Computer Architectures
Lab Report 1

PARAMETERIZABLE ALU & PARAMETERIZABLE REGISTER BANK

Y3839090 & Y3840426| Computer Architectures | ELE00009I

Session 1

ALUParam.vhd

```
----------------------------------------------------------------------------------
--
-- Uni             :     University of York
-- Course          :     Electronic Engineering
-- Module          :     Computer Architectures
-- Engineers       :     Y3839090 & Y3840426
--
-- Create Date     :     13:19:20 02/17/2017
-- Design Name     :     ALU_param - Behavioral
-- Description     :     A paramateriable integer ALU.
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity ALU_param is
      Generic (
            N : natural := 8 -- data size in bits
      );
      Port (
            A : in  STD_LOGIC_VECTOR (N-1 downto 0);
            B : in  STD_LOGIC_VECTOR (N-1 downto 0);
            X : in  STD_LOGIC_VECTOR (log2(N)-1 downto 0); -- shift/rotate amount input
            ctrl : in  STD_LOGIC_VECTOR (3 downto 0); -- control signals from opcode
            O : out  STD_LOGIC_VECTOR (N-1 downto 0);
            flags :out  STD_LOGIC_VECTOR (7 downto 0) -- flags
      );
end ALU_param;

architecture Behavioral of ALU_param is

      -- internal signed signal for A and B inputs
      signal A_itrn : SIGNED (N-1 downto 0);
      signal B_itrn : SIGNED (N-1 downto 0);

      -- internal integer for X
      signal X_itrn : integer;

      -- internal signed signal for output
      signal O_itrn : SIGNED (N-1 downto 0);


      -- max positive and negative N bit signed numbers
      constant max_pos : SIGNED (N-1 downto 0) := to_signed(( 2 ** (N-1) ) - 1, N);
      constant max_neg : SIGNED (N-1 downto 0) := to_signed( -2 ** (N-1) , N);

begin
```

(Continued.)

```
A_itrn <= signed(A); -- converts A to signed and maps the result to A_itrn
B_itrn <= signed(B); -- converts A to signed and maps the result to B_itrn

-- converts X to integer and maps the result to X_itrn
X_itrn <= to_integer(unsigned(X));

-- converts O_itrn to a plain std_logic_vector and maps it to O
O <= std_logic_vector(O_itrn);

-- Main ALU multiplexer for each possible command
O_itrn <=
      A_itrn                  when ctrl = "0000" else       -- Output A
      A_itrn and B_itrn       when ctrl = "0100" else       -- Output A & B
      A_itrn or B_itrn        when ctrl = "0101" else       -- Output A || B
      A_itrn xor B_itrn       when ctrl = "0110" else       -- Output A xor B
      not A_itrn              when ctrl = "0111" else       -- Output not A
      A_itrn + 1              when ctrl = "1000" else       -- Output A + 1
      A_itrn - 1              when ctrl = "1001" else       -- Output A - 1
      A_itrn + B_itrn         when ctrl = "1010" else       -- Output A + B
      A_itrn - B_itrn         when ctrl = "1011" else       -- Output A - B
      SHIFT_LEFT (A_itrn , X_itrn)        when ctrl = "1100" else -- Output A sla X
      SHIFT_RIGHT (A_itrn , X_itrn)       when ctrl = "1101" else -- Output A sra X
      ROTATE_LEFT (A_itrn , X_itrn)       when ctrl = "1110" else -- Output A rotl X
      ROTATE_RIGHT (A_itrn , X_itrn)      when ctrl = "1111" else -- Output A rotr X
      (others =>'U');

-- Overflow flag
flags(7) <=
      -- Will overflow if you add one to the max positive value
      '1' when ctrl = "1000" and A_itrn = max_pos else

      -- Will overflow if you minus one to the max negitive value
      '1' when ctrl = "1001" and A_itrn = max_neg else

      -- Will overflow if two neg values added give a pos result
      '1' when ctrl = "1010" and A_itrn(N-1) = '1' and  B_itrn(N-1) = '1' and O_itrn(N-1) = '0'
      else

      -- Will overflow if two pos values added give a neg result
      '1' when ctrl = "1010" and A_itrn(N-1) = '0' and  B_itrn(N-1) = '0' and O_itrn(N-1) = '1'
      else

      -- Will overflow if a pos value is subtracted from a neg value gives a pos result
      '1' when ctrl = "1011" and A_itrn(N-1) = '1' and  B_itrn(N-1) = '0' and O_itrn(N-1) = '0'
      else

      -- Will overflow if a neg value is subtracted from a pos value gives a neg result
      '1' when ctrl = "1011" and A_itrn(N-1) = '0' and  B_itrn(N-1) = '1' and O_itrn(N-1) = '1'

      -- If none of the above are true then the result hasn't overflown
      else '0';
```

(Continued.)

```
      flags(6) <= '1'   when O_itrn >= 0  else '0'; -- greater than or equal to zero
      flags(5) <= '1'   when O_itrn <= 0  else '0'; -- less than or equal to zero
      flags(4) <= '1'   when O_itrn >  0  else '0'; -- greater than zero
      flags(3) <= '1'   when O_itrn <  0  else '0'; -- less than zero
      flags(2) <= '1'   when O_itrn =  1  else '0'; -- one flag
      flags(1) <= '1'   when O_itrn /= 0  else '0'; -- not zero flag
      flags(0) <= '1'   when O_itrn =  0  else '0'; -- zero flag


end Behavioral;
```

ALUParamTB.vhd

```
--------------------------------------------------------------------------------
--
-- Uni             :     University of York
-- Course          :     Electronic Engineering
-- Module          :     Computer Architectures
-- Engineers       :     Y3839090 & Y3840426
--
-- Create Date     :     14:47:27 02/17/2017
-- Design Name     :     ALU_param_TB - TestBench
-- Description     :     A testbench for a 16 bit integer ALU.
--
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.DigEng.ALL;

entity ALU_param_TB is
end ALU_param_TB;

architecture behavior of ALU_param_TB is

      -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====
      -- From http://stackoverflow.com/a/24336034 By Morten Zilmer
      -- Allows printing a std_logic_vector as a string that represents it's binary form.
      -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====
      function to_bstring(sl : std_logic) return string is
            variable sl_str_v : string(1 to 3);  -- std_logic image with quotes around
      begin
            sl_str_v := std_logic'image(sl);
            return "" & sl_str_v(2);  -- "" & character to get string
      end function;
```

