## Exercise 6 (Reducing and Saturating) (HW2 prob 1):

With digital control schemes. control input decisions are made based on error terms. Take our specific case of "Segway" balance control. If the platform is pitching forward more than desired we want to drive the motors harder forward. If this pitch error is small we want to make a slight adjustment *(we want good granularity).* If this pitch error is large we want to make a large adjustment. This implies we need a lot of bits to store the error, so we can have both fine granularity and large range. However, in control systems there gets to be a point where we don't care how large the error is we just need to know it is large and the system should do as much as it can to correct it. If the "Segway" platform is pitched forward way more than we want it to, we no longer care exactly how much, at that point we know we want to drive the motors forward really hard.

So a wide error term could be reduced and saturated to a smaller number of bits to make down stream calculations narrower *(fewer bits wide)* and thus use fewer gates. This is something we will do in the balance controller math. 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 (Reducing and Saturating) (HW2 prob 1):

| Signal: | Dir: | Description: |
|---|---|---|
| unsigned_err[15:0] | in | 16-bit **unsigned** error term to be reduced/saturated to 10-bits |
| unsigned_err_sat[9:0] | out | 10-bit saturated version of unsigned_err[15:0]. How would unsigned saturation to a 10-bit value work? |
| signed_err[15:0] | in | 16-bit **signed** number to be reduced/saturated to 10-bits. |
| signed_err_sat[9:0] | out | 10-bit saturated version of signed_err[15:0]. If result is positive and greater than 0x1FF then it saturates to 0x1FF. If result is more negative than 0x200 then it saturates to 0x200 |
| signed_D_diff[9:0] | in | 10-bit signed version of a difference term used for derivative calculation |
| signed_D_diff_sat[6:0] | out | 7-bit signed saturated version of signed_D_diff[9:0] |

The name of the DUT should be ***saturate.sv***

The name of the testbench should be ***saturate_tb.sv*** *(should be self checking)*

## **Exercise 6 (Reducing and Saturating) (HW2 prob 1)**:

Write the Verilog for the DUT (***saturate.sv***) and the testbench (***saturate_tb.sv***) Apply several values to each input and observe the saturation of the outputs. You should choose several values that would cause the signed value to saturate in various ways, and also values that would not cause saturation. Testbench should be self checking *(compare values with known correct values)*

**Submit to the Exercise06 dropbox** as much of this as you have done by the end of class. You will have to complete and submit the requested files for HW2.

AI/LLM is likely to produce a ternary with a straight comparison of the number vs upper limit magnitudes. This is not always the most efficient synthesis results *(more gates)*. See following for how I would like structure of code.

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

Perhaps **better** pseudo code given on next page:

## Hints on Saturating: (3 branch saturation logic)

- The pseudo code on the previous slide was done with nested **if**s first checking the sign of the number, then checking if it needs to saturated or not.

- The structure below is an **if…else if…else** that is perhaps a bit more concise

```
if ((number is negative) && (too negative)) {
    saturate to most negative number
}
else if {(number is positive) && (too positive))
    saturate to most positive number
}
else {
    number not out of range so just copy over lower bits
}
```