Exercise 21: Split Your Team (cmd_proc / cmd_proc_vld / piezo)

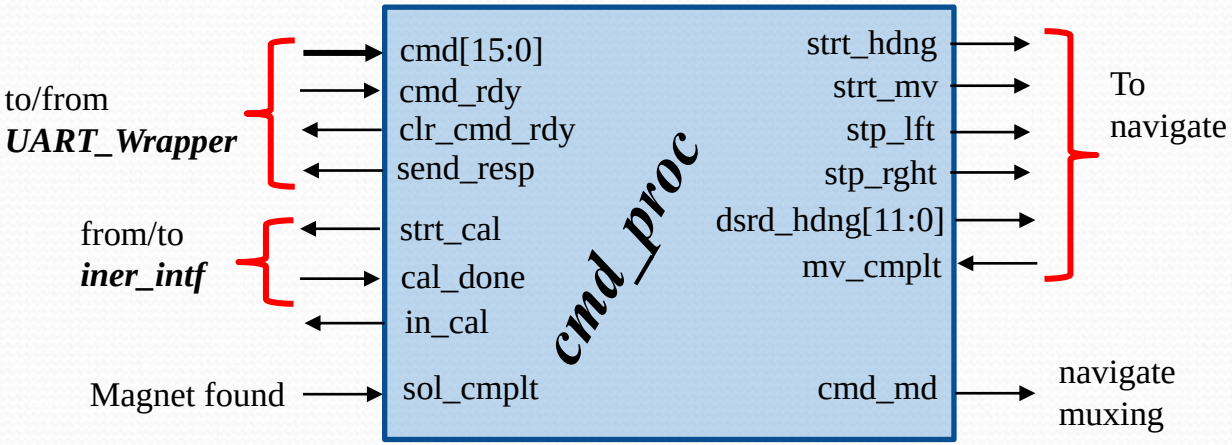
- Reccommended Split
  - 2 people on ***cmd_proc*** implementation & test
  - 2 people on ***piezo_drv*** implementation & test

- For this exercise you will split your team and either work on **cmd_proc** & testing or on the unit to drive “Charge!” fanfare on the piezo buzzer. They are of similar difficulty.

- Decide how you are splitting roles and jump to the appropriate section of this document. ***cmd_proc*** is covered first.

Exercise 21: (cmd_proc)

The ***cmd_proc*** unit's main job is to process incoming commands from Bluetooth (***UART_Wrapper)***. The command could be calibrate gyro, a heading/movement command, or a solve maze command



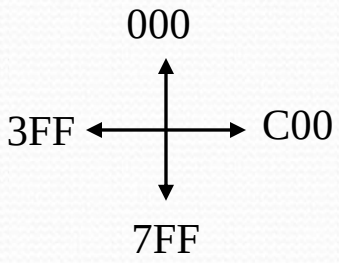
When any new **cmd**[15:0] is **rdy** the ***cmd_proc*** unit will dispatch from its IDLE state to a state to "execute" the command and assert **clr_cmd_rdy** in the process. If the command is calibrate then **strt_cal** will be asserted for 1 clock and the unit will wait for **cal_done** to be asserted. During this time **in_cal** should be asserted. When **cal_done** is asserted the unit should **send_resp** and return to IDLE. If a heading command is received the heading direction should be registered into **dsrd_hdng** and **strt_hdng** should be asserted for 1 clock. When the mazeRunner is at the new heading **mv_cmplt** will be asserted and the unit will **send_resp** and return to IDLE. If a movement command is received **strt_mv** should be asserted for 1 clock, and {**stp_lft**,**stp_rght**} should be registered from **cmd**[1:0]. When ***navigate*** is done executing the move **mv_cmplt** will be asserted. **stp_lft**/**stp_rght** indicate the movement should terminate at a left or right opening in the maze if discovered *(by IR sensors).* Finally a solve command will simply have the unit deassert **cmd_md** *(should have been high otherwise)* and wait for **sol_cmplt** to be asserted. The "intelligence" of solving the maze is in a unit called ***maze_solve***.

