EECE 359 – Computer Communications and Networking Spring 2020

Lab #6 – Error Detection Performance of CRC Codes (Parts 2 and 3)

Objectives

This lab exercise continues the multi-part lab that began in the previous lab exercise (Part 1). Recall that the objectives are

1. to develop a system for evaluating the performance of CRC error detection,
2. to implement the system in hardware using the FPGA board, and
3. to use the system to study the performance of CRC error detection.

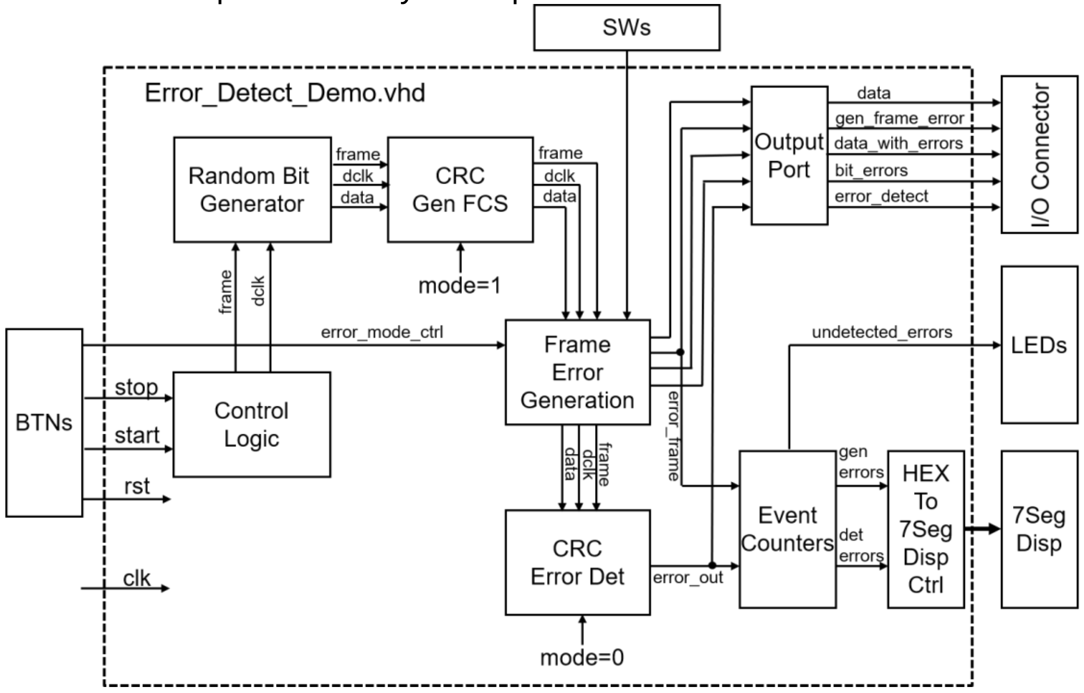
In Part 1 of the lab exercise, you developed a *pseudorandom bit generator* to be used to generate data frames with randomized data for evaluating the performance of CRC error detection as well as to simulate random error patterns in the data frame.

In Part 2 of the lab, you will develop a module used to inject simulated bit errors into some of the data frames and you will use the simulator to verify its operation. Two methods will be used to inject patterns: (1) a fixed programmable error pattern, and (2) a random error pattern.

In Part 3 of the lab, you will implement the “error detect demo” system on your FPGA board and then use the system to study the effectiveness and limitations of CRC error detection. First, you will implement your VHDL code on the FPGA board and verify the operation of the “error detect demo” system. You will then observe the frames with errors on the oscilloscope and whether or not the errors were detected. You will draw conclusions on the performance of the CRC and document them in the lab assignment.

System Description

The block diagram below is a high-level view of the overall system. Refer to the Part 1 procedure for the description of the system operation.



Design Hierarchy/Files

For the entire evaluation system, the hierarchy will look like the following:

Test_Error_Detect_Demo – Behavior (Test_Error_Detect_Demo.vhd)

uut – Error_Detect_Demo – Behavioral (Error_Detect_Demo.vhd)

HEXon7segDispA – HEXon7segDisp – Behavioral (HEXon7segDisp.vhd)

RST_debounce – debounce – Behavioral (debounce.vhd)

Start_debounce – debounce – Behavioral (debounce.vhd)

Stop_debounce – debounce – Behavioral (debounce.vhd)

Control – Control_Logic – Behavioral (Control_Logic.vhd)

RandBitGenA – RandBitGen – Behavioral (RandBitGen.vhd)

FCS_Gen – CRC – Behavioral (CRC.vhd)

Frame_Error – Frame_Error_Gen – Behavioral (Frame_Error_Gen.vhd)

RandBitGenB – RandBitGen – Behavioral(RandBitGen.vhd)

Error_Detect – CRC – Behavioral (CRC.vhd)

Counters – Event_Counters – Behavioral (Event_Counters.vhd)

Output_Connector – Output_Port – Behavioral (Output_Port.vhd)

Lab6.ucf

In Part 1 of this lab, you developed the random bit generator, **RandBitGen**.

