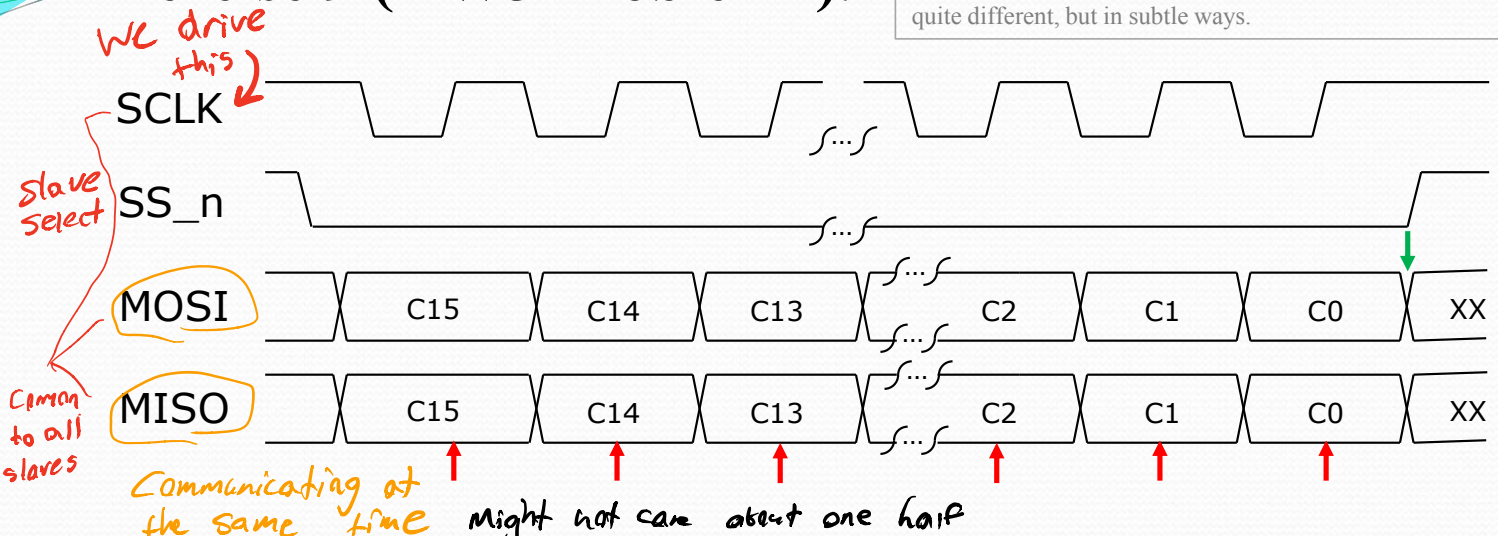
Serial protocol

Master-slave based

Exercise 9 (HW3 Problem4):

"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.



Shown above is a 16-bit SPI packet. The master is changing (shifting) **MOSI** on the falling edge of **SCLK**. The slave device (6-axis inertial sensor) changes **MISO** on the falling edge too. We sample **MISO** on the rising edge (see red arrows).

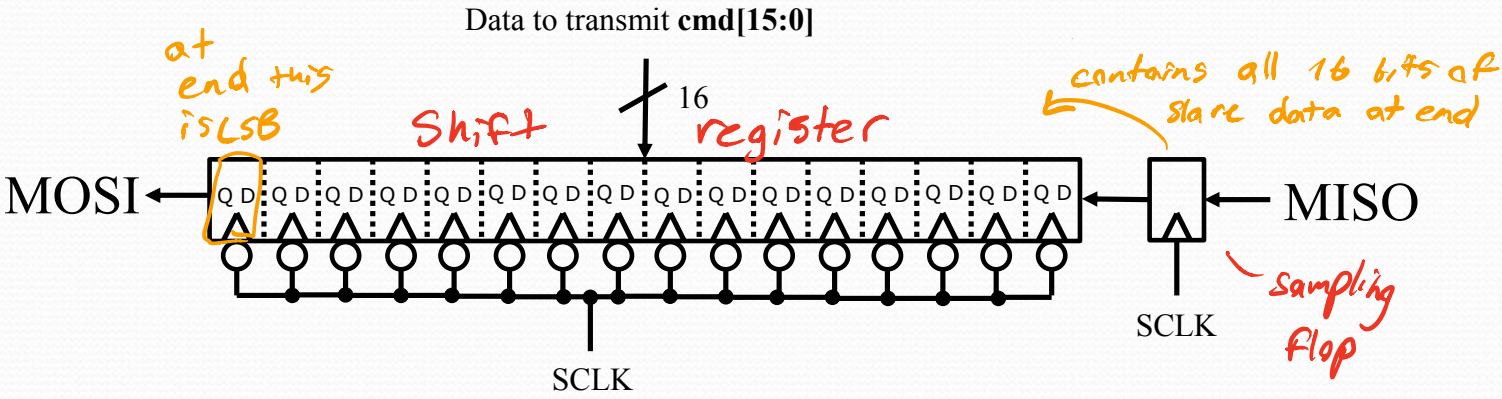
The sampled version of **MISO** in turn gets shifted into our 16-bit shift register. (on **SCLK** fall)