Exercise 21: (cmd_proc)

The main job of ***cmd_proc*** is to interpret incoming commands. The command is The command opcode is contained in **cmd**[15:13] there are currently only 4 commands "The MazeRunner" can interpret.

cmd[15:13]	Description:
3'b000	Calibrate command (kick off **strt_cal** and wait for **cal_done**). **send_resp**
3'b001	Heading command. Bits [11:0] represent the desired heading
3'b010	Move command. Bit[1] if set implies movement should stop at a left opening. Bit[0] if set implies movement should stop at a right opening.
3'b011	Start solve command: Deassert **cmd_md** and wait for **sol_cmplt**. **cmd**[0] determines affinity. 1→ left affinity, 0→ right affinity.
3'b1xx	Reserved for future expansion.

cmd[11:0] → Heading	Direction:
12'h000	North
12'h3FF	West
12'h7FF	South
12'hC00	East

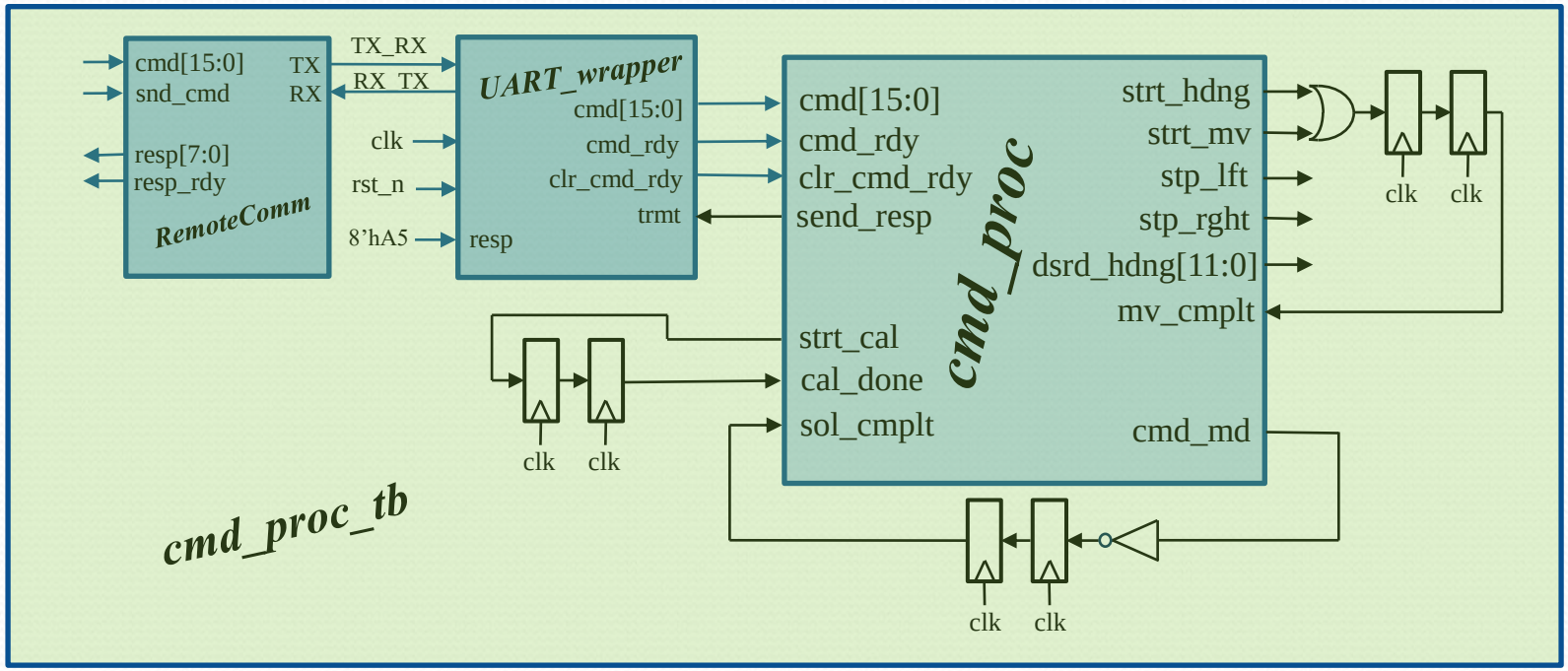


Exercise 21: (cmd_proc) (SM Control)

Extract the needed states from this description:

- The machine sits in ***IDLE*** until there is a **cmd_rdy.** Then it should dispatch to the appropriate state based on the opcode (**cmd**[15:13]). It should immediately **clr_cmd_rdy,**
- If the command is to calibrate is simply asserts **strt_cal** and waits for **cal_done.** It should also **send_resp** *(0xA5 positive ack)* when **cal_done**.
- If a heading command is given it must capture the desired heading *(into **dsrd_hdng** register)* and assert strt_hdng for a single clk. Then it waits for **mv_cmplt** to be asserted, **send**ing a **resp**onse once it is**.**
- If a movement command is given it must assert **strt_mv** and register if the movement should stop at a left or right opening in the maze (**stp_lft**/**stp_rght**). When the ***navigate*** unit is done executing the movement it will assert **mv_cmpt** and ***cmd_proc*** should send a response.
- If a solve maze command is received it must simply deassert **cmd_md** *(which had been high otherwise)* and wait for the **sol_cmplt** signal to assert. **sol_cmpt** asserts when the hall sensor detects a magnet. **cmd_md** forms a mux select which determines if control signals to the ***navigate*** unit come from ***cmd_proc***, or the ***maze_solve*** unit.

Exercise 21: (cmd_proc) (Testing it)



Since ***cmd_proc*** is so central to the design it has an extensive interface. For blocks like this it can be good to instantiate in a testbench with some context. In this recommended context you can apply a 16-bit command to **cmd[15:0]** into ***RemoteComm*** and then assert **snd_cmd** for one clock. This will send the command and you can check for its proper operation.

NOTE: the use of flops to provide **cmplt/done** signals a few clks after their respective **strt** signals.

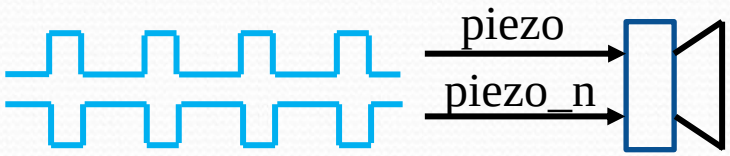
Exercise 21: (cmd_proc) (Testing it)

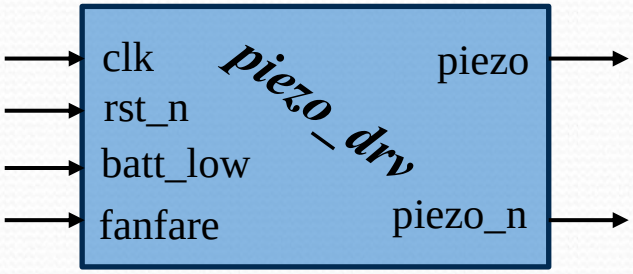
There is a lot to test with ***cmd_proc***, and some of it might be better tested at the actual "fullchip" level when a physics model of the mazeRunner and maze are available. However, the following are tests you should consider performing at this level:

Sequence:	Description:
1	Initialize all inputs to known values and assert/deassert reset
2	Send Calibrate command (0x0000). Wait for **cal_done** or timeout if it does not occur Wait for **resp_rdy** or timeout if it does not occur
3	Send heading command and look for **strt_hdng** at time of **cmd_rdy**. One clock later check that **dsrd_hdng** is heading direction you sent. Check that it sends a 0xA5 response when **cal_cmplt** occurs.
4	Send move command and look for **strt_mv** at time of **cmd_rdy**. One clock later check that either **stp_lft** or **stp_rght** *(depending)* is asserted.
5	Send solve command and look for **cmd_md** to be low one clock after **cmd_rdy**.

Exercise 21: piezo_drv.sv (Fanfare & Battery Low)

- There are two sound sequences the ***mazeRunner*** will make:
  - "Charge!" fanfare when the magnet is found.
  - The quarter notes G6, C7, E7 continuously when the battery is low.
- Battery low has priority over magnet found.
  - If both are asserted battery low wins
- Once "Charge!" is started it is allowed to complete *(no need to preempt "charge!" if battery becomes low).*



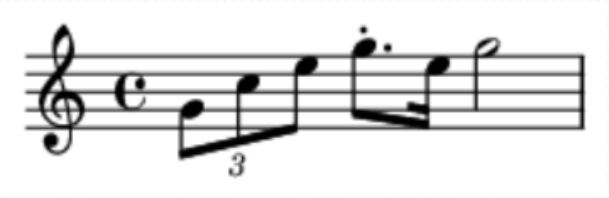


A piezo bender is a "speaker" that can be driven with the GPIO's of our FPGA. We simply drive with a square wave of the frequency we want to generate a tone for.

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

Exercise 21: (Charge! Fanfare)

- When the mazeRunner encounters the magnet it will play "Charge!" fanfare.



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^{24} clocks

- **NOTE**: The battery low sound *(G6,C7,E7 played for 2^{23} clocks each)* is comprised of 3-notes that were already part of "Charge!"
- You will need two counters in the design. A note frequency counter, and a note duration counter.

Signal:	Dir:	Description:
clk,rst_n	In	50MHz clk & asynch reset
batt_low	In	Asserted if battery voltage is low *(has priority over fanfare)*
fanfare	In	Asserted if magnet has been found
piezo,piezo_n	out	Drive the piezo buzzer with desired note frequency for desired duration.

