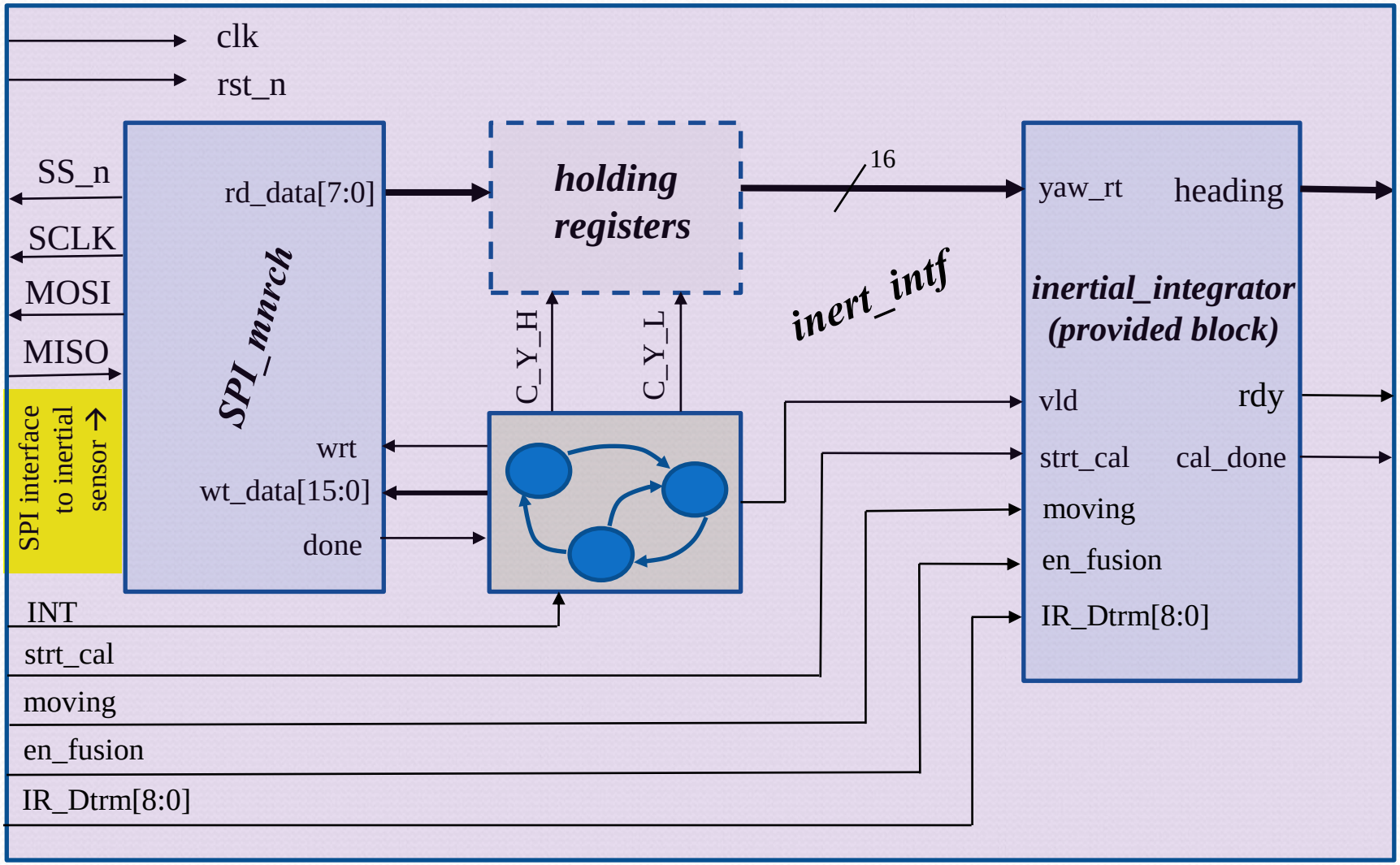
Exercise 19 Inertial Interface:

- Break into two teams of 2 for this exercise.
- Each team of 2 should perform the exercise *(that way your team will have two possible choices for your **inert_intf.sv** design.)*

Exercise 19 Inertial Interface: (NOTE: a shell is provided)

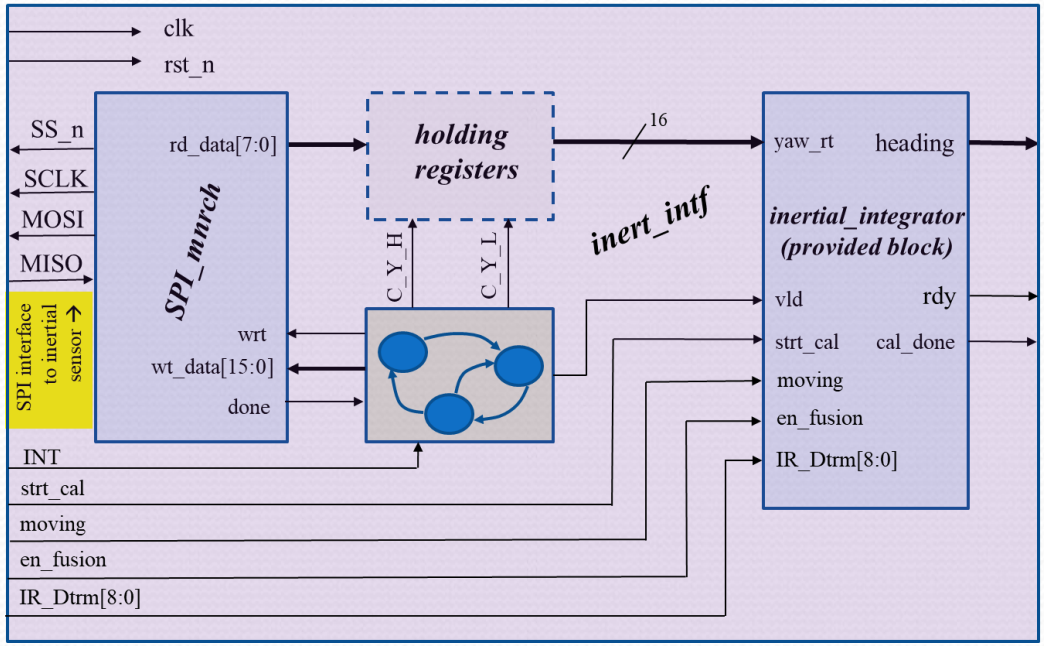


Exercise 19 Inertial Interface: (overview)

inert_intf will configure the inertial sensor and obtain readings from it. It will perform and initial calibration (**strt_cal**) on the yaw readings when it is known that the mazeRunner is not moving. After the calibration is over (**cal_done**) it will be in a mode where when **moving** it will be reading and integrating the yaw readings to provide heading. When known to be moving forward (en_fusion) the IR_Dtrm[8:0] will be used in a form of sensor fusion to correct drift in the gyro readings.

