SFU

ENSC350: DIGITAL SYSTEMS DESIGN

SPRING 2024 - 1241

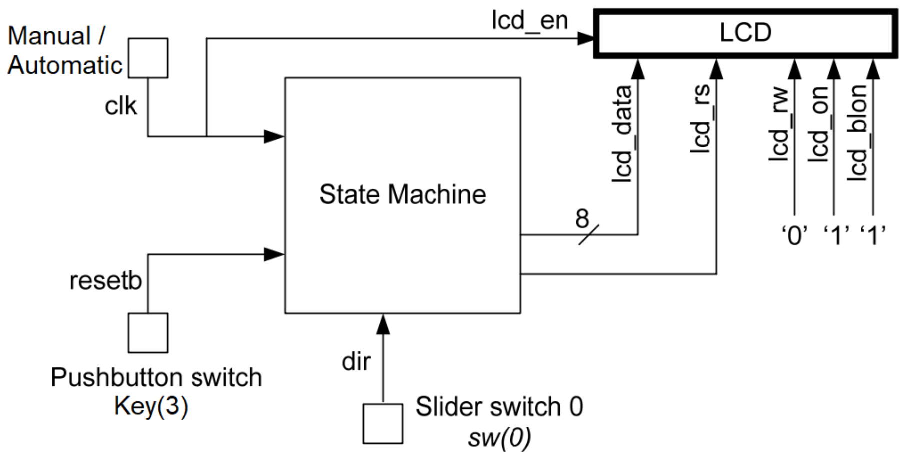
LAB 1: STATE MACHINES AND THE LCD

DESCRIPTION

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. You will first build a state machine which you can clock using a push-button switch. You will then modify the design to use an on-board oscillator and an on-chip frequency divider.

- Working week: January 22-29, 2024
- Marking week: January 30 & February 01, 2024

The following diagram shows the overall system you will build:



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. Then, at every cycle, 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 sixth 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".

The user can change the direction using the slider switch **SW(0)**. If this switch is "down" (0), the system operates as described above. If this switch is "up" (1), 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).

Also, 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 for this combination of "AbcdcbcdeA".

There is also a reset input; this will be controlled by the pushbutton switch **Key(3)**. 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).

LEARN HOW TO USE THE LCD

In this section, you will learn how the embedded LCD operates. 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.
- **lcd_rs:** this allows you to indicate whether the **lcd_data** bus is being used to send characters or instructions.

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. 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 set 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 all of 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 was provided on the Canvas site.

Once you have set up the LCD as described above, you can send characters, one character per cycle. So, for example, if you wanted to display an "a", you would send 01100001 on the **lcd_data** bus. Note that the table below 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 Canvas site if you want more information.

There are stringent timing requirements that must be met using the LCD. Therefore, we have to be careful while implementing high speed oscillators. The key here is to reduce the speed of the oscillators as low as possible to visualize the changes at the LCD. Most importantly, you need to make sure that the control lines are steady when the clock (**led_en**) switches from high to low.

The following tables show 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

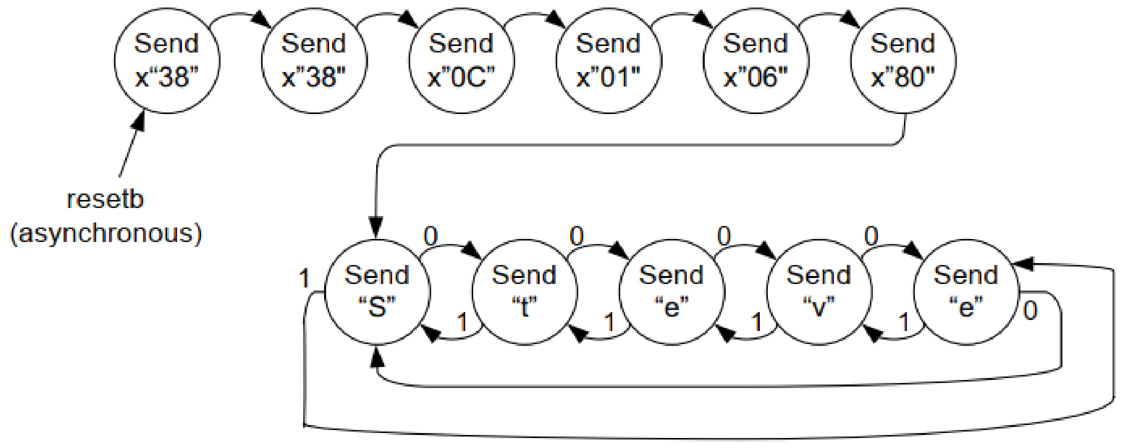
Character	Code	
	Binary	Hex
`	01100000	60
a	01100001	61
b	01100010	62
c	01100011	63
d	01100100	64
e	01100101	65
f	01100110	66
g	01100111	67
h	01101000	68
i	01101001	69
j	01101010	6A
k	01101011	6B
l	01101100	6C
m	01101101	6D
n	01101110	6E
o	01101111	6F
p	01110000	70
q	01110001	71
r	01110010	72
s	01110011	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

Marks

This section has no marks value.

