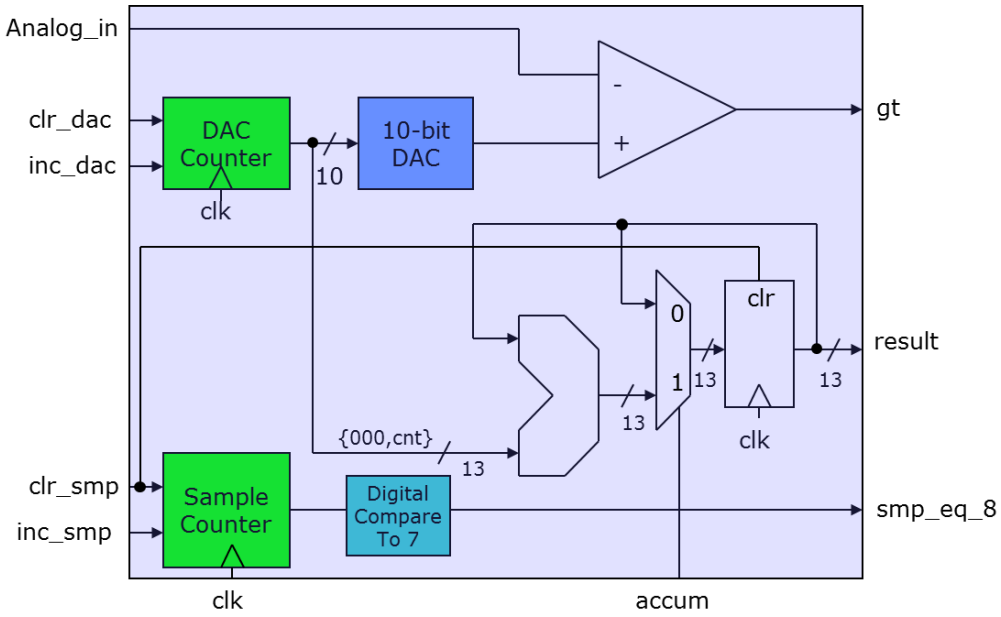
# ECE 551
## HW3

- Due Mon Oct 18th @ 11:55PM
- Work Individually
- Remember What You Learned From the Cummings SNUG paper
- Use descriptive signal names and comment your code

# HW3 Problem 1 (15pts) SM Design

Not Started as Exercise…you are on your own for this one.

A Single Slope A2D converter (capable of averaging 8 samples) was discussed in Lecture1. The block diagram is shown below. On the course webpage under HW3 you will find files: **ss_A2D_tb.v**, **ss_A2D.v**, **ss_A2D_datapath.v**, **ss_A2D_SM_shell.sv**.




Flesh out **ss_A2D_SM_shell.sv** to complete the control (state machine). Simulate your resulting design using the provided self checking test bench. Submit your verilog for the state machine (file should be called **ss_A2D_SM.sv** to the dropbox for HW3). Also submit proof that it worked.

Recall there is a video in week1 (SS_A2D) that discusses this design.

