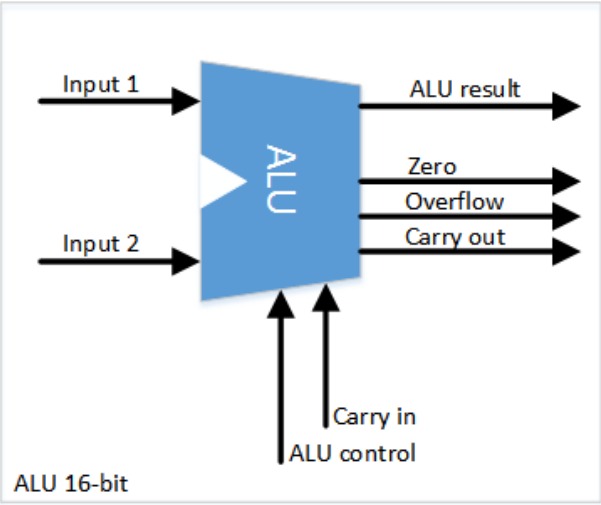
TUNG THANH LE
CACS - UL Lafayette
ttungl at gmail

A design of ALU-16bit using Xilinx ISE 14.7, ModelSim, iSIM & implementation via Spartan 3E FPGA Starter Kit

*Authors: **Tung Thanh Le and Mingmin Bai** have completed on this project. The first author has worked on designing ALU16bit and simulated via Xilinx ISE 14.7, and ModelSim, the second author has worked on LCD display. Then, we have merged the design to display ALU results on LCD of Spartan 3E FPGA Starter kit.*
*The documentation has been submitted to CMPS430 course, taught by **Dr. Danella Zhao**, CACS, UL Lafayette.*

1. Design principle:

In this project, I have designed a 16-bit ALU that can be able to control the operations including arithmetic operations (ADD16, SUB16), logic operations (AND, OR, NOR, SLT), and shifting operations (SLL, SRL). The ALU 16-bit operation is described as in Figure below.



The ALU control operates to switch between operations through the binary codes as shown in Table below.

ALU Control Bits	Operation
000	Addition
001	Subtraction
010	And logic
011	Or logic
100	Nor logic
101	Set on less than
110	Shift left logic
111	Shift right logic

In addition, the outputs are consisting of ALU result, Zero, Overflow, and Carry out. ALU result is obtained through each operation's result which is selected. While Zero bit is set '1' if all ALU result equals to zero, otherwise. Overflow shows the result of the last carry_in XOR carry_out in an operation of ALU. If this XOR outcome is '1', Overflow bit is set '1', otherwise. Carry out is the combination of input1, input2, and carry in through AND and OR logic.

To sum up, the ALU 16-bit has four inputs and four outputs as illustrated in Figure above. Now, we move to the simulation in the next section.

2. Simulation

In this section, VHDL coding is used for simulation on Xilinx ISE 14.7 and ModelSim/iSIM simulators. I think that it could be better to show the codes since it is very straight-forward to understand what it does.

i. Full Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
--
entity FullAdder is
    Port ( input1 : in  STD_LOGIC;
```

```
           input2 : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           cout : out  STD_LOGIC;
           sum : out  STD_LOGIC);
end FullAdder;

architecture Behavioral of FullAdder is

begin

 sum <=  input1 xor input2 xor cin;
 cout <= (input1 and input2) or (input1 and cin) or (input2
and cin);

end Behavioral;
```

ii. ADD16b:

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

entity ADD16b is
    Generic (n : integer :=16);
    Port ( input1 : in  STD_LOGIC_VECTOR(n-1 downto 0); --
           input2 : in  STD_LOGIC_VECTOR(n-1 downto 0);
                    cin: in STD_LOGIC;
                    cout_add : inout STD_LOGIC;
                    overflow_add : out STD_LOGIC;
           out_add : out  STD_LOGIC_VECTOR (n-1 downto 0));
end ADD16b;

architecture Behavioral of ADD16b is
-- component definition
    COMPONENT FullAdder
    PORT(
      input1, input2: in STD_LOGIC; --
      cin : in STD_LOGIC;
      sum: out STD_LOGIC;
      cout: out STD_LOGIC --
      );
      END COMPONENT;

-- signals definition
```

```
      signal t: STD_LOGIC_VECTOR (1 to n-1);
      signal c_cout: STD_LOGIC;

begin
      -- Adder 16bit
      U1: FullAdder
            port map (cin, input1(0), input2(0),
out_add(0), t(1));

      U_Generator: For i in 1 to n-2 generate
            U_i: FullAdder
                  port map (t(i), input1(i), input2(i),
out_add(i), t(i+1));
      end generate;

      U_n: FullAdder
            port map (t(n-1), input1(n-1), input2(n-1),
out_add(n-1), cout_add);

      -- Overflow checker
      c_cout <= cout_add;
      overflow_add <= t(n-1) xor c_cout;


end Behavioral;
```

iii. SUB16b:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.NUMERIC_STD.ALL;
--
entity SUB16b is
       Generic (n: integer :=16);
    Port ( input1 : in  STD_LOGIC_VECTOR (n-1 downto 0);
           input2 : in  STD_LOGIC_VECTOR (n-1 downto 0);
                    cin: in STD_LOGIC;
                    cout_sub : inout STD_LOGIC;
                    overflow_sub : out STD_LOGIC;
           out_sub : out  STD_LOGIC_VECTOR (n-1 downto 0));
end SUB16b;

architecture Behavioral of SUB16b is

      -- component definition
      COMPONENT FullAdder
    PORT(
      input1, input2: in STD_LOGIC; --
```

```
            cin : in STD_LOGIC;
      sum: out STD_LOGIC;
            cout: out STD_LOGIC --
            );
      END COMPONENT;

      -- signals definition
      signal t: STD_LOGIC_VECTOR (1 to n-1); -- carries
      signal invertInput2 : STD_LOGIC_VECTOR (0 to n-1);
      signal cin_0: STD_LOGIC_VECTOR(0 to n-1) := X"0001";
      signal c_cout: STD_LOGIC;
begin
inv_Subtraction: for i in 0 to n-1 generate
      invertInput2(i) <= input2(i) XOR cin_0(i);
            end generate inv_Subtraction;


      -- Adder 16bit
U1: FullAdder
   port map (cin, input1(0), invertInput2(0), out_sub(0),
t(1));

U_Generator: For i in 1 to n-2 generate
   U_i: FullAdder
   port map (t(i), input1(i), invertInput2(i), out_sub(i),
t(i+1));
   end generate;

U_n: FullAdder
   port map (t(n-1), input1(n-1), invertInput2(n-1),
out_sub(n-1), cout_sub);

      -- Overflow checker
      c_cout <= cout_sub;
      overflow_sub <= t(n-1) xor c_cout;

end Behavioral;
```

i. AND16b:

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

entity AND16b is
       Generic (n : integer :=16);
    Port ( input1 : in  STD_LOGIC_VECTOR (n-1 downto 0);
```

```
           input2 : in  STD_LOGIC_VECTOR (n-1 downto 0);
                     cout_and: out STD_LOGIC := '0';
           out_and : out  STD_LOGIC_VECTOR (n-1 downto 0));
end AND16b;

architecture Behavioral of AND16b is

begin

      AND_proc: process
            begin

            for i in n-1 downto 0 loop
                  out_and(i) <= input1(i) AND input2(i);
            end loop;
            wait for 0.1 ns;
end process AND_proc;
end Behavioral;
```

ii. OR16b:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity OR16b is
       Generic (n : integer :=16);
    Port ( input1 : in  STD_LOGIC_VECTOR (n-1 downto 0);
           input2 : in  STD_LOGIC_VECTOR (n-1 downto 0);
                     cout_or : out STD_LOGIC := '0';
           out_or : out  STD_LOGIC_VECTOR (n-1 downto 0));
end OR16b;

architecture Behavioral of OR16b is

begin

      OR_proc: process
            begin
            for i in n-1 downto 0 loop
                  out_or(i) <= input1(i) OR input2(i);
            end loop;
            wait for 0.1 ns;
            end process OR_proc;
end Behavioral;
```

iii. NOR16b:

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

