ECE 551

HW4 *(100 pts)*

- Due Fri Apr 1st at class
- Work Individually
- Use descriptive signal names
- Comment & indent your code
- Code will be judged on coding style

HW4 Problems 1&2 **(10pts) + (10pts)**

1. **(10pts)** Complete the Synopsys Design Vision tutorial. Sign below, (preferably in blood).

I, ____ completed the Synopsys Design Vision tutorial. If I had any problems with it, I discussed them with the TA or Instructor, either in person, or through email.

2. **(10pts)** Project Team Formation

Form a 3 or 4 person project team, **Come up with a team name**, and fill in the table below:

Team Name:	
Person1:	
Person2:	
Person3:	
Person4:	

HW4 Problem 3 (**20pts)** Synthesize your telemetry.sv

This problem started as Ex14 on Mon Mar 21st

- In HW3 you produced the telemetry module (**telemetry.sv**) which instantiated **UART_tx.sv** (**NOTE:** if your **telemetry.sv** code is a big ball of stink, and one of your project partners has a better **telemetry** you can use their code. You **still** have to do the synthesis **individually**)

- In this HW you will synthesize **telemetry** using Synopsys on the CAE linux machines.

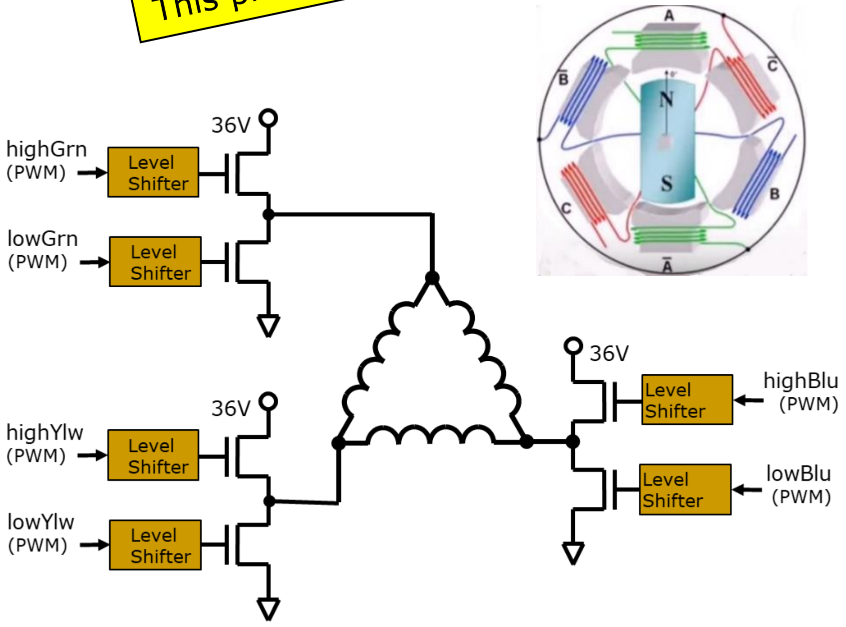
- Write a synthesis script to synthesize your telemetry. The script should perform the following:
  - Defines a clock of 500MHz frequency and sources it to clock
  - Performs a set don't touch on the clock network
  - Defines input delays of 0.5 ns on all inputs other than clock
  - Defines a drive strength equivalent to a 2-input nand of size 1 from the Synopsys 32nm library (NAND2X1_LVT) for all inputs except clock and rst_n
  - Defines an output delay of 0.75ns on all outputs.
  - Defines a 0.15pf load on all outputs.
  - Sets a max transition time of 0.15ns on all nodes.
  - Employs the Synopsys 32nm wire load model for a block of sise 16000 sq microns
  - Compiles, then flattens the design so it has no hierarchy, and compiles again.
  - Produces a min_delay report
  - Produces a max_delay report
  - Produces an area report
  - Flattens the design so it has no hierarchy
  - Writes out the gate level verilog netlist (**telemetry.vg**)

- Submit to the dropbox.
  - Your synthesis script (**telemetry.dc**)
  - The output reports for area (**telemetry_area.tx**t)
  - The gate level verilog netlist (**telemetry.vg**)

