

Exercise 6 Part I (Reducing and Saturating) (HW2 prob 2):

With digital control schemes, control input decisions are made based on sensor inputs. Take our specific case of an eBike. We use an inertial sensor to measure the incline the bike is on. If it is a steep uphill we want to assist the rider more. Our sensor, however, measures a greater range of inclines than we are really interested in. By 35 degree incline we are already going to be maxing out motor drive. By a 35 degree downhill we will have the motor shut down. Thus we could possibly reduce the width of the data coming from the incline sensor and just saturate if it is too large or too negative.

By reducing to a smaller number of bits we can make down stream calculations narrower (fewer bits wide) and thus use fewer gates. This is something we will do in the control scheme of the eBike. OK...now that I have setup the motivation for the problem. Here is the problem:

You are going to write a dataflow Verilog module with the following interface:

Exercise 6 Part I (Reducing and Saturating)

Signal:	Dir:	Description:
incline[12:0]	in	13-bit signed incline from inertial sensor
incline_sat[9:0]	out	10-bit saturated (signed) result

- There are some **hints on saturating** presented on the next slide.
- The DUT should be in a file called **`incline_sat.v`**
- The testbench should test several values (saturating both directions and not saturating) and be self checking. It should be in a file called **`incline_sat_tb.v`**
- That completes Part I of Exercise 6. Do you still have time on this fine Wednesday? If so move on to Part II.

Hints on Saturating:

- Lets take a look at some examples saturating an 8-bit signed number to a 5-bit signed number.

Lets say you had the 8-bit number: **01010110**

You know it is positive because the MSB is 0

What is the greatest positive number we can represent in 5-bits? ... **01111**

Is **01010110** greater than **01111**? Yes. How do we know? Because it is positive and it has bit(s) in the [6:4] range set. So we saturate to **01111**

Lets say you had the 8-bit number: **11010110**

You know it is negative because the MSB is 1

What is the greatest negative number we can represent in 5-bits? ... **10000**

Is **11010110** more negative than **10000**? Yes. How do we know? Because it is negative and it has bit(s) in the [6:4] range clear. So we saturate to **10000**

Lets say you had the 8-bit number: **11110110**

You know it is negative because the MSB is 1

What is the greatest negative number we can represent in 5-bits? ... **10000**

Is **11110110** more negative than **10000**? No. How do we know? Because it is negative but all the bits in the [6:4] are also set. So we simply keep bits [4:0] as our 5-bit result.

(more on next slide)

Hints on Saturating:

- So..from the previous slide you can derive the following pseudo code:

```
if ((number is negative) &&  
    (any upper bits (in certain range) are zero)) {  
    saturate to most negative number 1000000000  
}  
else if ((number is positive) &&  
    (any upper bits (in certain range) are one)) {  
    saturate to most positive number  
}  
else {  
    number not too positive so just copy over lower bits  
}
```

Of course this is **dataflow** Verilog, so we are NOT using **if**. Instead use tertiary assign statements.

What is an effective way to tell if any bits in a range are 1 or 0? Think about the reduction operator.

Exercise 6 Part II (Math on sensor inputs to get desired drive)

You are creating a module called **desiredDrive.v**. It has the following interface and should be done completely in dataflow verilog.

Signal:	Dir:	Description:
avg_torque[11:0]	in	Unsigned number representing the torque the rider is putting on the cranks (force of their pedaling)
cadence_vec[4:0]	in	Unsigned number representing the speed of the rider's pedaling.
incline[12:0]	in	Signed number (you just dealt with this in Part I)
setting[1:0]	in	unsigned. Represents level of assist motor provides 11 => a lot of assist, 10 => medium, 01 => little
target_curr[11:0]	out	Unsigned output setting the target current the motor should be running at. This will go to the PID controller to eventually form the duty cycle the motor driver is run at.

(see following slides for math details)

Exercise 6 Part II (Math on sensor inputs to get desired drive)

Still need to

copy this

- First step is what you already did. The 13-bit signed **incline** signal should be saturated to a 10-bit signed value (**incline_sat**)
- Next create a signal called **incline_factor** that is simply `incline_sat + 256`. How wide should that signal be? Is it capable of being negative?
- Next you will limit **incline_factor** and create a new 9-bit saturated signal that is clipped with respect to negative values. Meaning if **incline_factor** was negative it will be clipped to zero. If **incline_factor** was > 511 it will be saturated to 511. This new signal should be called **incline_lim** (limited).
- A 5-bit input called **cadence_vec** provides the rider's cadence (rate of pedaling). If the rider is not pedaling at all (or pedaling really, really slow) we shut down drive. You will create a 5-bit signal called **cadence_factor** that is zero if (**cadence_vec** \leq 5'b00001).

(continued)

Exercise 6 Part II (Math on sensor inputs to get desired drive)

- If **cadence_vec** is greater than 1 then **cadence_factor** should simply be **cadence_vec + 32**. How wide should **cadence_factor** be?
- **avg_torque[11:0]** is an unsigned number representing the force of the pedaling. The torque sensor, however, has a significant offset that needs to be subtracted out. Create a signal (called **torque_off**) that is **avg_torque** minus **TORQUE_MIN**. **TORQUE_MIN** should be a **localparam** and set to 12'h380. How wide does **torque_off** need to be?
- Ask yourself. Can the above **torque_off** reading be negative? Answer is there will be variation in offsets between different torque sensors. So perhaps a sensor could have a lower offset than **TORQUE_MIN**, in which case the above subtraction of **TORQUE_MIN** could produce a slightly negative value. We really want to treat torque as an unsigned number, so this negative value would look like a huge positive torque. We cannot have this.

(continued)

Exercise 6 Part II (Math on sensor inputs to get desired drive)

- Create a new signal (called **torque_pos**) that is just a zero clipped version of **torque_off**. Meaning, if **torque_off** is negative it assigns it to zero, otherwise it is simply a copy of the lower N-1 bits of **torque_off**.
- **setting** is a 2-bit input that can be thought of as an (off,low,medium,high) setting for the controller. 00=>off (no assist), 01=>low (little assist), 10=>medium, 11=>high (most assist to rider from motor).
- Create a new signal (called **assist_prod**) which will represent how much we wish the motor to assist the rider. This will simply be a product of: **torque_pos**, **incline_lim**, **cadence_factor**, and **setting**. How wide does this signal need to be? (OK...I'll tell you...29-bits, but make sure that jives with your thinking)
- Finally implement **target_curr** (the output that will eventually go to the PID controller). If any of bits [28:26] of **assist_prod** are set then we set this signal to 12'hFFF otherwise we set it to **assist_prod[25:14]**.

(that's it for implementation next we test/debug)

0001/1111/0010/0010/

Exercise 6 Part III (Testing of desiredDrive.v)

- Create a testbench and apply the input (orange) values shown in the following table. Some intermediate values (blue) are shown, and the expected output (green).

0001/1110/1111/0000

avg_torque	cadence	incline	setting	torque_pos	incline_lim	target_curr
0x800	0x10	0x0150	0b10	0x480	0x1FF	0xD79
0x800	0x10	0x1F22	0b11	0x480	0x022	0x158
0x360	0x10	0x0C0	0b11	0x000	0x1C0	0x000
0x800	0x18	0x1EF0	0b11	0x480	0x000	0x000
0x7E0	0x18	0x0000	0b11	0x460	0x100	0xB7C
0xF00	0x18	0x0000	0b11	0x460	0x180	0xFFF

- Your testbench should be self checking against this stimulus/expected response. Submit **desiredDrive_tb.sv** to the dropbox.