(Continued.)

```
    function to_bstring(slv : std_logic_vector) return string is
          alias    slv_norm : std_logic_vector(1 to slv'length) is slv;
          variable sl_str_v : string(1 to 1);  -- String of std_logic
          variable res_v    : string(1 to slv'length);
    begin
          for idx in slv_norm'range loop
                sl_str_v := to_bstring(slv_norm(idx));
                res_v(idx) := sl_str_v(1);
          end loop;
          return res_v;
    end function;
    -- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====


    -- converts an std_logic_vector to a string that represents it's signed value
    function s_tostr(val : std_logic_vector) return string is
    begin
          return integer'image( to_integer(signed(val)) );
    end function;


    -- Constants
    constant M : NATURAL := 16; -- We are testing a 16 bit ALU
    constant wait_time : TIME := 10 ns;

    -- Component Declaration for the Unit Under Test (UUT)
    component ALU_param
          generic
          (
                N : NATURAL -- The number of bits this alu will operate on
          );
          port
          (
                A : IN  STD_LOGIC_VECTOR(M-1 downto 0);
                B : IN  STD_LOGIC_VECTOR(M-1 downto 0);
                X : IN  STD_LOGIC_VECTOR(log2(M)-1 downto 0);
                ctrl : IN  STD_LOGIC_VECTOR(3 downto 0);
                O : OUT  STD_LOGIC_VECTOR(M-1 downto 0);
                flags : OUT  STD_LOGIC_VECTOR(7 downto 0)
          );
    end component;

--Inputs
signal A : STD_LOGIC_VECTOR(M-1 downto 0) := (others => '0');
signal B : STD_LOGIC_VECTOR(M-1 downto 0) := (others => '0');
signal X : STD_LOGIC_VECTOR(log2(M)-1 downto 0) := (others => '0');
signal ctrl : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');

--Outputs
signal O : STD_LOGIC_VECTOR(M-1 downto 0);
signal flags : STD_LOGIC_VECTOR(7 downto 0);
```

(Continued.)

```
type TEST_VECTOR_ARRAY is ARRAY(NATURAL range <>) of TEST_VECTOR;

constant test_vectors : TEST_VECTOR_ARRAY := (
      -- CTRL, A ,      B ,      X ,      O ,      FLAGS

      -- A
      ("0000", X"0000", X"0000", X"0", X"0000", "01100001"),
      ("0000", X"FF9C", X"0000", X"0", X"FF9C", "00101010"),
      ("0000", X"03E8", X"0000", X"0", X"03E8", "01010010"),
      ("0000", X"0001", X"0000", X"0", X"0001", "01010110"),

      -- A and B
      ("0100", X"0000", X"0000", X"0", X"0000", "01100001"),
      ("0100", X"FF94", X"FC17", X"0", X"FC14", "00101010"),
      ("0100", X"03E8", X"5213", X"0", X"0200", "01010010"),
      ("0100", X"FFFF", X"10E0", X"0", X"10E0", "01010010"),
      ("0100", X"55FF", X"AAAA", X"0", X"00AA", "01010010"),

      -- A or B
      ("0101", X"0000", X"0000", X"0", X"0000", "01100001"),
      ("0101", X"FF94", X"FC17", X"0", X"FF97", "00101010"),
      ("0101", X"03E8", X"5213", X"0", X"53FB", "01010010"),
      ("0101", X"FFFF", X"10E0", X"0", X"FFFF", "00101010"),
      ("0101", X"5555", X"AAAA", X"0", X"FFFF", "00101010"),

      -- A xor B
      ("0110", X"0000", X"0000", X"0", X"0000", "01100001"),
      ("0110", X"FF94", X"FC17", X"0", X"0383", "01010010"),
      ("0110", X"03E8", X"5213", X"0", X"51FB", "01010010"),
      ("0110", X"FFFF", X"10E0", X"0", X"EF1F", "00101010"),
      ("0110", X"5555", X"AAAA", X"0", X"FFFF", "00101010"),

      -- not A
      ("0111", X"0000", X"0000", X"0", X"FFFF", "00101010"),
      ("0111", X"FFFF", X"0000", X"0", X"0000", "01100001"),
      ("0111", X"8111", X"0000", X"0", X"7EEE", "01010010"),
      ("0111", X"0001", X"0000", X"0", X"FFFE", "00101010"),

      -- A + 1
      ("1000", X"0000", X"0000", X"0", X"0001", "01010110"),
      ("1000", X"FFFF", X"0000", X"0", X"0000", "01100001"),
      ("1000", X"7fff", X"0000", X"0", X"8000", "10101010"),

      -- A - 1
      ("1001", X"0000", X"0000", X"0", X"ffff", "00101010"),
      ("1001", X"0001", X"0000", X"0", X"0000", "01100001"),
      ("1001", X"8000", X"0000", X"0", X"7fff", "11010010"),

      -- A + B
      ("1010", X"0000", X"0000", X"0", X"0000", "01100001"),
      ("1010", X"0310", X"0a00", X"0", X"0D10", "01010010"),
      ("1010", X"09E2", X"f43e", X"0", X"FE20", "00101010"),
      ("1010", X"7fff", X"0001", X"0", X"8000", "10101010"),
```