The **FCS_Gen** and **Error_Detect** functions are provided by your **CRC** module from Lab 4.

In Part 2 of this lab, you will develop the **Frame Error Generator** for which a template is provided (Frame_Error_Gen.vhd). The remaining files are provided.

Procedure:

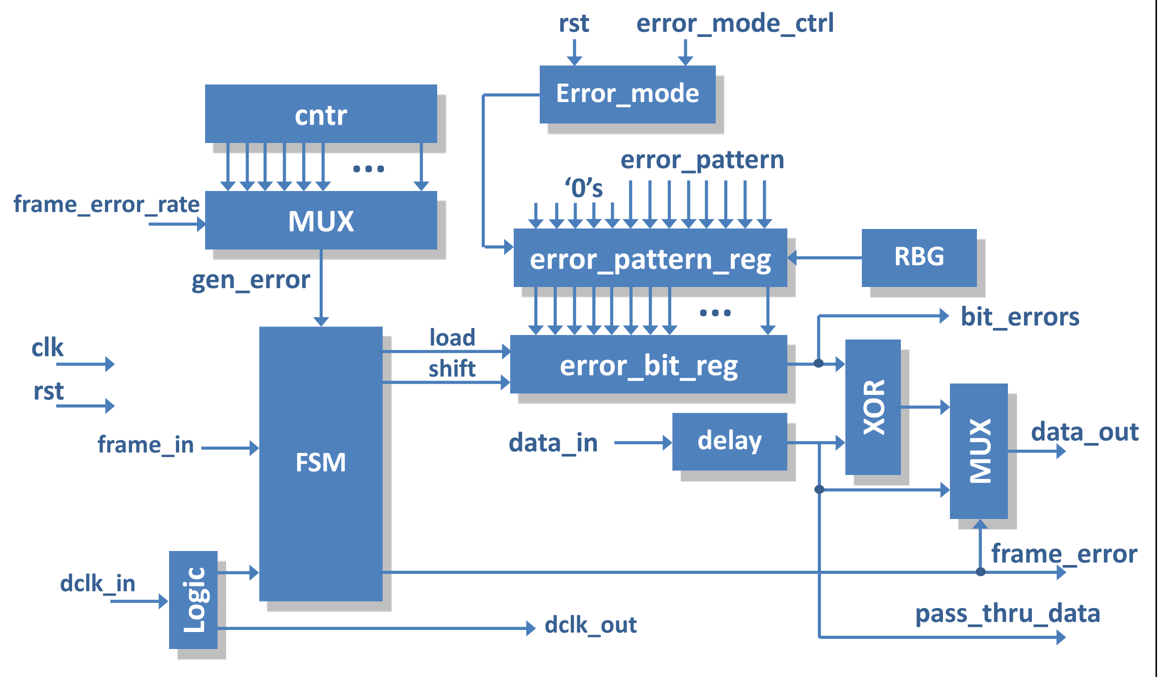
Part 2 – Frame Error Generator Design and Simulation

☐ **Complete the design of the Frame Error Generator**

1. In the ISE Project Navigator, make a copy of your Lab5 project and name it Lab6
2. Add the provided template, *Frame_Error_Gen.vhd*, as a source file to your project
3. Add your *CRC.vhd* source file from Lab 4
4. Add the remaining provided source files for this lab, including the test bench (associate the test bench only with simulation in its source properties)
5. Develop your design for the Frame Error Generator function in *Frame_Error_Gen.vhd*

   The purpose of this function is to simulate bit errors that would be encountered when transmitting data through a transmission medium. The 8 board switches are provided as an input (called error_pattern) to this function. You will use these inputs to provide a fixed error pattern. The random bit generator that you developed in the previous lab is provided as a component in this function as well. It will be used to generate a random bit pattern. The means for selecting one of the two error modes is provided by the “rst” and “error_mode_ctrl” inputs from momentary switches. The “error_mode_ctrl” puts the Frame Error Generator into the random bit pattern mode. “rst” returns it to the fixed error pattern mode.

The block diagram below shows the general functionality of the frame error generator module.



