# ECE 551

## HW4 *(100 pts)*

- Due Friday Apr 3rd@ 11:00AM
- Work Individually
- Use descriptive signal names
- Comment & indent your code
- Code will be judged on coding style

# HW4 Problems 1&2 (**10pts) + (10pts)**

1. **(10pts)** Complete the Synopsys Design Vision tutorial. Sign below, (preferably in blood).

I, _____ completed the Synopsys Design Vision tutorial. If I had any problems with it, I discussed them with the TA or Instructor, either in person, or through email.

2. **(10pts)** Project Team Formation….become one with yourself:

I, _____ am a team of 1 and I will do the project on my own because I am awesome. I will not collaborate with others because I know Hoffman will catch me, (he has code comparison tools) and because I have too much pride for that.

Sign these problems (either electronically or print, sign and take a picture) and turn in an image:
**Probs1and2.jpg**

# HW4 Problem 3 (**20pts)** Synthesize your UART

Started as Exercise14 on Mon Mar 23rd

- In HW3 you produced a UART transmitter (UART_tx.sv) and a UART receiver (UART_rx.sv). Combine the two module (by simply instantiating each) into a UART transceiver (UART.sv).

- In this HW you will synthesize both modules using Synopsys on the CAE linux machines.

- **NOTE:** We started this as an exercise and you might have done it with a partner. That is fine, but both submit individually.

- Write a synthesis script to synthesize your UART.sv. The script should perform the following:
  - Defines a clock of 500MHz frequency and sources it to clock
  - Performs a set don't touch on the clock network
  - Defines input delays of 0.5 ns on all inputs other than clock
  - Defines a drive strength equivalent to a 2-input nand of size 1 from the Synopsys 32nm library (NAND2X1_RVT) for all inputs except clock and rst_n
  - Defines an output delay of 0.5ns on all outputs.
  - Defines a 0.10pf load on all outputs.
  - Sets a max transition time of 0.15ns on all nodes.
  - Employs the Synopsys 32nm wire load model for a block of size 16000 sq microns
  - Compiles, then flattens the design so it has no hierarchy, and compiles again.
  - Produces a min_delay & max delay report
  - Produces an area report
  - Flattens the design so it has no hierarchy
  - Writes out the gate level verilog netlist (**UART.vg**)
  - Writes out a SDC file. (**UART.sdc**)

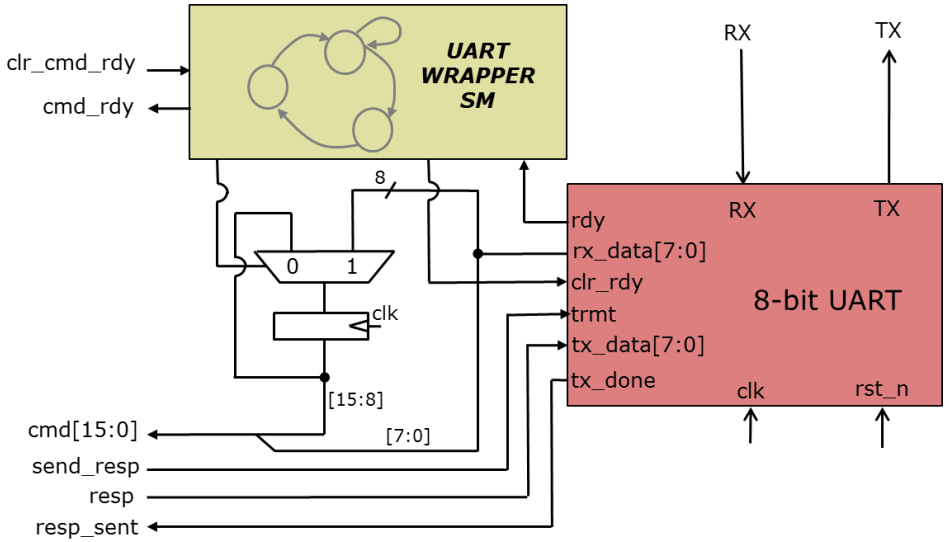
- Submit to the dropbox.
  - Your synthesis scripts (UART.dc)
  - The output reports for area (UART_area.txt)
  - The gate level verilog netlist (UART.vg)

# HW4 Problem 4 (**25pts)** UART Wrapper

Started as Exercise15 on Weds Mar 25th

- The Logic Analyzer receives 16-bit command from the ne… These commands are sent via UART (a byte based protocol). You need t… a wrapper to package two bytes into a single 16-bit command.

- A diagram of the datapath and control needed are shown




- All commands to the Logic analyzer are responded to with at least 1-byte responses. (Either a positive acknowledge, or data read, or a dump of channel data)

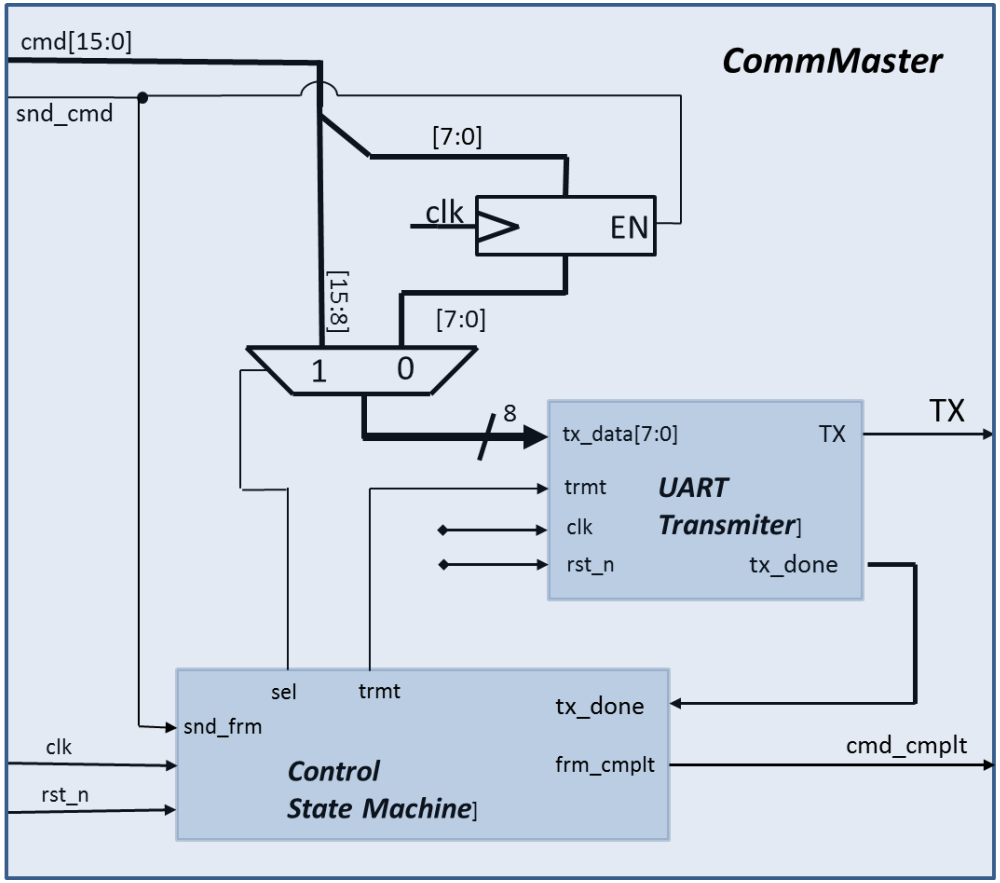
- Create **UART_wrapper.sv**.

- Testing **UART_wrapper** on its own is not so productive. See the next 2 slides first.

