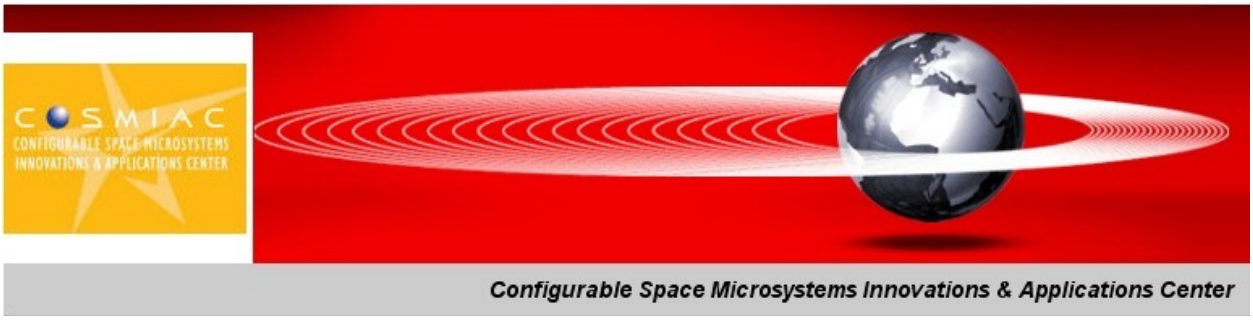


Tutorial 6

Using the Spartan-3E Starter Board LCD Display

With ISE 10.1

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.

A detailed description of the operation of an LCD is far beyond the scope of this document; however, it is yet worthwhile to understand the general operation of an LCD device. Essentially, what a traditional black & white LCD does is to selectively toggle what are termed pixels in the display to allow or block light from passing. These pixels are abstractly considered to be the points that compose the display; of course, in our particular case the pixels themselves are clearly visible and are perfect squares. In any case, to generate something like the letter 'I', the LCD would merely toggle the pixels in a straight vertical line.

This lab will assume the completion or understanding of the material in labs 1, 2, 3 and 5. There are two fundamental sections for this lab. These consist of:

1. The implementation of hardware to control the onboard LCD.
2. A simulation of the hardware to verify functionality.

The enterprising reader may idly consider the fact that it is easier to simply program the board than verify the hardware design in software first. Unfortunately, that is only true of very small projects, as projects become larger, so do the problems with no seeming source or solution. Repeatedly programming the board in trial and error fashion will only waste time.

LCD Controller: Sitronix ST7066U

Relevant Pins

There are two possible interfaces to the LCD controller, one 8 bits wide and another 4 bits wide. The designers of the Spartan-3E chose to use the four bit interface and share it with the onboard Intel StrataFlash storage device to minimize pin count. This will slightly complicate the procedure of initializing and writing to the display. The following pin definitions will be used in the "ucf" constraints file (manual pg. 43). LCD_RW will be pulled low, as this application will not be reading data from the display. Driving LCD_E low causes the LCD to ignore all inputs. A high LCD_RS specifies a data write operation, whereas a low LCD_RS specifies a command.

```
NET "LCD_E" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "LCD_RS" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "LCD_RW" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
# The LCD four-bit data interface is shared with the StrataFlash.
NET "SF_D<8>" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<9>" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<10>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<11>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
```

LCD Memory Management

The LCD device has three internal regions of memory. The Data Display RAM (DD RAM), which references the data to be displayed on the screen, the Character Generator RAM (CG RAM), which stores user-defined patterns and the Character Generator ROM (CG ROM), which includes a number of predefined patterns that correspond to ASCII symbols. In this tutorial only the DD-RAM and the CG-ROM will be used. To reference a value in the CG-ROM, the value in the figure 1 needs to be written into the DD-RAM. For example, the character 'S' from the CG-ROM would have the value "01010011" (manual pg. 45). At least one *Set DD-RAM Address* command should precede either one or many *Data Write* operations.

