Timers and Clock Dividers

Introduction

Timers use counters to create delays. Clock dividers[1] use counters to generate periodic outputs.

In this lab you will use a clock divider to generate an audible tone by switching the voltage applied to a speaker. The frequency of the tone and the duration of the tone will be determined by timers.

You will use the 50 MHz clock on the CPLD board. The period of this clock is $1/50\times10^6 = 20$ ns.

Components

You will need:

- your CPLD board, Byte Blaster JTAG interface and mini-USB power connector,
- the matrix keypad
- the speaker from your ELEX 2117 parts kit
- two cables with alligator clips on both ends from your ELEX 1117 parts kit[2]

Requirements

Customize your design using the last three digits of your BCIT ID: n_1, n_2, and n_3. For example, if your BCIT ID is A00123456 then $n_1 = 4$, $n_2 = 5$ and $n_3 = 6$.

Pushing the keypad key $\boxed{n_1}$ should result in a tone at a frequency of $f = 500 + 100 \times n_2$ Hz that lasts for $0.5 \times n_3$ seconds.

For the ID given above, pressing $\boxed{4}$ should generate a $500 + 100 \times 5 = 1000$ Hz tone for $0.5 \times 6 = 3$ seconds.

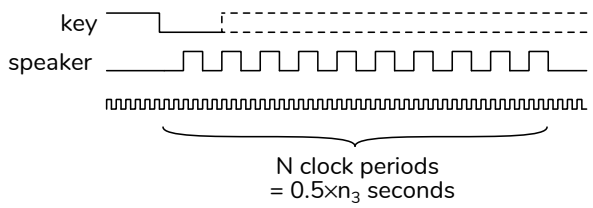
The tone should last for the required duration regardless of how long the key is pressed. The tone should be a square wave. The speaker output should be set low when no tone is being generated.

[1] Also called frequency dividers.

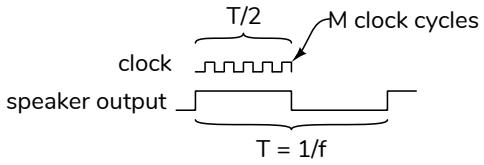
[2] You can put alligator clips over the banana plugs.

Hints

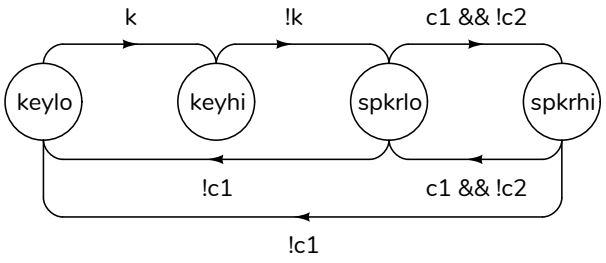
To detect the keypress, wait for the key to be released then wait for it to be pressed. When the key is pressed, set a counter to $N - 1$ where N is the tone duration divided by the clock period. Decrement this counter at the clock frequency. When it reaches 0 turn off the tone:



A second counter counts down from $M - 1$ to 0 where M is half of the tone period ($T/2$) divided by the clock period. When this counter reaches zero the output is toggled (inverted) and the counter is reset to $M - 1$. This creates a square wave:



The sample solution uses a state machine to detect key presses and control two timers. The state machine's state transition diagram is:



where **k** is the input from the appropriate **col** input and is low (0) when the key is pressed; **c1** is the value of the first counter and **c2** is the value of the second counter. The speaker output is set high (1) in state **spkrhi** and low (0) in others.

Using the convention that the first matching transition condition is followed and that the state does not change if no condition matches, the state transition table could be written as:

state	k	c1	c2	next state
keylo	1	x	x	keyhi
keyhi	0	x	x	spkrlo
spkrlo	x	0	x	keylo
spkrhi	x	0	x	keylo
spkrlo	x	x	0	spkrhi
spkrhi	x	x	0	spkrlo

The state durations are determined by M and N. **c1** is set to $N-1$ when the state changes from **keyhi** to **spkrlo**. **c2** is set to $M-1$ when the state changes from **keyhi** to **spkrlo** or when it changes between states **sprklow** and **spkrhi**. **c1** and **c2** decrement when the next state is **sprklow** or **spkrhi**.

The state transition tables for the counters can be most easily written using the current state and next state as the inputs:

c1	state	next state	next c1
x	keyhi	spkrlo	$N-1$
n	x	spkrlo	$n-1$
n	x	spkrhi	$n-1$

c2	state	next state	next c2
x	keyhi	spkrlo	$M-1$
x	spkrhi	spkrlo	$M-1$
x	spkrlo	spkrhi	$M-1$
n	x	spkrhi	$n-1$
n	x	spkrlo	$n-1$

You can declare enumerated state variables and timer values as follows[3]:

```
enum int unsigned { keylo, keyhi, spkrlo, spkrhi }
     state, state_next ;

localparam N=150000000 ;  // 3s/20ns: 28 bits
localparam M=25000 ;      // 1/1000/2/20ns: 15 bits
```

You can implement the state machine by **assign**ing to the **state_next** variable and then creating a **state** register as follows:

[3]The values are for the example ID.