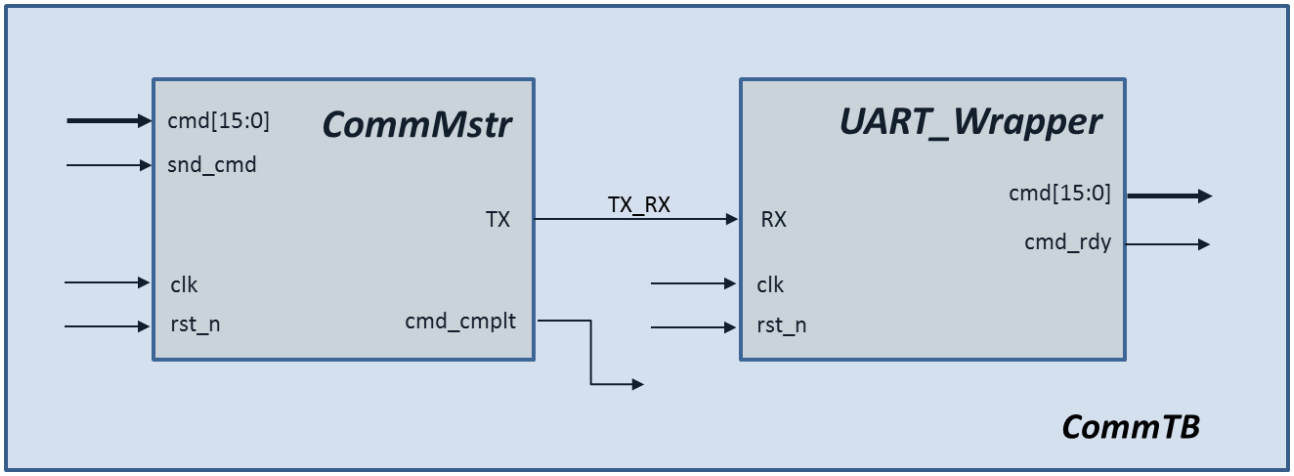
# HW4 Problem 4 (**25pts)** CommMaster

- The last problem on your midterm was creating **CommMaster.sv** that did the opposite (took a 16-bit command and sent it as two 8-bit bytes over UART).

- Create **CommMaster.sv**
- See next slide for testing hints.



# HW4 Problem 4 (**25pts)** Testing UART_wrapper and CommMaster




- Once you have both ***CommMstr*** and ***UART_wrapper*** creating a self checking testbench is easier. Simply instantiate both as shown above.
- Send a 16-bit command via ***CommMstr***. Look for **cmd_rdy**. When it asserts does the received **cmd** match the sent command?
- Submit **UART_wrapper.sv**, **CommMaster.sv**, and **CommTB.sv** to the dropbox.

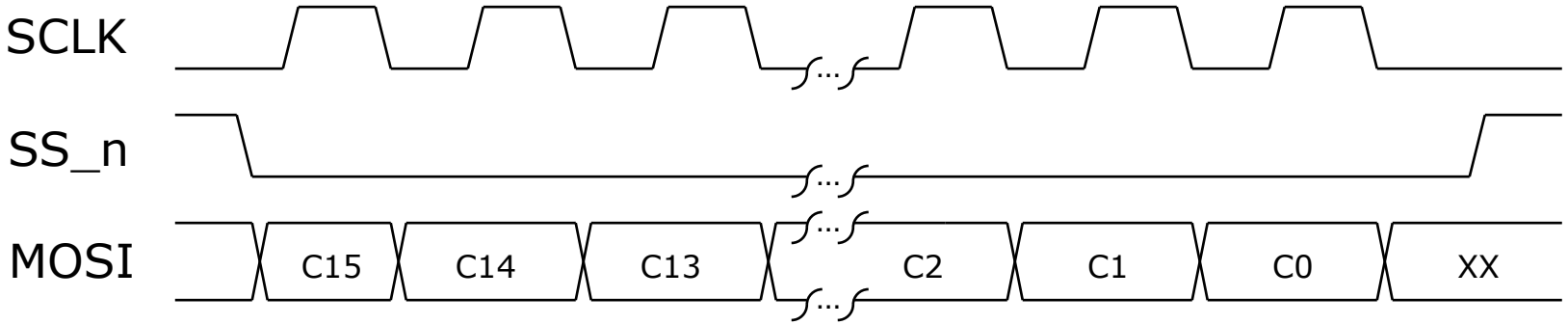
# HW4 Problem 5 (**35pts)** SPI receiver for use as protocol trig

