Laboratory Exercise 3: Integrating ADD/SUB LPM module with SRAM LPM module.

Zi Xuan Li

The Grove School of Engineering, The City College of New York

CSC 34300 5DE[43223]: Computer Systems Design Laboratory

Professor Izidor Gertner, TA: Albi Arapi

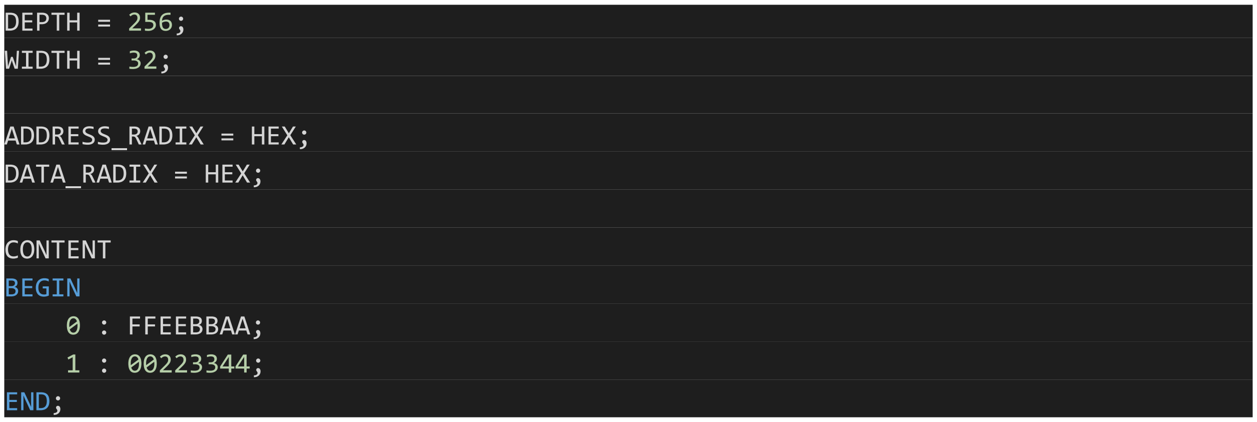
April 3rd, 2024

Table of Contents

Objective: The objective of this exercise is to design and verify the functionality of the ADD/SUB LPM module alongside the SRAM LPM module using Quartus and ModelSim.

Functionality and Specifications:

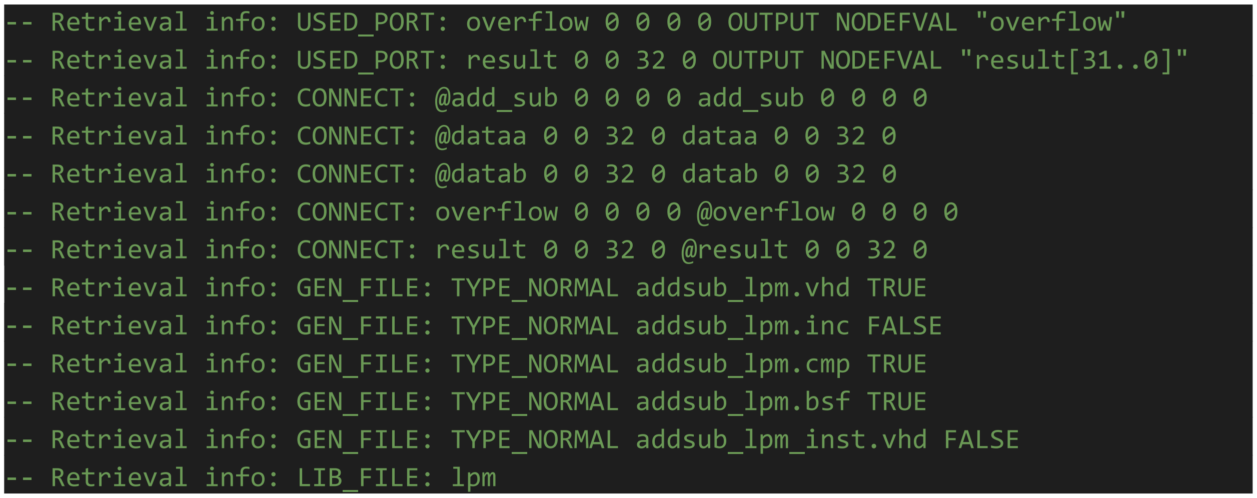
What is the VHDL code of the circuit?



```
DEPTH = 256;
WIDTH = 32;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT
BEGIN
    0 : FFEEBBAA;
    1 : 00223344;
END;
```