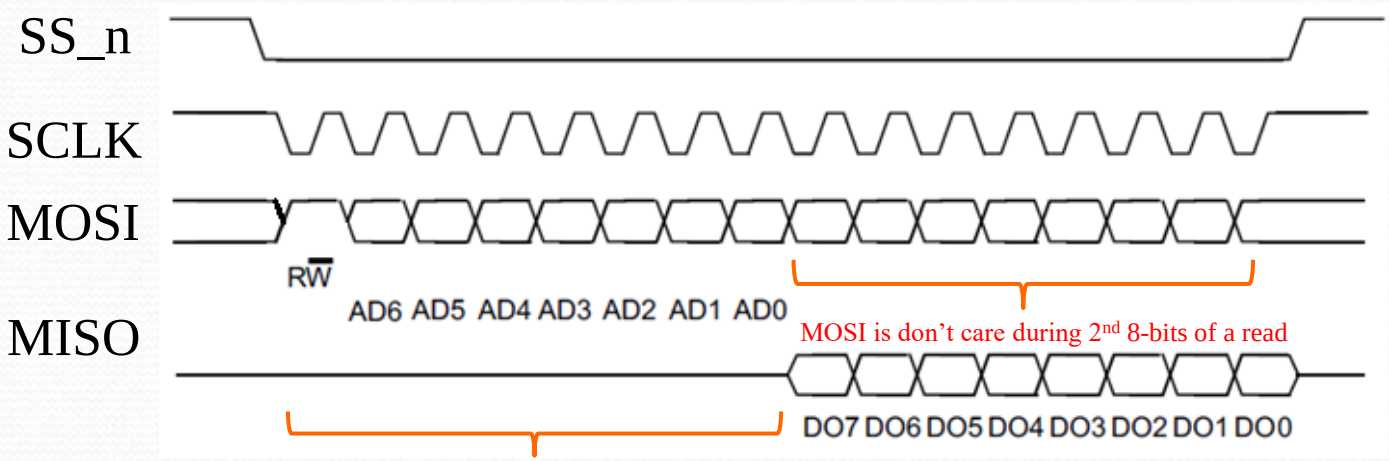
The calibration of the gyro readings, and the sensor fusion is done by (***inertial_integrator***) which is a provided block. It is not that ***inertial_integrator*** is anything too complex for you to implement. It was just too difficult for me to specify succinctly *(was easier to provide it)*

There is a parameter (**FAST_SIM**) passed to ***inertial_integrator*** that when set true reduces the number of samples taken during calibration from 2048 to 8. This is used to speed up simulation times. We will use this **FAST_SIM** concept a few more times in the project.



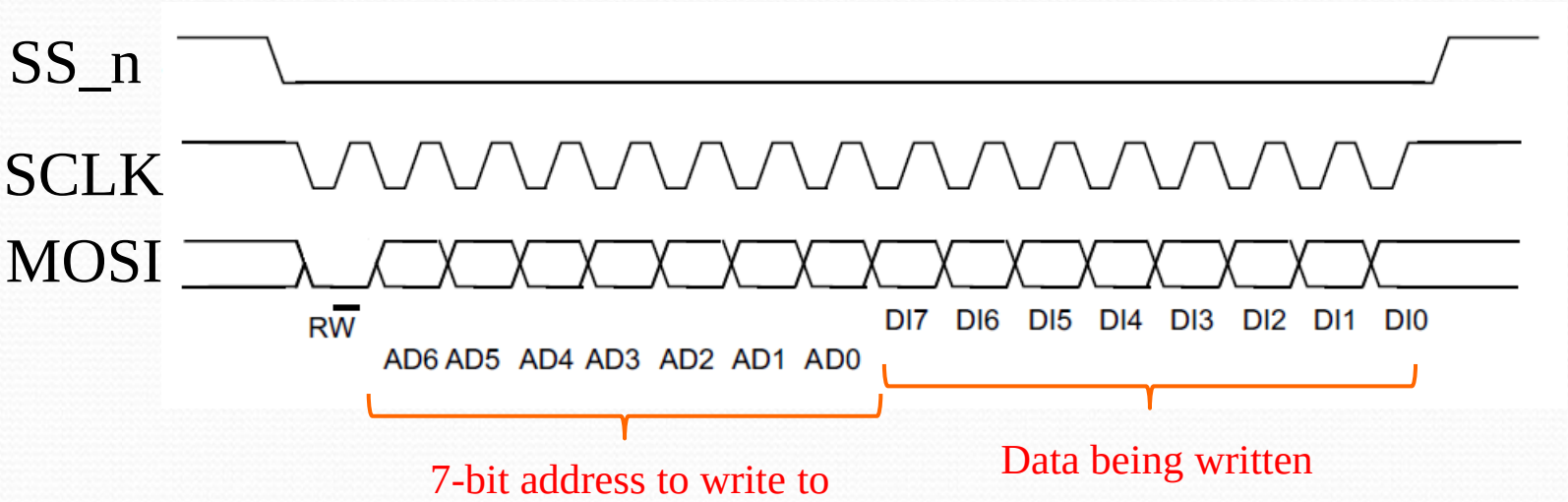
Exercise 19 Inertial Interface: (read of inertial sensor)

