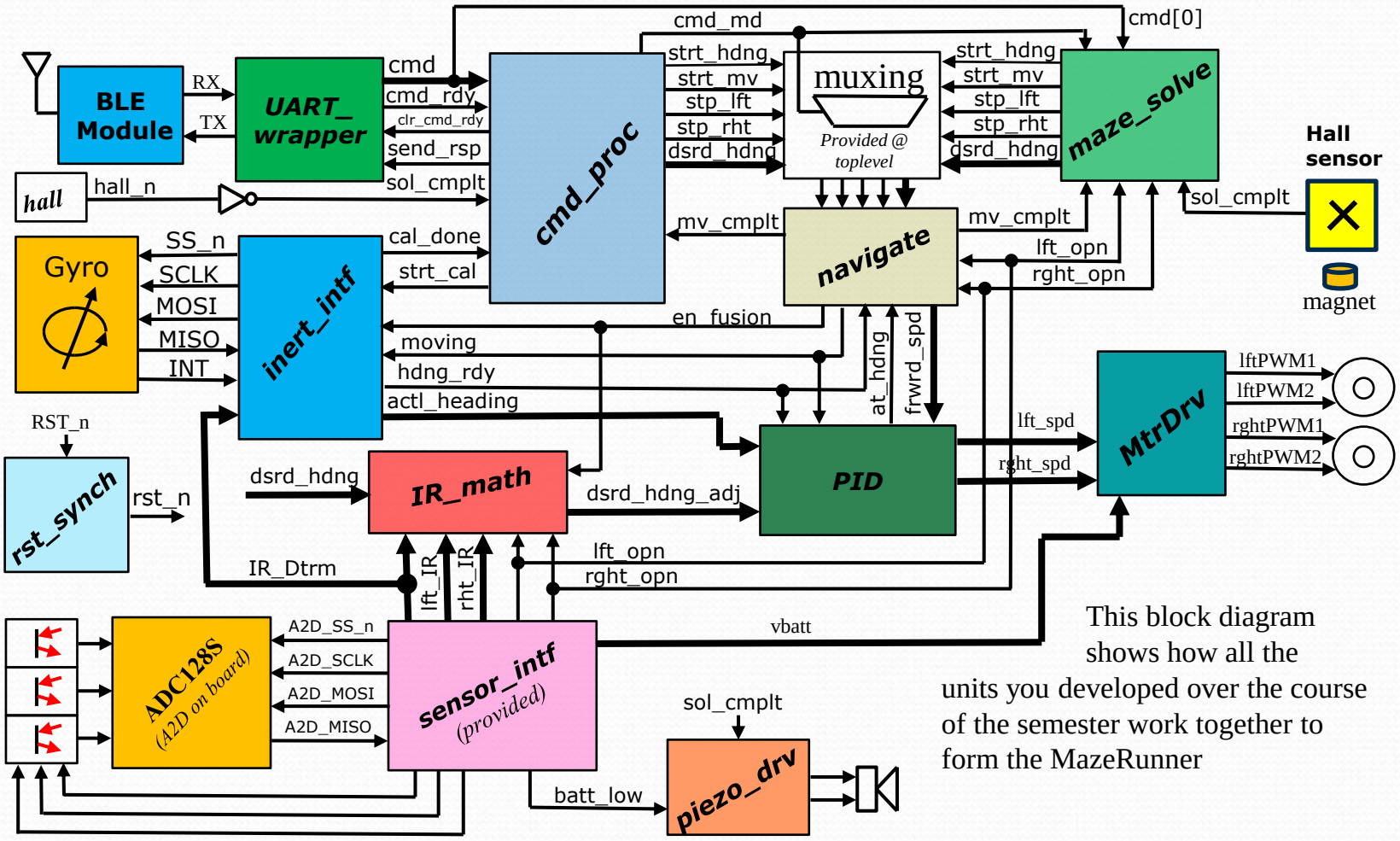
Can you complete a course evaluation?
Feedback is appreciated, and response rates have been low.

