School of Engineering Science
Simon Fraser University
EECE 350 – Digital Systems Design
Spring 2009

Lab 1: State Machines and the LCD

Lab Demos in the Week of February 2nd-6th

In this lab, you will create a simple driver for the LCD on the Altera board. The driver will be a simple state machine that you will describe using VHDL.

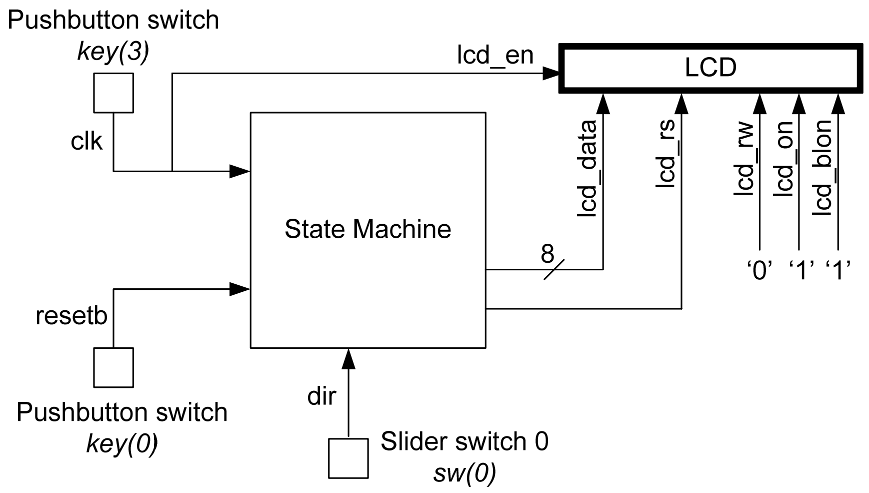
The system you will build will cycle through and display the first five characters of your name. Each cycle, the LCD will display one character. The clock comes from pushbutton switch **key3**. Every time you depress the switch, another character appears. So if your name is “Abcde”, the LCD would display an “A” in the first cycle, add a “b” in the second cycle (so the display is "Ab"), add a “c” in the third cycle (so the display is "Abc"), etc. On the six cycle, it would cycle back to “A” and start again. Each character is displayed to the right of the previous characters, so after 12 cycles (for example), the LCD would display “AbcdeAbcdeAb”.

To make things more interesting, the user can change the direction using the slider switch **SW0**. If this switch is “up” (1), the system operates as described above. If this switch is “down” (0), the system counts backwards (but still starts with the first character). So, in the above example, after 12 cycles, the LCD would display “AedcbAedcbAe”. In this example, it started with A, but then counted backwards (e, d, c, etc).

To make things even more interesting, the user can change the slider switch during any cycle. So, for example, you might count “forwards” for 4 cycles, “backwards” for 2 cycles, and “forwards” for 4 cycles, giving an LCD display of “AbcdcbcdeA”.

There is also a reset input; this will be controlled by the pushbutton switch **key0**. When this pushbutton switch is lowered, the system resets immediately. After a reset (and at the start), the state machine takes 6 cycles before starting with the first cycle of your name (this is due to the need to reset the LCD display; this will become clear later).

The following diagram shows the overall system you will build:



How to use the LCD:

To do this assignment, you have to know a bit about how the LCD works and how to interface with it. The LCD has five single-bit control inputs, and one 8-bit wide input bus. The five control inputs are as follows:

lcd_on : '1' turns the LCD on. In this lab, this should always be '1'
lcd_blon : '1' turns the backlight on. In this lab, this should always be '1'
lcd_rw : whether you want to read or write to the internal registers. In this lab, we will only be writing, meaning this should always be '0'
lcd_en : this is an enable signal. It is also used to latch in data and instructions (see below)
lcd_rs : this allows you to indicate whether the **lcd_data** bus is being used to send characters or instructions (see below).

The 8-bit bus is called **lcd_data** and is used to send instructions or characters to the LCD display.

You can communicate with the LCD using either instructions (to set the LCD in a certain mode or tell it to do something like clear the display) or using characters (in which case the character is displayed on the screen). Each cycle, you can send either one instruction or one character on the 8-bit bus. If you are sending an instruction, the **lcd_rs** signal should be set to 0, and if you are sending a character, the **lcd_rs** signal should be set to 1.

The data bus is sampled on the *falling* edge of the **lcd_en** signal. In this lab, we will drive **lcd_en** with the system clock (which comes from one of the pushbuttons). It is important to remember that the LCD instruction or character is accepted on the falling edge of this clock (this is different than the state machine, which changes states on the rising edge of the clock).

So, to be clear, to send an instruction or character, you would do the following. First, **lcd**_on, **lcd_blon**, **lcd_rw** should be fixed as described above. **lcd_en** would initially be 1. You would then drive **lcd_rs** with a 0 (if you want to send an instruction) or 1 (if you want to send a character). At the same time, you would drive either the instruction code or character code (either of which is 8 bits) on **lcd_data**. Then, **lcd_en** would drop to 0, and the LCD would either accept and execute the instruction, or accept and display the character.

