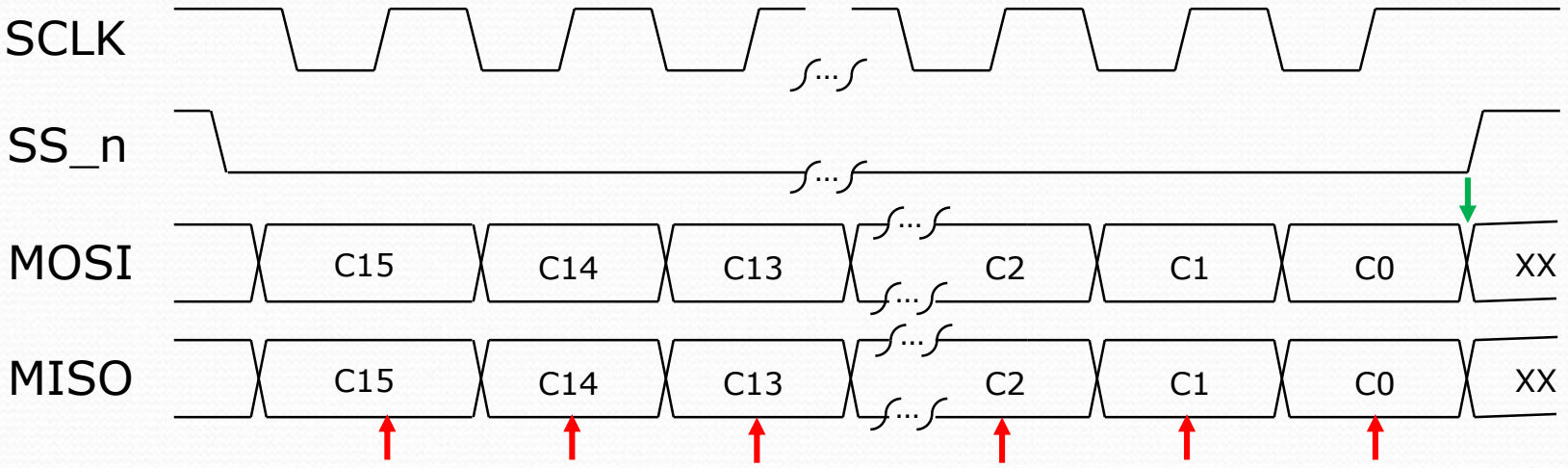
Exercise 15 SPI Tranceiver (*HW4 Problem 5)*

- Simple Master/Slave serial interface (Motorola long long ago)
  - **S**erial **P**eripheral **I**nterconnect (very popular physical interface)
  - 4-wires for full duplex
    - ✓ MOSI (Master Out Slave In) (We drive this (we are master) to 6-axis inertial)
    - ✓ MISO (Master In Slave Out) (Inertial sensor drives this back to us)
    - ✓ SCLK (Serial Clock)
    - ✓ SS_n (Active low Slave Select) (For us we only have one SS per SPI channel)
  - There are many different variants
    - ✓ MOSI shifted on SCLK rise vs fall, MISO sampled on SCLK rise vs fall
    - ✓ SCLK normally high vs normally low
    - ✓ Widths of packets can vary from application to applications
    - ✓ Really is a very loose standard (barely a standard at all)
  - We will stick with:
    - ✓ SCLK normally high, 16-bit packets only
    - ✓ MOSI shifted on SCLK fall
    - ✓ MISO sampled on SCLK rise

Exercise 15 SPI Tranceiver (*HW4 Problem 5*)



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).

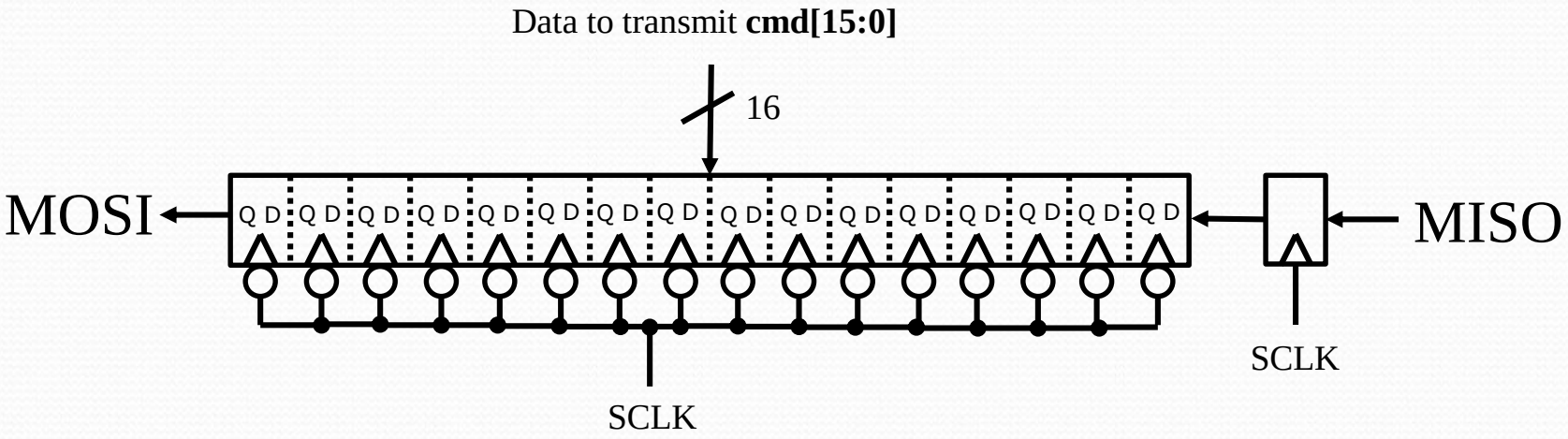
The sampled version of **MISO** in turn gets shifted into our 16-bit shift register. (on **SCLK** fall)

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 15 SPI Tranceiver (*HW4 Problem 5)*

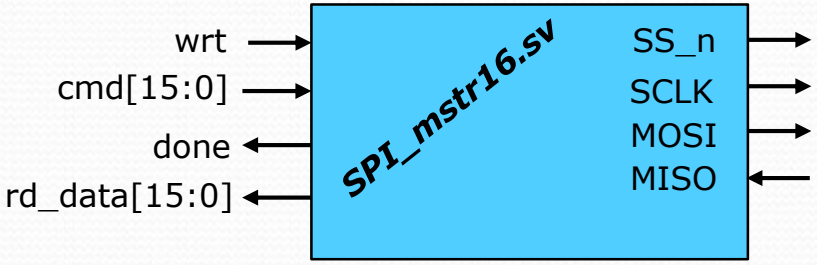
- Essentially we need a 16-bit shift register that can parallel load data that we want to transmit, then shift it out (MSB first), at the same time it receives data from the slave in the LSB
- The bit coming from the slave (MISO) is sampled on the rise of SCLK, and then put into our shift register on 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 15 SPI Tranceiver (*HW4 Problem 5*)

- Both the 6-axis inertial sensor, and the A2D on the DE-0 Nano board can be read with a SPI master that implements the 16-bit SPI transaction mentioned above.
- You will implement **SPI_mstr16.sv** with the interface shown.
- SCLK frequency will be 1/32 of the 50MHz clock (i.e. it comes from the MSB of a 5-bit counter running off clk)
- Although the description says things like: "the shift register is shifted on **SCLK** fall" and "**MISO** is sampled on **SCLK** rise". 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 5-bit counter. So for example, when that 5-bit counter equals 5'b01111 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.

