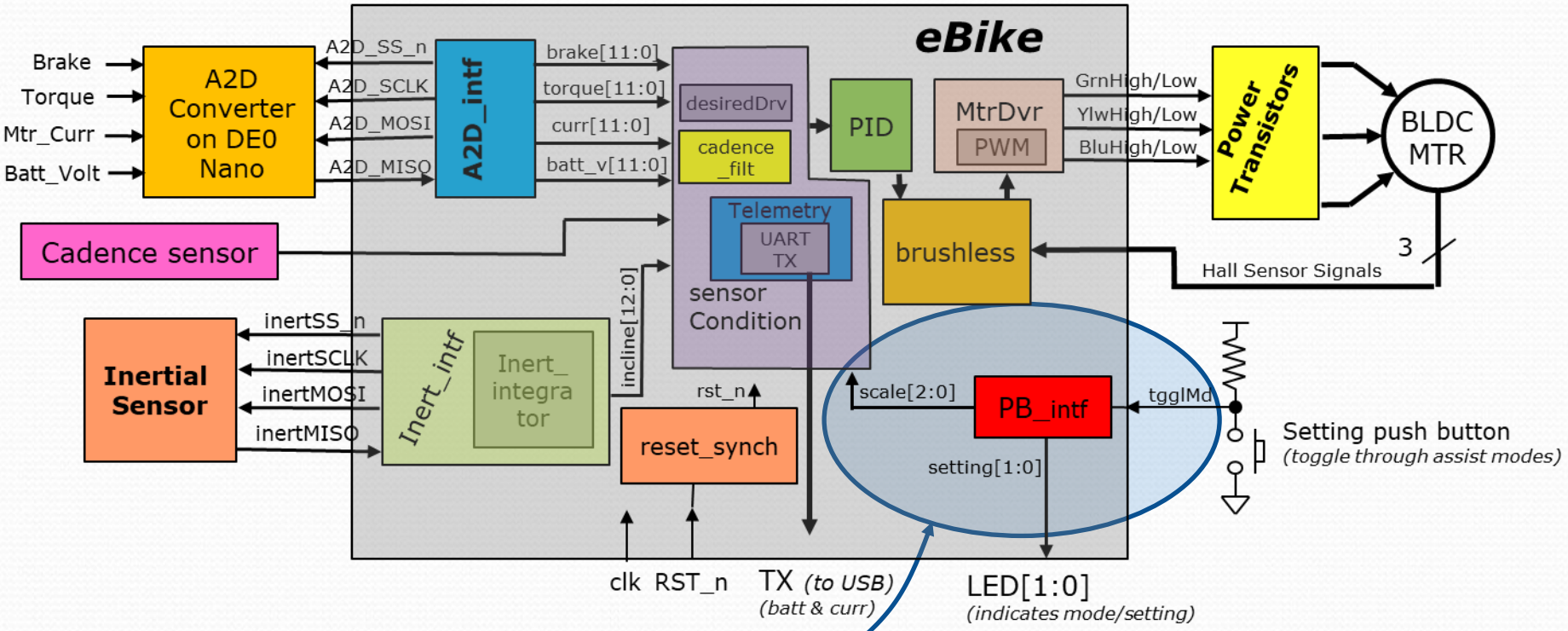
Ex23…..What Remains?

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

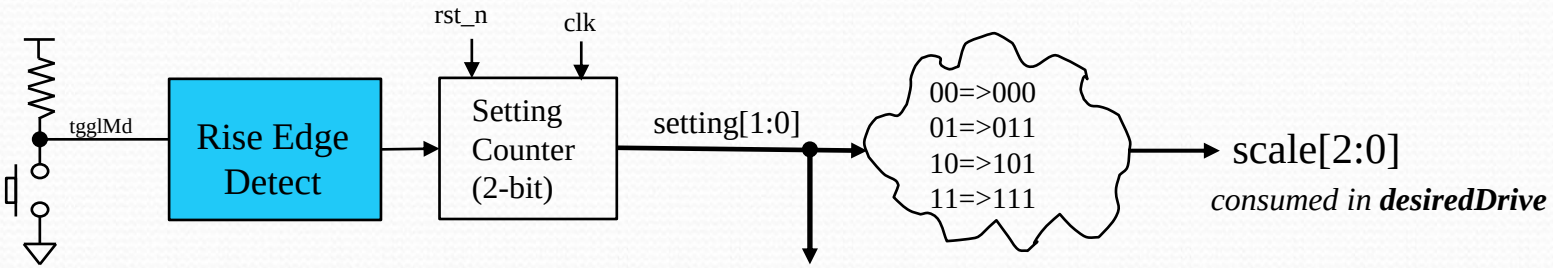
Students can find all of their course evaluations here:
https://aefis.wisc.edu/

What do you have to do next



- If you have finished all the exercises through Ex22 then the only block you should have left for the DUT is this *(which is super trivial)*
- But don't think you are done…**you have to test & synthesize this thing**

Setting Push Button Interface.



Also an output that drives 2 least significant LEDs on DE0-Nano board so user has an indication of what the setting is.

The last thing you have to make is rather trivial. There is a push button hooked to a signal called **tgglMd**. Every time it is released the 2-bit setting (00=>off, 01=>low assist, 10=>medium assist, 11=> max assist) should be incremented.

