| 6. Shameem_03_23_22_LPMAddSubCircuit: | 5 |
| 7. Shameem_03_23_22_LPMAddSub:..... | 5 |
| Functionality: | 6 |
| <i>Assignment 1:.....</i> | 6 |
| Assignment 2:..... | 11 |
| Simulation..... | 20 |
| <i>Assignment 1:.....</i> | 20 |
| Assignment 2: | 26 |
| Conclusion..... | 31 |

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.

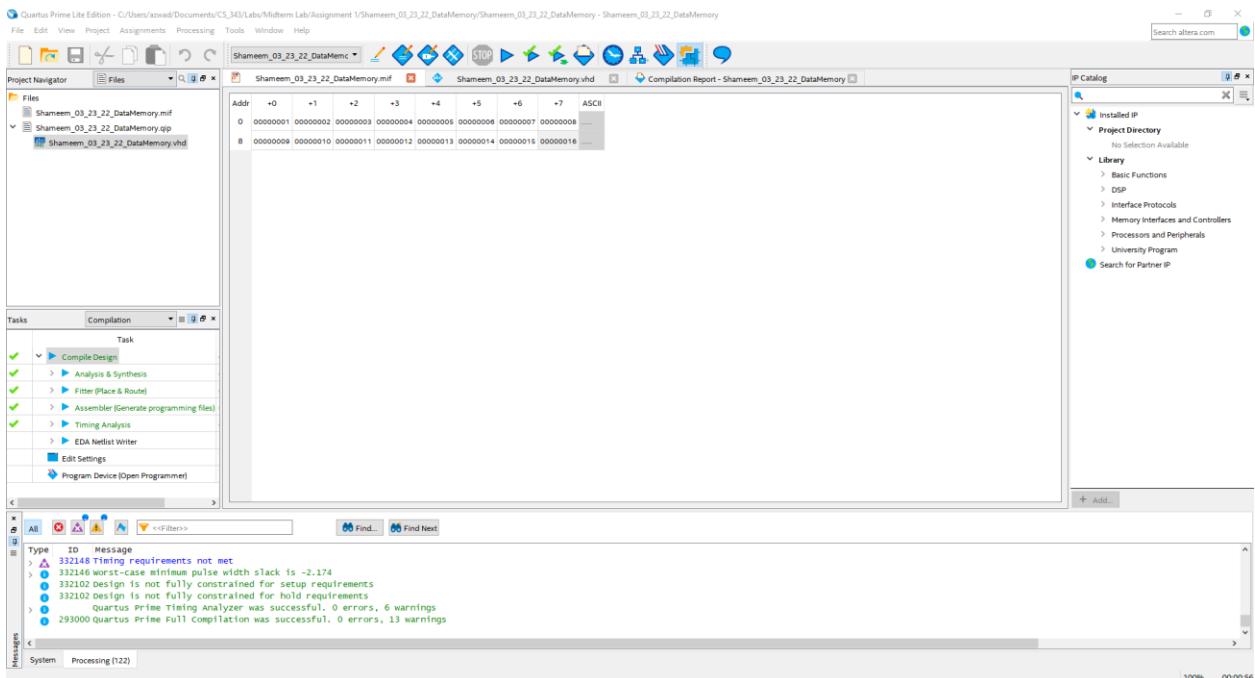


Figure 1: The mif file used in Data Memory module

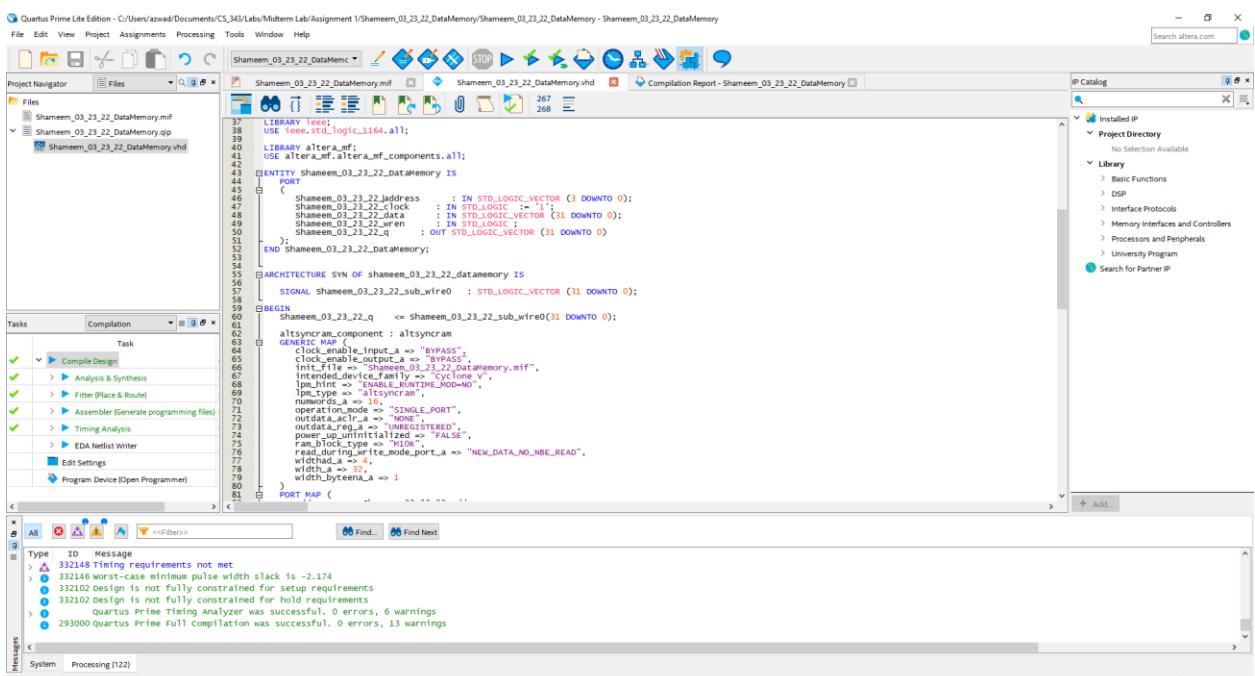


Figure 2: Data Memory module VHDL code generated for RAM 1-port from Quartus IP Catalog

```

library IEEE;
use IEEE.STD.TEXT.all;
library altera;
use altera.alteramf_components.all;
entity shameem_03_23_22_DataMemory is
  port(
    Shameem_03_23_22_q : out STD_LOGIC_VECTOR (31 DOWNTO 0);
    altsyncram_component : altsyncram;
    generic map(
      input_a=>"BYPASS",
      clock_enable_output_a=>"BYPASS",
      init_file=>"Shameem_03_23_22.datamemory.mif",
      internal_device_family=>"cyclone V",
      ipm_hint=>"ENABLE_RUNTIME_MOD=NO",
      ipm_type=>"altsyncram",
      numwords=>16,
      operation_mode=>"SINGLE_PORT",
      outdata_regr_a=>"REGISTERED",
      outdata_regr_b=>"UNREGISTERED",
      power_up_uninitialized=>"FALSE",
      ram_block_type=>"LUTRAM",
      read_during_write_mode_port_a=>"NEW_DATA_NO_NBE_READ",
      widthad_a=>5,
      widthdata_a=>32,
      widthbyteena_a=>1
    );
  );
end entity;
architecture SYN of shameem_03_23_22_DataMemory is
begin
  Shameem_03_23_22_q <= Shameem_03_23_22_sub_wire(31 DOWNTO 0);
end architecture;
  
```

Messages

- 332148 Timing requirements not met
- 332146 Worst-case minimum pulse width slack is -2.174
- 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, 13 warnings

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.

```

library IEEE;
use IEEE.STD.TEXT.all;
library altera;
use altera.alteramf_components.all;
entity shameem_03_23_22_InstructionMemory is
  port(
    Shameem_03_23_22_address : in STD_LOGIC_VECTOR (4 DOWNTO 0);
    Shameem_03_23_22_clock : in STD_LOGIC;
    Shameem_03_23_22_wren : in STD_LOGIC_VECTOR (31 DOWNTO 0);
    Shameem_03_23_22_q : out STD_LOGIC_VECTOR (31 DOWNTO 0)
  );
end entity;
architecture SYN of shameem_03_23_22_InstructionMemory is
begin
  Shameem_03_23_22_q <= Shameem_03_23_22_sub_wire(31 DOWNTO 0);
end architecture;
  
```

Messages

- 332148 Timing requirements not met
- 332146 Worst-case minimum pulse width slack is -2.174
- 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, 13 warnings

Figure 4: Instruction Memory module generated VHDL code

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

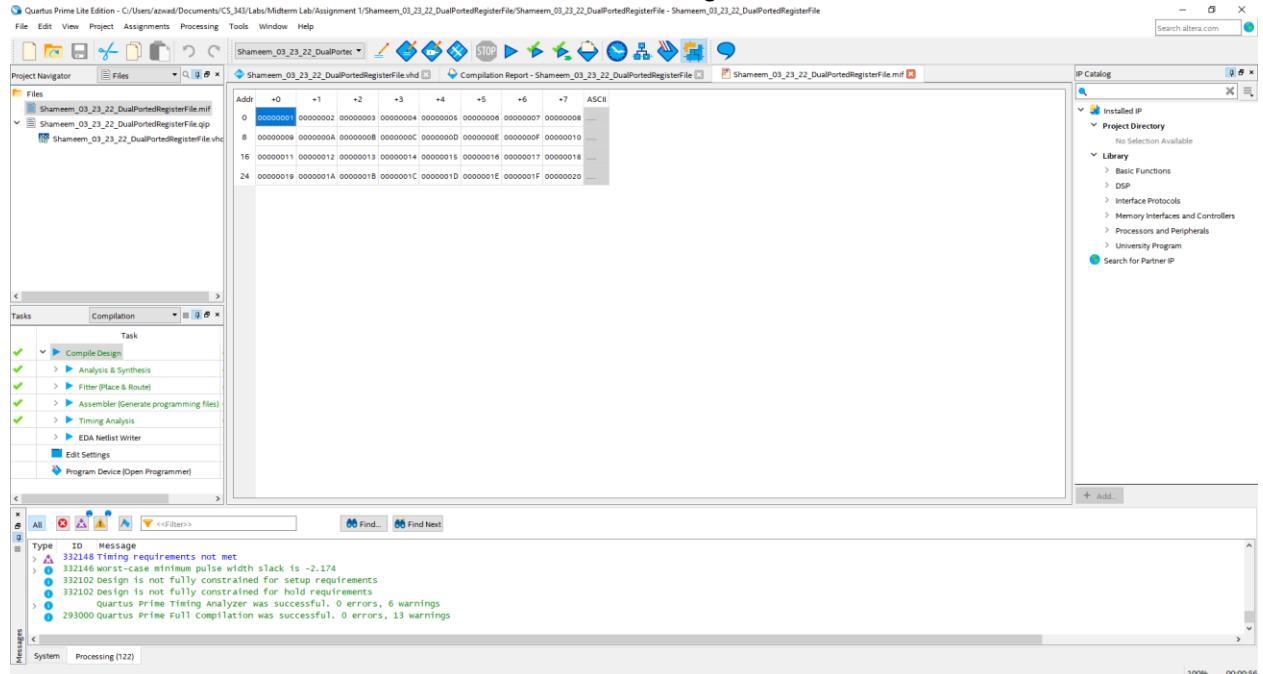


Figure 6: mif file for Dual Ported Register File module

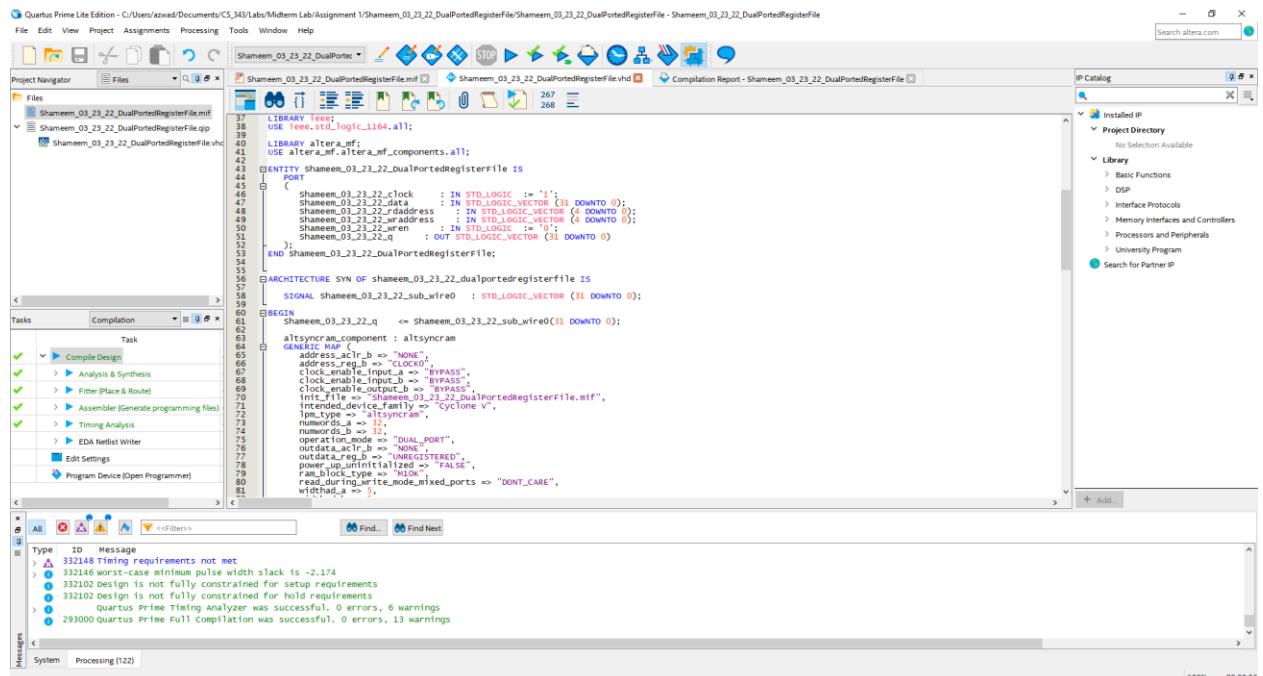


Figure 7: Dual Ported Register File module based on 2-port RAM generated VHDL code from Quartus IP Catalog

```

SIGNAL Shameem_03_23_22_sub_wire0 : STD_LOGIC_VECTOR (31 DOWNTO 0);
BEGIN
  Shameem_03_23_22_q <= Shameem_03_23_22_sub_wire0(31 DOWNTO 0);
  altsyncram_component : altsyncram
    generic map (
      address_aclr_b => "NONE",
      address_regr_b => "CLOCK0",
      clock_enable_input_b => "BYPASS",
      clock_enable_output_b => "BYPASS",
      init_file => "shameem_03_23_22_DualPortedRegisterFile.mif",
      intended_device_family => "Cyclone V",
      lpm_hint => "USE_ALTSYNCRAM",
      numrowds_a => 32,
      numrowds_b => 32,
      operation_mode => "DUAL_PORT",
      outdata_aclr_b => "NONE",
      outdata_regr_b => "REGISTERED",
      power_up_uninitialized => "FALSE",
      read_during_write_mode_mixed_ports => "DONT_CARE",
      read_oob_type => "MIXED",
      widthad_a => 3,
      widthad_b => 3,
      width_a => 32,
      width_b => 32,
      width_bytelen_a => 1
    )
    PORT MAP (
      address_a => Shameem_03_23_22_ur_address,
      address_b => Shameem_03_23_22_ur_address,
      clock0 => Shameem_03_23_22_clock,
      data_a => Shameem_03_23_22_data,
      data_b => Shameem_03_23_22_data,
      wren_a => Shameem_03_23_22_wren,
      wren_b => Shameem_03_23_22_wren
    );
  END SYN;
  --- CNX file retrieval info
END;

```

Messages

- Type ID Message
- 332148 Timing requirements not met
- 332146 Worst-case minimum pulse width slack is -2.174
- 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, 13 warnings

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.

