Exercise 7 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 7 Part I (Reducing and Saturating)

Signal:	Dir:	Description:
incline[12:0]	in	13-bit signed incline from inertial senor
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.sv**
- 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.sv**

- That completes Part I of Exercise 7.
  - Submit: **incline_sat.sv** & **incline_sat_tb.sv**

- Do you still have time on this fine Friday? 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
}
else if ((number is positive) &&
          (any upper bits (in certain range) are one)) {
    saturate to most positive number
}
else {
    number not too negative or 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 7 Part II (Math on sensor inputs to get desired drive)

You are creating a module called **desiredDrive.sv**. 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 toque the rider is putting on the cranks (force of their pedaling)
cadence[4:0]	in	Unsigned number proportional to the sqrt the speed of the rider's pedaling.
not_pedaling	in	Asserts if cadence is so slow it has been determined rider is not pedaling.
incline[12:0]	in	Signed number (you just delt with this in Part I)
scale[2:0]	in	unsigned. Represents level of assist motor provides 111 => a lot of assist, 101 => medium, 011 => little, 000 => no assist
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 7 Part II (Math on sensor inputs to get desired drive)

- 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 a sign extended (to 11-bits) **incline_sat** + 256. 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** provides a signal proportional to the rider's cadence *(rate of pedaling)*. It is also accompanied by a signal called **not_pedaling** that is asserted if the **cadence** is too slow.

(continued)

Exercise 7 Part II (Math on sensor inputs to get desired drive)

- **avg_torque[11:0]** is an unsigned number representing the force of the pedaling. The toque 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?

- Can the above **torque_off** reading be negative? There will be variation in offsets between different torque sensors. So perhaps a sensor could have a lower offset then TORQUE_MIN, in which case the above subtraction of TORQUE_MIN could produce a slightly negative value. **torque_off** should be 13-bits wide and formed from the subtraction of two zero extended 12-bit quantities (**avg_torque** & TORQUE_MIN).

- We treat torque as an unsigned number *(you only pedal forward)* , however, a slightly negative value *(resulting from an overflow of avg_torque – TORQUE_MIN)* would look like a huge positive torque. We cannot have this.

(continued)

Exercise 7 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.**

- **scale** is a 3-bit input that can be thought of as an (off, low, medium, high) setting for the controller. 000=>off (no assist), 011=>low (little assist), 101=>medium, 111=>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. When pedaling this will simply be a product of: **torque_pos**, **incline_lim**, **cadence**, and **scale**. When **not_pedaling** it is zero. How wide does this signal need to be? (OK…I'll tell you…30-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 [29:27] of **assist_prod** are set then we set this signal to 12'hFFF otherwise we set it to **assist_prod[26:15]**.

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

Exercise 7 Part III (Testing of desiredDrive.sv)

- 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).

avg_torque	cadence	incline	scale	not_ pedaling	torque_pos	incline_lim	target_curr
0x800	0x10	0x0150	0x3	0	0x480	0x1FF	0xA1A
0x800	0x10	0x1F22	0x5	0	0x480	0x022	0x11E
0x360	0x10	0x0C0	0x5	0	0x000	0x1C0	0x000
0x800	0x18	0x1EF0	0x5	0	0x480	0x000	0x000
0x7E0	0x18	0x0000	0x7	0	0x460	0x100	0xD66
0x7E0	0x18	0x0080	0x7	0	0x460	0x180	0xFFF
0x7E0	0x18	0x0080	0x7	1	0x460	0x180	0x000

- Your testbench should be self checking against this stimulus/expected response. **Submit** both the DUT and testbench to the dropbox.