NOTE: setting should **default to 2'b10** upon reset (medium assist).

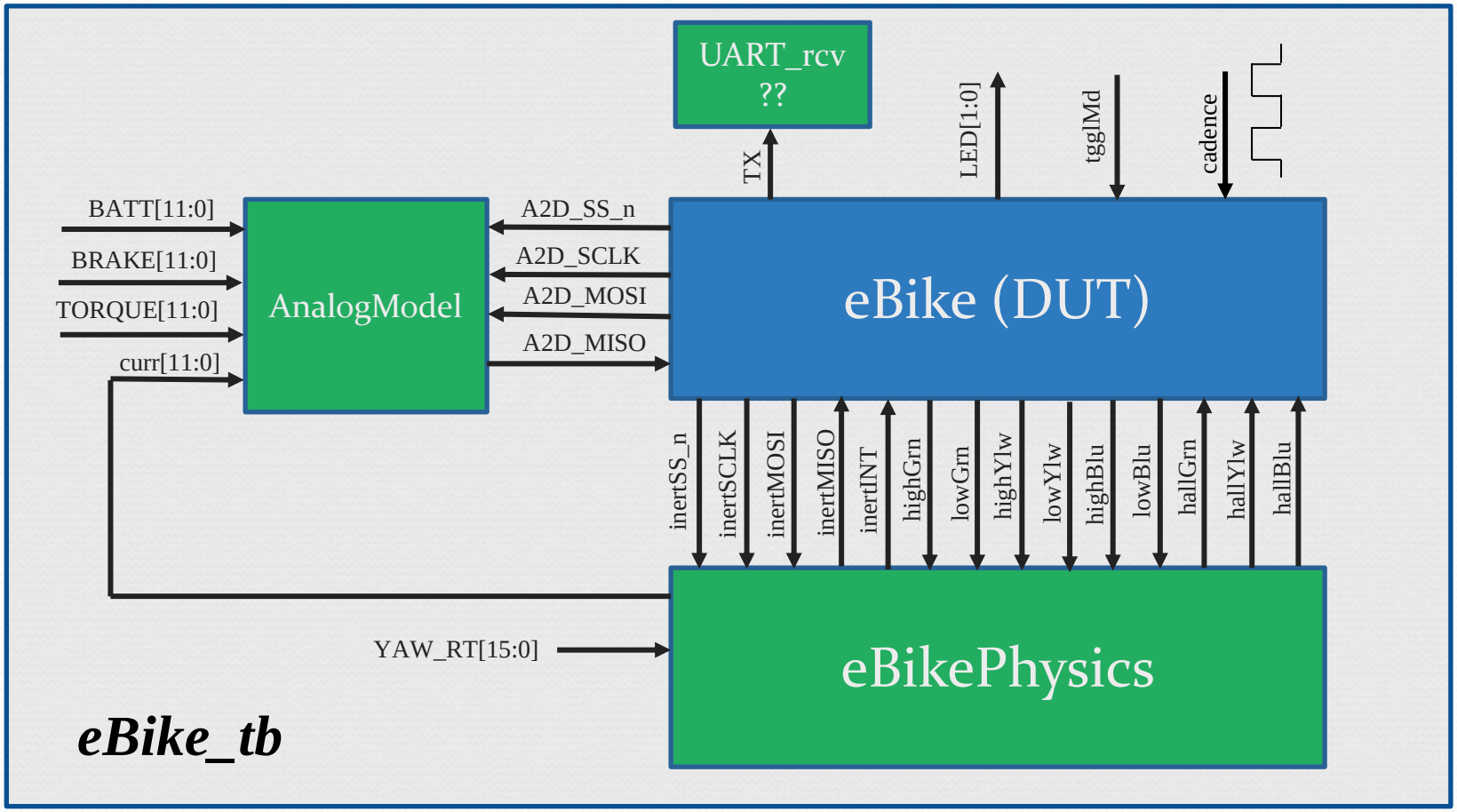
Using dataflow infer a combinational block to map the **setting** to a **scale**[2:0] factor.

Recall a pushbutton is asynch to our clock domain so the rise edge detector should have 3 flops in total.

Assembling the Top Level (eBike.sv)

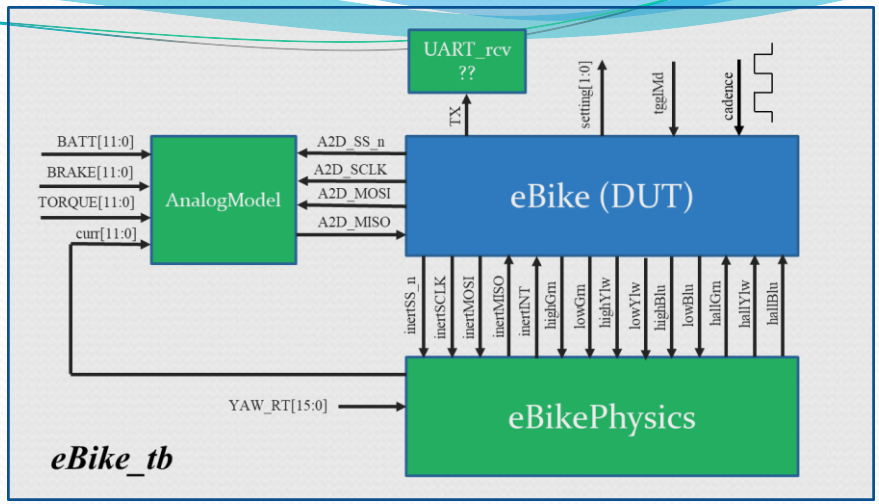
- Use the **provided eBike.sv** skeleton!!
- Follow the comments in it to instantiate the various sub-units.
- Your done…you're welcome.