HW4 Problem 4 (**15pts**) brushless.sv

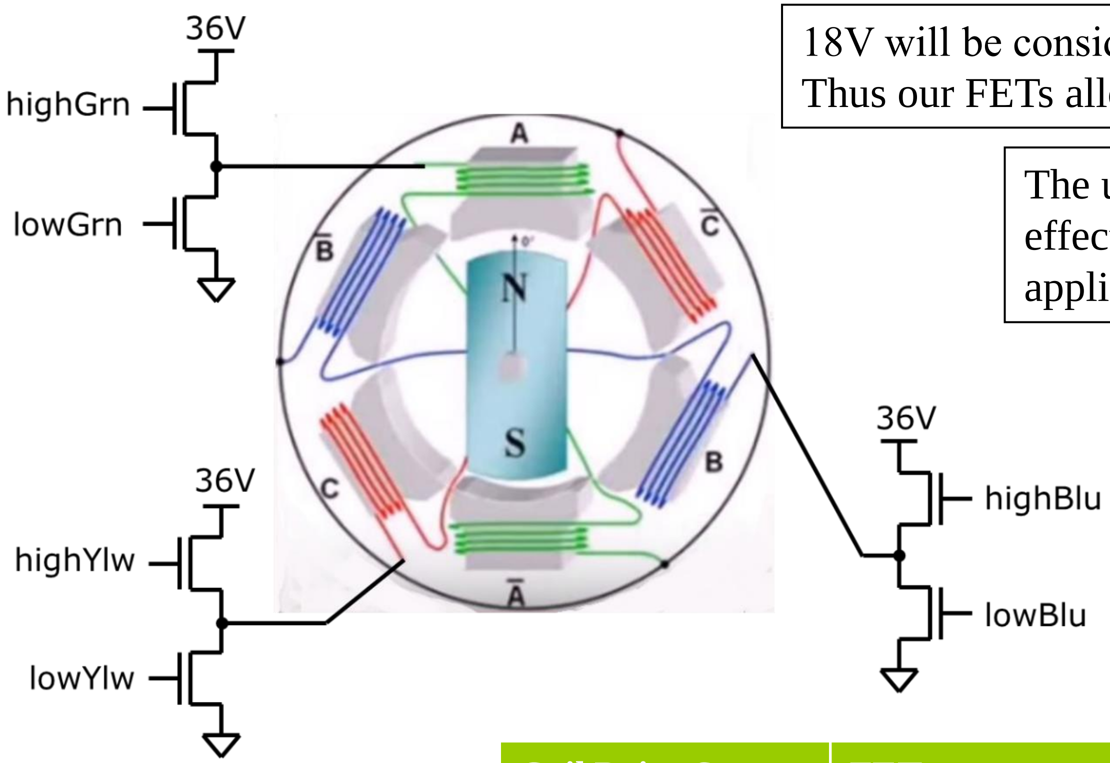
This problem started as Ex12 on Mon Mar 7th

- You researched brushless motor drives in HW1…you might not remember much.
- We have 3 coil connections that we have to drive. We call them green, yellow, and blue. For any given coil we drive current in one direction, the opposite direction, or not at all.
- How do we know the proper coil drive at any given time? That is determined by the position of the rotor. We know the rotor position from the hall sensor inputs. Switching coil drive is referred to as commutation.



- **brushless.sv** will be the block that inspects the hall sensor signals and determines how to drive each coil. Each coil can be driven 1 of 4 ways: not driven (both high/low FETs off); driven "forward"; driven "reverse"; driven for dynamic braking (high FET off, low FET PWMed)
- Are the hall effect sensors synchronous to our clock domain? What does that mean?
- If you are curious why PWMing the lower FETs creates dynamic braking I think I can stumble through an explanation, but I suspect most of you lemmings will be content to just implement it as specified.

HW4 Problem 4 (**15pts**) brushless.sv



18V will be considered "virtual ground" for the coils. Thus our FETs allow us to drive positive or negative.

The use of PWM allows us to control the effective magnitude of the voltage applied across the coil. (hence speed/torque)

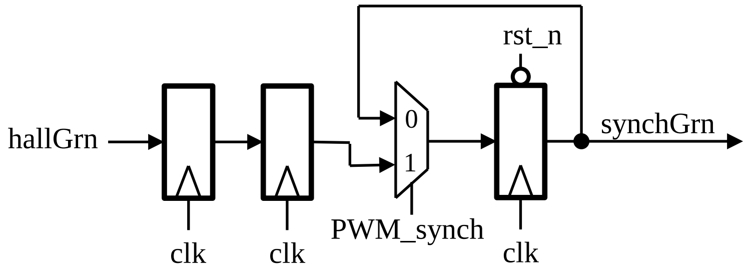
A PWM setting of 0x400 would be 50% and would represent driving a coil with 18V (virtual ground).

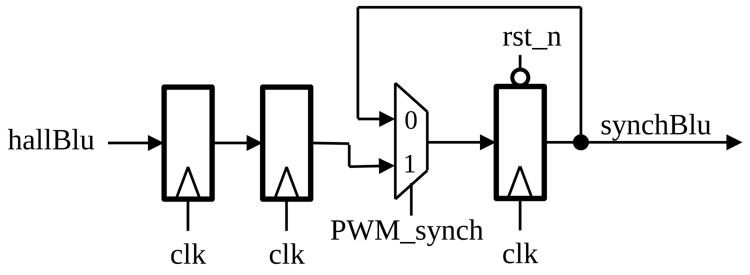
There are 4 possible states a coil can be driven in. Regenerative braking is a special case indicated by **brake_n** signal being low.

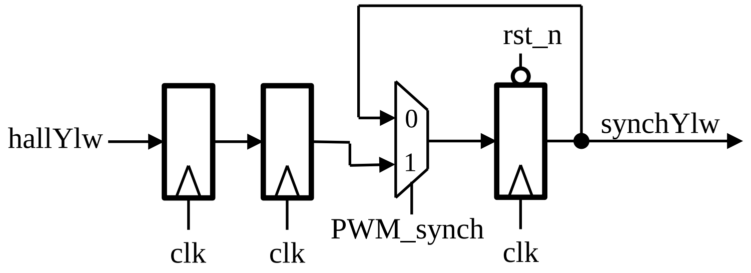
Coil Drive State:	FET gate controls:
Not driven (High Z)	Both high and low side low (both off)
Forward Current	High side driven with PWM, low side driven with ~PWM
Reverse Current	High side driven with ~PWM, low side driven with PWM
Regen Braking	High side low (off), low side driven with PWM

HW4 Problem 4 (**20pts)** brushless.sv

- There are two forms of synchronizing we must concern ourselves with:
  - Synchronizing Hall sensors to our clock domain….ie. Double flopping
  - Synchronizing commutation to our PWM cycle
    - Coil drive and commutation are determined by hall sensor readings
    - Hall sensor readings are not correlated to our PWM cycle *(change in hall sensor reading could happen anywhere in duty cycle of our PWM).*
    - We ideally would like to switch commutation when all power FETs are off *(i.e. when the PWM is in its deadtime)*
    - So we want to create a set of hall sensor signals that are correlated to our PWM. We will use the **PWM_synch** signal to synchronize hall readings to the PWM cycle *(and thus ensure commutation occurs during the deadband nonoverlap block enforces)*







The hall effect sensor wires are also green, yellow, & blue.

Infer the shown synchronizer circuits to form double synchronized signals for all 3 hall effect signals.

HW4 Problem 4 (**15pts)** brushless.sv

Determining next drive conditions from hall effect sensor readings:

- The hall effect sensors tell us the current position of the rotor
- The hall effect sensor wires are also green, yellow, and blue.
- Form a 3-bit vector:

  assign rotation_state = {synchGrn,synchYlw,synchBlu};

- The following table outlines how we drive the coils relative to **rotation_state**

rotation_state	**3'b101**	**3'b100**	**3'b110**	**3'b010**	**3'b011**	**3'b001**
coilGrn	for_curr	for_curr	High Z	rev_curr	rev_curr	High Z
coilYlw	rev_curr	High Z	for_curr	for_curr	High Z	rev_cur
coilBlu	High Z	rev_curr	rev_curr	High Z	for_curr	for_curr

- In the case of **brake_n == 1'b0** (braking) all coils are driven in the regenerative braking state with the high side FET off and the low side FET PWMing.

HW4 Problem 4 (**15pts)** brushless.sv

Signal:	Dir:	Description:
clk,rst_n	in	50MHz clock & active low reset
drv_mag[11:0]	in	From PID control. How much motor assists (unsigned)
hallGrn,hallYlw, hallBlu	in	Raw hall effect sensors (asynch)
brake_n	in	If low activate regenerative braking at 75% duty cycle
duty[10:0]	out	Duty cycle to be used for PWM inside ***mtr_drv***. Should be 0x400+drv_mag[11:2] in normal operation and 0x600 if braking.
selGrn[1:0], selYlw[1:0], selBlu[1:0]	out	2-bit vectors directing how ***mtr_drv*** should drive the FETs. 00=>HIGH_Z, 01=>rev_curr, 10=>frwrd_curr, 11=>regen braking

Code and test ***brushless.sv*** with the interface specified in this table.

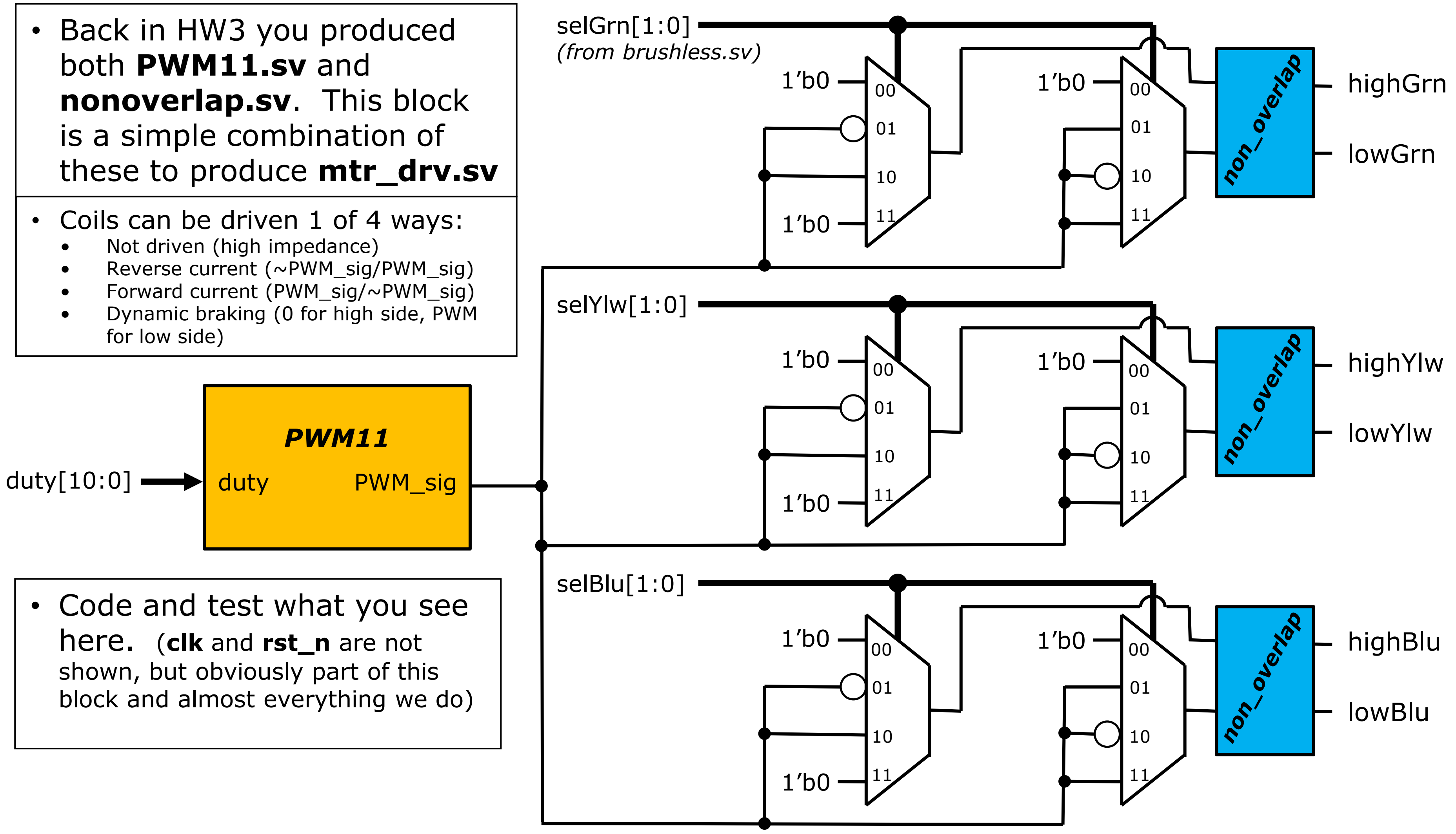
Submit: **brushless.sv**

This problem started as Ex13 on Weds Mar 9th

HW4 Problem 5 (**20pts**) mtr_drv.sv & Testing

