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 2019

Lecture PSet #3 of 8

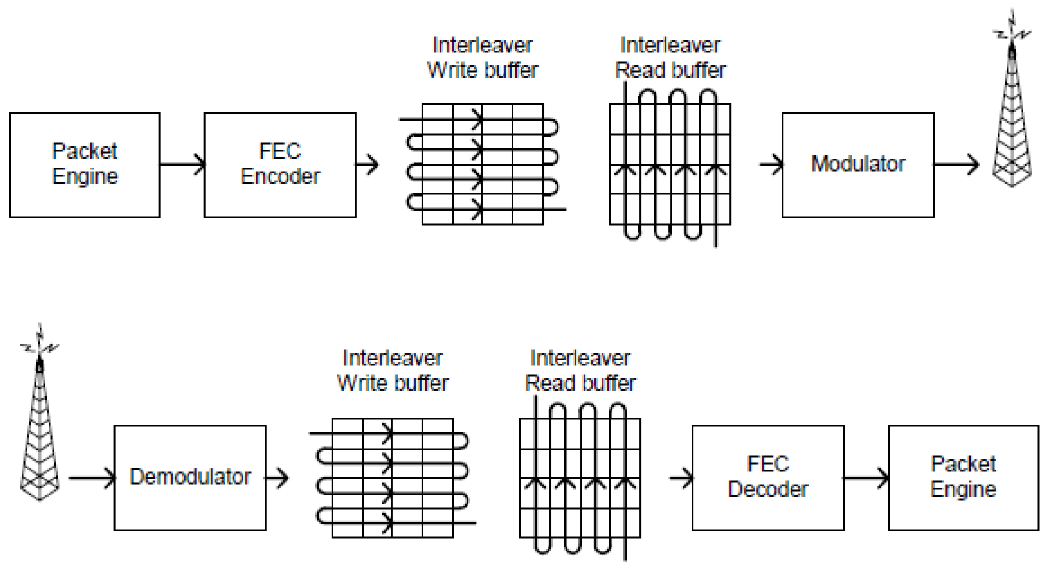
Due: Thu, 14:30 09/19/2019

Note: Submit PDF online

Optional docx available for submission

Problem 1. **Interleaving** [This problem is based on the implementation of a digital communications research project at MIT. When facing a tough problem, coming up with a solution is easier when the problem is broken down to simpler smaller blocks. The solution to this part together with parts 2 and 3 in future lpsets will form the complete solution.]

For many communications systems, a forward error correcting (FEC) code such as a convolution code, is used during transmission. This will allow the receiver to correct erroneous bits when errors occur randomly in a coded sequence. [More on FEC in future lpsets.] However, the bursty nature of noise will often wipe out large number of adjacent data bits - defeating the convolution code. A simple solution is to interleave the data bits of a four byte packet so that adjacent data bits are spaced out in the transmitted sequence. Instead of sending all 8 bits of the byte 0, the low order bit pair of bytes 3, 2, 1, and 0 (starting at the LSB end) are transmitted followed by the next set of bit pairs until all bits are transmitted. This is implemented in many satellite communication systems[1].



(see other side)

[1] from http://www.ti.com/lit/an/swra113a/swra113a.pdf

(A) Implement a Verilog module that will interleave 4 bytes as described above.

```
module interveaver(
    input [7:0] byte0,  // 00
    input [7:0] byte1,  // 0E
    input [7:0] byte2,  // 8C
    input [7:0] byte3,  // 03
    output [7:0] out0,
    output [7:0] out1,
    output [7:0] out2,
    output [7:0] out3
    );

    assign out0 =
    assign out1 =
    assign out2 =
    assign out3 =

endmodule
```

There are multiple implementations. To receive credit your interleaver must encode this input [**00 0E 8C 03]** to the following output [**C8 3C 00 20].** This will ensure compatibility with the deinterleaver. [This Verilog was used in a research project.]

(B) Write the Verilog for a deinterleaver. Any interesting observation?

Problem 2 TMDS

In HDMI video, the pixel and audio data are transmitted down wires at extremely high data rates. Depending on the resolution and version of HDMI, up to 18 Gbits per second of data may need to get transmitted using three separate channels (we'll talk about this in Lecture 5). That's a lot of 1's and 0's to be transmitted. Unfortunately sending a lot of 1's and 0's means transitioning the voltage on the line at ~GHz rates, which means the wires can be giving off tons of RF noise which can be extremely detrimental to the whole process (red pixel info interferes blue pixel info).