MANUALLY-CLOCKED STATE MACHINE

Design a state machine to implement the circuit as described on the first page of this handout. The state diagram might be something like this (if your name is "Steve"):



In this first version of your implementation, the clock (Manual) comes from pushbutton switch **Key(0)**.

Upon reset (asynchronous), the state machine cycles through the first six states regardless of the input. Importantly, the reset is 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 output of the state machine is 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 each cycle. Note that this is a Moore state machine, meaning the output depends only on the current state.

You can use the template (called **Lab1.vhd**) on the Canvas site. You should make all your changes to this file.

Simulate your design using Modelsim, and make sure that it works as expected. A very simple testbench should be designed for this. This testbench resets the system, changes de direction, and toggles the clock. If the clock cycle in the testbench is set to 6 ns, the suggested run-time length will be of 80 ns to ensure you see all the state transitions. Don't forget to Zoom Full to see the whole thing. Even so, you'll probably have to Zoom in to see enough detail to convince you it is working. Manually observe the waveform and make sure it matches what you expect. Present the simulation on your demo date.

Once you are satisfied with your simulation, download your design to the board. Remember to use the pin assignments properly. Once in the board, manually 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. Interestingly, this highlights a fundamental limitation of debugging directly on hardware because you can not automatically see any of the internal signals unless you have manually connected them to an output before compilation.

Hint: Earlier it was 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 all output changes) all happen on the rising clock edge. This is a normal state machine.

Marks

This section will have a value of 6 marks.

ON-BOARD CLOCK AND CLOCK FREQUENCY DIVIDER

In a real system, you would replace the pushbutton for a clock that is automatically generated. To generate a clock signal (with, say, a 40ns period), you might be tempted to use VHDL code that looks something like this:

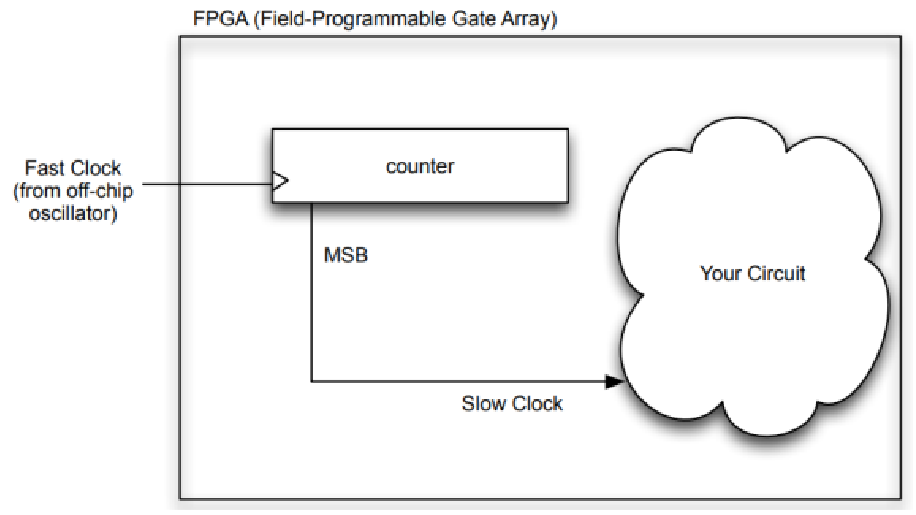
```
clk <= '0';
WAIT FOR 20 ns;
clk <= '1' ;
WAIT FOR 20 ns;
clk <= ‘0’;
WAIT FOR 20 ns;
… etc
```

While this is legal VHDL, it is not synthesizable (This sort of code appears in testbenches though, and as you remember, testbench code does not need to be synthesizable).

In most designs, the clock signal is not generated on-chip, but instead comes from an off-chip source. However, the DE2 board has two oscillators (clock sources) that are permanently wired to specific pins of our FPGA device. These two clock sources are called **CLOCK_50** (which is a 50MHz clock) and **CLOCK_27** (which is a 27MHz clock).

