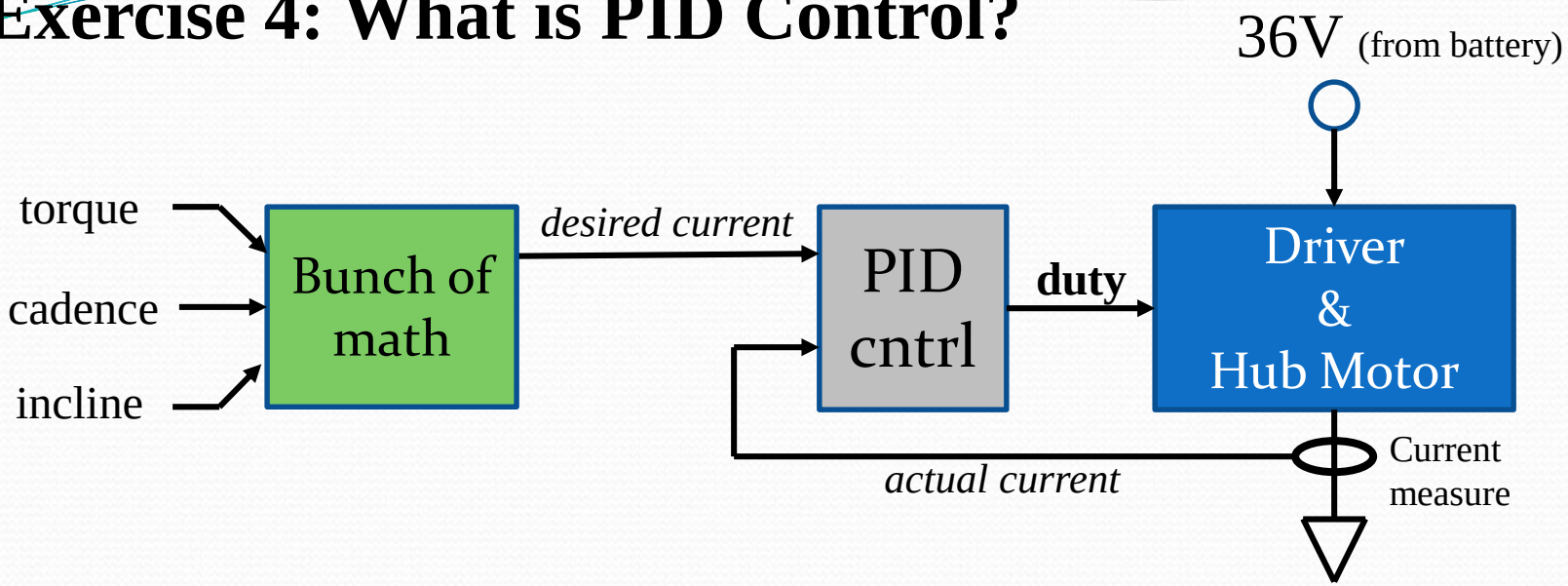
# Exercise 4: What is PID Control?

- **PID** is a very common control scheme.
- Lets take an example from our eBike project.
  - We want the motor to “amplify” the rider’s effort
    - If they put in 100W of power we will control the motor to add 100W of power.
  - Our actual control knob (duty cycle of the PWM) controls motor speed more so than motor power
  - We can measure motor current, and we know the voltage to the motor is constant, so we can measure motor power but don’t have direct control over it.

