EXERCISE 1 (Ethernet Frame Check Sequence)

Daniel Duggan

March 5th 2025

Contents

1 Introduction

The purpose of this exercise is to design a bit error checker for an Ethernet frame. An Ethernet frame is divided as in Table 1.

Field	Size	Description
Preamble	7 bytes	Synchronization pattern for the receiver.
Start Frame Delimiter (SFD)	1 byte	Marks the start of the frame.
Destination MAC Address	6 bytes	Identifies the recipient device.
Source MAC Address	6 bytes	Identifies the sender device.
EtherType/Length	2 bytes	Indicates payload type or length.
Payload (Data + Padding)	46 - 1500 bytes	The actual transmitted data.
Frame Check Sequence (FCS)	4 bytes	Ensures data integrity using CRC.

Table 1: Ethernet Frame Structure

The Frame Check Sequence (FCS) is the last field of the Ethernet frame, and it is used to check for transmission errors. It contains a Cyclic Redundancy Check (CRC) value that is computed from the rest of the frame (excluding the preamble, SFD, and FCS itself). The CRC is encoded via a generating polynomial $G(x)$

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The 32-bit CRC vale is placed in the FCS field and all data is fed into a shift register when received by the receiver. The shift register is required to contain all 0's before any data enters it. Polynomial division is carried out, and once all data has moved through, the shift register contains all 0's if no errors were present in the received Ethernet frame data. This process is termed a Cyclic Redundancy Check CRC. Please note all code used in this exercise is available on github at https://github.com/duggan9265/FPGA-Design-For-Communication-Systems/tree/master/Course_work/Ex_1_Ethernet$_F$CS

2 Serial Implementation

The serial implementation of the CRC required the creation of the design given in figure 1. This has been designed as a Finite State Machine (FSM)[1]. The state machine is in an idle state until the first bit of the ethernet frame is detected i.e. when START_OF_FRAME goes high. When this occurs the data_recieve state is entered. If the bit count is below 32, the input data is complimented i.e. the first 32 bits of the frame are complimented as required. After, the data input is not complimented. The required XOR operations are carried out. When END_OF_FRAME goes high, the fcs_recieve state is entered. This is used to control the complimenting of the last 32 bits of the data frame, and the required XOR operations are carried out. When the bit count is 511 i.e. when all bits have passed through, the check_fcs state is entered which is used to reset the state machine back to idle via the error check

[1] A system top file has been created as implementation was failing with only the fcs_serial_FSM.vhd file. This file can be viewed in the appendix section 4.1

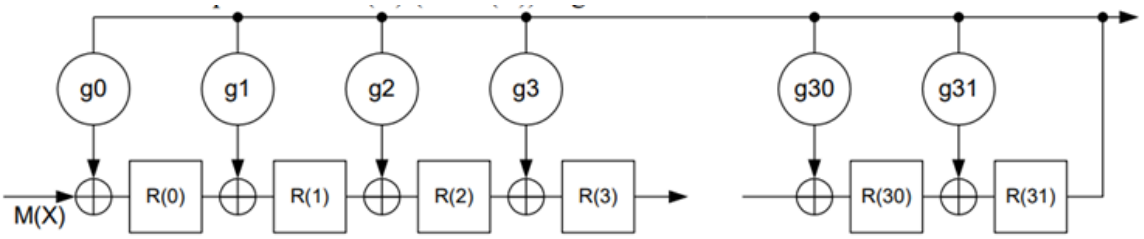

Figure 1: Block diagram of the serial CRC circuit.

state once the FCS error has been calculated to exist or not. If the shift register contains all 0's, FCS_ERROR remains low. Otherwise it goes high to indicate an error has been detected. Please see section 4.2 for the VHDL implementation and section 4.3 for the corresponding testbench.

2.1 Synthesis Report

The synthesis timing report is given in figure 2. The target device is the ZCU102 evaluation board from AMD (XCZU9EG-2FFVB1156E MPSoC). The F_{max} is not a directly obtainable quantity in Vivado. It is calculated by taking the inverse of T - Worst Negative Slack (WNS) i.e. $F_{max} = \frac{1}{4ns - 2.5ns} = 689.655MHz$ where T is the target clock period (4 ns in this case). Figure 4 shows the resource utilisation as determined by synthesis. Furthermore, the synthesised schematic is shown in figure 3.

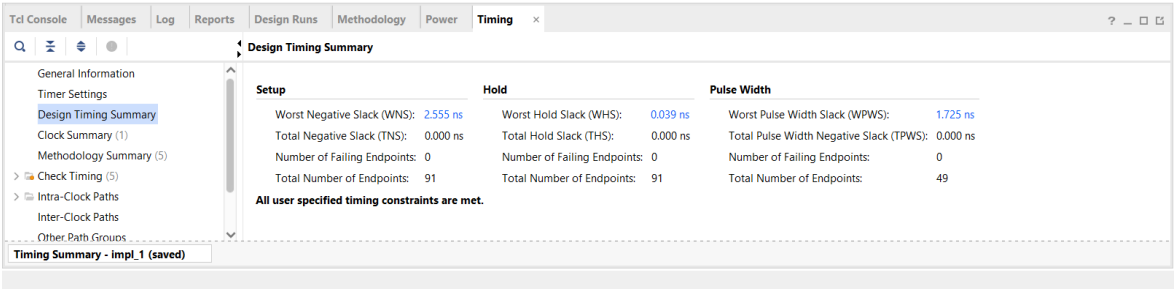

Figure 2: Timing summary of the serial implementation. The WNS is used to calculate F_{max} of 689.655 MHz.

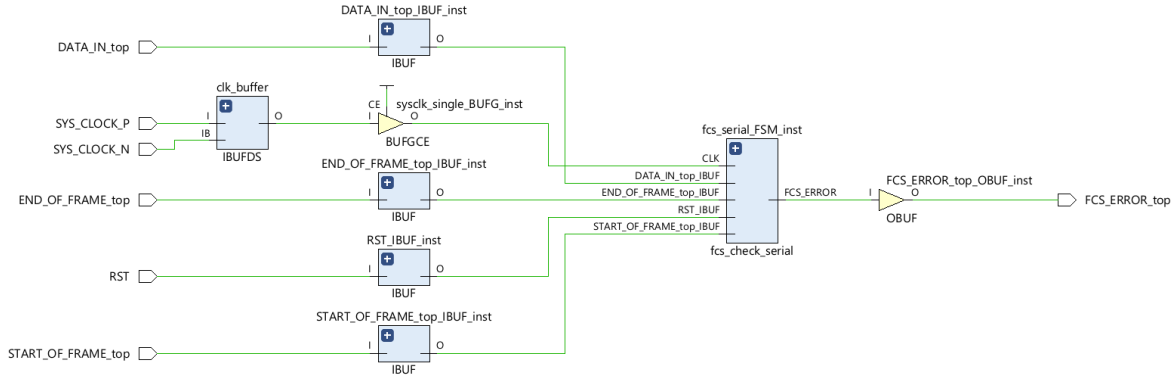

Figure 3: Schematic of the synthesised design.

Resource	Utilization	Available	Utilization %
LUT	43	274080	0.02
FF	48	548160	0.01
IO	7	328	2.13
BUFG	1	404	0.25

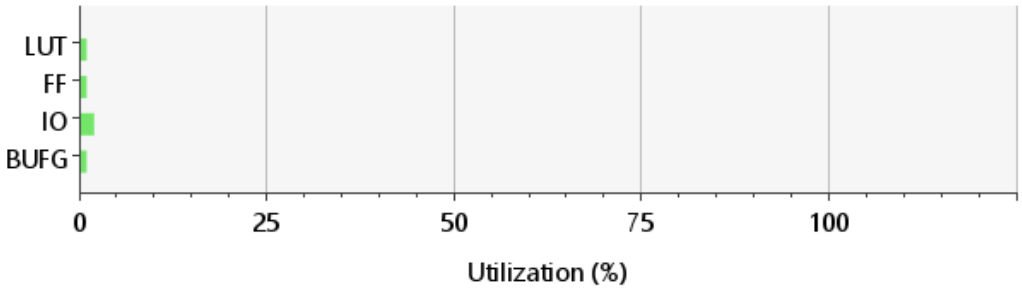

Figure 4: Synthesis report of the serial implementation.

2.2 Implementation - Place and Route

The corresponding place and route utilisation is given in figure 5

Resource	Utilization	Available	Utilization %
LUT	42	274080	0.02
FF	48	548160	0.01
IO	7	328	2.13
BUFG	1	404	0.25

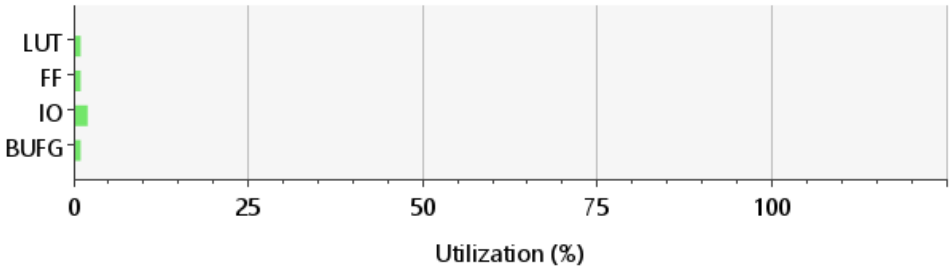

Figure 5: Place and route report

It is effectively equivalent to the synthesis version except one less look up table is used.

3 Parallel Implementation

The parallel implementation is also designed as a finite state machine. Minor changes such as having a byte counter as opposed to a bit counter were made. The major change however is the more complicated boolean expressions required to determine the values of the registers for each shift. These expressions were calculated using a 40x40 matrix taken to the power of 8. Python code was used to determine the expressions. The code for this is given in 4.4. The VHDL implementation of this design is given in section 4.6 along with the testbench in section 4.7.

3.1 Synthesis Report

The synthesis report for the parallel design is given in figure 9. It is seen that the obtained values give a better F_{max} value of 645.161 MHz. Although this value is $\frac{689.655}{645.161} = 1.07$times faster, it is still very slow. However it should be noted that as Vivado does not directly report the F_{max} value, nor try to maximise it, this methodology may not give the most precise results[1]. From the synthesis utilisation report, although the design is quicker, it uses slightly more resources e.g. 57 LUT's v 43 LUT's in the serial design.

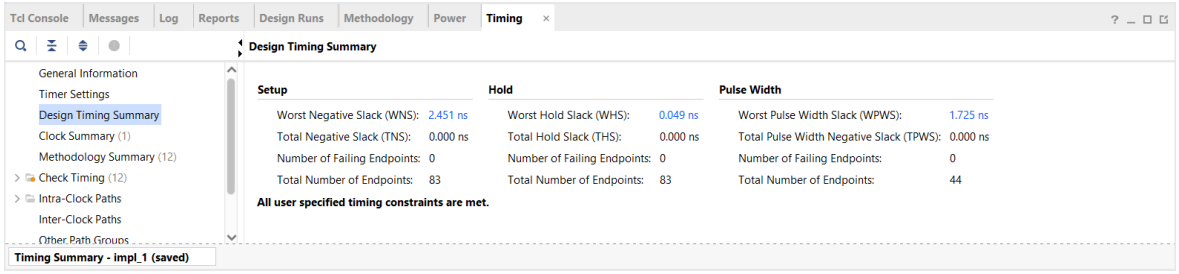

Figure 6: Timing summary of the parallel implementation. The WNS is used to calculate F_{max} of 645.161 MHz.

Resource	Utilization	Available	Utilization %
LUT	59	274080	0.02
FF	43	548160	0.01
IO	14	328	4.27
BUFG	1	404	0.25

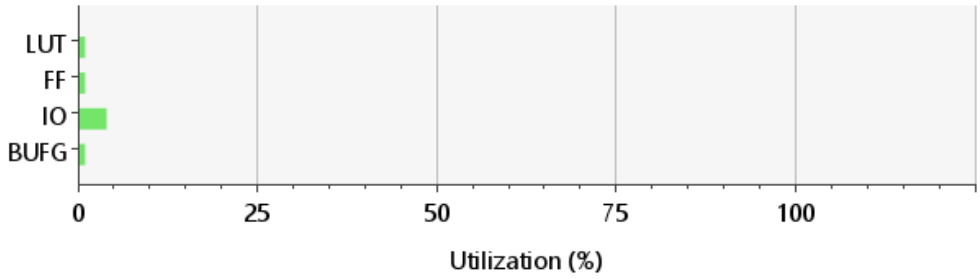

Figure 7: Synthesis utilisation report of the parallel implementation.

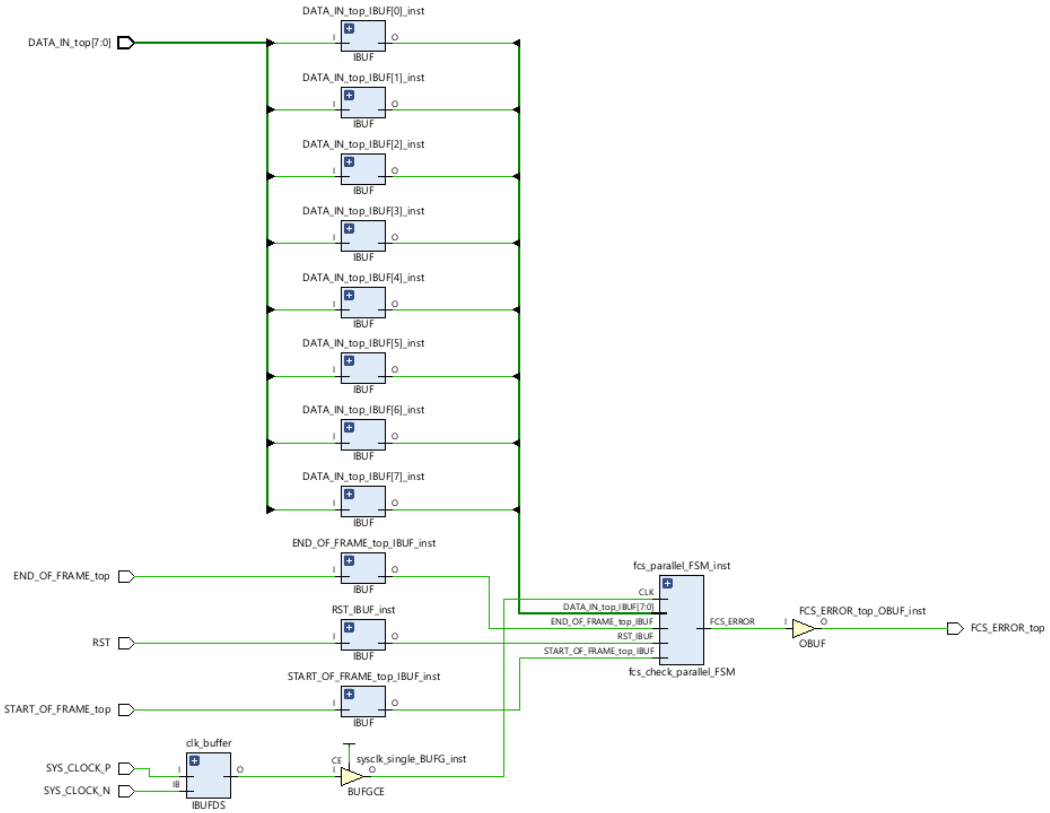

Figure 8: Schematic of the parallel implementation.

3.2 Implementation Report - Place and Route

The implementation report is given below.

Resource	Utilization	Available	Utilization %
LUT	57	274080	0.02
FF	43	548160	0.01
IO	14	328	4.27
BUFG	1	404	0.25

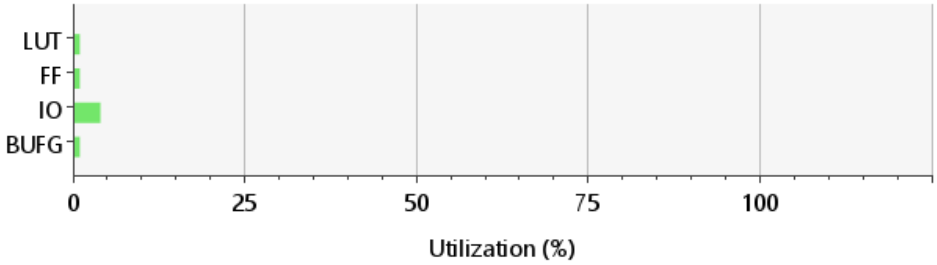

Figure 9: Utilisation of resources from the implementation of the parallel design.

References

[1] https://adaptivesupport.amd.com/s/question/0D52E000077tumHSAQ/which-one-is-better-high-or-low-positive-wns?language=en_US

4 Appendix

4.1 Serial Implementation - System Top (Sys_top.vhd)

```
1  -- Author Daniel Duggan
2  -- vhdl-linter-disable type-resolved
3  LIBRARY IEEE;
4  USE IEEE.STD_LOGIC_1164.ALL;
5
6  USE IEEE.NUMERIC_STD.ALL;
7
8  LIBRARY UNISIM;
9  USE UNISIM.VCOMPONENTS.ALL; -- Required for IBUFDS -- vhdl-linter-disable-
       line not-declared
10
11 ENTITY sys_top IS
12     PORT(
13         SYS_CLOCK_P : IN STD_LOGIC;  -- LVDS clock positive
14         SYS_CLOCK_N : IN STD_LOGIC;  -- LVDS clock negative
15         RST : IN STD_LOGIC;
16         START_OF_FRAME_top : IN STD_LOGIC;
17         END_OF_FRAME_top : IN STD_LOGIC;
18         DATA_IN_top : IN STD_LOGIC;
19         FCS_ERROR_top : OUT STD_LOGIC -- vhdl-linter-disable-line type-
               resolved
20     );
21 END sys_top;
22
23 ARCHITECTURE rtl OF sys_top IS
24     -- Single-ended clock signal after differential conversion
25     SIGNAL sysclk_single : STD_LOGIC; -- vhdl-linter-disable-line type-
           resolved
26
27 BEGIN
28     -- Convert the differential clock to single-ended using IBUFDS
29     clk_buffer: IBUFDS -- don't use work as this is not a user defined
           entity -- vhdl-linter-disable-line not-declared
30     PORT MAP (
31         I  => SYS_CLOCK_P,  -- Positive clock input
```

```
        IB => SYS_CLOCK_N,  -- Negative clock input
        O  => sysclk_single -- Output single-ended clock
    );

    -- Instance of fcs_check_serial using the single-ended clock
    fcs_serial_FSM_inst: entity work.fcs_check_serial
    PORT MAP (
        CLK => sysclk_single,  -- Send the converted clock
        RST => RST,
        START_OF_FRAME => START_OF_FRAME_top,
        END_OF_FRAME => END_OF_FRAME_top,
        DATA_IN => DATA_IN_top,
        FCS_ERROR => FCS_ERROR_top
    );

END rtl;
```

4.2 Serial Implementation - fcs_serial_FSM.vhd

```
-- Auther Daniel Duggan
-- vhdl-linter-disable type-resolved
LIBRARY IEEE;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_1164.ALL;

ENTITY fcs_check_serial IS --fsm_fcs
    GENERIC (
        Depth : INTEGER := 32
    );

    PORT (
        CLK : IN STD_ULOGIC; --Sys Clock
        RST : IN STD_ULOGIC; --Async Reset
        START_OF_FRAME : IN STD_ULOGIC := '0'; --Arrival of first bit
        END_OF_FRAME : IN STD_ULOGIC := '0'; --Arrival of 1st bit in FCS
        DATA_IN : IN STD_ULOGIC; --serial input data
        FCS_ERROR : OUT STD_ULOGIC --indicates an error
    );
END ENTITY;

ARCHITECTURE fsm OF fcs_check_serial IS
    TYPE fsm_fcs_type IS
    (idle, data_recieve, fcs_recieve, check_fcs, error_check);
```

```
25      SIGNAL state, next_state : fsm_fcs_type; --state, next_state can take on
            all values inside fsm_fcs_type
26      SIGNAL bit_count : unsigned(9 DOWNTO 0);
27      SIGNAL shift_mem : STD_LOGIC_VECTOR(DEPTH - 1 DOWNTO 0);
28      SIGNAL check_data : STD_LOGIC; -- For debug purposes only
29  BEGIN
30      fsm_process : PROCESS (state, START_OF_FRAME, END_OF_FRAME, bit_count)
31      BEGIN
32          CASE state IS
33              WHEN idle =>
34                  IF START_OF_FRAME = '1' THEN
35                      next_state <= data_recieve;
36                  ELSE
37                      next_state <= idle;
38                  END IF;
39
40              WHEN data_recieve =>
41                  IF END_OF_FRAME = '1' THEN
42                      next_state <= fcs_recieve;
43                  ELSE
44                      next_state <= data_recieve;
45                  END IF;
46
47              WHEN fcs_recieve =>
48                  IF bit_count = 511 THEN
49                      next_state <= check_fcs;
50                  ELSE
51                      next_state <= fcs_recieve;
52                  END IF;
53
54              WHEN CHECK_FCS =>
55                  next_state <= error_check;
56
57              WHEN error_check =>
58                  next_state <= idle; -- Reset for the next frame
59
60              WHEN OTHERS =>
61                  next_state <= idle;
62          END CASE;
63      END PROCESS;
64
65      PROCESS (CLK, RST)
66      BEGIN
67          IF RST = '0' THEN --active low reset
68              state <= idle;
```

```
69              shift_mem <= (OTHERS => '0');
70              bit_count <= (OTHERS => '0');
71              bit_count <= (OTHERS => '0');
72          ELSIF rising_edge(CLK) THEN
73              state <= next_state;
74
75              CASE state IS
76                  WHEN idle =>
77                      shift_mem <= (OTHERS => '0');
78                      bit_count <= (OTHERS => '0');
79                      FCS_ERROR <= '0';
80
81                  WHEN data_recieve =>
82                      check_data <= NOT DATA_IN;
83                      IF bit_count < 32 THEN
84                          shift_mem(0) <= shift_mem(31) XOR (NOT(DATA_IN)); --
                                Complement first 32 bits
85                          shift_mem(1) <= shift_mem(0) XOR shift_mem(31);
86                          shift_mem(2) <= shift_mem(1) XOR shift_mem(31);
87                          shift_mem(3) <= shift_mem(2);
88                          shift_mem(4) <= shift_mem(3) XOR shift_mem(31);
89                          shift_mem(5) <= shift_mem(4) XOR shift_mem(31);
90                          shift_mem(6) <= shift_mem(5);
91                          shift_mem(7) <= shift_mem(6) XOR shift_mem(31);
92                          shift_mem(8) <= shift_mem(7) XOR shift_mem(31);
93                          shift_mem(9) <= shift_mem(8);
94                          shift_mem(10) <= shift_mem(9) XOR shift_mem(31);
95                          shift_mem(11) <= shift_mem(10) XOR shift_mem(31);
96                          shift_mem(12) <= shift_mem(11) XOR shift_mem(31);
97                          shift_mem(13) <= shift_mem(12);
98                          shift_mem(14) <= shift_mem(13);
99                          shift_mem(15) <= shift_mem(14);
100                         shift_mem(16) <= shift_mem(15) XOR shift_mem(31);
101                         shift_mem(17) <= shift_mem(16);
102                         shift_mem(18) <= shift_mem(17);
103                         shift_mem(19) <= shift_mem(18);
104                         shift_mem(20) <= shift_mem(19);
105                         shift_mem(21) <= shift_mem(20);
106                         shift_mem(22) <= shift_mem(21) XOR shift_mem(31);
107                         shift_mem(23) <= shift_mem(22) XOR shift_mem(31);
108                         shift_mem(24) <= shift_mem(23);
109                         shift_mem(25) <= shift_mem(24);
110                         shift_mem(26) <= shift_mem(25) XOR shift_mem(31);
111                         shift_mem(27) <= shift_mem(26);
112                         shift_mem(28) <= shift_mem(27);
```

```
113                     shift_mem(29) <= shift_mem(28);
114                     shift_mem(30) <= shift_mem(29);
115                     shift_mem(31) <= shift_mem(30);
116 
117                 ELSE
118                     shift_mem(0) <= shift_mem(31) XOR DATA_IN; -- CRC
                        shift logic
119                     shift_mem(1) <= shift_mem(0) XOR shift_mem(31);
120                     shift_mem(2) <= shift_mem(1) XOR shift_mem(31);
121                     shift_mem(3) <= shift_mem(2);
122                     shift_mem(4) <= shift_mem(3) XOR shift_mem(31);
123                     shift_mem(5) <= shift_mem(4) XOR shift_mem(31);
124                     shift_mem(6) <= shift_mem(5);
125                     shift_mem(7) <= shift_mem(6) XOR shift_mem(31);
126                     shift_mem(8) <= shift_mem(7) XOR shift_mem(31);
127                     shift_mem(9) <= shift_mem(8);
128                     shift_mem(10) <= shift_mem(9) XOR shift_mem(31);
129                     shift_mem(11) <= shift_mem(10) XOR shift_mem(31);
130                     shift_mem(12) <= shift_mem(11) XOR shift_mem(31);
131                     shift_mem(13) <= shift_mem(12);
132                     shift_mem(14) <= shift_mem(13);
133                     shift_mem(15) <= shift_mem(14);
134                     shift_mem(16) <= shift_mem(15) XOR shift_mem(31);
135                     shift_mem(17) <= shift_mem(16);
136                     shift_mem(18) <= shift_mem(17);
137                     shift_mem(19) <= shift_mem(18);
138                     shift_mem(20) <= shift_mem(19);
139                     shift_mem(21) <= shift_mem(20);
140                     shift_mem(22) <= shift_mem(21) XOR shift_mem(31);
141                     shift_mem(23) <= shift_mem(22) XOR shift_mem(31);
142                     shift_mem(24) <= shift_mem(23);
143                     shift_mem(25) <= shift_mem(24);
144                     shift_mem(26) <= shift_mem(25) XOR shift_mem(31);
145                     shift_mem(27) <= shift_mem(26);
146                     shift_mem(28) <= shift_mem(27);
147                     shift_mem(29) <= shift_mem(28);
148                     shift_mem(30) <= shift_mem(29);
149                     shift_mem(31) <= shift_mem(30);
150                 END IF;
151                 bit_count <= bit_count + 1;
152 
153             WHEN fcs_recieve =>
154 
155                 IF bit_count > 479 THEN
156                 check_data <= NOT DATA_IN;
```

```
157                     bit_count <= bit_count + 1;
158                     shift_mem(0) <= shift_mem(31) XOR (NOT(DATA_IN)); --
                            Complement bits
159                     shift_mem(1) <= shift_mem(0) XOR shift_mem(31);
160                     shift_mem(2) <= shift_mem(1) XOR shift_mem(31);
161                     shift_mem(3) <= shift_mem(2);
162                     shift_mem(4) <= shift_mem(3) XOR shift_mem(31);
163                     shift_mem(5) <= shift_mem(4) XOR shift_mem(31);
164                     shift_mem(6) <= shift_mem(5);
165                     shift_mem(7) <= shift_mem(6) XOR shift_mem(31);
166                     shift_mem(8) <= shift_mem(7) XOR shift_mem(31);
167                     shift_mem(9) <= shift_mem(8);
168                     shift_mem(10) <= shift_mem(9) XOR shift_mem(31);
169                     shift_mem(11) <= shift_mem(10) XOR shift_mem(31);
170                     shift_mem(12) <= shift_mem(11) XOR shift_mem(31);
171                     shift_mem(13) <= shift_mem(12);
172                     shift_mem(14) <= shift_mem(13);
173                     shift_mem(15) <= shift_mem(14);
174                     shift_mem(16) <= shift_mem(15) XOR shift_mem(31);
175                     shift_mem(17) <= shift_mem(16);
176                     shift_mem(18) <= shift_mem(17);
177                     shift_mem(19) <= shift_mem(18);
178                     shift_mem(20) <= shift_mem(19);
179                     shift_mem(21) <= shift_mem(20);
180                     shift_mem(22) <= shift_mem(21) XOR shift_mem(31);
181                     shift_mem(23) <= shift_mem(22) XOR shift_mem(31);
182                     shift_mem(24) <= shift_mem(23);
183                     shift_mem(25) <= shift_mem(24);
184                     shift_mem(26) <= shift_mem(25) XOR shift_mem(31);
185                     shift_mem(27) <= shift_mem(26);
186                     shift_mem(28) <= shift_mem(27);
187                     shift_mem(29) <= shift_mem(28);
188                     shift_mem(30) <= shift_mem(29);
189                     shift_mem(31) <= shift_mem(30);
190
191                     else
192                     check_data <= DATA_IN;
193                     bit_count <= bit_count + 1;
194                     shift_mem(0) <= shift_mem(31) XOR DATA_IN;
195                     shift_mem(1) <= shift_mem(0) XOR shift_mem(31);
196                     shift_mem(2) <= shift_mem(1) XOR shift_mem(31);
197                     shift_mem(3) <= shift_mem(2);
198                     shift_mem(4) <= shift_mem(3) XOR shift_mem(31);
199                     shift_mem(5) <= shift_mem(4) XOR shift_mem(31);
200                     shift_mem(6) <= shift_mem(5);
```

```
201                         shift_mem(7) <= shift_mem(6) XOR shift_mem(31);
202                         shift_mem(8) <= shift_mem(7) XOR shift_mem(31);
203                         shift_mem(9) <= shift_mem(8);
204                         shift_mem(10) <= shift_mem(9) XOR shift_mem(31);
205                         shift_mem(11) <= shift_mem(10) XOR shift_mem(31);
206                         shift_mem(12) <= shift_mem(11) XOR shift_mem(31);
207                         shift_mem(13) <= shift_mem(12);
208                         shift_mem(14) <= shift_mem(13);
209                         shift_mem(15) <= shift_mem(14);
210                         shift_mem(16) <= shift_mem(15) XOR shift_mem(31);
211                         shift_mem(17) <= shift_mem(16);
212                         shift_mem(18) <= shift_mem(17);
213                         shift_mem(19) <= shift_mem(18);
214                         shift_mem(20) <= shift_mem(19);
215                         shift_mem(21) <= shift_mem(20);
216                         shift_mem(22) <= shift_mem(21) XOR shift_mem(31);
217                         shift_mem(23) <= shift_mem(22) XOR shift_mem(31);
218                         shift_mem(24) <= shift_mem(23);
219                         shift_mem(25) <= shift_mem(24);
220                         shift_mem(26) <= shift_mem(25) XOR shift_mem(31);
221                         shift_mem(27) <= shift_mem(26);
222                         shift_mem(28) <= shift_mem(27);
223                         shift_mem(29) <= shift_mem(28);
224                         shift_mem(30) <= shift_mem(29);
225                         shift_mem(31) <= shift_mem(30);
226                         END IF;
227
228                 WHEN CHECK_FCS =>
229                     IF shift_mem = (shift_mem'RANGE => '0') THEN
230                         FCS_ERROR <= '0';
231                     ELSE
232                         FCS_ERROR <= '1';
233                     END IF;
234
235                 WHEN error_check =>
236                     FCS_ERROR <= '0'; -- Reset for next frame
237
238                 WHEN OTHERS =>
239                     NULL;
240             END CASE;
241         END IF;
242     END PROCESS;
243 END ARCHITECTURE;
```

4.3 Serial Implementation Testbench (fcs_serial_FSM.vhd)

```
1  //Author Daniel Duggan
2
3  module FCS_serial;
4
5    timeunit 1ns;
6    timeprecision 1ps; //time precision specifies how delay values are rounded
         relative to timeunit
7
8    logic sysclk_p;
9    logic sysclk_n;
10   logic rst; //reset logic
11   logic start_of_frame_top, end_of_frame_top, data_in, fcs_error_top;
12   parameter clk_period = 4ns; // Since sys_clock every 2ns
13
14
15   // clock generation
        ------------------------------------------------------------------
16   logic sys_clk = 0;
17   always #2ns sys_clk = ~sys_clk; // Toggle every 2 ns (1 cycle = 4 ns)
18   assign sysclk_p = sys_clk;
19   assign sysclk_n = ~sys_clk;
20   //
        ------------------------------------------------------------------------------------


21
22   // Data in (Ethernet Frame)
        ------------------------------------------------------------
23   logic [511:0] ethernet_frame = {
24     128'h00_10_A4_7B_EA_80_00_12_34_56_78_90_08_00_45_00,
25     128'h00_2E_B3_FE_00_00_80_11_05_40_C0_A8_00_2C_C0_A8,
26     128'h00_04_04_00_04_00_00_1A_2D_E8_00_01_02_03_04_05,
27     128'h06_07_08_09_0A_0B_0C_0D_0E_0F_10_11_E6_C5_3D_B2
28   };
29
30     logic [511:0] ethernet_frame_incorrect = {
31       128'h00_10_A4_7B_EA_80_00_12_34_56_78_90_08_00_45_00,
32       128'h00_2E_B3_FE_00_00_80_11_05_40_C0_A8_00_2C_C0_A8,
33       128'h00_04_04_00_04_00_00_1A_2D_E8_00_01_02_03_04_05,
34       128'h06_07_08_09_0A_0B_0C_0D_0E_0F_10_11_FF_FF_FF_FF
35     };
36
```

```
37  //
        ----------------------------------------------------------------------------------
38
39  // DUT
        ----------------------------------------------------------------------------------
40  Sys_top Sys_top_DUT(
41            .SYS_CLOCK_P (sysclk_p),
42            .SYS_CLOCK_N (sysclk_n),
43            .RST (rst),
44            .START_OF_FRAME_top (start_of_frame_top),
45            .END_OF_FRAME_top (end_of_frame_top),
46            .DATA_IN_top (data_in),
47            .FCS_ERROR_top (fcs_error_top)
48          );
49  //
        ----------------------------------------------------------------------------------
50
51  initial
52  begin
53    // Initialize signals
54    start_of_frame_top = 0;
55    end_of_frame_top   = 0;
56    data_in            = 0;
57    rst = 1'b0;
58
59    // Apply Reset
60    repeat (5) @(posedge sys_clk);
61    rst = 1'b1;
62    repeat (5) @(posedge sys_clk);
63
64    // Begin Transmission
65    start_of_frame_top = 1'b1;  // Indicate start of frame
66    @(posedge sys_clk);//#posedge sys_clk;
67    start_of_frame_top = 1'b0;
68
69    for (int i = 511; i >= 0; i--)
70    begin
71      //#1;
72      data_in = ethernet_frame[i];  // Send data in serially
73      /// Set end_of_frame at the last bit
74      if (i == 32) //31 should go high at bit 480? Does at 481. Why?
75        end_of_frame_top = 1'b1;
```

```
76          else
77            end_of_frame_top = 1'b0;
78          @(posedge sys_clk); //#clk_period;
79
80        end
81
82        repeat (3) @(posedge sys_clk);
83        $finish;
84      end
85  endmodule
```

4.4 Python Code for Boolean Expressions in Parallel Implementation

```
import numpy as np
g_matrix_values = np.array([[1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
g_matrix_values.shape
zero_col_top_left = np.zeros((31,1))
iden_top_left = np.eye(31)
iden_top_left.shape
join_zero_ide_upper_left = np.hstack((zero_col_top_left, iden_top_left))
join_zero_ide_upper_left.shape
join_upper_left = np.vstack((join_zero_ide_upper_left, g_matrix_values))
join_upper_left
zero_col_top_right = np.zeros((32,8))
zero_col_top_right.shape
join_upper_right_to_upper_left = np.hstack((join_upper_left, zero_col_top_right)
top_half = join_upper_right_to_upper_left
zero_bottom_left = np.zeros((8,32))
zero_bottom_left.shape
zero_bottom_left[0,0] = 1
zero_bottom_left.shape
iden_bottom_right = np.eye(7)
row_zeros_col = np.zeros((1,7))
bot_right_7x7 = np.vstack((row_zeros_col,iden_bottom_right ))
bot_right_7x7.shape
bot_right_zeros_row = np.zeros((8,1))
stack_bot_right = np.hstack((bot_right_7x7, bot_right_zeros_row))
stack_bot_right
stack_bot_right[0,7] = 1
bottom_half = np.hstack((zero_bottom_left,stack_bot_right))
bottom_half.shape
```

```
full_matrix = np.vstack((top_half,bottom_half))
full_matrix.shape
# Compute A^8 and mod 2
full_matrix_power = np.linalg.matrix_power(full_matrix, 8) % 2
num_cols = full_matrix_power.shape[1]

for x in range(num_cols):
    if x > 31:  # Stop after 32 registers
        break

    shift_memx = []
    print(f'shift_mem({x}) <= ', end='')

    for y in range(full_matrix_power.shape[0]):  # Iterate over rows
        if full_matrix_power[y, x]:  # If the element is 1
            if y > 31:
                shift_memx.append(f'NOT(DATA_IN({y - 32}))')
            else:
                shift_memx.append(f'shift_mem({y})')

    print(' xor '.join(shift_memx) + ';')  # Print the XOR boolean expression
```

4.5 Parallel Implementation - System Top (Sys_top_para.vhd)

```
1  -- Auther Daniel Duggan
2  -- Author Daniel Duggan
3  -- vhdl-linter-disable type-resolved
4  LIBRARY IEEE;
5  USE IEEE.STD_LOGIC_1164.ALL;
6
7  USE IEEE.NUMERIC_STD.ALL;
8
9  LIBRARY UNISIM;
10 USE UNISIM.VCOMPONENTS.ALL; -- Required for IBUFDS -- vhdl-linter-disable-
       line not-declared
11
12 ENTITY Sys_Top_para IS
13     PORT(
14         SYS_CLOCK_P : IN STD_LOGIC;  -- LVDS clock positive
15         SYS_CLOCK_N : IN STD_LOGIC;  -- LVDS clock negative
16         RST : IN STD_LOGIC;
17         START_OF_FRAME_top : IN STD_LOGIC;
```

```
18          END_OF_FRAME_top : IN STD_LOGIC;
19          DATA_IN_top : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
20          FCS_ERROR_top : OUT STD_LOGIC -- vhdl-linter-disable-line type-
                resolved
21      );
22  END Sys_Top_para;
23
24  ARCHITECTURE rtl OF sys_top_para IS
25      -- Single-ended clock signal after differential conversion
26      SIGNAL sysclk_single : STD_LOGIC; -- vhdl-linter-disable-line type-
            resolved
27
28  BEGIN
29      -- Convert the differential clock to single-ended using IBUFDS
30      clk_buffer: IBUFDS -- don't use work as this is not a user defined
            entity -- vhdl-linter-disable-line not-declared
31      PORT MAP (
32          I  => SYS_CLOCK_P,  -- Positive clock input
33          IB => SYS_CLOCK_N,  -- Negative clock input
34          O  => sysclk_single -- Output single-ended clock
35      );
36
37      -- Instance of fcs_check_serial using the single-ended clock
38      fcs_parallel_FSM_inst: entity work.fcs_check_parallel_FSM
39      PORT MAP (
40          CLK => sysclk_single,  -- Send the converted clock
41          RST => RST,
42          START_OF_FRAME => START_OF_FRAME_top,
43          END_OF_FRAME => END_OF_FRAME_top,
44          DATA_IN => DATA_IN_top,
45          FCS_ERROR => FCS_ERROR_top
46      );
47
48  END rtl;
```

4.6 Parallel Implementation - fcs_parallel.vhd

```
1  -- Author Daniel Duggan
2  -- vhdl-linter-disable type-resolved
3  LIBRARY IEEE;
4  USE ieee.std_logic_1164.ALL;
5  USE ieee.numeric_std.ALL;
6
7  ENTITY fcs_check_parallel_FSM IS
```

```
8
9       GENERIC (
10          Depth : INTEGER := 32
11      );
12
13      PORT (
14
15          CLK : IN STD_LOGIC; -- system clock
16          RST : IN STD_LOGIC; -- asynchronous reset
17          START_OF_FRAME : IN STD_LOGIC; -- arrival of the first byte.
18          END_OF_FRAME : IN STD_LOGIC; -- arrival of the first byte in FCS.
19          DATA_IN : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- input data.
20          FCS_ERROR : OUT STD_LOGIC -- indicates an error.
21      );
22  END fcs_check_parallel_FSM;
23
24  ARCHITECTURE fsm OF fcs_check_parallel_FSM IS
25      TYPE fsm_fcs_type IS
26      (idle, data_recieve, fcs_recieve, check_fcs, error_check);
27      SIGNAL state, next_state : fsm_fcs_type; --state, next_state can take on
            all values inside fsm_fcs_type
28      SIGNAL byte_count : unsigned(6 DOWNTO 0);
29      SIGNAL shift_mem : STD_LOGIC_VECTOR(DEPTH - 1 DOWNTO 0);
30      SIGNAL check_data : STD_LOGIC_VECTOR(7 DOWNTO 0); -- For debug purposes
           only
31  BEGIN
32      fsm_process : PROCESS (state, START_OF_FRAME, END_OF_FRAME, byte_count)
33      BEGIN
34          CASE state IS
35              WHEN idle =>
36                  IF START_OF_FRAME = '1' THEN
37                      next_state <= data_recieve;
38                  ELSE
39                      next_state <= idle;
40                  END IF;
41
42              WHEN data_recieve =>
43                  IF END_OF_FRAME = '1' THEN
44                      next_state <= fcs_recieve;
45                  ELSE
46                      next_state <= data_recieve;
47                  END IF;
48
49              WHEN fcs_recieve =>
50                  IF byte_count = 64 THEN
```

```
51                      next_state <= check_fcs;
52                  ELSE
53                      next_state <= fcs_recieve;
54                  END IF;
55
56              WHEN CHECK_FCS =>
57                  next_state <= error_check;
58
59              WHEN error_check =>
60                  next_state <= idle; -- Reset for the next frame
61
62              WHEN OTHERS =>
63                  next_state <= idle;
64          END CASE;
65      END PROCESS;
66
67      PROCESS (CLK, RST)
68      BEGIN
69          IF RST = '0' THEN --active low reset
70              state <= idle;
71              shift_mem <= (OTHERS => '0');
72              byte_count <= (OTHERS => '0');
73          ELSIF rising_edge(CLK) THEN
74              state <= next_state;
75
76              CASE state IS
77                  WHEN idle =>
78                      shift_mem <= (OTHERS => '0');
79                      byte_count <= (OTHERS => '0');
80                      FCS_ERROR <= '0';
81
82                  WHEN data_recieve =>
83
84                      IF byte_count < 3 THEN
85                          check_data <= NOT DATA_IN;
86                          shift_mem(0) <= shift_mem(24) XOR shift_mem(30) XOR
                                NOT(DATA_IN(7));
87                          shift_mem(1) <= shift_mem(24) XOR shift_mem(25) XOR
                                shift_mem(30) XOR shift_mem(31) XOR NOT(DATA_IN
                                (6));
88                          shift_mem(2) <= shift_mem(24) XOR shift_mem(25) XOR
                                shift_mem(26) XOR shift_mem(30) XOR shift_mem(31)
                                 XOR NOT(DATA_IN(5));
```

```
89                  shift_mem(3) <= shift_mem(25) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(31) XOR NOT(DATA_IN
                        (4));
90                  shift_mem(4) <= shift_mem(24) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(30)
                         XOR NOT(DATA_IN(3));
91                  shift_mem(5) <= shift_mem(24) XOR shift_mem(25) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(29)
                         XOR shift_mem(30) XOR shift_mem(31) XOR NOT(
                        DATA_IN(2));
92                  shift_mem(6) <= shift_mem(25) XOR shift_mem(26) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem(30)
                         XOR shift_mem(31) XOR NOT(DATA_IN(1));
93                  shift_mem(7) <= shift_mem(24) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(29) XOR shift_mem(31)
                         XOR NOT(DATA_IN(0));
94                  shift_mem(8) <= shift_mem(0) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(27) XOR shift_mem(28)
                        ;
95                  shift_mem(9) <= shift_mem(1) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(28) XOR shift_mem(29)
                        ;
96                  shift_mem(10) <= shift_mem(2) XOR shift_mem(24) XOR
                        shift_mem(26) XOR shift_mem(27) XOR shift_mem(29)
                        ;
97                  shift_mem(11) <= shift_mem(3) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(27) XOR shift_mem(28)
                        ;
98                  shift_mem(12) <= shift_mem(4) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(26) XOR shift_mem(28)
                         XOR shift_mem(29) XOR shift_mem(30);
99                  shift_mem(13) <= shift_mem(5) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(27) XOR shift_mem(29)
                         XOR shift_mem(30) XOR shift_mem(31);
100                 shift_mem(14) <= shift_mem(6) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(30)
                         XOR shift_mem(31);
101                 shift_mem(15) <= shift_mem(7) XOR shift_mem(27) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem(31)
                        ;
102                 shift_mem(16) <= shift_mem(8) XOR shift_mem(24) XOR
                        shift_mem(28) XOR shift_mem(29);
103                 shift_mem(17) <= shift_mem(9) XOR shift_mem(25) XOR
                        shift_mem(29) XOR shift_mem(30);
```

```
104                 shift_mem(18) <= shift_mem(10) XOR shift_mem(26) XOR
                        shift_mem(30) XOR shift_mem(31);
105                 shift_mem(19) <= shift_mem(11) XOR shift_mem(27) XOR
                        shift_mem(31);
106                 shift_mem(20) <= shift_mem(12) XOR shift_mem(28);
107                 shift_mem(21) <= shift_mem(13) XOR shift_mem(29);
108                 shift_mem(22) <= shift_mem(14) XOR shift_mem(24);
109                 shift_mem(23) <= shift_mem(15) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(30);
110                 shift_mem(24) <= shift_mem(16) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(31);
111                 shift_mem(25) <= shift_mem(17) XOR shift_mem(26) XOR
                        shift_mem(27);
112                 shift_mem(26) <= shift_mem(18) XOR shift_mem(24) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem
                       (30);
113                 shift_mem(27) <= shift_mem(19) XOR shift_mem(25) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem
                       (31);
114                 shift_mem(28) <= shift_mem(20) XOR shift_mem(26) XOR
                        shift_mem(29) XOR shift_mem(30);
115                 shift_mem(29) <= shift_mem(21) XOR shift_mem(27) XOR
                        shift_mem(30) XOR shift_mem(31);
116                 shift_mem(30) <= shift_mem(22) XOR shift_mem(28) XOR
                        shift_mem(31);
117                 shift_mem(31) <= shift_mem(23) XOR shift_mem(29);
118
119             ELSE
120                 check_data <= DATA_IN;
121                 shift_mem(0) <= shift_mem(24) XOR shift_mem(30) XOR
                       DATA_IN(7);
122                 shift_mem(1) <= shift_mem(24) XOR shift_mem(25) XOR
                       shift_mem(30) XOR shift_mem(31) XOR DATA_IN(6);
123                 shift_mem(2) <= shift_mem(24) XOR shift_mem(25) XOR
                       shift_mem(26) XOR shift_mem(30) XOR shift_mem(31)
                        XOR DATA_IN(5);
124                 shift_mem(3) <= shift_mem(25) XOR shift_mem(26) XOR
                       shift_mem(27) XOR shift_mem(31) XOR DATA_IN(4);
125                 shift_mem(4) <= shift_mem(24) XOR shift_mem(26) XOR
                       shift_mem(27) XOR shift_mem(28) XOR shift_mem(30)
                        XOR DATA_IN(3);
126                 shift_mem(5) <= shift_mem(24) XOR shift_mem(25) XOR
                       shift_mem(27) XOR shift_mem(28) XOR shift_mem(29)
                        XOR shift_mem(30) XOR shift_mem(31) XOR DATA_IN
                       (2);
```

```
127                 shift_mem(6) <= shift_mem(25) XOR shift_mem(26) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem(30)
                         XOR shift_mem(31) XOR DATA_IN(1);
128                 shift_mem(7) <= shift_mem(24) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(29) XOR shift_mem(31)
                         XOR DATA_IN(0);
129                 shift_mem(8) <= shift_mem(0) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(27) XOR shift_mem(28)
                        ;
130                 shift_mem(9) <= shift_mem(1) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(28) XOR shift_mem(29)
                        ;
131                 shift_mem(10) <= shift_mem(2) XOR shift_mem(24) XOR
                        shift_mem(26) XOR shift_mem(27) XOR shift_mem(29)
                        ;
132                 shift_mem(11) <= shift_mem(3) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(27) XOR shift_mem(28)
                        ;
133                 shift_mem(12) <= shift_mem(4) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(26) XOR shift_mem(28)
                         XOR shift_mem(29) XOR shift_mem(30);
134                 shift_mem(13) <= shift_mem(5) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(27) XOR shift_mem(29)
                         XOR shift_mem(30) XOR shift_mem(31);
135                 shift_mem(14) <= shift_mem(6) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(30)
                         XOR shift_mem(31);
136                 shift_mem(15) <= shift_mem(7) XOR shift_mem(27) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem(31)
                        ;
137                 shift_mem(16) <= shift_mem(8) XOR shift_mem(24) XOR
                        shift_mem(28) XOR shift_mem(29);
138                 shift_mem(17) <= shift_mem(9) XOR shift_mem(25) XOR
                        shift_mem(29) XOR shift_mem(30);
139                 shift_mem(18) <= shift_mem(10) XOR shift_mem(26) XOR
                         shift_mem(30) XOR shift_mem(31);
140                 shift_mem(19) <= shift_mem(11) XOR shift_mem(27) XOR
                         shift_mem(31);
141                 shift_mem(20) <= shift_mem(12) XOR shift_mem(28);
142                 shift_mem(21) <= shift_mem(13) XOR shift_mem(29);
143                 shift_mem(22) <= shift_mem(14) XOR shift_mem(24);
144                 shift_mem(23) <= shift_mem(15) XOR shift_mem(24) XOR
                         shift_mem(25) XOR shift_mem(30);
145                 shift_mem(24) <= shift_mem(16) XOR shift_mem(25) XOR
                         shift_mem(26) XOR shift_mem(31);
```

```
146                     shift_mem(25) <= shift_mem(17) XOR shift_mem(26) XOR
                            shift_mem(27);
147                     shift_mem(26) <= shift_mem(18) XOR shift_mem(24) XOR
                            shift_mem(27) XOR shift_mem(28) XOR shift_mem
                           (30);
148                     shift_mem(27) <= shift_mem(19) XOR shift_mem(25) XOR
                            shift_mem(28) XOR shift_mem(29) XOR shift_mem
                           (31);
149                     shift_mem(28) <= shift_mem(20) XOR shift_mem(26) XOR
                            shift_mem(29) XOR shift_mem(30);
150                     shift_mem(29) <= shift_mem(21) XOR shift_mem(27) XOR
                            shift_mem(30) XOR shift_mem(31);
151                     shift_mem(30) <= shift_mem(22) XOR shift_mem(28) XOR
                            shift_mem(31);
152                     shift_mem(31) <= shift_mem(23) XOR shift_mem(29);
153                 END IF;
154                 byte_count <= byte_count + 1;
155
156             WHEN fcs_recieve =>
157
158                 IF byte_count > 61 THEN -- XOR last 32 bits (bytes
                       60,61,62)
159                     check_data <= NOT DATA_IN;
160                     byte_count <= byte_count + 1;
161                     shift_mem(0) <= shift_mem(24) XOR shift_mem(30) XOR
                        NOT(DATA_IN(7));
162                     shift_mem(1) <= shift_mem(24) XOR shift_mem(25) XOR
                        shift_mem(30) XOR shift_mem(31) XOR NOT(DATA_IN
                        (6));
163                     shift_mem(2) <= shift_mem(24) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(30) XOR shift_mem(31)
                         XOR NOT(DATA_IN(5));
164                     shift_mem(3) <= shift_mem(25) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(31) XOR NOT(DATA_IN
                        (4));
165                     shift_mem(4) <= shift_mem(24) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(30)
                         XOR NOT(DATA_IN(3));
166                     shift_mem(5) <= shift_mem(24) XOR shift_mem(25) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(29)
                         XOR shift_mem(30) XOR shift_mem(31) XOR NOT(
                        DATA_IN(2));
167                     shift_mem(6) <= shift_mem(25) XOR shift_mem(26) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem(30)
                         XOR shift_mem(31) XOR NOT(DATA_IN(1));
```

```
168                 shift_mem(7) <= shift_mem(24) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(29) XOR shift_mem(31)
                         XOR NOT(DATA_IN(0));
169                 shift_mem(8) <= shift_mem(0) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(27) XOR shift_mem(28)
                        ;
170                 shift_mem(9) <= shift_mem(1) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(28) XOR shift_mem(29)
                        ;
171                 shift_mem(10) <= shift_mem(2) XOR shift_mem(24) XOR
                        shift_mem(26) XOR shift_mem(27) XOR shift_mem(29)
                        ;
172                 shift_mem(11) <= shift_mem(3) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(27) XOR shift_mem(28)
                        ;
173                 shift_mem(12) <= shift_mem(4) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(26) XOR shift_mem(28)
                         XOR shift_mem(29) XOR shift_mem(30);
174                 shift_mem(13) <= shift_mem(5) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(27) XOR shift_mem(29)
                         XOR shift_mem(30) XOR shift_mem(31);
175                 shift_mem(14) <= shift_mem(6) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(30)
                         XOR shift_mem(31);
176                 shift_mem(15) <= shift_mem(7) XOR shift_mem(27) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem(31)
                        ;
177                 shift_mem(16) <= shift_mem(8) XOR shift_mem(24) XOR
                        shift_mem(28) XOR shift_mem(29);
178                 shift_mem(17) <= shift_mem(9) XOR shift_mem(25) XOR
                        shift_mem(29) XOR shift_mem(30);
179                 shift_mem(18) <= shift_mem(10) XOR shift_mem(26) XOR
                         shift_mem(30) XOR shift_mem(31);
180                 shift_mem(19) <= shift_mem(11) XOR shift_mem(27) XOR
                         shift_mem(31);
181                 shift_mem(20) <= shift_mem(12) XOR shift_mem(28);
182                 shift_mem(21) <= shift_mem(13) XOR shift_mem(29);
183                 shift_mem(22) <= shift_mem(14) XOR shift_mem(24);
184                 shift_mem(23) <= shift_mem(15) XOR shift_mem(24) XOR
                         shift_mem(25) XOR shift_mem(30);
185                 shift_mem(24) <= shift_mem(16) XOR shift_mem(25) XOR
                         shift_mem(26) XOR shift_mem(31);
186                 shift_mem(25) <= shift_mem(17) XOR shift_mem(26) XOR
                         shift_mem(27);
```

```
187                 shift_mem(26) <= shift_mem(18) XOR shift_mem(24) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem
                        (30);
188                 shift_mem(27) <= shift_mem(19) XOR shift_mem(25) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem
                        (31);
189                 shift_mem(28) <= shift_mem(20) XOR shift_mem(26) XOR
                        shift_mem(29) XOR shift_mem(30);
190                 shift_mem(29) <= shift_mem(21) XOR shift_mem(27) XOR
                        shift_mem(30) XOR shift_mem(31);
191                 shift_mem(30) <= shift_mem(22) XOR shift_mem(28) XOR
                        shift_mem(31);
192                 shift_mem(31) <= shift_mem(23) XOR shift_mem(29);
193
194             ELSE
195                 check_data <= DATA_IN;
196                 byte_count <= byte_count + 1;
197                 shift_mem(0) <= shift_mem(24) XOR shift_mem(30) XOR
                        DATA_IN(7);
198                 shift_mem(1) <= shift_mem(24) XOR shift_mem(25) XOR
                        shift_mem(30) XOR shift_mem(31) XOR DATA_IN(6);
199                 shift_mem(2) <= shift_mem(24) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(30) XOR shift_mem(31)
                        XOR DATA_IN(5);
200                 shift_mem(3) <= shift_mem(25) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(31) XOR DATA_IN(4);
201                 shift_mem(4) <= shift_mem(24) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(30)
                        XOR DATA_IN(3);
202                 shift_mem(5) <= shift_mem(24) XOR shift_mem(25) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(29)
                        XOR shift_mem(30) XOR shift_mem(31) XOR DATA_IN
                        (2);
203                 shift_mem(6) <= shift_mem(25) XOR shift_mem(26) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem(30)
                        XOR shift_mem(31) XOR DATA_IN(1);
204                 shift_mem(7) <= shift_mem(24) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(29) XOR shift_mem(31)
                        XOR DATA_IN(0);
205                 shift_mem(8) <= shift_mem(0) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(27) XOR shift_mem(28)
                        ;
206                 shift_mem(9) <= shift_mem(1) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(28) XOR shift_mem(29)
                        ;
```

```
207                 shift_mem(10) <= shift_mem(2) XOR shift_mem(24) XOR
                        shift_mem(26) XOR shift_mem(27) XOR shift_mem(29)
                        ;
208                 shift_mem(11) <= shift_mem(3) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(27) XOR shift_mem(28)
                        ;
209                 shift_mem(12) <= shift_mem(4) XOR shift_mem(24) XOR
                        shift_mem(25) XOR shift_mem(26) XOR shift_mem(28)
                         XOR shift_mem(29) XOR shift_mem(30);
210                 shift_mem(13) <= shift_mem(5) XOR shift_mem(25) XOR
                        shift_mem(26) XOR shift_mem(27) XOR shift_mem(29)
                         XOR shift_mem(30) XOR shift_mem(31);
211                 shift_mem(14) <= shift_mem(6) XOR shift_mem(26) XOR
                        shift_mem(27) XOR shift_mem(28) XOR shift_mem(30)
                         XOR shift_mem(31);
212                 shift_mem(15) <= shift_mem(7) XOR shift_mem(27) XOR
                        shift_mem(28) XOR shift_mem(29) XOR shift_mem(31)
                        ;
213                 shift_mem(16) <= shift_mem(8) XOR shift_mem(24) XOR
                        shift_mem(28) XOR shift_mem(29);
214                 shift_mem(17) <= shift_mem(9) XOR shift_mem(25) XOR
                        shift_mem(29) XOR shift_mem(30);
215                 shift_mem(18) <= shift_mem(10) XOR shift_mem(26) XOR
                         shift_mem(30) XOR shift_mem(31);
216                 shift_mem(19) <= shift_mem(11) XOR shift_mem(27) XOR
                         shift_mem(31);
217                 shift_mem(20) <= shift_mem(12) XOR shift_mem(28);
218                 shift_mem(21) <= shift_mem(13) XOR shift_mem(29);
219                 shift_mem(22) <= shift_mem(14) XOR shift_mem(24);
220                 shift_mem(23) <= shift_mem(15) XOR shift_mem(24) XOR
                         shift_mem(25) XOR shift_mem(30);
221                 shift_mem(24) <= shift_mem(16) XOR shift_mem(25) XOR
                         shift_mem(26) XOR shift_mem(31);
222                 shift_mem(25) <= shift_mem(17) XOR shift_mem(26) XOR
                         shift_mem(27);
223                 shift_mem(26) <= shift_mem(18) XOR shift_mem(24) XOR
                         shift_mem(27) XOR shift_mem(28) XOR shift_mem
                        (30);
224                 shift_mem(27) <= shift_mem(19) XOR shift_mem(25) XOR
                         shift_mem(28) XOR shift_mem(29) XOR shift_mem
                        (31);
225                 shift_mem(28) <= shift_mem(20) XOR shift_mem(26) XOR
                         shift_mem(29) XOR shift_mem(30);
226                 shift_mem(29) <= shift_mem(21) XOR shift_mem(27) XOR
                         shift_mem(30) XOR shift_mem(31);
```

```
227                         shift_mem(30) <= shift_mem(22) XOR shift_mem(28) XOR
                                shift_mem(31);
228                         shift_mem(31) <= shift_mem(23) XOR shift_mem(29);
229                     END IF;
230
231                 WHEN CHECK_FCS =>
232                     IF shift_mem = (shift_mem'RANGE => '0') THEN
233                         FCS_ERROR <= '0';
234                     ELSE
235                         FCS_ERROR <= '1';
236                     END IF;
237
238                 WHEN error_check =>
239                     FCS_ERROR <= '0'; -- Reset for next frame
240
241                 WHEN OTHERS =>
242                     NULL;
243             END CASE;
244         END IF;
245     END PROCESS;
246 END ARCHITECTURE;
```

4.7 Parallel Implementation - Testbench (fcs_parallel_sim.vhd)

```
1  //Author Daniel Duggan
2
3  module FCS_para;
4
5    timeunit 1ns;
6    timeprecision 1ps; //time precision specifies how delay values are rounded
         relative to timeunit
7
8    logic sysclk_p;
9    logic sysclk_n;
10   logic rst; //reset logic
11   logic start_of_frame_top, end_of_frame_top, fcs_error_top;
12   logic [7:0] data_in;
13   parameter clk_period = 4ns; // Since sys_clock every 2ns
14
15   // clock generation
        ------------------------------------------------------------------
16   logic sys_clk = 0;
17   always #2ns sys_clk = ~sys_clk; // Toggle every 2 ns (1 cycle = 4 ns)
18   assign sysclk_p = sys_clk;
```

```
19  assign sysclk_n = ~sys_clk;
20  //
        ------------------------------------------------------------------------------------

21
22  // Data in (Ethernet Frame)
        ------------------------------------------------------------
23  logic [511:0] ethernet_frame = {
24          128'h00_10_A4_7B_EA_80_00_12_34_56_78_90_08_00_45_00,
25          128'h00_2E_B3_FE_00_00_80_11_05_40_C0_A8_00_2C_C0_A8,
26          128'h00_04_04_00_04_00_00_1A_2D_E8_00_01_02_03_04_05,
27          128'h06_07_08_09_0A_0B_0C_0D_0E_0F_10_11_E6_C5_3D_B2
28        };
29
30  logic [511:0] ethernet_frame_incorrect = {
31          128'h00_10_A4_7B_EA_80_00_12_34_56_78_90_08_00_45_00,
32          128'h00_2E_B3_FE_00_00_80_11_05_40_C0_A8_00_2C_C0_A8,
33          128'h00_04_04_00_04_00_00_1A_2D_E8_00_01_02_03_04_05,
34          128'h06_07_08_09_0A_0B_0C_0D_0E_0F_10_11_FF_FF_FF_FF
35        };
36
37  //
        ------------------------------------------------------------------------------------

38
39  // DUT
        ------------------------------------------------------------------------------------

40  Sys_Top_para Sys_Top_para_DUT(
41            .SYS_CLOCK_P (sysclk_p),
42            .SYS_CLOCK_N (sysclk_n),
43            .RST (rst),
44            .START_OF_FRAME_top (start_of_frame_top),
45            .END_OF_FRAME_top (end_of_frame_top),
46            .DATA_IN_top (data_in),
47            .FCS_ERROR_top (fcs_error_top)
48          );
49  //
        ------------------------------------------------------------------------------------

50
51  initial
52  begin
53    // Initialize signals
54    start_of_frame_top = 0;
```

```
55      end_of_frame_top    = 0;
56      data_in             = 0;
57      rst = 1'b0;
58
59      // Apply Reset
60      repeat (5) @(posedge sys_clk);
61      rst = 1'b1;
62      repeat (5) @(posedge sys_clk);
63
64      // Begin Transmission
65
66      @(posedge sys_clk);
67      start_of_frame_top = 1'b1;  // Assert start_of_frame_top one cycle
            before the first byte
68
69      for (int i = ($bits(ethernet_frame)/8) - 1; i >= 0; i--)
70      begin
71        @(posedge sys_clk);
72        start_of_frame_top = 1'b0;  // Deassert start_of_frame_top after the
              first cycle
73
74        data_in = ethernet_frame[((i+1)*8 - 1) -: 8];  // Send data in bytes
75
76        if (i == 3) // Indicate the last 3 bytes
77          end_of_frame_top = 1'b1;
78        else
79          end_of_frame_top = 1'b0;
80      end
81
82      @(posedge sys_clk);
83      end_of_frame_top = 1'b0;  // Deassert end_of_frame_top after the last
            byte
84
85      repeat (3) @(posedge sys_clk);
86      $finish;
87    end
88  endmodule
```

4.8 Serial Implementation - Constraints Files

```
1  set_property -dict {PACKAGE_PIN H13 IOSTANDARD LVCMOS33} [get_ports {
       DATA_IN_top}]
2  set_property -dict {PACKAGE_PIN F20 IOSTANDARD LVCMOS33} [get_ports {
       START_OF_FRAME_top}]
```

```
set_property -dict {PACKAGE_PIN E20 IOSTANDARD LVCMOS33} [get_ports {END_OF_FRAME_top}]
set_property -dict {PACKAGE_PIN G20 IOSTANDARD LVCMOS33} [get_ports {RST}]
set_property -dict {PACKAGE_PIN D22 IOSTANDARD LVCMOS33} [get_ports {FCS_ERROR_top}]

## External clock input

# Create a clock constraint using the P-side of the differential pair
create_clock -name sys_clk -period 4 [get_ports SYS_CLOCK_P]

# Define the differential clock input pins, -dict combines properties into single command
set_property -dict {PACKAGE_PIN AL8 IOSTANDARD LVDS} [get_ports SYS_CLOCK_P]
set_property -dict {PACKAGE_PIN AL9 IOSTANDARD LVDS} [get_ports SYS_CLOCK_N]
```

4.9 Parallel Implementation - Constraints Files

```
set_property -dict {PACKAGE_PIN G13 IOSTANDARD LVCMOS33} [get_ports {DATA_IN_top[0]}]
set_property -dict {PACKAGE_PIN H16 IOSTANDARD LVCMOS33} [get_ports {DATA_IN_top[1]}]
set_property -dict {PACKAGE_PIN G16 IOSTANDARD LVCMOS33} [get_ports {DATA_IN_top[2]}]
set_property -dict {PACKAGE_PIN J16 IOSTANDARD LVCMOS33} [get_ports {DATA_IN_top[3]}]
set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports {DATA_IN_top[4]}]
set_property -dict {PACKAGE_PIN G15 IOSTANDARD LVCMOS33} [get_ports {DATA_IN_top[5]}]
set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports {DATA_IN_top[6]}]
set_property -dict {PACKAGE_PIN J14 IOSTANDARD LVCMOS33} [get_ports {DATA_IN_top[7]}]
set_property -dict {PACKAGE_PIN F20 IOSTANDARD LVCMOS33} [get_ports {START_OF_FRAME_top}]
set_property -dict {PACKAGE_PIN E20 IOSTANDARD LVCMOS33} [get_ports {END_OF_FRAME_top}]
set_property -dict {PACKAGE_PIN G20 IOSTANDARD LVCMOS33} [get_ports {RST}]
set_property -dict {PACKAGE_PIN D22 IOSTANDARD LVCMOS33} [get_ports {FCS_ERROR_top}]
## External clock input

# Create a clock constraint using the P-side of the differential pair
```

```
16 create_clock -name sys_clk -period 4 [get_ports SYS_CLOCK_P]
17
18 # Define the differential clock input pins, -dict combines properties into
      single command
19 set_property -dict {PACKAGE_PIN AL8 IOSTANDARD LVDS} [get_ports SYS_CLOCK_P]
20 set_property -dict {PACKAGE_PIN AL9 IOSTANDARD LVDS} [get_ports SYS_CLOCK_N]
```