ECE 551
HW5

- Due Mon Nov 22nd by 11:55PM
- Work Individually on Problems 1 & 2
- Rest of HW can be done as a team
  - These problems are submitted via dropbox
  - **Please**…only **one** person submit all problems for the team!

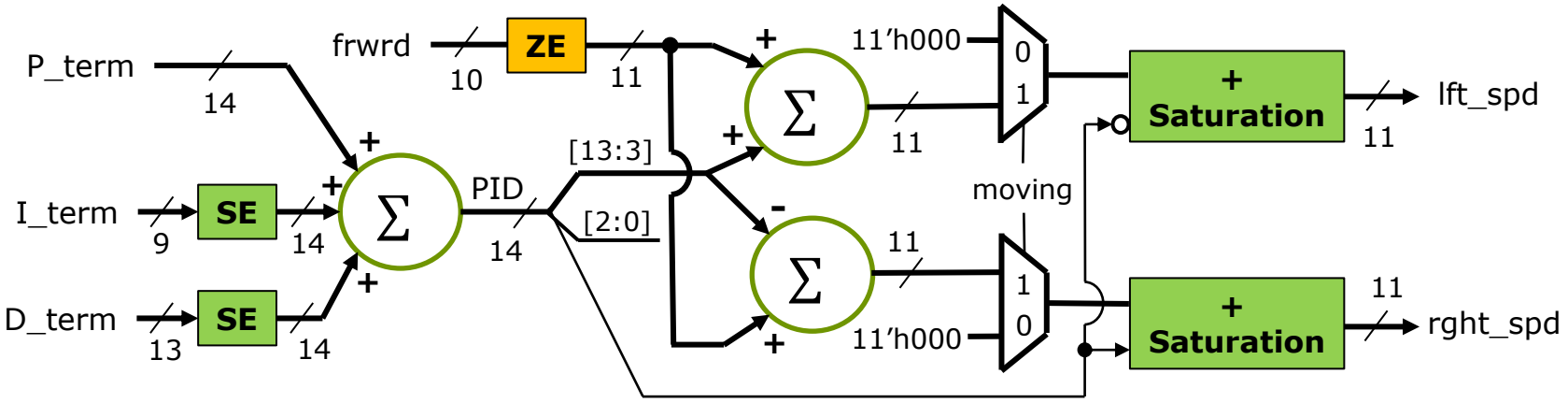
HW5 Problem 1 (This problem is **individual** basis)

Individual problem and not covered by exercise.

1. (16pts) PID creation & random testing.

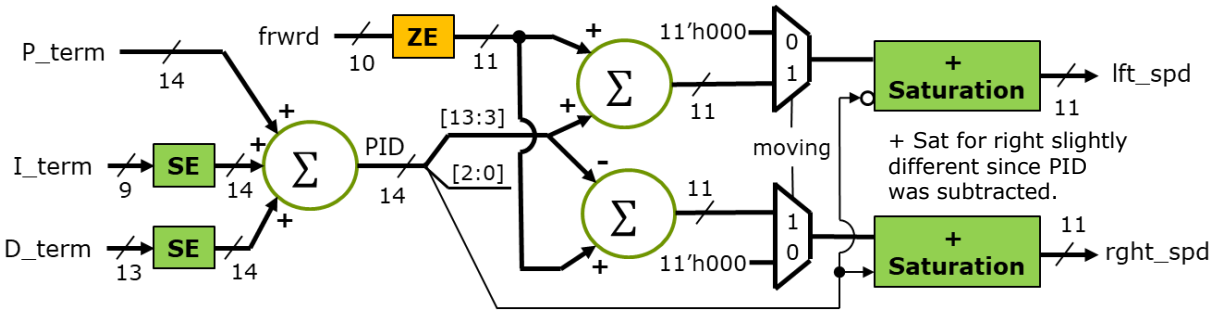
Way back in Ex06, Ex08, and Ex10 you created the P,I,and D terms of our PID block. Your first simple task is to combine these into one block. I highly recommend you **do not** do this by instantiation, but rather just copy and paste your various P,I,&D implementations into a single **PID.sv** file and create a flat solution.

