Exercise 20: piezo_drv

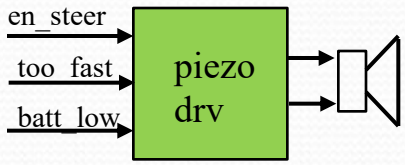
- Work as 2 teams of 2
  - That way your team will have two possible solutions to pick from.

- Recommended division of labor:
  - Work together on high level design
    - ✓ What counters/timers do you need
    - ✓ How will **fast_sim** has its effect
      - ➢ Shorten 3 sec period
      - ➢ Shorten note durations
      - ➢ Speed up frequency of notes?
    - ✓ Sketch bubble diagram

  - Then one codes DUT and other codes testbench

Exercise 20: piezo_drv

- The Piezo element is used to provide warnings.
  - Warning to people in vicinity when rider on
  - Warn rider when going too fast
  - Warn rider when battery is getting low



- 3 Signals come to piezo to inform of various conditions.
- Drive to piezo is simply digital square wave, and its complement.
  - Piezo will respond to signals in the 300Hz to 7kHz range
- The tune for **en_steer** *(normal rider on and riding case)* should occur once every 3 seconds and will be *charge fanfare*
- Tone for **too_fast** should be the first 3 notes of charge fanfare played continuously. **too_fast** tone has priority over all others as it is indicating a dangerous condition.
- If **batt_low** is asserted *(and it is not **too_fast**)* then *charge fanfare* should be played backwards once every 3 seconds, regardless the state of **en_steer**.
- If no input signals are asserted the piezo should be silent.

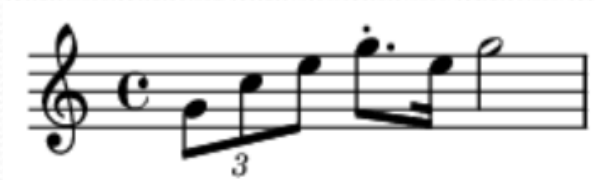
Exercise 20: piezo_drv (Charge! Fanfare)

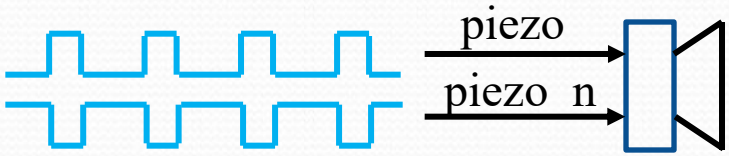
Charge Fanfare as frequencies & durations:

Note:	Freq:	Duration:
G6 (G in octave 6)	1568	2^{23} clocks
C7 (C in octave 7)	2093	2^{23} clocks
E7	2637	2^{23} clocks
G7	3136	$2^{23} + 2^{22}$ clocks
E7	2637	2^{22} clocks
G7	3136	2^{25} clocks

piezo_drv.sv Interface:

Signal:	Dir:	Description:
clk,rst_n	in	50MHz clk
en_steer	in	"normal" operation
too_fast	in	priority over other inputs
batt_low	in	*Charge* backwards
piezo, piezo_n	out	Differential piezo drive





A piezo bender is a "speaker" that can be driven with the GPIO's of our FPGA. We simply drive with a square wave of the frequency we want to generate a tone for.

The duty cycle is not so important (anything from 20% to 80% will do). We will drive differentially for increased sound amplitude.

The interface of the block should be as shown in the table.

The implementation will require a couple of counter/timers *(frequency & duration)* and a controlling SM.

Exercise 20: piezo_drv (Hints)

- In addition to a SM my solution has 3 (counters/timers).

- Duration timer
  - Counter to determine duration note is played. When this count value == the duration the SM is dictating, the note is over and SM moves to next state
- Repeat timer
  - This timer just capable of timing out 3 seconds. Used to repeat "charge" every 3 seconds
- Period *(frequency)* timer
  - Timer used to establish the note frequency *(actually period)*. When the timer hits the note period established from SM, this timer resets.
  - When timer reaches half value of note period then piezo switches from high to low *(or low to high (polarity does not matter))*.

Exercise 20: piezo_drv *(parameter **fast_sim**)*

- The duration of the notes (2^{23} clock cycles) is quite long for simulation purposes *(especially when we get to fullchip simulations where **piezo_drv.sv** will be one of many blocks being simulated).*