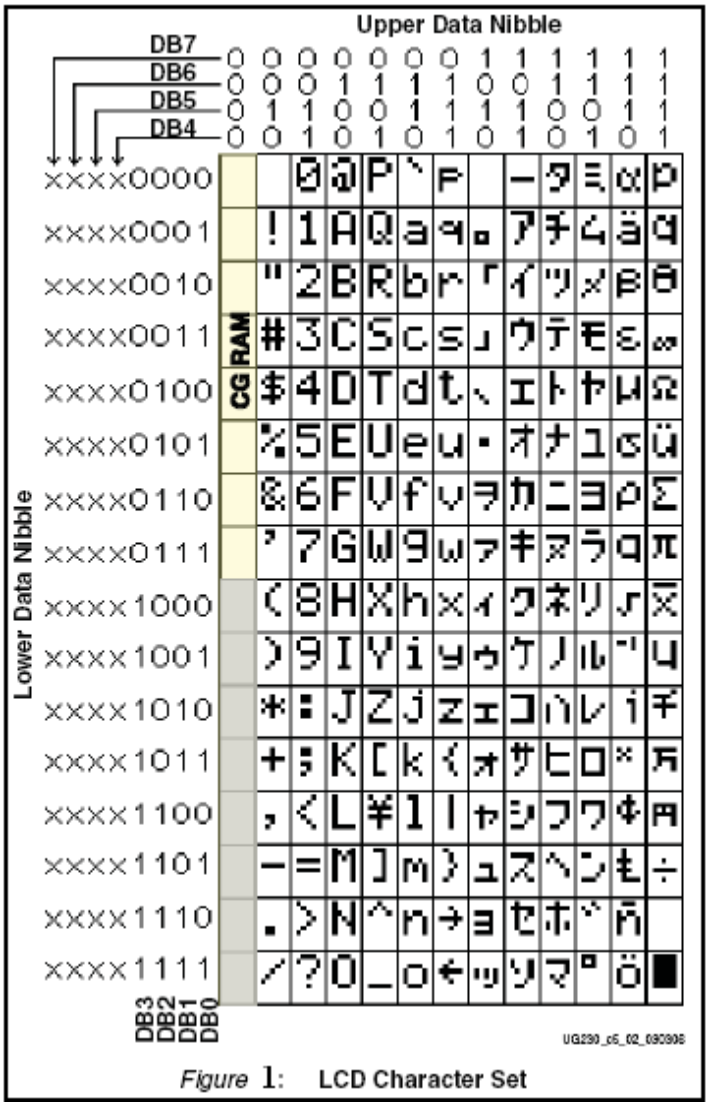


Figure 1: **LCD Character Set**

In addition, the following addresses are locations in the DD-RAM that physically correspond to the locations of characters displayed on the LCD.

	Character Display Addresses																Undisplayed Addresses		
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	…	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	…	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	…	40

Figure 5-3: **DD RAM Hexadecimal Addresses (No Display Shifting)**

LCD Command Format

Each 8 bit command to the LCD controller occurs over a 4 bit interface, thus, each command is decomposed into two four bit transmissions spaced by 1us. Subsequent commands (each sequential 4 bit transmission) must be spaced from the next by at least 40us. A detailed description follows (manual pg. 50).

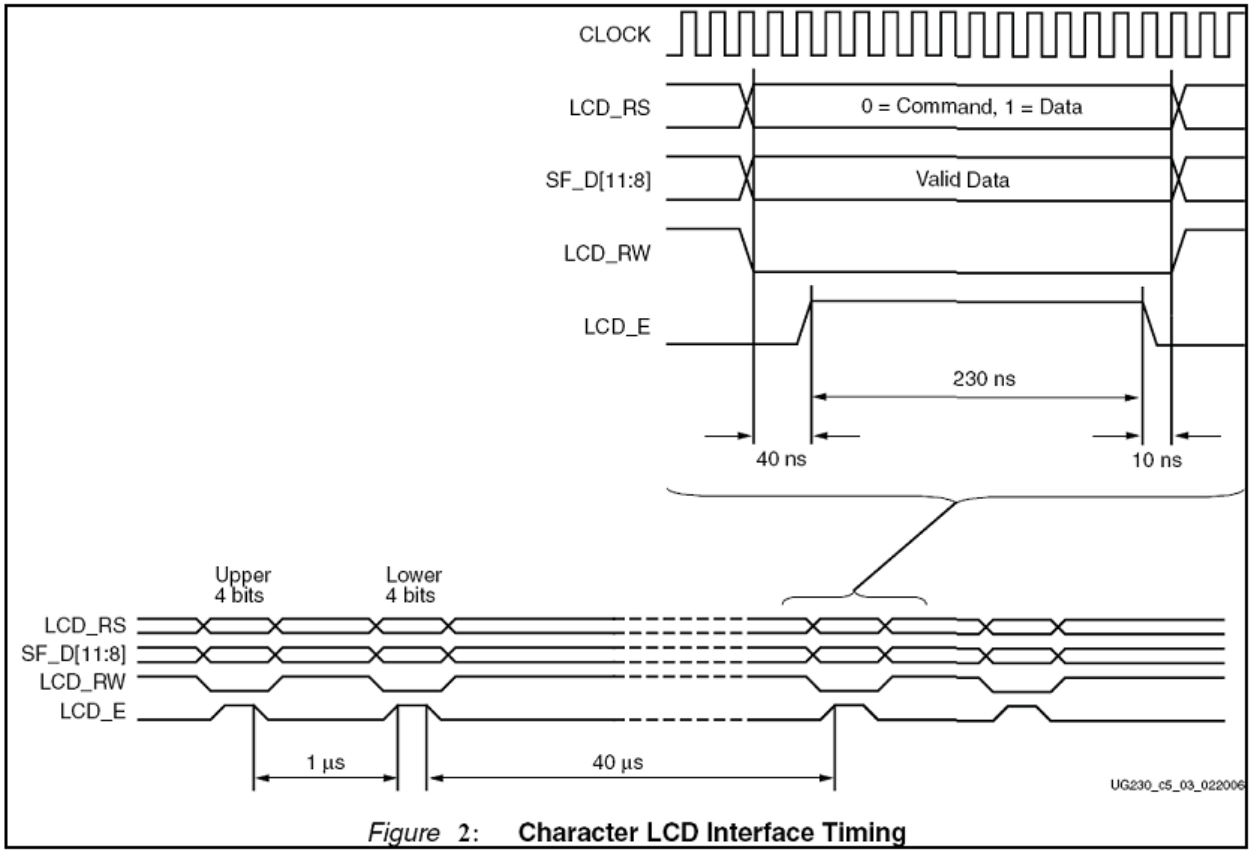


Figure 2: **Character LCD Interface Timing**

Note that the period of the 50MHz onboard clock is 20ns. The time between corresponding nibbles is 1us, which is equivalent to 50 clock cycles. The time between successive commands is 40us, which corresponds to 2000 clock cycles. The delay after a *Clear Display* command is 1.64ms, rather than the usual 40us and corresponds to 82000 clock cycles. Setup time (time for the outputs to stabilize) is 40ns, which is 2 clock cycles, the hold time (time to assert the LCD_E pin) is 230ns, which translates to roughly 12 clock cycles, and the fall time (time to allow the outputs to stabilize) is 10ns, which translates to roughly 1 clock cycle.