Exercise 21: piezo_drv.sv (Charge! Fanfare & batt_low)

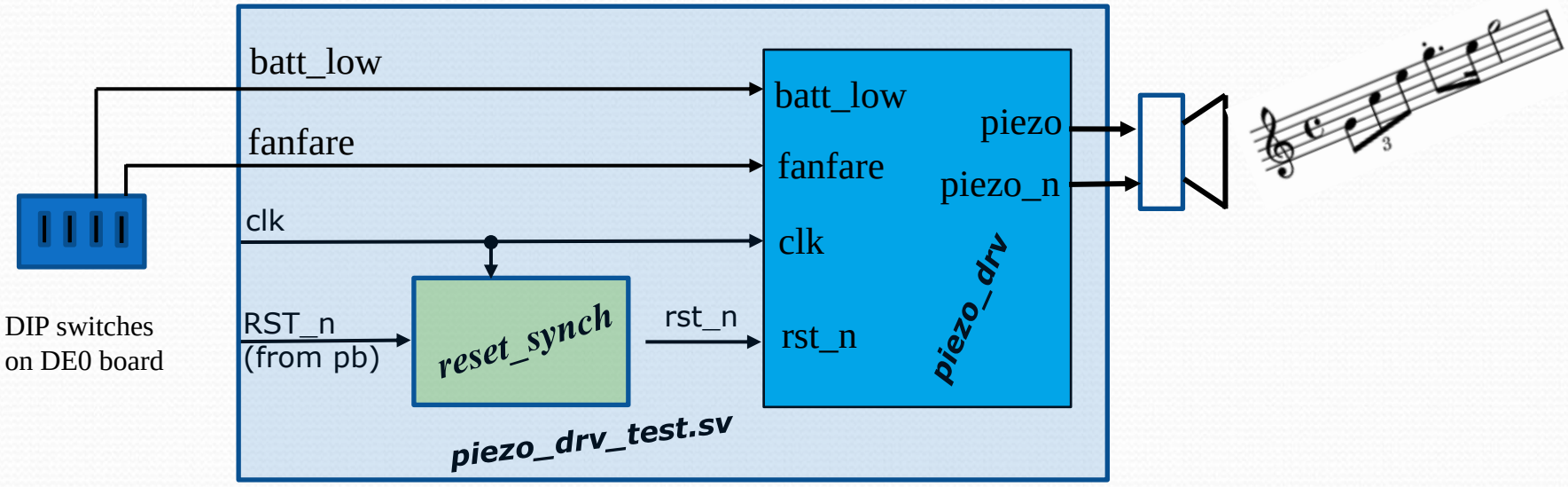
- The duration of the notes (2^{23} clock cycles) is quite long for simulation purposes *(especially when we get to fullchip simulations where **piezo_drv.sv** will be one of many blocks being simulated).*

- 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 should be as specified. When **FAST_SIM** is true then the duration of notes should be 1/16 their normal length. **HINT**: use a **generate if** statement to create an amount by which you increment your duration counter. Incement by 1 or by 16 depending on **FAST_SIM**.

- Create a simple testbench (***piezo_drv_tb.sv***) that simply instantiates the DUT, applies clock and reset and asserts **fanfare** for 1 clk. Note **piezo** and **piezo_n** should not be toggling before **fanfare** is asserted or after the "tune" has completed. Then the testbench should assert **batt_low** and the G6, C7, E7 continuously.

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

Exercise 21: (Charge! Fanfare…testing on DE0)



- Create **piezo_drv_test.sv** to map your **piezo_drv.sv** to the DE0 and test in "real life". Don't forget to pass a **FAST_SIM** parameter of 0 to charge since we are testing it on real HW now.

Signal:	Dir:	Description:
clk	in	50MHz clock
RST_n	in	Unsynchronized input from push button
Batt_low	In	Indicates battery is low
fanfare	in	Magnet has been found, "Charge!" fanfare should be played
piezo/piezo_n	out	Differential drive of piezo bender

Exercise 21: (Charge! Fanfare…testing on DE0)

- There are Quartus project file and settings file available for download: (**piezo_drv_test.qpf, piezo_drv_test.qsf**).
- Open the .qpf and **ensure you add** all necessary files to the project.
- Ensure the project builds in Quartus with no errors
- Once it does call Eric, Harish, or Khailani over for a demo with the DE0.
- All groups must demo a working **piezo_drv.sv** block