```
entity NOR16b is
       Generic (n : integer :=16);
    Port ( input1 : in  STD_LOGIC_VECTOR (n-1 downto 0);
           input2 : in  STD_LOGIC_VECTOR (n-1 downto 0);
                     cout_nor : out STD_LOGIC := '0';
           out_nor : out  STD_LOGIC_VECTOR (n-1 downto 0));
end NOR16b;

architecture Behavioral of NOR16b is

begin
      NOR_proc: process

      begin

            for i in n-1 downto 0 loop
                  out_nor(i) <= input1(i) NOR input2(i);
            end loop;

            wait for 0.1 ns;
            end process NOR_proc;
end Behavioral;
```

iv. SLT16b:

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

entity SLT16b is
       Generic (n: integer :=16);
    Port( input1 : in  STD_LOGIC_VECTOR (n-1 downto 0);
           input2 : in  STD_LOGIC_VECTOR (n-1 downto 0);
                     cout_slt : out STD_LOGIC := '0';
           out_slt : out  STD_LOGIC_VECTOR (n-1 downto 0));
end SLT16b;

architecture Behavioral of SLT16b is


begin
      SetLessThan: process
            begin

                  if (input1 < input2) then
                        out_slt <= "0000000000000001";
                  else
```

```
                        out_slt <= "0000000000000000";
                    end if;
                wait for 0.1 ns;

      end process SetLessThan;
end Behavioral;
```

v. SLL16b:

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

entity SLL16b is
       Generic (n : integer :=16);
    Port ( input1 : in  STD_LOGIC_VECTOR (n-1 downto 0);
                     cout_sll : out STD_LOGIC := '0';
           out_sll : out  STD_LOGIC_VECTOR (n-1 downto 0));
end SLL16b;

architecture Behavioral of SLL16b is

begin

      Result_proc1: process
      begin
            for i in 0 to 2 loop
                  out_sll(i) <= '0';
                  end loop;
            wait for 0.1 ns;


            for i in 15 downto 3 loop
                  out_sll(i) <= input1(i-3);
                  end loop;
            wait for 0.1 ns;

end process Result_proc1;

end Behavioral;
```

vi. SRL16b:

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

```
entity SRL16b is
       Generic (n: integer :=16);
    Port ( input1 : in  STD_LOGIC_VECTOR (n-1 downto 0);
                      cout_srl : out STD_LOGIC := '0';
           out_srl : out  STD_LOGIC_VECTOR (n-1 downto 0));
end SRL16b;

architecture Behavioral of SRL16b is


begin

      Result_proc: process
            begin
                  for i in 15 downto 13 loop
                        out_srl(i) <= '0';
                        end loop;
                  wait for 0.1 ns;

                  for i in 12 downto 0 loop
                        out_srl(i) <= input1(i+3);
                        end loop;
                  wait for 0.1 ns;

      end process Result_proc;

end Behavioral;
```

b. Testbench

```
LIBRARY ieee;
  USE ieee.std_logic_1164.ALL;
  USE ieee.std_logic_ARITH.ALL;
  USE ieee.std_logic_UNSIGNED.ALL;
  USE ieee.numeric_std.ALL;

ENTITY ALU16bit_testbench IS
  END ALU16bit_testbench;

  ARCHITECTURE behavior OF ALU16bit_testbench IS

-- signal declaration
signal t_input1:  STD_LOGIC_VECTOR (15 downto 0);
signal t_input2 :  STD_LOGIC_VECTOR (15 downto 0);
signal t_cin :  STD_LOGIC;
signal t_ALUcontrol:  STD_LOGIC_VECTOR (2 downto 0);
signal t_Overflow :  STD_LOGIC;
signal t_carryout :  STD_LOGIC;
```

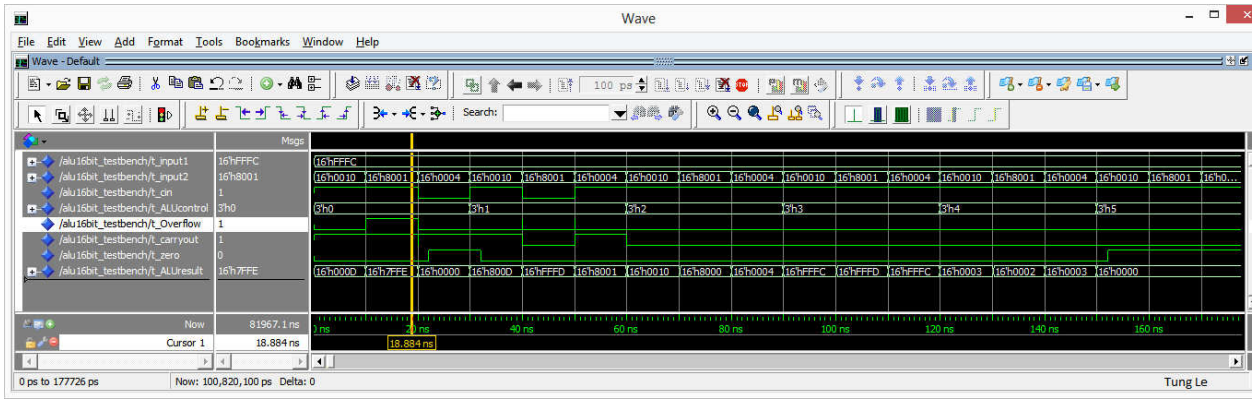
```
signal t_zero :  STD_LOGIC;
signal t_ALUresult :  STD_LOGIC_VECTOR (15 downto 0);


-- Component Declaration
COMPONENT ALU16bit
PORT(
  input1 : in  STD_LOGIC_VECTOR (15 downto 0);
  input2 : in  STD_LOGIC_VECTOR (15 downto 0);
  cin : in STD_LOGIC;
  ALUcontrol: in STD_LOGIC_VECTOR (2 downto 0);
  Overflow : out  STD_LOGIC;
  carryout : inout  STD_LOGIC;
  zero : out  STD_LOGIC;
  ALUresult : inout  STD_LOGIC_VECTOR (15 downto 0)
  );
END COMPONENT;

BEGIN

-- Component Instantiation
uut: ALU16bit PORT MAP(
input1 => t_input1,
input2 => t_input2,
cin => t_cin,
ALUcontrol => t_ALUcontrol,
Overflow => t_Overflow,
carryout => t_carryout,
zero => t_zero,
ALUresult => t_ALUresult

);

tb : PROCESS
     BEGIN

-- ADD16b

             t_input1 <= X"FFFC";
             t_input2 <= X"0010";
             t_cin <= '1';
             t_ALUcontrol <= "000";
             WAIT FOR 10 ns;

             t_input1 <= X"FFFC";
             t_input2 <= X"8001";
             t_cin <= '1';
             t_ALUcontrol <= "000";
             WAIT FOR 10 ns;
```

```
            t_input1 <= X"FFFC";
            t_input2 <= X"0004";
            t_cin <= '0';
            t_ALUcontrol <= "000";
            WAIT FOR 10 ns;