Started as Exercise16 on Mon Mar 30th

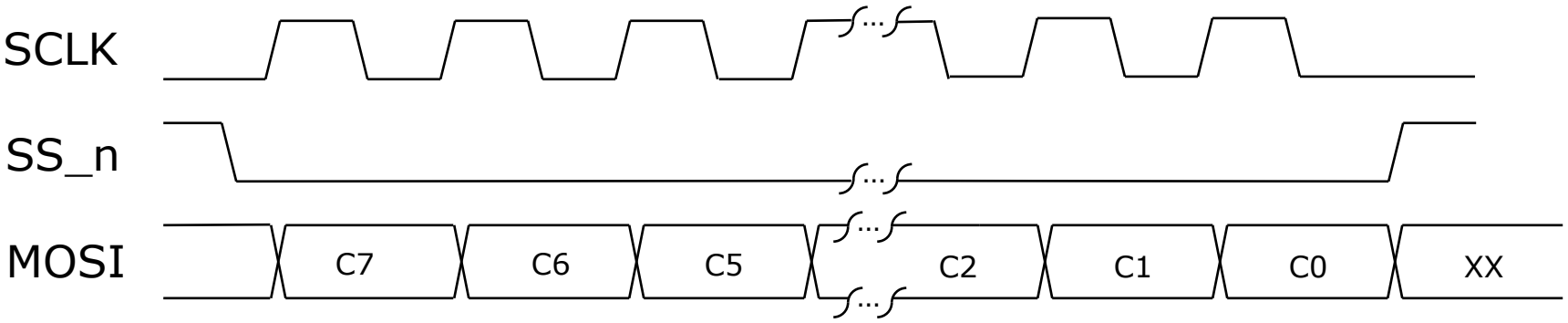
What is SPI

- Simple uni-directional 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) (digital core will drive this to AFE)
    - ✓ MISO (Master In Slave Out) (not used in connection to AFE digital pots, only EEP)
    - ✓ SCLK (Serial Clock)
    - ✓ SS_n (Active low Slave Select) (Our system has 4 individual slave selects to address the 4 dual potentiometers, and a fifth to address the EEPROM)
  - There are many different variants
    - ✓ MOSI Sampled (shifted) on clock low vs clock high
    - ✓ 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 low, but have configuration for the rest
    - ✓ MOSI sampled on SCLK rise if ***edg*** is high, SCLK fall if ***edg*** is low.
    - ✓ 8-bit packets if ***len8_16*** is high, and 16-bit packets if ***len8_16*** is low.

# SPI Packets



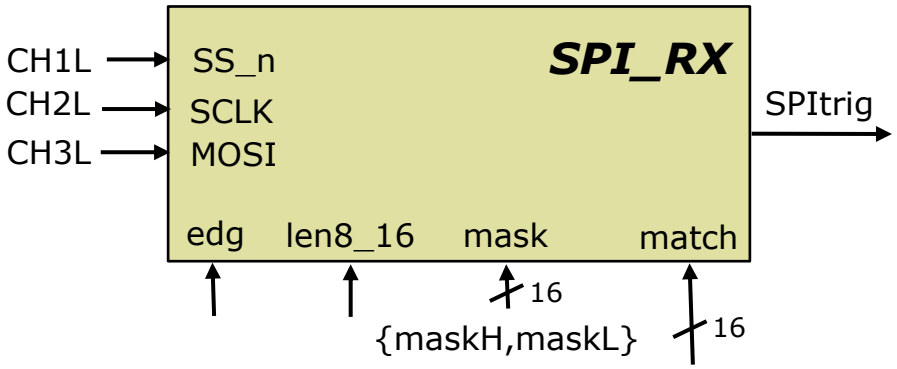

Shown above is a 16-bit SPI packet where the master is changing (shifting) MOSI on the falling edge of SCLK. This is a case where one would probably choose to have ***edg*** high so MOSI was sampled on SCLK rise.




Shown above is an 8-bit SPI packet where the master is changing (shifting) MOSI on the rising edge of SCLK. This is a case where one would probably choose to have ***edg*** low so MOSI was sampled on SCLK fall.

# SPI Unit for Protocol Triggering

- Make a SPI_RX module. It should not operate with ***SCLK*** as a clock, but rather treat ***SCLK*** as a signal it samples using the main 100MHz system ***clk***.
- The SPI module should continue to sample and shift ***MOSI*** (on the specified edge of ***SCLK*** (specified by ***edg***)) until ***SS_n*** returns high.
- When ***SS_n*** returns high the SPI transaction can be considered complete and the contents in the receive shift register can be compared to ***match[15:0]***.
- if ***len8_16*** is high then comparison is only 8-bits, and the lower 8-bits of the receive shift register are compared against ***match[7:0]***.
- If ***edg*** is high then the receive shift register should shift (sample ***MOSI***) on the rise of ***SCLK***. If ***edg*** is low then the shift is on the fall of ***SCLK***.
- Mask is used to mask off the bits of match and treat them as a don't care in the comparison. High bits in mask represent bits of match that should be treated as don't care.




| Signal: | Dir: | Description: |
|---|---|---|
| clk, rst_n | in | 100MHz system clock and reset |
| SS_n, SCLK, MOSI | in | SPI protocol signals. Coming from VIL comparators |
| edg | in | When high the receive shift register should shift on SCLK rise. |
| len8_16 | in | When high we are doing an 8-bit comparison to ***match[7:0]*** |
| mask[15:0] | in | Used to mask off bits of ***match*** to a don't care for comparison |
| match[15:0] | in | Data unit is looking to match for a trigger |
| SPItrig | out | Asserted for 1 clock cycle at end of a reception if received data matches ***match*** |

# HW4 Problem 5 (**35pts)** SPI receiver for use as protocol trigger

- Create SPI_RX block
- Download SPI_TX module from the HW4 folder that you can use for testing.
- Link the two blocks together to form a self-checking test bench.
- Submit:
  - SPI_RX.sv
  - SPI_tb.sv
  - Output from your self checking test bench proving you ran it successfully