Students can find all of their course evaluations here:
https://aefis.wisc.edu/

Block Diagram of MazeRunner *(study this)*

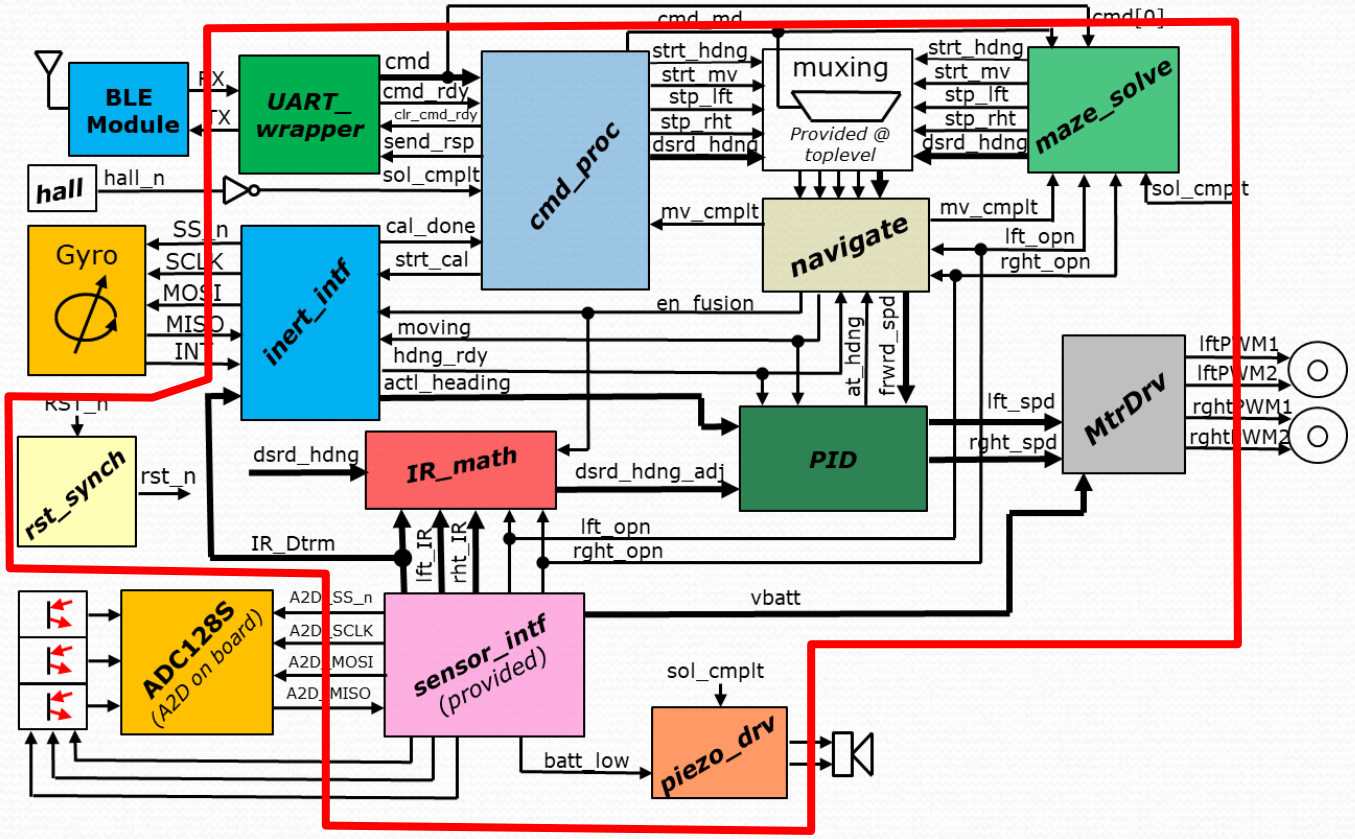


This block diagram shows how all the units you developed over the course of the semester work together to form the MazeRunner