In order to at least partially mitigate high frequency value transitions, HDMI encodes its data using a scheme known as **T**ransmission **M**inimized **D**ifferential **S**ignaling (TMDS). In this scheme, the number of 1→0 or 0→1 transitions in a portion of data is reduced at the expense of sending slightly more bits overall. Specifically, TMDS will take every 8bits of data and transform it into 10 bits for sending. This problem concerns the first of those extra bits (the tenth bit we'll address in a future homework) *So to clarify, for this problem we'll be creating 9 bits to represent 8 bits*:

The strategy of TMDS is to take an input byte X with bits $\{x_7\ x_6\ x_5\ x_4\ x_3\ x_2\ x_1\ x_0\}$ and generate a processed output byte Y with bits $\{y_7\ y_6\ y_5\ y_4\ y_3\ y_2\ y_1\ y_0\}$ from one of two options:

1. Option One:
    a. The original lsb is assigned to the lsb of the new data frame: $y_0 = x_0$
    b. The remaining 7 output bits are the XOR of the preceding two input bits as expressed: $y_n = x_n \oplus x_{n-1}$ for $7 \geq n \geq 1$ where n is the bit number
2. Option Two:
    a. The original lsb is assigned to the lsb of the new data frame: $y_0 = x_0$
    b. The remaining 7 output bits are the XNOR of the preceding two input bits as expressed $y_n = \overline{(x_n \oplus x_{n-1})}$ for $7 \geq n \geq 1$ where n is the bit number
3. Y becomes the option with fewer internal transitions. If Option 1 is chosen, append a 1 as the ninth bit else, append a 0 as the ninth bit.
4. Send all 9 bits.

On the receiving side, the system receives packet Z with bits $\{z_8\ z_7\ z_6\ z_5\ z_4\ z_3\ z_2\ z_1\ z_0\}$ and builds up a decoded byte W with bits $\{w_7\ w_6\ w_5\ w_4\ w_3\ w_2\ w_1\ w_0\}$ using the following process:

1. Use the ninth bit to determine if the data was XOR or XNOR processed
1. If XOR:
    a. The received lsb is assigned to the lsb of the new data frame: $w_0 = z_0$
    b. The remaining 7 output bits are the XOR of the preceding two input bits as expressed: $w_n = z_n \oplus w_{n-1}$ for $7 \geq n \geq 1$ where n is the bit number
2. If XNOR:
    a. The received lsb is assigned to the lsb of the new data frame: $w_0 = z_0$
    b. The remaining 7 output bits are the XNOR of the preceding two input bits as expressed: $w_n = \overline{(z_n \oplus w_{n-1})}$ for $7 \geq n \geq 1$ where n is the bit number

Carry out the following calculations (*note you should be able to "check" your math by making sure you can decode what you create using the scheme provided up above)*:

Part A) Our input data byte is **8'b1010_0101** :

i) Within the data byte how many 0→ 1 or 1→0 transitions are there? :

ii) What would processing the data byte with Option 1 look like:

iii) What would processing the data byte with Option 2 look like:

iv) Which one has fewer transitions? If Option 1, add a 1 as a ninth bit, else add a 0 as the ninth bit. What are the nine bits sent?:

(see other side)

Part B) Our input data byte is **8'b1111_1111:**

i) Within the data byte how many 0→ 1 or 1→0 transitions are there?:

ii) What would processing the data byte with Option 1 look like:

iii) What would processing the data byte with Option 2 look like:

iv) Which one has fewer transitions? If Option 1, add a 1 as a ninth bit, else add a 0 as the ninth bit. What are the nine bits sent?:

Part 2) We won't build the entire TMDS system today, but one module that is needed to get it working is a "one-tallier" that takes in a byte and returns the number of 1's present in it. For example:

- Input Byte: 8'b1011_1100: has a one tally of five
- Input Byte: 8'b0000_1100: has a one tally of two

Build a module in SystemVerilog that takes in one input (the byte being analyzed) and produces an output that indicates the one tally. The module should be purely combinational since we need it to be fast and low-latency. *(Do not overthink this. This should be a pretty simple answer. We'll worry about what needs to happen because it is simple later)*

```
module tallier(
    input [7:0]        byte_in,
    output logic [2:0] tally_out
    );

// Your Verilog

endmodule
```