The following is the LCD command set.

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Function Set	0	0	0	0	1	0	1	0	-	-
Set CG RAM Address	0	0	0	1	A5	A4	A3	A2	A1	A0
Set DD RAM Address	0	0	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Address	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG RAM or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Read Data from CG RAM or DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Cursor Home	0	0	0	0	0	0	0	0	1	-
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display On/Off	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	-	-

LCD Initialization, Configuration and Display

There are three main steps in using the display, the first being the initialization of the four bit interface itself, the second being the commands to set the display options and the third being the writing of character data (manual pg. 51).

Initialization

1. Wait 15 ms or longer, although the display is generally ready when the FPGA finishes configuration. The 15 ms interval is 750,000 clock cycles at 50 MHz.
2. Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
3. Wait 4.1 ms or longer, which is 205,000 clock cycles at 50 MHz.
4. Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
5. Wait 100 µs or longer, which is 5,000 clock cycles at 50 MHz.
6. Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
7. Wait 40 µs or longer, which is 2,000 clock cycles at 50 MHz.
8. Write SF_D<11:8> = 0x2, pulse LCD_E High for 12 clock cycles.
9. Wait 40 µs or longer, which is 2,000 clock cycles at 50 MHz.

The second step involves the configuration and actual writing to the LCD ram. To configure the LCD, the following commands will be given (manual pg. 51).

Configuration

1. Issue a *Function Set* command, 0x28, to configure the display for operation on the Spartan-3E Starter Kit board.
2. Issue an *Entry Mode Set* command, 0x06, to set the display to automatically increment the address pointer.
3. Issue a *Display On/Off* command, 0x0C, to turn the display on and disables the cursor and blinking.
4. Finally, issue a *Clear Display* command. Allow at least 1.64 ms (82,000 clock cycles) after issuing this command.

The third and last step involves the actual process of writing data to the DD-RAM.

Display

1. Specify the start address with a *Set DD-RAM Address* command.
2. Display a character with a *Write Data* command.

Objective

To use the S3E Starter Board's LCD to display "FPGA", and learn more about digital logic design in the process.

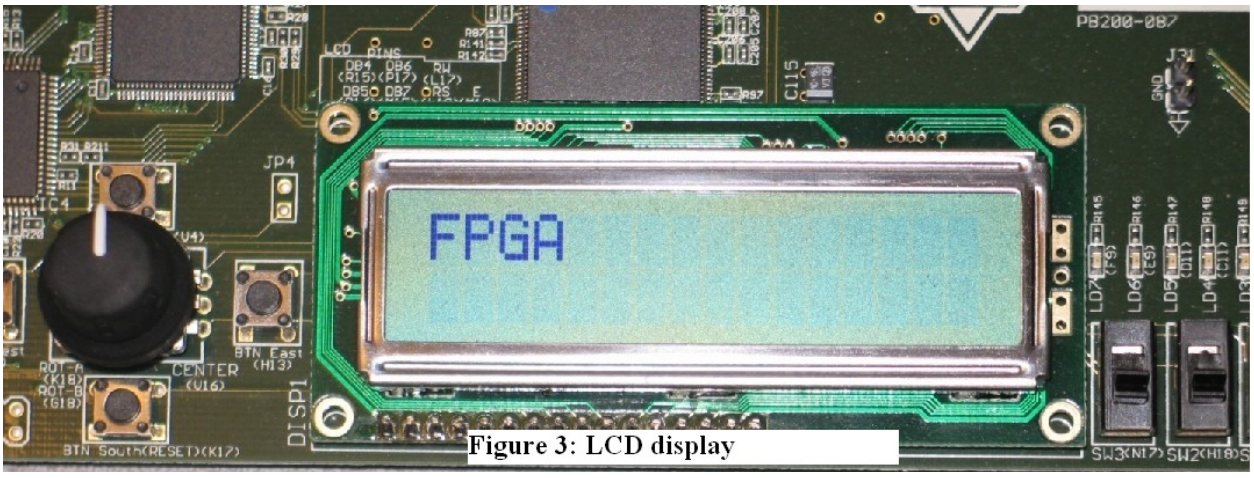


Figure 3: LCD display

Process

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

Implementation

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.

Note that one will have to make sure that the state machines are synchronized properly, that is, that main state machine remains in the “INIT” state until the initialization state machine is in the “DONE” state, and that subsequent command states not change until the data is fully transferred by the transmission state machine.
Figure 4 shows the main state machine that controls the initialization sequence and the transmission state machines. Note that each and every command implicitly requires the transmission state machine.