- Back in HW3 you produced both **PWM11.sv** and **nonoverlap.sv**. This block is a simple combination of these to produce **mtr_drv.sv**

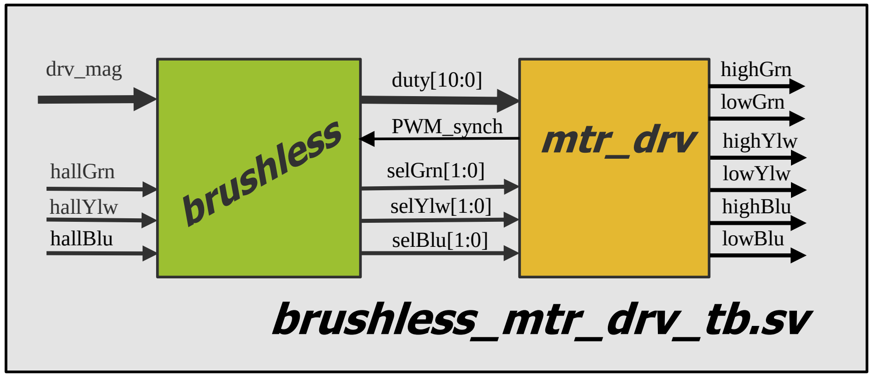
- Coils can be driven 1 of 4 ways:
  - Not driven (high impedance)
  - Reverse current (~PWM_sig/PWM_sig)
  - Forward current (PWM_sig/~PWM_sig)
  - Dynamic braking (0 for high side, PWM for low side)



- Code and test what you see here. (**clk** and **rst_n** are not shown, but obviously part of this block and almost everything we do)

HW4 Problem 5 (**20pts**) mtr_drv.sv & Testing

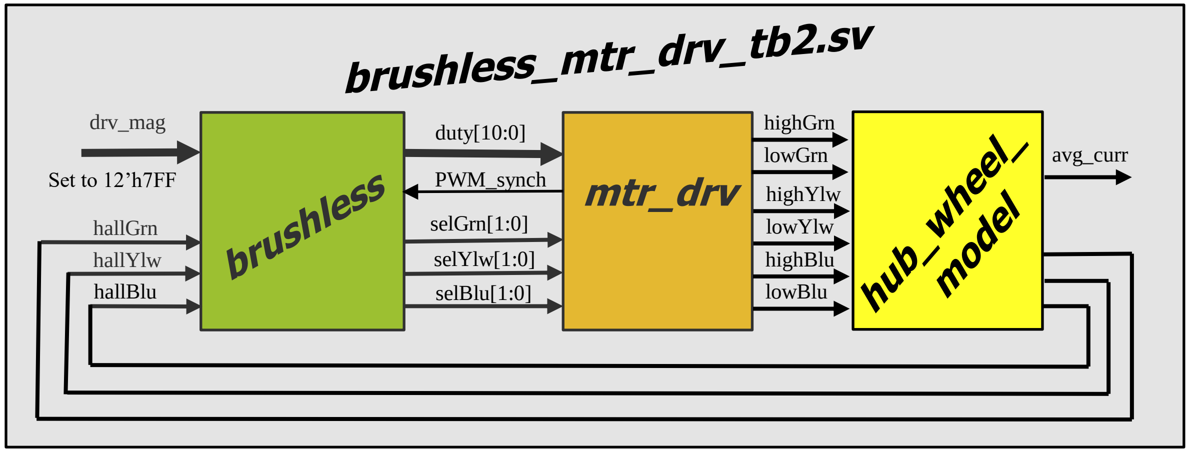
- The outputs of **brushless.sv** feed the inputs to **mtr_drv.sv,** they are best tested in combination.
- A combined testbench (**brushless_mtr_drv_tb.sv)** is provided. Flesh out the stimulus, run it, and manually check the behavior against the table below.