Signal:	Dir:	Description:
clk,rst_n	in	You know what these are
moving	in	Clear I_term if not moving
err_vld	in	Compute I & D again when vld
error[11:0]	in	Signed error into PID
frwrd[9:0]	in	Summed with PID to form lft_spd,right_spd
lft_spd[10:0], rght_spd[10:0]	out	These form the input to ***mtr_drv***



How P_term, I_term, & D_term form lft_spd & rght_spd

HW5 Problem 1 (PID creation & random testing)



Since **frwrd** speed is only a positive number we only have to worry about + saturation. For left if **PID** positive and the sum with **frwrd** resulted in negative we need to saturate to 0x3FF.

- You will test your **PID.sv** unit using stimulus and expected response read from a file. On the website you will find **PID_stim.hex** and **PID_resp.hex**. These represent stimulus and expected response.

- For **PID_stim.hex** the vector is 25 bits wide and is assigned as follows:

Stimulus Bit Range:	Signal Assignment:
stim[24]	rst_n
stim[23]	moving
stim[22]	err_vld
stim[21:10]	error[11:0]
stim[9:0]	frwrd[9:0]

- For **PID_resp.hex** the vector is 22 bits wide and is assigned as follows:

Resp Bit Range:	Signal Assignment:
resp[21:11]	lft_spd
resp[10:0]	rght_spd

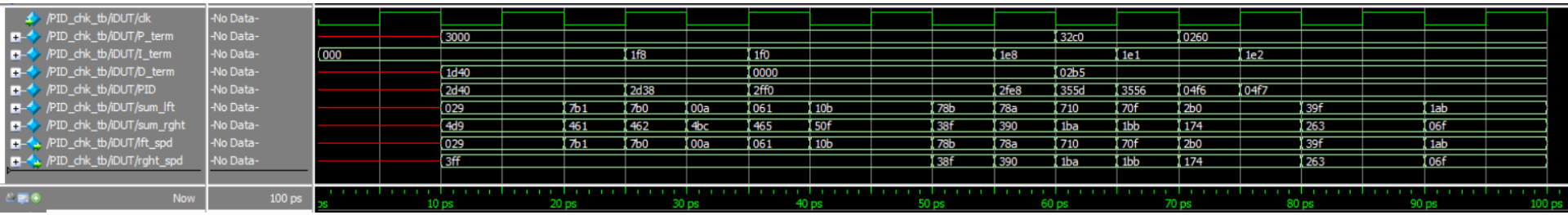
Submit:
- PID.sv
- PID_tb.sv
- Image of your transcript window

- Create **PID_tb.sv** that instantiate your PID and declares a stimulus and resp memory to hold the 2000 vectors of stim & resp. Read each provided file into its respective memory using **$readmemh.** ***(see next 2 slides)***

- Loop through the 2000 vectors and apply the stimulus vectors *(starting with clk low)* to the inputs as specified. Then wait till #1 time unit after the rise of **clk** and compare the DUT outputs to the response vector (self check). Do all 2000 vectors match? If so print a happy message to the transcript window **that includes your name.**

HW5 Problem 1 (Debug…some intermediate values)

I realize debug of random vectors can be difficult. Shown below are some of my intermediate values for the first 10 vectors. This may aid in your debug efforts.