# HW3 Problem 2 (25pts) Signed Divider

Started as Ex11 on Fri Oct 8th

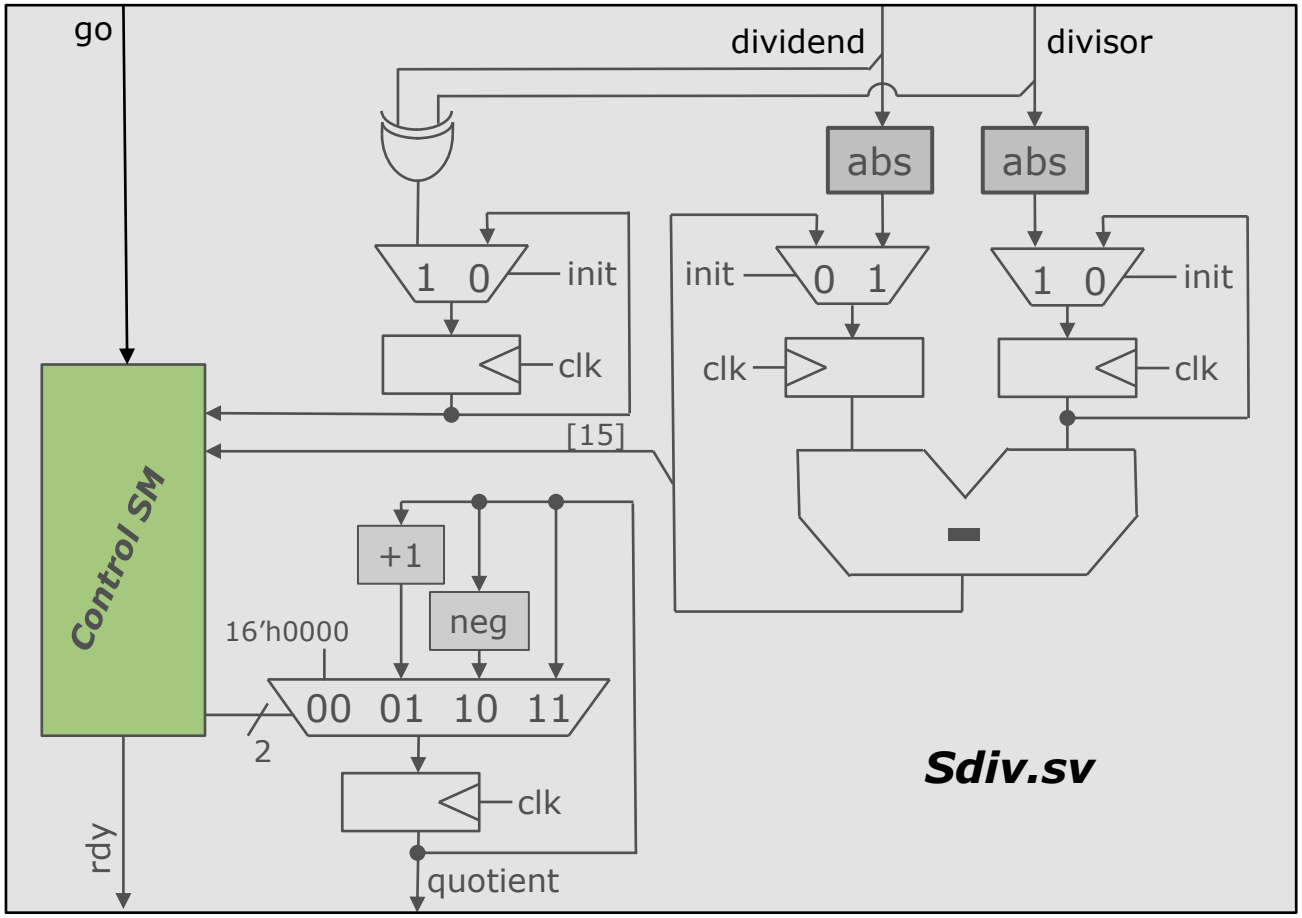
A brute force way to do division is count how many times you can subtract the divisor from the dividend.

A more elegant and efficient way would be to first start with the divisor shifted left N-1 bits and then see if it can be subtracted from the dividend. If so set the MSB of the quotient, shift the divisor right 1-bit and test for the next bit… For simplicity we will, not do this and rather just count how many time the non-scaled divisor can be subtracted from the dividend.

However, I don't want the problem to be too simple, so we will add some complexity by making this a signed divider.

The easiest way to accomplish this is to perform the actual division on the absolute values of the dividend & divisor and then negate the result if either the dividend xor the divisor was negative.

# HW3 Problem 2 (25pts) Signed Divider



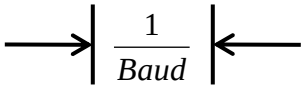

Study this proposed implementation of a signed divider then implement it in verilog. A high level (or one clock cycle pulse) on **go** initiates a divide. When the quotient is correct **rdy** should be asserted. **rdy** should stay high until the next time **go** is asserted (use a SR flop to implement **rdy**)

Create a testbench (***SDiv_tb.sv***) and test it with ++, +-, -+, and -- values of dividend and divisor. Testbench should be self checking.

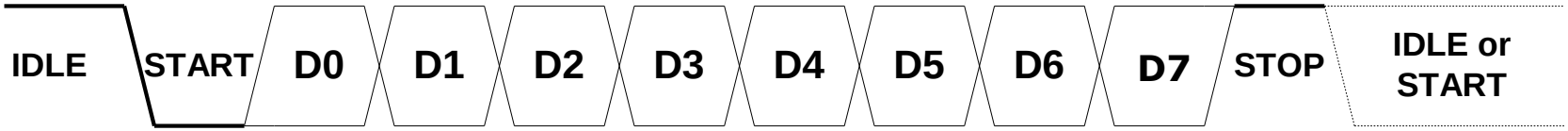
**Submit: SDiv.sv**, **SDiv_tb.sv**, and proof your self-checking testbench ran.