Top Level Testbench (eBike_tb.sv)



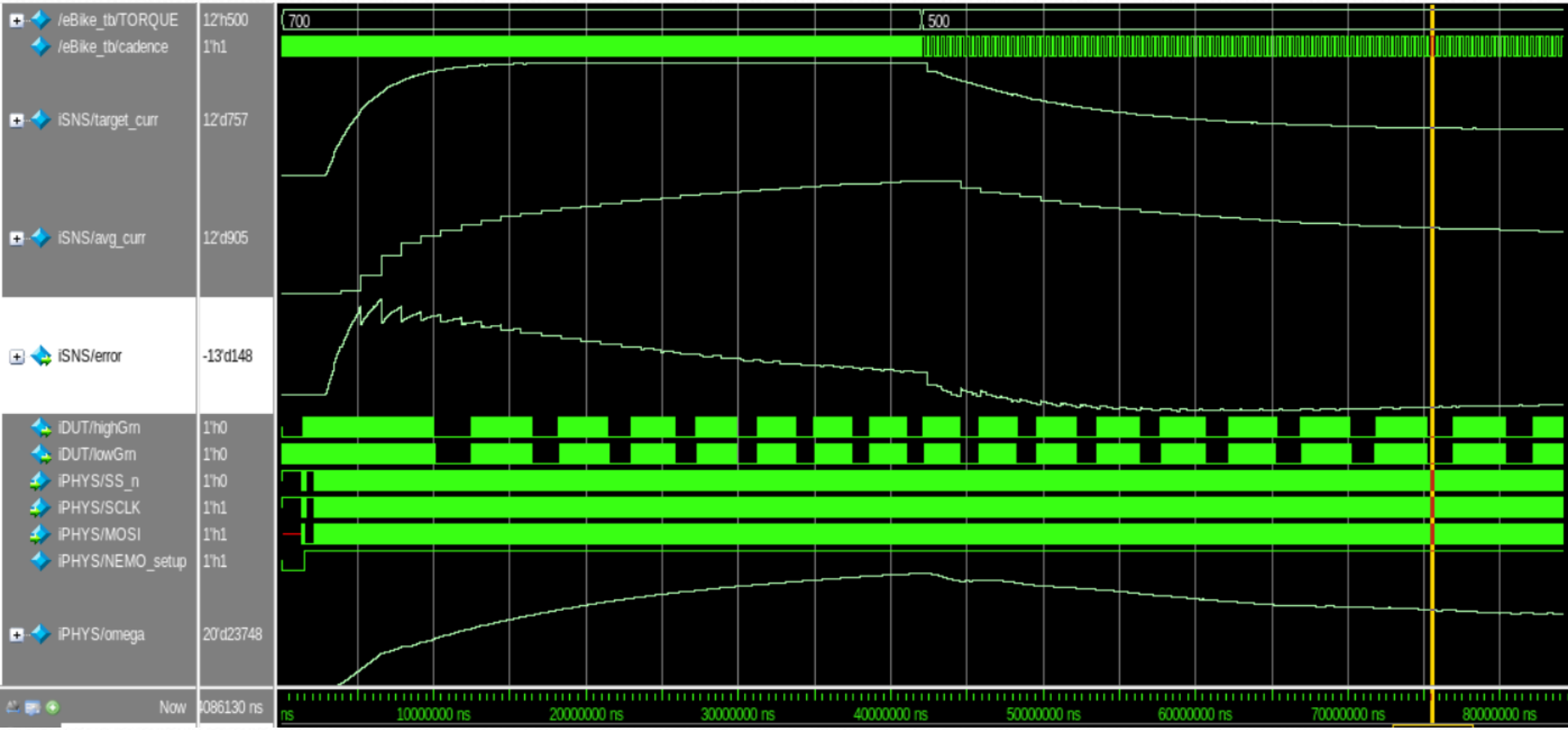
- **eBike_tb.sv** is provided as a skeleton you can use. The **UART_rcv** is not part of it, but you might want to add it.

Top Level Testbench



- ***eBikePhysics*** is a model of the hub motor and the inertial sensor.
- Provide it with the FET control PWM signals and it will return hall effect sensor readings and current the motor is consuming.
- Provide it a 16-bit reading of YAW_RT and it will model the inertial sensor so your ***inert_intf*** block can compute incline.
- **YAW_RT** represent angular rate of change (not angular position). If you apply a **YAW_RT** in the vicinity of 0x2000 for about a million clock cycles they your bike is going up a pretty steep hill.
- ***eBikePhysics*** uses your FET control PWM signals to compute the voltage across the coils. From there it calculates a net torque of the motor, and from there an angular acceleration which is integrated to form angular speed (Ω). There is a variable called omega inside ***eBikePhysics***. You can plot it.
- Angular speed (omega) is also integrated to get angular position (theta). This is modulo at 360 to give a theta that is always in the 0 to 359 range. Every 60 degrees the hall sensor outputs change.
- ***eBikePhysics*** also performs calculations to estimate the motor current. This current is fed back into the AnalogModel and your ***A2D_intf*** reads it and feed it to your PID block to close the loop.
- You can apply a positive value of TORQUE and a pulse train for cadence and plot variables as the motor starts to “turn”.