Loading Memory Data From Files

Inserted from lecture slides for ease of reference

- This is very useful (memory modeling & testbenches)
  - $readmemb("<file_name>",<memory>);
  - $readmemb("<file_name>",<memory>,<start_addr>,<finish_addr>);
  - $readmemh("<file_name>",<memory>);
  - $readmemh("<file_name>",<memory>,<start_addr>,<finish_addr>);
- **$readmemh** ➔ Hex data…**$readmemb** ➔ binary data
  - But they are reading ASCII files either way (just how numbers are represented)

```
// addr     data
@0000 10100010
@0001 10111001
@0002 00100011
```

example "binary" file

```
// addr     data
@0000   A2
@0001   B9
@0002   23
```

example "hex" file

```
//data
A2
B9
23
```

address is optional for the lazy

Example of **$readmemh**

Inserted from lecture slides for ease of reference

```
module rom(input clk; input [7:0] addr; output [15:0] dout);

reg [15:0] mem[0:255];      // 16-bit wide 256 entry ROM
reg [15:0] dout;

initial
   $readmemh("constants",mem);

always @(negedge clk) begin
   ////////////////////////////////////////////////////////
   // ROM presents data on clock low //
   ////////////////////////////////////////////////////////
   dout <= mem[addr];
end

endmodule
```

HW5 Problem 2 (This problem is **individual** basis)

Individual problem and not covered by exercise.

2. (17pts) Post synthesis simulation

Posted on the class webpage (under the tutorials section) is a file about Post Synthesis Validation (simulation). Read it carefully.

As part of HW4 (and Ex16) you created a synthesis script to synthesize your **UART**. Adapt that synthesis script to apply to your **PID.sv** block. Synthesize and produce **PID.vg**.

Now incorporate that gate level netlist into your test bench (**PID_tb.sv**) and prove your post synthesis netlist works.

Submit to the dropbox:

a) PID.vg

b) Simulation results proving success.

i. Image of your transcript loading the library cells
ii. Image of your transcript window showing success

You are submitting images to prove:
1.) This is indeed a post synth simulation
2.) You indeed ran it *(**your name**/username somewhere on image)*
3.) Evidence it passed *(or not…some credit for trying)*

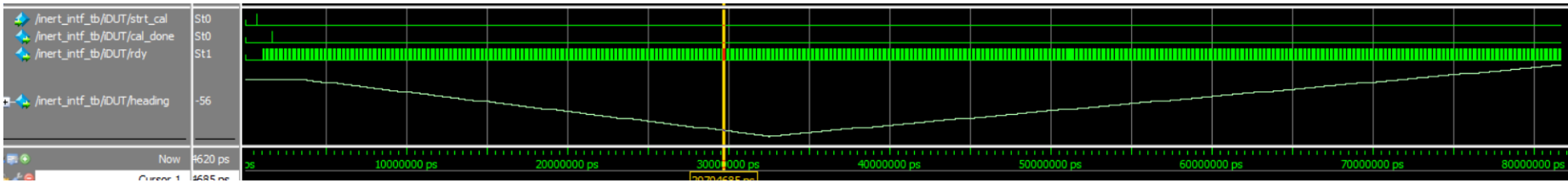
(**Note:** you may want to try post synthesis simulation of a few of your other blocks to ensure they are ready for the project)

HW5 Problem 3 (This problem is done on **Team** basis)

3. (20pts) inertial_interface

Started as Ex19 on Nov 8th

From Exercise19 your team should ideally have two implementations of inert_intf.sv. Pick one of the implementations for submission for this HW and for use on the project.



One person *(per team)* Submit: **inert_intf.sv**, **inert_intf_tb.sv**, and proof it ran. The other member submit a simple stating your team name.

HW5 Problem 4 (This problem is done on **Team** basis)

4. (25pts) cmd_proc

Started as Ex21 on Nov 15th

See Exercise 21 for specification

Same 1 person as submitted inert_intf, submit:

- cmd_proc.sv
- cmd_proc_tb.sv
- Proof it ran and passed your testbench

HW5 Problem 5 (This problem is done on **Team** basis)

5. (22pts) charge fanfare

Started as Ex21 on Nov 15th

See Exercise 21 for specification

Same 1 person as submitted inert_intf, submit:

- charge.sv
- charge_tb.sv
- (check off on DE-0 nano is also required)