Tutorial 7
Using the Spartan-3E Starter Board LCD Display with a Counter

-Introduction

The LCD (Liquid Crystal Display) in question is included with the Spartan-3E Starter Board Kit sold by both Digilent. LCD's in general offer a cheap and convenient way to deliver information from electronic devices. Indeed, it is that very convenience that has led to the LCD's near ubiquity in today's electronic world.

Information relevant to the LCD's operation is in the previous tutorial, where the LCD was used to display "FPGA". Other than the alteration of the main state machine, this lab is a near duplicate of the previous.

Writing numbers on the LCD is surprisingly simple. The chart in the previous lab or manual specifies that for the numbers 0-9 the upper data nibble is always 3. Furthermore, the lower data nibble corresponds exactly to the number. Thus, the number 8, is "0011" concatenated with "1000", or 0x38.

-Objective

To use the S3E Starter Board's LCD display to display the consecutive numbers of a counter up to 9, and learn more about digital logic design in the process.

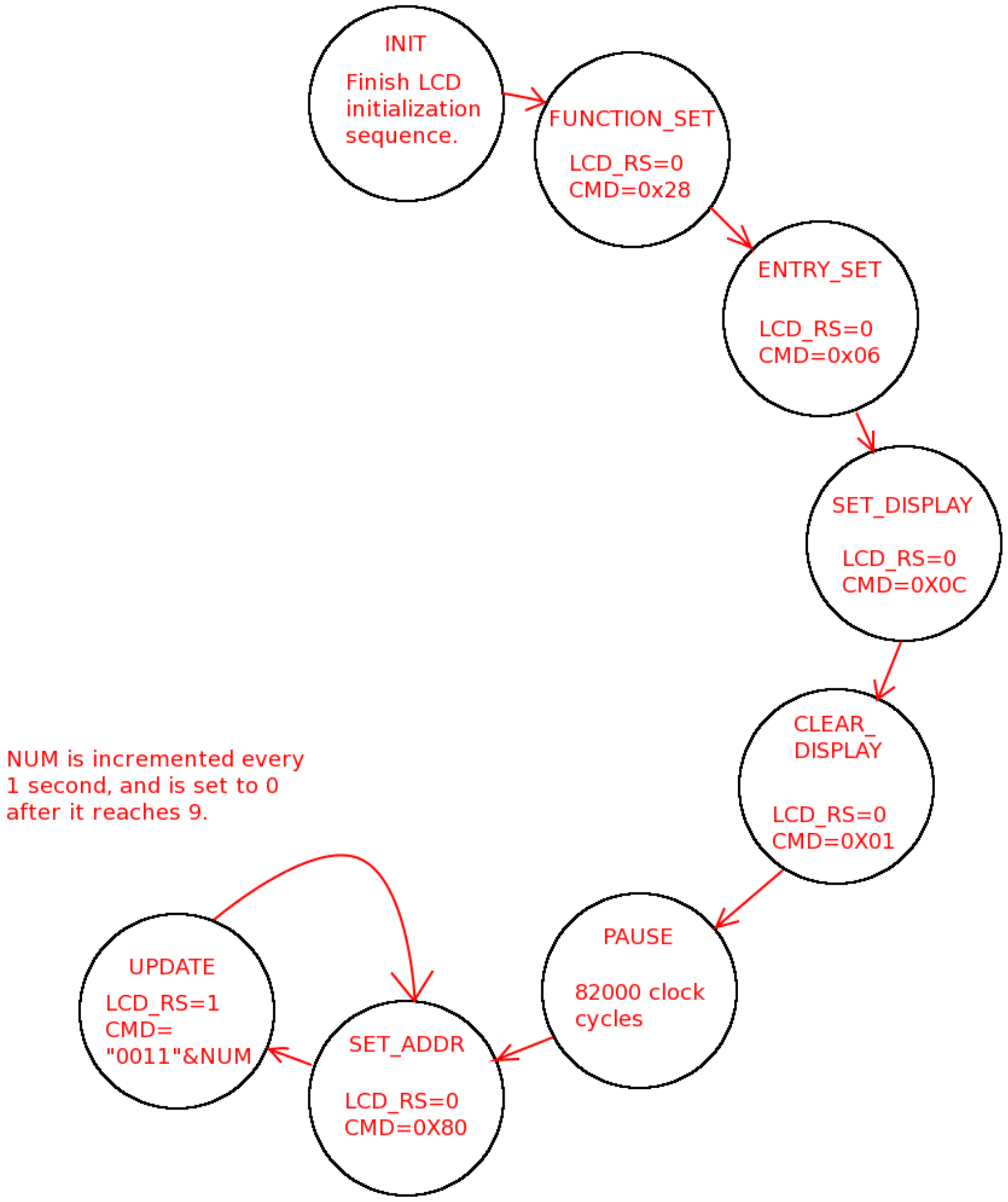
-Process

1. Implement hardware to control the LCD.
2. Verify the hardware in software (Modelsim).
3. Program the S3E Starter Kit Board.

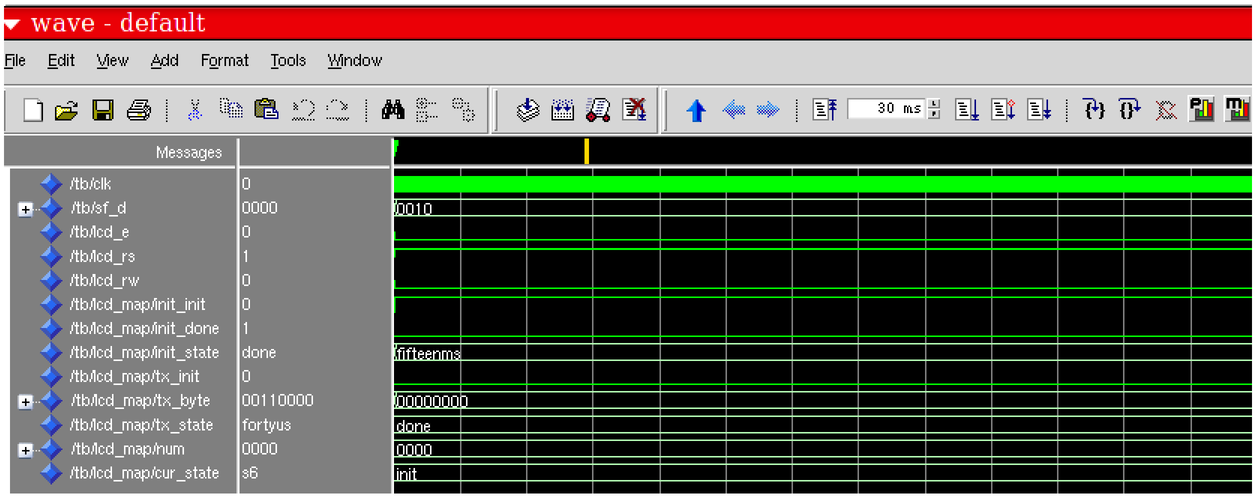
-Implementation

Again, this project requires 3 state machines. One for the power on initialization sequence, one to transmit commands and data to the LCD and lastly, one to start the power on initialization sequence, then configure and write to the LCD. In this case, the main state machine that controls the others will never end. Instead it will loop forever, setting the DD-RAM address to 0x00, and then writing the character that corresponds to the number of a counter.

