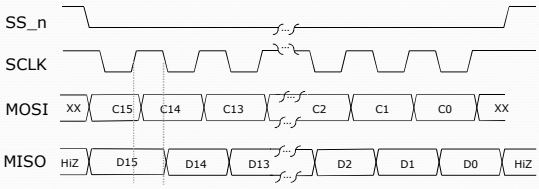
Exercise 8 (HW3 Problem3):

"A friend of mine took 551 last semester and did a SPI peripheral. I will just copy theirs". Nope…this one is specified quite different, but in subtle ways.



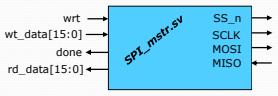
A SPI packet inherently involves a send and receive (full duplex). The full duplex packet is always initiated by the master. Master controls SCLK, SS_n, and MOSI. The slave drives MISO if it is selected. If the slave is not selected it should leave MISO high impedance. The A2D converter is the only SPI peripheral we have in the system.

The SPI master will have a 16-bit shift register. The MSB of this shift register is MOSI. MISO will feed into the LSB of this shift register. The shift register should shift **two system clocks after** the rise of SCLK, this eliminates any timing difficulties. The A2D on the DE0-Nano samples MOSI on the positive edge of SCLK, and changes MISO on the negative edge of SCLK. Of course all your flops are based purely on clk (system clock), not SCLK! SCLK is a signal output from your SPI master.

SCLK will be 1/16 of our system clock (50MHz/16 = 3.125MHz

Exercise 8 (HW3 Problem3):

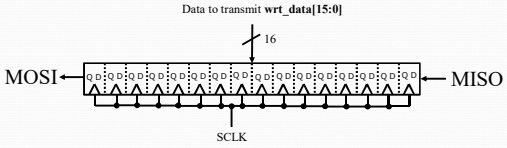
- You will implement **SPI_mstr.sv** with the interface shown.
- SCLK frequency will be 1/16 of the 50MHz clock (i.e. it comes from the MSB of a 4-bit counter running off clk)
- I had better not see any ***always*** blocks triggered directly on **SCLK**. We only use **clk** when inferring flops.
- Remember you are producing **SCLK** from the MSB of a 4-bit counter. So for example, when that 4-bit counter equals 4'b0111 you know **SCLK** rise happens on the next clk. Perhaps more pertinent…when that 4-bit counter equals 4'b1001 you should enable the shift register because you would then force a sample of MISO into the LSB of the shift register at two system clocks after SCLK rise.



Signal:	Dir:	Description:
clk, rst_n	in	50MHz system clock and reset
SS_n, SCLK, MOSI,MISO	3-out 1-in	SPI protocol signals outlined above
wrt	in	A high for 1 clock period would initiate a SPI transaction
wt_data[15:0]	in	Data (command) being sent to inertial sensor or A2D converter.
done	out	Asserted when SPI transaction is complete. Should stay asserted till next **wrt**
rd_data[15:0]	out	Data from SPI slave. For inertial sensor we will only ever use [7:0] for A2D converter we will use bits [11:0]

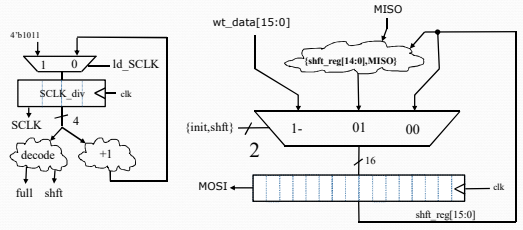
Exercise 8 (HW3 Problem3): Cartoon Diagram!

- Essentially we need a 16-bit shift register that can parallel load data that we want to transmit, then shift it out (MSB first), at the same time it receives data from the slave in the LSB
- The bit coming from the slave (MISO) is shifted into the LSB position of the shift register



- Don't make me regret drawing this diagram by actually implementing it this way. All our flops are always on **clk** rise, nothing else.