If you would want to replace **Key(0)** with the clock from the oscillator, you could change **KEY(0)** to **CLOCK_50** everywhere it appears in your previous version code. You can try this and recompile; your circuit will now be clocked by the 50MHz clock rather than the **Key(0)**.

If you do this, you will run into another problem. The 50MHz clock is too fast for you to see anything happen in hardware. Even if you could see things happen that fast, it is too fast for the LCD to respond. Therefore, we need to slow down the clock. One way to do this is shown below:


FPGA (Field-Programmable Gate Array)
Fast Clock (from off-chip oscillator)
counter
MSB
Your Circuit
Slow Clock

In the above diagram, the “fast clock” (50MHz) is an input to our chip that feeds the clock input of a counter. Suppose, for the purpose of this discussion, your counter is 8 bits wide (so it can count from 0 to 255, and then rolls back to 0). On each rising edge of the fast clock, the 8-bit counter increments by 1. If we think about the bits inside the counter, we can see that bit 0 (the least significant bit) changes every 1/50Mhz = 20ns. Bit 1 changes every 40ns. Bit 2 changes every 80ns. Bit 3 changes every 160ns and so on.

Notice that any one of these bits could be used as a slow clock signal. As described above, bit 0 changes every 20ns. This means that if bit 0 was used as a clock signal, it would go through the complete clock cycle (low to high and high to low) every 40ns, which translates into a clock of 25MHz. Bit 1 changes at half the frequency of bit 0, meaning if we used bit 1 as a slow clock, since it would change at a rate of 12.5MHz. If we go all the way to the Most Significant Bit (MSB) which is bit 7 of the counter (8 bit counter), we see that the MSB could be used as a 50MHz/256 = 195KHz clock. In the above diagram, this bit is used to clock the user circuit, meaning the user circuit is clocked at 195KHz.

In our design, even this is too fast, since we want to be able to see the effects of transitions. But, as you can deduce from the above, by simply adding more bits to the counter, we can slow down the clock as much as we want. Your first step in this section is to determine how many bits we would need in the counter to slow down a 50 MHz clock to 1 Hz. Then, if the clock is running at 1 Hz, that should be slow enough for us to observe the effect of state transitions.

Hint: the number of bits required is more than 16 and less than 64.

The second step in this version is to integrate this clock divider into your LCD controller from the previous version. Your design will no longer have **Key(0)** as an input. Instead, it will have **CLOCK_50** as an input. When you download and run your design, it will step through the states approximately one state per second. Since the first six states are sending control characters, you should not expect to see anything on the LCD until 6 seconds after you run it.

Hint #1: The counter itself can be done as a process. You can either add this process to your existing architecture or you can create an architecture/entity for the counter and an architecture/entity for the state machine, and combine them structurally.

Hint #2: Consider connecting the slow clock signal to an LED on the board, so you can observe whether the clock signal is working. It will make debugging easier.

Marks

This section will have a value of 3 marks.

CHALLENGE TASK

In this challenge task, you are to modify your circuit from your second version such that it keeps track of the number of characters printed and clear the screen when the screen is full.

The reason this is not trivial is because you don't know exactly what characters you will have displayed (because the user can change the sequence using **SW(0)** switch).

For the challenge mark, you must demo your working circuit on the FPGA board (simulation is not enough). Demo and explain your circuit to the TA.

Marks

This section will have a value of 1 mark.

WHAT TO DEMO

Each section of this lab builds on the previous section. Because you have limited demo time with your TA, you don't need to demo each version individually.

- If you successfully demo the challenge task, we can infer that all the other sections worked.
- If you successfully completed version 2, you could demo that, and we can infer that version 1 worked.
- If you only have part of any version working, you can demo what you have, and explain what you think is wrong for part marks.
- Present the simulation of the operation of the first version of your design.

In all cases, for full marks, you must demo your working circuit on the FPGA board (simulation will get you part marks).

Remember to always:

- Have support material to explain the operation of your code (State Machine diagrams, Logic blocks diagrams, Commented code, etc.).
- Get a backed up fully operational code copy in case any corrupted file or unexpected situation happens to your original files.
- Be certain of how your device operates and how to explain its operation so that full marks could be achieved.
- Use additional hardware resources to make your implementation easier to explain (LEDs to show changes or events).

Finally, a reminder that **all team members must be present** to demo the operation of your design.