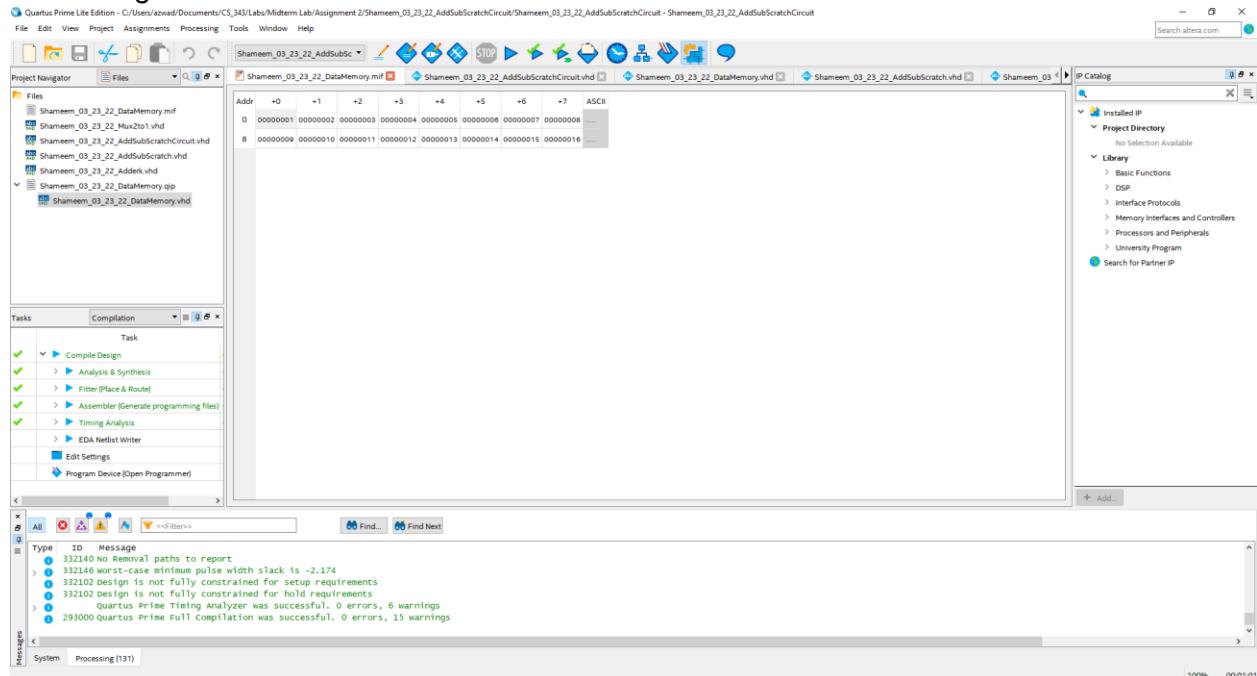


Figure 9: mif file used for the Data Memory module

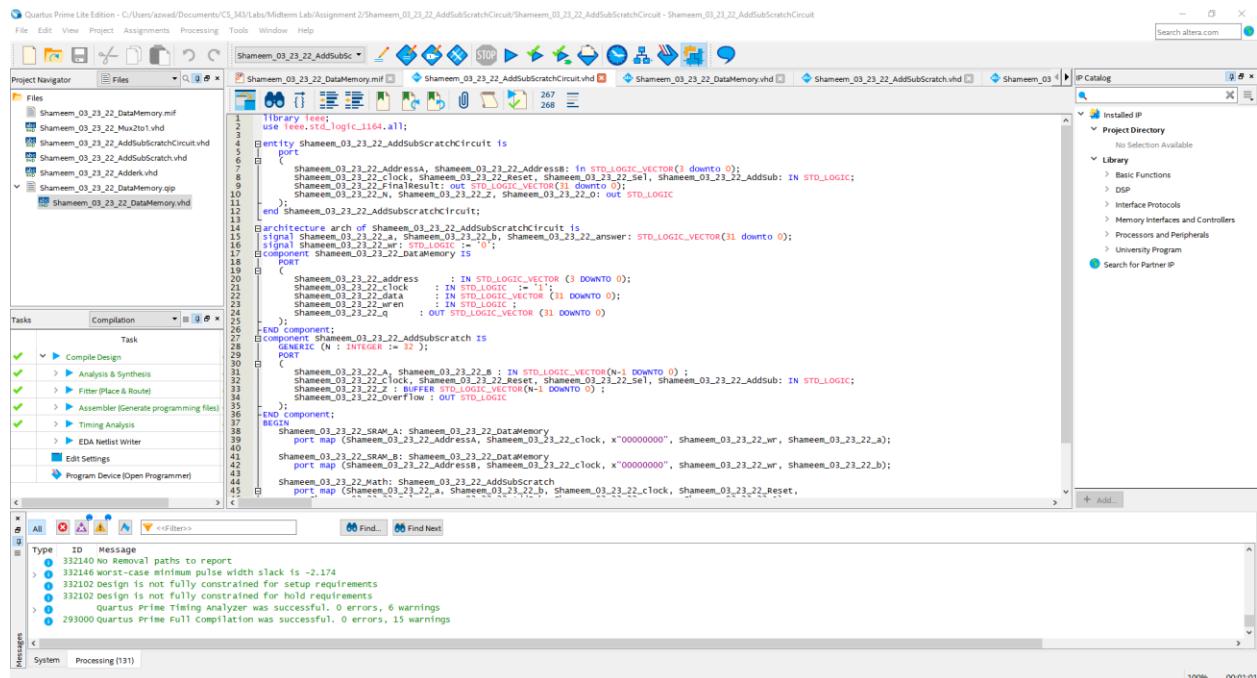


Figure 10: ScratchAddSubCircuit VHDL code. This circuit utilizes the Data Memory module and Add/Sub unit as components to load data to memory from MIF file and then use that data in order to do computation on the Add/Sub unit.

```

library IEEE;
use IEEE.STD_LOGIC.all;
library altera_mf;
use altera_mf.altera_mf_components.all;
entity shameem_03_23_22_dataMemory is
  port (
    Shameem_03_23_22_address : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    Shameem_03_23_22_clock : IN STD_LOGIC := '1';
    Shameem_03_23_22_data : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    Shameem_03_23_22_en : IN STD_LOGIC;
    Shameem_03_23_22_q : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );
end component;
component shameem_03_23_22_AddSubScratchCircuit is
  signal Shameem_03_23_22_Reset : STD_LOGIC_VECTOR(31 DOWNTO 0);
  signal Shameem_03_23_22_N : STD_LOGIC;
  signal Shameem_03_23_22_o : STD_LOGIC;
  port (
    Shameem_03_23_22_Address : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    Shameem_03_23_22_Clock : IN STD_LOGIC := '1';
    Shameem_03_23_22_Data : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    Shameem_03_23_22_En : IN STD_LOGIC;
    Shameem_03_23_22_Q : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );
end component;
GENERIC(N : INTEGER := 32);
PORT(
  Shameem_03_23_22_A : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
  Shameem_03_23_22_B : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
  Shameem_03_23_22_Reset : STD_LOGIC;
  Shameem_03_23_22_Sel : STD_LOGIC;
  Shameem_03_23_22_Clock : STD_LOGIC;
  Shameem_03_23_22_Data : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
  Shameem_03_23_22_Q : STD_LOGIC_VECTOR(N-1 DOWNTO 0)
);
BEGIN
  Shameem_03_23_22_A := shameem_03_23_22_DataMemory
    port map (Shameem_03_23_22_Address, shameem_03_23_22_clock, "00000000", shameem_03_23_22_wr, shameem_03_23_22_a);
  Shameem_03_23_22_BRAM_B := shameem_03_23_22_DataMemory
    port map (Shameem_03_23_22_Address, shameem_03_23_22_clock, "00000000", shameem_03_23_22_wr, shameem_03_23_22_b);
  Shameem_03_23_22_MATH := shameem_03_23_22_AddSubScratch
    port map (Shameem_03_23_22_A, shameem_03_23_22_B, shameem_03_23_22_clock, shameem_03_23_22_Reset, shameem_03_23_22_Wire0);
  Shameem_03_23_22_FinalResult := shameem_03_23_22_Answer;
  Shameem_03_23_22_N := "1" when(shameem_03_23_22_Answer = "00000000") else "0";
END arch;
  
```

Messages

- 332140 No Removal paths to report
- 332146 Worst-case minimum pulse width slack is -2.174
- 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, 15 warnings

Figure 11: ScratchAddSubCircuit VHDL code continued

```

library IEEE;
use IEEE.STD_LOGIC.all;
library altera_mf;
use altera_mf.altera_mf_components.all;
entity shameem_03_23_22_dataMemory is
  port (
    Shameem_03_23_22_address : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    Shameem_03_23_22_clock : IN STD_LOGIC := '1';
    Shameem_03_23_22_data : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    Shameem_03_23_22_en : IN STD_LOGIC;
    Shameem_03_23_22_q : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );
end component;
ARCHITECTURE SYN OF shameem_03_23_22_dataMemory IS
  SIGNAL Shameem_03_23_22_sub_wire0 : STD_LOGIC_VECTOR(31 DOWNTO 0);
BEGIN
  Shameem_03_23_22_q <= Shameem_03_23_22_sub_wire0(31 DOWNTO 0);
  altSyncram: component altSyncram
    generic map (
      clock_enable_input_a => "BYPASS";
      init_file => "shameem_03_23_22_dataMemory.mif";
      internal_device_family => "cyclone V";
      part_name => "CPLD-EZ300-NO";
      ipm_type => "altSyncram";
      num_ports => 1;
      operation_mode => "SINGLE_PORT";
      outdata_aclr_a => "NONE";
      outdata_ena_a => "REGISTERED";
      power_up_uninitialized => FALSE;
      read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ";
      widthadr_a => 4;
      widthdata_a => 32;
      widthbyteena_a => 1
    )
    port map (
      );
  END;
  
```

Messages

- 332140 No Removal paths to report
- 332146 Worst-case minimum pulse width slack is -2.174
- 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, 15 warnings

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

The screenshot shows the Quartus Prime Lite Edition interface with the project 'Shameem_03_23_22_DataMemory' open. The code editor displays the VHDL code for the 'DataMemory' component. The IP Catalog is visible on the right, and a messages window at the bottom shows compilation results.

Figure 13: Data Memory module VHDL code continued

The screenshot shows the Quartus Prime Lite Edition interface with the project 'Shameem_03_23_22_DataMemory' open. The code editor displays the VHDL code for the 'Add/Sub' unit. The IP Catalog is visible on the right, and a messages window at the bottom shows compilation results.

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

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use STD.TEXTIO.all;
use STD.TEXTIO.TEXTFILE.all;

entity Shameem_03_23_22_AddSubScratchCircuit is
    port (
        Shameem_03_23_22_carryin : IN STD_LOGIC ;
        Shameem_03_23_22_A : OUT STD_LOGIC_VECTOR (Shameem_03_23_22_k-1 DOWNTO 0) ;
        Shameem_03_23_22_B : IN STD_LOGIC_VECTOR (Shameem_03_23_22_k-1 DOWNTO 0) ;
        Shameem_03_23_22_S : OUT STD_LOGIC_VECTOR (Shameem_03_23_22_k-1 DOWNTO 0) ;
        Shameem_03_23_22_carryout : OUT STD_LOGIC
    );
end entity;

architecture Behavioural of Shameem_03_23_22_AddSubScratchCircuit is
begin
    process (Shameem_03_23_22_Clock)
    begin
        if Shameem_03_23_22_Reset = '1' then
            Shameem_03_23_22_Areg <= (OTHERS => '0');
            Shameem_03_23_22_Breg <= (OTHERS => '0');
            Shameem_03_23_22_SelR <= '0';
            Shameem_03_23_22_overflow <= '0';
        elsif rising_edge(Shameem_03_23_22_Clock) and Shameem_03_23_22_clock = '1' then
            Elsif Shameem_03_23_22_Areg = Shameem_03_23_22_Breg then
                Shameem_03_23_22_Breg <= Shameem_03_23_22_B;
                Shameem_03_23_22_SelR <= '1';
                Shameem_03_23_22_AdsubR <= Shameem_03_23_22_Adsub;
                Shameem_03_23_22_overflow <= Shameem_03_23_22_over_flow;
            else
                Shameem_03_23_22_AdsubR <= Shameem_03_23_22_Adsub;
                Shameem_03_23_22_Breg <= XOR (Shameem_03_23_22_Areg, Shameem_03_23_22_Adsub);
                Shameem_03_23_22_overflow <= Shameem_03_23_22_carryout XOR Shameem_03_23_22_G(N-1);
                Shameem_03_23_22_overflow <= Shameem_03_23_22_overflow(N-1) XOR Shameem_03_23_22_M(N-1);
            end if;
        end if;
    end process;
end architecture;

```

Messages

- 332140 No Removal paths to report
- 332146 worst-case minimum pulse width slack is -2.174
- 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, 15 warnings

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

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use STD.TEXTIO.all;
use STD.TEXTIO.TEXTFILE.all;

entity Shameem_03_23_22_DataMemory is
    port (
        Shameem_03_23_22_carryin : IN STD_LOGIC ;
        Shameem_03_23_22_A : OUT STD_LOGIC_VECTOR (Shameem_03_23_22_k-1 DOWNTO 0) ;
        Shameem_03_23_22_B : IN STD_LOGIC_VECTOR (Shameem_03_23_22_k-1 DOWNTO 0) ;
        Shameem_03_23_22_S : OUT STD_LOGIC_VECTOR (Shameem_03_23_22_k-1 DOWNTO 0) ;
        Shameem_03_23_22_carryout : OUT STD_LOGIC
    );
end entity;

architecture Behavioural of Shameem_03_23_22_DataMemory is
begin
    process (Shameem_03_23_22_Adderk)
    begin
        if Shameem_03_23_22_Adderk = '0' then
            Shameem_03_23_22_carryin <= '0';
            Shameem_03_23_22_S <= (OTHERS => '0');
            Shameem_03_23_22_carryout <= '0';
        else
            Shameem_03_23_22_Sum <= ('0' & Shameem_03_23_22_X + ('0' & Shameem_03_23_22_Y) + Shameem_03_23_22_carryin);
            Shameem_03_23_22_carryout <= Shameem_03_23_22_Sum(Shameem_03_23_22_k-1 DOWNTO 0);
        end if;
    end process;
end architecture;

```

Messages

- 332140 No Removal paths to report
- 332146 worst-case minimum pulse width slack is -2.174
- 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, 15 warnings

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

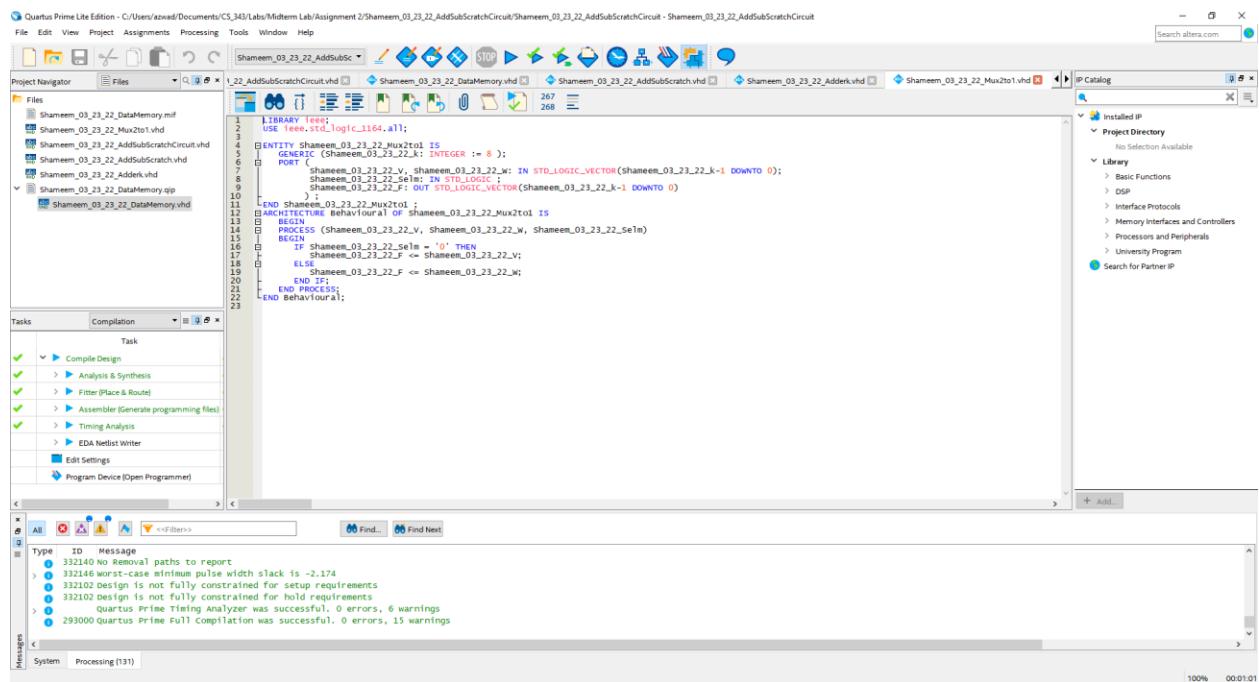


Figure 17: Mux 2:1 VHDL Code. This unit is utilized in scratch Add/Sub.

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.

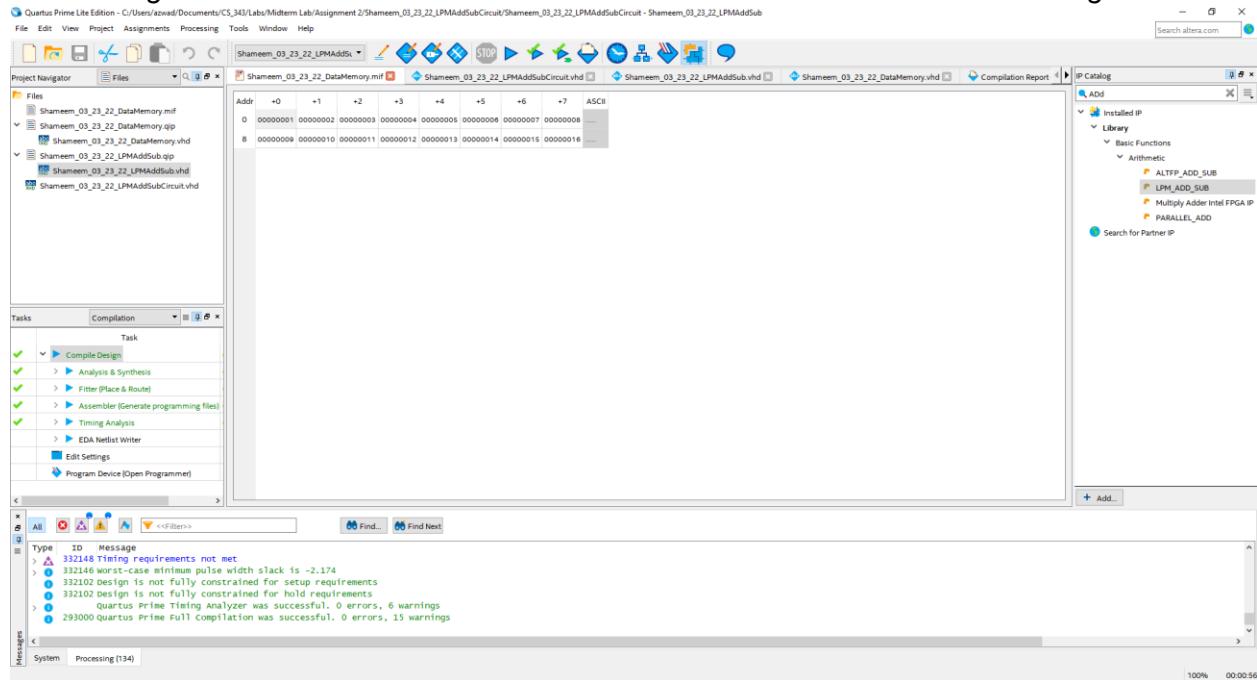


Figure 18: MIF file for the Data Memory

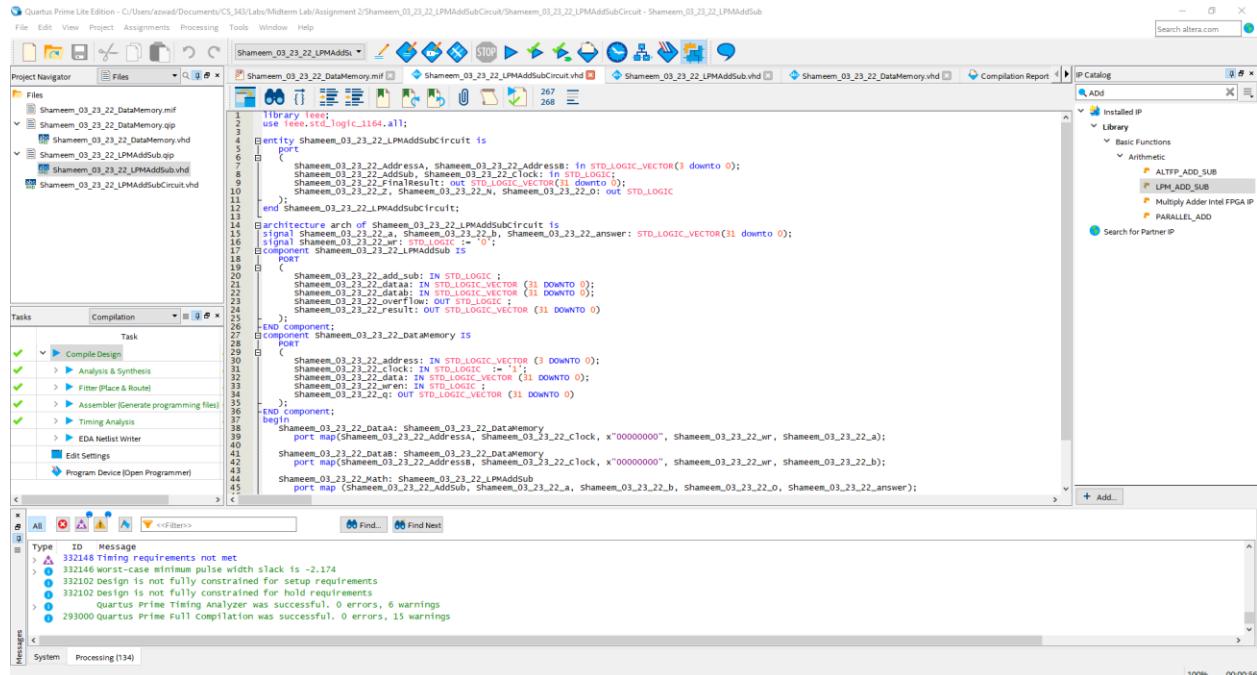


Figure 19: VHDL code for the LPMAddSubCircuit. This circuit utilizes LPMAddSub and Data Memory as components to load data to memory and then add or subtract the data to output results.

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

- Project Navigator:** Shows files: shameen_03_23_22_DataMemory.mif, shameen_03_23_22_DataMemory.vhd, shameen_03_23_22_LPMAddSub.vhd, shameen_03_23_22_LPMAddSubCircuit.vhd.
- Code Editor:** Displays VHDL code for the LPMAddSubCircuit. The code defines a component shameen_03_23_22_LPMAddSub with various ports and an architecture section. It includes a map clause for the component and a begin block with a process.
- IP Catalog:** Shows the LPM_ADD_SUB IP core under the Arithmetic category.
- Messages:** Shows compilation messages including timing requirements, worst-case minimum pulse width slack, and design status.
- System:** Shows processing tasks (134).

Figure 20: VHDL code for LPMAddSubCircuit continued.

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

- Project Navigator:** Shows files: shameen_03_23_22_DataMemory.mif, shameen_03_23_22_DataMemory.vhd, shameen_03_23_22_LPMAddSub.vhd, shameen_03_23_22_LPMAddSubCircuit.vhd.
- Code Editor:** Displays VHDL code for the LPMAddSub component. It includes a library declaration for IEEE, a component definition for Tpm_add_sub, and an architecture section named SYN_OF. The architecture section uses the LPM_ADD_SUB IP core.
- IP Catalog:** Shows the LPM_ADD_SUB IP core under the Arithmetic category.
- Messages:** Shows compilation messages including timing requirements, worst-case minimum pulse width slack, and design status.
- System:** Shows processing tasks (134).