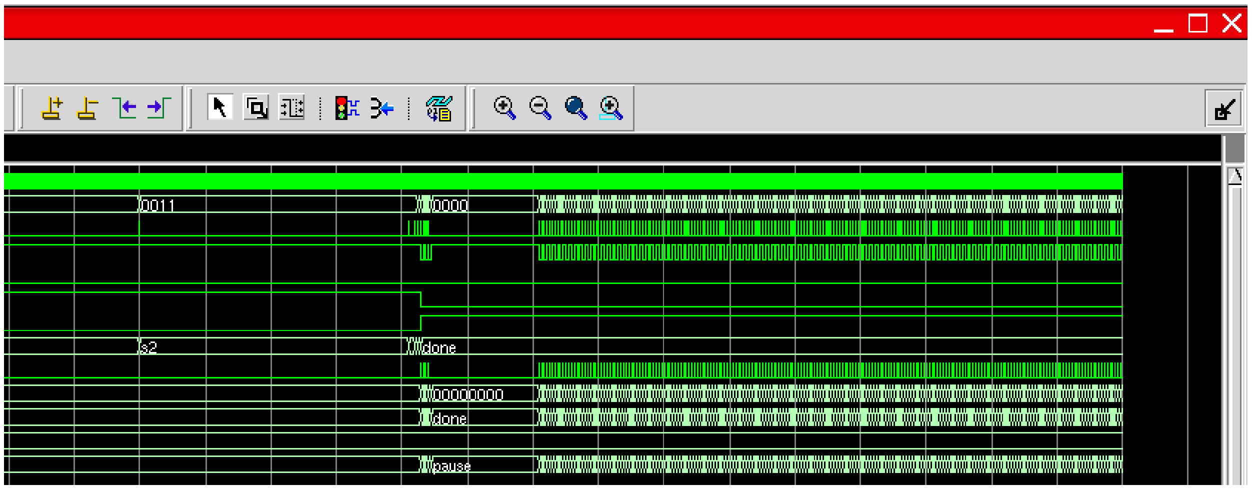
Here, the main state machine initializes the display as detailed previously, but then deviates by continuously setting the address to 0x00 and writing a character updated by a counter implemented as a separate process.



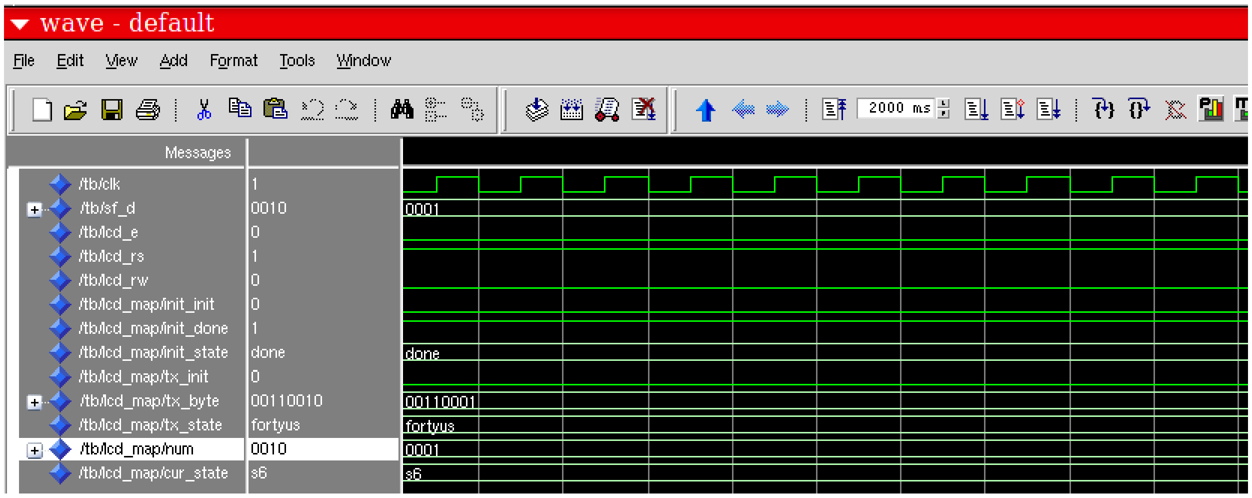
This is a 30ms simulation of properly working LCD-Counter hardware. Notice how although the number being displayed doesn't change, the main state machine continues to loop and set the address to 0x00 and rewrite the character.