There are several instructions that the LCD accepts. This handout will not describe them in detail. Instead, this handout will indicate a sequence of instructions which will set up the LCD properly. To set up the LCD, you should send the following instructions, in this order, once per cycle:

00111000	(hex "38")
00001100	(hex "0C")
00000001	(hex "01")
00000110	(hex "06")
10000000	(hex "80")

In fact, the first instruction (00111000) should be sent twice, since depending on how you implement the reset, you might miss the first one. Therefore, reseting the LCD will require 6 cycles. If you want to understand what these instructions mean, you can consult the LCD datasheet, which is on the course web site.

Once you have set up the LCD as described above, you can send characters, one character per cycle. The following diagram shows the character encoding.

Character	Code	
	Binary	Hex
Space	00100000	20
!	00100001	21
"	00100010	22
#	00100011	23
$	00100100	24
%	00100101	25
&	00100110	26
'	00100111	27
(	00101000	28
)	00101001	29
*	00101010	2A
+	00101011	2B
,	00101100	2C
-	00101101	2D
.	00101110	2E
/	00101111	2F
0	00110000	30
1	00110001	31
2	00110010	32
3	00110011	33
4	00110100	34
5	00110101	35
6	00110110	36
7	00110111	37
8	00111000	38
9	00111001	39
:	00111010	3A
;	00111011	3B
<	00111100	3C
=	00111101	3D
>	00111110	3E
?	00111111	3F

Character	Code	
	Binary	Hex
@	01000000	40
A	01000001	41
B	01000010	42
C	01000011	43
D	01000100	44
E	01000101	45
F	01000110	46
G	01000111	47
H	01001000	48
I	01001001	49
J	01001010	4A
K	01001011	4B
L	01001100	4C
M	01001101	4D
N	01001110	4E
O	01001111	4F
P	01010000	50
Q	01010001	51
R	01010010	52
S	01010011	53
T	01010100	54
U	01010101	55
V	01010110	56
W	01010111	57
X	01011000	58
Y	01011001	59
Z	01011010	5A
[	01011011	5B
¥	01011100	5C
]	01011101	5D
^	01011110	5E
_	01011111	5F

Character	Code	
	Binary	Hex
`	01100000	60
a	01100001	61
b	01100010	62
c	01100011	63
d	01100100	64
e	01100101	65
f	01100110	66
g	01100111	67
h	01101000	68
i	01101001	69
j	01101010	6A
k	01101011	6B
l	01101100	6C
m	01101101	6D
n	01101110	6E
o	01101111	6F
p	01110000	70
q	01110001	71
r	01110010	72
s	01110011	73
t	01110100	74
u	01110101	75
v	01110110	76
w	01110111	77
x	01111000	78
y	01111001	79
z	01111010	7A
(	01111011	7B
|	01111100	7C
)	01111101	7D
→	01111110	7E
←	01111111	7F

So, for example, if you wanted to display an "a", you would send 01100001 on the **lcd_data** bus. Note that the table above includes both binary and hexadecimal (base-16) for each code; computer engineers like to talk in hexadecimal, since it is more convenient than binary. Other characters are available, and you can even design your own characters. See the datasheet on the course web site if you want more information.

There are stringent timing requirements that must be met using the LCD. However, in this lab, we are using the pushbutton switch as a clock, and it is not possible for you to push the button so fast that you are in danger of violating any of these minimum times. All that matters for this lab is that you need to make sure that the control lines are steady when the clock (**led_en**) switches from high to low.

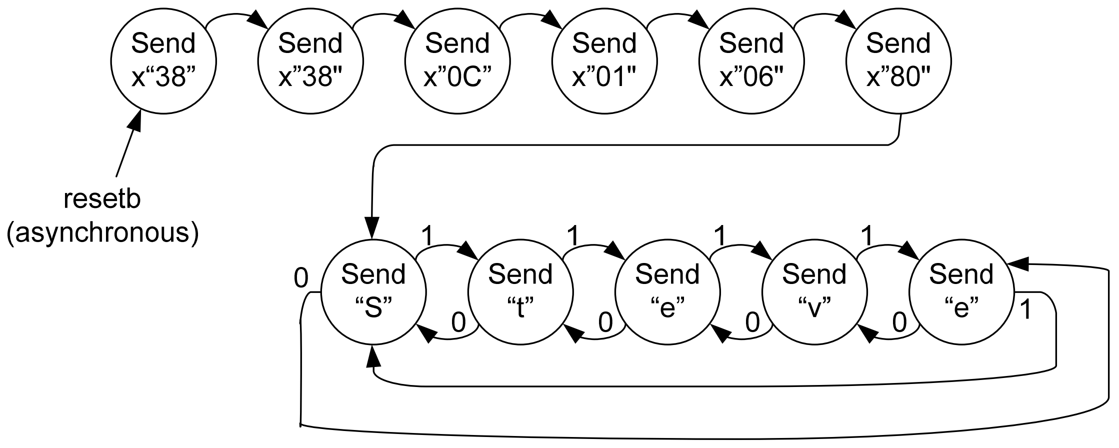
If you want more information on the LCD, check out the DE2-70 User's guide and the LCD datasheet on the course web site.