Example Top Level Simulation



Top Level Tests (on ModelSim FAST_SIM = 1)

- 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 ***eBikePhysics*** that you can use: **omega**, **torque**, **…**
  - You can self-check on some of these signals inside ***eBikePhysics***.

Why use Tasks?

- Tasks provide the ability to
  - Execute common procedures from multiple places
  - Divide large procedures into smaller ones
- Local variables can be declared & used
- Personally, I only use tasks in testbenches, but they are very handy there.
  - Break common testing routines into tasks
    - ✓Initialization tasks
    - ✓Stimulus generation tasks
    - ✓Self Checking tasks
  - Top level then becomes mainly calls to tasks.

See last 1/3 of Lecture07 on use of tasks in testbenches.

`Include Compiler Directive

- **`include** filename
  - Inserts entire contents of another file at
  - `include localparams near beginni
  - `include tasks/function near t want them to reference global signals

- Example 1:

```
module qua ();
    `inclu ams.sv"  // include params first so defined
    ...
    `include "tb_tasks.sv";
endmodule
```

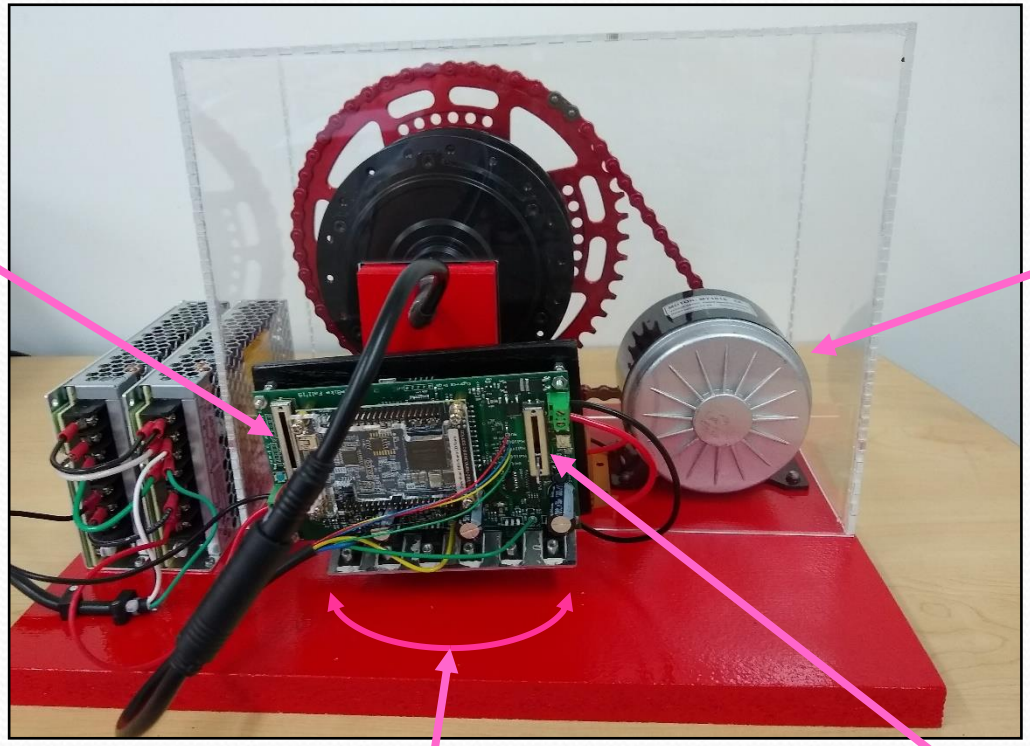
- Exa

Might want to have an `include file of handy testbench tasks *(or better yet a package)*

NOTE: has to be located Same directory as your ModelSim Project was created

Testing on the Test Platform

- .qsf and .qpf files are provided so you can map your design (via Quartus) to the test platform.
- REMEMBER to change **FAST_SIM** to false when testing on the "real thing"