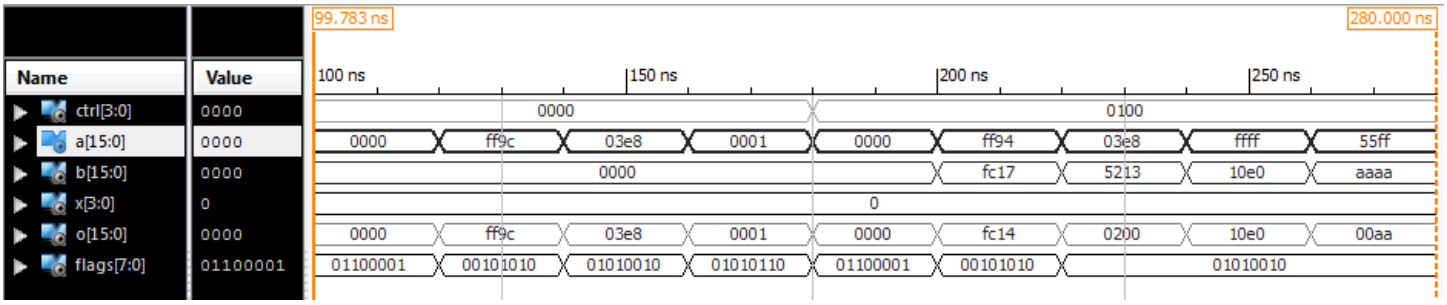
(Continued.)

```
            -- A - B
            ("1011", X"0000", X"0000", X"0", X"0000", "01100001"),
            ("1011", X"0310", X"0a00", X"0", X"F910", "00101010"),
            ("1011", X"09E2", X"0BC2", X"0", X"FE20", "00101010"),
            ("1011", X"8000", X"0001", X"0", X"7fff", "11010010"),


            -- A sla x
            ("1100", X"0000", X"0000", X"0", X"0000", "01100001"),
            ("1100", X"0000", X"0000", X"4", X"0000", "01100001"),
            ("1100", X"1111", X"0000", X"1", X"2222", "01010010"),
            ("1100", X"1111", X"0000", X"3", X"8888", "00101010"),
            ("1100", X"5555", X"0000", X"9", X"AA00", "00101010"),

            -- A sra x
            ("1101", X"0000", X"0000", X"0", X"0000", "01100001"),
            ("1101", X"0000", X"0000", X"4", X"0000", "01100001"),
            ("1101", X"8888", X"0000", X"1", X"C444", "00101010"),
            ("1101", X"8888", X"0000", X"3", X"f111", "00101010"),
            ("1101", X"AAAA", X"0000", X"9", X"FFD5", "00101010"),

            -- A rotl x
            ("1110", X"0000", X"0000", X"2", X"0000", "01100001"),
            ("1110", X"8888", X"0000", X"1", X"1111", "01010010"),
            ("1110", X"8101", X"0000", X"1", X"0203", "01010010"),

            -- A rotr x
            ("1111", X"0000", X"0000", X"2", X"0000", "01100001"),
            ("1111", X"1111", X"0000", X"1", X"8888", "00101010"),
            ("1111", X"1081", X"0000", X"1", X"8840", "00101010"),

            -- THIS TEST WILL FAIL
            ("1010", X"0000", X"0000", X"0", X"FFFF", "10000001")

      );
begin

      -- Instantiate the Unit Under Test (UUT)
      uut: ALU_param
      generic map
      (
            N => M
      )
      port map
      (
            A => A,
            B => B,
            X => X,
            ctrl => ctrl,
            O => O,
            flags => flags
      );
```

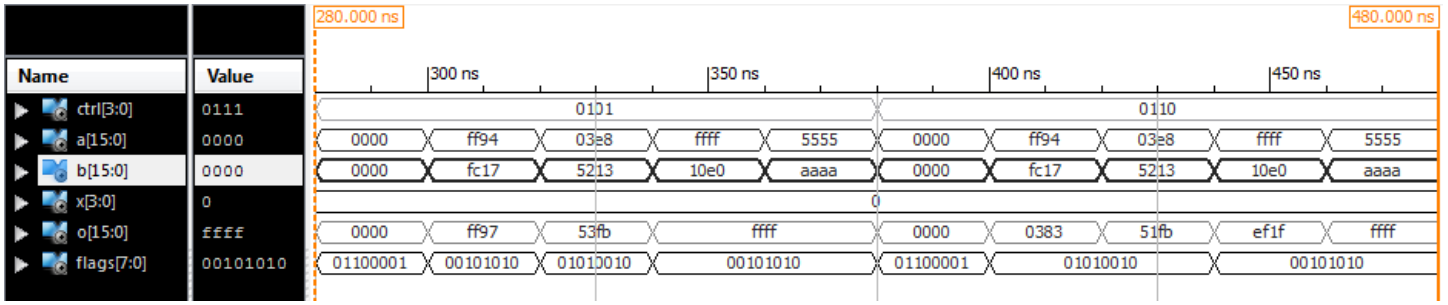
(Continued.)

```
-- Stimulus process
      stim_proc: process
      begin
            -- hold reset state for 100 ns.
            wait for 100 ns;

            -- run the test for every set of data
            for i in test_vectors'range loop

                  -- assign test inputs
                  ctrl <= test_vectors(i).ctrl;
                  A <= test_vectors(i).A;
                  B <= test_vectors(i).B;
                  X <= test_vectors(i).X;

                  -- wait long enough for the ALU to process
                  wait for wait_time;

                  -- check that the actual output is the same as the expect output
                  assert O = test_vectors(i).O
                  report " [ERR!] Test " & integer'image(i)&
                        " Actual output did not equal expected output: Actual " & s_tostr(O) &
                        " , Expected " & s_tostr(test_vectors(i).O)
                  severity error;

                  -- check that the actual flags is the same as the expect flags
                  assert flags = test_vectors(i).flags
                  report " [ERR!] Test " & integer'image(i)&
                        " Actual flags did not equal expected flags : Actual " &
                        to_bstring(flags) &
                        " , Expected " & to_bstring(test_vectors(i).flags)
                  severity error;

                  -- if there were no issues report that the test was successful
                  assert not ( O = test_vectors(i).O and  flags = test_vectors(i).flags )
                  report " [ OK ] Test " & integer'image(i)& " was successful!"
                  severity note;

                  wait for wait_time;

            end loop;

      wait;
      end process;

end;
```