- One of the primary functions of the inertial interface is to drive the SPI interface to configure and perform reads of the inertial sensor.
  - The inertial sensor performs an 8-bit read or write in a single 16-bit SPI transaction.
  - For the first 8-bits of the SPI transaction, the sensor is looking at MOSI to see what register is being read/written. The MSB is a R/$\overline{W}$ bit, and the next 7-bits comprise the address of the register being read or written.
  - If it is a read the data at the requested register will be returned on MISO during the 2nd 8-bits of the SPI transaction (see waveforms below for read)



Exercise 19 Inertial Interface: (write to inertial sensor)

- During a write to the inertial sensor the first 8-bits specify it is a write and the address of the register being written. The 2nd 8-bits specify the data being written. (see diagram below)



- Of course the sensor is returning data on MISO during this transaction, but this data is garbage and can be ignored.

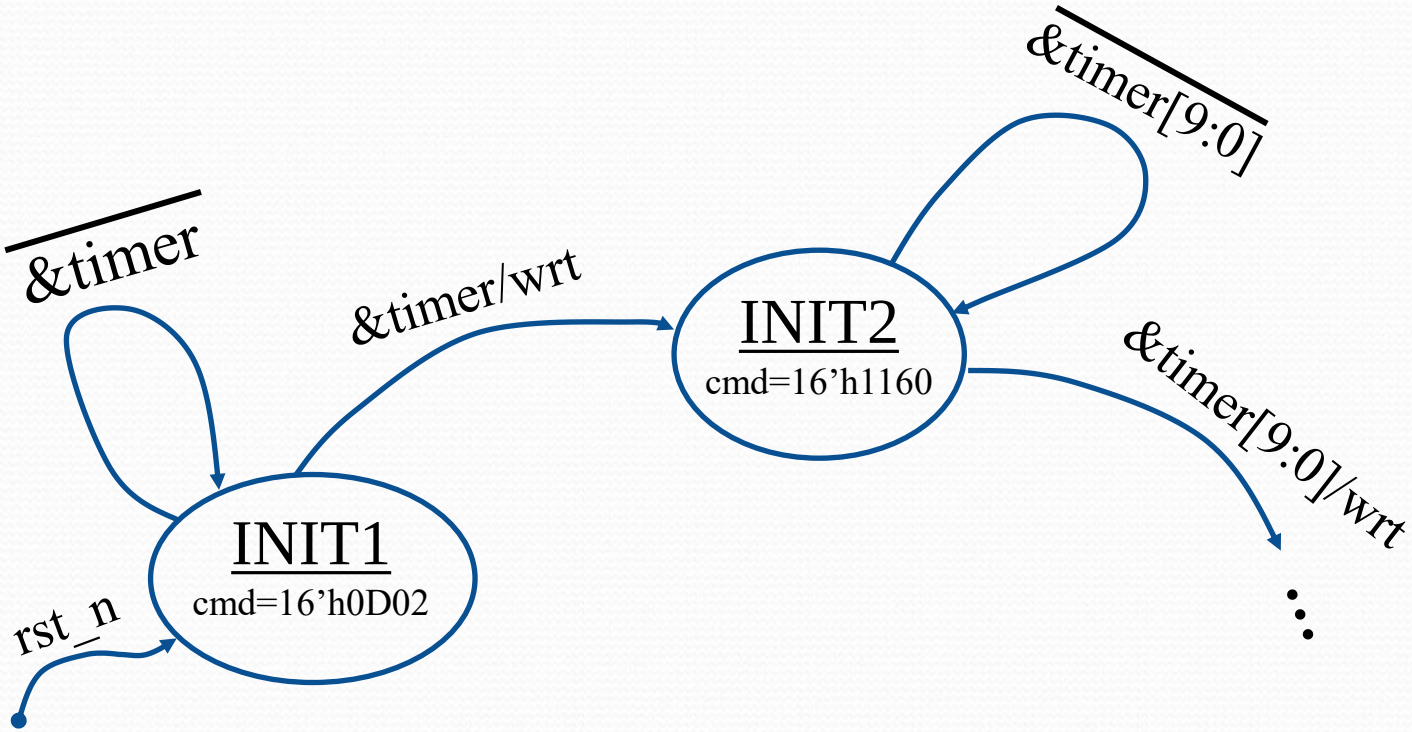
Exercise 19 Inertial Interface: (initializing inertial sensor)