This slide potentiometer on the board mimics the pedaling torque sensor

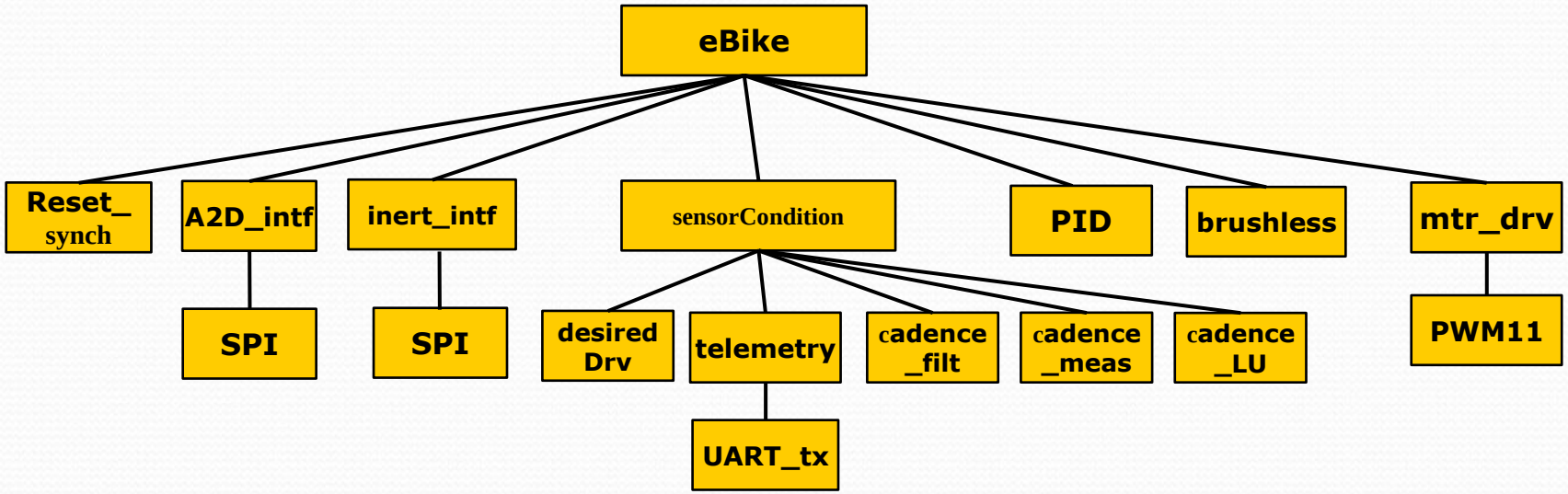
E-Bike motor is coupled (via chain) to generator. The generator serves as a mechanical load on the motor.

The whole board can tilt counter clockwise (uphill) or clockwise (downhill). Unfortunately the load on the hub motor does not vary with this, so the motor will run faster when you are going up hill.

This slide potentiometer mimics the cadence sensor

Synthesis

- You have to be able to synthesize your design at the **eBike** level of hierarchy. Write a synthesis script.



You are **NOT** allowed to use **compile_ultra**

- Your synthesis script should write out a gate level netlist of follower (eBike.vg).
- You should be able to demonstrate at least one of your tests running on this post synthesis netlist successfully.
- Timing (400MHz) is mildly challenging….some pipelining of logic will have to occur

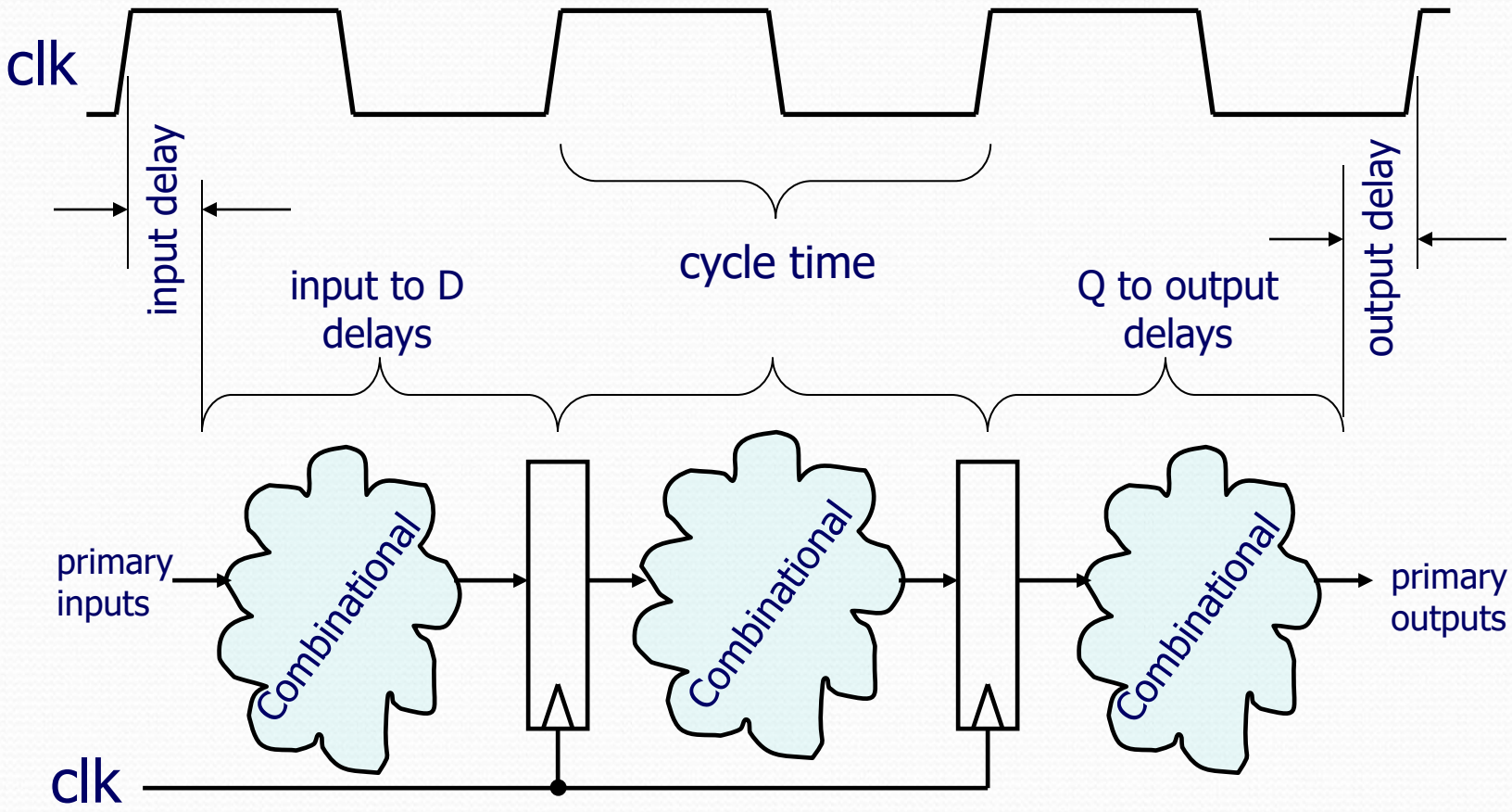
Synthesis Constraints:

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 minimum.
Input delay	0.3ns 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 NAND2X2_LVT gate from our library
Output load	50fF on all outputs
Wireload model	16000 square micron size
Max transition time	0.20ns
Clock uncertainty	0.15ns

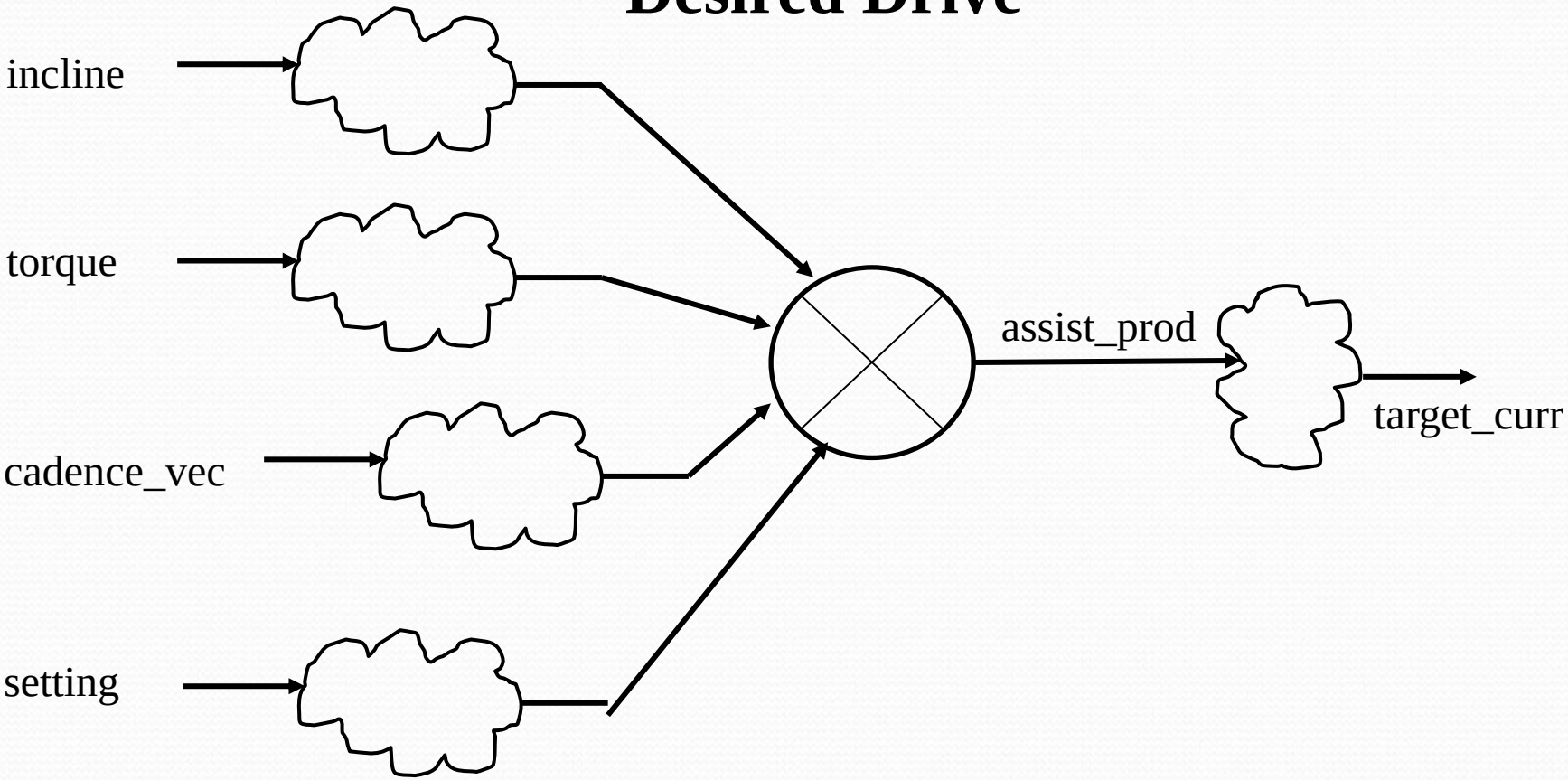
NOTE: Area should be taken after all hierarchy in the design has been smashed.
Area number to use is total area including interconnect estimate.