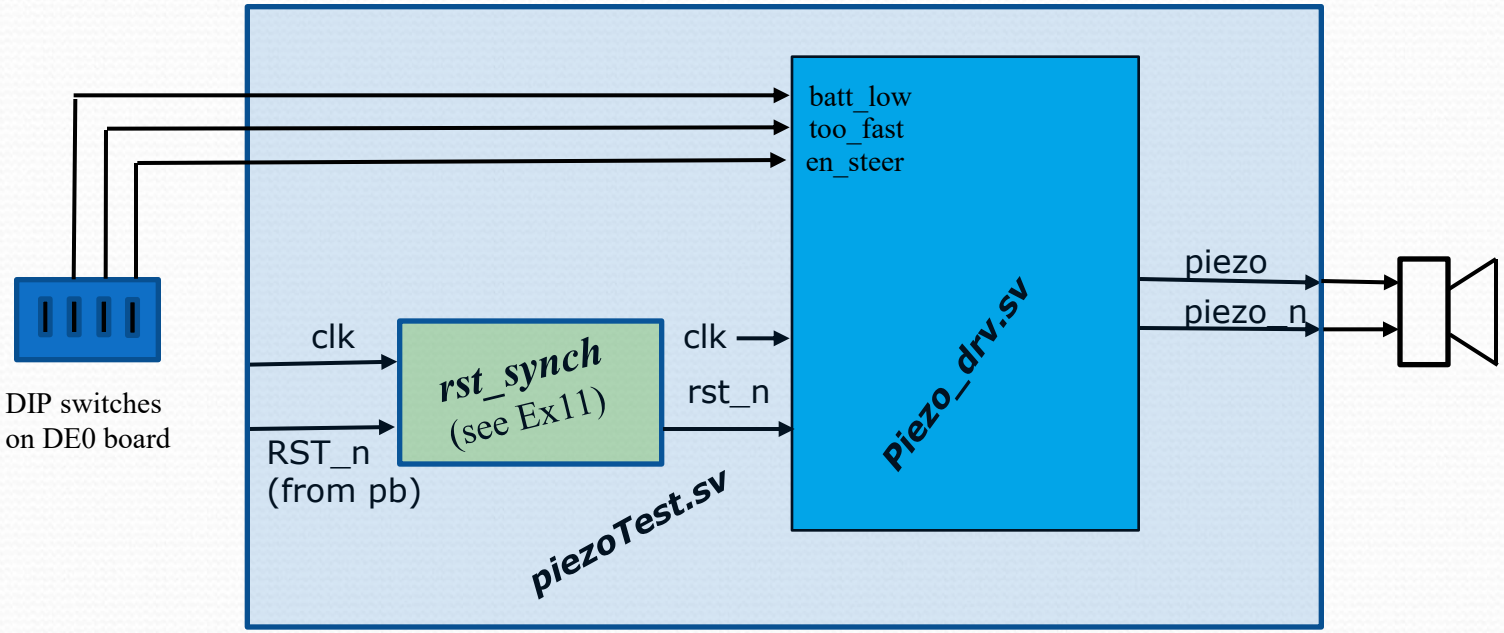
- We will need a method to speed up simulations. ***piezo_drv.sv*** should employ a parameter (called **fast_sim**). **fast_sim** should be defaulted true. When **fast_sim** is passed a 0 then durations and note frequencies should be as specified. When **fast_sim** is true then the duration of notes should be 1/64 their normal length, and all frequencies should be 64X their normal. Durations of the 3sec timer should also be 1/64 of normal. **HINT**: use a **generate if** statement to create an amount by which you increment your various counters. Increment by 1 or by 64 depending on **fast_sim**.

- Create a simple testbench (***piezo_drv_tb.sv***) that simply instantiates the DUT, applies clock and reset and asserts the inputs in order. Note **piezo** and **piezo_n** should not be toggling before any of the inputs are asserted.

- Once your testbench is passing visual inspection of **piezo/piezo_n** move on to the next portion (testing with DE0).

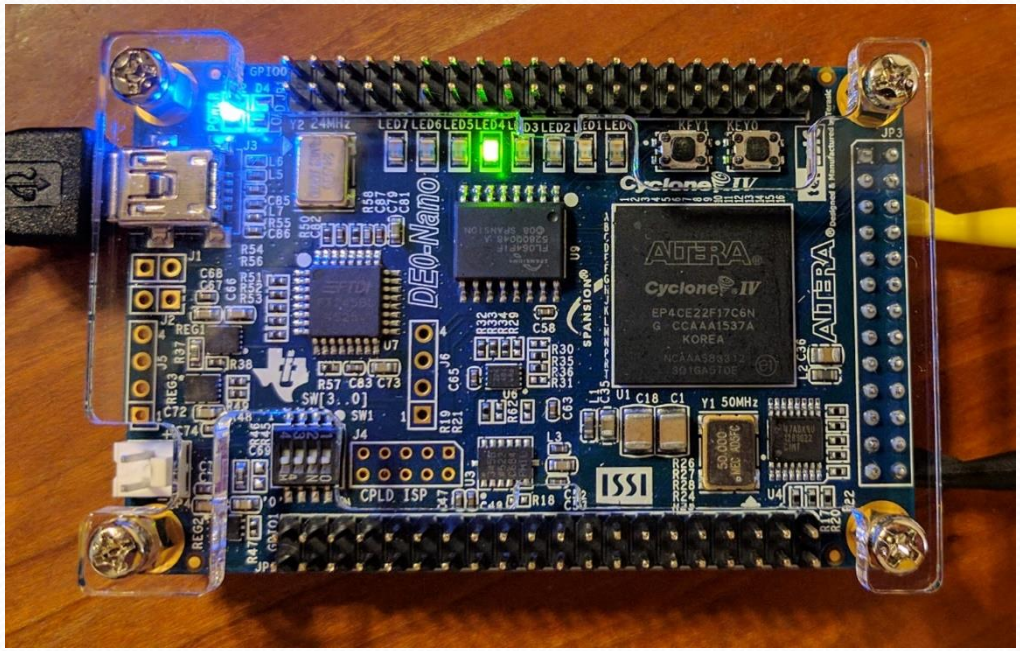
Exercise 20: piezo_drv *(testing on DE0-Nano)*

- It is not possible to test what it really sounds like in ModelSim. This exercise lets you map ***piezo_drv*** to a DE0-Nano FPGA board.
- ***piezoTest.sv*** is provided along with .qsf and .qpf files. Copy your ***rst_synch.sv*** block over to your Ex20 work area.



Exercise 20: Mapping Your piezo_drv to DE0

- There are Quartus project file and settings file available for download: (**piezoTest.qpf, piezoTest.qsf**).
- There is a **piezoTest.sv** that you can drop your **piezo_drv.sv** into.
- Ensure the project builds with no errors
- Call Eric, Julie, Khailanii, or Ting Hung over to demo



Pin mapping shown below: