**EE496 Computer Architecture and VHDL**

Assignment 2: Crypto processor

Due date: 4 December 2017

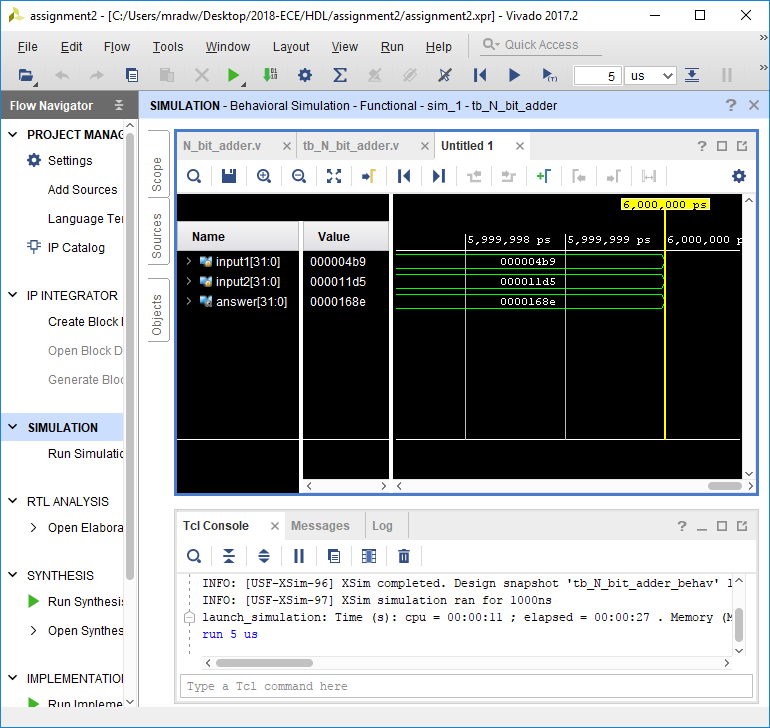
# **Section 1: Combinational Logic**

**Task: Write a Verilog module for an N-bit adder.**

|  |
| --- |
| **N\_bit\_adder** |
| `timescale 1ns / 1ps  module N\_bit\_adder(input1,input2,answer);  parameter N=32;  input [N-1:0] input1,input2;  output [N-1:0] answer;  wire carry\_out;  wire [N-1:0] carry;  genvar i;  generate  for(i=0;i<N;i=i+1)  begin: generate\_N\_bit\_Adder  if(i==0)  half\_adder f(input1[0],input2[0],answer[0],carry[0]);  else  full\_adder f(input1[i],input2[i],carry[i-1],answer[i],carry[i]);  end  assign carry\_out = carry[N-1];  endgenerate  endmodule  // Verilog code for half adder  module half\_adder(x,y,s,c);  input x,y;  output s,c;  assign s=x^y;  assign c=x&y;  endmodule // half adder  // Verilog code for full adder  module full\_adder(x,y,c\_in,s,c\_out);  input x,y,c\_in;  output s,c\_out;  assign s = (x^y) ^ c\_in;  assign c\_out = (y&c\_in)| (x&y) | (x&c\_in);  endmodule // full\_adder |

|  |
| --- |
| **ts\_N\_bit\_adder** |
| `timescale 1ns / 1ps  module tb\_N\_bit\_adder;  // Inputs  reg [31:0] input1;  reg [31:0] input2;  // Outputs  wire [31:0] answer;  // Instantiate the Unit Under Test (UUT)  N\_bit\_adder uut (  .input1(input1),  .input2(input2),  .answer(answer)  );  initial begin  // Initialize Inputs  input1 = 1209;  input2 = 4565;  #100;  // Add stimulus here  end  endmodule |