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

```

library IEEE;
use IEEE.STD_LOGIC.all;
library altera_mf;
use altera_mf.altera_mf_components.all;
entity shameen_03_23_22_DataMemory is
  port (
    Shameen_03_23_22_address : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    Shameen_03_23_22_Clock : IN STD_LOGIC := '1';
    Shameen_03_23_22_wren : IN STD_LOGIC;
    Shameen_03_23_22_wdata : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    Shameen_03_23_22_wire0 : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
  );
end entity;

architecture SYN of shameen_03_23_22_datamemory is
begin
  Shameen_03_23_22_q <= shameen_03_23_22_sub_wire0(31 DOWNTO 0);
end architecture;

```

Tasks Compilation

- Compile Design
- Analysis & Synthesis
- Fitter (Place & Route)
- Assembler (Generate programming files)
- Timing Analysis
- EDA Netlist Writer
- Edit Settings
- Program Device (Open Programmer)

IP Catalog

Messages

System Processing (134)

Figure 22: LPMAddSub VHDL code continued.

```

library IEEE;
use IEEE.STD_LOGIC.all;
library altera_mf;
use altera_mf.altera_mf_components.all;
entity shameen_03_23_22_DataMemory is
  port (
    Shameen_03_23_22_address : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    Shameen_03_23_22_Clock : IN STD_LOGIC := '1';
    Shameen_03_23_22_wren : IN STD_LOGIC;
    Shameen_03_23_22_wdata : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
    Shameen_03_23_22_wire0 : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
  );
end entity;

architecture SYN of shameen_03_23_22_datamemory is
begin
  Shameen_03_23_22_q <= shameen_03_23_22_sub_wire0(31 DOWNTO 0);
end architecture;

```

Tasks Compilation

- Compile Design
- Analysis & Synthesis
- Fitter (Place & Route)
- Assembler (Generate programming files)
- Timing Analysis
- EDA Netlist Writer
- Edit Settings
- Program Device (Open Programmer)

IP Catalog

Messages

System Processing (134)

Figure 23: VHDL code for Data Memory module.

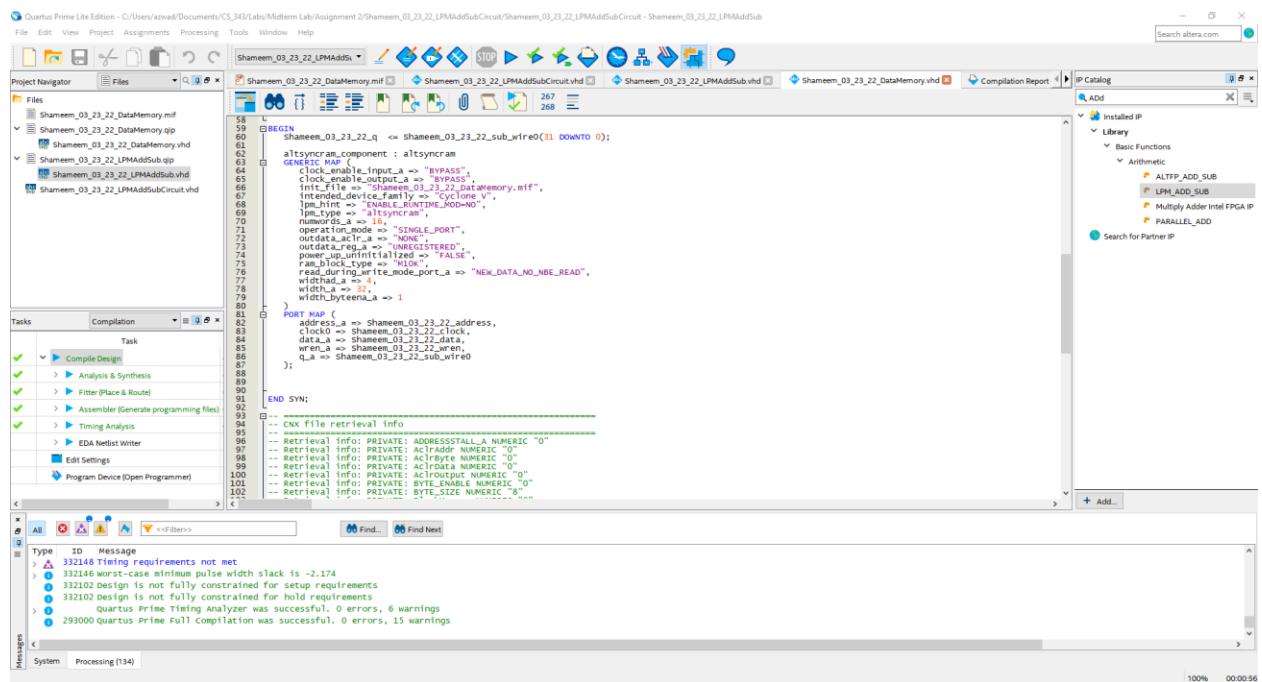


Figure 24: VHDL code continued for Data Memory module.

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.

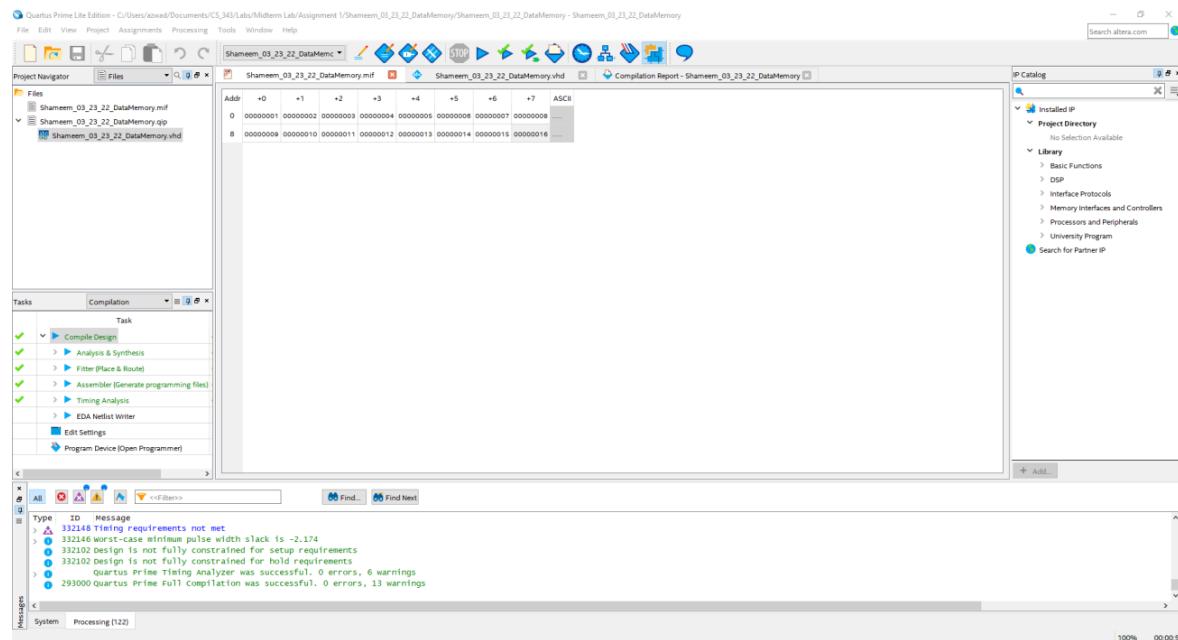


Figure 25: MIF file for Data Memory module.

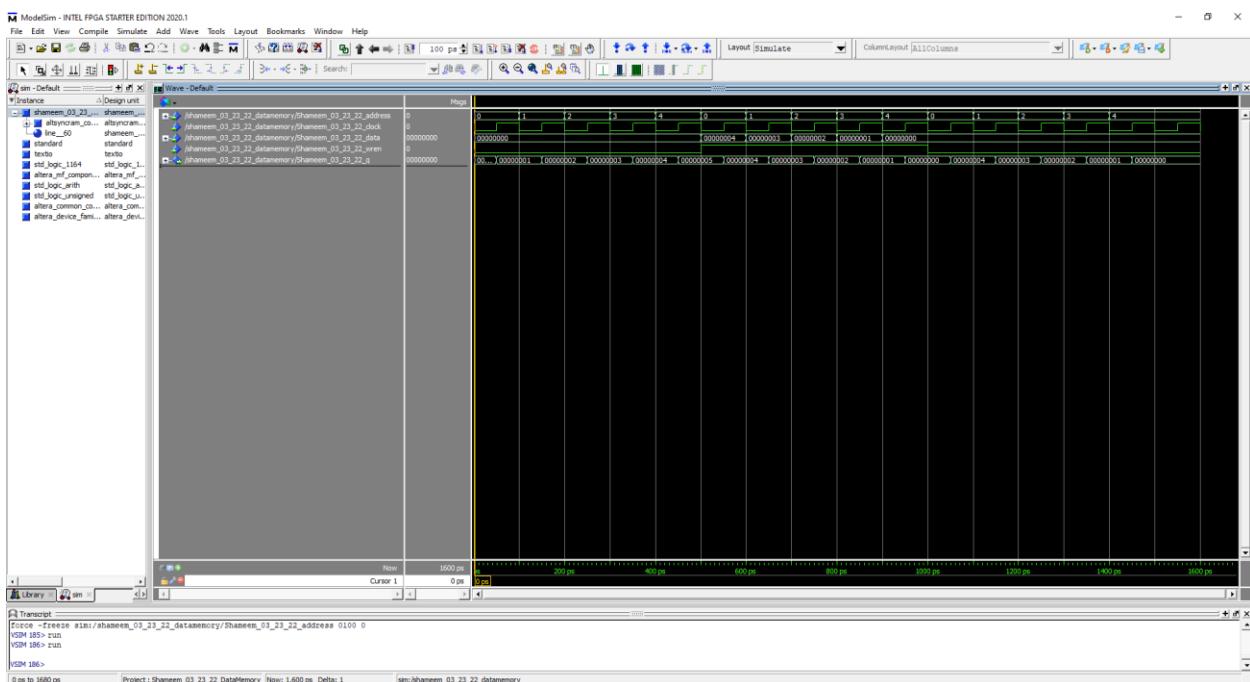


Figure 26: The waveform from the simulation of the Data Memory module using the MIF file in figure 25. Note: The simulation's waveforms are shown in **Hexadecimal**

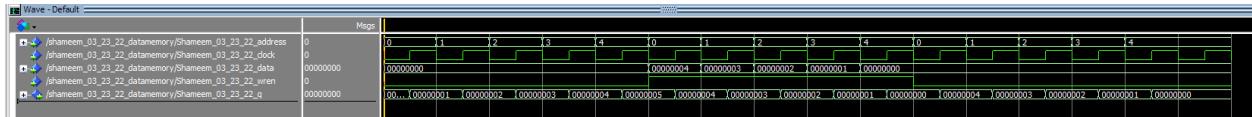


Figure 27: Waveform Simulation from figure 26 zoomed in.

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.

