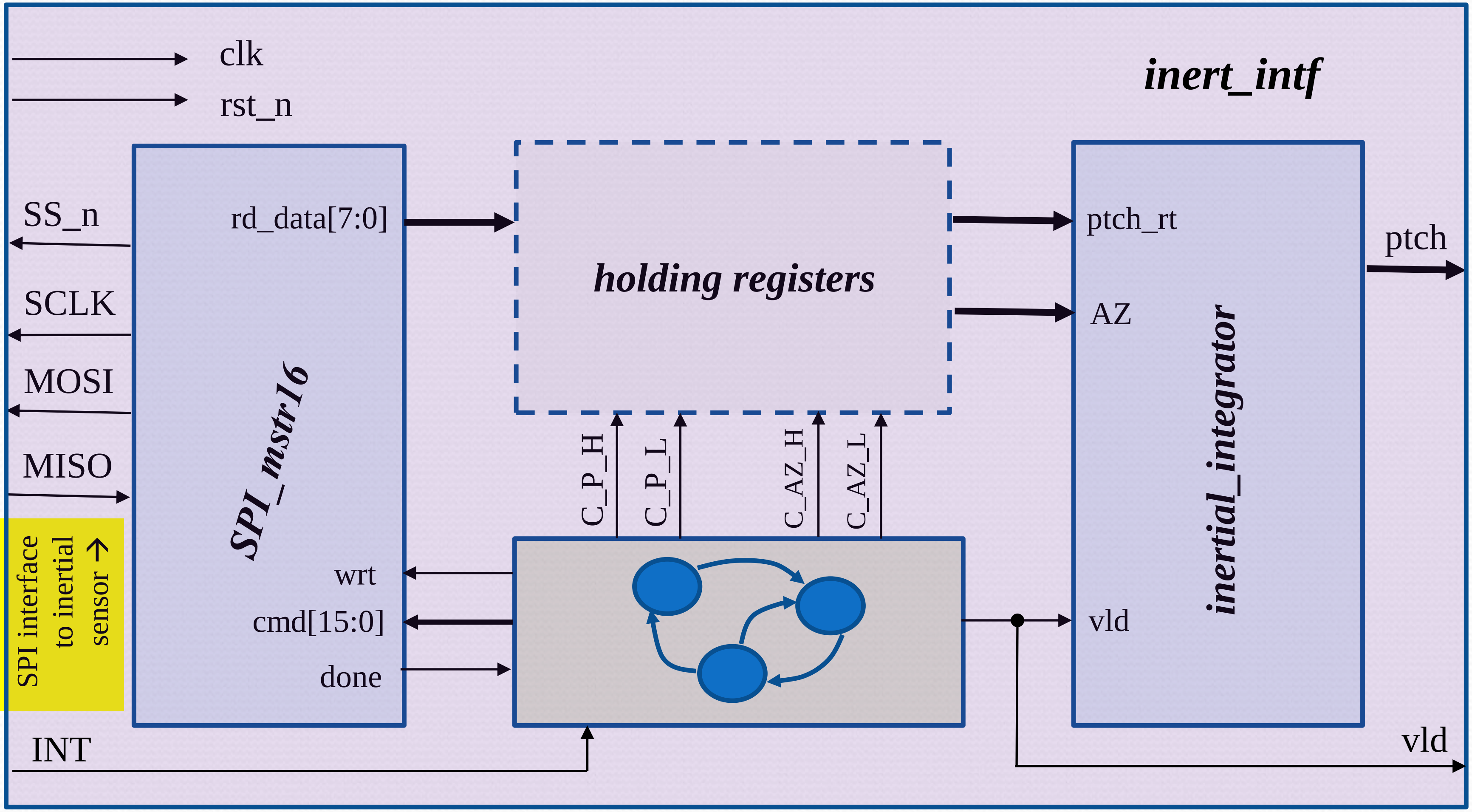
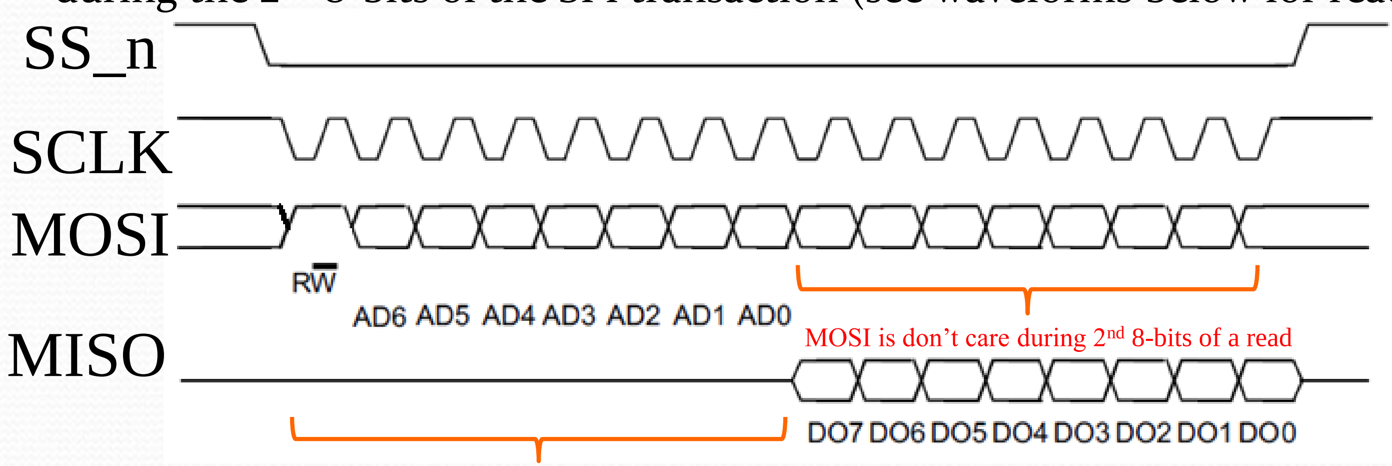
Exercise 21 Inertial Interface:



Exercise 21 Inertial Interface: (read of inertial sensor)

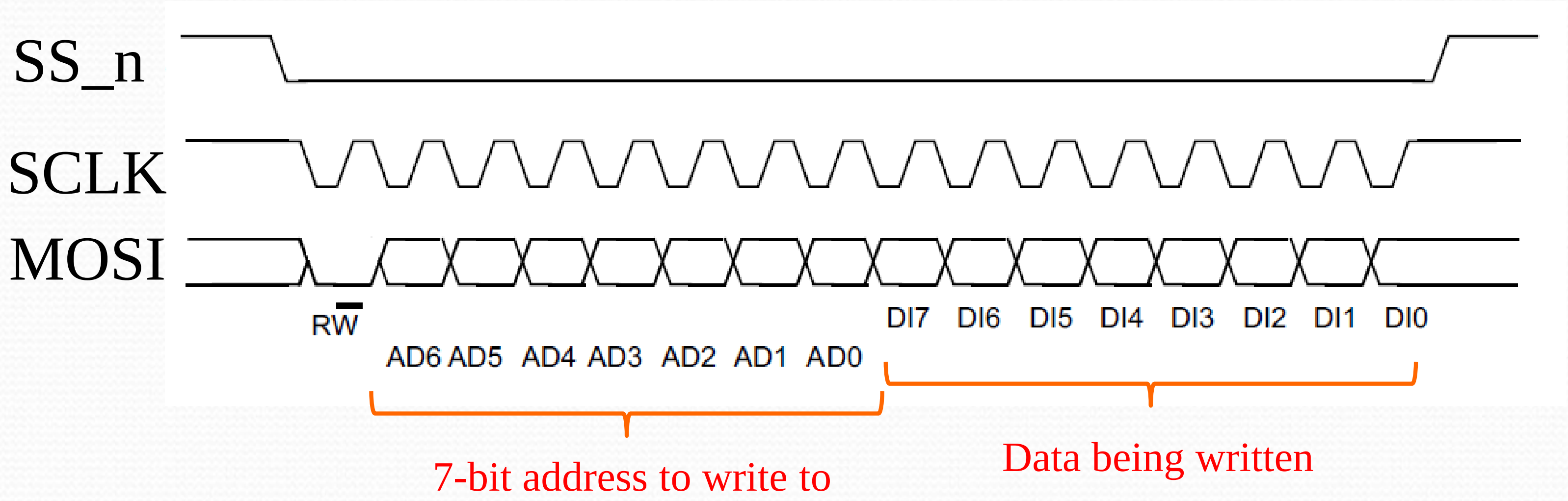
- One of the primary functions of the inertial interface is to drive the SPI interface to configure and perform reads of the inertial sensor.
  - Unlike the A2D which requires two 16-bit transactions to complete a single conversion with the inertial sensor reads/writes are accomplished with single 16-bit 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)



Sensor does drive MISO during first 8-bits of a read, but it is a don't care (garbage)

Exercise 21 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.