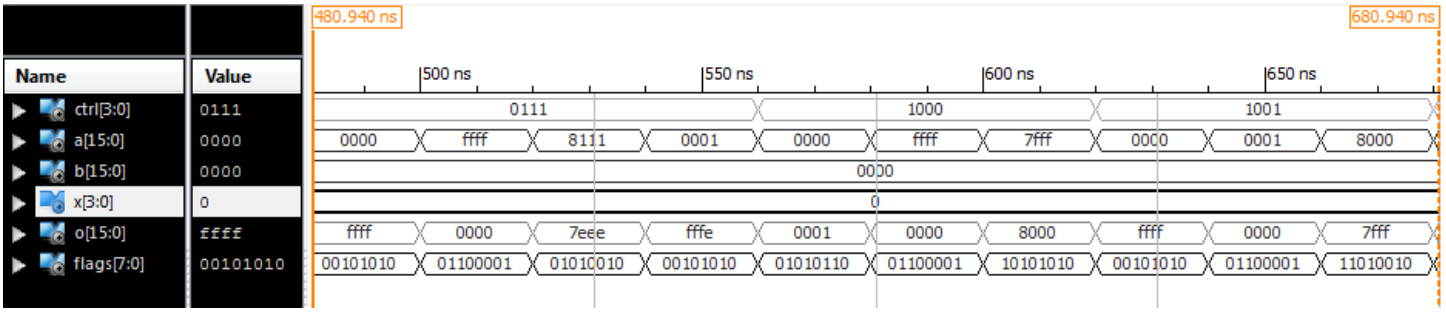
Simulations



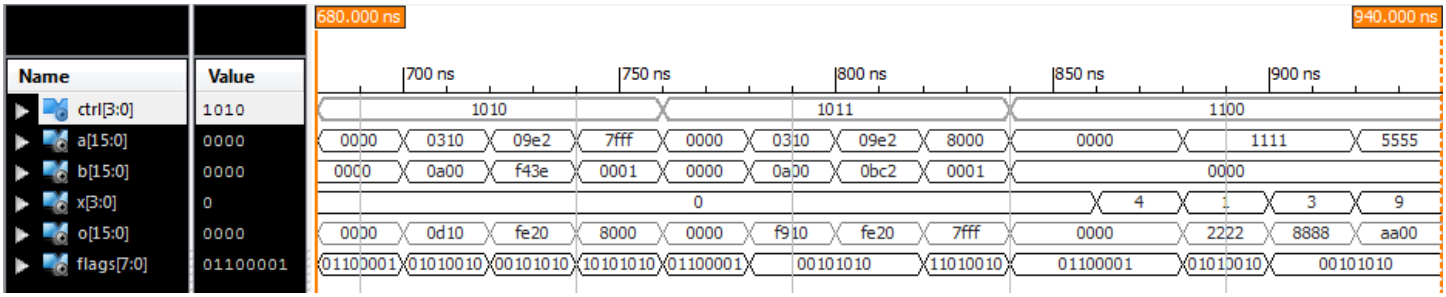
Shows the operation of giving A as the output (OP 0000), and giving the result of A AND B as the output (OP 0100)



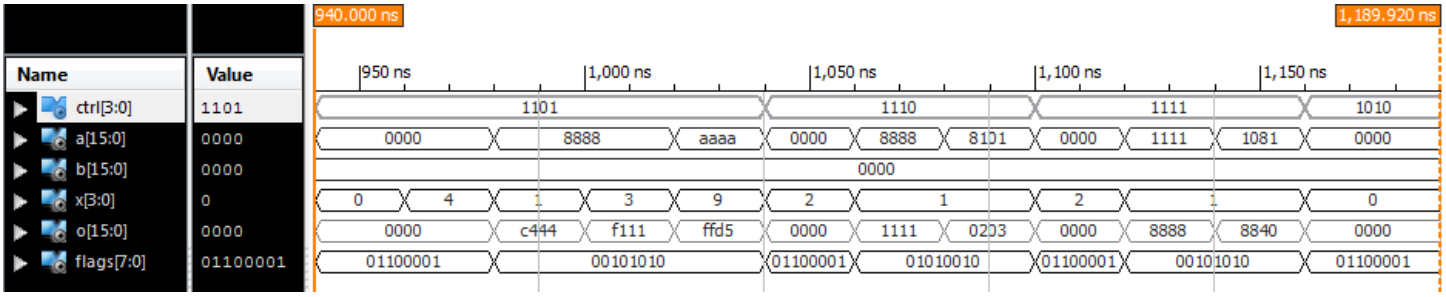
Shows the operation of giving the result of A OR B as the output (OP 0101), and giving the result of A XOR B as the output (OP 0110)



Shows the operation of giving the result of NOT A as the output (OP 0111), giving the result of A + 1 as the output (OP 1000) and giving the result of A-1 (OP 1001)



Shows the operation of giving the result of A+B as the output (OP 1010), giving the result of A - B as the output (OP 1011) and giving the result of an arithmetic shift of input A by X bits left as the output (OP 1100)



Shows the operation of giving the result of an arithmetic shift of input A by X bits right as the output (OP 1101), giving the result of a left rotation of A by X bits as the output (OP 1110) and giving the result of a right shift of A by X bits as the output (OP 1111). It also shows an A+B operation (OP 1010) used for the purpose of showing an erroneous set of inputs/outputs to verify the correct operation of assert clause. The error is seen below in the ISim console output.