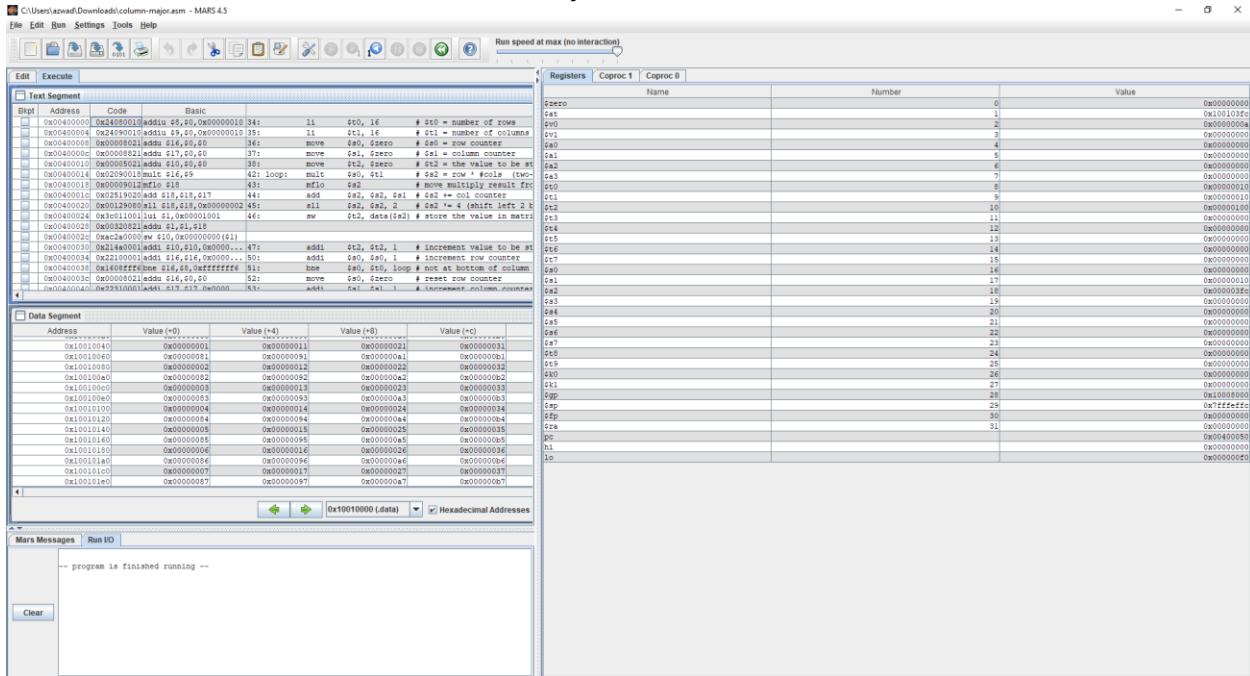


Figure 28: Mars output screen after fully running column-major.asm

| Text Segment | | | |
|--------------|------------|------------|-------------------------------|
| Bkpt | Address | Code | Basic |
| | 0x00400000 | 0x24080010 | addiu \$t0,\$0,0x00000010 34: |
| | 0x00400004 | 0x24090010 | addiu \$t1,\$0,0x00000010 35: |
| | 0x00400008 | 0x00008021 | addu \$t0,\$16,\$0,0 36: |
| | 0x0040000c | 0x00008821 | addu \$t1,\$0,\$0 37: |
| | 0x00400010 | 0x00005021 | addu \$t0,\$10,\$0,0 38: |
| | 0x00400014 | 0x02090018 | mult \$t0,\$9 42: loop: |
| | 0x00400018 | 0x00009012 | mflo \$t1 43: |
| | 0x0040001c | 0x02519020 | add \$t2,\$t1,\$17 44: |
| | 0x00400020 | 0x00129080 | sll \$t2,\$t2,2 45: |
| | 0x00400024 | 0x3c011001 | lui \$t2,1 46: |
| | 0x00400028 | 0x00320821 | addu \$t2,\$t2,\$18 47: |
| | 0x0040002c | 0xac2a0000 | sw \$t2,0x00000000(\$t1) 48: |
| | 0x00400030 | 0x214a0001 | addi \$t2,\$10,0x0000... 49: |
| | 0x00400034 | 0x22100001 | addi \$t2,\$16,0x0000... 50: |
| | 0x00400038 | 0x1608ffff | bne \$t2,\$t0,0xfffffff6 51: |
| | 0x0040003c | 0x00008021 | addu \$t2,\$t2,\$0,0 52: |
| | 0x00400040 | 0x22310001 | addi \$t2,\$t2,1 53: |

Figure 29: Zoomed in Mars text segment.

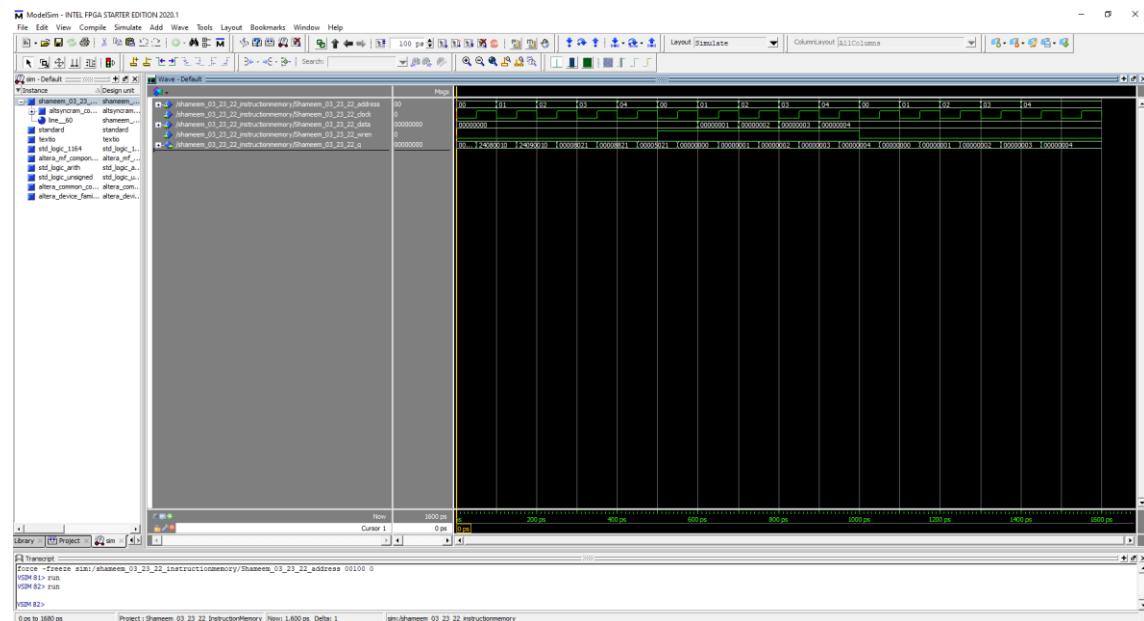


Figure 30: Instruction Memory waveforms from simulation.
Note: The simulation's waveforms are shown in **Hexadecimal**

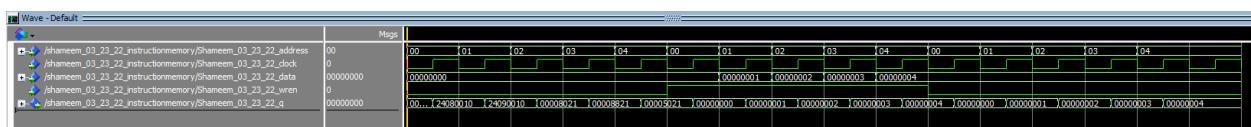


Figure 31: Instruction Memory waveforms simulation zoomed in.
Note: The simulation's waveforms are shown in **Hexadecimal**