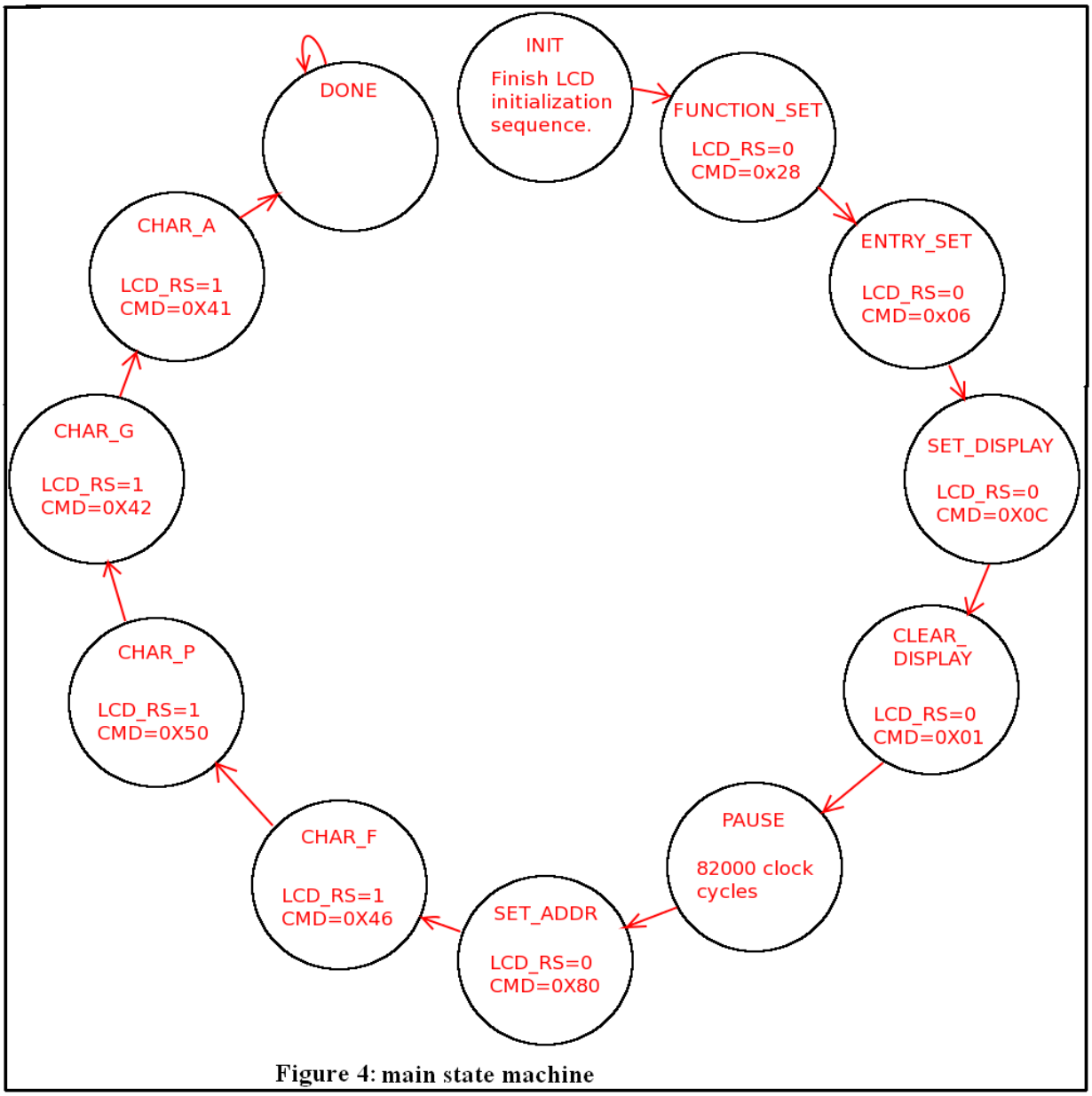


Figure 4: main state machine

Figure 5 is the initialization sequence state machine. It is only activated when the main state machine asserts “init”. Make certain that there is a way to notify the main state machine that the initialization is over and the four bit interface is established.