Part 1 (2 marks):

Design a Moore state machine to implement the circuit as described on the first page of this handout. Your design must include a complete **state table** and a **state diagram.** Note ***both*** lab partners are required to complete the necessary state table and diagram for their own names.

Your state table should show the value of lcd_data, lcd_rs, and the next state for all permutations of the current state, sw(0), and resetb. All states should be named and given a binary value. You must show all of the following encodings in your state table: sequential, gray, and one-hot state encodings. See Table 8.1 of your textbook for an example state table. When writing the VHDL you can use either an enumerated type for states (i.e. type StateType is…) or explicitly use a sequential state encoding (i.e. constant S0: …).

Your state diagram will be something like the picture below. Each state is represented by a circle, arrows indicate state transitions. Include a name for each state, the value of the outputs for each state, and the value of the inputs for each transition (if your name is "Steve").



Upon reset, the state machine cycles through the first six states regardless of the input. The reset is asynchronous; this means that when the resetb signal is asserted, the state machine is reset immediately, without waiting for a clock edge. The reset is also active low, meaning that a "0" means reset, and a "1" means normal operation (this makes it easier to use the pushbutton switch). The state machine is positive-edge triggered; this means that the transition from one state to the next occurs on the rising edge of the clock. The outputs of the state machine are the signals **lcd_rs** and **lcd_data**; given the discussion on the previous pages, you should be able to figure out what should be driven on these signals during each cycle. Note that this is a Moore state machine, meaning the output depends only on the state.

You can use the VHDL template file, lab1.vhd, from the course website to get started.

Simulate your design, and make sure that it works as expected. Be ready to show your design (including your state tables) *and your simulation* to the TA during your lab demo. Your Part 1 mark will be:

- 0/2: if you haven't done anything
- 1/2: if you have done something, but it doesn't work
- 2/2: you have entered *and simulated* your design, and it works

Hint: Earlier I mentioned that the LCD accepts data on the falling edge of the clock. Don't be confused. In the state machine you design here, the state changes (and hence output changes) all happen on the rising clock edge. This is a normal state machine, just like we discuss in class.

Part 2: (3 marks)

1. Before downloading your circuit to the board, it is a good idea to play with the LCD to make sure you understand how it works. To help you do this, we have created a VHDL design that does nothing more than make the following connections:

    lcd_rs is connected to slider switch SW8
    lcd_data is connected to slider switches SW7 to SW0
    lcd_en is connected to pushbutton switch KEY0
    lcd_rw is tied to 0
    lcd_on and **lcd_blon** are tied to 1

    You can download this VHDL file, test_lcd.vhd, from the course website. Download this design and use the switches to first initialize the LCD and then send characters to the LCD. Remember to use the pin assignments file from the tutorials. If you have any questions at this stage, please talk to your TA before moving on. You should be able to successfully send characters and see them on the LCD screen. Don't skip this step, because if you are not sure you understand how the LCD works, you will have problems debugging later.

2. Download the circuit you designed in your preparation. Again, remember to use the pin assignments file from the tutorials. Cycle through the states and show that it operates as expected. Test the reset button to make sure that works too. You will probably find it easier to see what is going on by wiring the state bits (probably called something like "present_state" or "current_state" in your VHDL code) to the green LEDs so you can easily see what state you are in.

    Demo your working circuit to your TA before you leave the lab. Your mark will be one of:

    0/3: if you don't even show up, or show up and don't even try to complete the lab.
    1/3: if you do some work, but really can't demonstrate your state machine sequencing through states at all, or if you can't answer questions about your work satisfactorily.
    2/3: if you can sequence through states, but the LCD isn't displaying characters properly, or if some of your state transitions are incorrect.
    3/3: it works correctly, and you can explain your code to the TA

Note that the TA will ask to see your code, and will ask questions about it. The TA may also ask questions related to state machine design. You should be able to answer these questions to get full marks.

Bonus Marks (0.5 marks)

If you finish early, you can earn an extra 0.5 bonus marks by doing one (or both) of the following. It may not seem that 0.5 marks is a lot, but the performance part of the lab is only out of 3, so 0.5 marks is significant. Please do not feel you must do either of these; they are challenging and only for those who are really keen. No part-marks for bonus, and you only get 0.5, even if you do both. Note that, just like the regular performance marks, you can only get bonus marks if you finish your design and have it ready for your lab demo time.

To earn bonus marks, you can do one or both of the following:

1. From the datasheet, figure out how to create your own characters. Create at least four new characters, and modify your state machine so it displays your name (or some other message) using this new character set.

2. At the end of 18 characters, the screen is full, and no more characters are displayed. Include a counter that keeps track of how many characters have been displayed and clears the screen after 18 characters have been displayed (and then continues normally). The instruction code to clear the screen is 00000001. The challenge here is that you don't know which characters will be displayed (since the user can change the sequence using the SW0 switch).