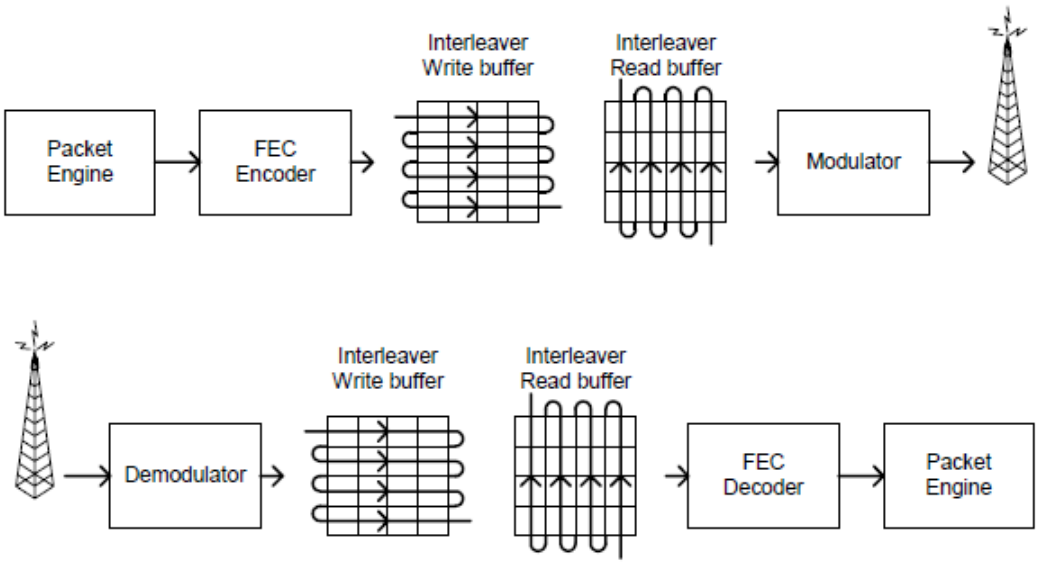
M A S S A C H U S E T T S I N S T I T U T E O F T E C H N O L O G Y
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.111 Introductory Digital Systems Laboratory
Fall 2018

Lecture PSet #3 of 8
Due: Tue, 14:30 09/18/2018
Note: Submit PDF online

Problem 1. [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 Lpset 7 and 8 forms 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, // data = 8'h00
    input [7:0] byte1, // data = 8'h0E
    input [7:0] byte2, // data = 8'h8C
    input [7:0] byte3, // data = 8'h0C
    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 [For full credit, Verilog must be syntactically correct.]

For each of the parts below write one or more statements of Verilog that implement the desired functionality. Your Verilog just has to produce the same values for its outputs – it doesn't have to replicate the schematic logic gate-for-gate (in fact, you should not use those "structural" constructs). Be sure to include the appropriate declarations for any wires or regs used in your code.

(A) A circuit that divides an 8-bit input operand by 16 and produces a 4-bit value.
[Division with with numbers other than powers of two requirement special circuits.]

(B) A circuit to compute the 17-bit sum of two 16-bit operands.

(C) Implement a 2-bit priority encoder (a circuit which examines its four inputs (I0, I1, I2, I3) and outputs a 2-bit binary number indicating the highest-priority input which has a value of "1", where I3 has the highest priority and I0 the lowest) with "dataflow" (assign) constructs.

(D) Implement a 2-bit priority encoder with "sequential" (always) construct and "case" statement.

Problem 3. [For full credit, Verilog must be syntactically correct.]

Using the Verilog parameterized module mechanism it's possible to write modules whose operations depends on parameters specified when the module is instantiated.

(A) Write a parameterized parity check module which takes as input a bus whose size is set by a parameter. The module has a single output which is 0 if the number of 1's in the input vector is even and 1 otherwise. [This is referred to as even parity.]

(B) How would one instantiate an instance of your module to compute parity check on the 16-bit data bus DATA[15:0]?