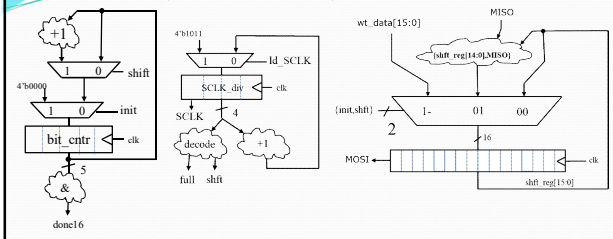
Exercise 8 (HW3 Problem3):



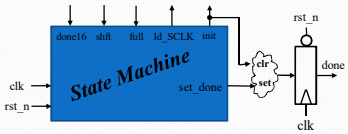
The main datapath of the SPI master consists of a 16-bit shift register. The MSB of this shift register provides MOSI. The shift register can be parallel loaded with the data to send, or it can left shifted one position taking MISO as the new LSB, or it can simply maintain.

Since the SPI master is also generating SCLK it can choose to shift this register in any relationship to SCLK that it desires. To alleviate timing difficulties it is best that the shift register is shifted two system clocks after SCLK rise. Note the value SCLK_div is loaded with (4'b1011). Look back at the waveforms. There is a little time from when SS_n falls till the first fall of SCLK. Do you get the idea of loading with 4'b1011?

Exercise 8 (HW3 Problem3):



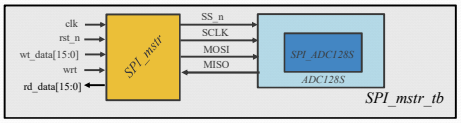
In addition to **SCLK_div** and main shift register you also need a **bit_cntr** to keep track of how many times the shift regiter has shifted. Of course you also need a state machine.



Exercise 8 (HW3 Problem3):

- SCLK Requirements:
  - SCLK will be 1/16 of our system clock (50MHz/16 = 3.125MHz)
  - SCLK is normally high and toggles during SPI transactions
  - Want a delay from start of transaction (SS_n fall) till first fall of SCLK
  - Look back 2-slides at the waveworms. We want a bit of a "back porch" on SCLK. A time in which it is high prior to SS_n returning high.
- Recommended SCLK implementation
  - SCLK comes from bit[3] of a 4-bit counter
  - This 4-bit counter is only counts during SPI transactions (otherwise loads 4'b1011)
  - The bits of this counter are not all preset or reset, but rather a combination such that SCLK is normally high and has its first negative edge a few system clocks after the transaction starts.
  - Perhaps will need to dedicate a state to creating the "back porch".
- Remember…for DUT Verilog (Verilog you intend to synthesize). If I see: ***always @(posedge*** … ← This next signal better be ***clk*** or I am going to blow a gasket.

Exercise 8 (HW3 Problem3) (Testing your SPI_mstr.sv):



- Create **SPI_mstr.sv** block
- Download **ADC128S.sv** (model of A2D converter on DE0-Nano, and a SPI slave)
- Also download **SPI_ADC128S.sv** (child of ADC128S.sv that you need)
- Create a testbench in which the **SPI_mstr.sv** drives the **ADC128S**. Test and debug.
- To read a channel from the ADC128S you send: {2'b00,chnl[2:0],11'h000} (i.e. the channel is specified by bits [13:11] of the packet you send.
- During a read the ADC128S is returning the channel you requested in the last SPI packet. Since it obviously cannot respond with data for the current SPI packet since you are just now telling it what channel you want.

Exercise 8 (HW3 Problem3) (Testing your SPI_mstr.sv):

- The response of ADC128S is: 0xC00 + chnnl for the first two reads. The 0xC00 part decrements by 0x10 for every 2 reads. For the first read it assumes you are reading channel 0 so it would return 0xC00.
- If you gave it 4 reads in a row:

NOTE: when performing consecutive reads to the **ADC128S.sv** model you have to give it a clock period to breath between transactions. So delay one system clock after ***done*** before sending another SPI transaction.

Channel Read	Expected Response	Description:
2	0xC00	You are requesting channel 2 for next time, but it returns channel 0 for first read.
2	0xC02	Has not decremented 0xC00 by 0x10 yet, but this is channel 2 from last request
3	0xBF2	Two reads have been performed so it decremented by 0x10, but this is still channel 2.
3	0xBF3	This is a channel 3 response from last request

- **Submit:**
  - **SPI_mstr.sv (this is individual exercise, everyone submits their own)**
  - Your testbench (**SPI_mstr_tb.sv**) (should be self-checking, and I recommend what is shown in table above)
  - Output from your self checking test bench proving you ran it successfully