Figure 1: This is the .mif file that the SRAM I will be using is initialized with.

```
-- megafunction wizard: %LPM_ADD_SUB%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: LPM_ADD_SUB

-- ============================================================
-- File Name: addsub_lpm.vhd
-- Megafunction Name(s):
--          LPM_ADD_SUB
--
-- Simulation Library Files(s):
--          lpm
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 22.1std.0 Build 915 10/25/2022 SC Lite Edition
-- ************************************************************


--Copyright (C) 2022  Intel Corporation. All rights reserved.
--Your use of Intel Corporation's design tools, logic functions
--and other software and tools, and any partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Intel Program License
--Subscription Agreement, the Intel Quartus Prime License Agreement,
--the Intel FPGA IP License Agreement, or other applicable license
--agreement, including, without limitation, that your use is for
--the sole purpose of programming logic devices manufactured by
--Intel and sold by Intel or its authorized distributors.  Please
--refer to the applicable agreement for further details, at
--https://fpgasoftware.intel.com/eula.


LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.all;

ENTITY addsub_lpm IS
    PORT
    (
        add_sub     : IN STD_LOGIC ;
        dataa       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        datab       : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        overflow        : OUT STD_LOGIC ;
        result      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END addsub_lpm;


ARCHITECTURE SYN OF addsub_lpm IS

    SIGNAL sub_wire0    : STD_LOGIC ;
    SIGNAL sub_wire1    : STD_LOGIC_VECTOR (31 DOWNTO 0);



    COMPONENT lpm_add_sub
    GENERIC (
        lpm_direction       : STRING;
        lpm_hint        : STRING;
        lpm_representation      : STRING;
        lpm_type        : STRING;
        lpm_width       : NATURAL
    );
    PORT (
            add_sub : IN STD_LOGIC ;
            dataa   : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            datab   : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            overflow    : OUT STD_LOGIC ;
            result  : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
    END COMPONENT;

BEGIN
    overflow    <= sub_wire0;
    result    <= sub_wire1(31 DOWNTO 0);

    LPM_ADD_SUB_component : LPM_ADD_SUB
    GENERIC MAP (
        lpm_direction => "UNUSED",
        lpm_hint => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
        lpm_representation => "UNSIGNED",
        lpm_type => "LPM_ADD_SUB",
        lpm_width => 32
    )
    PORT MAP (
        add_sub => add_sub,
        dataa => dataa,
        datab => datab,
        overflow => sub_wire0,
        result => sub_wire1
    );



END SYN;

-- ============================================================
-- CNX file retrieval info
-- ============================================================
-- Retrieval info: PRIVATE: CarryIn NUMERIC "0"
-- Retrieval info: PRIVATE: CarryOut NUMERIC "0"
-- Retrieval info: PRIVATE: ConstantA NUMERIC "0"
-- Retrieval info: PRIVATE: ConstantB NUMERIC "0"
-- Retrieval info: PRIVATE: Function NUMERIC "2"
-- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
-- Retrieval info: PRIVATE: LPM_PIPELINE NUMERIC "0"
-- Retrieval info: PRIVATE: Latency NUMERIC "0"
-- Retrieval info: PRIVATE: Overflow NUMERIC "1"
-- Retrieval info: PRIVATE: RadixA NUMERIC "10"
-- Retrieval info: PRIVATE: RadixB NUMERIC "10"
-- Retrieval info: PRIVATE: Representation NUMERIC "1"
-- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
-- Retrieval info: PRIVATE: ValidCtA NUMERIC "0"
-- Retrieval info: PRIVATE: ValidCtB NUMERIC "0"
-- Retrieval info: PRIVATE: WhichConstant NUMERIC "0"
-- Retrieval info: PRIVATE: aclr NUMERIC "0"
-- Retrieval info: PRIVATE: clken NUMERIC "0"
-- Retrieval info: PRIVATE: nBit NUMERIC "32"
-- Retrieval info: PRIVATE: new_diagram STRING "1"
-- Retrieval info: LIBRARY: lpm lpm.lpm_components.all
-- Retrieval info: CONSTANT: LPM_DIRECTION STRING "UNUSED"
-- Retrieval info: CONSTANT: LPM_HINT STRING
"ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO"
-- Retrieval info: CONSTANT: LPM_REPRESENTATION STRING "UNSIGNED"
-- Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_ADD_SUB"
-- Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "32"
-- Retrieval info: USED_PORT: add_sub 0 0 0 0 INPUT NODEFVAL "add_sub"
-- Retrieval info: USED_PORT: dataa 0 0 32 0 INPUT NODEFVAL "dataa[31..0]"
-- Retrieval info: USED_PORT: datab 0 0 32 0 INPUT NODEFVAL "datab[31..0]"
```

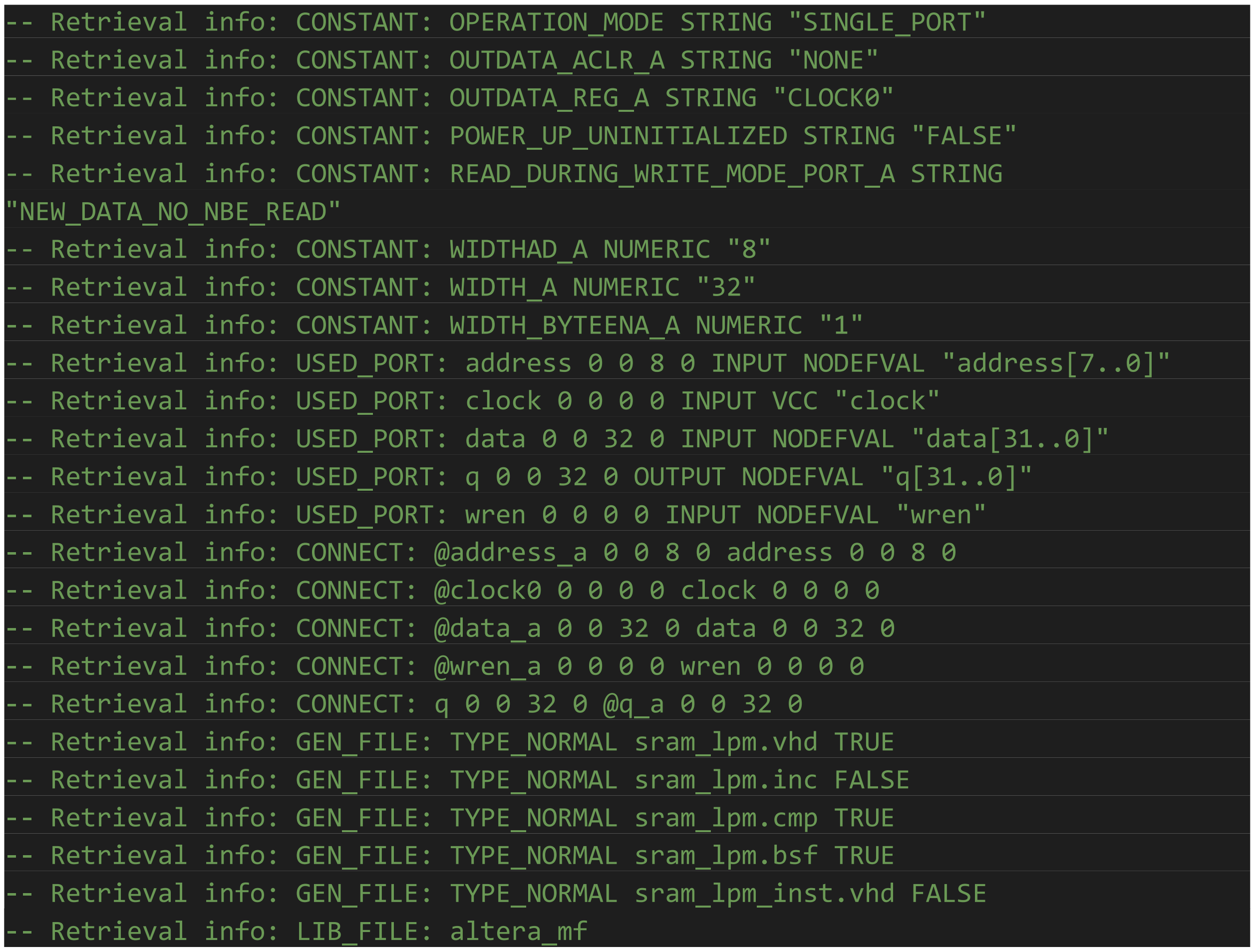


```
-- Retrieval info: USED_PORT: overflow 0 0 0 0 OUTPUT NODEFVAL "overflow"
-- Retrieval info: USED_PORT: result 0 0 32 0 OUTPUT NODEFVAL "result[31..0]"
-- Retrieval info: CONNECT: @add_sub 0 0 0 0 add_sub 0 0 0 0
-- Retrieval info: CONNECT: @dataa 0 0 32 0 dataa 0 0 32 0
-- Retrieval info: CONNECT: @datab 0 0 32 0 datab 0 0 32 0
-- Retrieval info: CONNECT: overflow 0 0 0 0 @overflow 0 0 0 0
-- Retrieval info: CONNECT: result 0 0 32 0 @result 0 0 32 0
-- Retrieval info: GEN_FILE: TYPE_NORMAL addsub_lpm.vhd TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL addsub_lpm.inc FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL addsub_lpm.cmp TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL addsub_lpm.bsf TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL addsub_lpm_inst.vhd FALSE
-- Retrieval info: LIB_FILE: lpm
```