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

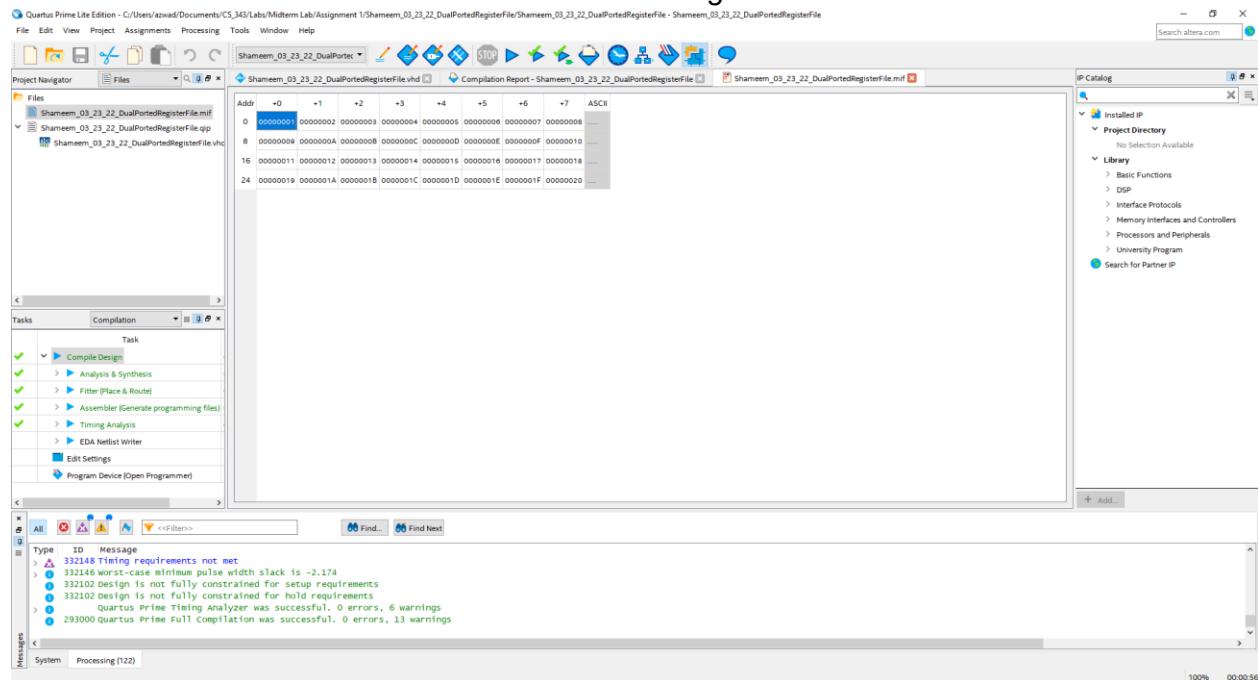


Figure 32: MIF file for Dual Ported Register File

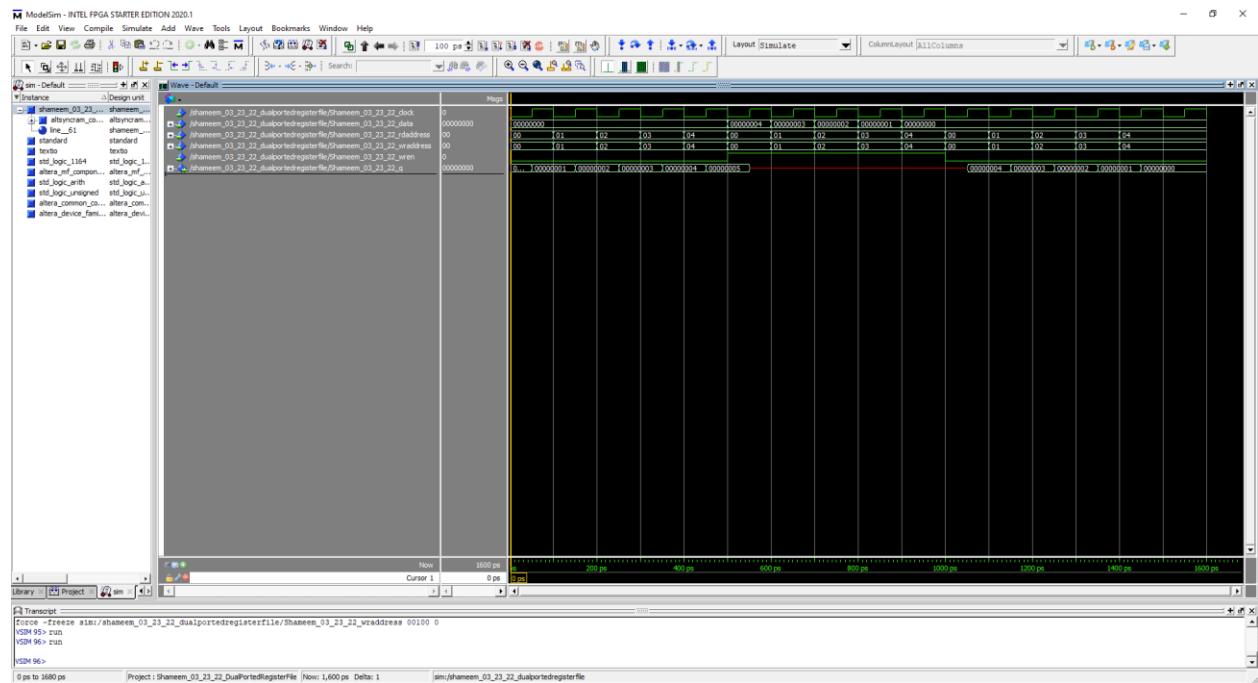


Figure 33: Simulation waveform for the Dual Ported Register File using the MIF file in figure 32.
Note: The simulation's waveforms are shown in **Hexadecimal**

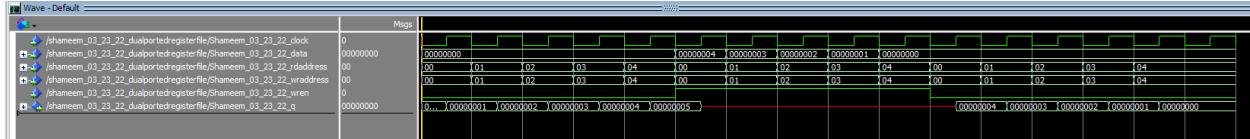


Figure 34: Simulation waveform for the Dual Ported Register File zoomed in.

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.

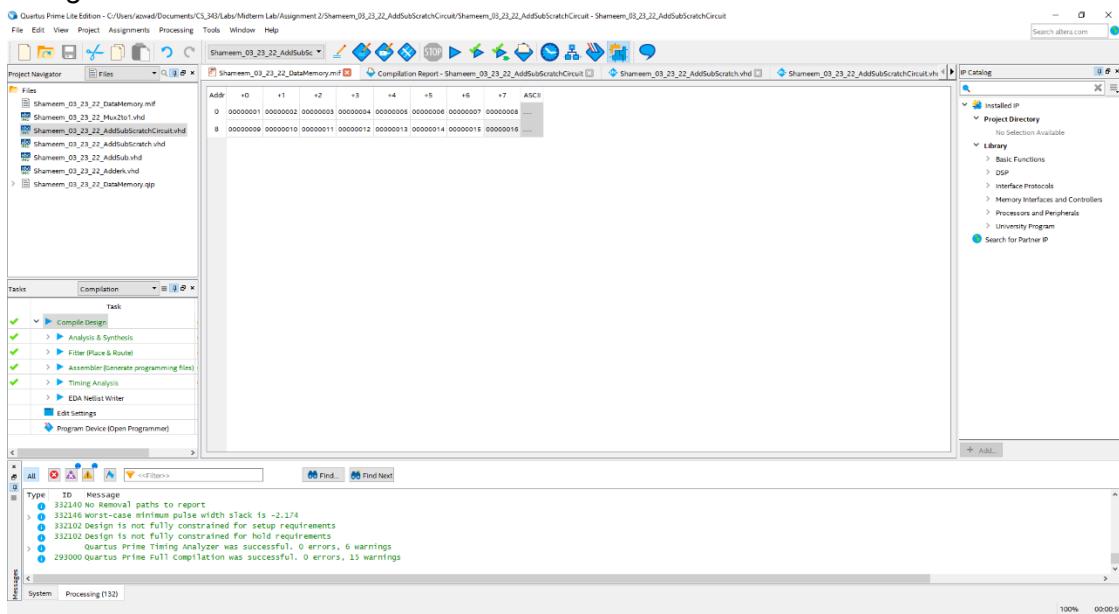


Figure 35: MIF file for the Add/Sub circuit from scratch

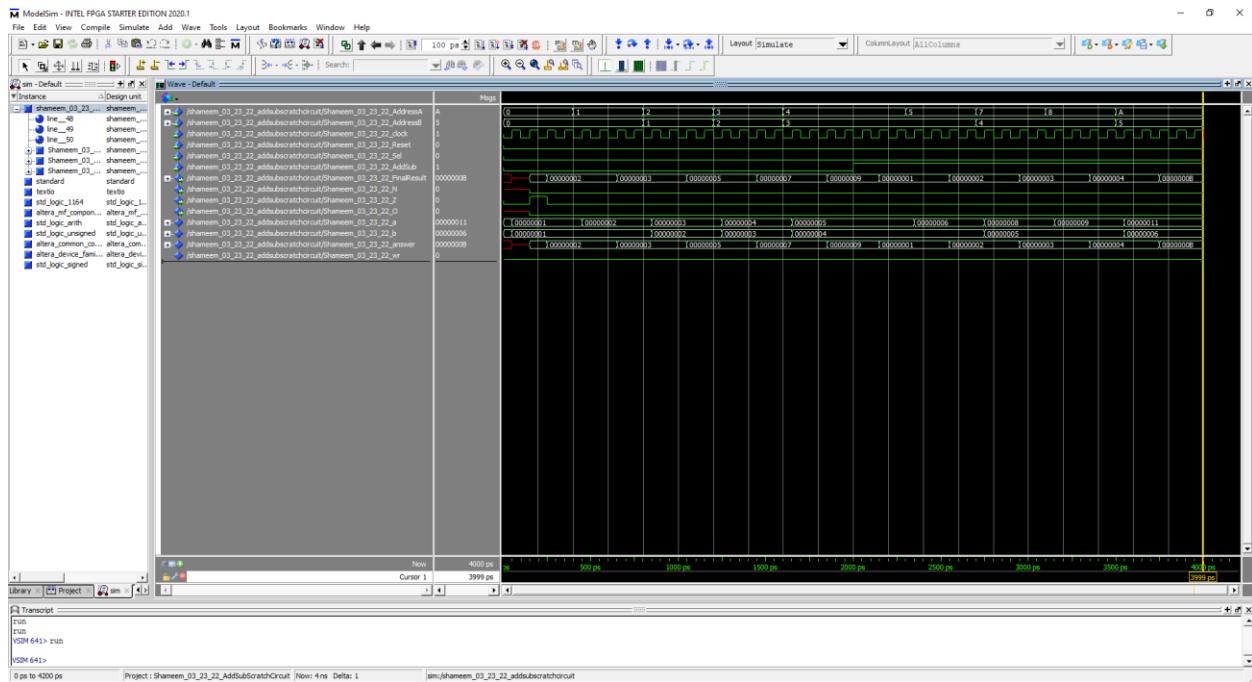


Figure 36: Simulation of ScratchAddSubCircuit using MIF file from figure 35

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

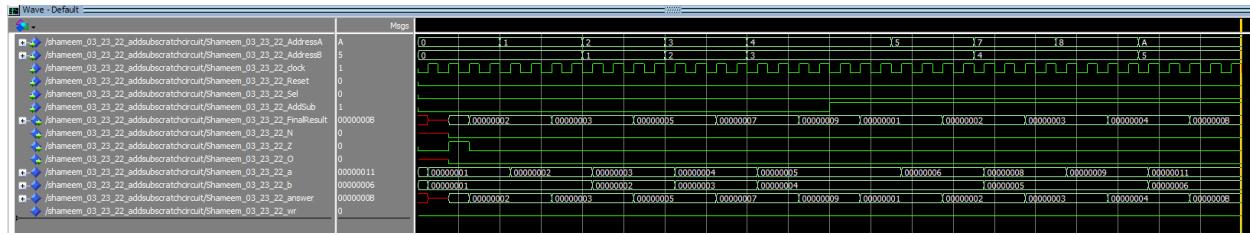


Figure 37: Zoomed in waveform simulation from figure 36

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.