-- SUB16b
            t_input1 <= X"FFFC";
            t_input2 <= X"0010";
            t_cin <= '1';
            t_ALUcontrol <= "001";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"8001";
            t_cin <= '0';
            t_ALUcontrol <= "001";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"0004";
            t_cin <= '1';
            t_ALUcontrol <= "001";
            WAIT FOR 10 ns;
-- AND16b
                  t_input1 <= X"FFFC";
                  t_input2 <= X"0010";
            t_ALUcontrol <= "010";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"8001";
            t_ALUcontrol <= "010";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"0004";
            t_ALUcontrol <= "010";
            WAIT FOR 10 ns;
-- OR16b
            t_input1 <= X"FFFC";
            t_input2 <= X"0010";
            t_ALUcontrol <= "011";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"8001";
            t_ALUcontrol <= "011";
            WAIT FOR 10 ns;
```

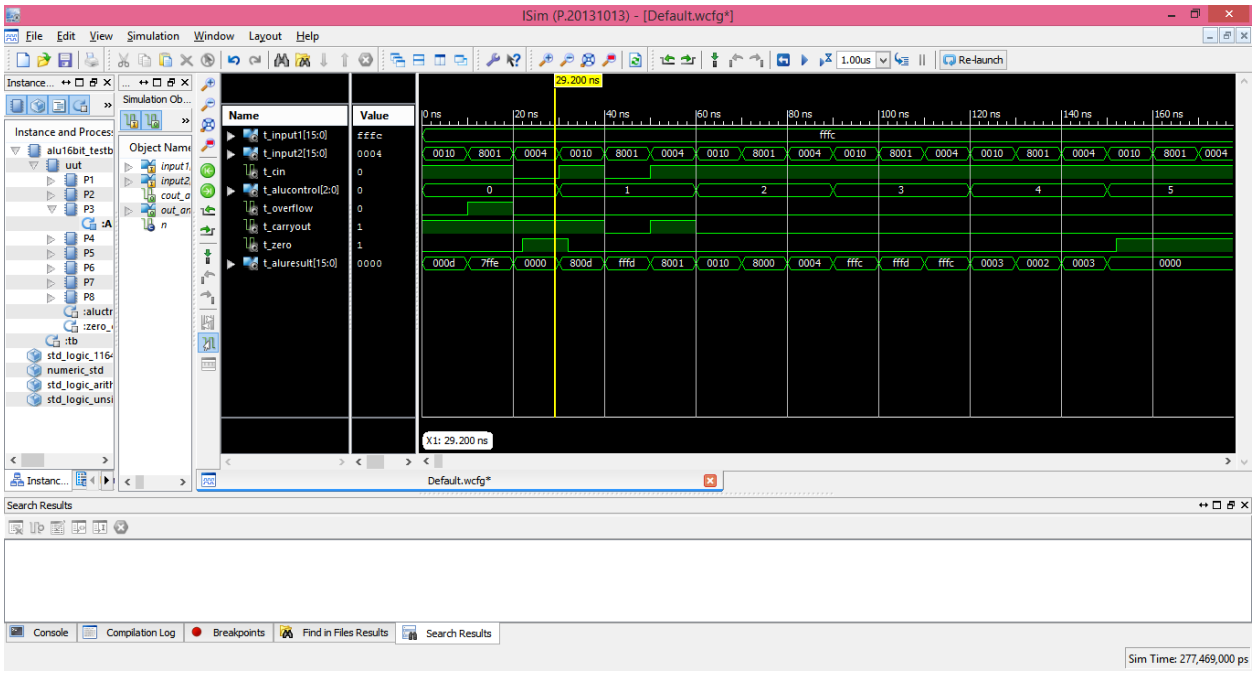
```
            t_input1 <= X"FFFC";
            t_input2 <= X"0004";
            t_ALUcontrol <= "011";
            WAIT FOR 10 ns;
-- NOR16b
            t_input1 <= X"FFFC";
            t_input2 <= X"0010";
            t_ALUcontrol <= "100";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"8001";
            t_ALUcontrol <= "100";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"0004";
            t_ALUcontrol <= "100";
            WAIT FOR 10 ns;
-- SLT16b
            t_input1 <= X"FFFC";
            t_input2 <= X"0010";
            t_ALUcontrol <= "101";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"8001";
            t_ALUcontrol <= "101";
            WAIT FOR 10 ns;

            t_input1 <= X"FFFC";
            t_input2 <= X"0004";
            t_ALUcontrol <= "101";
            WAIT FOR 10 ns;
-- SLL16b
            t_input1 <= X"0004";
            t_ALUcontrol <= "110";
            WAIT FOR 10 ns;
-- SRL16b
             t_input1 <= X"0004";
             t_ALUcontrol <= "111";
             WAIT FOR 10 ns;

wait; -- will wait forever
END PROCESS tb;
--  End Test Bench

END;
```

c. Results

ModelSim simulator:



Xilinx iSIM simulator:



2. Implementation

a. LCD display components:

In this section, I would like to show ALU results via LCD on Spartan 3E FPGA Starter Kit. Basically, LCD display module (called display controller) includes the ALU module, and BinDecConverter, and display_controller.ucf, in order to drive ALU results to LCD display on the board.

BinDecConverter

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

entity dec17_bcd is
    Port ( Cout_in : in  STD_LOGIC;
           Sum_in : in  STD_LOGIC_VECTOR (15 downto 0);
           Unit : out  STD_LOGIC_VECTOR (3 downto 0);
           Ten : out  STD_LOGIC_VECTOR (3 downto 0);
           Hundred : out  STD_LOGIC_VECTOR (3 downto 0);
                     Thousand : out STD_LOGIC_VECTOR (3 downto 0);
                     Tthousand : out STD_LOGIC_VECTOR (3 downto
0);
                     Hthousand : out STD_LOGIC_VECTOR (3 downto
0));
end dec17_bcd;

architecture behavioral of dec17_bcd is

signal BIN2DEC : integer range 0 to 131071;

begin

BIN2DEC <= conv_integer(unsigned(Cout_in&Sum_in (15 downto 0)));


conversion: process (BIN2DEC)

variable RES99_VAR : integer range 0 to 99999;
variable VAR_UNIS : integer range 0 to 9;
variable VAR_TENS : integer range 0 to 9;
variable VAR_HUNDS : integer range 0 to 9;
variable VAR_THOUS : integer range 0 to 9;
variable VAR_TTHOUS : integer range 0 to 9;
variable VAR_HTHOUS : integer range 0 to 1;

begin
      if BIN2DEC > 99999 then VAR_HTHOUS := 1;
      else VAR_HTHOUS := 0;
      end if;

      RES99_VAR := BIN2DEC - (VAR_HTHOUS * 100000);
      if RES99_VAR > 89999 then VAR_TTHOUS := 9;
      elsif RES99_VAR > 79999 then VAR_TTHOUS := 8;
      elsif RES99_VAR > 69999 then VAR_TTHOUS := 7;
```

```
      elsif RES99_VAR > 59999 then VAR_TTHOUS := 6;
      elsif RES99_VAR > 49999 then VAR_TTHOUS := 5;
      elsif RES99_VAR > 39999 then VAR_TTHOUS := 4;
      elsif RES99_VAR > 29999 then VAR_TTHOUS := 3;
      elsif RES99_VAR > 19999 then VAR_TTHOUS := 2;
      elsif RES99_VAR > 9999 then VAR_TTHOUS := 1;
      else VAR_TTHOUS := 0;
      end if;

      RES99_VAR := RES99_VAR - (VAR_TTHOUS * 10000);
      if RES99_VAR > 8999 then VAR_THOUS := 9;
      elsif RES99_VAR > 7999 then VAR_THOUS := 8;
      elsif RES99_VAR > 6999 then VAR_THOUS := 7;
      elsif RES99_VAR > 5999 then VAR_THOUS := 6;
      elsif RES99_VAR > 4999 then VAR_THOUS := 5;
      elsif RES99_VAR > 3999 then VAR_THOUS := 4;
      elsif RES99_VAR > 2999 then VAR_THOUS := 3;
      elsif RES99_VAR > 1999 then VAR_THOUS := 2;
      elsif RES99_VAR > 999 then VAR_THOUS := 1;
      else VAR_THOUS := 0;
      end if;

      RES99_VAR := RES99_VAR - (VAR_THOUS * 1000);
      if RES99_VAR > 899 then VAR_HUNDS := 9;
      elsif RES99_VAR > 799 then VAR_HUNDS := 8;
      elsif RES99_VAR > 699 then VAR_HUNDS := 7;
      elsif RES99_VAR > 599 then VAR_HUNDS := 6;
      elsif RES99_VAR > 499 then VAR_HUNDS := 5;
      elsif RES99_VAR > 399 then VAR_HUNDS := 4;
      elsif RES99_VAR > 299 then VAR_HUNDS := 3;
      elsif RES99_VAR > 199 then VAR_HUNDS := 2;
      elsif RES99_VAR > 99 then VAR_HUNDS := 1;
      else VAR_HUNDS := 0;
      end if;

      RES99_VAR := RES99_VAR - (VAR_HUNDS * 100);
      if RES99_VAR > 89 then VAR_TENS := 9;
      elsif RES99_VAR > 79 then VAR_TENS := 8;
      elsif RES99_VAR > 69 then VAR_TENS := 7;
      elsif RES99_VAR > 59 then VAR_TENS := 6;
      elsif RES99_VAR > 49 then VAR_TENS := 5;
      elsif RES99_VAR > 39 then VAR_TENS := 4;
      elsif RES99_VAR > 29 then VAR_TENS := 3;
      elsif RES99_VAR > 19 then VAR_TENS := 2;
      elsif RES99_VAR > 9 then VAR_TENS := 1;
      else VAR_TENS := 0;
      end if;

      VAR_UNIS := RES99_VAR - (VAR_TENS * 10);
Unit <= conv_std_logic_vector (VAR_UNIS, 4);
```

```
      Ten <= conv_std_logic_vector (VAR_TENS, 4);
      Hundred <= conv_std_logic_vector (VAR_HUNDS, 4);
      Thousand <= conv_std_logic_vector (VAR_THOUS, 4);
      Tthousand <= conv_std_logic_vector (VAR_TTHOUS, 4);
      Hthousand <= conv_std_logic_vector (VAR_HTHOUS, 4);
      end process conversion;


end behavioral;
```