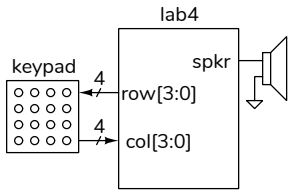
```
// state machine
assign state_next
   = state == keylo && k ? keyhi :
     state == keyhi && !k ? spkrlo :
...

always_ff @(posedge clk50) state
   <= state_next ;
```

You can then use both **state** and **state_next** to set the value of the counters:

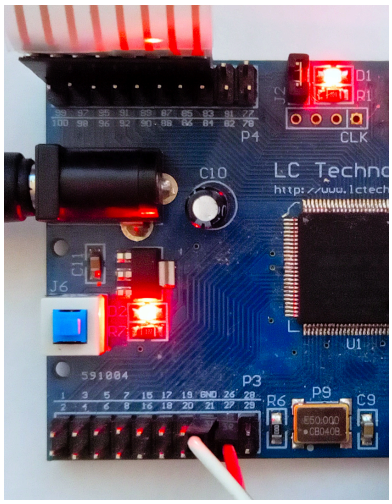
```
always_ff @(posedge clk50) c1
   <= state == keyhi && state_next == spkrlo ? N-1 :
      state_next == spkrlo ? c1-1 :
...
```

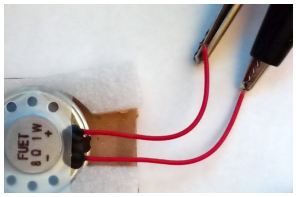
CPLD I/O

The following diagram shows the connections to the CPLD:



Connect the matrix keypad to the CPLD as in previous labs. Connect a CPLD I/O pin to the speaker pin and a ground pin with alligator clip cables (pin 26 was used as the **spkr** pin here, a ground pin is next to it):





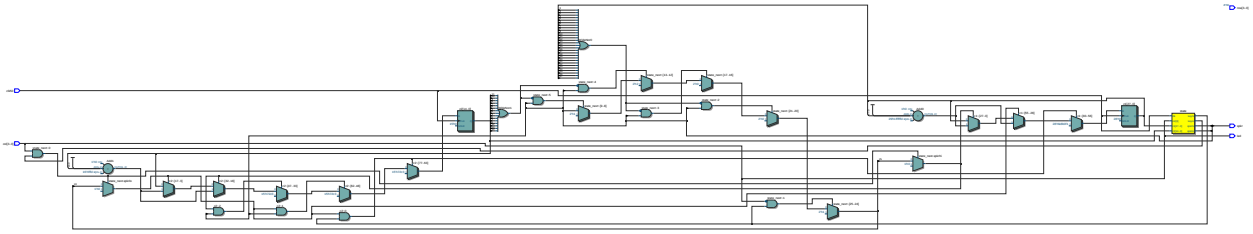


Figure 1: Example RTL Schematic for Lab 4.

Procedure

Create a project, compile it, and configure the CPLD.

If you use the same keypad pins as in the previous lab and Pin 26 for the speaker output, you should end up with the following pin assignments:

atu	From	To	Assignment Name	Value	Enal
		out row[3]	Location	PIN_99	Yes
		out row[2]	Location	PIN_97	Yes
		out row[1]	Location	PIN_95	Yes
		out row[0]	Location	PIN_91	Yes
		in col[3]	Location	PIN_89	Yes
		in col[2]	Location	PIN_87	Yes
		in col[1]	Location	PIN_85	Yes
		in col[0]	Location	PIN_83	Yes
		in clk50	Location	PIN_12	Yes
		out spkr	Location	PIN_26	Yes
		out led	Location	PIN_77	Yes
		in col	Weak Pull-Up Resistor	On	Yes
	<...>	<<new>>	<<new>>		

You can import these pin assignments above from the **`lab4.qsf`** file on the course website.

For troubleshooting you can assign internal signals to the **`led`** output on pin 77 . A high level turns on the on-board LED. You can also view this signal with an oscilloscope.

Test your design.

Troubleshooting

Troubleshoot any electronic device in the following order:

- check power and ground voltages (here: check the power switch and LED)
- check for a clock (here: assume OK)
- check the inputs (here: assign to on-board LED)
- check the outputs (here: assign to pin 77 and use a 'scope)

If you use the same pins as above you can program the **`.pof`** file from the course website to check your hardware.

Submission

To get credit for completing this lab, submit the following to the appropriate Assignment folder on the course website:

1. A PDF document containing:
   (a) A listing of your Verilog code.
   (b) A screen capture of your compilation report (**Window > Compilation Report**) similar to this:

Flow Summary	
<<Filter>>	
Flow Status	Successful - Sun Feb 6 10:57:11 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	lab4
Top-level Entity Name	lab4
Family	MAX II
Device	EPM240T100C5
Timing Models	Final
Total logic elements	72 / 240 (30 %)
Total pins	11 / 80 (14 %)
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

   (c) The schematic created by **Tools > Netlist Viewers > RTL Viewer** and then **File > Export...**. The schematic might look like Figure 1.

2. If you do not demonstrate your completed lab in person, submit a short video, including audio, showing: (1) the tone generated by a button press shorter than the tone duration, (2) a button press longer than the tone duration, and (3) that no tone is generated when pressing a button on a different row and a different column. The audio should be audible. An example video is available on the course website.