Toplevel Digital of our Design

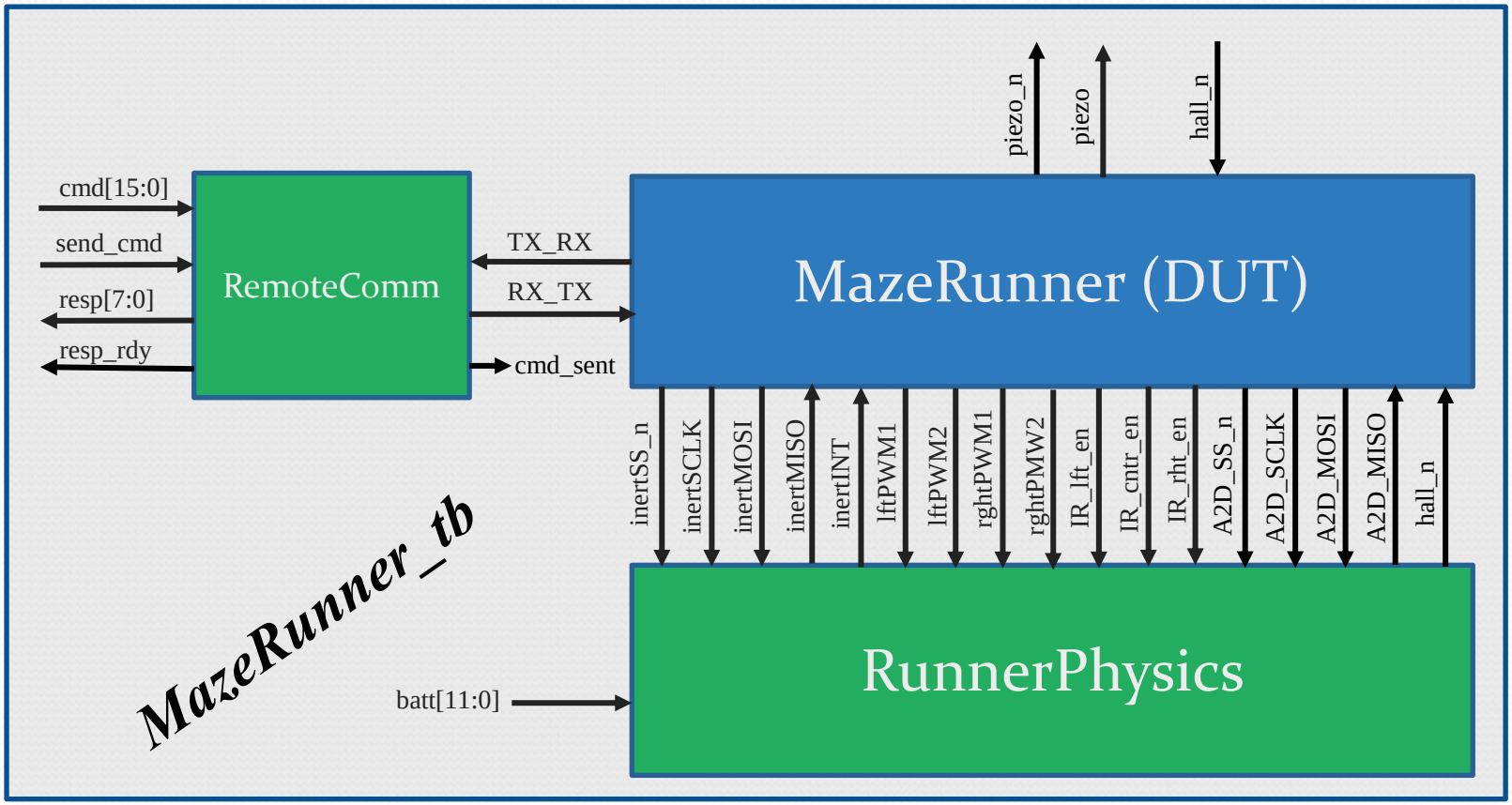
The blocks outlined in red are pure digital blocks, and will be coded with the intent of being synthesized via Synopsys to our standard cell library. For practical purposes we will also map that logic to a DE0 FPGA board so we can run the demos.

You Must have a block called **MazeRunner.sv** which is top level of what will be the synthesized DUT. It is **provided** to ensure we all have the same interface.



After having completed all the exercises/HW's you should have all the sub components designed. The toplevel (**MazeRunner.sv**) is provided. Your jobs are primarily toplevel validation, and toplevel synthesis, and post synthesis simulation of the toplevel for one simple test.