Figure 2: This is the VHDL code generated from the LPM_ADD_SUB wizard when creating an ADD/SUB unit that can perform arithmetic operations on the integers 0xFFEEBBAA and 0x00223344. The width of the ADD/SUB unit is 32 bits and has a input for addition and subtraction as well as an output for overflow.

```
-- megafunction wizard: %RAM: 1-PORT%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altsyncram

-- ============================================================
-- File Name: sram_lpm.vhd
-- Megafunction Name(s):
--          altsyncram
--
-- Simulation Library Files(s):
--          altera_mf
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 22.1std.0 Build 915 10/25/2022 SC Lite Edition
-- ************************************************************


--Copyright (C) 2022  Intel Corporation. All rights reserved.
--Your use of Intel Corporation's design tools, logic functions
--and other software and tools, and any partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Intel Program License
--Subscription Agreement, the Intel Quartus Prime License Agreement,
--the Intel FPGA IP License Agreement, or other applicable license
--agreement, including, without limitation, that your use is for
--the sole purpose of programming logic devices manufactured by
--Intel and sold by Intel or its authorized distributors.  Please
--refer to the applicable agreement for further details, at
--https://fpgasoftware.intel.com/eula.


LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY sram_lpm IS
    PORT
    (
        address     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        clock       : IN STD_LOGIC  := '1';
        data        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
        wren        : IN STD_LOGIC ;
        q       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
    );
END sram_lpm;


ARCHITECTURE SYN OF sram_lpm IS

    SIGNAL sub_wire0    : STD_LOGIC_VECTOR (31 DOWNTO 0);

BEGIN
    q    <= sub_wire0(31 DOWNTO 0);

    altsyncram_component : altsyncram
    GENERIC MAP (
        clock_enable_input_a => "BYPASS",
        clock_enable_output_a => "BYPASS",
        init_file => "init.mif",
        intended_device_family => "Cyclone V",
        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
        lpm_type => "altsyncram",
        numwords_a => 256,
        operation_mode => "SINGLE_PORT",
        outdata_aclr_a => "NONE",
        outdata_reg_a => "CLOCK0",
        power_up_uninitialized => "FALSE",
        read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
        widthad_a => 8,
        width_a => 32,
        width_byteena_a => 1
    )
    PORT MAP (
        address_a => address,
        clock0 => clock,
        data_a => data,
        wren_a => wren,
        q_a => sub_wire0
    );



END SYN;
```

```
-- ============================================================
-- CNX file retrieval info
-- ============================================================
-- Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
-- Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
-- Retrieval info: PRIVATE: AclrByte NUMERIC "0"
-- Retrieval info: PRIVATE: AclrData NUMERIC "0"
-- Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
-- Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
-- Retrieval info: PRIVATE: Clken NUMERIC "0"
-- Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
-- Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
-- Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
-- Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
-- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
-- Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
-- Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
-- Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
-- Retrieval info: PRIVATE: MIFfilename STRING "init.mif"
-- Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "256"
-- Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
-- Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
-- Retrieval info: PRIVATE: RegAddr NUMERIC "1"
-- Retrieval info: PRIVATE: RegData NUMERIC "1"
-- Retrieval info: PRIVATE: RegOutput NUMERIC "1"
-- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
-- Retrieval info: PRIVATE: SingleClock NUMERIC "1"
-- Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
-- Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
-- Retrieval info: PRIVATE: WidthAddr NUMERIC "8"
-- Retrieval info: PRIVATE: WidthData NUMERIC "32"
-- Retrieval info: PRIVATE: rden NUMERIC "0"
-- Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
-- Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
-- Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
-- Retrieval info: CONSTANT: INIT_FILE STRING "init.mif"
-- Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
-- Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
-- Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
-- Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "256"
```

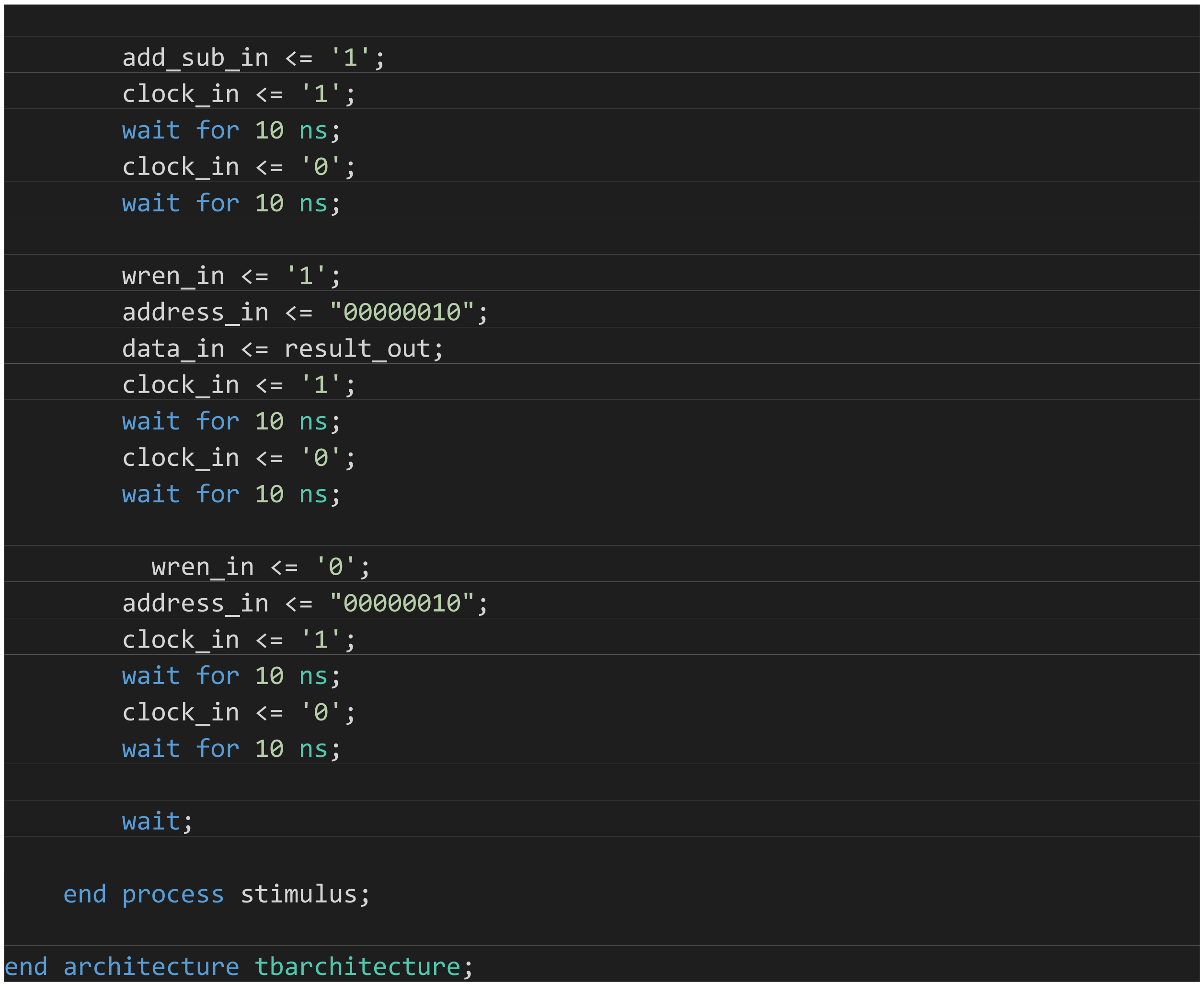


```
-- Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
-- Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
-- Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
-- Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
-- Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING
"NEW_DATA_NO_NBE_READ"
-- Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "8"
-- Retrieval info: CONSTANT: WIDTH_A NUMERIC "32"
-- Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
-- Retrieval info: USED_PORT: address 0 0 8 0 INPUT NODEFVAL "address[7..0]"
-- Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
-- Retrieval info: USED_PORT: data 0 0 32 0 INPUT NODEFVAL "data[31..0]"
-- Retrieval info: USED_PORT: q 0 0 32 0 OUTPUT NODEFVAL "q[31..0]"
-- Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
-- Retrieval info: CONNECT: @address_a 0 0 8 0 address 0 0 8 0
-- Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
-- Retrieval info: CONNECT: @data_a 0 0 32 0 data 0 0 32 0
-- Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
-- Retrieval info: CONNECT: q 0 0 32 0 @q_a 0 0 32 0
-- Retrieval info: GEN_FILE: TYPE_NORMAL sram_lpm.vhd TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL sram_lpm.inc FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL sram_lpm.cmp TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL sram_lpm.bsf TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL sram_lpm_inst.vhd FALSE
-- Retrieval info: LIB_FILE: altera_mf
```

Figure 3: This is the VHDL code generated from the RAM: 1-PORT wizard. The width of the SRAM is 32 bits, and the depth is 256 addresses.

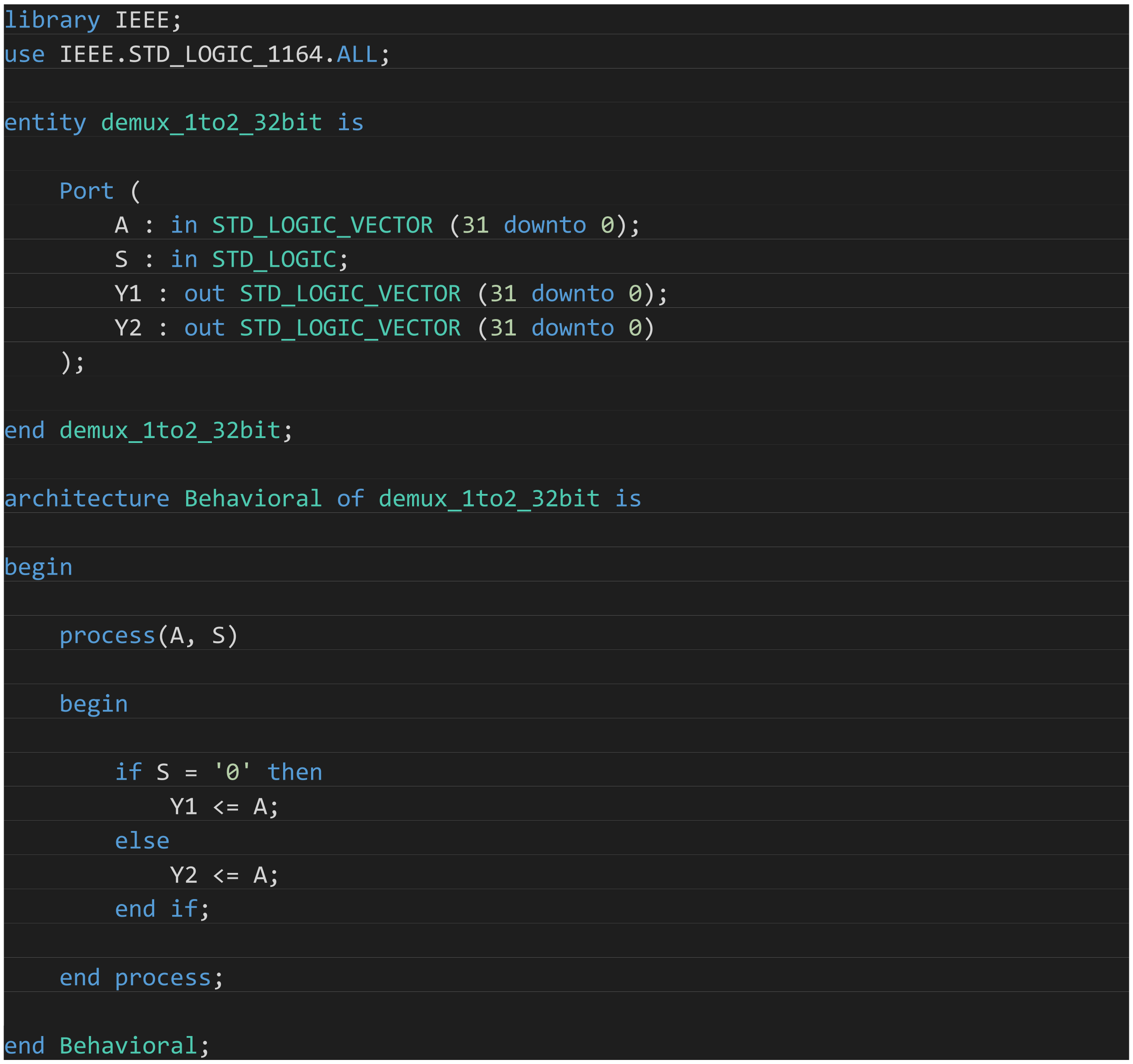
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity testbench is
end entity testbench;