# Exercise 4: What is PID Control?

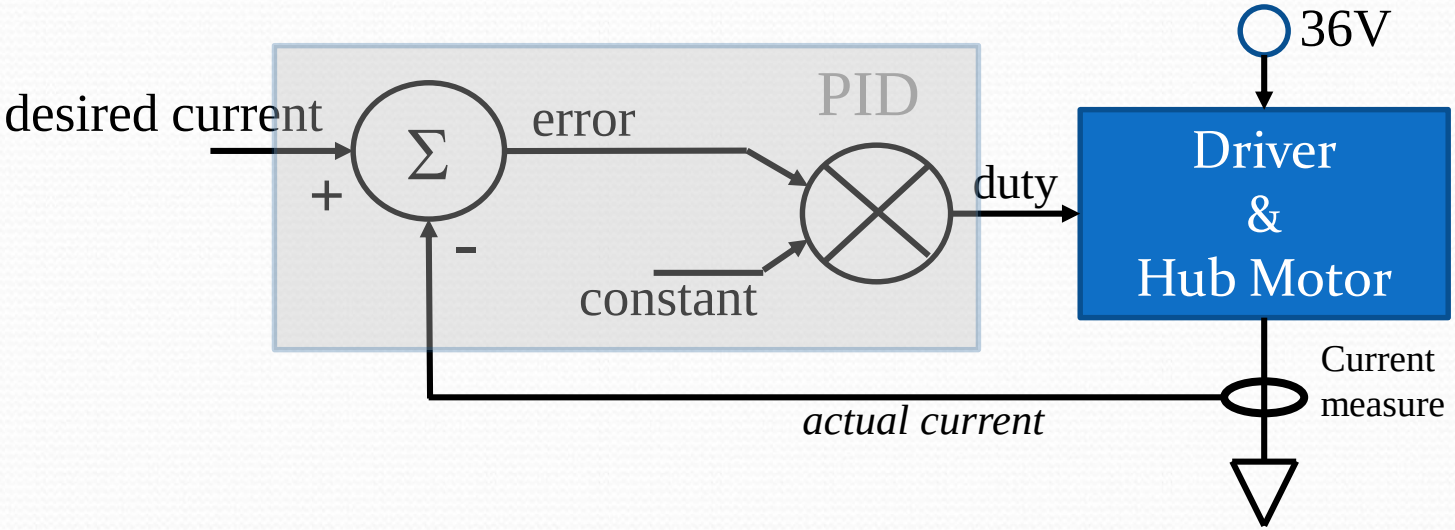



- We measure the riders effort via torque and cadence sensors in the crank shaft *(plus we decide to help them out a bit if they are going up hill)*
- We do a bunch of math on this sensor data to come up with a desired power assist level which is directly proportional to current ... so we come up with a desired current the motor should be consuming
- Now we need to "turn our knob" (duty cycle of PWM) until the motor's actual current matches the desired current. This is the job of the PID block.

# Exercise 4: What is PID Control?

- All **PID** controllers work from an "error term" which is the "desired" thing minus the "measured" thing. In our case that is:

  **error = "desired current" – "actual current"**

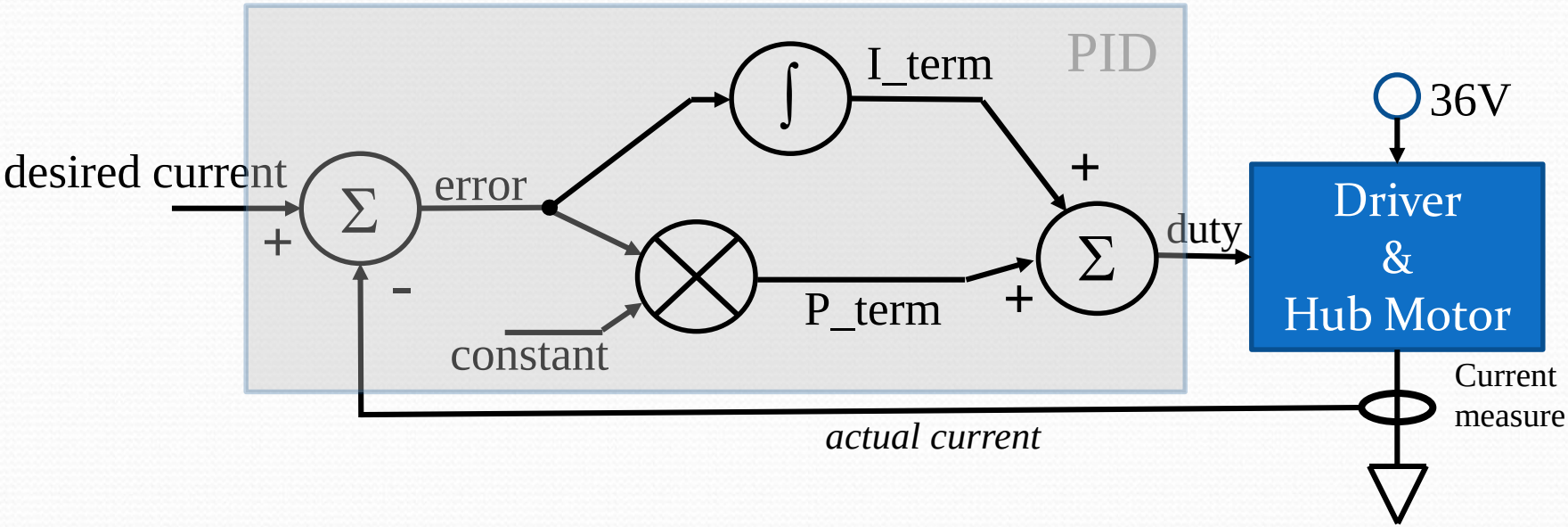



- Since **error** is **desired – actual** then if the error is larger we want the motor driven stronger. What if **duty** was simply **error** scaled up (multiplied) by a constant? Would not this do what we want?
- This is in fact the **P** in **PID**. The control is **p**roportional to the error

# Exercise 4: PID Control (P alone is not enough)

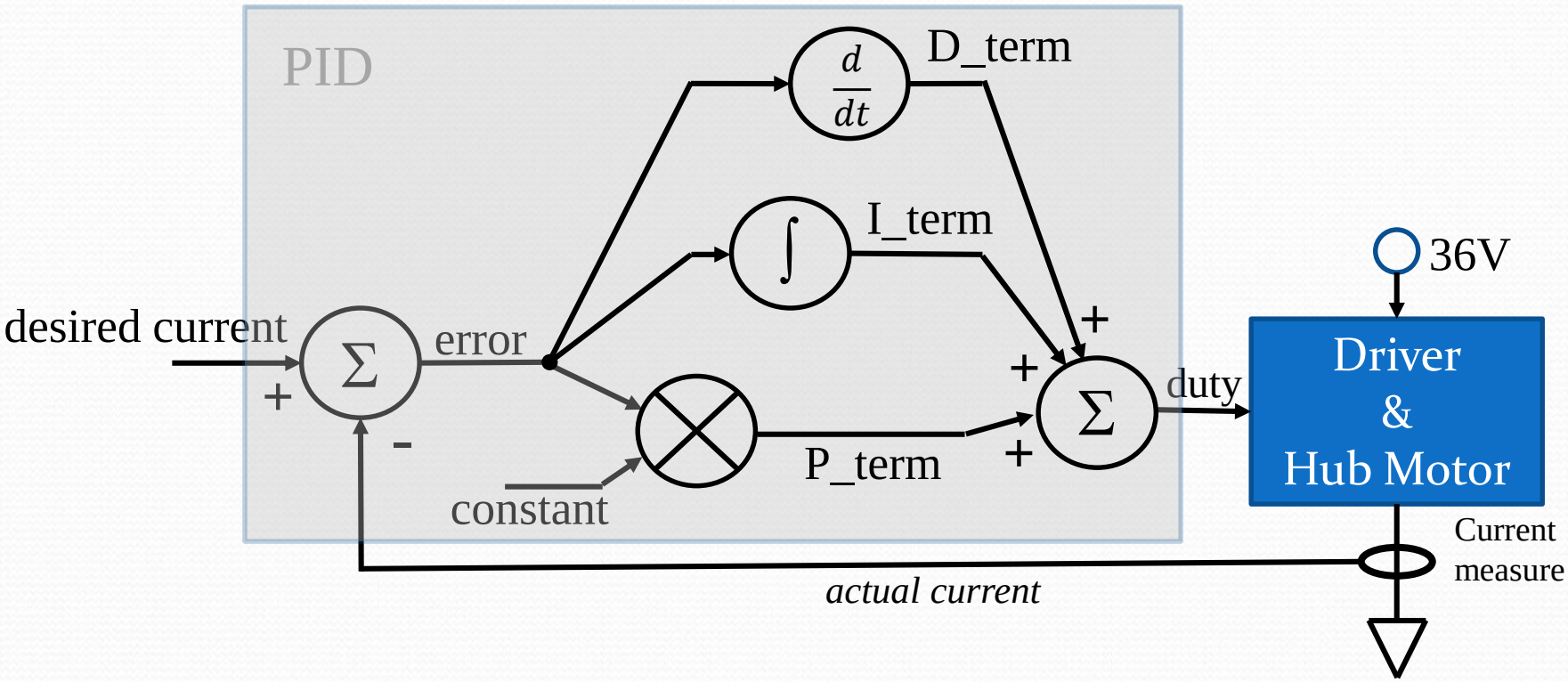
- Imagine the desired current is 3A. Further imagine to achieve this the **duty** cycle of the PWM to the motor would have to be 40%.
- **error = desired – actual**
- You want **error** to be zero right?
- **duty** = **contant** * **error**
- How can **duty** be the required 40% if the error has been driven to zero? Any constant times zero (**error**) is zero.
- Proportional control alone cannot drive **error** to zero. We need something more.
- We need to **i**ntegrate error over time and use this to drive the error to zero

# Exercise 4: PID Control (I is for Integral)




- If **error** is positive and non-zero then the **I_term** ramps up. As the error approaches zero the P_term is backing off and approaches zero.
- In our example the **I_term** ramped up to 40% to maintain **duty** at 40% even after **error** had been driven to zero.
- Many systems have lag. The response of the system is not immediate to a change in the control variable. In which case by the time **error** hits zero the **I_term** might have ramped up too far…now for the **D_term**

# Exercise 4: PID Control (D is for Derivative)




- If **error** is positive and non-zero then the **I_term** ramps up. As the error approaches zero the P_term is backing off and approaches zero.

- By using the time rate of change (derivative) of the **error** term you can keep your control scheme from developing too much "momentum" and overshooting your desired target.

# Exercise 4: PID Control (What you do)

- There are 3 Verilog files that are part of this exercise:

**plant.sv** ➔ Controls folks often refer to the system being controlled as the plant.
**PID_cntrl.sv** ➔ You will modify a few constants in this PID controller to see the effects
**PID_cntrl_tb.sv** ➔ testbench that you will run

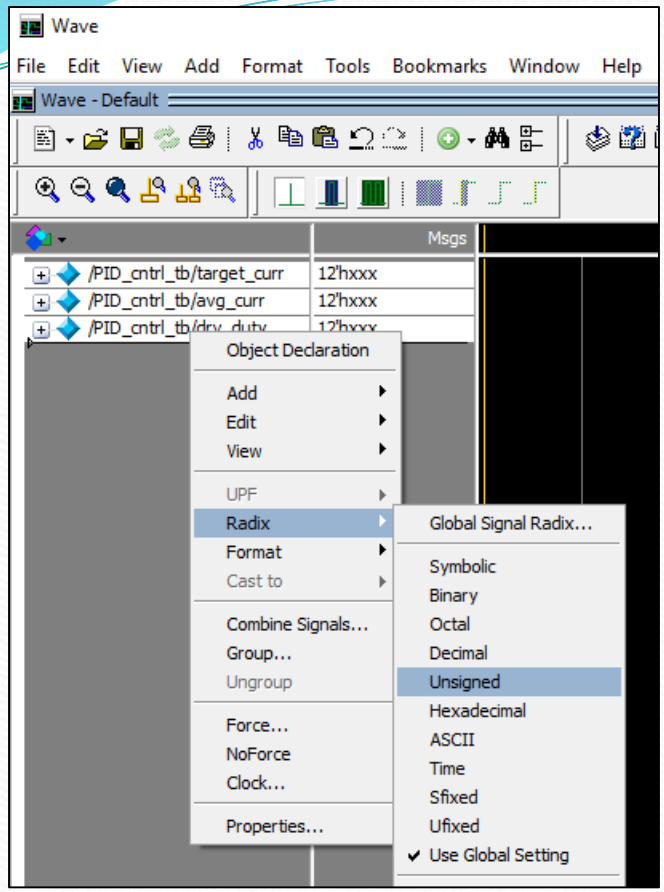
- Download these files and make a ModelSim project from them.
- Take a few minutes to study the implementation of **PID_cntrl.sv**
- Simulate the testbench (**PID_cntrl_tb.sv**)
- View the signals: **target_curr**, **avg_curr**, and **drv_duty** as analog waves
  *(see following slides for how to display waves as analog)*