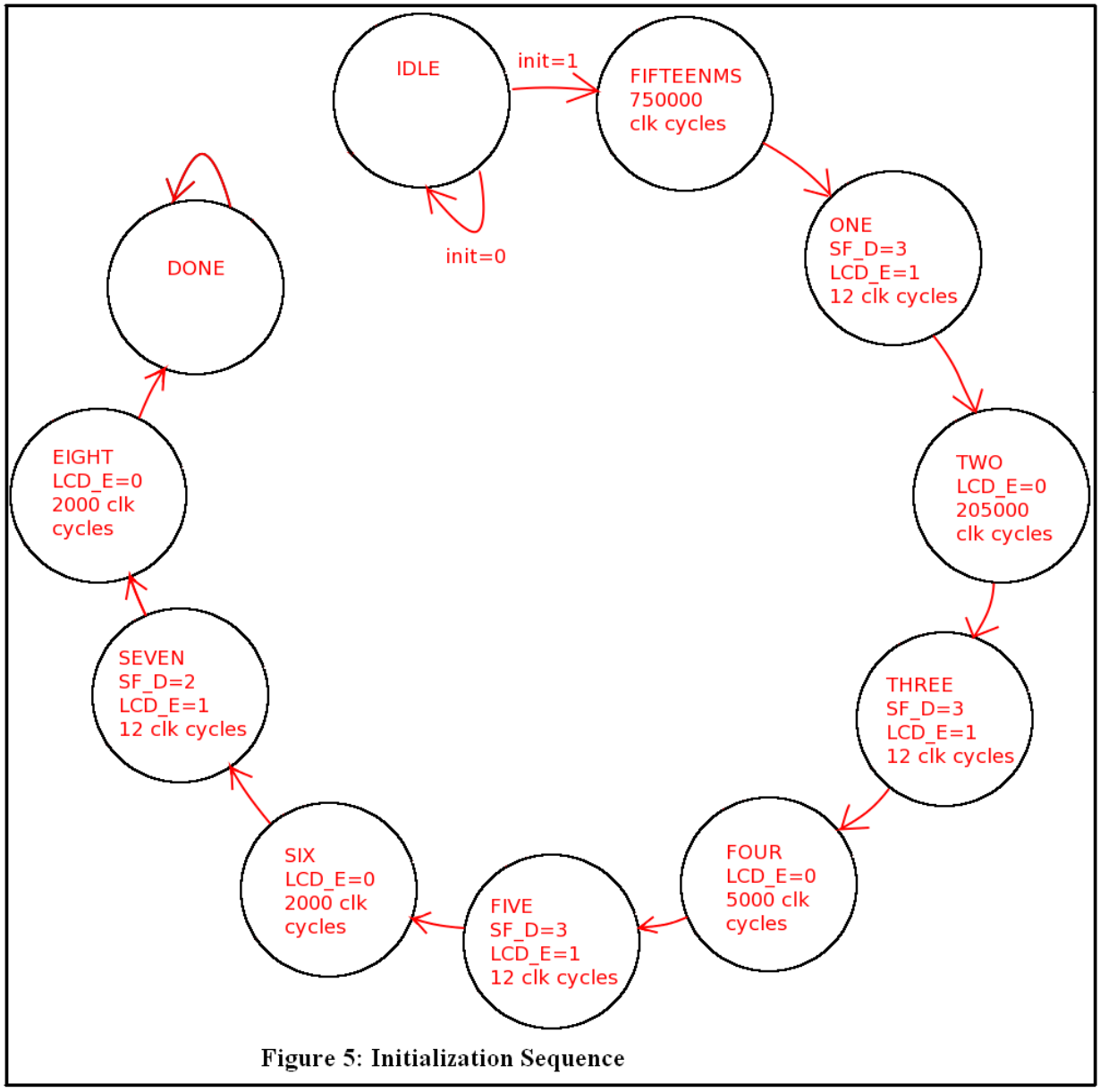


Figure 5: Initialization Sequence

This is the last state machine, and potentially the most troublesome. The main state machine will set the values of LCD_RS and the values of the high and low nibbles of an LCD command. This state machine will simply verify the strict timing constraints given above for an LCD command, otherwise it would be totally unnecessary. Again, a command will only be transferred once tx_init is asserted by the main state machine.