Display_controller.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Display_controller is

      Port (clk    : in std_logic;
                  rst    : in std_logic;
                  SF_D   : out std_logic_vector(11 downto 8);
                  LCD_E  : out std_logic;
                  LCD_RS : out std_logic;
                  LCD_RW : out std_logic;
                  t_ALUcontrol : in std_logic_vector(2 downto 0);
                  t_Overflow : out std_logic;
                  t_zero : out std_logic
                  );

      end Display_controller;

architecture Behavioral of Display_controller is

component ALU16bit is
Port (          input1 : in  STD_LOGIC_VECTOR (15 downto 0);
           input2 : in  STD_LOGIC_VECTOR (15 downto 0);
           cin : in  STD_LOGIC;
                     ALUcontrol: in STD_LOGIC_VECTOR (2 downto 0);

                     Overflow : out  STD_LOGIC;
           carryout : inout  STD_LOGIC;
           zero : out  STD_LOGIC;
           ALUresult : inout  STD_LOGIC_VECTOR (15 downto 0)
```

```
                    );
end component;


component dec17_bcd is
Port ( Cout_in : in  STD_LOGIC;
           Sum_in : in  STD_LOGIC_VECTOR (15 downto 0);
           Unit : out  STD_LOGIC_VECTOR (3 downto 0);
           Ten : out  STD_LOGIC_VECTOR (3 downto 0);
           Hundred : out  STD_LOGIC_VECTOR (3 downto 0);
                    Thousand : out STD_LOGIC_VECTOR (3 downto 0);
                    Tthousand : out STD_LOGIC_VECTOR (3 downto
0);
                    Hthousand : out STD_LOGIC_VECTOR (3 downto
0));
end component;

type istate_t is (istep_one, istep_two, istep_three, istep_four,
istep_five,
                  istep_six, istep_seven, istep_eight, istep_nine,
                  function_set, entry_mode, control_display,
clear_display,
                  init_done);

type dstate_t is (didle, set_start_address, write_data_HT,
write_data_TT,
                  write_data_TH, write_data_H, write_data_T,
write_data_U, --return_home,
            address_digit, write_digit);


signal istate, next_istate   : istate_t;--state and next state of
the init. sm
signal dstate, next_dstate   : dstate_t;--state and next state of
the display sm
signal idone, next_idone     : std_logic;--initialization done
signal count, next_count     : integer range 0 to 750000;
signal nibble                : std_logic_vector(3 downto 0);
signal enable, next_enable   : std_logic;--register enable signal
put out to LCD_E
signal regsel, next_regsel   : std_logic;--register select signal
put out to LCD_RS
signal byte                  : std_logic_vector(7 downto 0); --
data to pass to SF_D
signal timer_15ms            : std_logic;
signal timer_4100us          : std_logic;
signal timer_100us           : std_logic;
signal timer_40us            : std_logic;
signal timer_1640us          : std_logic;
signal txdone, next_txdone   : std_logic;
```

```
signal txcount, next_txcount : integer range 0 to 2068;
signal selnibble             : std_logic;
signal next_selnibble        : std_logic;
signal digit, next_digit     : std_logic_vector(3 downto 0);
signal cnt, next_cnt         : integer range 0 to 50000000;

-- ALU16bit
signal input1 : STD_LOGIC_VECTOR (15 downto 0) := X"0010";
signal input2 : STD_LOGIC_VECTOR (15 downto 0) := X"FFFC";
signal cin : STD_LOGIC :='0';
signal t_carryout : STD_LOGIC;
signal t_ALUresult : STD_LOGIC_VECTOR (15 downto 0);

-- LCD
signal digitHT :  STD_LOGIC_VECTOR (3 downto 0);
signal digitTT :  STD_LOGIC_VECTOR (3 downto 0);
signal digitTH :  STD_LOGIC_VECTOR (3 downto 0);
signal digitH :  STD_LOGIC_VECTOR (3 downto 0);
signal digitT :  STD_LOGIC_VECTOR (3 downto 0);
signal digitU :  STD_LOGIC_VECTOR (3 downto 0);


begin

      P1: ALU16bit
            port map (input1 => input1, input2 => input2, cin =>
cin, ALUcontrol => t_ALUcontrol,
            Overflow => t_Overflow, carryout => t_carryout, zero
=> t_zero, ALUresult => t_ALUresult);
      -- Convert bin to dec.
      P2: dec17_bcd
            port map (Cout_in => t_carryout, Sum_in =>
t_ALUresult, Hthousand => digitHT,
            Tthousand => digitTT, Thousand => digitTH, Hundred =>
digitH, Ten => digitT, Unit => digitU);


  --
  -- concurrent assignments (LCD interface)
  --
  LCD_RW <= '0'; --write LCD (LCD accepts data).
                 --putting LCD_RW=0 also prevent the LCD screen
                 --from presenting undesired data.
  SF_D   <= nibble;
  LCD_E  <= enable;
  LCD_RS <= regsel;

  --
  -- the data_selector choose what data to pass to the LCD
  -- depending on the operation's state of the system
```

```
  --
  data_selector: process(istate, digit)
  begin
    -- the following section of the code is for the
    -- LCD's initialization process so it is always
    -- the same
    case istate is
      when istep_two | istep_four | istep_six =>
        byte <= X"30";
      when istep_eight =>
        byte <= X"20";
      when function_set =>
        byte <= X"28";
      when entry_mode =>
        byte <= X"06";
      when control_display =>
        byte <= X"0C";
      when clear_display =>
        byte <= X"01";
      when others =>
        byte <= (others => '0');
    end case;

    -- the following section of code
    -- needs to be modified depending on what
    -- the user want to display on the screen
    if istate = init_done then
      case dstate is
        when set_start_address =>
          byte <= X"80"; -- CG RAM
        when write_data_HT =>
          byte <= "0011" & digitHT;
        when write_data_TT =>
          byte <= "0011" & digitTT;
        when write_data_TH =>
          byte <= "0011" & digitTH;
        when write_data_H =>
          byte <= "0011" & digitH;
        when write_data_T =>
          byte <= "0011" & digitT;
        when write_data_U =>
          byte <= "0011" & digitU;

        when address_digit =>
          byte <= X"CF";
        when write_digit =>
          byte <= "0011" & digit;
        when others =>
          byte <= (others => '0');
      end case;
```

```
    end if;

  end process data_selector;

  --
  -- generate a 0 to 9 "running digit"
  -- the following block increments the digit once every sec.
  -- at the end of the trasmission of the address of the
  -- desired display location
  --
 digit_incr: process (dstate, txdone, digit, cnt)
  begin
    -- by defaukt hold
    next_digit <= digit; -- hold the value
    next_cnt   <= cnt;

    if (cnt = 50000000) then
      if (dstate = address_digit and txdone = '1') then
        if digit = X"9" then
          next_digit <= (others => '0');
        else
          next_digit <= digit + 1;
        end if;
        next_cnt <= 0;
     end if;
    else
      next_cnt <= cnt + 1;
    end if;

  end process digit_incr;