- After reset de-asserts the system must write to some registers to configure the inertial sensor to operate in the mode we wish. The table below specifies the writes to perform.

Addr/Data to write:	Description:
0x0D02	Enable interrupt upon data ready
0x1160	Setup gyro for 416Hz data rate, +/- 250°/sec range.
0x1440	Turn rounding on for gyro readings

- You will need a state-machine to control communications with the inertial sensor. Obviously we are also reading the inertial sensor constantly during normal operation. The initialization table above just specifies what some of the first states of the SM need to do.

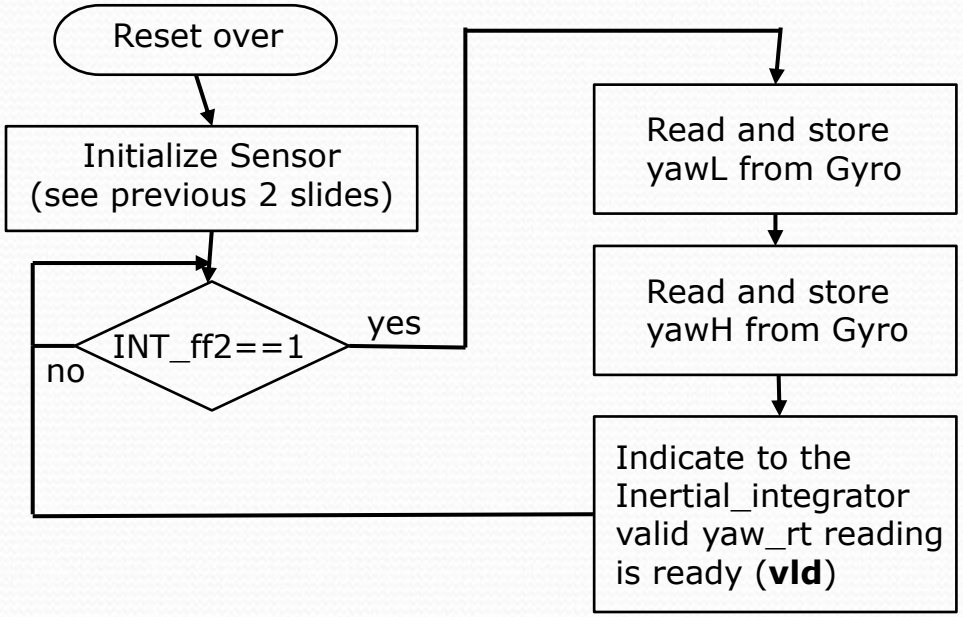
Exercise 19 Inertial Interface: (initializing inertial sensor)



- The inertial sensor has a reset sequence that takes a while. If we start initializing it immediately after reset it will not be ready. I used a 16-bit timer, and only wrote the first initialization commands once the 16-bit timer was full. I wrote the subsequent based on **&timer[9:0]** but probably should have used **done** signal from ***SPI_mnrch***.

Exercise 19 Inertial Interface: (flow)

- After initialization of the inertial sensor is complete the inertial interface state-machine should go into an infinite loop of reading gyro yaw data.
- The sensor provides an active high interrupt (**INT**) that tells when new data is ready. Double flop that signal *(for meta-stability reasons)* and use the double flopped version to initiate a sequence of two from the gyro.