Frame Error Generator Block Diagram

It is desired to generate more than a single bit error in a frame since we know that the CRC will always detect single bit errors. So, we would like to simulate some form of burst errors. Both of the two methods can generate multiple bit errors.

The “frame_length” is passed to this module as a generic parameter. This parameter is used to set the size of the error bit frames generated in the demo to match the size of the data frames.

“data_in”, “dclk_in”, and “frame_in” are input signals that come from the CRC module generating and adding the FCS to each data frame. They connect to “data_out”, “dclk_out”, and “frame_out” of the FCS generator, respectively.

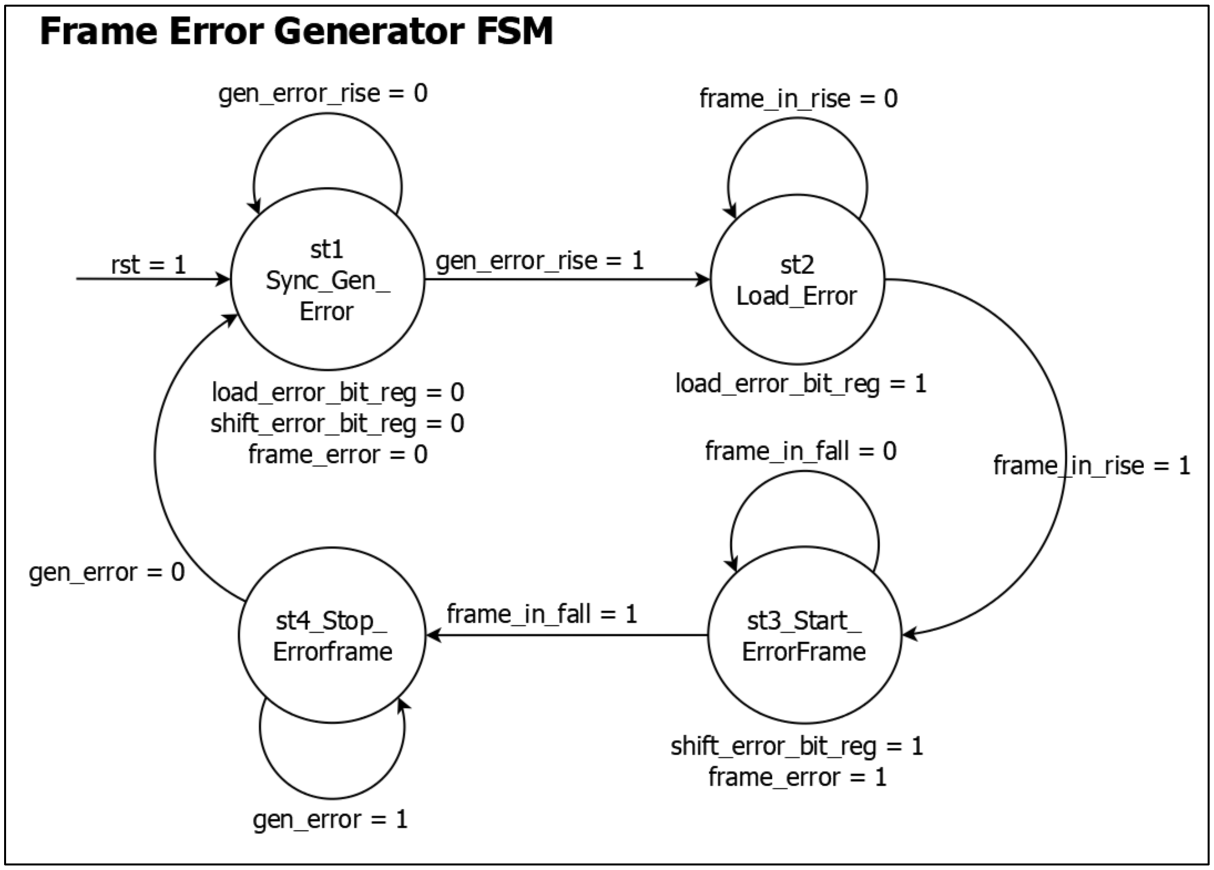
“data_out” of the Frame Error Generator is the data with the errors injected from the error pattern register. “frame_out” is a signal that frames the data. “dclk_out” provides the clock for the output data.

“frame_error” is a frame signal that is only generated for the frames that have errors. The signal “bit_errors” shows the position of the bit errors in the corresponding bits of data in “data_out”. The signal “pass_thru_data” is simply the input data passed through without inserting errors. These signals will be used to observe the operation of the demo on the oscilloscope as well as in the simulator.

In the frame error generator module, a counter is provided that will be used to set the frequency at which error frames are inserted into the data stream. *frame_error_rate* is a parameter passed from the top level as an input to set the rate of error frames. A process for decoding *frame_error_rate* is provided in the code template. Using a MUX, it will select the respective bit from the counter to be used to set the frame error rate. Take a look at the VHDL code to see how this works.

A finite state machine will be used to handle the insertion of the error pattern. The FSM controls the loading of the bit error pattern into the *error_bit_reg*, and shifting of the error pattern out of the *error_bit_reg* in synchronism with the *data frame* and the *dclk* signals. The state diagram for the FSM is shown below.

Some of the VHDL code for this module is provided in the template. You are to complete the code for the *error_bit_reg*, the *error_pattern_reg*, the logic to generate *bit_errors* and *data_out*, and the *FSM*.



- **Use simulation to verify correct operation of the FEG with your code for generating the error pattern**

  Use the provided test bench (Test_Error_Detect_Demo.vhd)

  In the top-level module, Error_Detect_Demo.vhd, the *frame_error_rate* is set to “1111” by the test bench using this generic parameter. This sets the error frame generator for the maximum error rate to minimize the time that simulation needs to run.

  In the top-level module, the *FCS_length* constant should be set to 5 and the value of the CRC predetermined divisor, *P*, should be “110101”. If not, assign these values.

  Run the simulation for <u>800 microseconds</u>.

  Determine correct operation by viewing the waveforms on the “Output Port”. This is a group of signals with names beginning with “testsig_”.

  Verify correct operation of the Frame Error Generator module by viewing the waveforms for at least the following signals:

error_pattern	data_out	rbg_out
error_mode	dclk_out	error_bit_reg
data_in	frame_out	load_error_bit_reg
dclk_in	frame_error	shift_error_bit_reg
frame_in	gen_error	frame_errror_i
bit_errors	error_pattern_reg	state

  View the Console output of iSim. You should see the following output:

```
# run 800 us
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 22100 ns: Note: New Error Pattern (/test_error_detect_demo/).
at 22200 ns: Note: Start Button Pressed (/test_error_detect_demo/).
at 83678125 ps: Note: New Error Pattern (/test_error_detect_demo/).
at 83778125 ps, Instance /test_error_detect_demo/ : Warning: Error Detected
at 221678125 ps: Note: New Error Pattern (/test_error_detect_demo/).
at 221778125 ps, Instance /test_error_detect_demo/ : Warning: Error Detected
at 336678125 ps: Note: New Error Pattern (/test_error_detect_demo/).
at 336778125 ps, Instance /test_error_detect_demo/ : Warning: Error Not Detected
at 474678125 ps: Note: New Error Pattern (/test_error_detect_demo/).
at 474778125 ps, Instance /test_error_detect_demo/ : Warning: Error Detected*
at 612678125 ps: Note: New Error Pattern (/test_error_detect_demo/).
at 612778125 ps, Instance /test_error_detect_demo/ : Warning: Error Detected*
at 727778125 ps, Instance /test_error_detect_demo/ : Warning: Error Not Detected*
at 747678125 ps: Note: Stop Button Pressed (/test_error_detect_demo/).

ISim>
```

  ***Your messages may vary depending upon your specific random numbers**

  The test bench monitors the progress of the simulation and issues “Note” messages for button presses and new error patterns. It monitors the “testsig_error_detect” signal and gives a “warning message” indicating whether an error was detected for the current error pattern, or not.

☐ **Demonstrate your simulation results**

Demonstrate your simulation results to a lab proctor and get checked off.

1. Show your simulation waveforms as described above.
2. Show that the simulator console is giving the correct sequence of messages.

☐ **Capture a PDF of the waveforms for your simulation**

Include in your displayed waveforms:

- Top level outputs:
  - testsig_data
  - testsig_error_frame
  - testsig_data_with_errors
  - testsig_error_detect
  - testsig_bit_errors
- The Frame_Error_Gen signals listed on the previous page
- Zoom-in to the time interval of 200 µs to 350 µs

Submit your waveform PDF in the lab assignment.

Part 3 – Hardware Implementation and CRC Error Detection Characterization

☐ **Set the design parameters for hardware implementation**

In the top-level module, Error_Detect_Demo.vhd, the frame_error_rate constant is set to the generic default value of “0100” which will set the frame error rate to a few error frames per second.

Keep the current values for FCS_length and P used in simulation for a 5-bit FCS.

☐ **Synthesize the FPGA design**

Synthesize the FPGA and review the error and warning messages. A list of expected messages is provided in a separate HTML file.

☐ **Generate a bit file for the Error_Detect_Demo and download it to your FPGA Board**

☐ **Verify the operation of the Error Detect Demo**

In the initial mode, the demo will operate with error patterns generated from the 8 slide switches on the board.

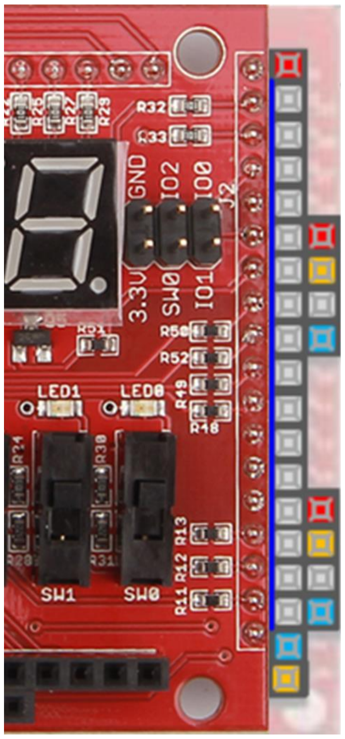
After pressing the “start” button (btn1), the 7-segment display should be showing the error frame count (left 2 digits) and the count of frames with errors detected by the CRC. The LEDs will display the difference between the two counts, or the number of undetected error frames.

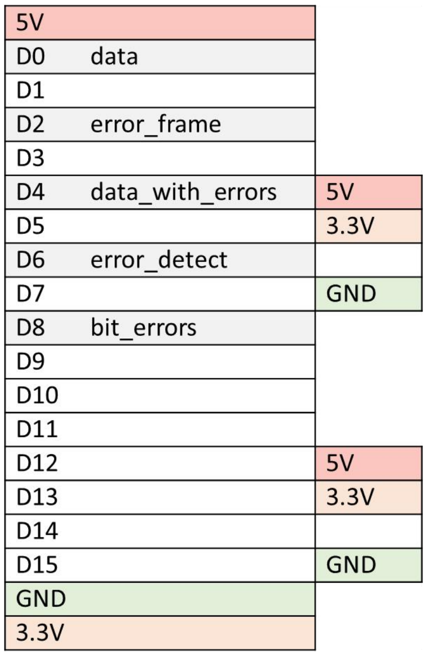
Press the mode control button (btn3) to change the error pattern generation to the random bit mode.

☐ **Use the oscilloscope to observe the data with and without errors**

Use the signal “testsig_error_frame” to trigger the oscilloscope using its external trigger. The trigger input is located on the scope’s rear panel. Select this source in the trigger source menu. Triggering on this signal will allow you to easily observe only the frames with errors on the scope.

Observe the signals “testsig_data”, “testsig_data_with_errors”, “testsig_error_detect” and “testsig_bit_errors” on the four channels of the scope.





5V		
D0	data	
D1		
D2	error_frame	
D3		
D4	data_with_errors	5V
D5		3.3V
D6	error_detect	
D7		GND
D8	bit_errors	
D9		
D10		
D11		
D12		5V
D13		3.3V
D14		
D15		GND
GND		
3.3V		

☐ **Demonstrate your results**

<u>Demonstrate your results to a lab proctor and get checked off.</u>

1. Show the operation of your system with an error pattern that is detected
2. Show the operation with an error pattern that is not detected
3. Show the operation with the random error pattern generator
4. Using the random error pattern generator, show the error frames with both the data before and after errors are injected using the oscilloscope.

☐ **Experiment with the board and draw your own conclusions about the performance of CRC error detection**

As part of your experimentation, change the FCS length and P parameters to those for a 4-bit FCS and compare the results you see with the 5-bit FCS.

Run using both the fixed error pattern and random error patterns. With the latter, you can compare performance by running the demo for a specified time for each FCS length and comparing the number of undetected error frames.

For one of the FCS sizes, change the length of data frames from 10 bits to 20 bits and compare the detection performance to determine if frame length is a performance factor.

(Hint: save the programming (bit) files to different file names so that you can reload them without taking the time to resynthesize them over again.)

☐ **Write up your conclusions about CRC error detection performance**

Write up you observations and conclusions and include the following as a minimum:

- Describe your specific experiments and observations
- Describe the effects of the FCS length on error detection performance
- List the undetected error patterns for both the 4-bit FCS and 5-bit FCS. (You can use Lab 4 Test Bench 2 with k=3, fcs_length=5 or k=4, fcs_length=4 to generate the list of undetectable error patterns)
- Describe the effect of frame length on the CRC performance
- Summarize the burst error detection performance for a given FCS length in terms of burst size and probabilities.

Submit your write-up in the lab assignment.

☐ **Submit your VHDL code**

Submit your Frame_Error_Gen.vhd file in the lab assignment.