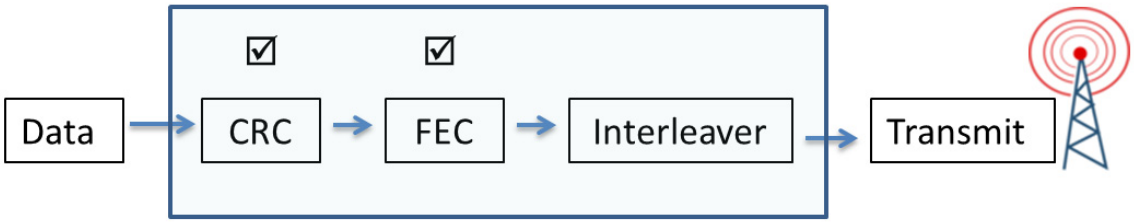
M A S S A C H U S E T T S I N S T I T U T E O F T E C H N O L O G Y
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.111 Introductory Digital Systems Laboratory
Fall 2018

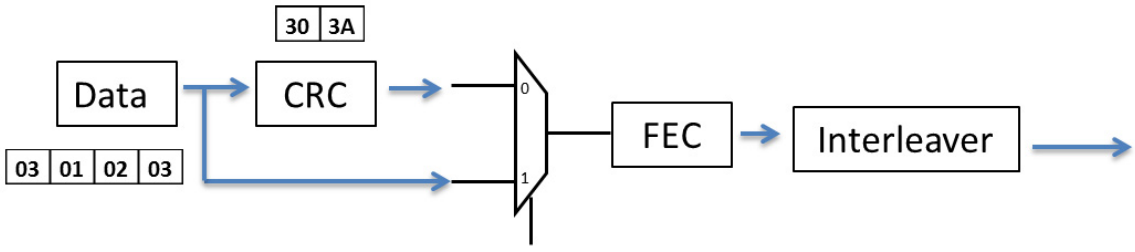
Lecture PSet #8 Last one!
Due: Tues 10/16/2018 14:30
Upload solution as one pdf

Problem 1 (6 point) This is the last of four parts in implementing a communications system. For this Lpset, we will optimize the process to calculate the CRC and generate the parity bits for the (FEC) convolution encoder.



0x03_01_02_03 + CRC: 30_3A

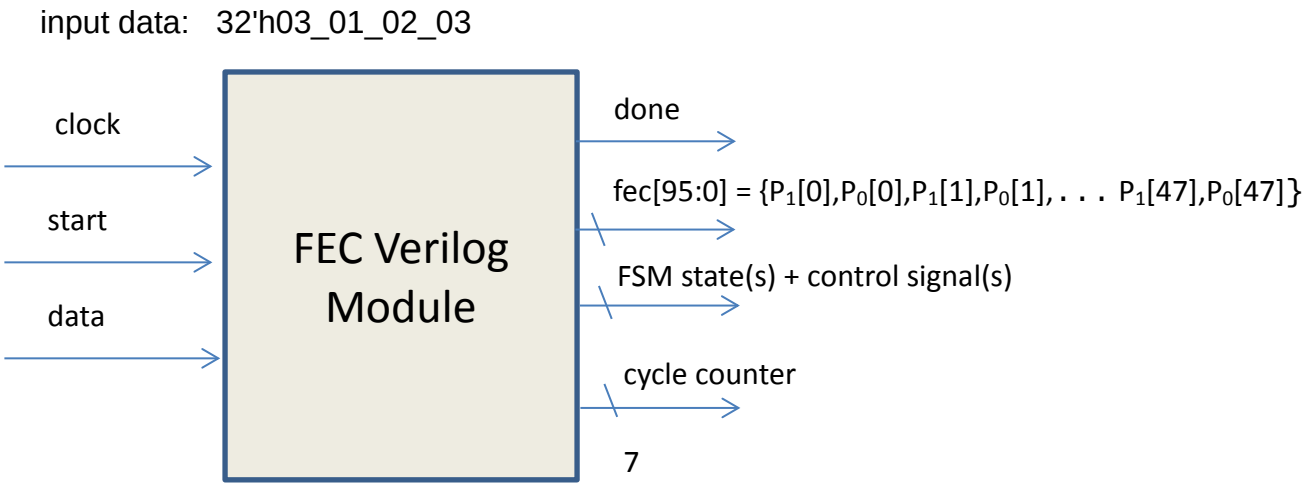
The complete process, in serial mode, requires **32 clock cycle** to calculate the CRC followed by **48 clock cycles** for the convolution encoder totaling **80 clock cycles**. Note the CRC result is not required while the convolution encoder is processing the first four bytes of data. Consider this faster approach: as CRC is being calculated, the convolution encoder at the same time is generating parity bits for the first four bytes of data. After the last data bit has been shifted in, CRC has been computed and is available. At this time the input to the convolution encoder can be switched by control to the output of the CRC registers.