- You will have two 8-bit flops to store the 2 needed reading from the inertial sensor (yawl, yawH).
- See next slide for addresses of yawl & yawH.

Exercise 19 Inertial Interface: (read addresses)

- The table below specifies the addresses you need to use to read inertial data. Recall for a read the lower byte of the 16-bit packet sent (**wt_data[7:0]**) for a read is a don't care. You will ignore the upper byte of the data received (**rd_data[15:8]**)

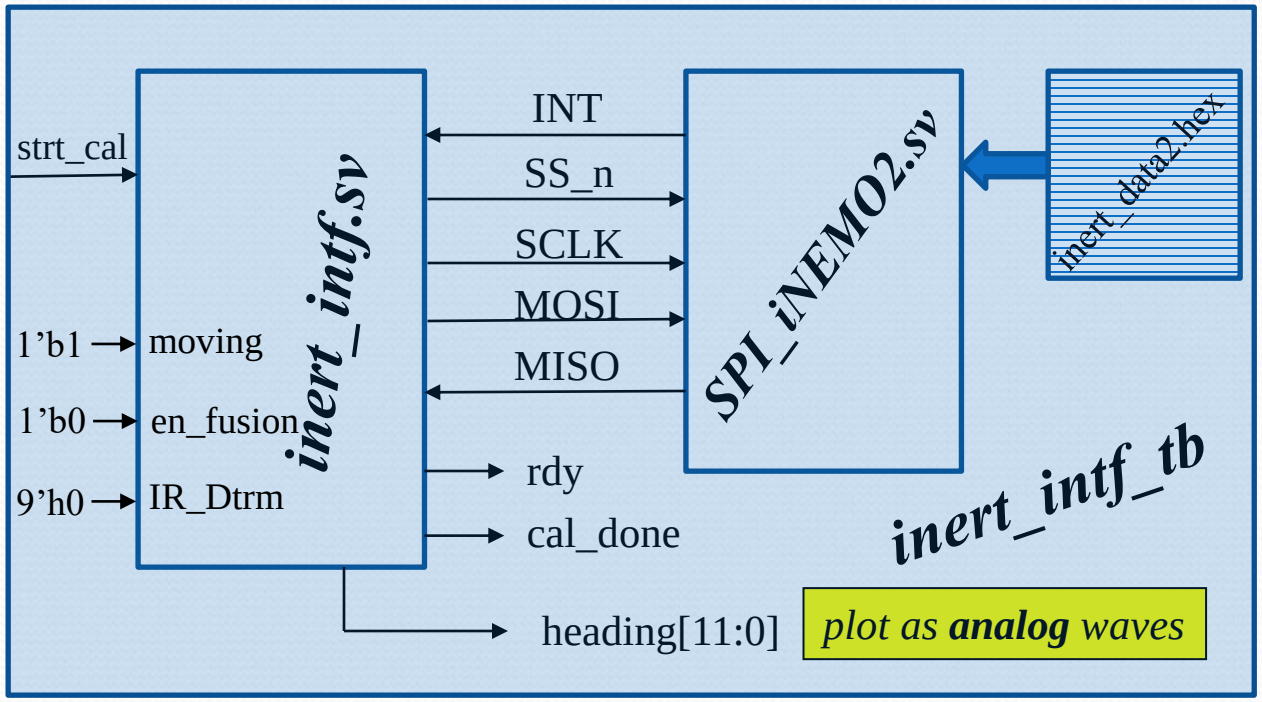
Addr/Data:	Description:
0xA6xx	yawL ➔ yaw rate low from gyro
0xA7xx	yawH ➔ yaw rate high from gyro

Exercise 19 Inertial Interface: (signal interface)