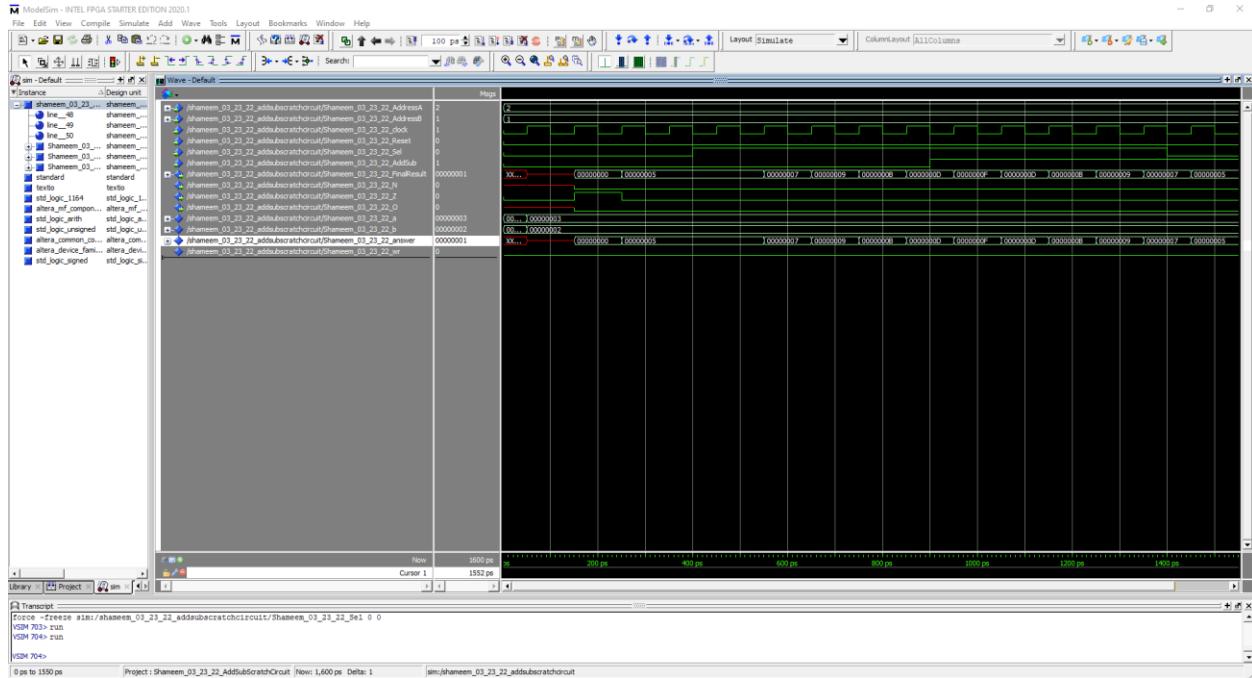


Figure 38: Simulation of ScratchAddSubCircuit using MIF file from figure 35
Note: The simulation's waveforms are shown in **Hexadecimal**

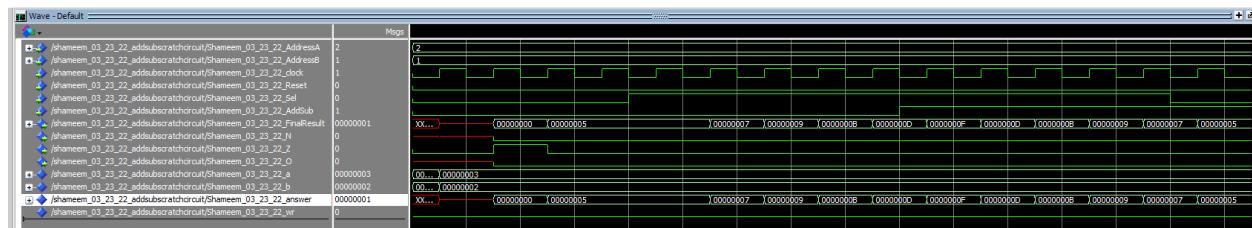


Figure 39: Zoomed in waveforms from simulation in figure 38
Note: The simulation's waveforms are shown in **Hexadecimal**

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.

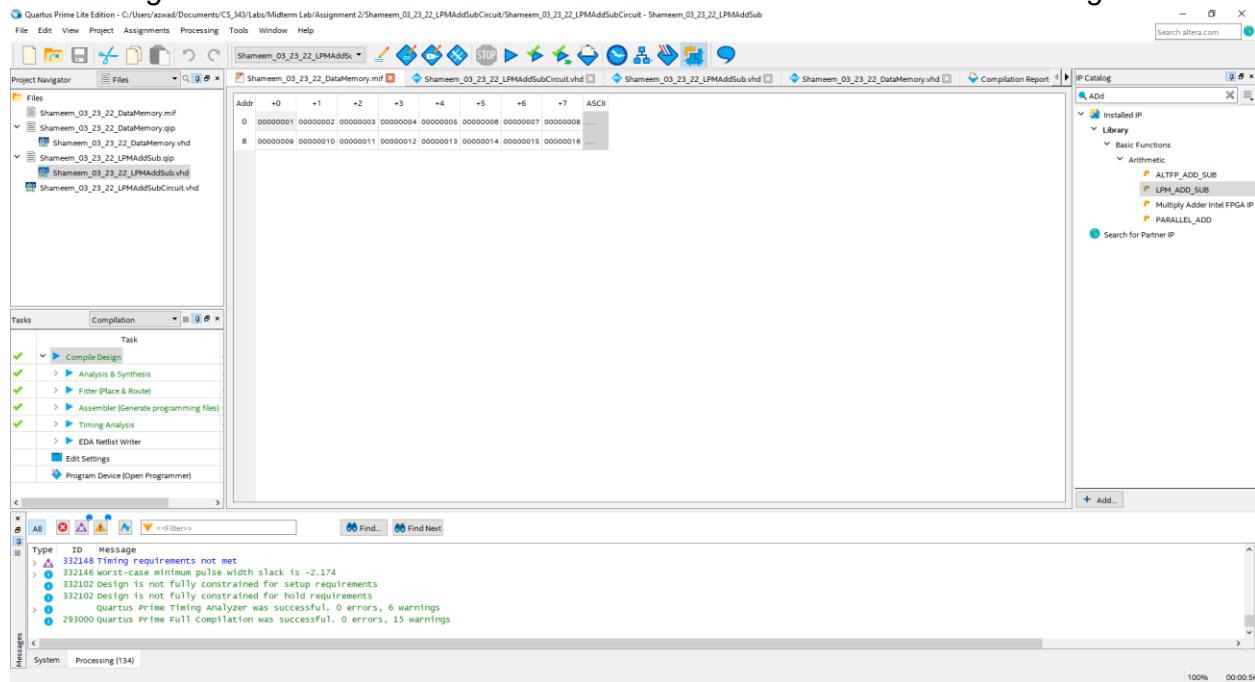


Figure 40: MIF file for the LPMAddSubCircuit

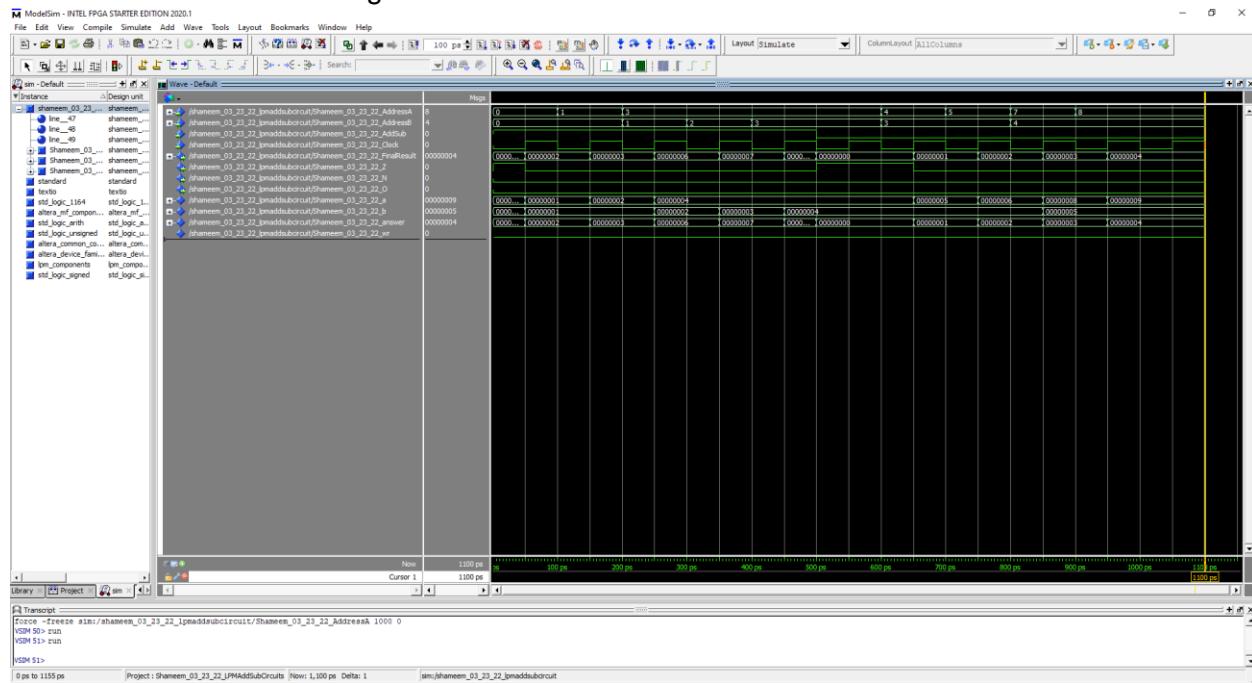


Figure 41: LPMAddSubCircuit simulation waveforms
Note: The simulation's waveforms are shown in **Hexadecimal**

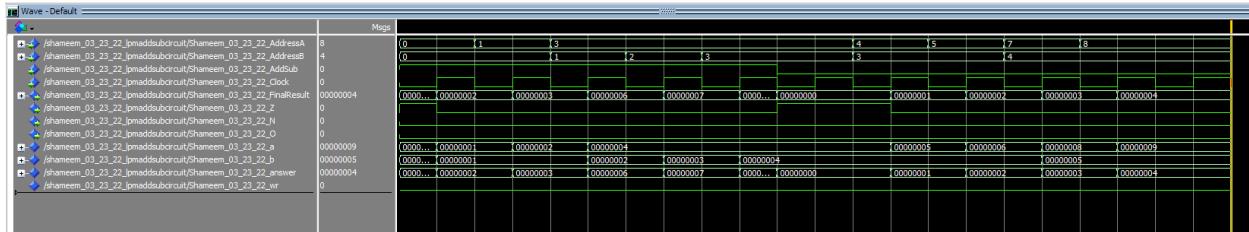


Figure 39: Zoomed in waveforms from simulation in figure 38

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.