# What is UART (RS-232)

- RS-232 signal phases
  - Idle
  - Start bit
  - Data (8-data for our project)
  - Parity (no parity for our project)
  - Stop bit – channel returns to idle condition
  - Idle or Start next frame

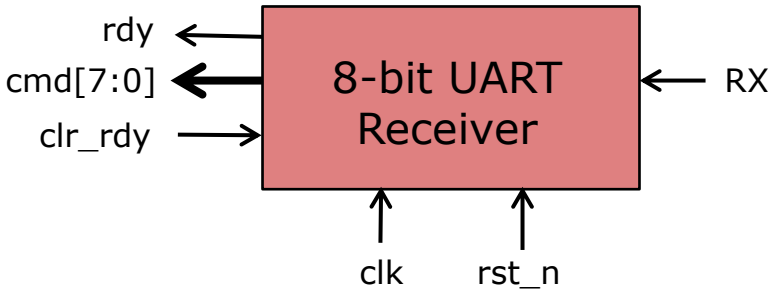



Baud rate will be 19200 with 50MHz clock ➔ 2604 divider ➔ 12-bit



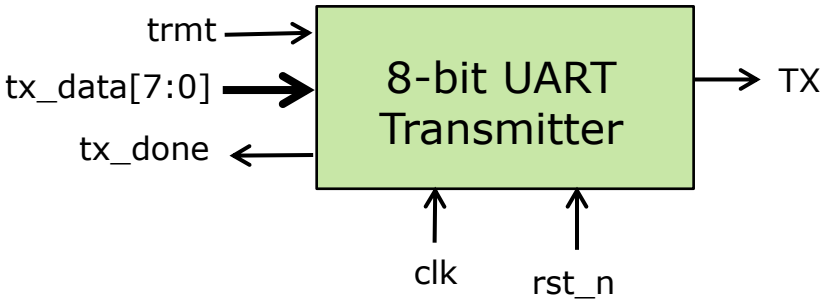

- Receiver monitors for falling edge of Start bit. Counts off 1.5 bit times and starts shifting (right shifting since LSB is first) data into a register.
- Transmitter sits idle till told to transmit. Then will shift out a 9-bit (start bit appended) register at the baud rate interval.

# UART Receiver/Transmitter




A host computer will send commands to the Logic Analyzer via a UART serial peripheral

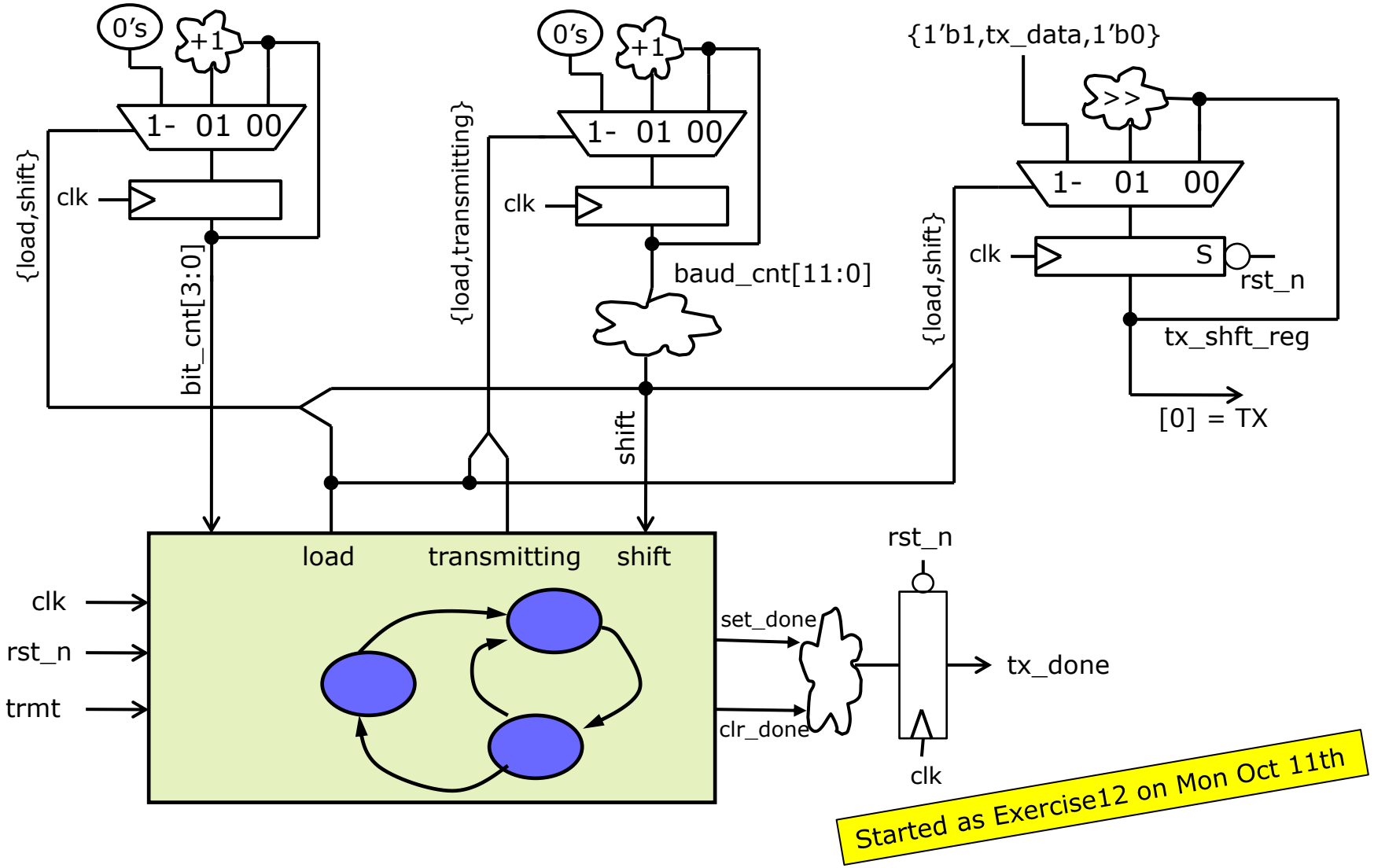
| Signal: | Dir: | Description |
|---|---|---|
| clk,rst_n | in | 50MHz system clock & active low reset |
| RX | in | Serial data carrying command from host computer |
| rdy | out | Asserted when a byte has been received |
| cmd[7:0] | out | Byte received (serves as command to LA) |
| clr_rdy | in | Asserted to knock down the **rdy** signal. |