  --
  -- select what nibble goes to the LCD's
  -- data interface
  --
  nibble_select: process (selnibble, byte)
  begin
    case selnibble is
      when '0' => -- pass lower nibble
        nibble <= byte(3 downto 0);
      when '1' => -- pass upper nibble
        nibble <= byte(7 downto 4);
      when others => -- nothing to do
    end case;
  end process nibble_select;


  --
  -- After power-on, the display must be initialized to
```

```
  -- a) establish the required communication protocol, and
  -- b) configure the diplay operation
  --
  -- a) Configuration of the Fout-bit Interface Protocol
  -- The initialization sequence establishes that the FPGA
application wishes to use
  -- the four-bit data interface to the LCD as follows:
  --   s1. Wait 15ms or longer, although the display is generally
ready when the FPGA
  --       finishes configuration. The 15ms interval is 750,000
clock cycles at 50 MHz.
  --   s2. Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock
cycles.
  --   s3. Wait 4.1 ms or longer, which is 205,000 clock cycles at
50 MHz.
  --   s4. Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock
cycles.
  --   s5. Wait 100 ?s or longer, which is 5,000 clock cycles at
50 MHz.
  --   s6. Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock
cycles.
  --   s7. Wait 40 ?s or longer, which is 2,000 clock cycles at 50
MHz.
  --   s8. Write SF_D<11:8> = 0x2, pulse LCD_E High for 12 clock
cycles.
  --   s9. Wait 40 ?s or longer, which is 2,000 clock cycles at 50
MHz.
  --
  -- b) Display Configuration
  -- The four-bit interface is now established. The next part of
the sequence
  -- configures the display:
  --   s10. Issue a Function Set command, 0x28, to configure the
display for operation
  --        on the Spartan-3E Starter Kit board.
  --   s11. Issue an Entry Mode Set command, 0x06, to set the
display to automatically
  --        increment the address pointer.
  --   s12. Issue a Display On/Off command, 0x0C, to turn the
display on and disables
  --        the cursor and blinking.
  --   s13. Issue a Clear Display command, 0x01.
  --   s14. Allow at least 1.64 ms (82,000 clock cycles) after
issuing a clear
  --        display command.
  --

  --
  -- The design is "partitioned" into 4 blocks:
  -- 1) init_sm
```

```
  -- 2) time_m
  -- 3) display_sm
  -- 4) tx_m
  --

  --
  -- Once power comes on, the init_sm makes sure we go through all
the
  -- necessary steps of the LCD initialization process.
  -- The only purpose of the sm is to control that the system
evolve
  -- properly through the various initialization steps. Besides
this task
  -- the sm doesn't do much. The "real work" is done behind the
scenes
  -- by tx_m and time_m
  --

  init_sm: process (istate, idone, timer_15ms, timer_4100us,
timer_100us,
                    timer_40us, timer_1640us, txdone )
  begin
    -- default assignments
    next_istate     <= istate;
    next_idone      <= idone;

    case istate is

      when istep_one => -- wait here for 15 ms

        if (timer_15ms = '1') then
          next_istate    <= istep_two;
        end if;

      when istep_two => -- write nibble (0x3)

        if (txdone = '1') then
          next_istate <= istep_three;
        end if;

      when istep_three => -- wait here for 4100 us

        if (timer_4100us = '1') then
          next_istate    <= istep_four;
        end if;

      when istep_four => -- write nibble (0x3)

        if (txdone = '1') then
          next_istate  <= istep_five;
```

```
        end if;

      when istep_five => -- wait here for 100 us

        if (timer_100us = '1') then
          next_istate   <= istep_six;
        end if;

      when istep_six => -- write nibble (0x3)

        if (txdone = '1') then
          next_istate <= istep_seven;
        end if;

      when istep_seven => -- wait here for 40 us

        if (timer_40us = '1') then
          next_istate   <= istep_eight;
        end if;

      when istep_eight => -- write nibble (0x2)

        if (txdone = '1') then
          next_istate  <= istep_nine;
        end if;

      when istep_nine => -- wait here for 40 us

        if (timer_40us = '1') then
          next_istate   <= function_set;
        end if;

      when function_set => -- istep 10:
                           -- write data (0x28)

       if (txdone = '1') then
         next_istate <= entry_mode;
       end if;

      when entry_mode => -- istep 11
                         -- write data 0x06

        if (txdone = '1') then
          next_istate <= control_display;
        end if;

      when control_display => -- istep 12
                              -- enable display, disable cursor,
disable blinking
                              -- write data 0x0C
```

```
        if (txdone = '1') then
          next_istate <= clear_display;
        end if;

      when clear_display => -- istep 13
                            -- write data 0x01

        if (txdone = '1') then
          next_istate <= init_done; -- init. done
        end if;

      when init_done => -- istep 14

        -- the state machine will remain in init_done for good

        -- must wait 1.64 ms after issuing a clear display command
        if (timer_1640us = '1') then
          next_idone     <= '1';
        end if;

      when others => -- nothing to do

    end case;

  end process init_sm;


  --
  -- time_m provides all signals needed to correctly "time" the
various
  -- LCD's operations
  --

  time_m: process(istate, count, idone)
  begin
    --
    -- by default hold the state, keep the counter at rest, and
    -- hold the timer's outputs low
    --
    next_count   <= count;
    timer_15ms   <= '0'; -- combinational output
    timer_4100us <= '0'; -- combinational output
    timer_100us  <= '0'; -- combinational output
    timer_40us   <= '0'; -- combinational output
    timer_1640us <= '0'; -- combinational output

    case istate is
      when istep_one   =>
        next_count <= count + 1;
```

```
        if (count = 750000) then
          next_count  <= 0;
          timer_15ms  <= '1';
        end if;
      when istep_three =>
        next_count <= count + 1;
        if (count = 205000) then
          next_count   <= 0;
          timer_4100us <= '1';
        end if;
      when istep_five  =>
        next_count <= count + 1;
        if (count = 5000) then
          next_count  <= 0;
          timer_100us <= '1';
        end if;
      when istep_seven | istep_nine =>
        next_count <= count + 1;
        if (count = 2000) then
          next_count  <= 0;
          timer_40us  <= '1';
        end if;
      when init_done =>
        if (idone = '0') then
          next_count <= count + 1;
        end if;
        if (count = 82000) then
          next_count   <= 0;
          timer_1640us <= '1';
        end if;
      when others => -- nothing to do
    end case;

  end process time_m;


  --
  -- tx_m generate the control (LCD_E, LCD_RS, LCD_RW, SF_CE0) and
data (SF_D)
  -- signals needed to drive the LCD according to the 4-bit
interface protocol
  -- used by the Spartan 3E starter kit board.
  --

  tx_m: process(istate, txcount, byte, selnibble, enable, txdone,
                idone)
  begin
    next_selnibble <= selnibble;
    next_txdone    <= txdone;
    next_txcount   <= txcount;
```

```
    next_enable    <= enable;

    --
    -- the following section of the state machine allow
    -- to transmit the data necessary for the LCD's
    -- initialization  (which is pretty much the same
    -- no matter what the user what to write on the
    -- display), as well as transmitting the bytes
    -- needed to display the text the user want to put
    -- on the screen (which is user specific so it
    -- will require cutomization)
    --
    case istate is
      when istep_one | istep_three | istep_seven | istep_nine =>
        next_selnibble <= '1'; -- pass hign nibble
        -- transmit a nibble
      when istep_two | istep_four | istep_six | istep_eight =>
         next_txcount <= txcount + 1;
         if (txcount = 1) then
           next_enable <= '1';
         end if;
         if (txcount = 10) then
           next_enable <= '0';
           next_txdone <= '1';
         end if;
         if (txcount = 11) then
           next_txcount    <= 0;
           next_txdone     <= '0';
           --next we could pass zeros on the SF_D bus
         end if;
       -- transmit a byte
       when function_set | entry_mode | control_display |
            clear_display | init_done =>

