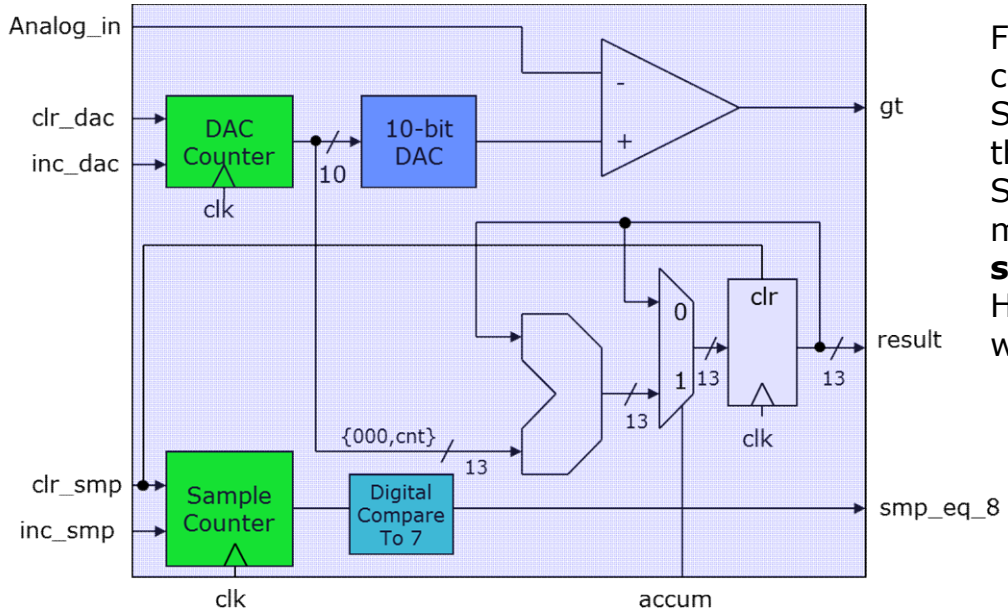
ECE 551

HW3 Solution

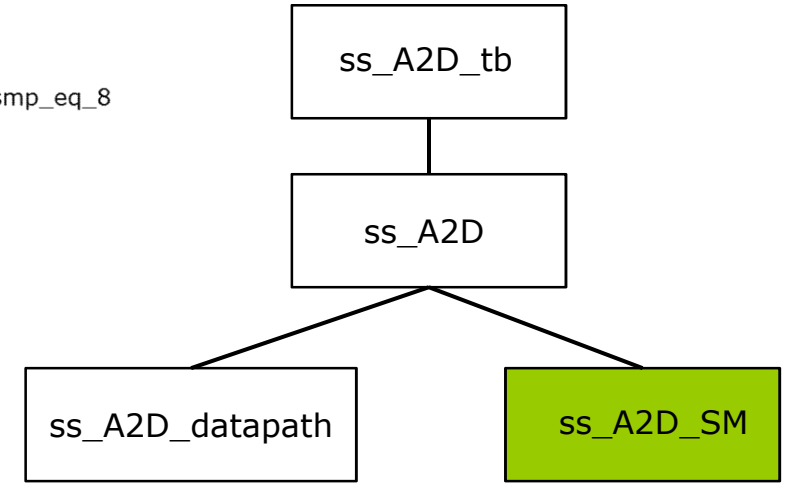
HW3 Problem 1 (**20pts)** SM Design

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**.



Flush 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.