**Result:**

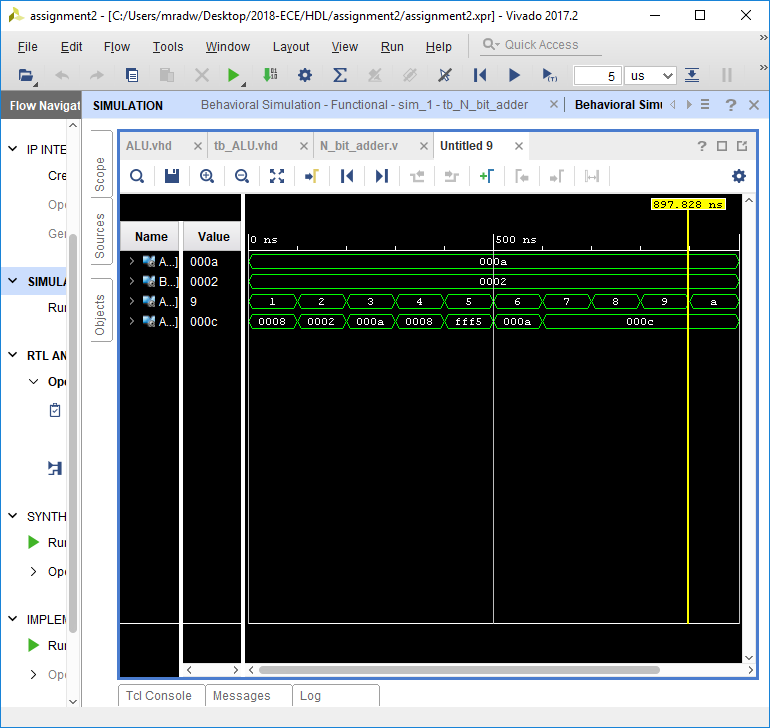


**Task: Implement the ALU described above in VHDL, making use of the Verilog N-bit adder that you implemented in the last section.**

|  |
| --- |
| **ALU** |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  entity ALU is  port (  ABUS: in std\_logic\_vector(15 downto 0); -- ABUS data input of the 16-bit ALU  BBUS: in std\_logic\_vector(15 downto 0); -- BBUS data input of the 16-bit ALU  ALUctrl: in std\_logic\_vector(3 downto 0); -- ALUctrl control input of the 16-bit ALU  ALUOUT: out std\_logic\_vector(15 downto 0) -- 16-bit data output of the 16-bit ALU  );  end ALU;  architecture Behavioral of ALU is  -- N-bit Adder in Verilog  component N\_bit\_adder is  generic (  N: integer:=32  );  port( input1: in std\_logic\_vector(N-1 downto 0);  input2: in std\_logic\_vector(N-1 downto 0);  answer: out std\_logic\_vector(N-1 downto 0)  );  end component N\_bit\_adder;  signal BBUS\_not: std\_logic\_vector(16-1 downto 0);  signal tmp\_out1: std\_logic\_vector(16-1 downto 0);  signal tmp\_out2: std\_logic\_vector(16-1 downto 0);  signal tmp: std\_logic\_vector(16-1 downto 0);  begin  -- instantiate Verilog N-bit Adder in VHDL code  u1\_N\_bit\_adder: N\_bit\_adder generic map ( N => 16) -- ABUS + BBUS  port map( input1 => ABUS, input2 => BBUS,answer => tmp\_out1 );  u2\_N\_bit\_adder: N\_bit\_adder generic map ( N => 16) -- ABUS + (~BBUS)  port map( input1 => ABUS, input2 => BBUS\_not,answer => tmp\_out2 );  u3\_N\_bit\_adder: N\_bit\_adder generic map ( N => 16) -- ABUS + (~BBUS) + 1 = ABUS - BBUS  port map( input1 => tmp\_out2, input2 => x"0001",answer => tmp );  BBUS\_not <= not BBUS;  -- Other instructions of the 16-bit ALU in VHDL  process(ALUctrl,ABUS,BBUS,tmp\_out1,tmp)  begin  case(ALUctrl) is  when "0000" => ALUOUT <= tmp\_out1; -- ADD  when "0001" => ALUOUT <= tmp ;-- SUB  when "0010" => ALUOUT <= ABUS and BBUS; -- AND  when "0011" => ALUOUT <= ABUS or BBUS; -- OR  when "0100" => ALUOUT <= ABUS xor BBUS; -- XOR  when "0101" => ALUOUT <= not ABUS; -- NOT  when "0110" => ALUOUT <= ABUS; -- MOVE  when others => ALUOUT <= tmp\_out1;  end case;  end process;    end Behavioral; |