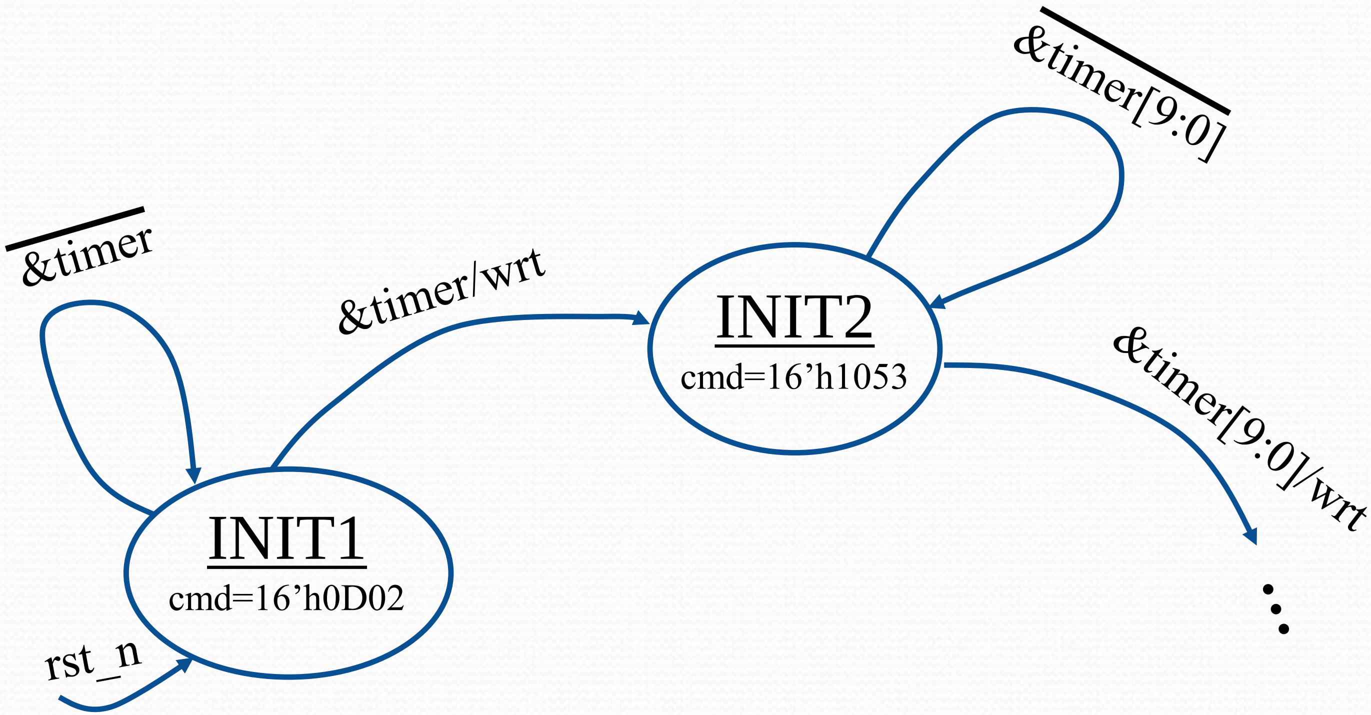
Exercise 21 Inertial Interface: (initializing inertial sensor)

- 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
0x1053	Setup accel for 208Hz data rate, +/- 2g accel range, 50Hz LPF
0x1150	Setup gyro for 208Hz data rate, +/- 245°/sec range.
0x1460	Turn rounding on for both accel and gyro

- 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 your first states in that state-machine are doing.