Signal:	Dir:	Description:
clk, rst_n	in	50MHz clock and active low asynch reset
strt_cal	in	From ***cmd_proc*** unit. Tells ***inertial_integrator*** to start cal.
INT	in	INT pin from inertial sensor, tells us a new measurement is ready. Is this synched to your **clk** domain?
cal_done	out	From ***inertial_integrator*** to ***cmd_proc***. Asserted when calibration complete
rdy	out	Asserted from SM inside ***inert_intf***. Consumed in ***inertial_integrator***, but also an output to ***flght_cntrl***.
heading[11:0]	out	This is the primary output of ***inert_intf***. This is a signed number where 000 represents original direction, 3FF is 90° CCW, 7FF = 180° CCW, BFF = 90° CW...
moving	in	Integration of gyro yaw readings is inhibited when not **moving**
en_fusion	in	Enables fusion correction.
SS_n, SCLK, MOSI, MISO	out/ in	SPI interface to inertial sensor

A shell (**inert_intf_shell.sv**) is provided, rename to **inert_intf.sv** and flesh it out.

Exercise 19 Inertial Interface: (testing it)



NOTE this is using **SPI_iNEMO2.sv** (note the **2**). This is a slightly different version of ***SPI_iNEMO*** model than you used in Exercise17. This one reads a different .hex file for its data and uses a longer data set.

Build ***inert_intf_tb*** as shown above**. SPI_iNEMO2.sv** is provided as is the file it reads (**inert_data2.hex**). The next slide outlines some of what your testbench should do.

NOTE: The connections of **moving**, **en_fusion**, & **IR_Dtrm[8:0]**, are not tested. Ensure they are properly connected from ***inert_intf*** toplevel to ***inertial_integrator***.

Exercise 19 Inertial Interface: (steps to testing it)

1. Reset the DUT but don't assert **strt_cal** early. Recall the SM inside ***inert_intf*** is first waiting for a 16-bit timer to expire before it does anything.

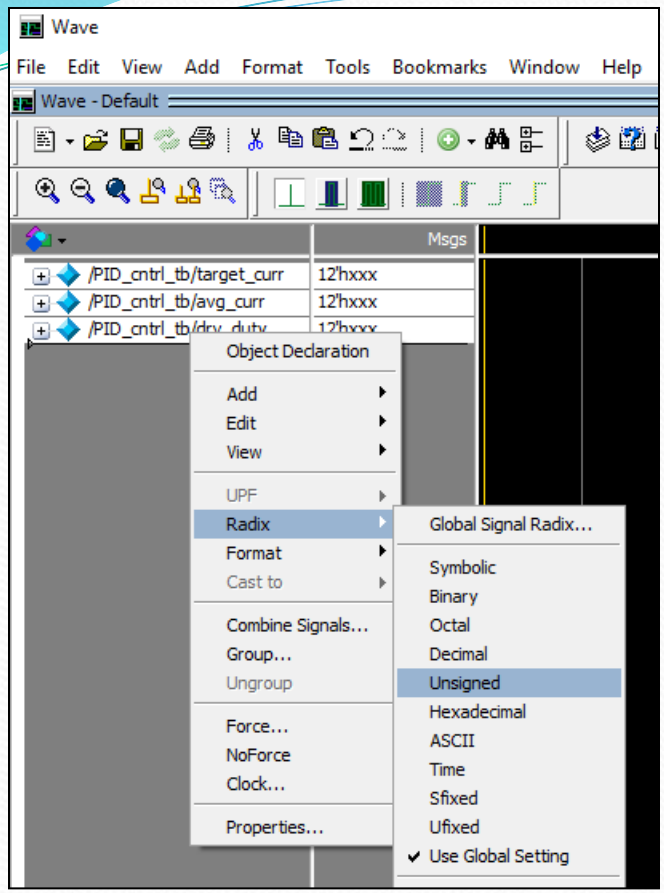
2. Wait for **NEMO_setup** inside ***SPI_iNEMO2*** to get asserted. This would be a good place to use the **fork/join** technique for waiting for a signal to assert, but having a timeout if it doesn't.

3. Assert **strt_cal** for one clock cycle.

4. Wait for **cal_done** to get asserted (timeout loop might have to be near 1 million clock cycles).

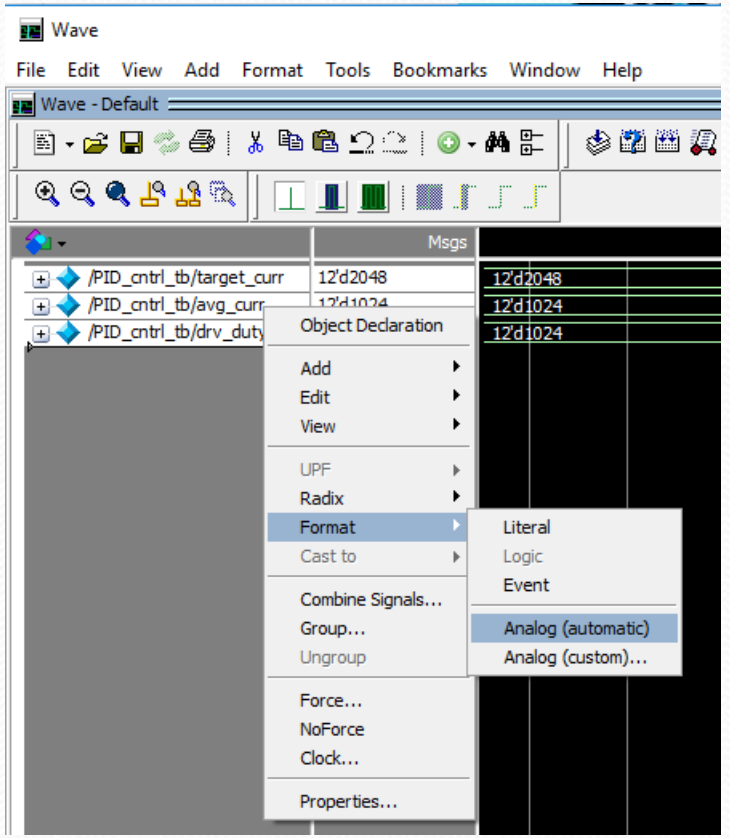
5. After this just let it run for 8 million more clocks (will take a while), and we will plot the results of **heading**.

Viewing waveforms as analog



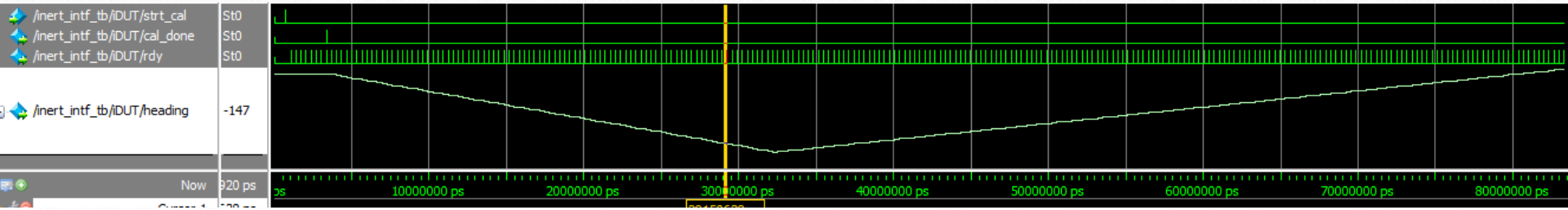
First select the signals of interest and change the radix (right click). If the signals are signed you would choose *Decimal*. Choose *Decimal* for **error**, and choose *Unsigned* for **drv_duty** and **line_deviation**.

If you have not already…run the simulation



Finally right click on the signals again and change the display format to “Analog (automatic)” It can’t automatically scale signals for which it does not have data, so you have to have run the simulation to do this.

Exercise 19 Inertial Interface: (steps to testing it)



A plot of **yaw** is shown above as an analog wave. Yours should look similar if **inert_intf** is working properly.

One person *(per 2 person team)* Submit: **inert_intf.sv**, **inert_intf_tb.sv**, and proof it ran. The other member submit a simple text file stating who you worked with.