architecture tbarchitecture of testbench is
    signal add_sub_in, overflow_out : std_logic;
    signal dataa_in, datab_in, result_out : std_logic_vector (31 downto 0);

    signal address_in : std_logic_vector (7 downto 0);
    signal clock_in, wren_in : std_logic;
    signal data_in, q_out : std_logic_vector (31 downto 0);

    component addsub_lpm is
        PORT (
            add_sub   : IN STD_LOGIC ;
            dataa     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            datab     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            overflow  : OUT STD_LOGIC ;
            result    : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
              );
    end component addsub_lpm;

    component sram_lpm is
        PORT (
            address   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            clock     : IN STD_LOGIC  := '1';
            data      : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
            wren      : IN STD_LOGIC ;
            q         : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
              );
    end component sram_lpm;

begin

    -- Instantiate ADD/SUB unit
    ADDSUB_inst: addsub_lpm
        port map (
            add_sub => add_sub_in,
            dataa => dataa_in,
            datab => datab_in,
            overflow => overflow_out,
            result => result_out
        );
```

```
-- Instantiate SRAM module
SRAM_inst: sram_lpm
    port map (
        address => address_in,
        clock => clock_in,
        data => data_in,
        wren => wren_in,
        q => q_out
    );

-- Stimulus process
stimulus: process
begin
    wren_in <= '1';
    address_in <= "00000000";
    data_in <= x"FFEEBBAA";
    clock_in <= '1';
    wait for 10 ns;
    clock_in <= '0';
    wait for 10 ns;

      wren_in <= '1';
    address_in <= "00000001";
    data_in <= x"00223344";
    clock_in <= '1';
    wait for 10 ns;
    clock_in <= '0';
    wait for 10 ns;

    wren_in <= '0';
    address_in <= "00000000";
    clock_in <= '1';
    wait for 10 ns;
    clock_in <= '0';
    wait for 10 ns;
    dataa_in <= q_out;

    wren_in <= '0';
    address_in <= "00000001";
    clock_in <= '1';
    wait for 10 ns;
    clock_in <= '0';
    wait for 10 ns;
    datab_in <= q_out;
```



```vhdl
        add_sub_in <= '1';
        clock_in <= '1';
        wait for 10 ns;
        clock_in <= '0';
        wait for 10 ns;

        wren_in <= '1';
        address_in <= "00000010";
        data_in <= result_out;
        clock_in <= '1';
        wait for 10 ns;
        clock_in <= '0';
        wait for 10 ns;

          wren_in <= '0';
        address_in <= "00000010";
        clock_in <= '1';
        wait for 10 ns;
        clock_in <= '0';
        wait for 10 ns;

        wait;

    end process stimulus;

end architecture tbarchitecture;
```

Figure 4: This is the VHDL testbench code I wrote to test the functionality of the SRAM alongside the ADD/SUB unit.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity demux_1to2_32bit is

    Port (
        A : in STD_LOGIC_VECTOR (31 downto 0);
        S : in STD_LOGIC;
        Y1 : out STD_LOGIC_VECTOR (31 downto 0);
        Y2 : out STD_LOGIC_VECTOR (31 downto 0)
    );

end demux_1to2_32bit;

architecture Behavioral of demux_1to2_32bit is

begin

    process(A, S)

    begin

        if S = '0' then
            Y1 <= A;
        else
            Y2 <= A;
        end if;

    end process;

end Behavioral;
```

Figure 5: This is the VHDL code I wrote to generate the 32-bit 1 to 2 demultiplexer for the bdf file.

Include a screenshot of your design steps and of final the circuit.

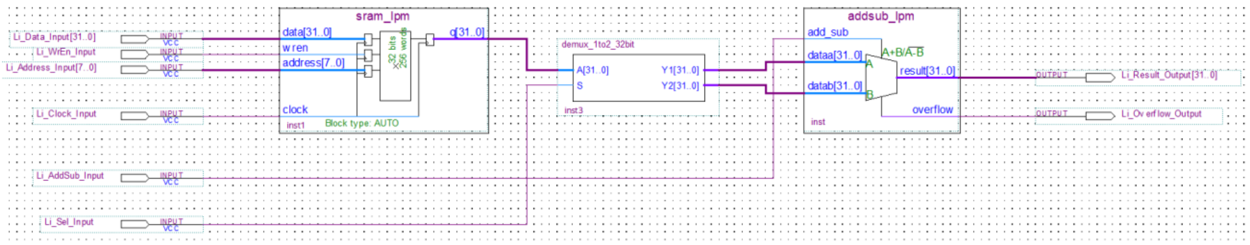


Figure 6: This is the .bdf file of the SRAM LPM working alongside the ADD/SUB LPM.