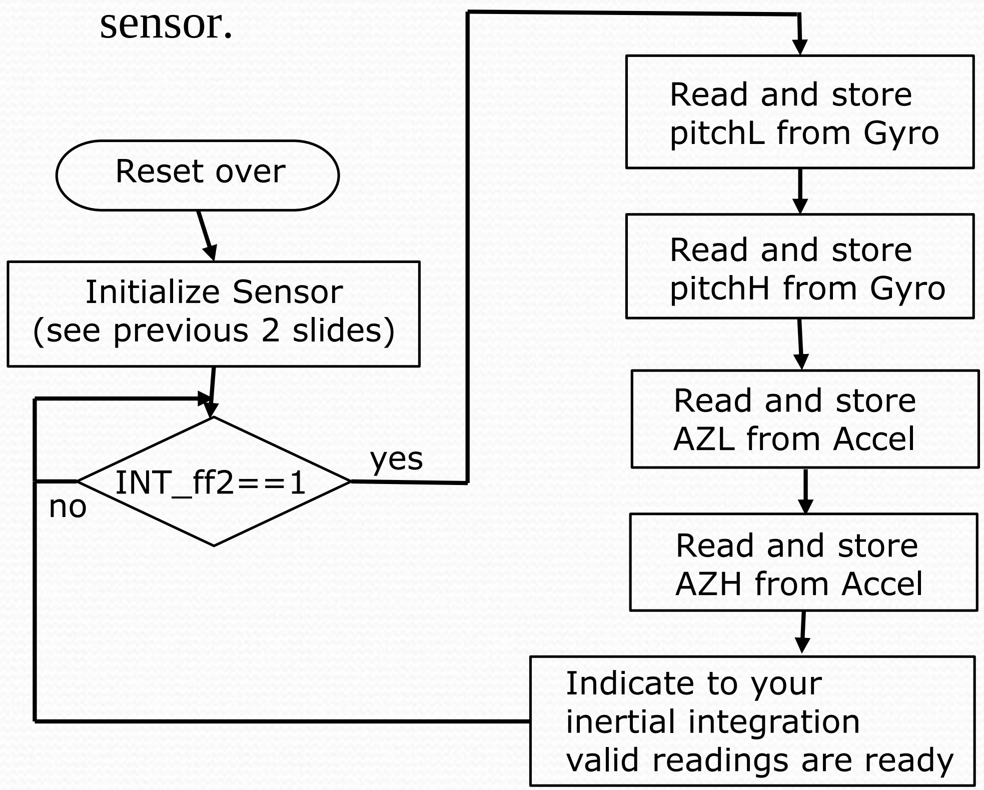
Exercise 21 Inertial Interface: (initializing inertial sensor)



- The inertial sensor has a reset sequence that takes a while. If we start initializing it immediately after our reset it will not be ready. I used a 16-bit timer, and only started the initialization sequence once the 16-bit timer was full. Subsequent writes were based on the lower 10-bits of the timer being full. Of course you could base subsequent writes off of the "done" bit as well.

Exercise 21 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 and accel 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 reads (4 reads in all) from the inertial sensor.



- You will have four 8-bit flops to store the 4 needed readings from the inertial sensor. These are: pitchL, pitchH, AZL, AZH. We are only interested in determining pitch (forward/backward lean) of the platform. Since the sensor is mounted at 90° the AZ component can be used to determine pitch

Exercise 21 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 is a don't care.