The follower sends responses back to the host computer. These responses are sent via a UART serial peripheral.

| Signal: | Dir: | Description |
|---|---|---|
| clk,rst_n | in | 50MHz system clock & active low reset |
| TX | out | Serial data output back to host |
| trmt | in | Asserted for 1 clock to initiate transmission |
| tx_data[7:0] | in | Byte to transmit (response from LA) |
| tx_done | out | Asserted when byte is done transmitting. Stays high till next byte transmitted. |

# Possible Topology of UART_tx



# HW3 Problem 3 (**20pts)** UART Transmitter

Started as Exercise12 on Mon Oct 11th

Implement a the UART Transmitter (**UART_tx.sv**).

Make a simple test bench for it. This is one instance in which I would not spend too much time on the test bench. You can just instantiate your transmitter and send a few bytes. Verify the correct functionality (including baud rate) by staring at the green waveforms. You will make a more comprehensive test bench in the next problem.

Submit **UART_tx.sv** to the dropbox for HW3.

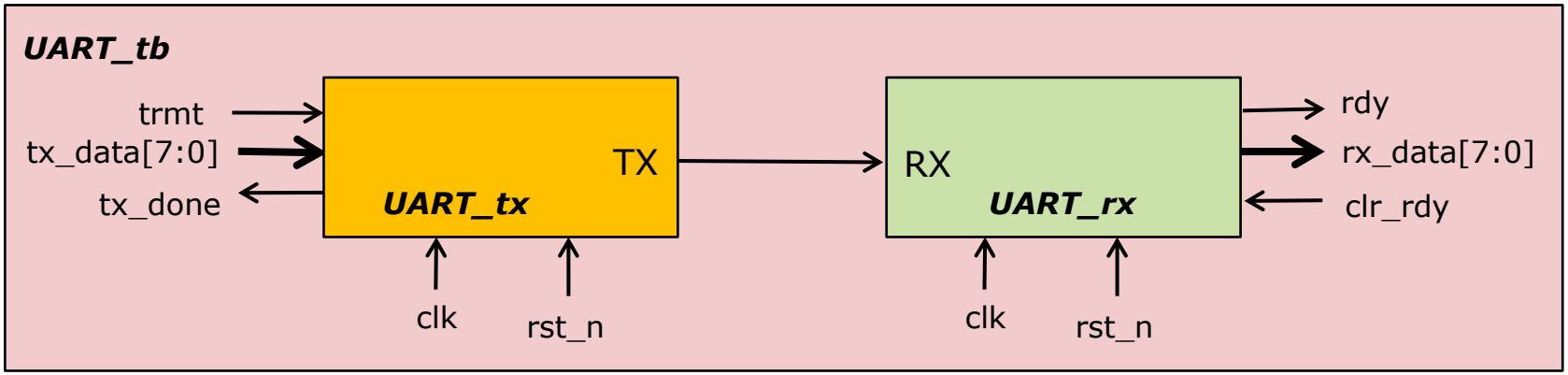
# HW3 Problem 4 (**20pts + 10pts)** UART Receiver

Started as Exercise13 on Weds Oct 13th

Implement a the UART Receiver (**UART_rx.sv**).