ISim Console Output

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.
Time resolution is 1 ps
WARNING: Simulation object /alu_param_tb/test_vectors was not traceable in the design for the following reason:
ISim does not yet support tracing of constant and generic multi-dimensional arrays.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 110 ns: Note:  [ OK ] Test 0 was successful! (/alu_param_tb/).
at 130 ns: Note:  [ OK ] Test 1 was successful! (/alu_param_tb/).
at 150 ns: Note:  [ OK ] Test 2 was successful! (/alu param tb/).
at 170 ns: Note:  [ OK ] Test 3 was successful! (/alu_param_tb/).
at 190 ns: Note:  [ OK ] Test 4 was successful! (/alu_param_tb/).
at 210 ns: Note:  [ OK ] Test 5 was successful! (/alu_param_tb/).
at 230 ns: Note:  [ OK ] Test 6 was successful! (/alu_param_tb/).
at 250 ns: Note:  [ OK ] Test 7 was successful! (/alu_param_tb/).
at 270 ns: Note:  [ OK ] Test 8 was successful! (/alu param tb/).
at 290 ns: Note:  [ OK ] Test 9 was successful! (/alu_param_tb/).
at 310 ns: Note:  [ OK ] Test 10 was successful! (/alu_param_tb/).
at 330 ns: Note:  [ OK ] Test 11 was successful! (/alu_param_tb/).
at 350 ns: Note:  [ OK ] Test 12 was successful! (/alu_param_tb/).
at 370 ns: Note:  [ OK ] Test 13 was successful! (/alu_param_tb/).
at 390 ns: Note:  [ OK ] Test 14 was successful! (/alu param tb/).
at 410 ns: Note:  [ OK ] Test 15 was successful! (/alu_param_tb/).
at 430 ns: Note:  [ OK ] Test 16 was successful! (/alu_param_tb/).
at 450 ns: Note:  [ OK ] Test 17 was successful! (/alu_param_tb/).
at 470 ns: Note:  [ OK ] Test 18 was successful! (/alu_param_tb/).
at 490 ns: Note:  [ OK ] Test 19 was successful! (/alu_param_tb/).
at 510 ns: Note:  [ OK ] Test 20 was successful! (/alu param tb/).
at 530 ns: Note:  [ OK ] Test 21 was successful! (/alu_param_tb/).
at 550 ns: Note:  [ OK ] Test 22 was successful! (/alu_param_tb/).
at 570 ns: Note:  [ OK ] Test 23 was successful! (/alu_param_tb/).
at 590 ns: Note:  [ OK ] Test 24 was successful! (/alu_param_tb/).
at 610 ns: Note:  [ OK ] Test 25 was successful! (/alu_param_tb/).
at 630 ns: Note:  [ OK ] Test 26 was successful! (/alu_param_tb/).
at 650 ns: Note:  [ OK ] Test 27 was successful! (/alu param tb/).
at 670 ns: Note:  [ OK ] Test 28 was successful! (/alu_param_tb/).
at 690 ns: Note:  [ OK ] Test 29 was successful! (/alu_param_tb/).
at 710 ns: Note:  [ OK ] Test 30 was successful! (/alu_param_tb/).
at 730 ns: Note:  [ OK ] Test 31 was successful! (/alu param tb/).
at 750 ns: Note:  [ OK ] Test 32 was successful! (/alu_param_tb/).
at 770 ns: Note:  [ OK ] Test 33 was successful! (/alu param tb/).
at 790 ns: Note:  [ OK ] Test 34 was successful! (/alu_param_tb/).
at 810 ns: Note:  [ OK ] Test 35 was successful! (/alu_param_tb/).
at 830 ns: Note:  [ OK ] Test 36 was successful! (/alu_param_tb/).
at 850 ns: Note:  [ OK ] Test 37 was successful! (/alu param tb/).
at 870 ns: Note:  [ OK ] Test 38 was successful! (/alu_param_tb/).
at 890 ns: Note:  [ OK ] Test 39 was successful! (/alu param tb/).
at 910 ns: Note:  [ OK ] Test 40 was successful! (/alu_param_tb/).
at 930 ns: Note:  [ OK ] Test 41 was successful! (/alu_param_tb/).
at 950 ns: Note:  [ OK ] Test 42 was successful! (/alu_param_tb/).
at 970 ns: Note:  [ OK ] Test 43 was successful! (/alu param tb/).
at 990 ns: Note:  [ OK ] Test 44 was successful! (/alu_param_tb/).
at 1010 ns: Note:  [ OK ] Test 45 was successful! (/alu param tb/).
at 1030 ns: Note:  [ OK ] Test 46 was successful! (/alu_param_tb/).
at 1050 ns: Note:  [ OK ] Test 47 was successful! (/alu_param_tb/).
at 1070 ns: Note:  [ OK ] Test 48 was successful! (/alu_param_tb/).
at 1090 ns: Note:  [ OK ] Test 49 was successful! (/alu_param_tb/).
at 1110 ns: Note:  [ OK ] Test 50 was successful! (/alu param tb/).
at 1130 ns: Note:  [ OK ] Test 51 was successful! (/alu_param_tb/).
at 1150 ns: Note:  [ OK ] Test 52 was successful! (/alu_param_tb/).
at 1170 ns: Error:  [ERR!] Test 53 Actual output did not equal expected output: Actual 0 , Expected -1
at 1170 ns: Error:  [ERR!] Test 53 Actual flags did not equal expected flags : Actual 01100001 , Expected 10000001

ISim>
```

HDL Synthesis Report

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Synthesizing Unit <ALU_param>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_1\ParameterizableALU\ALU_param.vhd".
        N = 8
    Found 8-bit adder for signal <A itrn[7] B itrn[7] add 27 OUT> created at line 69.
    Found 8-bit adder for signal <A_itrn[7]_GND_5_o_add_31_OUT> created at line 1253.
    Found 8-bit subtractor for signal <A_itrn[7]_B_itrn[7]_sub_26_OUT<7:0>> created at line 70.
    Found 8-bit subtractor for signal <A_itrn[7]_GND_5_o_sub_30_OUT<7:0>> created at line 1320.
    Found 8-bit shifter rotate right for signal <A_itrn[7]_X_itrn[30]_rotate_right_17_OUT> created at line 3021
    Found 8-bit shifter rotate left for signal <A_itrn[7]_X_itrn[30]_rotate_left_19_OUT> created at line 3012
    Found 8-bit shifter arithmetic right for signal <A itrn[7] X itrn[30] shift right 21 OUT> created at line 2982
    Found 8-bit shifter logical left for signal <A_itrn[7]_X_itrn[30]_shift_left_23_OUT> created at line 2973
    Found 8-bit 13-to-1 multiplexer for signal <O> created at line 27.
    Found 8-bit comparator greater for signal <flags<3>> created at line 99
    Found 8-bit comparator greater for signal <flags<4>> created at line 100
    Summary:
        inferred   1 Adder/Subtractor(s).
        inferred   2 Comparator(s).
        inferred  16 Multiplexer(s).
        inferred   4 Combinational logic shifter(s).
Unit <ALU_param> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1
 8-bit addsub                                         : 1
# Comparators                                          : 2
 8-bit comparator greater                             : 2
# Multiplexers                                         : 16
 1-bit 2-to-1 multiplexer                             : 6
 8-bit 2-to-1 multiplexer                             : 10
# Logic shifters                                       : 4
 8-bit shifter arithmetic right                       : 1
 8-bit shifter logical left                           : 1
 8-bit shifter rotate left                            : 1
 8-bit shifter rotate right                           : 1
# Xors                                                 : 1
 8-bit xor2                                           : 1


=========================================================================
```

Session 2

ParamRegister.vhd

```
----------------------------------------------------------------------------------
--
-- Uni            :     University of York
-- Course         :     Electronic Engineering
-- Module         :     Computer Architectures
-- Engineers      :     Y3839090 & Y3840426
--
-- Create Date    :     20:11:17 02/23/2017
-- Design Name    :     Param_Registers - Behavioral
-- Description    :     A parameteriable dual read single write register array.
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.DigEng.ALL;

entity Pram_Registers is
      Generic
      (
            num_reg : natural := 8; -- The number of registers to create
            data_size : natural := 16 -- The data size of these registers in bits
      );
      Port (
            addr_A : in  STD_LOGIC_VECTOR (log2(num_reg)-1 downto 0); -- A address to read from
            addr_B : in  STD_LOGIC_VECTOR (log2(num_reg)-1 downto 0); -- B address to read from
            addr_C : in  STD_LOGIC_VECTOR (log2(num_reg)-1 downto 0); -- C address to write to.
            data_in : in  STD_LOGIC_VECTOR (data_size-1 downto 0); -- The data to write
            wr_en : in  STD_LOGIC; -- write enable
            clk : in STD_LOGIC;
            A_out : out  STD_LOGIC_VECTOR (data_size-1 downto 0); -- Data read from addr_A
            B_out : out  STD_LOGIC_VECTOR (data_size-1 downto 0) -- Data read from addr_B
      );
end Pram_Registers;

architecture Behavioral of Pram_Registers is

      signal A_index : STD_LOGIC_VECTOR (num_reg-1 downto 0); -- Decoded address signal for A
      signal B_index : STD_LOGIC_VECTOR (num_reg-1 downto 0); -- Decoded address signal for B
      signal C_index : STD_LOGIC_VECTOR (num_reg-1 downto 0); -- Decoded address signal for C
                                                              (writes)

      signal reg_out : STD_LOGIC_VECTOR ((data_size*num_reg)-1 downto 0); -- output bus from all
                                                                          of the registers

begin

      -- Address decoder for reads on A
      A_decoder: entity work.decoder
      GENERIC MAP( num_reg => num_reg )
      PORT MAP(
            addr => addr_A,
            enables => A_index,
            en => '1'
      );
```

(Continued.)

```
    -- Address decoder for reads on B
    B_decoder: entity work.decoder
    GENERIC MAP( num_reg => num_reg )
    PORT MAP(
          addr => addr_B,
          enables => B_index,
          en => '1'
    );

    -- Address decoder for writing
    C_decoder: entity work.decoder
    GENERIC MAP( num_reg => num_reg )
    PORT MAP(
          addr => addr_C,
          enables => C_index,
          en => wr_en
    );

    -- Create the special R(0) 'register' by have zero inputs to it's tristate buffers
    triestate_R0_A: entity work.triestate_buffer
    GENERIC MAP( data_size => data_size )
    PORT MAP(
          data_in => (others => '0'),
          en => A_index(0),
          data_out => A_out
    );
    triestate_R0_B: entity work.triestate_buffer
    GENERIC MAP( data_size => data_size )
    PORT MAP(
          data_in => (others => '0'),
          en => B_index(0),
          data_out => B_out
    );


    -- Generate registers and tri-states for R(1) to R(N-1)
    generator: for i in 1 to num_reg-1 generate

          -- Tristate for the A_itrn bus
          triestate_A: entity work.triestate_buffer
          GENERIC MAP( data_size => data_size )
          PORT MAP(
                data_in => reg_out( (i+1)*data_size -1 downto i*data_size),
                en => A_index(i),
                data_out => A_out
          );

          -- Tristate for the B_itrn bus
          triestate_B: entity work.triestate_buffer
          GENERIC MAP( data_size => data_size )
          PORT MAP(
                data_in => reg_out( (i+1)*data_size -1 downto i*data_size),
                en => B_index(i),
                data_out => B_out
          );
```

(Continued.)

```
            -- Registers
            Inst_param_reg: entity work.param_reg
            GENERIC MAP( data_size => data_size )
            PORT MAP(
                  data_in => data_in,
                  load => C_index(i),
                  clk => clk,
                  data_out => reg_out( (i+1)*data_size -1 downto i*data_size)
            );

      end generate;
```

Register.vhd

```
----------------------------------------------------------------------------------
--
-- Uni             :     University of York
-- Course          :     Electronic Engineering
-- Module          :     Computer Architectures
-- Engineers       :     Y3839090 & Y3840426
--
-- Create Date     :     20:12:10 02/23/2017
-- Design Name     :     register - Behavioral
-- Description     :     A parameterizable synchronous write asynchronous read register.
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity param_reg is
      Generic
      (
            data_size : natural := 16 -- The data size of the value stored in bits
      );
      Port (
            data_in : in  STD_LOGIC_VECTOR (data_size-1 downto 0); -- Data input
            load : in  STD_LOGIC; -- Loads the data from data_in and stores it
            clk : in  STD_LOGIC;
            data_out : out  STD_LOGIC_VECTOR (data_size-1 downto 0) -- The currently stored
value
      );
end param_reg;

architecture Behavioral of param_reg is
begin

      reg_proc : process (clk) is
      begin
```

(Continued.)

```
            -- synchronously write data_in to data_out if load is high
            if (rising_edge(clk) and load = '1') then
                  data_out <= data_in;
            end if;

      end process;

end Behavioral;
```

TristateBuffer.vhd

```
-- Uni             :     University of York
-- Course          :     Electronic Engineering
-- Module          :     Computer Architectures
-- Engineers       :     Y3839090 & Y3840426
--
-- Create Date     :     21:02:48 02/23/2017
-- Design Name     :     tristate_buffer - Behavioral
-- Description     :     A parameteriable tristate buffer array.
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity triestate_buffer is
      Generic
      (
            data_size : natural := 8 -- number of trie state buffers
      );
      Port (
            data_in : in  STD_LOGIC_VECTOR(data_size-1 downto 0); -- data input
            en : in  STD_LOGIC; -- tristate enable
            data_out : out  STD_LOGIC_VECTOR(data_size-1 downto 0) -- data out
      );
end triestate_buffer;

architecture Behavioral of triestate_buffer is
begin

      -- Tristate buffer implementation from lab script
      DATA_OUT <=
            DATA_IN when (En = '1') else
            (others => 'Z'); -- Z = high-impedance


end Behavioral;
```