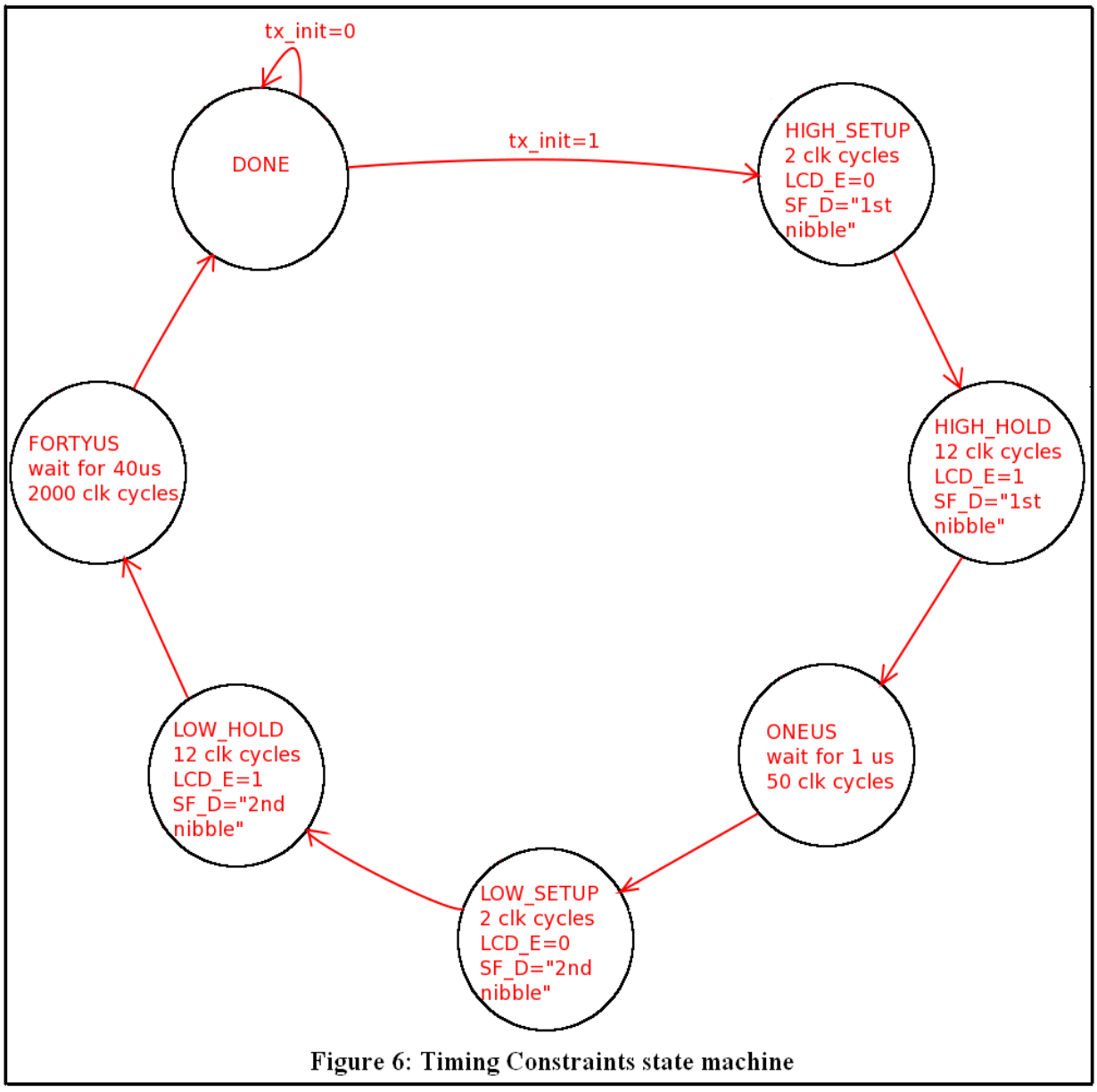


Figure 6: Timing Constraints state machine

Figure 7 shows a simulation of a properly working implementation of LCD controller hardware. This simulation demonstrates the way the disparate state