Hierarchy of design is as shown. Testbench instantiates DUT (ss_A2D). Which in turn instantiates datapath and statemachine. You are simply flushing out the SM code and renaming the file from ss_A2D_SM_shell.sv to ss_A2D_SM.sv. You are also simulating to prove it works.



```
module ss_A2D_SM(clk,rst_n,strt_cnv,smp_eq_8,gt,clr_dac,inc_dac,
                 clr_smp,inc_smp,accum,cnv_cmplt);

  input clk,rst_n;          // clock and asynch reset
  input strt_cnv;           // asserted to kick off a conversion
  input smp_eq_8;           // from datapath, tells when we have 8 samples
  input gt;                 // gt signal, has to be double flopped

  output clr_dac;           // clear the input counter to the DAC
  output inc_dac;           // increment the counter to the DAC
  output clr_smp;           // clear the sample counter
  output inc_smp;           // increment the sample counter
  output accum;             // asserted to make accumulator accumulate sample
  output cnv_cmplt;         // indicates when the conversion is complete


  typedef enum reg [1:0] {IDLE,CONV,ACCUM} state_t;

  state_t state,nstate;          // both state and nstate of the enum state type
  reg gt_ff1, gt_ff2;            // flops for gt metastability purposes

  //// All signals that are outputs of SM are of type logic ////
  logic clr_dac,inc_dac,clr_smp,inc_smp,accum,cnv_cmplt;

  //// Infer state flops next ////
  always_ff @(posedge clk, negedge rst_n)
    if (!rst_n)
      state <= IDLE;
    else
      state <= nstate;
```

See next Slide for rest

```
//// flop gt twice for meta-stability purposes ////
always @(posedge clk)
  if (clr_dac) begin
    gt_ff1 <= 1'b0;
    gt_ff2 <= 1'b0;
  end else begin
    gt_ff1 <= gt;
    gt_ff2 <= gt_ff1;
  end

always_comb begin
  //// default all outputs ////
  clr_dac = 0;
  inc_dac = 0;
  clr_smp = 0;
  inc_smp = 0;
  accum = 0;
  cnv_cmplt = 0;
  nstate = IDLE;

  case (state)
    IDLE : if (strt_cnv) begin
      clr_dac = 1;
      clr_smp = 1;
      nstate = CONV;
    end
    CONV : if (!gt_ff2) begin
      inc_dac = 1;
      nstate = CONV;
    end else begin
      accum = 1;
      nstate = ACCUM;
    end
    default : if (smp_eq_8) begin // this is the accum state
      cnv_cmplt = 1;
      nstate = IDLE;
    end else begin
      clr_dac = 1;
      inc_smp = 1;
      nstate = CONV;
    end
  endcase
end
```

HW3 Problem 2 (**25pts**) ESC Interface

In the vicinity of slide 38 of the Project Spec you will find a section on ESC (Electronic Speed Control). Read these slides carefully.

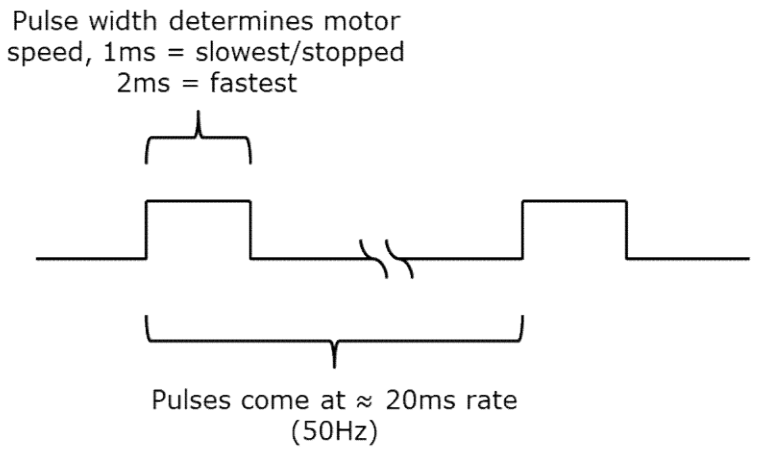
You will be implementing the ESC controller, with the following interface:

Signal:	Dir:	Description:
clk	in	50MHz clock
rst_n	in	Active low asynch res…
SPEED[10:0]	in	Result from …
OFF[9:0]	in	…iation in ESC/motor pairs. Give all …n close to the same speed. This offset is
PWM		…otor speed. It is effectively a PWM signal.