|  |
| --- |
| **ts\_ALU** |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  use IEEE.std\_logic\_unsigned.all;  -- Testbench  ENTITY tb\_ALU IS  END tb\_ALU;    ARCHITECTURE behavior OF tb\_ALU IS    -- Component Declaration for the 16-bit ALU    COMPONENT ALU  PORT(  ABUS : IN std\_logic\_vector(15 downto 0);  BBUS : IN std\_logic\_vector(15 downto 0);  ALUctrl : IN std\_logic\_vector(3 downto 0);  ALUOUT : OUT std\_logic\_vector(15 downto 0)  );  END COMPONENT;    --Inputs  signal ABUS : std\_logic\_vector(15 downto 0) := (others => '0');  signal BBUS : std\_logic\_vector(15 downto 0) := (others => '0');  signal ALUctrl : std\_logic\_vector(3 downto 0) := (others => '0');  --Outputs  signal ALUOUT : std\_logic\_vector(15 downto 0);    BEGIN    -- Instantiate the 16-bit ALU  uut: ALU PORT MAP (  ABUS => ABUS,  BBUS => BBUS,  ALUctrl => ALUctrl,  ALUOUT => ALUOUT  );  stim\_proc: process  begin  ABUS <= x"000A";  BBUS <= x"0002";  ALUctrl <= x"0";  -- change ALU Control input  for i in 0 to 15 loop  ALUctrl <= ALUctrl + x"1";  wait for 100 ns;  end loop;  ABUS <= x"00F6";  BBUS <= x"000A";  wait;  end process;  END; |

**Result:**



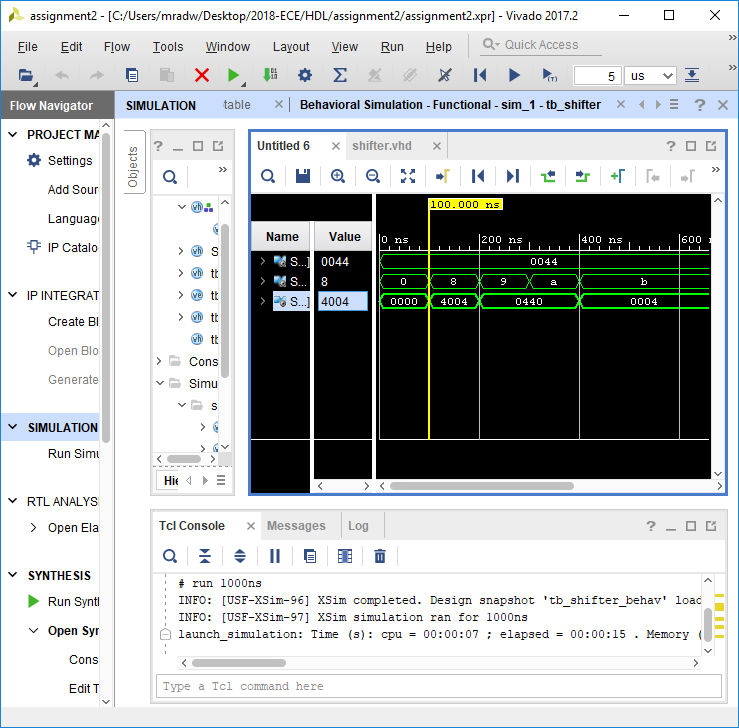
**Task: Implement the Shifting Unit described above in VHDL.**

|  |
| --- |
| case(SHIFT\_Ctrl) is  when "1000" => SHIFTOUT <= SHIFTINPUT(3 downto 0)&SHIFTINPUT(15 downto 4);  when "1001" => SHIFTOUT <= SHIFTINPUT(11 downto 0)&SHIFTINPUT(15 downto 12);  when "1010" => SHIFTOUT <= SHIFTINPUT(11 downto 0) & "0000";  when "1011" => SHIFTOUT <= "0000" & SHIFTINPUT(15 downto 4);  when others => SHIFTOUT <= x"0000";  end case; |

The above code for the shifter is based on the below table.

|  |  |  |
| --- | --- | --- |
| 1000 | ROR4 | Rotate right 4 bits |
| 1001 | ROL4 | Rotate left 4 bits |
| 1010 | SLL4 | Shift left logic 4 bits |
| 1011 | SRL4 | Shift right logic 4 bit |

Below is an image of the stimulation of the code above.