machines work together. As the initialization sequence finishes, the command states of the main state machine begin.

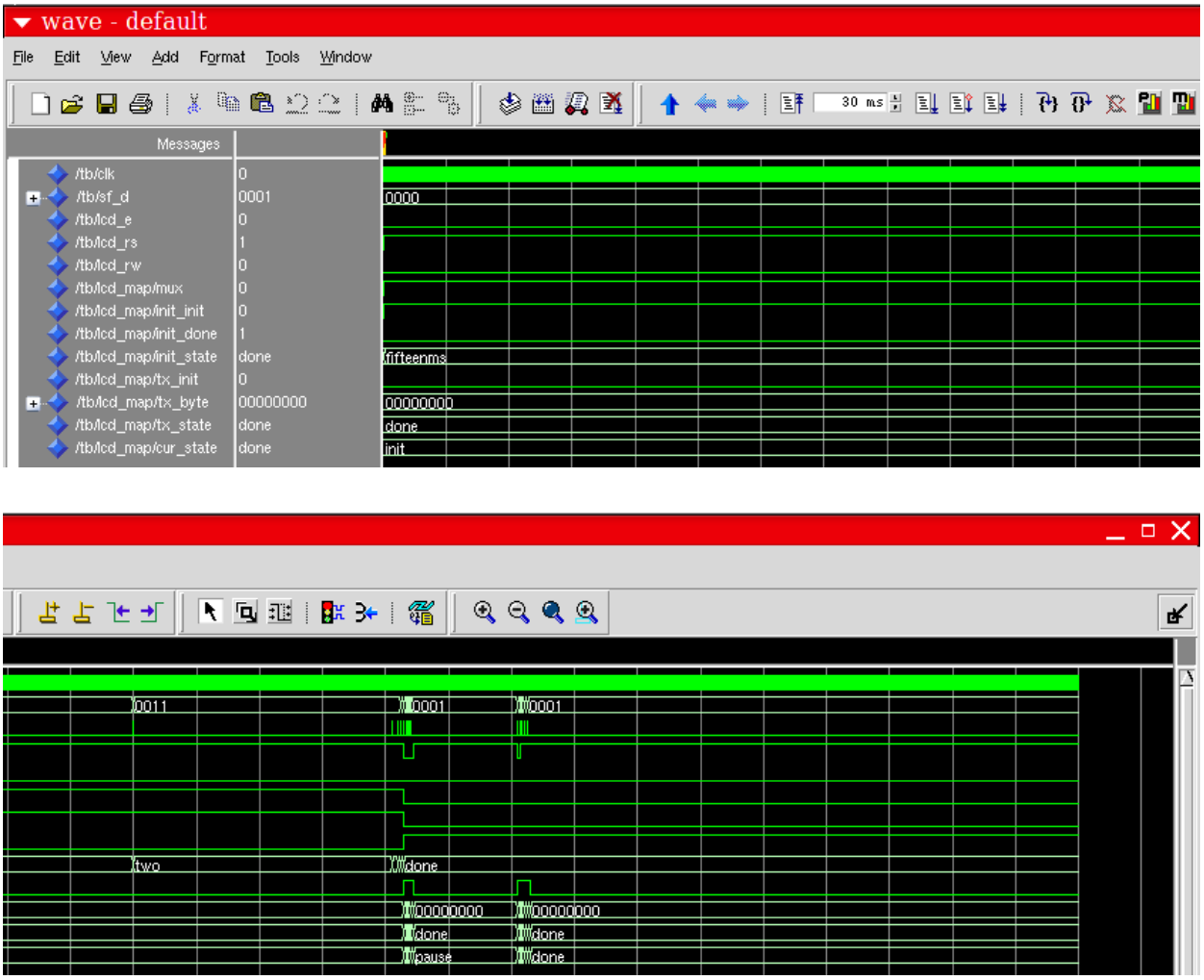
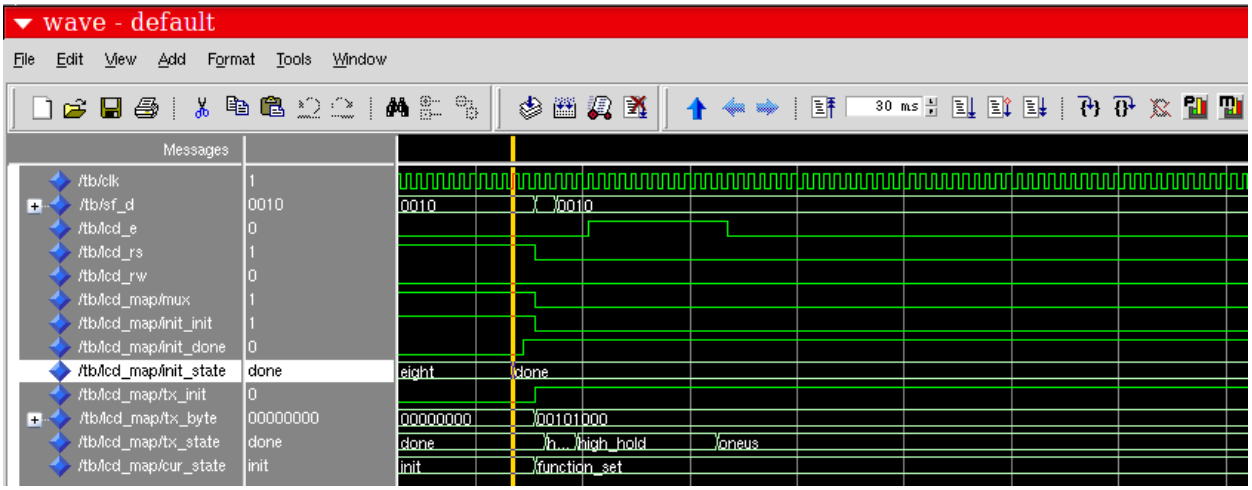
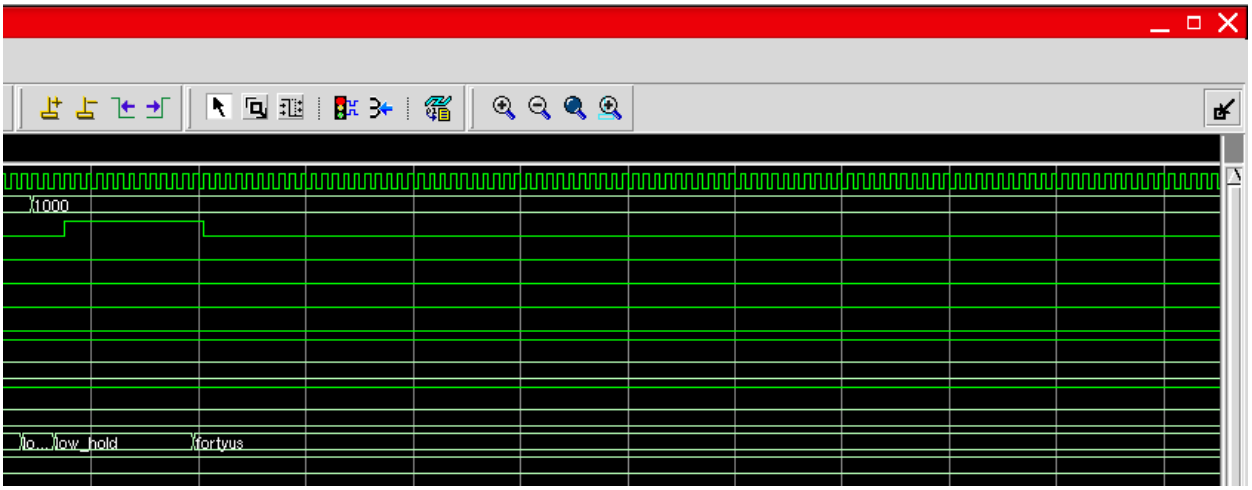