# Exercise 4: PID Control (What you do)

- PID_cntrl.sv as given has zero coefficient for both **I** and **D** terms.
- What is the steady state error with just a P term?
- What happens if you increase the value of P_coeff?
- Now add in a little bit of I term (pick a small value for I_coeff)
- What do you see? (submit an screen capture of this waveform)
- Can you improve this performance? If you see some overshoot and slight oscillations they can be improved by adding in some D term.
- Submit an image of your improved response with D term.

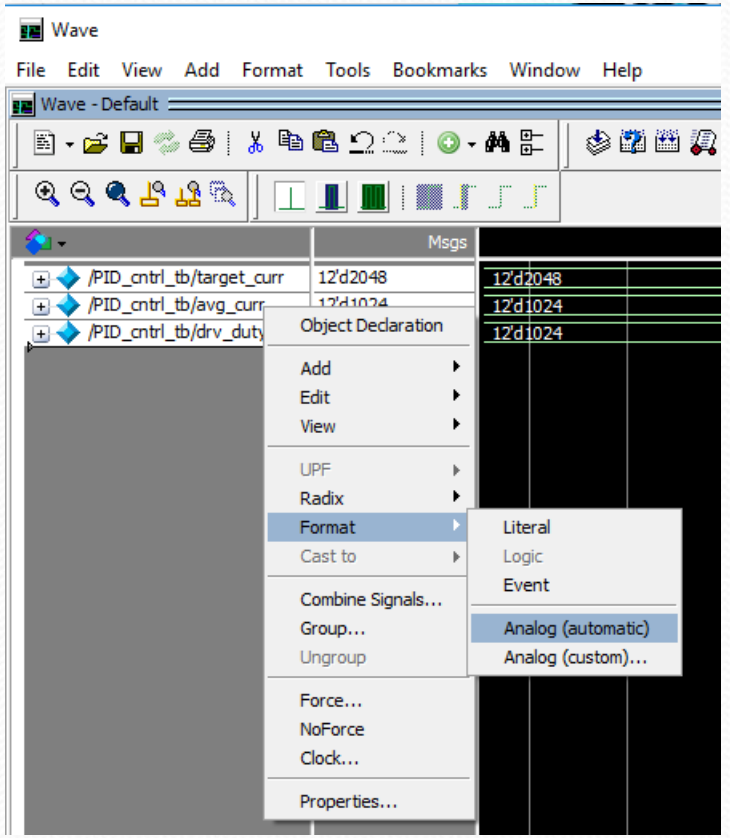
***(see next slide for viewing a waveform as analog)***

# Viewing waveforms as analog




First select the signals of interest and change the radix (right click). If the signals are signed you would choose **Signed**. These signals are unsigned so we choose **Unsigned**.

If you have not already…run the simulation




Finally right click on the signals again and change the display format to “Analog (automatic)” It can’t automatically scale signals for which it does not have data, so you have to have run the simulation to do this.