{hallGrn,hallYlw,hallBlu}	**Expected Output for Grn, Ylw, Blu given as high/low**
101	Grn = PWM/~PWM, Ylw = ~PWM/PWM, Blu = 0/0
100	Grn = PWM/~PWM, Ylw = 0/0, Blu = ~PWM/PWM
110	Grn = 0/0, Ylw = PWM/~PWM, Blu = ~PWM/PWM
010	Grn = ~PWM/PWM, Ylw = PWM/~PWM, Blu = 0/0
011	Grn = ~PWM/PWM, Ylw = 0/0, Blu = PWM/~PWM
001	Grn = 0/0, Ylw = ~PWM/PWM, Blu = PWM/~PWM
000 or 111	Both high and low at 0 for all channels
brake_n == 1'b0	All high channels at 0. All low channels at PWM (75%)

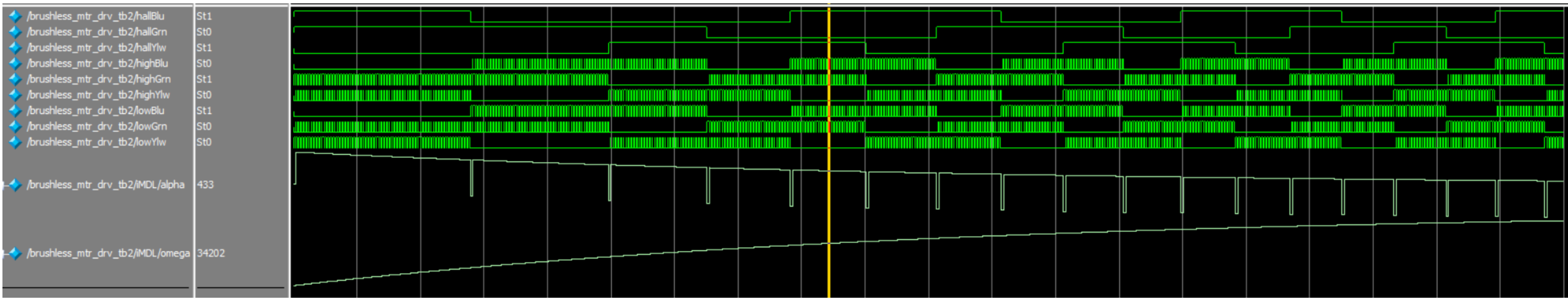
HW4 Problem 5 **(20pts)** mtr_drv.sv & Testing

- A model of coil drive and a brushless DC motor (**hub_wheel_model.sv**) is introduced to help further check your **brushless/mtr_drv** combination.



- This model evaluates the drive on **highGrn**, **lowGrn**, … and models the physics of the torque.
- It contains a signals **alpha** and **omega** which represent the angular acceleration and angular velocity of the motor.
- **brushless_mtr_drv_tb2.sv** is available for download, as is **hub_wheel_model.sv**.
- **hub_wheel_models.sv** contains a child (**coil_volt.sv**) that you also have to download and include in the project build.