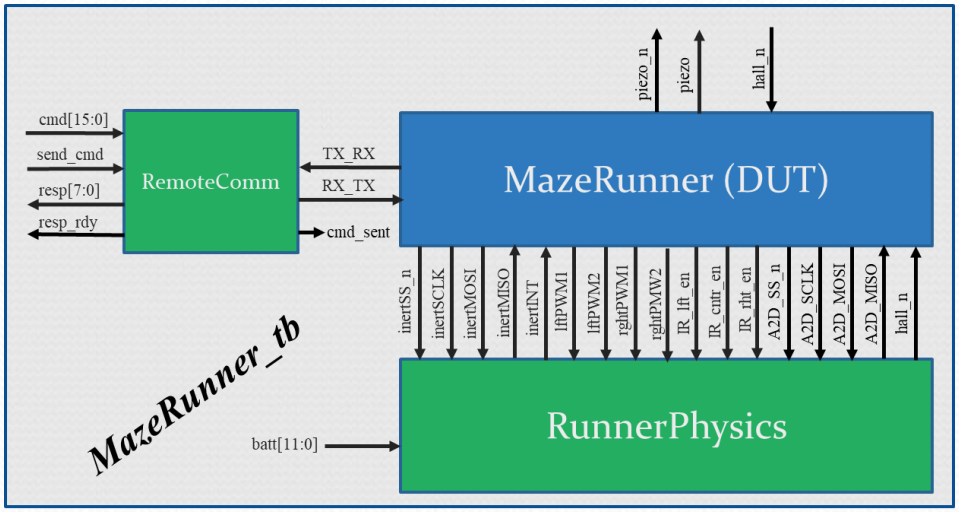
Top Level Testbench (MazeRunner_tb.sv)



Testing the ***MazeRunner*** without some model of its behavior and the maze it runs in is close to impossible. **RunnerPhysics.sv** models the motion physics and inertial sensor and responds to the inertial sensor SPI bus. It also models the IR sensors and responds on the A2D SPI bus. A simple maze model is also "baked" into ***RunnerPhysics***.

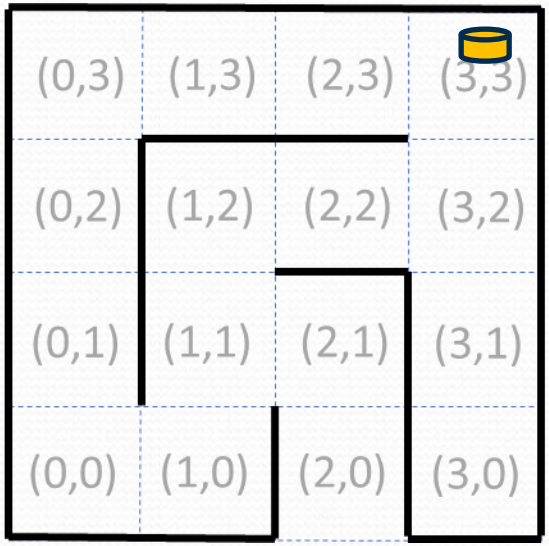
Top Level Testbench

- eBikePhysics is a model of the hub motor and the inertial sensor.
- Provide RunnerPhysics with PWM signals and it will return SPI readings for both the inertial sensor and the IRs *(battery too, but you input battery level).*
- Take a look at RunnerPhysics. It has some internal signals that can be handy for performing self checks.



- Heading_robot[19:8] can be good to check if heading of MazeRunner is similar to commanded heading
- xx & yy represent the position of the robot within the "maze". Inspect bits [14:12] for (xx,yy) coordinates.

- Check RunnerPhysics for magnet_xx_pos, magnet_yy_pos. They are set to (3,3). Feel free to change for your testing
- Maze Runner starts at (2,0) opening to maze.
- Maze topology modeled in RunnerPhysics is as shown →