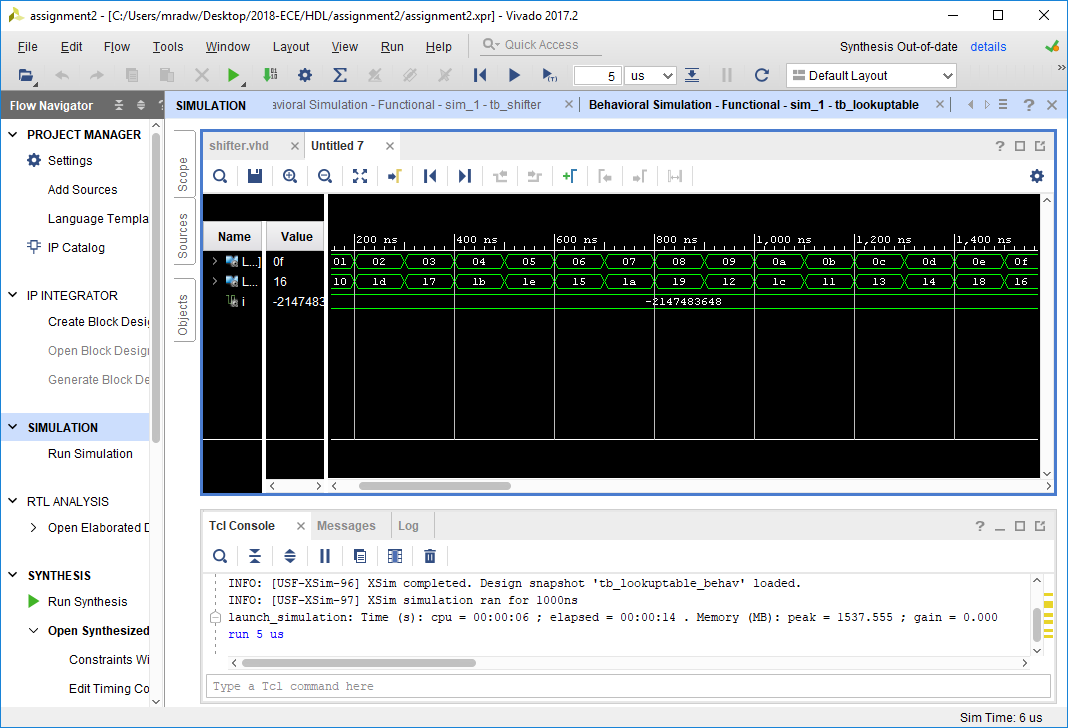
1.4 Non-Linear Lookup Operations in VHDL

To create the non-linear lookup table, the look up table entity is declared which takes in one input and returns a single output.

The table below is the input and output of the lookup table.

|  |  |
| --- | --- |
| INPUT(Hex) | OUTPUT(hex) |
| 00 | 1F |
| 01 | 10 |
| 02 | 1D |
| 03 | 17 |
| 04 | 1B |
| 05 | 1E |
| 06 | 15 |
| 07 | 1A |
| 08 | 19 |
| 09 | 12 |
| 0A | 1C |
| 0B | 11 |
| 0C | 13 |
| 0D | 14 |
| 0E | 18 |
| 0F | 16 |