Since you have a transmitter too, it is now easy to make a self checking test bench. Architect the test bench as shown. Is does the 8-bit value you transmit match the value you receive when the transmission completes.

Submit **UART_rx.sv** and your self-checking test bench (and **proof** it ran) to the dropbox for HW3.

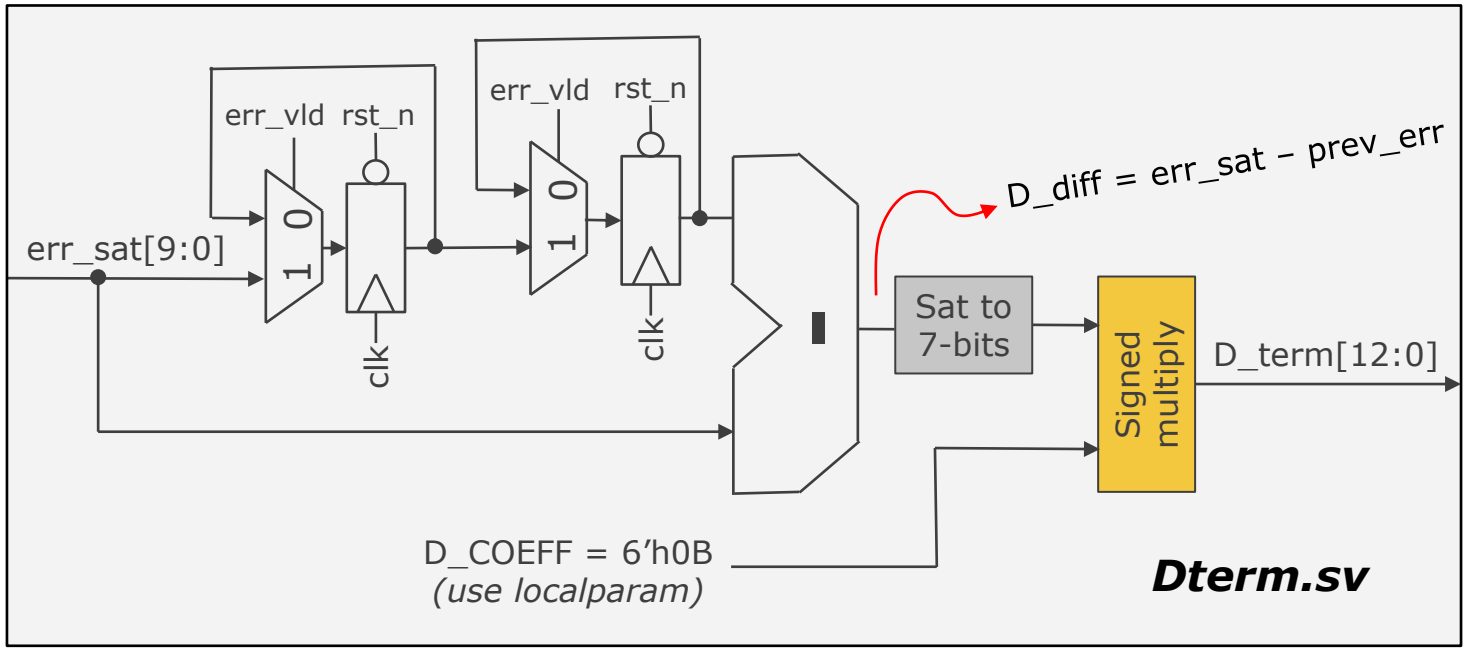


# HW3 Problem 5 (**10pts)** D_term of PID

Started as Exercise10 on Wed Oct 6th

The derivative of a function can be approximated by: $\frac{dF(t)}{dt} = \frac{f(t)-f(t-\Delta t)}{\Delta t}$

We only need something proportional to the derivative so the divide by $\Delta t$ is unnecessary.
Consider the following circuit:




**Submit: Dterm.sv**, **Dterm_tb.sv**, and proof it ran.