Decoder.vhd

```
----------------------------------------------------------------------------------
--
-- Uni             :     University of York
-- Course          :     Electronic Engineering
-- Module          :     Computer Architectures
-- Engineers       :     Y3839090 & Y3840426
--
-- Create Date     :     21:02:48 02/23/2017
-- Design Name     :     decoder - Behavioral
-- Description     :     A parameteriable address decoder.
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.DigEng.ALL;

entity decoder is
      Generic
      (
            num_reg : natural := 8 -- number of address locations
      );
      Port (
            addr : in  STD_LOGIC_VECTOR (log2(num_reg)-1 downto 0); -- input address
            en : in STD_LOGIC; -- enable
            enables : out  STD_LOGIC_VECTOR (num_reg-1 downto 0) -- decoded output.
       );
end decoder;

architecture Behavioral of decoder is
begin

      -- decodes the addresses
      process(addr, en) is
      begin

            -- loop over every bit in the output
            for i in 0 to num_reg-1 loop
                  -- if it this bit corresponds to the input address set it high otherwise set
                     it low.
         if (i = to_integer(unsigned(addr)) and en = '1') then
           enables(i) <= '1';
         else
           enables(i) <= '0';
         end if;

            end loop;

      end process;

end Behavioral;
```

ParamRegisterTB.vhd

```
----------------------------------------------------------------------------------
--
-- Uni                  :     University of York
-- Course          :     Electronic Engineering
-- Module          :     Computer Architectures
-- Engineers       :     Y3839090 & Y3840426
--
-- Create Date     :     23:08:04 02/23/2017
-- Design Name     :     Param_Registers_TB - TestBench
-- Description     :     A 16 bit 8 register dual read single write register array.
--
----------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE work.DigEng.ALL;

ENTITY Param_Registers_TB IS
END Param_Registers_TB;

ARCHITECTURE behavior OF Param_Registers_TB IS


      -- Constants
      constant num_reg : natural := 8;
      constant data_size : natural := 16;

      -- Component Declaration for the Unit Under Test (UUT)
      COMPONENT Pram_Registers
            GENERIC(
                  num_reg : natural;
                  data_size : natural
            );
            PORT(
                  addr_A : IN  std_logic_vector(log2(num_reg)-1 downto 0);
                  addr_B : IN  std_logic_vector(log2(num_reg)-1 downto 0);
                  addr_C : IN  std_logic_vector(log2(num_reg)-1 downto 0); -- Address to write
                                                                              to
                  data_in : IN  std_logic_vector(data_size-1 downto 0); -- Data to write
                  wr_en : IN  std_logic;
                  clk : IN  std_logic;
                  A_out : OUT  std_logic_vector(data_size-1 downto 0);
                  B_out : OUT  std_logic_vector(data_size-1 downto 0)
            );
    END COMPONENT;


   --Inputs
   signal addr_A : std_logic_vector(log2(num_reg)-1 downto 0) := (others => '0');
   signal addr_B : std_logic_vector(log2(num_reg)-1 downto 0) := (others => '0');
   signal addr_C : std_logic_vector(log2(num_reg)-1 downto 0) := (others => '0');
   signal data_in : std_logic_vector(data_size-1 downto 0) := (others => '0');
   signal wr_en : std_logic := '0';
   signal clk : std_logic := '0';
```

(Continued.)

```
   --Outputs
   signal A_out : std_logic_vector(data_size-1 downto 0);
   signal B_out : std_logic_vector(data_size-1 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

      type TEST_VECTOR is RECORD
            wr_en    : std_logic; -- Write enable
            addr_A   : std_logic_vector(log2(num_reg)-1 downto 0); -- A address to read from
      addr_B   : std_logic_vector(log2(num_reg)-1 downto 0); -- B address to read from
      addr_C   : std_logic_vector(log2(num_reg)-1 downto 0); -- C address to write to
      data_in : std_logic_vector(data_size-1 downto 0); -- Data to write at addr_C
            A_out    : std_logic_vector(data_size-1 downto 0); -- Expected data to be read from
                                                               addr_A
      B_out    : std_logic_vector(data_size-1 downto 0); -- Expected data to be read from addr_B
      end RECORD;

      type TEST_VECTOR_ARRAY is ARRAY(NATURAL range <>) of TEST_VECTOR;

      constant test_vectors : TEST_VECTOR_ARRAY :=
      (
          -- wr_en,    addr_A,     addr_B,     addr_C,     data_in,      A_out,        B_out
            ('0',      "000",      "000",      "000",      X"0000",      X"0000",      X"0000"),
            ('1',      "000",      "000",      "001",      X"FFFF",      X"0000",      X"0000"),
            ('1',      "000",      "000",      "010",      X"1111",      X"0000",      X"0000"),
            ('0',      "001",      "010",      "000",      X"0000",      X"FFFF",      X"1111"),
            ('1',      "000",      "000",      "111",      X"0011",      X"0000",      X"0000"),
            ('1',      "111",      "111",      "110",      X"8800",      X"0011",      X"0011"),
            ('1',      "111",      "110",      "000",      X"AAAA",      X"0011",      X"8800"),
            ('0',      "000",      "000",      "000",      X"0000",      X"0000",      X"0000"),

            ('1',      "000",      "001",      "010",      X"1111",      X"0000",      X"FFFF"),
            ('1',      "001",      "010",      "011",      X"2222",      X"FFFF",      X"1111"),
            ('1',      "010",      "011",      "100",      X"3333",      X"1111",      X"2222"),
            ('1',      "011",      "100",      "101",      X"4444",      X"2222",      X"3333"),
            ('1',      "100",      "101",      "110",      X"5555",      X"3333",      X"4444"),
            ('1',      "101",      "110",      "111",      X"6666",      X"4444",      X"5555"),
            ('0',      "110",      "111",      "000",      X"0000",      X"5555",      X"6666")


      );

BEGIN

      -- Instantiate the Unit Under Test (UUT)
      uut: Pram_Registers
      GENERIC MAP(
            num_reg => num_reg,
            data_size => data_size
      )
```