Every one demoed a working version of this on their DE0-Nano boards, so perhaps your code is not pretty, but it works and synthesizes...no solution provided

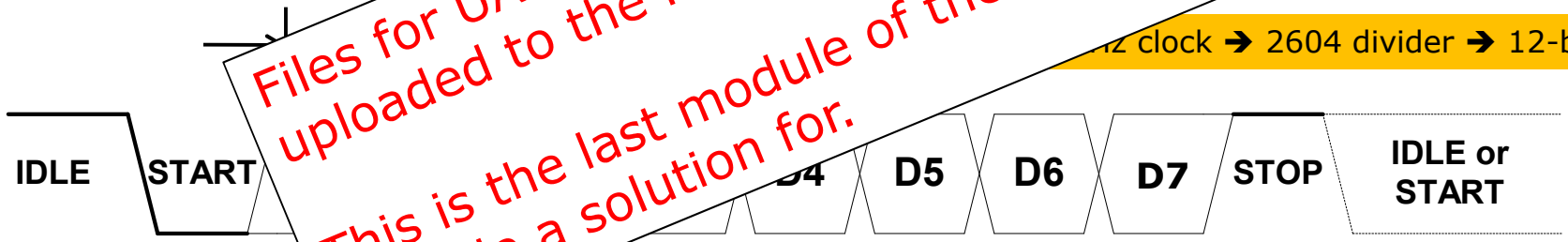
- If both SPEED … the PWM width …s.
- For every count of SPEED or OFF the speed would increase by 0.32usec

This problem will have been completed as part of in class exercises. Just submit your ESC_interface.sv that was checked off in class.



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 retur
  - Idle or Start nex

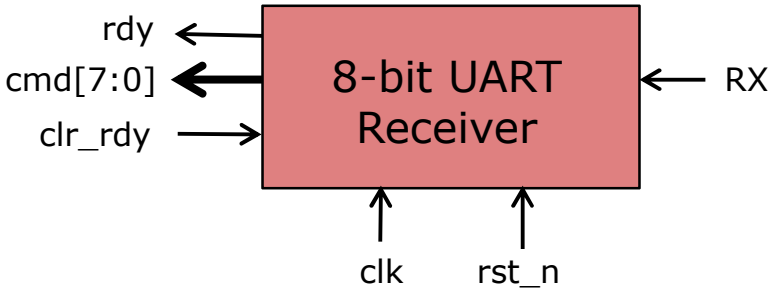


Files for UART Transmitter and Receiver are uploaded to the HW3 Solution file on Moodle.

This is the last module of the project I will provide a solution for.

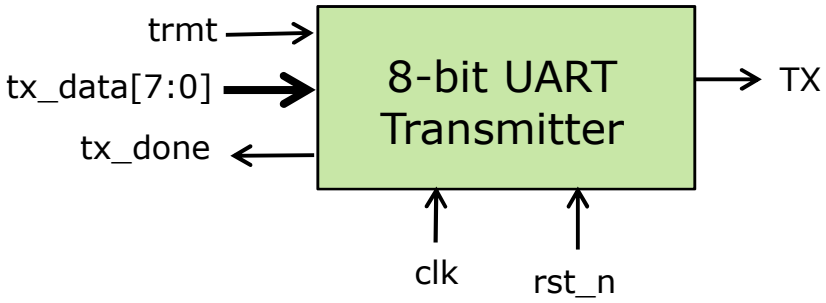
- Receiver m itors for falling edge of Start bit. Counts off 1.5 bit times and starts shifting (right shifting since LSB is first) data into a register.
- Transmitter sits idle till told to transmit. Then will shift out a 9-bit (start bit appended) register at the baud rate interval.