EricsSynthesizedArea = 14740

Timing in Synchronous Design

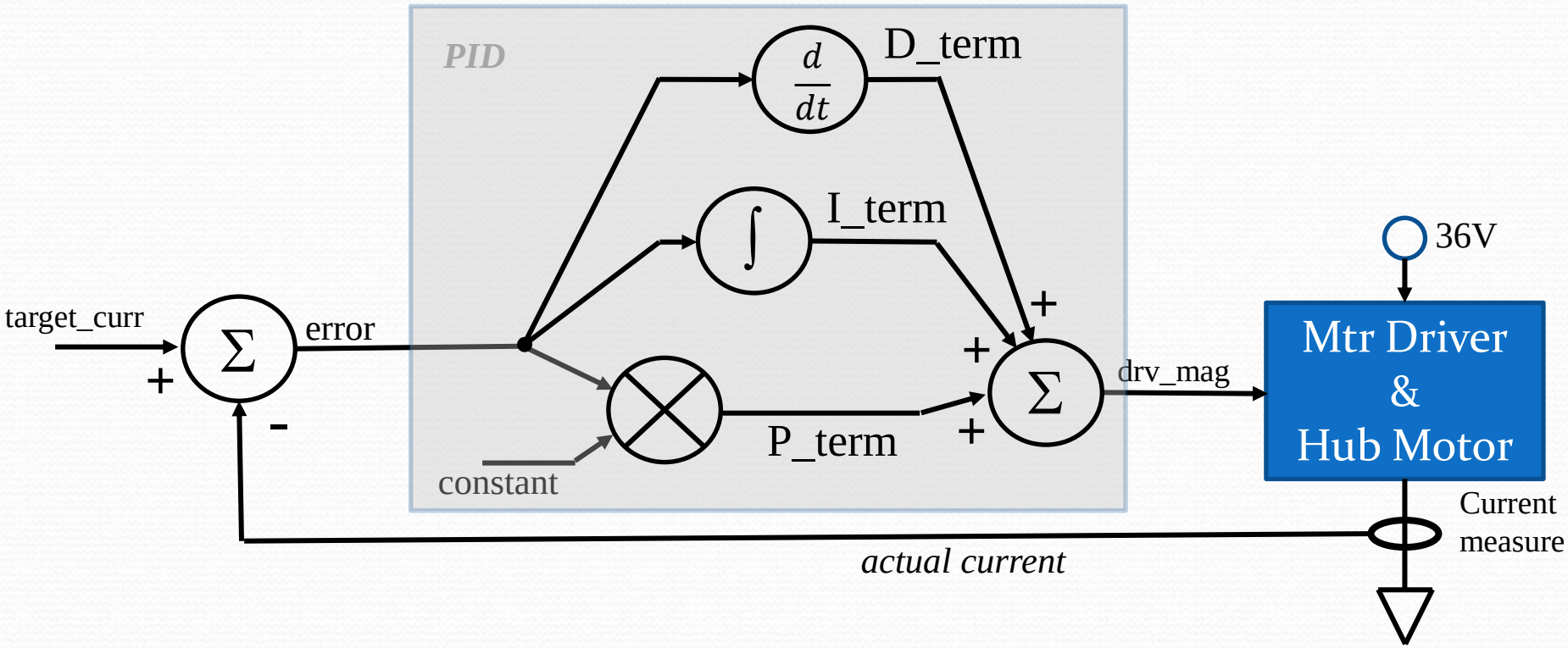


Desired Drive



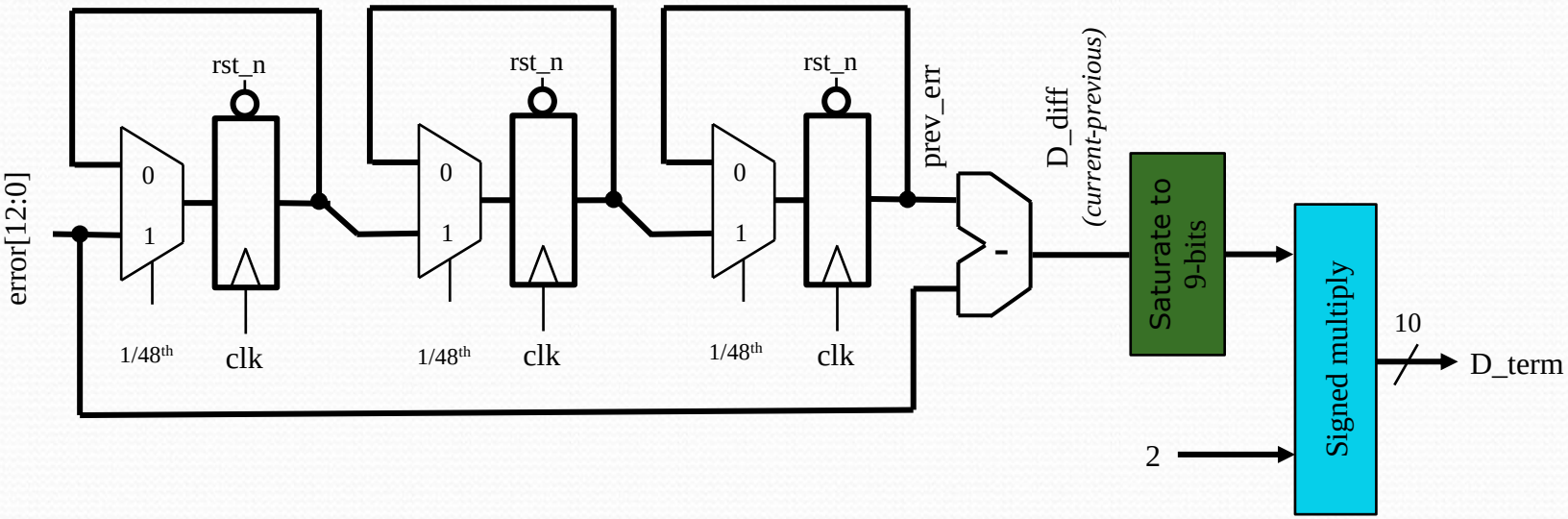
Long Pure Combinational Path

What paths from target_curr to drv_mag?

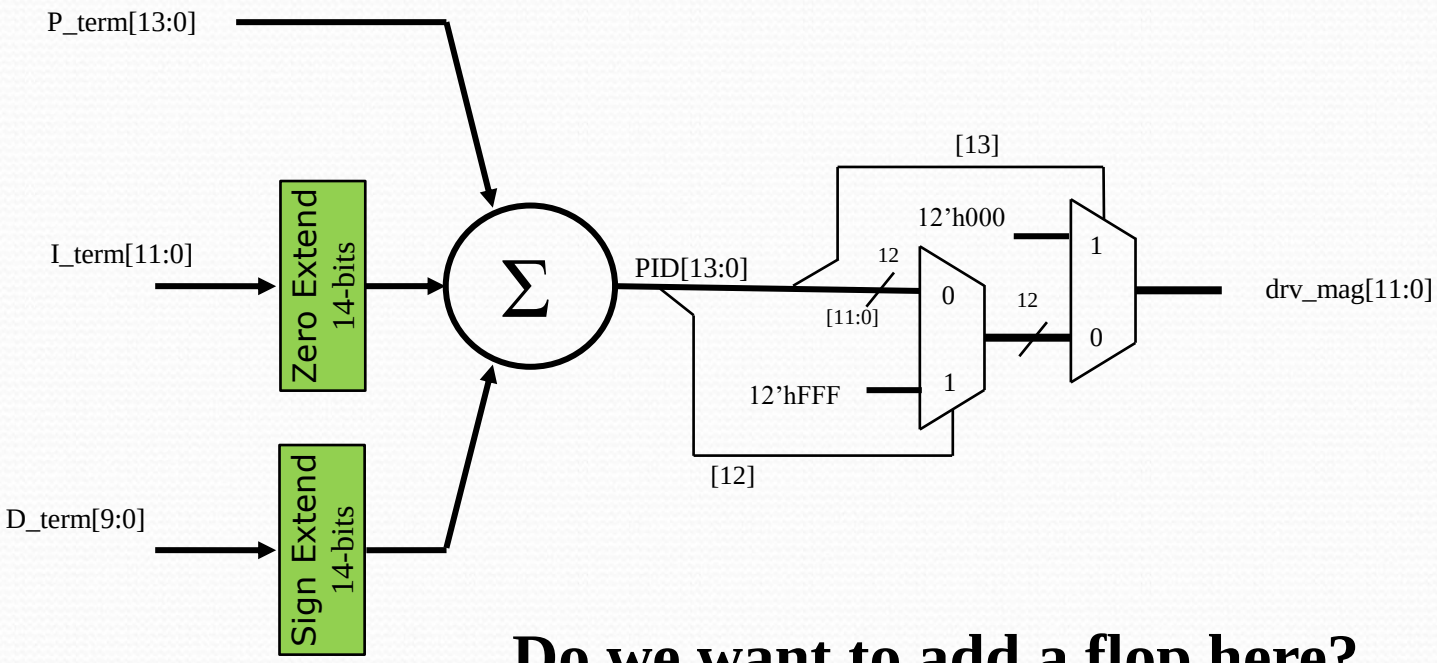


- Integration is nothing more than summing over time, so this is implemented with an accumulator
- A derivative can be approximated by how much a value changed over a given period of time, so this can be implemented by keeping track of previous values of **error**, and subtracting them from the current value of **error**.

PID Controller D_term



PID Controller putting it together



Do we want to add a flop here?

Does drv_mag end at a flop, or is there more path inside mtr_drv?

Pipelining Strategy:

- Debug functionality and full chip tests with non-pipelined version
- Separately a teammate is working on synthesis
  - Get whole thing reading in and synth script working
  - Look at max delay paths
  - Experiment and document where to add pipelining flops to make timing.
- Once non-pipelined design is functionally clean and passing full chip tests you introduce the pipeling flops where your synthesis person recommended.
- Re-run functional tests to ensure addition of pipeling flops did not break anything.

Post Synthesis Simulation of eBike.vg

- You have to show us post synthesis simulation running on at least one fullchip test on your testbench.