         -- if we are in init_done the LCD's initialization
         -- phase is completed so we only need to transmit
         -- the bytes needed to display the text the user
         -- want to put on the screen
         --
         -- the following condition makes sure we can transmit
         -- the bytes needed for initializing the LCD as well
         -- as the bytes the user need to display the desired
         -- text on the LCD.
         -- The condition must be customized according
         -- to the user needs
         --

         if (istate /= init_done or
            (istate = init_done and
            (dstate = set_start_address or
```

```
              dstate = write_data_HT or
              dstate = write_data_TT or
              dstate = write_data_TH or
              dstate = write_data_H or
                           dstate = write_data_T or
                           dstate = write_data_U or
              dstate = address_digit or
              dstate = write_digit
             ))) then

            next_txcount <= txcount + 1;
            if (txcount = 1) then
              next_enable <= '1';
            end if;
            if (txcount = 10) then
              next_enable <= '0';
            end if;
            if (txcount = 11) then
              -- next we could pass zeros on the SF_D bus
            end if;
            -- the timing between the falling edge of the upper
nibble's enable
            -- and the rising edge of the lower nibble's rising
edge of an
            -- operation is 1us (50 clock cycles)
            if (txcount = 58) then    -- 10 + 1 + 50 - 2 = 58
              next_selnibble <= '0'; -- pass lower nibble
            end if;
            if (txcount = 60) then
              next_enable <= '1';
            end if;
            if (txcount = 69) then
              next_enable <= '0';
            end if;
            if(txcount = 70) then -- done with the lower nibble
data
              -- next we could pass zeros on the SF_D bus
            end if;
            -- the timing between the falling edge of the lower
nibble's enable
            -- and the rising edge of the upper nibble's rising
edge
            -- of two adjacent operations is 40us (2000 clock
cycles)
            if (txcount = 2067) then
              next_txdone <= '1';
            end if;
            if (txcount = 2068) then -- 69 + 1 + 2000 - 2 =
              next_txcount   <= 0;
              next_txdone    <= '0';
```

```
              next_selnibble <= '1'; -- pass upper nibble
            end if;
          end if;
        when others => --nothing to do
    end case;

  end process tx_m;


  --
  -- display state machine
  --
  -- Once the LCD has been properly initialized finally the
display sm kicks in and
  -- it makes possible to display characters on the screen.

  -- The character code to be displayed on the screen are first
stored in the Display
  -- Data RAM (DDRAM).
  -- There are 32 character locations on the display. The upper
line of characters
  -- is stored between addresses 0x00 and 0x0F of the DDRAM. The
second line of
  -- characters is stored between addresses 0x40 and 0x4F of the
DDRAM.
  -- The character code stored in a DDRAM location "references" a
specific character
  -- bitmap stored in the predefined Character Generator ROM
(CGROM) character set.
  -- The CGROM contains the font bitmap for each of the predefined
characters that
  -- the LCD screen can display.
  -- The sequence of commands to display characters to the screen
is as follows:
  -- 1. Set DDRAM Address command
  --    It sets the initial DDRAM address by initializing an
internal address counter.
  --    The DDRAM address counter either remains constant or auto-
increments or
  --    auto-decrements by one location, as defined by the I/D set
bit of the
  --    entry mode set command.
  -- 2. Write Data to DDRAM command
  --    Write data into DDRAM if the command follows a previous
Set DDRAM Address.
  -- 3. Return Cursor Home command
  --    Return the cursor to the home position, the top-left
corner. DDRAM contents are
  --    unaffected.
  --
```

```
  -- The specific sequence of commands executed in the state
machine depend on the text
  -- the user want to display on the LCD
  --
  -- the logic to activate the register select signal is placed in
this block
  --
  display_sm: process(dstate, txdone, idone, regsel, txcount)
  begin

    -- by default hold state
    next_dstate <= dstate;
    next_regsel <= regsel;

    if txcount = 11 then
      next_regsel <= '0';
    end if;
    if txcount = 58 then
      next_regsel <= idone; --high for active write dstates, low
for istates
    end if;
    if txcount = 70 then
      next_regsel <= '0';
    end if;

    case dstate is

      when didle =>
        next_regsel <= '0'; -- must be low for active istates
        if (idone = '1') then
          next_dstate  <= set_start_address;
          next_regsel  <= '0'; -- must be low for address commands
        end if;

      when set_start_address => -- start the text at the first
                                -- location of the first line
                                -- of the LCD (0x80)
        next_regsel <= '0';
        if (txdone = '1') then
          next_dstate <= write_data_HT;
          next_regsel <= '1'; --must be high for write commands
        end if;

      when write_data_HT => -- V = 0x56
        if (txdone = '1') then
          next_dstate <= write_data_TT;
          next_regsel <= '1';
        end if;

      when write_data_TT => -- H = 0x48
```

```
        if (txdone = '1') then
          next_dstate <= write_data_TH;
          next_regsel <= '1';
        end if;

      when write_data_TH => -- D = 0x44
        if (txdone = '1') then
          next_dstate <= write_data_H;
          next_regsel <= '1';
        end if;
      when write_data_H => -- D = 0x44
        if (txdone = '1') then
          next_dstate <= write_data_T;
          next_regsel <= '1';
        end if;
      when write_data_T => -- D = 0x44
        if (txdone = '1') then
          next_dstate <= write_data_U;
          next_regsel <= '1';
        end if;

      when write_data_U => -- L = 0x4C
        if (txdone = '1') then
          next_dstate <= address_digit;
          next_regsel <= '0';
        end if;

      when address_digit => -- 0x80
        next_regsel <= '0';
        if (txdone = '1') then
          next_dstate <= write_digit;
          next_regsel <= '1';
        end if;

      when write_digit => -- the digit running from 0 to 9
                          -- 0x30, 0x31, 0x32, 0x33, 0x34,
                          -- 0x35, 0x36, 0x37, 0x38, 0x39
        if (txdone = '1') then
          next_dstate <= address_digit; --return_home;
          next_regsel <= '0';
        end if;

      --when return_home =>
        -- by default the state machine will remain in the
        -- return_home state for good
        -- NOTE: this is not a "real" return_home command
        --       (see the Spartan 3E starter kit user guide).
        --       If we know that once we reach this state we
        --       are not sending anything new to the display
        --       we can simply stall the sm in this state
```

```
      when others => -- nothing to do;

    end case;

  end process display_sm;


  registers: process(rst, clk)
  begin
    if rst = '1' then
      istate    <= istep_one;
      dstate    <= didle;
      idone     <= '0';
      count     <=  0;
      txcount   <=  0;
      selnibble <= '1'; -- upper nibble
      enable    <= '0';
      txdone    <= '0';
      regsel    <= '0';
      digit     <= (others => '0');
      cnt       <= 0;
    elsif clk = '1' and clk'event then
      istate    <= next_istate;
      dstate    <= next_dstate;
      idone     <= next_idone;
      count     <= next_count;
      txcount   <= next_txcount;
      selnibble <= next_selnibble;
      enable    <= next_enable;
      txdone    <= next_txdone;
      regsel    <= next_regsel;
      digit     <= next_digit;
      cnt       <= next_cnt;
    end if;
  end process registers;