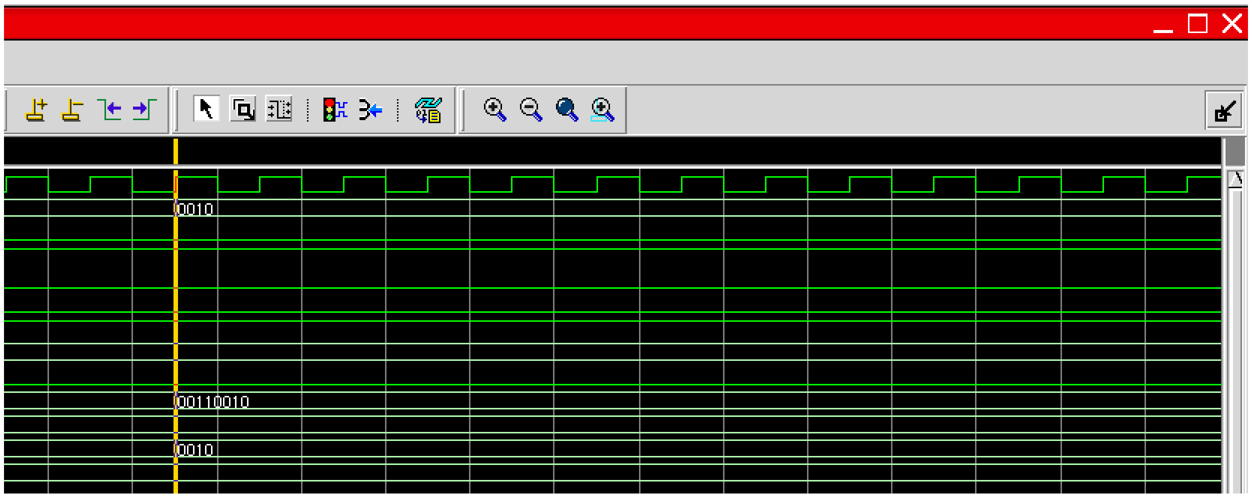
```
/tb/clk                   0
/tb/sf_d                  0000        0010
/tb/lcd_e                 0
/tb/lcd_rs                1
/tb/lcd_rw                0
/tb/lcd_map/init_init     0
/tb/lcd_map/init_done     1
/tb/lcd_map/init_state    done        fifteenms
/tb/lcd_map/tx_init       0
/tb/lcd_map/tx_byte       00110000    00000000
/tb/lcd_map/tx_state      fortyus     done
/tb/lcd_map/num           0000        0000
/tb/lcd_map/cur_state     s6          init
```



This simulation shows an actual change in the number being displayed.



Messages		
/tb/clk	1	
/tb/sf_d	0010	0001
/tb/lcd_e	0	
/tb/lcd_rs	1	
/tb/lcd_rw	0	
/tb/lcd_map/init_init	0	
/tb/lcd_map/init_done	1	
/tb/lcd_map/init_state	done	done
/tb/lcd_map/tx_init	0	
/tb/lcd_map/tx_byte	00110010	00110001
/tb/lcd_map/tx_state	fortyus	fortyus
/tb/lcd_map/num	0010	0001
/tb/lcd_map/cur_state	s6	s6



5

```
--Written by Rahul Vora
--for the University of New Mexico
--rhlvora@gmail.com

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

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComps1nts.all;

entity lcd is
	port(
	clk, reset : in std_logic;
	SF_D : out std_logic_vector(3 downto 0);
	LCD_E, LCD_RS, LCD_RW, SF_CE0 : out std_logic;
	LED : out std_logic_vector(7 downto 0)
);
end lcd;

architecture behavior of lcd is

type display_state is (init, function_set, s1, entry_set, s2, set_display, s3, clr_display, s4, pause, set_addr, s5, update, s6,
done);
signal cur_state : display_state := init;

signal SF_D0, SF_D1 : std_logic_vector(3 downto 0);
signal LCD_E0, LCD_E1 : std_logic;
signal mux : std_logic;

type tx_sequence is (high_setup, high_hold, oneus, low_setup, low_hold, fortyus, done);
signal tx_state : tx_sequence := done;
signal tx_byte : std_logic_vector(7 downto 0);
signal tx_init : std_logic := '0';
signal tx_rdy : std_logic := '0';

type init_sequence is (idle, fifteenms, s1, s2, s3, s4, s5, s6, s7, s8, done);
signal init_state : init_sequence := idle;
signal init_init, init_done : std_logic := '0';

signal i : integer range 0 to 750000 := 0;
signal i2 : integer range 0 to 2000 := 0;
signal i3 : integer range 0 to 82000 := 0;
signal i4 : integer range 0 to 50000000 := 0;

signal num : std_logic_vector(3 downto 0);

begin
	LED <= tx_byte; --for diagnostic purposes

	SF_CE0 <= '1'; --disable intel strataflash
	LCD_RW <= '0'; --write only

	--when to transmit a command/data and when not to
	with cur_state select
		tx_init <= '1' when function_set | entry_set | set_display | clr_display | set_addr | update,
			'0' when others;

	--control the bus
	with cur_state select
```

```
          mux <= '1' when init,
                    '0' when others;

--control the initialization sequence
with cur_state select
          init_init <= '1' when init,
                    '0' when others;

--register select
with cur_state select
          LCD_RS <= '0' when s1|s2|s3|s4|s5,
                    '1' when others;

with cur_state select
          tx_byte <= "00101000" when s1,
                    "00000110" when s2,
                    "00001100" when s3,
                    "00000001" when s4,
                    "10000000" when s5,
                    "0011"&num when s6,
                    "00000000" when others;

counter: process(clk, reset)
begin
          if(reset = '1') then
                    i4 <= 0;
                    num <= "0000";
          elsif(clk='1' and clk'event) then
                    if(i4 = 50000000) then
                              i4 <= 0;
                              if(num = "1001") then
                                        num <= "0000";
                              else
                                        num <= num + '1';
                              end if;
                    else
                              i4 <= i4 + 1;
                    end if;
          end if;
end process counter;

--main state machine
display: process(clk, reset)
begin
          if(reset='1') then
                    cur_state <= init;
          elsif(clk='1' and clk'event) then
                    case cur_state is
                              when init =>
                                        if(init_done = '1') then
                                                  cur_state <= function_set;
                                        else
                                                  cur_state <= init;
                                        end if;

                              when function_set =>
                                        cur_state <= s1;

                              when s1 =>
                                        if(tx_rdy = '1') then
                                                  cur_state <= entry_set;
                                        else
                                                  cur_state <= s1;
                                        end if;

                              when entry_set =>
```

```
                                        cur_state <= s2;

                                when s2 =>
                                        if(tx_rdy = '1') then
                                                cur_state <= set_display;
                                        else
                                                cur_state <= s2;
                                        end if;

                                when set_display =>
                                        cur_state <= s3;

                                when s3 =>
                                        if(tx_rdy = '1') then
                                                cur_state <= clr_display;
                                        else
                                                cur_state <= s3;
                                        end if;

                                when clr_display =>
                                        cur_state <= s4;

                                when s4 =>
                                        i3 <= 0;
                                        if(tx_rdy = '1') then
                                                cur_state <= pause;
                                        else
                                                cur_state <= s4;
                                        end if;

                                when pause =>
                                        if(i3 = 82000) then
                                                cur_state <= set_addr;
                                                i3 <= 0;
                                        else
                                                cur_state <= pause;
                                                i3 <= i3 + 1;
                                        end if;

                                when set_addr =>
                                        cur_state <= s5;

                                when s5 =>
                                        if(tx_rdy = '1') then
                                                cur_state <= update;
                                        else
                                                cur_state <= s5;
                                        end if;

                                when update =>
                                        cur_state <= s6;

                                when s6 =>
                                        if(tx_rdy = '1') then
                                                cur_state <= set_addr;
                                        else
                                                cur_state <= s6;
                                        end if;

                                when done =>
                                        cur_state <= done;

                        end case;
                end if;
        end process display;
```

```
with mux select
          SF_D <= SF_D0 when '0', --transmit
                    SF_D1 when others;          --initialize
with mux select
          LCD_E <= LCD_E0 when '0', --transmit
                    LCD_E1 when others; --initialize

with tx_state select
          tx_rdy <= '1' when done,
                    '0' when others;

with tx_state select
          LCD_E0 <= '0' when high_setup | oneus | low_setup | fortyus | done,
                    '1' when high_hold | low_hold;