When **SS_n** first goes low there is a bit of a period before **SCLK** goes low. Our 16-bit shift register does not shift on the first fall of **SCLK**. This is called the "front porch".

At the end of the transaction C0 from the slave (on **MISO**) is sampled on the last rise of **SCLK**. Then there is a bit of a "back porch" before **SS_n** returns high. When **SS_n** returns high we shift our 16-bit shift register one last time (see green arrow) so "C0" captured on **SCLK** rise (last red arrow) is shifted into our shift register and we have received 16-bits from the slave.

Exercise 9 (HW3 Problem4 (SPI)):

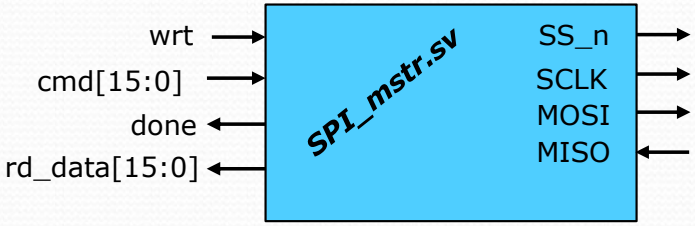
- Essentially we need a 16-bit shift register that can parallel load data that we want to transmit, then shift it out (MSB first), at the same time it receives data from the slave in the LSB
- The bit coming from the slave (MISO) is sampled on the rise of SCLK, and then put into our shift register on the fall of SCLK.



- 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 9 (HW3 Problem3):

- You will implement **SPI_mstr.sv** with the interface shown.
- SCLK frequency will be 1/64 of the 50MHz clock (i.e. it comes from the MSB of a 6-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 6-bit counter. So for example, when that 6-bit counter equals 6'b011111 you know **SCLK** rise happens on the next clk, so you can enable a sample of **MISO** then. Similar logic is used for when to shift the main 16-bit shift register.