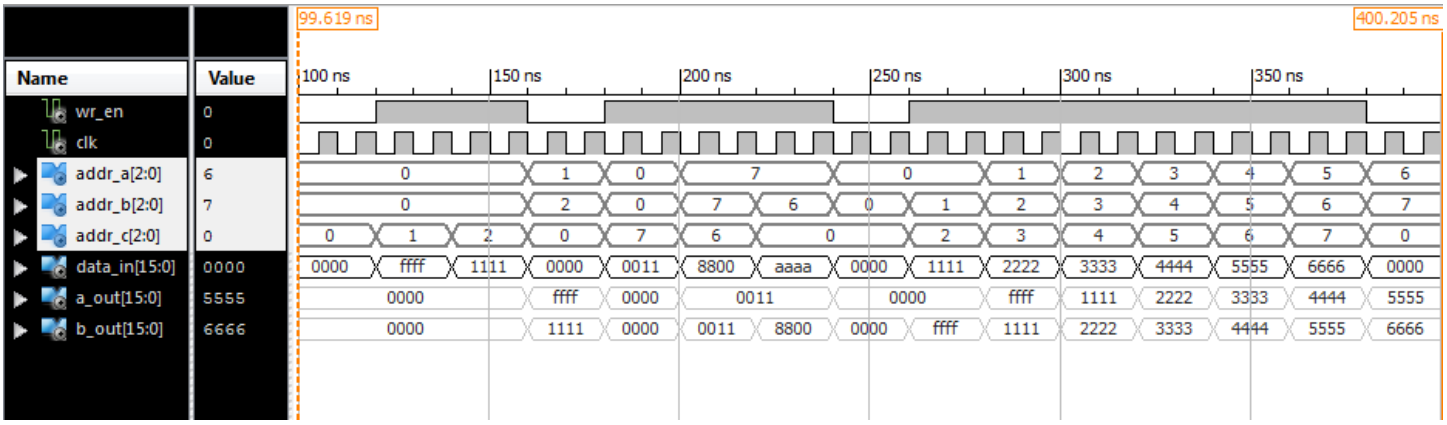
(Continued.)

```
   PORT MAP (
         addr_A => addr_A,
         addr_B => addr_B,
         addr_C => addr_C,
         wr_en => wr_en,
         clk => clk,
         data_in => data_in,
         A_out => A_out,
         B_out => B_out
   );

-- Clock process definitions
clk_process :process
begin
         clk <= '0';
         wait for clk_period/2;
         clk <= '1';
         wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin

         -- hold reset state for 100 ns.
         wait for 100 ns;

         -- Loop over all of our test data sets
         for i in test_vectors'range loop

               -- assign test inputs
               wr_en <= test_vectors(i).wr_en;
               addr_A <= test_vectors(i).addr_A;
               addr_B <= test_vectors(i).addr_B;
               addr_C <= test_vectors(i).addr_C;
               data_in <= test_vectors(i).data_in;


               -- wait for a clock cycle
               wait for clk_period;


               -- check that output A is the same as the expected
               assert A_out = test_vectors(i).A_out
               report " [ERR!] Test " & integer'image(i) & " : A output is not what was
               expected!"
               severity error;

               -- check that output A is the same as the expected
               assert B_out = test_vectors(i).B_out
               report " [ERR!] Test " & integer'image(i) & " : B output is not what was
               expected!"
               severity error;
```

(Continued.)

```
                -- check that output A is the same as the expected
                assert (B_out /= test_vectors(i).B_out or A_out /= test_vectors(i).A_out)
                report " [ OK ] Test " & integer'image(i) & " passed!"
                severity note;

                wait for clk_period;

            end loop;

    wait;
    end process;

END;
```

Simulations



Simulation Screenshot showing the full operation of the Parameterizable Register Bank

ISim Console Output

```
ISim P.28xd (signature 0xa0883be4)
This is a Full version of ISim.

# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 110 ns: Note:  [ OK ] Test 0 passed! (/param_registers_tb/).
at 130 ns: Note:  [ OK ] Test 1 passed! (/param_registers_tb/).
at 150 ns: Note:  [ OK ] Test 2 passed! (/param_registers_tb/).
at 170 ns: Note:  [ OK ] Test 3 passed! (/param_registers_tb/).
at 190 ns: Note:  [ OK ] Test 4 passed! (/param_registers_tb/).
at 210 ns: Note:  [ OK ] Test 5 passed! (/param_registers_tb/).
at 230 ns: Note:  [ OK ] Test 6 passed! (/param_registers_tb/).
at 250 ns: Note:  [ OK ] Test 7 passed! (/param_registers_tb/).
at 270 ns: Note:  [ OK ] Test 8 passed! (/param_registers_tb/).
at 290 ns: Note:  [ OK ] Test 9 passed! (/param_registers_tb/).
at 310 ns: Note:  [ OK ] Test 10 passed! (/param_registers_tb/).
at 330 ns: Note:  [ OK ] Test 11 passed! (/param_registers_tb/).
at 350 ns: Note:  [ OK ] Test 12 passed! (/param_registers_tb/).
at 370 ns: Note:  [ OK ] Test 13 passed! (/param_registers_tb/).
at 390 ns: Note:  [ OK ] Test 14 passed! (/param_registers_tb/).
ISim>
```

HDL Synthesis Report

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Synthesizing Unit <Pram_Registers>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_1\ParameterizableRegisterBank\Pram_Registers.vhd".
        num_reg = 8
        data_size = 16
    Summary:
      no macro.
Unit <Pram_Registers> synthesized.

Synthesizing Unit <decoder>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_1\ParameterizableRegisterBank\decoder.vhd".
        num_reg = 8
    Summary:
      no macro.
Unit <decoder> synthesized.

Synthesizing Unit <triestate_buffer>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_1\ParameterizableRegisterBank\triestate_buffer.vhd
".
```

(Continued.)

```
Synthesizing Unit <triestate_buffer>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_1\ParameterizableRegisterBank\triestate_buffer.vhd
".
        data_size = 16
    Found 1-bit tristate buffer for signal <data_out<15>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<14>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<13>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<12>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<11>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<10>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<9>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<8>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<7>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<6>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<5>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<4>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<3>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<2>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<1>> created at line 38
    Found 1-bit tristate buffer for signal <data_out<0>> created at line 38
    Summary:
      inferred  16 Tristate(s).
Unit <triestate_buffer> synthesized.

Synthesizing Unit <param_reg>.
    Related source file is "E:\University\_Second Year\Computer
Architectures\assesment\MyMicroProccessor\Lab_1\ParameterizableRegisterBank\register.vhd".
        data_size = 16
    Found 16-bit register for signal <internal>.
    Summary:
      inferred  16 D-type flip-flop(s).
Unit <param_reg> synthesized.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers                                            : 7
 16-bit register                                       : 7
# Tristates                                            : 256
 1-bit tristate buffer                                 : 256

=========================================================================
```