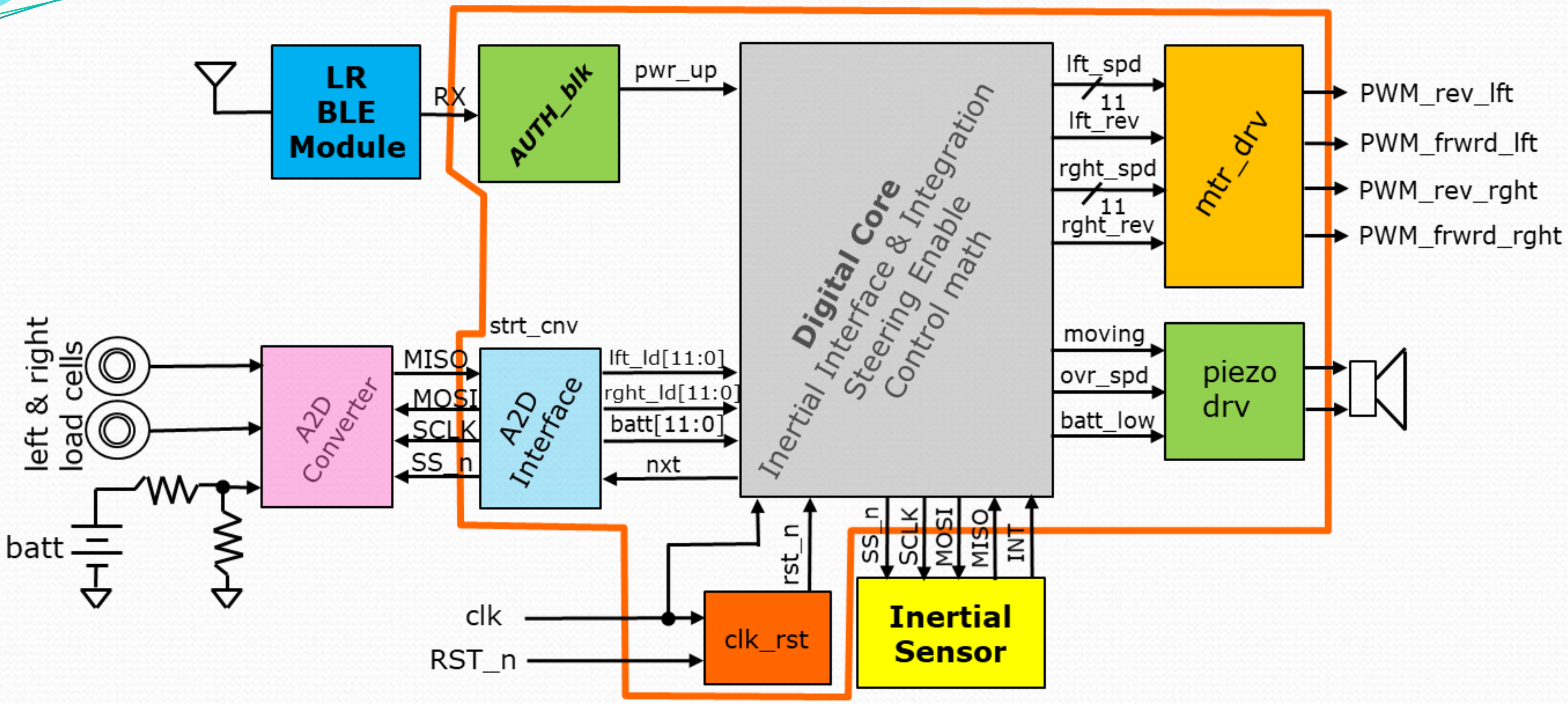
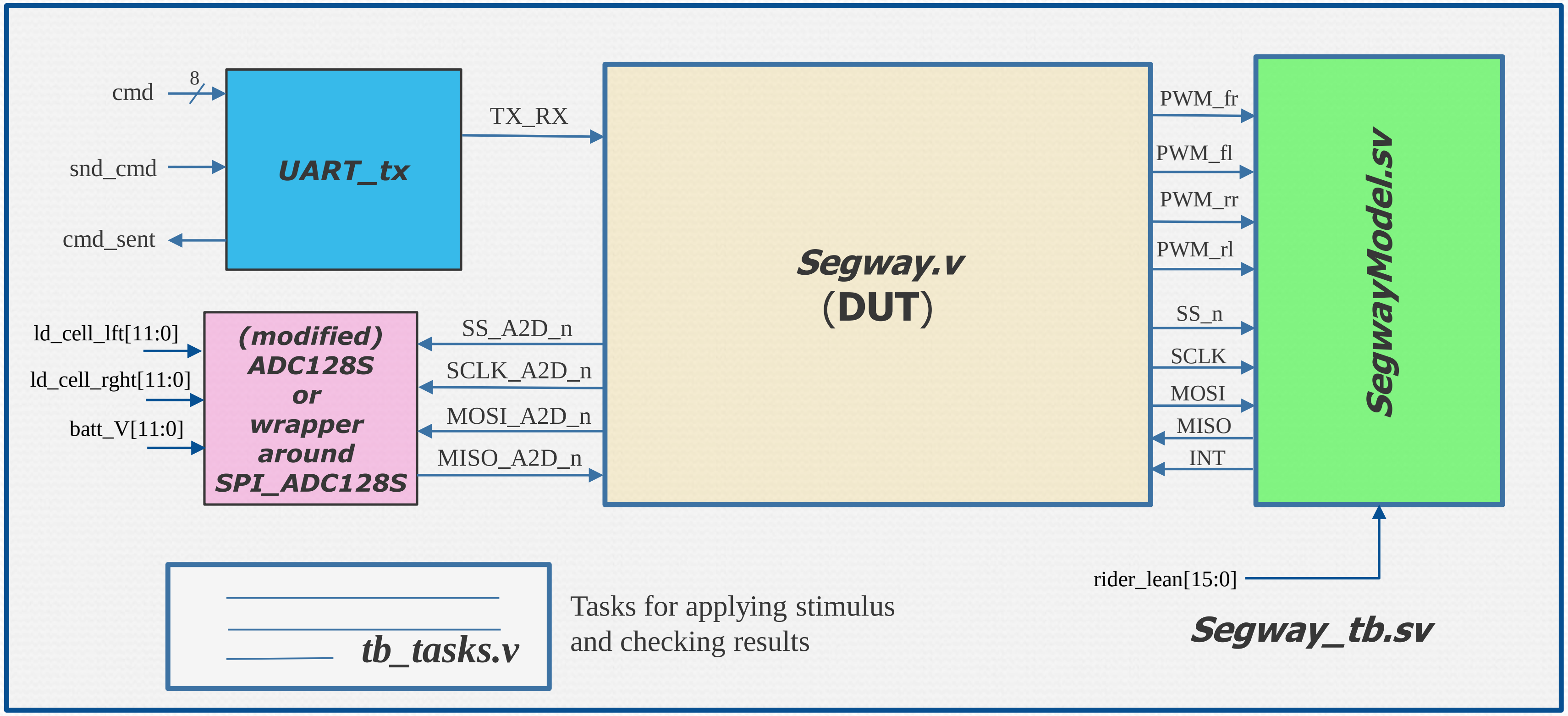
Tips for Testing Top Level of the Segway:



The orange box surrounds all the blocks that are your DUT (**Segway.v** *(provided)*). To test it you will need support blocks in your testbench. Something that mimics sending commands 'g' and 's'. Something that acts like the A2D Converter and provides load cell and battery info. And most importantly something that can read the control signals to the motors and model the "physics" of the quadcopter and provide corresponding inertial readings. This last block is provided and is called **SegwayModel.sv**.

Tips for Testing Top Level of the Segway:



A testbench shell (**Segway_tb.sv**) is provided. Its use is optional, but recommended. **SegwayModel.sv** is a model, not only of the inertial sensor, but also of the physics of the Segway. It monitors the PWM signals and does a crude model of pitch, angular acceleration/velocity/position, including modeling wheel rotation (omega_lft/omega_rght)

Tips for Testing top Level of the Segway:

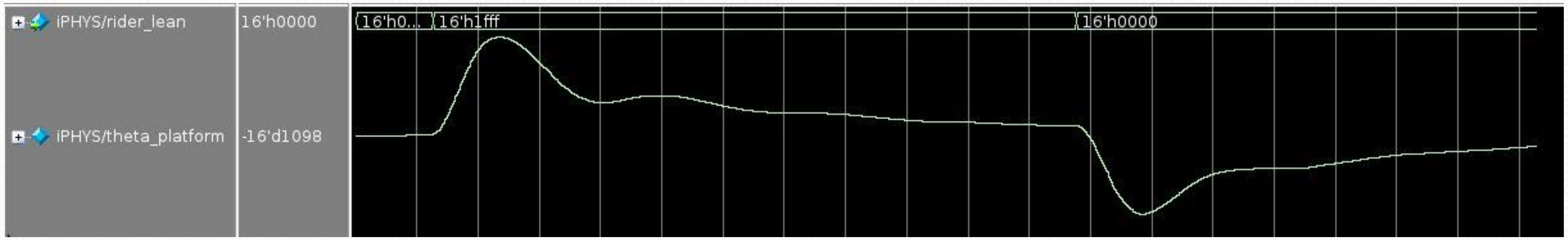
- Use of Tasks can be helpful to make your test bench more readable & less repetitive
  - Initialize
  - SendCmd
  - …
- What to test? *(start simple and work up to more complex)*
  - This is up to you.
  - Overall behavior of Segway is not that complex. Test its different cases
  - Fego and I will have a test suite. You want to think about what we might test.
  - One obvious test is to apply a step function of the rider's lean and see if the PID control system works.
    - Does it converge on getting theta of the platform back to zero?
    - Apply a large positive (and later negative) **rider_lean[13:0]**. What for near a million clock cycles (with **fast_sim** = 1).

What Does a "Correct" Response Look Like:

- Control systems are often tested with a step response, and looking at how the system converges.

- The job of the PID loop is to keep the tilt of the platform level (i.e. zero). There is a vector inside **SegwayModel** called **theta_platform** that represents the tilt of the platform.

- Apply a step input to **rider_lean**. **theta_platform** will be disturbed, but should converge and make its way back (slowly) toward zero.

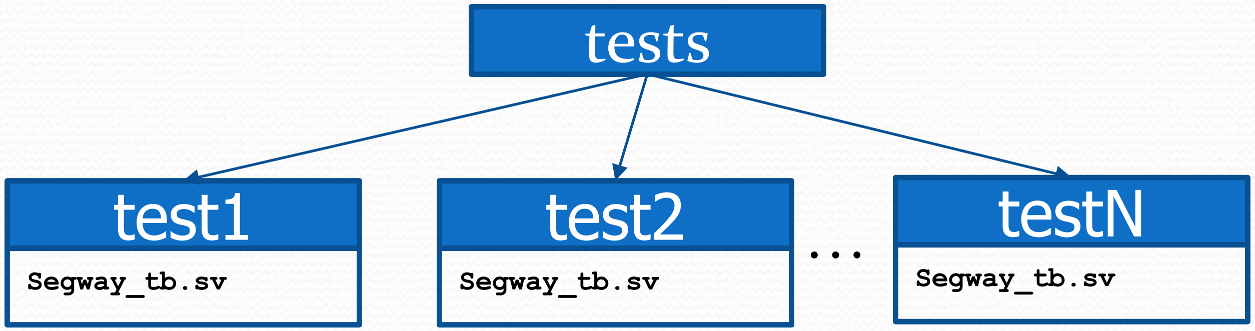


- The figure above shows the response to an abrupt step in rider_lean. Rider_lean is held at its max positive value for about 1million clock cycles (#10 high, #10 low clock in test bench).

- You can see an underdamped response and a slow convergence back toward zero. Then rider lean is abruptly set to zero.

Tips for Testing Top Level of the Segway:

- Seems like a lot to test…How to attack it.
  - Don’t try to put it all in one giant test!
  - Have a test suite
  - Perhaps a directory of tests
    - Store off a different version of Segway_tb.sv for each test
  - Could even have a python script to run the tests in batch



- Runs where you are using SegwayModel.sv to model the physics of the Segway take a long time to run.

- Remember **vsim** on Linux is much faster than ModelSim student edition.

Tips for Testing Top Level of the Segway:

- Abbreviating the Runs.
  - You should have a parameter called **fast_sim**. This should be passed to **balance_cntrl** and to **steer_en**. It is set to 1 to enable faster runs on ModelSim.
  - As part of HW5 you should have modified **balance_cntrl** to “speed up” the integral term by 16X.
  - In **steer_en** we have a 1.34 sec timer. This is far too long for simulation. **fast_sim** should shorten this timer to essentially be a 15-bit timer (14:0). But it should maintain its full effective duration if **fast_sim** = 0.
  - When we map to the DE0-Nano to test on the actual Segway we will set **fast_sim = 0.**