Figure 7: Waveform of circuit

NOTE: ISE Simulator may cause a memory error due to lack of recources. The waveforms shown are from MODELSIM XE

The following simulation waveforms show exactly what happens with all three state machines when the initialization sequence ends, and the states of the function_set command.





```
--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.VComponents.all;

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

architecture behavior of lcd is

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

type init_sequence is (idle, fifteenms, one, two, three, four, five, six, seven, eight, done);
signal init_state : init_sequence := idle;
signal init_init, init_done : bit := '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 SF_D0, SF_D1 : bit_vector(3 downto 0);
signal LCD_E0, LCD_E1 : bit;
signal mux : bit;

type display_state is (init, function_set, entry_set, set_display, clr_display, pause, set_addr, char_f, char_p, char_g,
char_a, done);
signal cur_state : display_state := init;

begin
	LED <= tx_byte; --for diagnostic purposes

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

	--The following "with" statements simplify the process of adding and removing states.

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

--what byte to transmit to lcd
--refer to datasheet for an explanation of these values
with cur_state select
	tx_byte <= "00101000" when function_set,
		"00000110" when entry_set,
		"00001100" when set_display,
		"00000001" when clr_display,
		"10000000" when set_addr,
		"01000110" when char_f,
		"01010000" when char_p,
		"01000111" when char_g,
		"01000001" when char_a,
		"00000000" when others;

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

			--every other state but pause uses the transmit state machine
			when function_set =>
				if(i2 = 2000) then
					cur_state <= entry_set;
				else
					cur_state <= function_set;
				end if;

			when entry_set =>
				if(i2 = 2000) then
					cur_state <= set_display;
				else
					cur_state <= entry_set;
				end if;

			when set_display =>
				if(i2 = 2000) then
					cur_state <= clr_display;
				else
					cur_state <= set_display;
				end if;

			when clr_display =>
				i3 <= 0;
				if(i2 = 2000) then
					cur_state <= pause;
				else
					cur_state <= clr_display;
				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 =>
                                        if(i2 = 2000) then
                                                cur_state <= char_f;
                                        else
                                                cur_state <= set_addr;
                                        end if;

                                when char_f =>
                                        if(i2 = 2000) then
                                                cur_state <= char_p;
                                        else
                                                cur_state <= char_f;
                                        end if;

                                when char_p =>
                                        if(i2 = 2000) then
                                                cur_state <= char_g;
                                        else
                                                cur_state <= char_p;
                                        end if;

                                when char_g =>
                                        if(i2 = 2000) then
                                                cur_state <= char_a;
                                        else
                                                cur_state <= char_g;
                                        end if;

                                when char_a =>
                                        if(i2 = 2000) then
                                                cur_state <= done;
                                        else
                                                cur_state <= char_a;
                                        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

--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
                                LCD_E0 <= '0';
                                SF_D0 <= tx_byte(7 downto 4);
                                if(i2 = 2) then
                                        tx_state <= high_hold;
                                        i2 <= 0;
                                else
                                        tx_state <= high_setup;
```

```
                                        i2 <= i2 + 1;
                                end if;

                        when high_hold => --230ns
                                LCD_E0 <= '1';
                                SF_D0 <= tx_byte(7 downto 4);
                                if(i2 = 12) then
                                        tx_state <= oneus;
                                        i2 <= 0;
                                else
                                        tx_state <= high_hold;
                                        i2 <= i2 + 1;
                                end if;

                        when oneus =>
                                LCD_E0 <= '0';
                                if(i2 = 50) then
                                        tx_state <= low_setup;
                                        i2 <= 0;
                                else
                                        tx_state <= oneus;
                                        i2 <= i2 + 1;
                                end if;

                        when low_setup =>
                                LCD_E0 <= '0';
                                SF_D0 <= tx_byte(3 downto 0);
                                if(i2 = 2) then
                                        tx_state <= low_hold;
                                        i2 <= 0;
                                else
                                        tx_state <= low_setup;
                                        i2 <= i2 + 1;
                                end if;

                        when low_hold =>
                                LCD_E0 <= '1';
                                SF_D0 <= tx_byte(3 downto 0);
                                if(i2 = 12) then
                                        tx_state <= fortyus;
                                        i2 <= 0;
                                else
                                        tx_state <= low_hold;
                                        i2 <= i2 + 1;
                                end if;

                        when fortyus =>
                                LCD_E0 <= '0';
                                if(i2 = 2000) then
                                        tx_state <= done;
                                        i2 <= 0;
                                else
                                        tx_state <= fortyus;
                                        i2 <= i2 + 1;
                                end if;

                        when done =>
                                LCD_E0 <= '0';
                                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;
```

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

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

			when one =>
				SF_D1 <= "0011";
				LCD_E1 <= '1';
				init_done <= '0';
				if(i = 11) then
					init_state<=two;
					i <= 0;
				else
					init_state<=one;
					i <= i + 1;
				end if;

			when two =>
				LCD_E1 <= '0';
				init_done <= '0';
				if(i = 205000) then
					init_state<=three;
					i <= 0;
				else
					init_state<=two;
					i <= i + 1;
				end if;

			when three =>
				SF_D1 <= "0011";
				LCD_E1 <= '1';
				init_done <= '0';
				if(i = 11) then
					init_state<=four;
					i <= 0;
				else
					init_state<=three;
					i <= i + 1;
				end if;

			when four =>
				LCD_E1 <= '0';
				init_done <= '0';
				if(i = 5000) then
					init_state<=five;
```

```
                                        i <= 0;
                                else
                                        init_state<=four;
                                        i <= i + 1;
                                end if;
                        when five =>
                                SF_D1 <= "0011";
                                LCD_E1 <= '1';
                                init_done <= '0';
                                if(i = 11) then
                                        init_state<=six;
                                        i <= 0;
                                else
                                        init_state<=five;
                                        i <= i + 1;
                                end if;
                        when six =>
                                LCD_E1 <= '0';
                                init_done <= '0';
                                if(i = 2000) then
                                        init_state<=seven;
                                        i <= 0;
                                else
                                        init_state<=six;
                                        i <= i + 1;
                                end if;
                        when seven =>
                                SF_D1 <= "0010";
                                LCD_E1 <= '1';
                                init_done <= '0';
                                if(i = 11) then
                                        init_state<=eight;
                                        i <= 0;
                                else
                                        init_state<=seven;
                                        i <= i + 1;
                                end if;
                        when eight =>
                                LCD_E1 <= '0';
                                init_done <= '0';
                                if(i = 2000) then
                                        init_state<=done;
                                        i <= 0;
                                else
                                        init_state<=eight;
                                        i <= i + 1;
                                end if;
                        when done =>
                                init_state <= done;
                                init_done <= '1';
                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.

This tutorial was updated and edited by Brian Zufelt. Brian is an engineering student in the department of Electrical and Computer Engineering at the University of New Mexico.