Generating CRC for the four bytes takes 32 clock cycles. Convolution encoding the first four bytes of data also takes 32 clock cycles. Another 16 clock cycle is required to complete the encoding for the CRC for a total of 40 clock cycles. By taking advantage of concurrent processing total processing is reduced from 80 clock cycles to 48 clock cycles.

[Total time can even be reduced further. At t=32, since all 16 bits of CRC are available, the remaining 32 parity bits can be computed in parallel in one clock cycle! (not required)]

Your task will be to implement in Verilog the process described above (without the interleaver). Using your CRC Verilog from Lpset 6 and FEC Verilog from Lpset 7, modified your design to take advantage of concurrency. ***Be sure to include the Verilog for CRC and FEC in the report.***



Here is how to get started.
Step 1: Using ISE, create a new Verilog module with inputs and outputs as shown above.
Step 2: The Verilog module: when **start** is asserted, reset your FSM; reset counters and other registers; and load any initial values required. With each clock pulse, shift in one bit of data, begin calculating CRC **and** generating parity bits using rate ½ constraint length 4 code (k=4) using these generators g_0 = 1,1,1,1 and g_1 = 1,1,0,1. [Same as Lpset 6 & Lpset 7]
Step 3: After 32 or 33 clock cycles (implementation dependent), use the output of the CRC register to complete the generation of the remaining parity bits. You may use any method to computer the remaining parity bits. After all parity bits have been calculated, assert **done**.
Step 4: Using ISE, create a behavior test bench (Verilog Test Fixture) and verify your design with a simulation using the process outlined in Lpset 6. You can use a 5ns clock in your test bench. The input data should be 32'h03_01_02_03
Step 5: When **done** is asserted your encoding (using hex radix) should be

fec[95:0] = 96'h000E_8C03_7C0D_F00E_828C_0E5E

Step 6: Take a screen shot of your system at **$30 \leq t \leq 34$** showing the control signal(s) and FSM state(s) as the input to the convolution encoder is switched from data input to the output of the CRC.
Step 7: Take a screen shot showing **fec[95:0]** when **done** is asserted. Use hex radix for **fec[95:0].** Include the Verilog (Verilog module and test bench) and screen shot in one pdf file and upload to the course website.

Lpset grading rubric

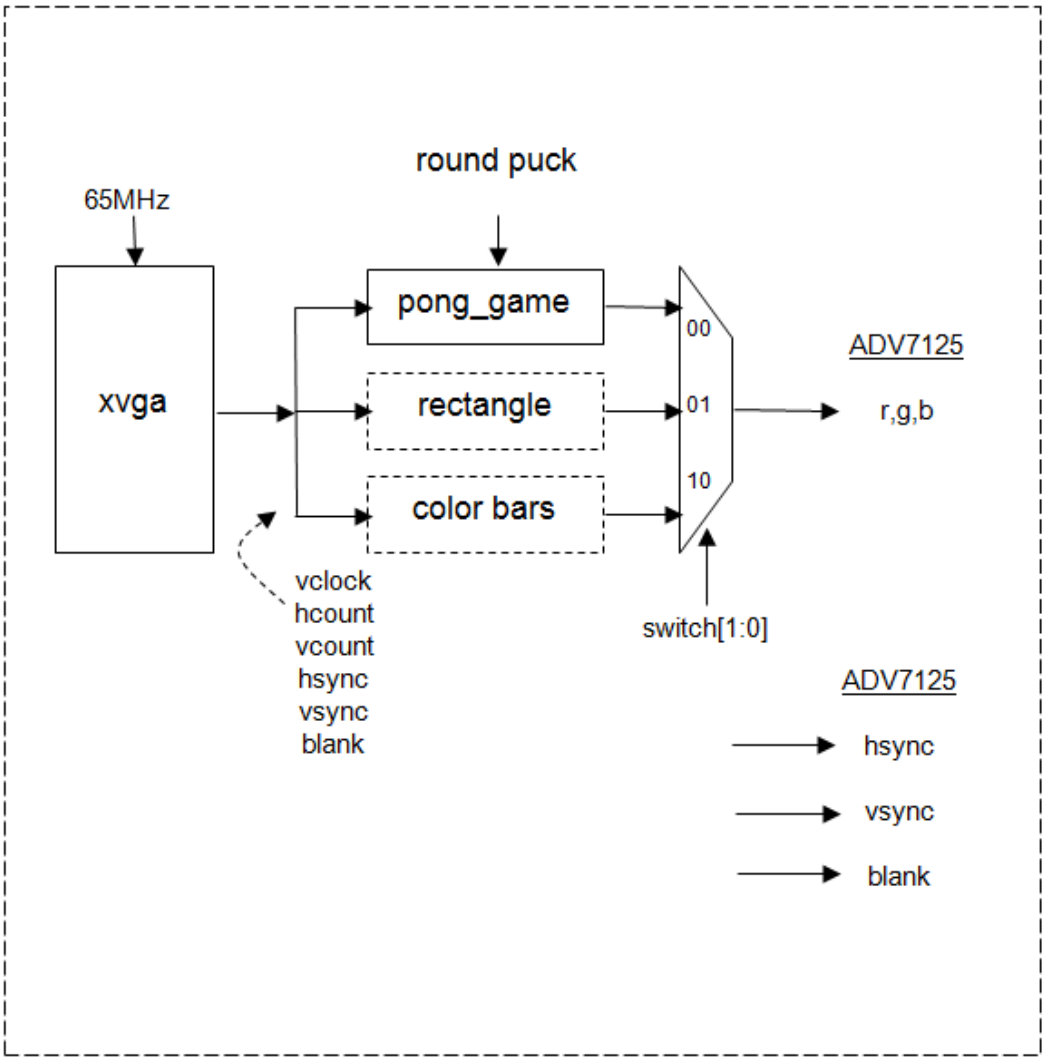
Grading	
1	Easy to read & formatted Verilog (See "Verilog Editors" tab for help.) Include Verilog for CRC and FEC in the PDF.
2	Verilog meeting all the specs
1	Functional test bench
1	Screenshot at **$30 \le t \le 34$** showing control signals & FSM states
1	Screenshot showing fec[95:0] when done is asserted
6	**Total Grade**

Problem 2 (4 points total) In the pong lab, pixel, vsync, hsync, and blank are signals set to the video DAC which generates the VGA signals. The system clock is 65mhz with a 15ns period. Assume the combinatorial logic generating the rectangular blob has a 9ns t_{pd}. Generating a round ball requires additional logic using a multiply with an 8ns t_{pd}. As a result this additional block added to the other logic does not meet the timing requirements (9ns + 8ns > 15ns clock). [Solution to this problem will be used in Lab 5.]

```
always @ *  begin   // generate round puck
     // compute x-xcenter and y-ycenter
     radiussquared = RADIUS*RADIUS;  // RADIUS is a paramater
    deltax = (hcount > (x+RADIUS)) ? (hcount-(x+RADIUS)) : ((x+RADIUS)-hcount);
     deltay = (vcount > (y+RADIUS)) ? (vcount-(y+RADIUS)) : ((y+RADIUS)-vcount);
     // check if distance is less than radius squared
     if(deltax*deltax+deltay*deltay <= radiussquared)
           pixel = COLOR;
     else pixel = 0;
   end
```

(A) Pipeline the above Verilog noting that the multiplies have the longest t_{pd}. (Logic with the longest tpd should be pipelined separately.) Use correct blocking/non-blocking assignments. Assume hcount, vcount, x and y are already registered glitch free inputs clocked by the system clock: clock_65mhz. **[2 point]**

(B) Indicate any additional register(s) required in this block diagram for correct VGA timing. The round puck Verilog is part of the pong game block. Indicate the number of registers per pipelined signal. **[2 point]** Note – you can copy and paste the image below for your solution.



Upload solutions to Problems 1 and 2 as one pdf file.