HW4 Problem 5 (**20pts**) mtr_drv.sv & Testing



```
/brushless_mtr_drv_tb2/hallBlu          St1
/brushless_mtr_drv_tb2/hallGrn          St0
/brushless_mtr_drv_tb2/hallYlw          St1
/brushless_mtr_drv_tb2/highBlu          St0
/brushless_mtr_drv_tb2/highGrn          St1
/brushless_mtr_drv_tb2/highYlw          St0
/brushless_mtr_drv_tb2/lowBlu           St1
/brushless_mtr_drv_tb2/lowGrn           St0
/brushless_mtr_drv_tb2/lowYlw           St0
/brushless_mtr_drv_tb2/iMDL/alpha       433
/brushless_mtr_drv_tb2/iMDL/omega       34202
```

- Above are the expected waveforms of this simulation with **alpha** and **omega** plotted as analog.

- The follow slides cover how to plot waveforms as analog in ModelSim.

- If you can reproduce similar results you can have pretty good confidence your **brushless** and **mtr_drv** implementations are functionally correct.

- **Submit:**
  - **mtr_drv.sv**
  - Your completed **brushless_mtr_drv.sv** and a capture of the waveforms
  - Waveforms from your simulation of **brushless_mtr_drv2.sv**.

Done as teams of 2

This problem started as Ex16 on Mon Mar 28th

HW4 Problem 6 (**25pts**) Testing **brushless** & **mtr_drv** on platform

Whole controller board mounted on a pivot to mimic going up/down hill

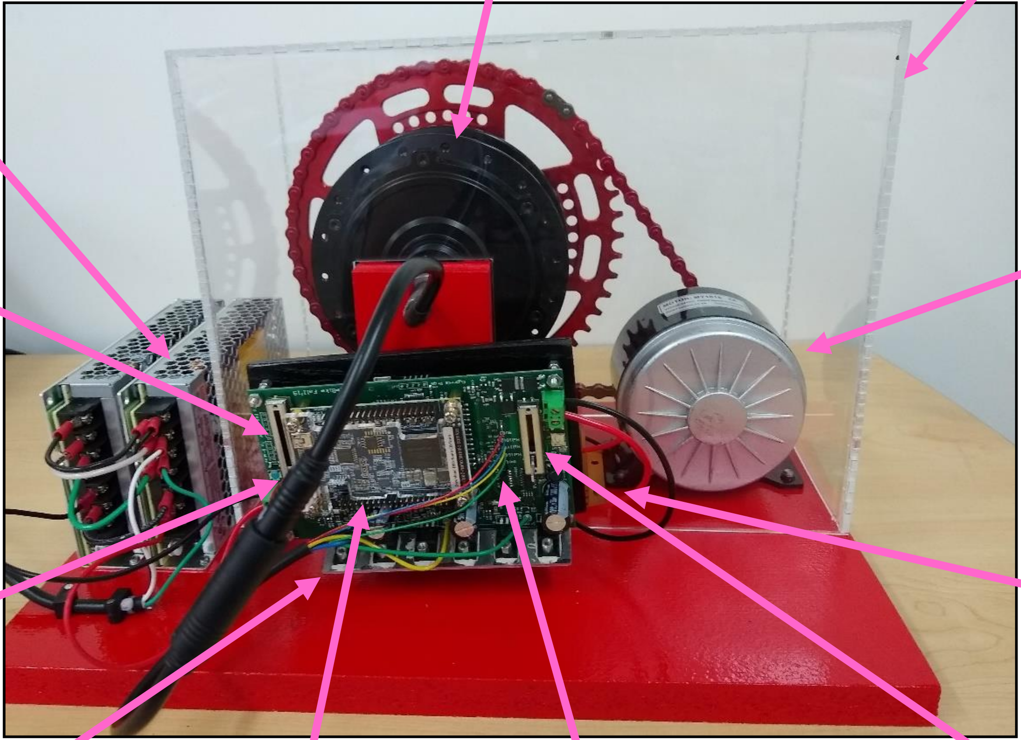
E-Bike hub motor (250W brushless DC)

Wouldn't want you getting your hair caught in the chain.