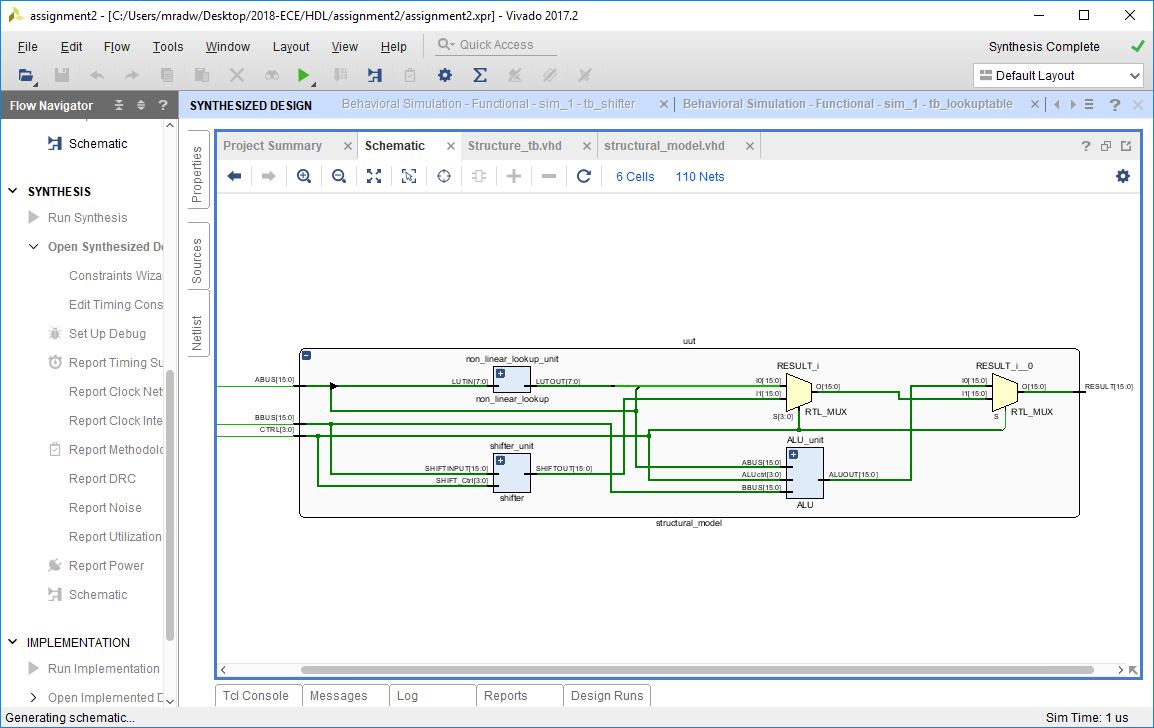
Below is the result of the stimulation.



1.5 Structural VHD model

The structural VHDL code is made up from adding the look up table, ALU and Shifter, the result is then dependent on what the CTRL signal. The non-linear lookup table, shifter and ALU are declared as components of the structural VHDL code.

The design van be seen below.



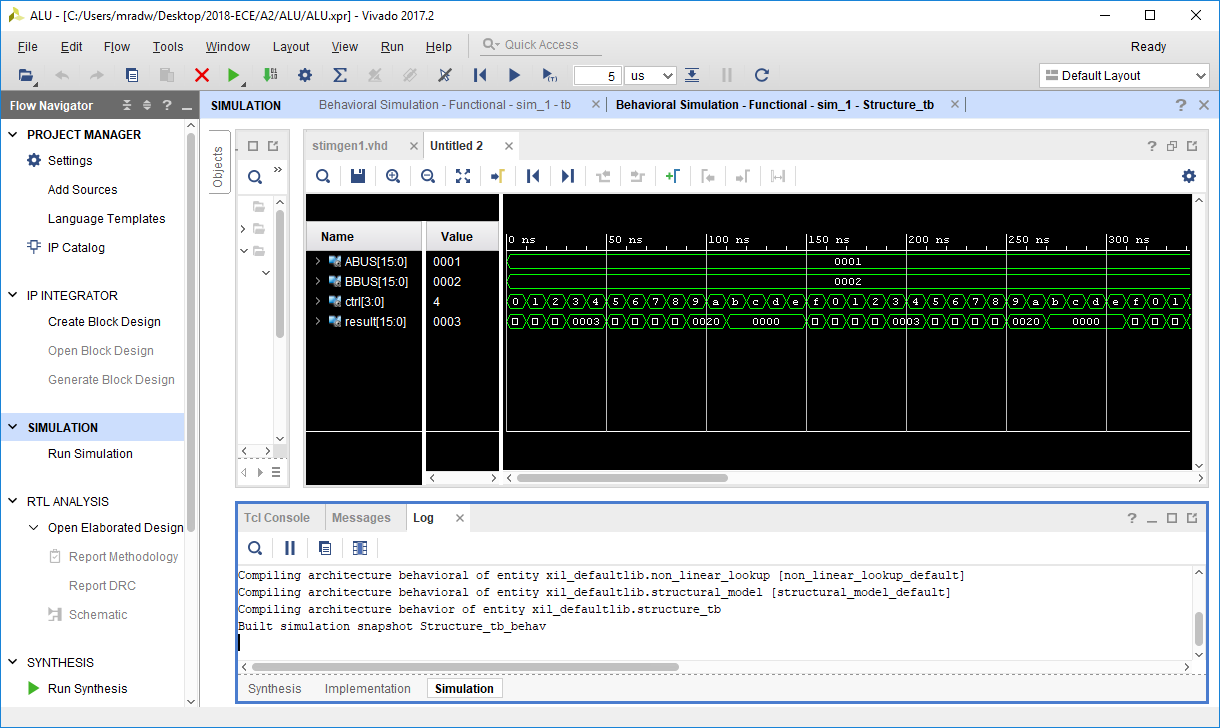
result from the lookup table will be shown and in all other cases, the result will be from the shifter.