Simulation:

Include a screenshot of the vector waveform output file for all possible inputs.

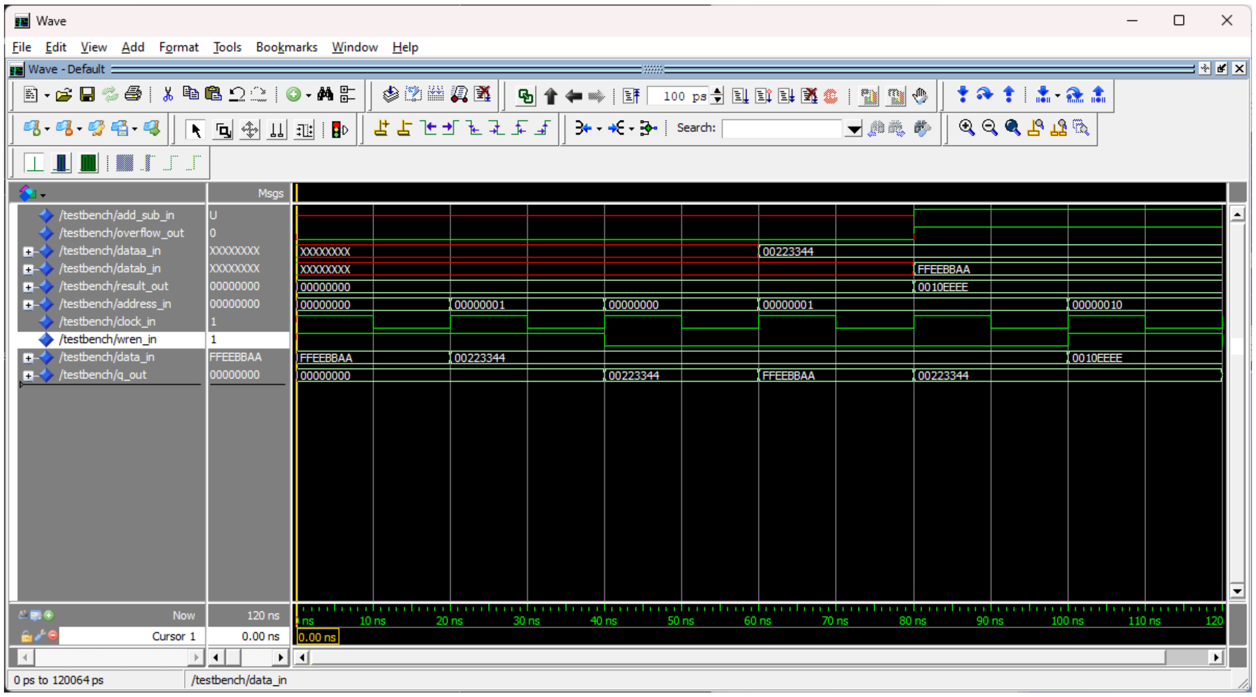

Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/testbench/add_sub_in U
/testbench/overflow_out 0
/testbench/dataa_in XXXXXXXX
/testbench/datab_in XXXXXXXX
/testbench/result_out 00000000
/testbench/address_in 00000000
/testbench/clock_in 1
/testbench/wren_in 1
/testbench/data_in FFEEBBAA
/testbench/q_out 00000000
XXXXXXXX
00223344
FFEEBBAA
0010EEEE
00000001
00000010
Now 120 ns
Cursor 1 0.00 ns
ns 10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns 100 ns 110 ns 120
0.00 ns
0 ps to 120064 ps
/testbench/data_in

Figure 7: This is the vector waveform output of the testbench in Figure 5. During the first 0ns to 40ns, the code instructs that 0xFFEEBBAA and 0x00223344 be written in address 00000000 and address 00000001 respectively in the SRAM. During the next 40ns to 80ns, the code instructs the stored data in those addresses be assigned to data A and data B in the ADD/SUB unit. In the last

80ns to 120ns, the code instructs the ADD/SUB unit to add data A and data B, which results in overflow and takes the 32-bit result and stores it in the address 00000010 in the SRAM.

Conclusion: This exercise offered a deeper understanding of designing and testing an ADD/SUB unit alongside the SRAM LPM module. I learned how the parametrization of ADD/SUB LPM works alongside the functionalities of its input ports, control signal, output port. I also learned about how the parameterization of the SRAM LPM works alongside the functionalities of its addresses, data ports, write and read signals.