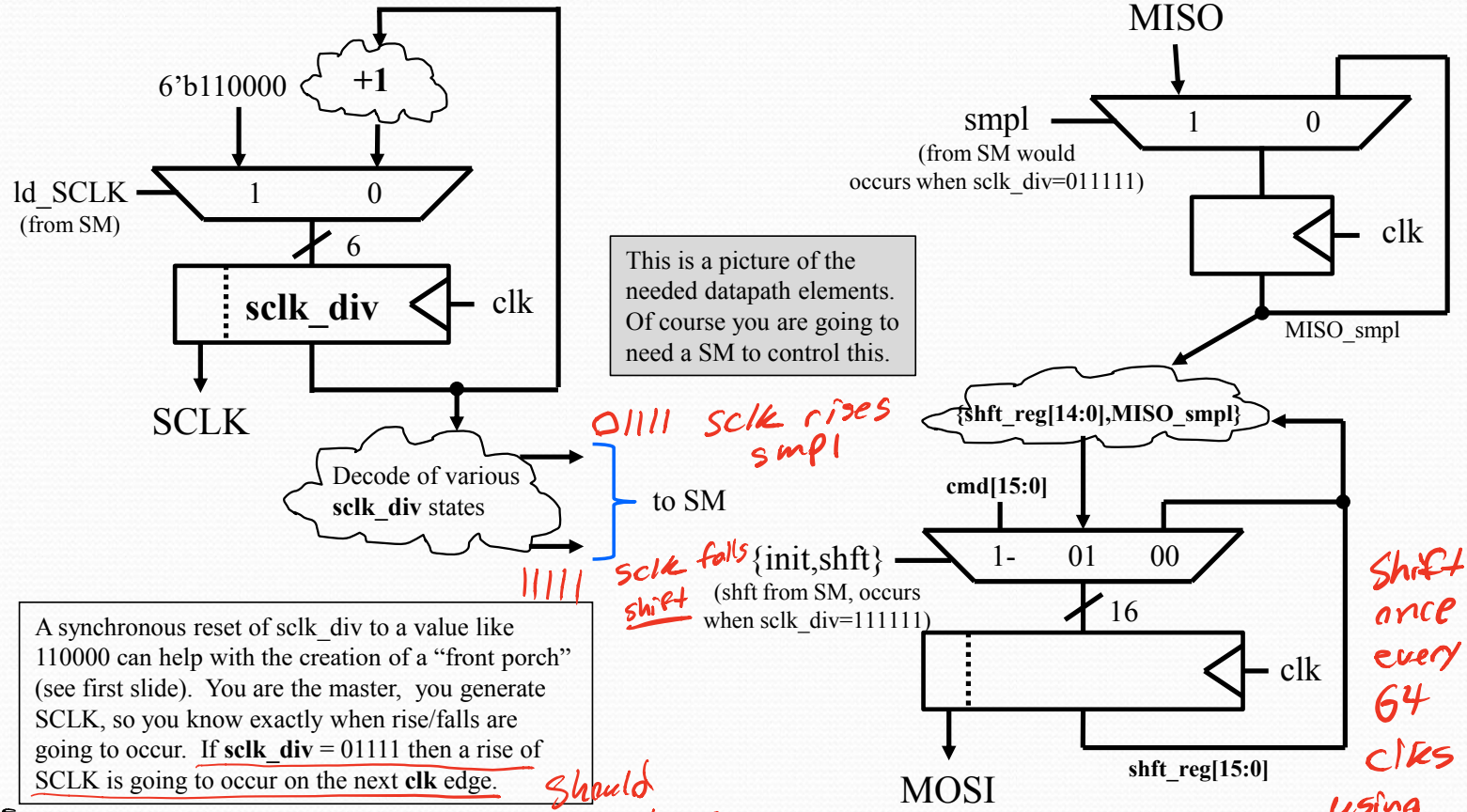


Signal:	Dir:	Description:
clk, rst_n	in	50MHz system clock and reset
SS_n, SCLK, MOSI,MISO	3-out 1-in	SPI protocol signals outlined above
wrt	in	A high for 1 clock period would initiate a SPI transaction
cmd[15:0]	in	Data (command) being sent to inertial sensor or A2D converter.
done	out	Asserted when 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 9 SPI Tranceiver Hints:

SCLK is an output of the SPI master, and is to be 1/64 of the system clock. Therefore it can come from the MSB of a 6-bit counter (**sclk_div**).

"Heart of you SPI master is a 16-bit shift register. The MSB of this shift register forms MOSI. A version of MISO sampled at "SCLK rise" is shifted in as the LSB. The register is shifted during a SPI transaction at "SCLK fall".



This is a picture of the needed datapath elements. Of course you are going to need a SM to control this.

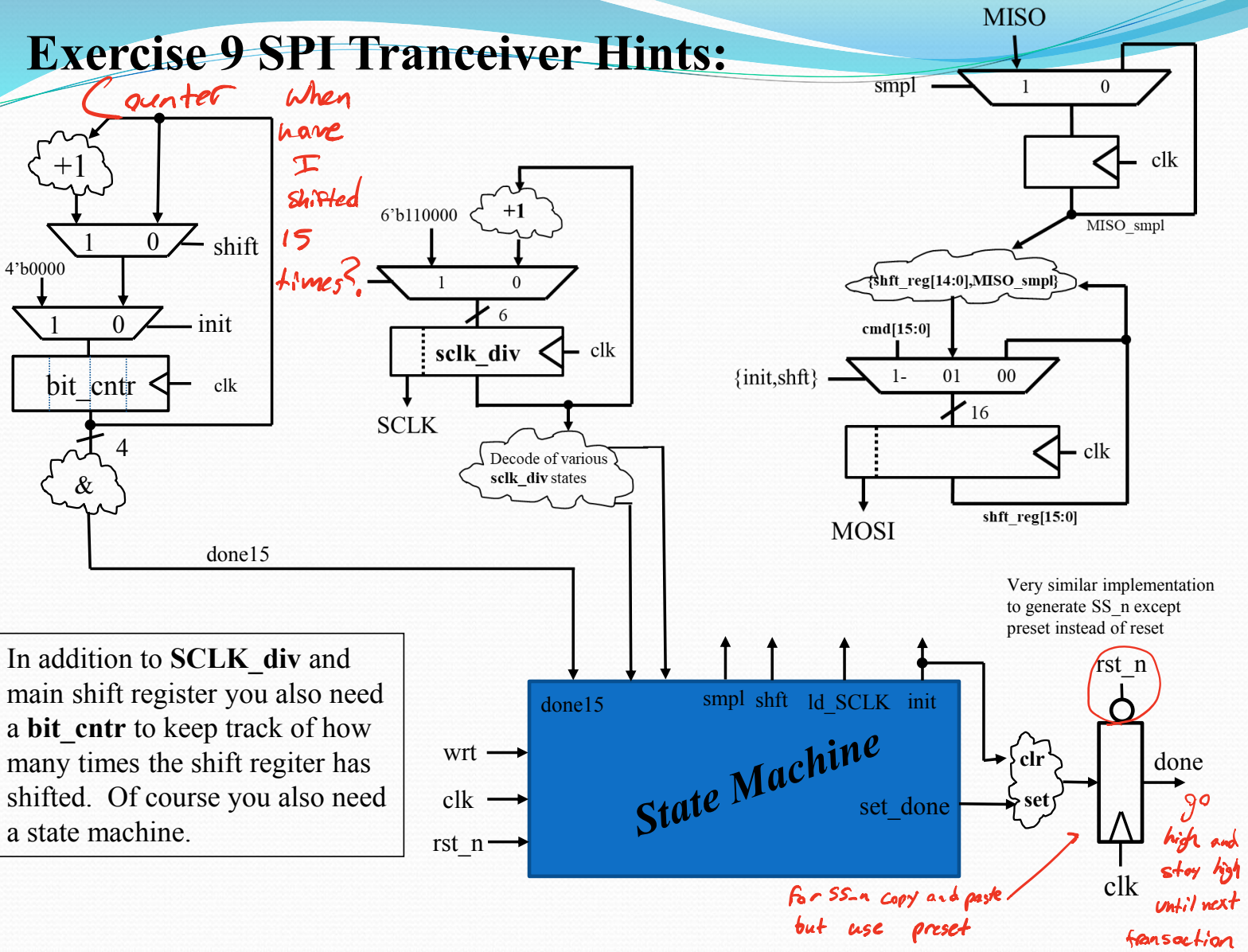
A synchronous reset of sclk_div to a value like 110000 can help with the creation of a "front porch" (see first slide). You are the master, you generate SCLK, so you know exactly when rise/falls are going to occur. If **sclk_div** = 01111 then a rise of SCLK is going to occur on the next **clk** edge.

Shift once every 64 clks using sclk_div

Should sample slaves data

1's SCLK ↷ 10000

Exercise 9 SPI Tranceiver Hints:



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.

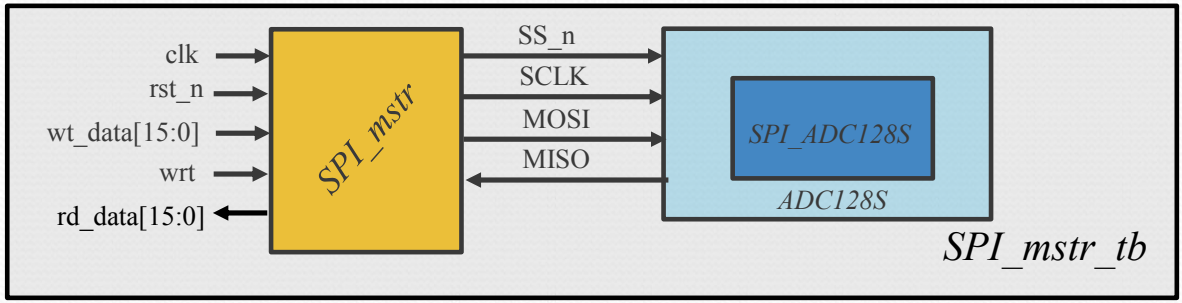
Exercise 9 (HW3 Problem3):

- SCLK Requirements:
  - SCLK will be 1/64 of our system clock (50MHz/64 = 781kHz)
  - 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 at the waveforms. 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[5] of a 6-bit counter
  - This 6-bit counter is only counts during SPI transactions (otherwise loads 6'b110000)
  - 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 9 (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 9 (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:
1	0xC00	You are requesting channel 2 for next time, but it returns channel 0 for first read.
1	0xC01	Has not decremented 0xC00 by 0x10 yet, but this is channel 1 from last request
4	0xBF1	Two reads have been performed so it decremented by 0x10, but this is still channel 1.
4	0xBF4	This is a channel 4 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