|  |  |  |
| --- | --- | --- |
| CTRL | Component | Instruction |
| 0000 | ALU | ADD |
| 0001 | ALU | SUB |
| 0010 | ALU | AND |
| 0011 | ALU | OR |
| 0100 | ALU | XOR |
| 0101 | ALU | NOT A |
| 0110 | ALU | MOVE |
| 0111 | ALU | ADD |
| 1000 | Shifter | ROR |
| 1001 | Shifter | ROL |
| 1010 | Shifter | SLL |
| 1011 | Shifter | SRL |
| 1100 | Shifter | 0 |
| 1101 | Shifter | 0 |
| 1110 | Shifter | 0 |
| 1111 | Lookup | lookup |

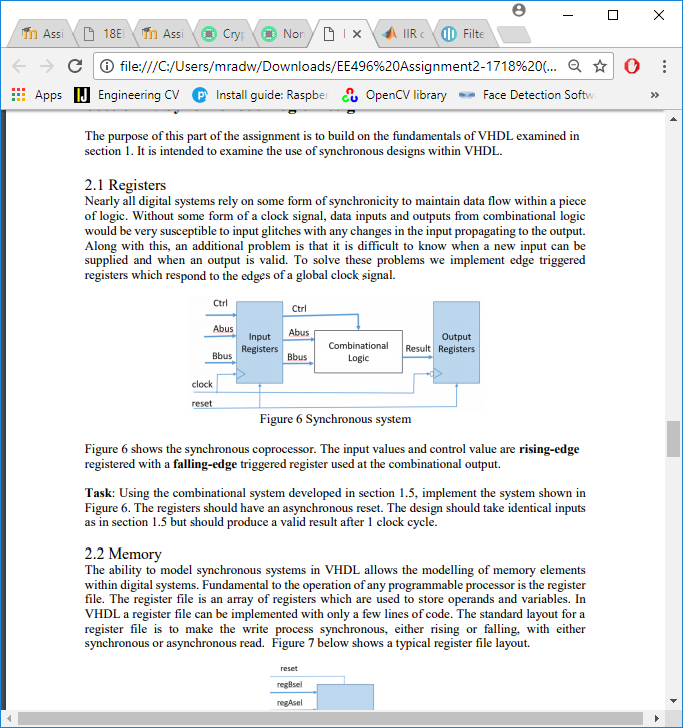
1.6 VHDL Test bench for Structural Design