2 series 18V supplies to provide 36V DC

This slide potentiometer on the board mimics the pedaling torque sensor

Push button that mimics rider squeezing the brake handle



E-Bike motor is coupled (via chain) to generator. The generator serves as a mechanical load on the motor.

Load resistor to dissipate the power the generator generates.

You can see the 6 Power FETs that drive the motor coils are mounted to an aluminum heat sink

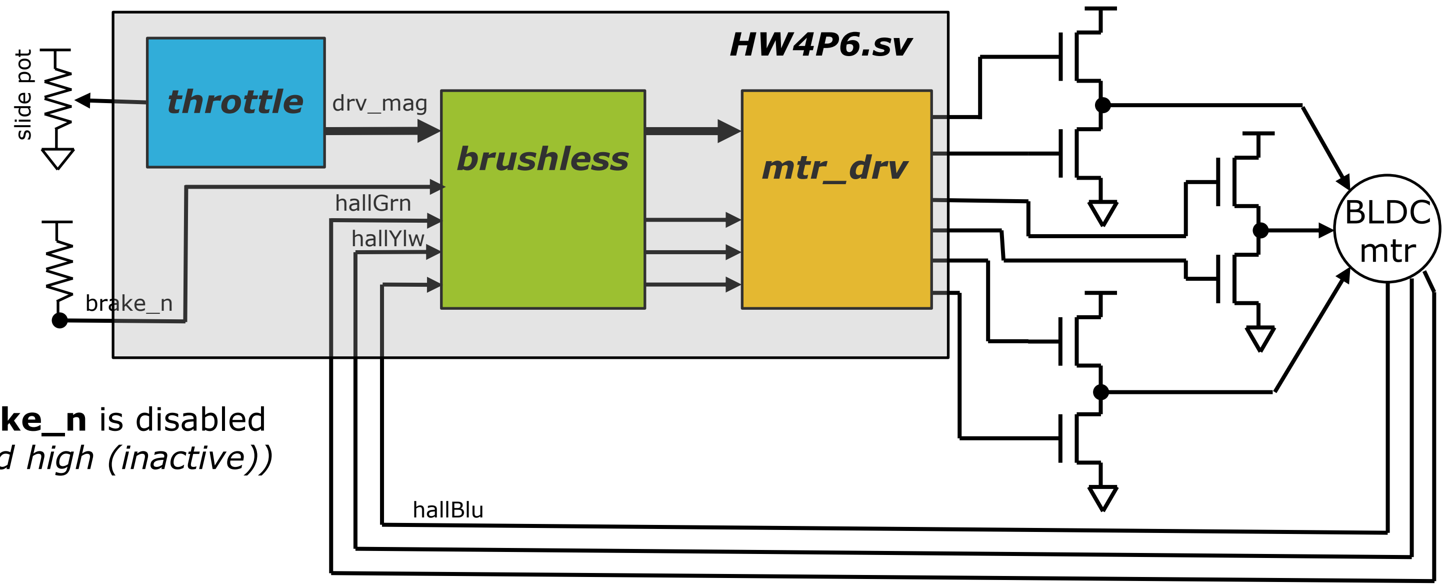
The DE-0 Nano board contains the FPGA that is the "brains" of the operation.

Here you can see the Hall effect sensor wires coming in from the motor.

This slide potentiometer mimics the cadence sensor

Done as teams of 2

HW4 Problem 6 (**25pts)** Testing **brushless** & **mtr_drv** on platform



brake_n is disabled
(tied high (inactive))

- For this problem you will map your **brushless** and **mtr_drv** designs to a FPGA that exists on the test platform, and test that they function properly by driving a ebike hub motor (BLDC mtr).
- The code to interface to the slide potentiometer (that serves as a throttle) is provided (**throttle.sv**). The shell to wrap things together **HW4P6.sv** is also provided.
- Download the provided .sv files along with **HW4P6.qpf** and **HW4P6.qsf**. Launch Quartus using the provided .qpf. Compile and map to the DE-0 on the test platform.
- Submit a video showing you download the code to the DE-0 and running the motor **at various speeds**. Include a shot of **your faces** in the video (helps me associate names with faces).