UART Receiver/Transmitter



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

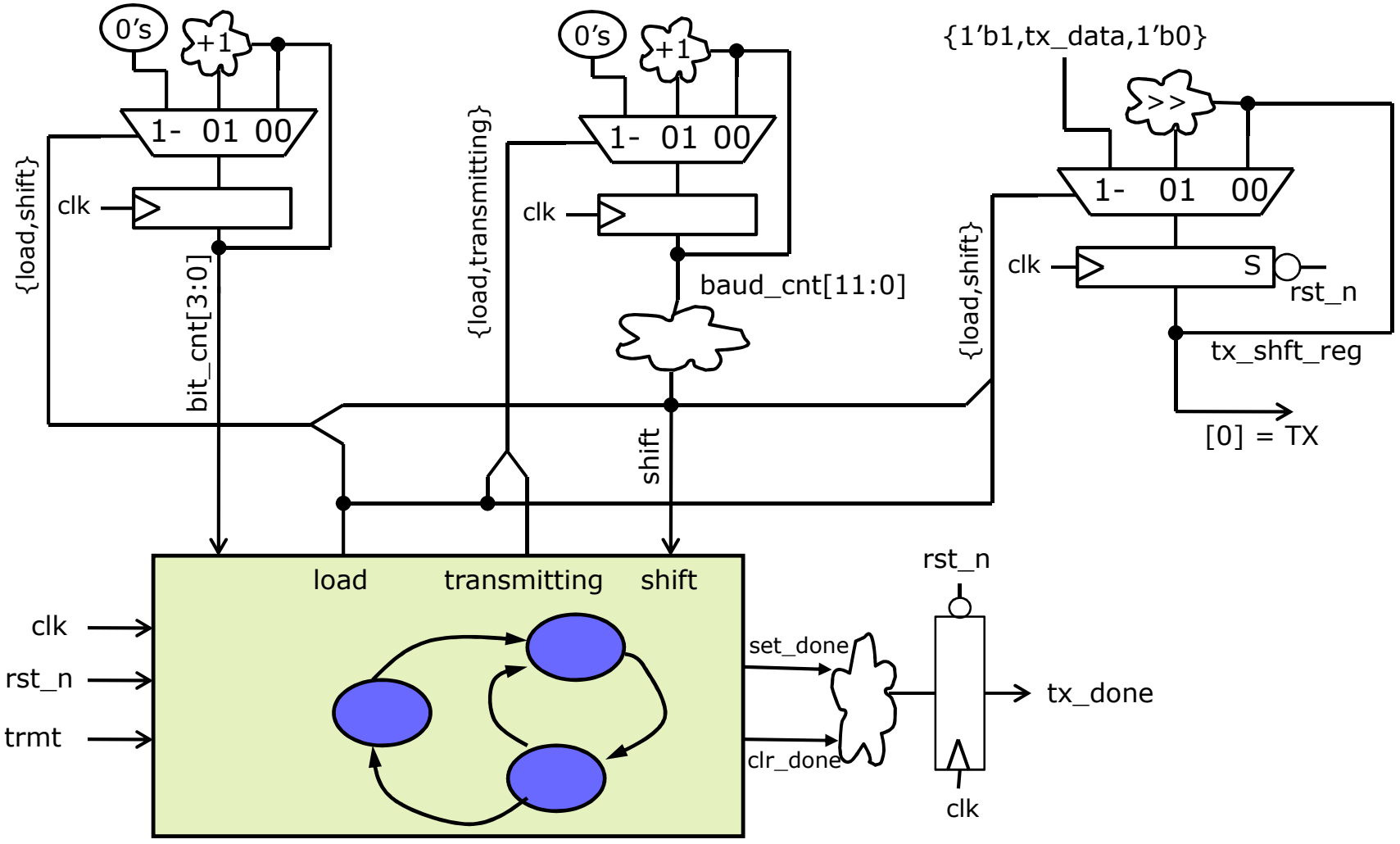
Signal:	Dir:	Description
clk,rst_n	in	100MHz system clock & active low reset
RX	in	Serial data carrying command from host computer
rdy	out	Asserted when a byte has been received. Falls when new start bit comes, or when ***clr_rdy*** knocks it down.
cmd[7:0]	out	Byte received (serves as command to LA)
clr_rdy	in	Asserted to knock down the **rdy** signal.



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

Signal:	Dir:	Description
clk,rst_n	in	100MHz 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

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 (**25pts + 10pts)** UART Receiver

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

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

Submit **UART_rcv.sv** and your test bench to the dropbox for HW3.