end Behavioral;
```

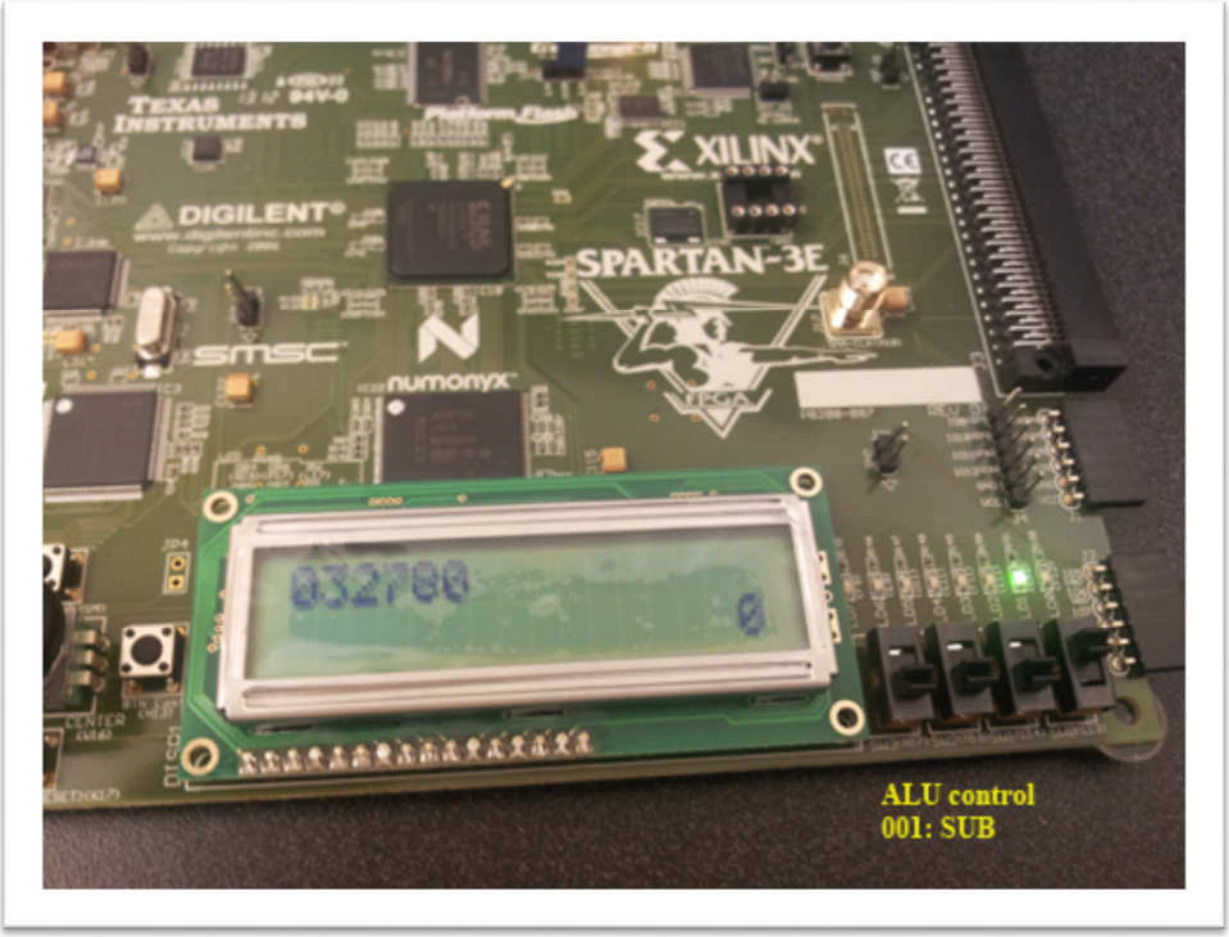
Display_controller.ucf

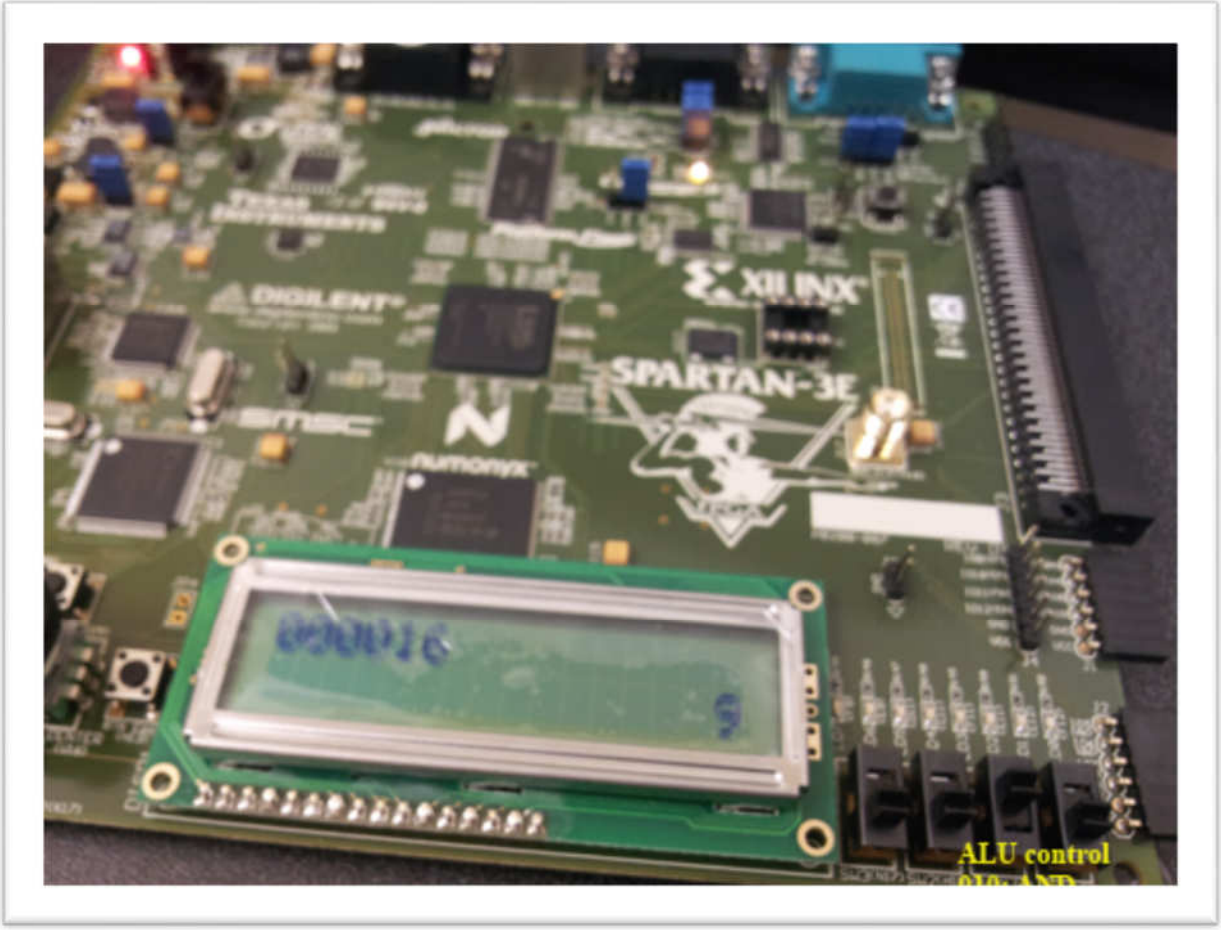
```
3. #PACE: Start of PACE I/O Pin Assignments
4. # For device: Spartan-3E Start Kit
5.
6. NET "rst" LOC = "V16" | IOSTANDARD = LVCMOS33 | PULLDOWN ;
7.
8. #Reset and Clock
```

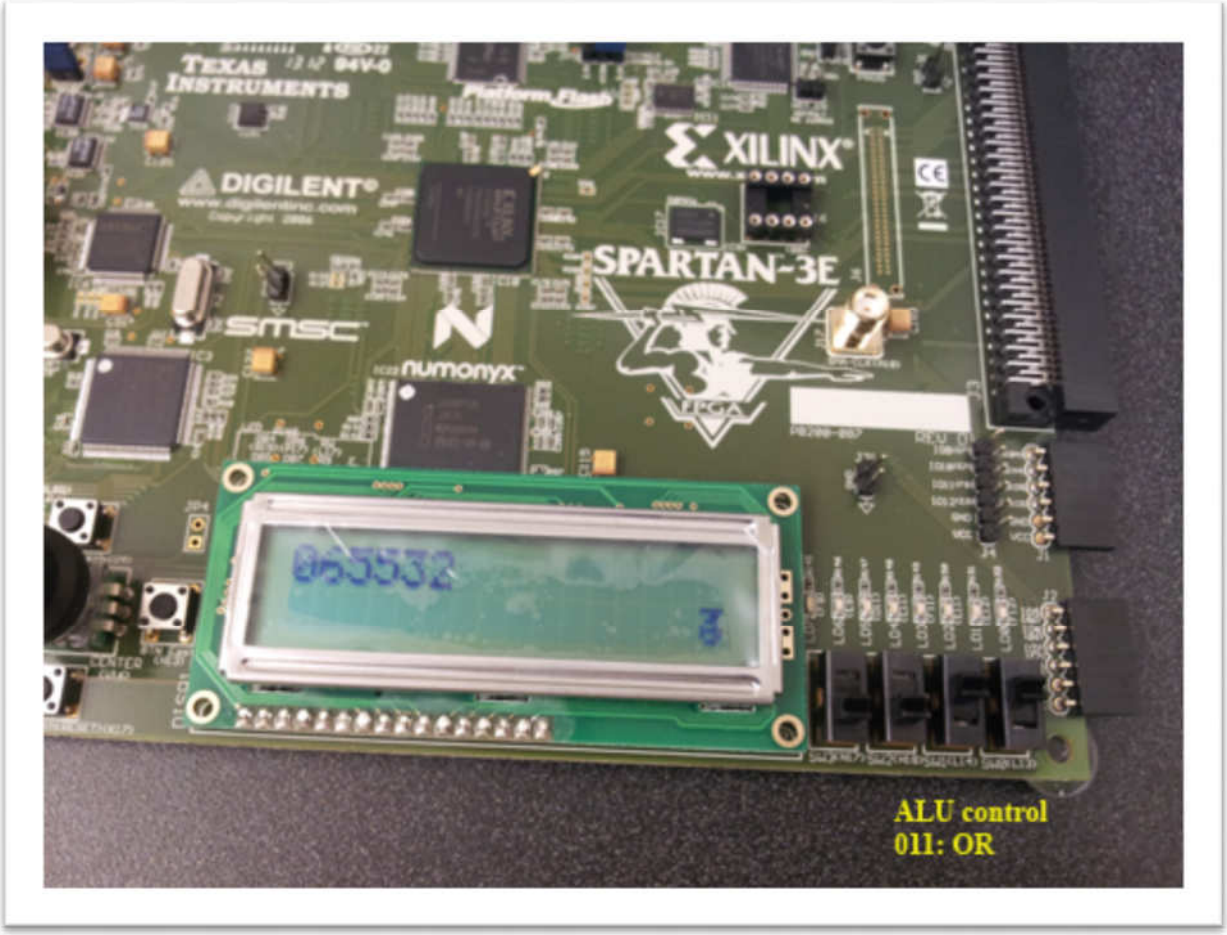
```
NET "clk"     LOC = "C9" | IOSTANDARD = LVCMOS33;