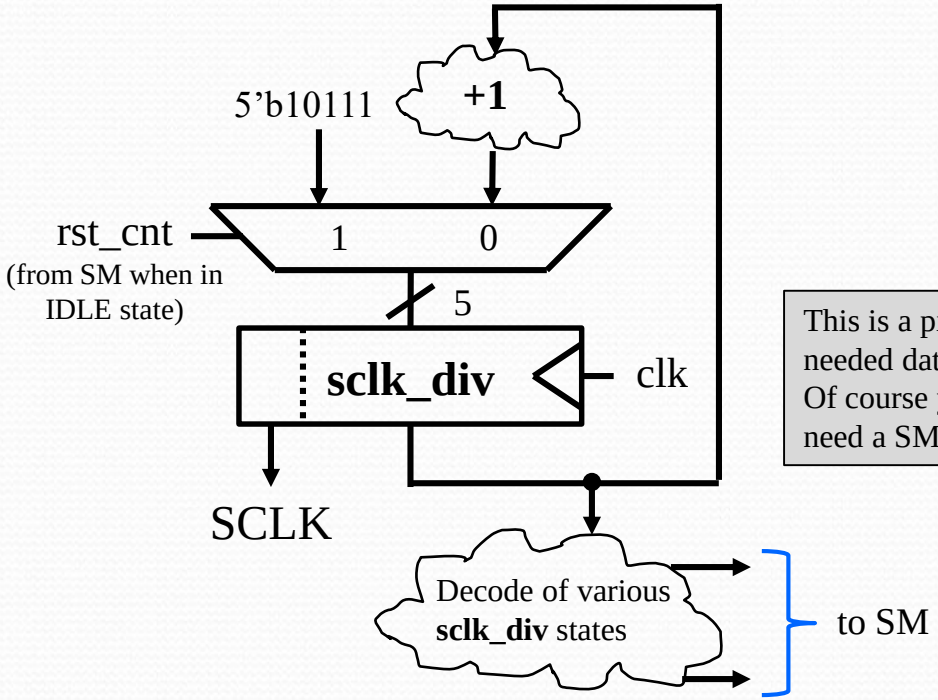


Signal:	Dir:	Description:
clk, rst_n	in	50MHz system clock and reset
SS_n, SCLK, MOSI, MISO	out/ in	SPI protocol signals outlined above
wrt	in	A high for 1 clock period would initiate a SPI transaction
cmd[15:0]	in	Data (command) being sent to inertial sensor or A2D converter.
done	out	Asserted when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

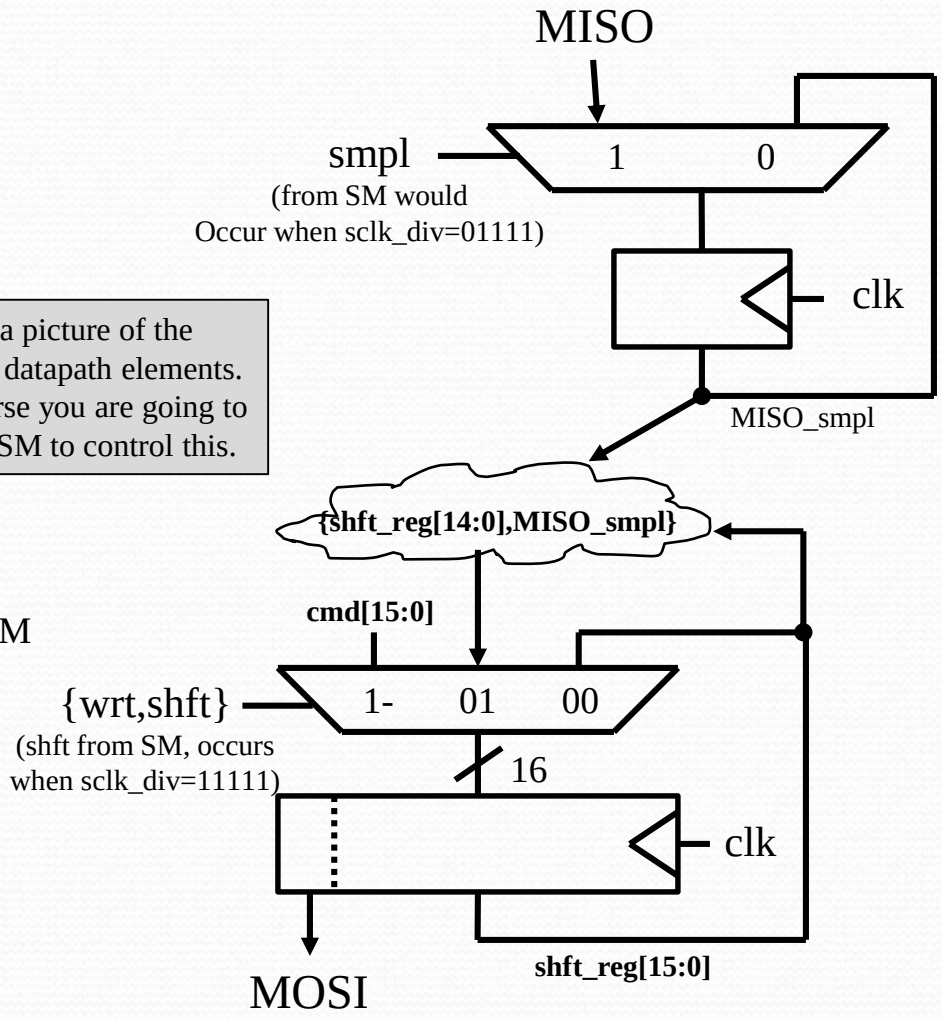
Exercise 15 SPI Tranceiver Hints:

SCLK is an output of the SPI master, and is to be 1/32 of the system clock. Therefore it can come from the MSB of a 5-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".

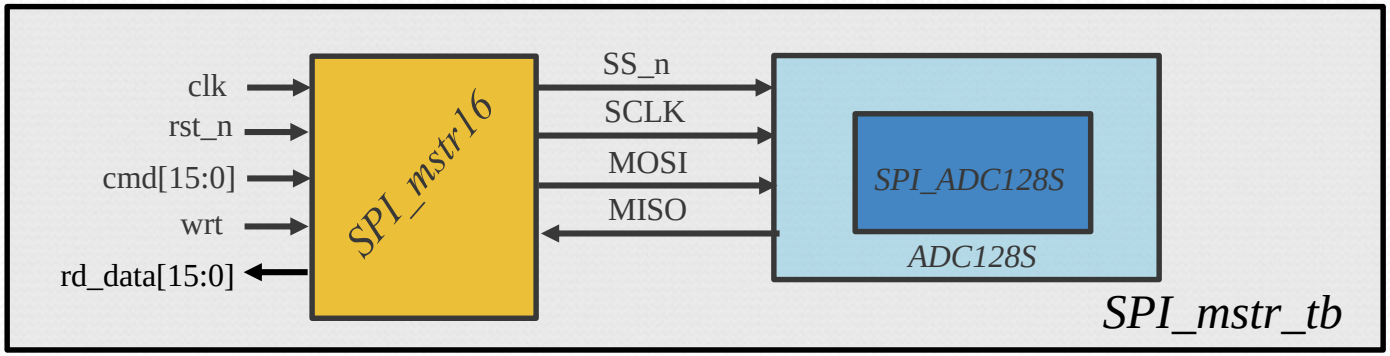


This is a picture of the needed datapath elements. Of course you are going to need a SM to control this.



A synchronous reset of sclk_div to a value like 10111 can help with the creation of a "front porch" (see previous 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.

Exercise 15 Testing SPI Tranceiver:



- Create **SPI_mstr16.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_mstr16.sv** drives the **ADC128S**. Test and debug. **NOTE:** ADC128S.sv produces a warning for reading any channel other than 0,4, and 5 (the channels we use on "Segway" project).
- 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 15 Testing SPI Tranceiver:

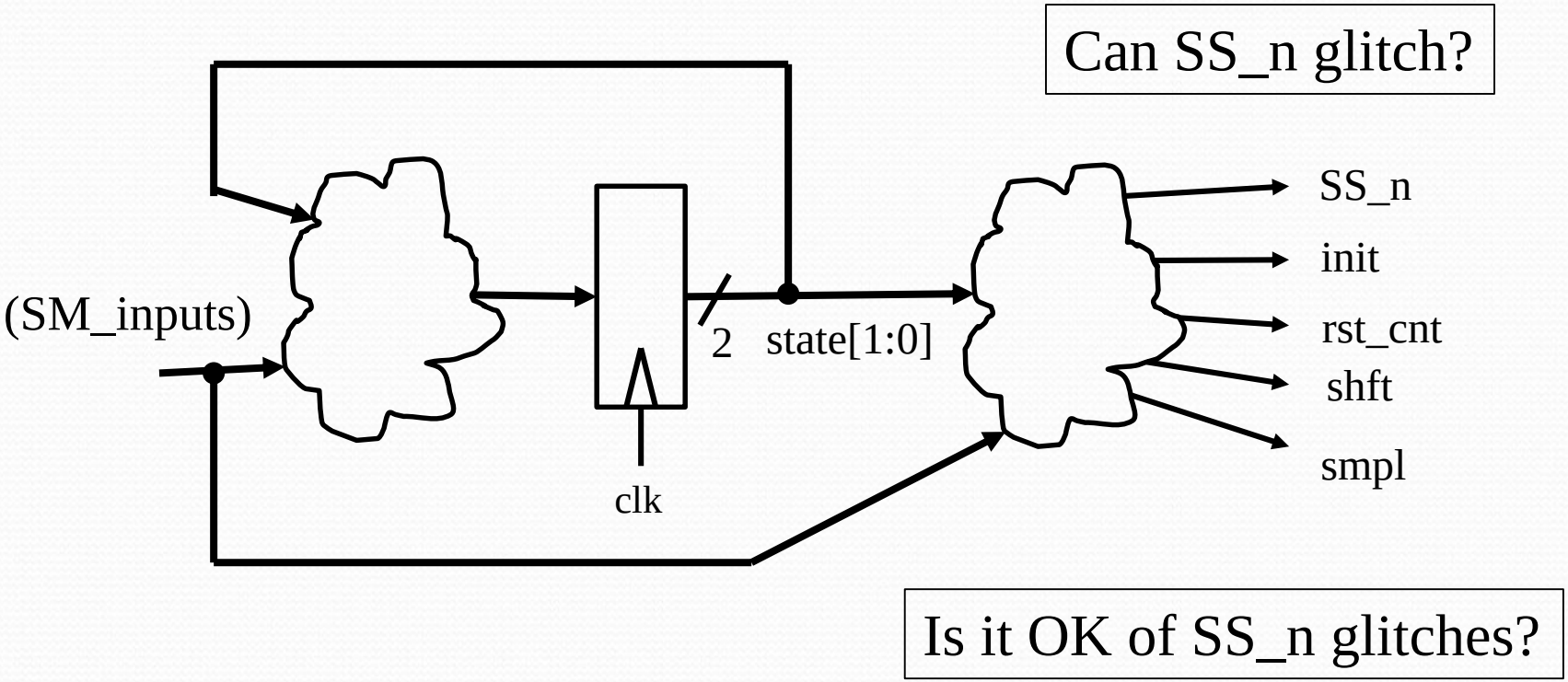
- The response of ADC128S is: 0xC00 + chnnl for the first two reads. The 0xC00 part decrements by 0x10 for every 2 reads. For the first read it assumes you are reading channel 0 so it would return 0xC00.

- If you gave it 4 reads in a row:

Channel Read	Expected Response	Description:
5	0xC00	You are requesting channel 5 for next time, but it returns channel 0 for first read.
5	0xC05	Has not decremented 0xC00 by 0x10 yet, but this is channel 5 from last request
4	0xB05	Two reads have been performed so it decremented by 0x10, but this is still channel 5.
4	0xB04	This is a channel 4 response from last request

- **Submit:**
  - **SPI_mstr16.sv (this is individual exercise, everyone submits their own)**
  - Your testbench (**SPI_mstr_16_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

Possible SPI SM Implementation:



Correct SPI SM Implementation: