Problem Set 7

Code:

```
module convolutional_encoder(
                                                        clk_in,
                                                        start_in,
                                                        data_in,
                                                        done_out,
                                                        fec_out
                                                );

        // This is the input clk, clk_in
        input                   clk_in;
        // This is the input start signal, start_in
        input                   start_in;
        // This is the input data signal, data_in
        // it is cereal.
        input                   data_in;
        // This is the done output signal
        output                  done_out;
        // This is the 96 bit fec output signal
        // given 48 bits input this block produces 2 outputs per input
        // 48 * 2 is 96
        output  [95:0] fec_out;

        // This is the local parameter window size
        localparam WIN_SIZE = 4;
        // This is the count value which we start decrementing from
        // After 48 input cycles we are done
        localparam DONE_COUNT = 48;

        // This is the input data shift register. It is size 4 because
        // we need x[n-3]
        reg [WIN_SIZE-1:0] data_in_sr = 'b0;
        // This is the start state
        reg                     started   = 1'b0;
        // This is the 96 bit fec output signal
        // given 48 bits input this block produces 2 outputs per input
        // 48 * 2 is 96
        reg             [95:0] fec_out   = 96'b0;
        // This is the counter to determine the number of bits in the shift
// register
reg         [5:0]  counter    = DONE_COUNT;

// This is a wire of P_0[n]
wire p0;
// This is a wire for P_1[n]
wire p1;

// This assigned the P_0[n] value to the value specified in the Pset
assign p0       = data_in ^ data_in_sr[0] ^ data_in_sr[1] ^ data_in_sr[2];
// This assigned the P_1[n] value to the value specified in the Pset
assign p1       = data_in ^ data_in_sr[1] ^ data_in_sr[2];
// We are done once the count hits 0
assign done_out = ~(|counter);

//This is an always block so stuff is synchronous
always @(posedge clk_in)
// This is the begin statement to look nice
begin
        // This is an if statement which tells us to start if we
        // get told to start but haven't started yet
        if ((~started) & start_in)
        // This is a begin statement to look nice
        begin
                // This is the beginning value of the counter
                counter    <= DONE_COUNT;
                // This initialised the fec_out to 0
                fec_out    <= 96'b0;
                // This tells us later that we have started
                started    <= 1'b1;
                // This is the shift in the shift register
                data_in_sr <= {data_in_sr[3:0], data_in};
                // This is an end statement for the if statement begin
        end
        // This is te check if we have started but aren't done yet
        else if (started & (~done_out))
        // This is the begin statement to look nice
        begin
                // This is the decrement of the counter
                counter    <= counter - 1;
```

```
                    // This is the filling up of the fec register
                    // We fill from the LSB
                    fec_out    <= {fec_out[93:0], p1, p0};
                    // This shifts the data into the shift register
                    // We fill from the LSB
                    data_in_sr <= {data_in_sr[3:0], data_in};
                    // This is an end statement from the else if
                end
                // This is the end statement for the always block
        end
// This is an endmodule for the convolutional_encoder module
endmodule
```

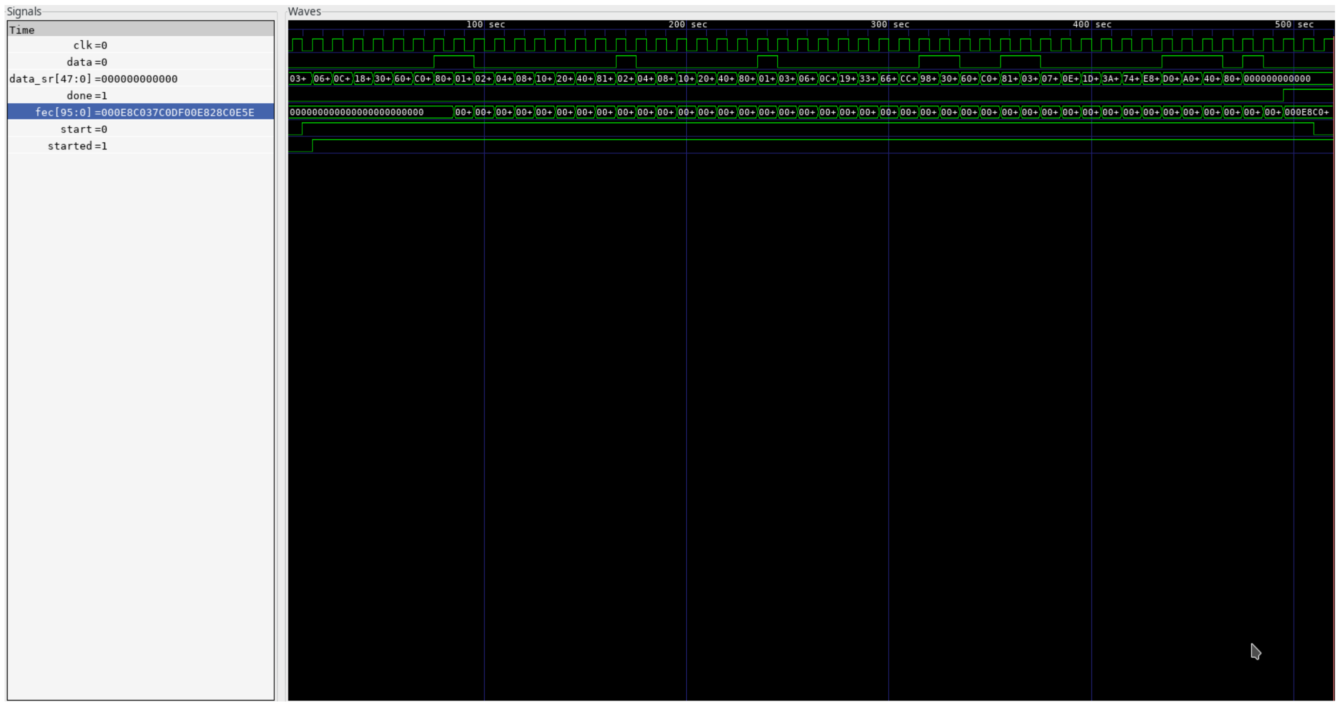
Testbench:

```
module convolutional_encoder_tb;
        // This is the count value which we start decrementing from
        // After 48 input cycles we are done
        localparam DONE_COUNT = 48;
        // This is the clock
        reg         clk   = 1'b0;
        // This is the start signal
        reg         start = 1'b0;
        // This is the data signal
        reg         data  = 1'b0;
        // This is the done output signal
        wire        done;
        // This is the output of the FEC encoder
        wire [95:0] fec;
        // This is the instantiation of the FEC encoder
        convolutional_encoder ce1(
                                                        .clk_in    (clk),
                                                        .start_in  (start),
                                                        .data_in   (data),
                                                        .done_out  (done),
                                                        .fec_out   (fec)
                                                        );
        // This is the input test data
        reg [DONE_COUNT-1:0] data_sr = 48'h03010203303A;
        // This ensures that we haven't started yet, so do not
        // shift data in
        reg                 started = 1'b0;

        // alonzi
        initial
        begin
                // This is so we can plot stuff in vcd viewer
                $dumpfile("test.vcd");
                // This is so we can see all the signals
                $dumpvars(0, convolutional_encoder_tb);
                // This tells the dut to start
                start   = 0;
                #10;
                start   = 1;
                // This is some delay so we get some output
                #500;
                // This is to deassert and make sure the
                // output stays fixed
                start   = 0;
                #10;
                // bye bye
                $finish;
        end

        // This is the always block where we shift in the cereal data
        always @(posedge clk)
        begin
                // if we start or have started
                if ( (~done) & (start | started))
                begin
                        // This is so we know next clock tick that we have started
                        started <= 1'b1;
                        // This puts the cereal into out encoder
                        data    <= data_sr[DONE_COUNT-1];
                        // This makes the cereal data
                        data_sr <= {data_sr[DONE_COUNT-2:0], 1'b0};
                end
        end
        // This is how we clock
        always #5 clk = ~clk;
endmodule
```

Screenshot:



```
Signals
Time
          clk =0
         data =0
data_sr[47:0] =000000000000
         done =1
    fec[95:0] =000E8C037C0DF00E828C0E5E
        start =0
      started =1
```

```
Waves
100 sec   200 sec   300 sec   400 sec   500 sec
data_sr: 03+ 06+ 0C+ 18+ 30+ 60+ C0+ 80+ 01+ 02+ 04+ 08+ 10+ 20+ 40+ 81+ 02+ 04+ 08+ 10+ 20+ 40+ 80+ 01+ 03+ 06+ 0C+ 19+ 33+ 66+ CC+ 98+ 30+ 60+ C0+ 81+ 03+ 07+ 0E+ 1D+ 3A+ 74+ E8+ D0+ A0+ 40+ 80+ 000000000000
fec: 000000000000000000000000 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 00+ 000E8C0+
```