|  |
| --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use ieee.numeric\_std.all;  use ieee.std\_logic\_unsigned.all;  entity Structure\_tb is  end Structure\_tb;  architecture Behavior of Structure\_tb is  -- component declaration of structure  component Structural\_model  port(  ABUS, BBUS: in std\_logic\_vector(15 downto 0);  ctrl: in std\_logic\_vector(3 downto 0);  result: out std\_logic\_vector(15 downto 0)  );  end component;  -- inputs  signal ABUS : std\_logic\_vector(15 downto 0) := (others => '0');  signal BBUS : std\_logic\_vector(15 downto 0) := (others => '0');  signal ctrl : std\_logic\_vector(3 downto 0) := (others => '0');  -- output  signal result : std\_logic\_vector(15 downto 0);  begin  -- initiate structural model  uut: structural\_model port map(  ABUS => ABUS,  BBUS => BBUS,  ctrl => ctrl,  result => result  );  stim\_proc: process  begin  ABUS <= x"0001";  BBUS <= x"0002";    ctrl <= x"0";    wait for 10ns;  if result = x"0003" then  report"PASS: 1";  else  report"FAIL: 1";  end if;  ctrl <= x"1";  wait for 10ns;  if result = x"FFFF" then  report"PASS: 2";  else  report"FAIL: 2";  end if;    ctrl <= x"2";  wait for 10ns;  if result = x"0000" then  report"PASS: 3";  else  report"FAIL: 3";  end if;      ctrl <= x"3";  wait for 10ns;  if result = x"0003" then  report"PASS: 4";  else  report"FAIL: 4";  end if;      ctrl <= x"4";  wait for 10ns;  if result = x"0003" then  report"PASS: 5";  else  report"FAIL: 5";  end if;    ctrl <= x"5";  wait for 10ns;  if result = x"FFFE" then  report"PASS: 6";  else  report"FAIL: 6";  end if;    ctrl <= x"6";  wait for 10ns;  if result = x"0001" then  report"PASS: 7";  else  report"FAIL: 7";  end if;    ctrl <= x"7";  wait for 10ns;  if result = x"0003" then  report"PASS: 8";  else  report"FAIL: 8";  end if;    ctrl <= x"8";  wait for 10ns;  if result = x"2000" then  report"PASS: 9";  else  report"FAIL: 9";  end if;    ctrl <= x"9";  wait for 10ns;  if result = x"0020" then  report"PASS: 10";  else  report"FAIL: 10";  end if;    ctrl <= x"a";  wait for 10ns;  if result = x"0020" then  report "PASS: 11";  else  report "FAIL: 11";  end if;    ctrl <= x"b";  wait for 10ns;  if result = x"0000" then  report "PASS: 12";  else  report "FAIL: 12";  end if;    ctrl <= x"c";  wait for 10ns;  if result = x"0" then  report "PASS: 13";  else  report "FAIL: 13";  end if;    ctrl <= x"d";  wait for 10ns;  if result = x"0" then  report "PASS: 14";  else  report "FAIL: 14";  end if;    ctrl <= x"e";  wait for 10ns;  if result = x"0" then  report "PASS: 15";  else  report "FAIL: 15";  end if;    ctrl <= x"f";  wait for 10ns;  if result = x"10" then  report "PASS: 16";  else  report "FAIL: 16";  end if;    end process;  end; |

Above is the test bench that was used to test all 12 possible modes of operation as can be seen when the A and B were set to 1, and 2 respectively the result will be as follows.



2.1 Registers



From the image above to create a synchronous system there needs to be an input and output registered the image above also includes what inputs are needed.

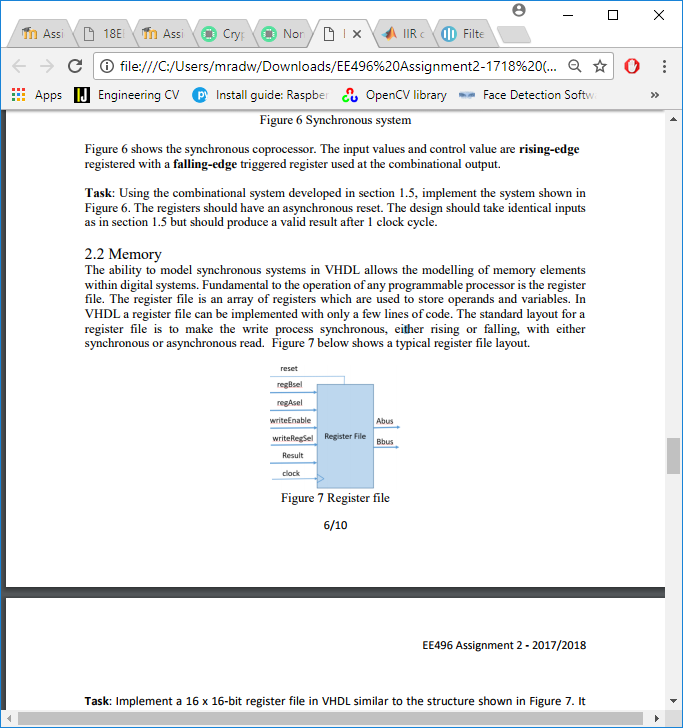
If the clock edge is rising and reset is disabled, the combinational logic system reads in the input values as seen below.

|  |
| --- |
| process(clock,reset)  begin  if rising\_edge(clock) then  if reset ='1' then  ctrl <= "0000";  ABUS <= x"0000";  BBUS <= x"0000";  else  ctrl <= ctrlin;  ABUS <= ABUSin;  BBUS <= BBUSin;  end if;  end if;  end process; |

If the clock edge is falling and reset is disabled, then the combinational logic system is writing out the result.