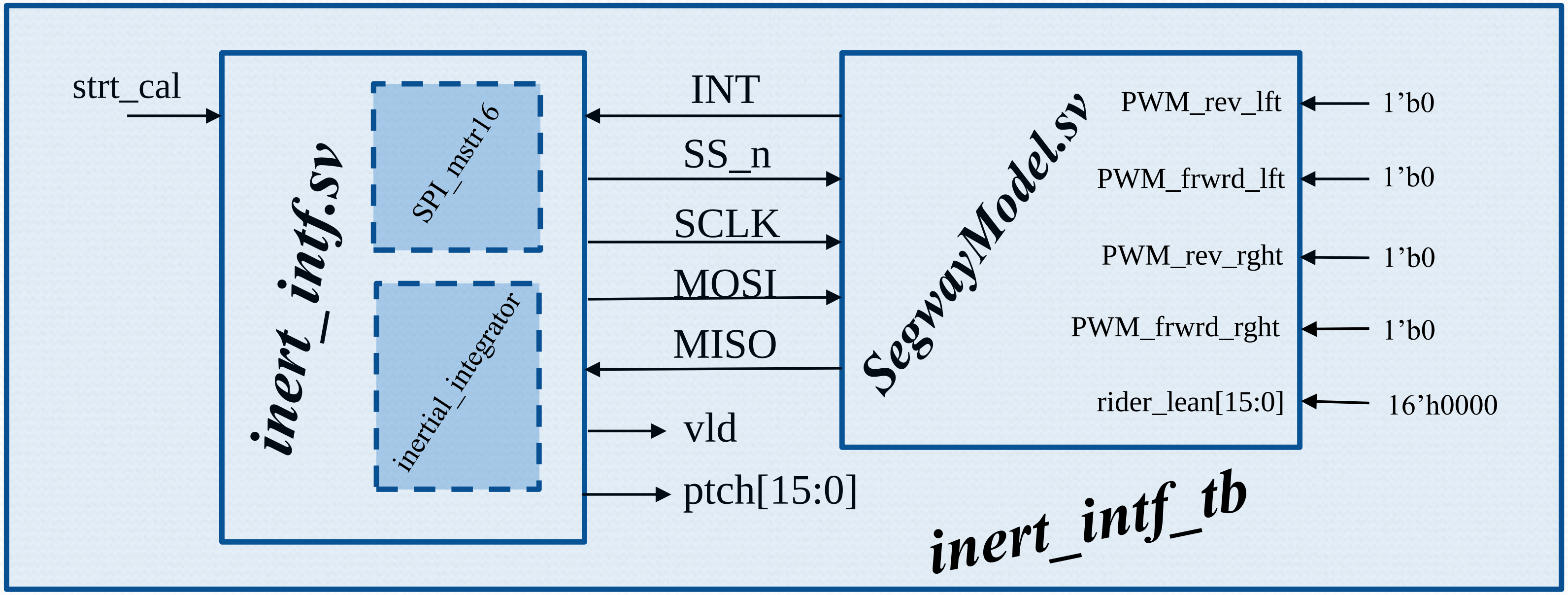
Addr/Data:	Description:
0xA2xx	pitchL → pitch rate low from gyro
0xA3xx	pitchH → pitch rate high from gyro
0xACxx	AZL → Acceleration in Z low byte
0xADxx	AZH → Acceleration in Z high byte

Exercise 21 Inertial Interface: (signal interface)

Signal:	Dir:	Description:
clk, rst_n	in	50MHz clock and active low asynch reset
vld	out	Asserted from SM inside ***inert_intf***. Consumed in ***inertial_integrator***, but also an output to ***balance_cntrl***.
ptch[15:0]	out	This is the primary output of ***inert_intf***. It is the fusion corrected **pitch** of the "Segway" platform.
SS_n, SCLK, MOSI, MISO	out/ in	SPI interface to inertial sensor
INT	In	Interrupt signal from inertial sensor, informing that a new measurement is ready to be read (active high).

Break up your team. Some work on **inert_intf.sv** and some work on a testbench for it.

Exercise 21 Inertial Interface: (testing it)



I provide a block called **SegwayModel.sv**. This models the inertial sensor, and the whole physics of the "Segway" like device. It will be extremely useful for you when you do full project simulations. It is also quite useful for simulating just **inert_intf.sv**, just tie off all the PWM inputs and **rider_lean** to zero as shown above.

What to expect: your testbench should not have to provide stimulus other than a **clk** and **rst_n**. The ***inert_intf*** block should eventually perform 4 initialization writes over SPI to setup the needed registers in ***SegwayModel***. There is a signal in ***SegwayModel*** called **NEMO_setup**. This will go high if you have written the proper registers. After that you should see your ***inert_intf*** block sending out periodic reads over SPI to read ptch low/high and AZ low/high. The **vld** signal from your ***inert_intf*** should pulse high after completing a read sequence. The **ptch** output should slowly creep up from 0x0000 as the simulation progresses.

Exercise 21 **Inertial Interface:**

Submit your favorite internet picture of a cat doing something cute to the dropbox. As you might have figured out by now. You need to do these exercises for your project to work, so I don't require hard proof to trust you are doing them.