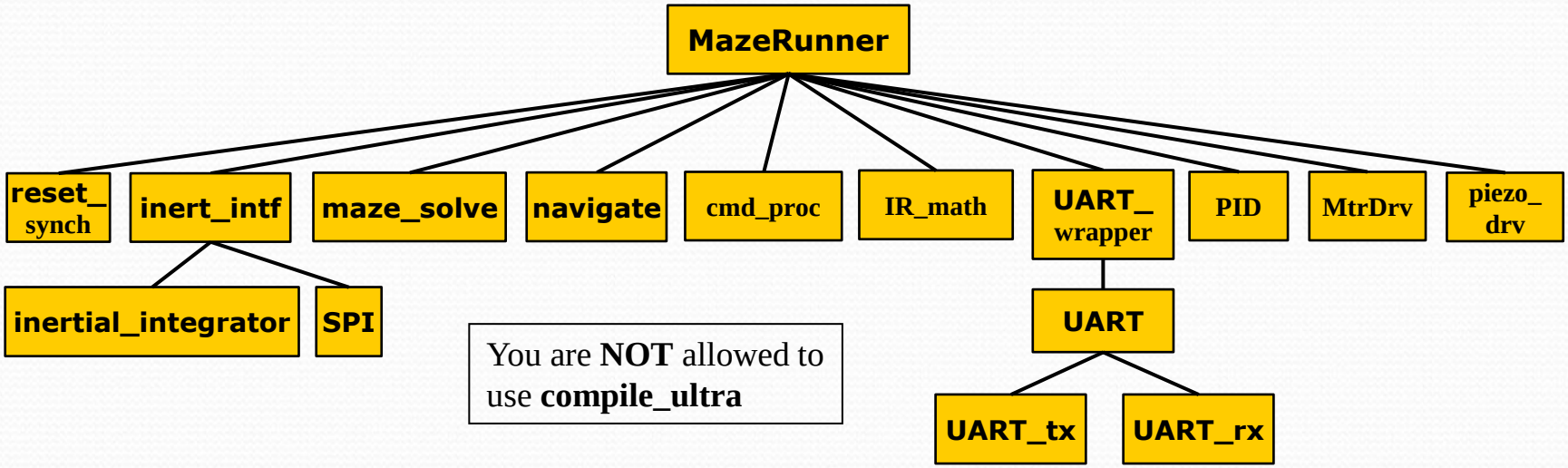


Top Level Tests (on ModelSim FAST_SIM = 0)

- This is where the bulk of your work remains:
  - What cases situations are you going to test?
  - Are you going to have multiple smaller test cases or one big super test run?
  - How are you going to make your tests self-checking?
  - What outputs are you looking at? Don't forget there are some handy signals inside RunnerPhysics that you can use: robot_heading, omega of wheels, xx yy (position)

Synthesis:

- You have to be able to synthesize your design at the **MazeRunner** level of hierarchy.



- Your synthesis script should write out a gate level netlist of follower (**MazeRunner.vg**).
- You should be able to demonstrate at least one of your tests running on this post synthesis netlist successfully.
- Timing (400MHz) (2.5ns) is mildly challenging. Your main objective is to minimize area.

Synthesis Constraints:

Contraint:	Value:
Clock frequency	400MHz (yes, I know the project spec speaks of 50MHz, but that is for the FPGA mapped version. The standard cell mapped version needs to hit 400MHz. 2.5ns period
Input delay	0.6ns after clock rise for all inputs
Output delay	0.5ns prior to next clock rise for all outputs
Drive strength of inputs	Equivalent to a size 2, 2-input NAND gate from our library
Output load	0.1pF on all outputs
Wireload model	16000
Max transition time	0.125ns
Clock uncertainty	0.125ns

NOTE: Area should be taken after all hierarchy in the design has been smashed.

Synthesis Hint:

- You will see a long speed path that does not meet timing.
- Look at the Startpoint and Endpoint in the Synopsys timing report.
- There is a lot of math happening in series with no flops breaking it up in ***IR_math*** that flows right into ***PID***.
- Remember your 352 lessons. Pipelining to break up long speed paths.
- Judiciously flop some signals along the path to break up the long combinational delays.
- Do this on a side branch so you don't screw up the folks trying to validate your design for functionality. Then integrate into the functionally correct design later. It is possible to break functionality via pipelining.

How I Broke Mine:

- One of the things I choose to pipeline was **dsrd_hdng_adj** in ***IR_math***. This caused a funny bug where the latency introduced delayed the fall of **at_hdng**. This caused ***navigate*** unit to immediately think it was done with the heading change.
- I fixed by filtering **at_hdng** inside ***navigate***. Could look for rising edge, or **at_hdng** to be high for multiple consecutive clks.

Post Synthesis Simulation of MazeRunner.vg

- You have to show us post synthesis simulation running on at least one fullchip test on your testbench. This can be a simple test that just performs a calibrate and a heading change.