|  |
| --- |
| process(clock,reset)  begin  if falling\_edge(clock) then  if reset = '1' then  result <= x"0000";  else  result <= resultin;  end if;  end if;  end process; |

2.2 Memory



|  |
| --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  entity register\_file is  Port (  Abus : out std\_logic\_vector(15 downto 0); --srca  Bbus : out std\_logic\_vector(15 downto 0); --srcb  result : in std\_logic\_vector(15 downto 0); --res  writeEnable : in std\_logic; --rdwen  regAsel : in std\_logic\_vector(3 downto 0); --ra  regBsel : in std\_logic\_vector(3 downto 0); --rb  writeRegSel : in std\_logic\_vector(3 downto 0); --rd  reset : in std\_logic;  clock : in std\_logic  );  end register\_file;  architecture Behavioral of register\_file is  type memory is array(0 to 15) of std\_logic\_vector(15 downto 0);  signal REG\_FILE: memory :=(  0 => x"0001",  1 => x"c505",  2 => x"3c07",  3 => x"d405",  4 => x"1186",  5 => x"f407",  6 => x"1086",  7 => x"4706",  8 => x"6808",  9 => x"baa0",  10 => x"c902",  11 => x"100b",  12 => x"c000",  13 => x"c902",  14 => x"100b",  15 => x"B000",  others => (others => '0')  );  begin  write\_op: process(clock)  begin  if(falling\_edge(clock)) then  if(writeEnable='1') then  REG\_FILE(to\_integer(unsigned(writeRegSel))) <= result;  end if;  end if;  end process;  read\_op: process(clock)  begin  if(rising\_edge(clock)) then  if(reset='1') then  ABUS <= x"0000";  BBUS <= x"0000";  else  ABUS <= REG\_FILE(to\_integer(unsigned(regAsel)));  BBUS <= REG\_FILE(to\_integer(unsigned(regBsel)));  end if;  end if;  end process;  end Behavioral; |

2.3 Complete Crypto Coprocessor

|  |
| --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  entity coprocessor is  Port (  clock: in std\_logic;  reset: in std\_logic;  ctrl: in std\_logic\_vector (3 downto 0);  regA : in std\_logic\_vector(3 downto 0); --ra  regB : in std\_logic\_vector(3 downto 0); --rb  writeReg : in std\_logic\_vector(3 downto 0)  );  end coprocessor;  architecture Behavioral of coprocessor is  --register file  component register\_file is  port(  Abus : out std\_logic\_vector(15 downto 0); --srca  Bbus : out std\_logic\_vector(15 downto 0); --srcb  result : in std\_logic\_vector(15 downto 0); --res  writeEnable : in std\_logic; --rdwrite\_enable  regAsel : in std\_logic\_vector(3 downto 0); --ra  regBsel : in std\_logic\_vector(3 downto 0); --rb  writeRegSel : in std\_logic\_vector(3 downto 0); --rd  reset : in std\_logic;  clock : in std\_logic  );  end component register\_file;  --structural component  component structural\_model is  port(  ABUS: in std\_logic\_vector(15 downto 0);  BBUS: in std\_logic\_vector(15 downto 0);  CTRL: in std\_logic\_vector(3 downto 0);  RESULT: out std\_logic\_vector(15 downto 0)  );  end component structural\_model;  -- internal signals  signal write\_enable: std\_logic;  signal read1, read2, write : std\_logic\_vector(15 downto 0);  signal tmp1, tmp2: std\_logic\_vector(3 downto 0);  begin  uut1: register\_file port map(  clock=> clock,  reset => reset,  writeEnable => write\_enable,  result => write,  regAsel => regA,  regBsel => regB,  writeRegSel => tmp2,  Abus => read1,  Bbus => read2  );  uut2: Structural\_model port map(  ABUS => read1,  BBUS => read2,  ctrl => tmp1,  result => write  );  process(clock,reset) begin  if(rising\_edge(clock)) then  if (reset ='1') then  tmp1 <= x"0";  tmp2 <= x"0";  else  tmp1 <= ctrl;  tmp2 <= writeReg;  if (tmp1 = "0111") then  write\_enable <= '0';  else  write\_enable <= '1';  end if;  end if;    end if;  end process;    end Behavioral; |

2.4 Test Program

A stimulus generator is created that implements the code specified in the task after performing a system reset.