with tx_state select
          SF_D0 <= tx_byte(7 downto 4) when high_setup | high_hold | oneus,
                    tx_byte(3 downto 0) when low_setup | low_hold | fortyus | done;


--specified by datasheet
transmit : process(clk, reset, tx_init)
begin
          if(reset='1') then
                    tx_state <= done;
          elsif(clk='1' and clk'event) then
                    case tx_state is
                              when high_setup => --40ns
                                        if(i2 = 2) then
                                                  tx_state <= high_hold;
                                                  i2 <= 0;
                                        else
                                                  tx_state <= high_setup;
                                                  i2 <= i2 + 1;
                                        end if;

                              when high_hold => --230ns
                                        if(i2 = 12) then
                                                  tx_state <= oneus;
                                                  i2 <= 0;
                                        else
                                                  tx_state <= high_hold;
                                                  i2 <= i2 + 1;
                                        end if;

                              when oneus =>
                                        if(i2 = 50) then
                                                  tx_state <= low_setup;
                                                  i2 <= 0;
                                        else
                                                  tx_state <= oneus;
                                                  i2 <= i2 + 1;
                                        end if;

                              when low_setup =>
                                        if(i2 = 2) then
                                                  tx_state <= low_hold;
                                                  i2 <= 0;
                                        else
                                                  tx_state <= low_setup;
                                                  i2 <= i2 + 1;
                                        end if;

                              when low_hold =>
                                        if(i2 = 12) then
                                                  tx_state <= fortyus;
                                        i2 <= 0;
                                else
                                        tx_state <= low_hold;
                                        i2 <= i2 + 1;
                                end if;

                        when fortyus =>
                                if(i2 = 2000) then
                                        tx_state <= done;
                                        i2 <= 0;
                                else
                                        tx_state <= fortyus;
                                        i2 <= i2 + 1;
                                end if;

                        when done =>
                                if(tx_init = '1') then
                                        tx_state <= high_setup;
                                        i2 <= 0;
                                else
                                        tx_state <= done;
                                        i2 <= 0;
                                end if;

                end case;
        end if;
end process transmit;

with init_state select
        init_done <= '1' when done,
                '0' when others;

with init_state select
        SF_D1 <= "0011" when s1 | s2 | s3 | s4 | s5 | s6,
                "0010" when others;

with init_state select
        LCD_E1 <= '1' when s1 | s3 | s5 | s7,
                '0' when others;

--specified by datasheet
power_on_initialize: process(clk, reset, init_init) --power on initialization sequence
begin
        if(reset='1') then
                init_state <= idle;
        elsif(clk='1' and clk'event) then
                case init_state is
                        when idle =>
                                if(init_init = '1') then
                                        init_state <= fifteenms;
                                        i <= 0;
                                else
                                        init_state <= idle;
                                        i <= i + 1;
                                end if;

                        when fifteenms =>
                                if(i = 750000) then
                                        init_state <= s1;
                                        i <= 0;
                                else
                                        init_state <= fifteenms;
                                        i <= i + 1;
                                end if;

                        when s1 =>
                if(i = 11) then
                        init_state<=s2;
                        i <= 0;
                else
                        init_state<=s1;
                        i <= i + 1;
                end if;

        when s2 =>
                if(i = 205000) then
                        init_state<=s3;
                        i <= 0;
                else
                        init_state<=s2;
                        i <= i + 1;
                end if;

        when s3 =>
                if(i = 11) then
                        init_state<=s4;
                        i <= 0;
                else
                        init_state<=s3;
                        i <= i + 1;
                end if;

        when s4 =>
                if(i = 5000) then
                        init_state<=s5;
                        i <= 0;
                else
                        init_state<=s4;
                        i <= i + 1;
                end if;

        when s5 =>
                if(i = 11) then
                        init_state<=s6;
                        i <= 0;
                else
                        init_state<=s5;
                        i <= i + 1;
                end if;

        when s6 =>
                if(i = 2000) then
                        init_state<=s7;
                        i <= 0;
                else
                        init_state<=s6;
                        i <= i + 1;
                end if;

        when s7 =>
                if(i = 11) then
                        init_state<=s8;
                        i <= 0;
                else
                        init_state<=s7;
                        i <= i + 1;
                end if;

        when s8 =>
                if(i = 2000) then
                        init_state<=done;
                        i <= 0;
```

```
                                        else
                                                init_state<=s8;
                                                i <= i + 1;
                                        end if;

                                when done =>
                                        init_state <= done;

                        end case;

                end if;
        end process power_on_initialize;

end behavior;
```

This tutorial was written by Rahul Vora. Rahul is an engineering student in the department of Electrical and Computer Engineering at the University of New Mexico. He can be reached at rvora@unm.edu.