NET "LCD_E" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
NET "LCD_RS" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
NET "LCD_RW" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;

# The LCD four-bit data interface is shared with the StrataFlash.
NET "SF_D[8]" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
NET "SF_D[9]" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
NET "SF_D[10]" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
NET "SF_D[11]" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;

#ALU control
NET "t_ALUcontrol[0]" LOC = "L13" | IOSTANDARD = LVCMOS33 | PULLUP;
NET "t_ALUcontrol[1]" LOC = "L14" | IOSTANDARD = LVCMOS33 | PULLUP;
NET "t_ALUcontrol[2]" LOC = "H18" | IOSTANDARD = LVCMOS33 | PULLUP;

# LED
NET "t_zero" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
NET "t_Overflow" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;
```

b. Results:

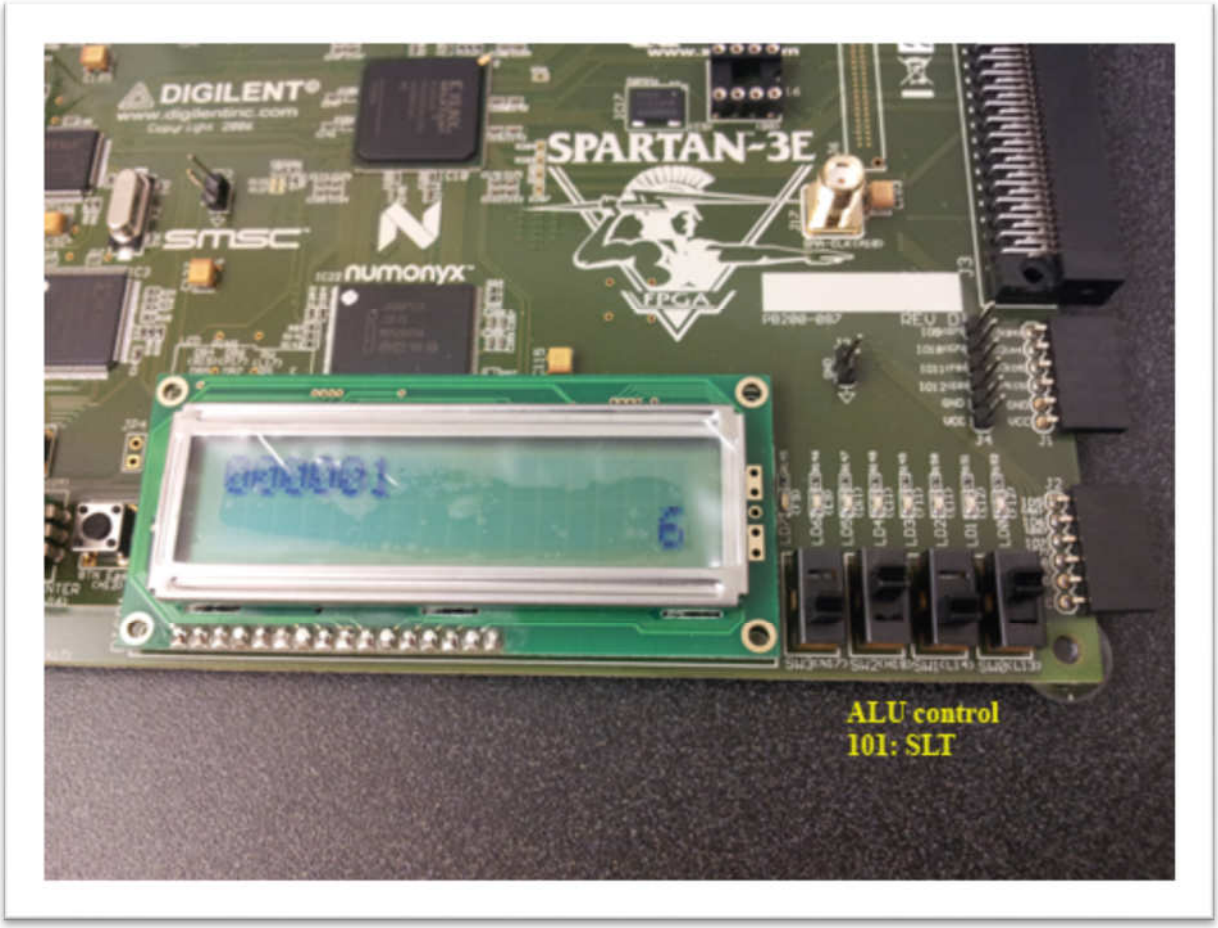

TEXAS INSTRUMENTS
XILINX
DIGILENT
SPARTAN-3E
smsc
numonyx
065548
3
ALU control
000: ADD


TEXAS INSTRUMENTS
XILINX
DIGILENT
SPARTAN-3E
smsc
numonyx
032780
ALU control
001: SUB


XILINX
SPARTAN-3E
DIGILENT
numonyx
ALU control


TEXAS INSTRUMENTS
DIGILENT
XILINX
SPARTAN-3E
smsc
numonyx
ALU control
011: OR


ALU control
100: NOR


DIGILENT
SPARTAN-3E
smsc
numonyx
ALU control
101: SLT


XILINX
DIGILENT
www.digilentinc.com
SPARTAN-3E
smsc
numonyx
FPGA
ALU control
110: SLL

End of document