EECE 359 – Computer Communications and Networking **Spring 2020**

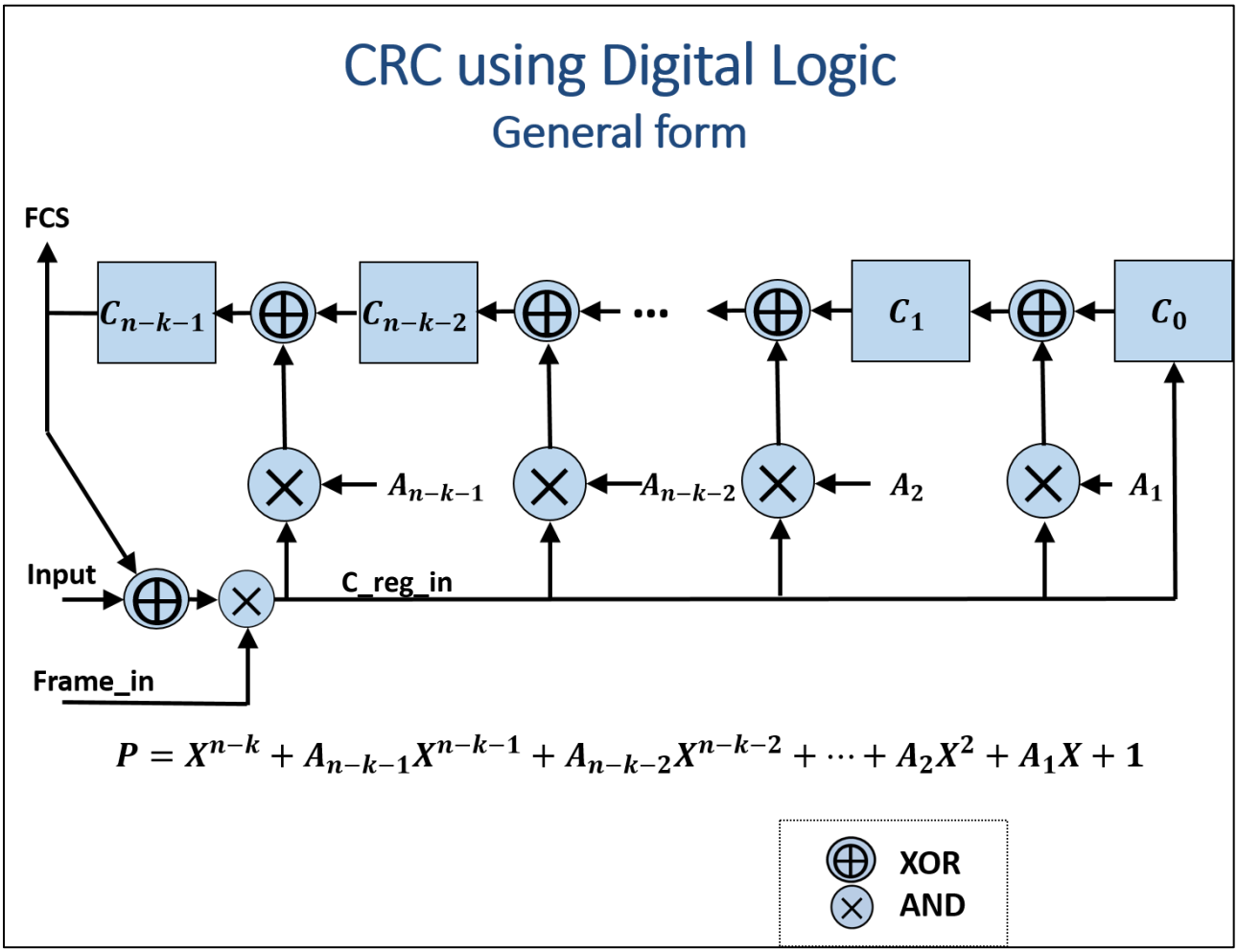
Lab #4 – Cyclic Redundancy Check (CRC) Error Detection Codes

Objectives

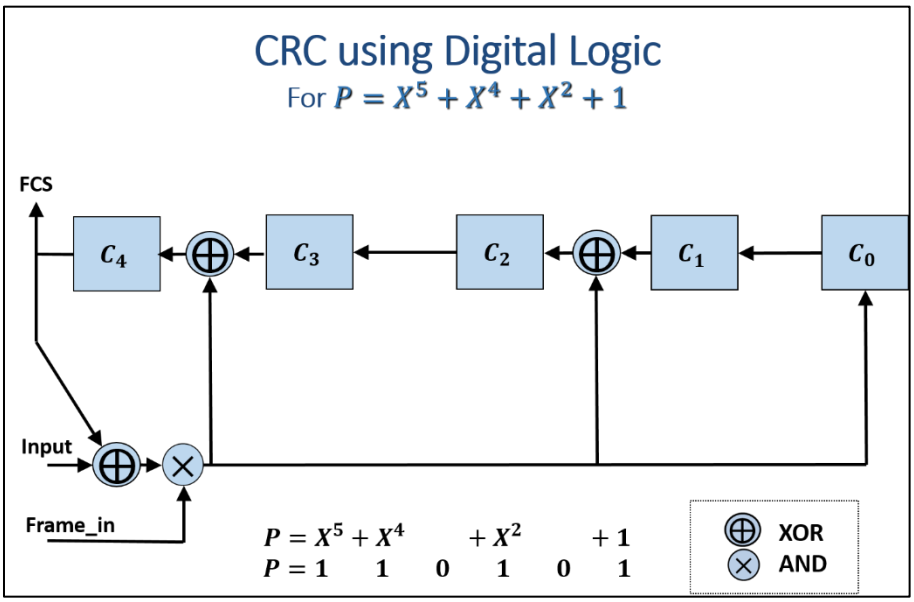
The objective of this lab is to develop an understanding of the Cyclic Redundancy Check (CRC) code that can be used for error detection in data communications. This will be accomplished by implementing the CRC code in digital logic. The operation of the CRC code will be verified and demonstrated using simulation. A further objective of this lab is to apply methods in VHDL for specifying digital designs to be flexible and scalable. That is, we want to put a design in a general form that can adapt to the specific requirements of an application without code changes. In the design of the CRC, we want the FCS length and the predetermined divisor to be generic input parameters. These generic parameters are specified in the CRC entity and then used in the architecture to specify the size of the related elements in the behavioral.

System Description

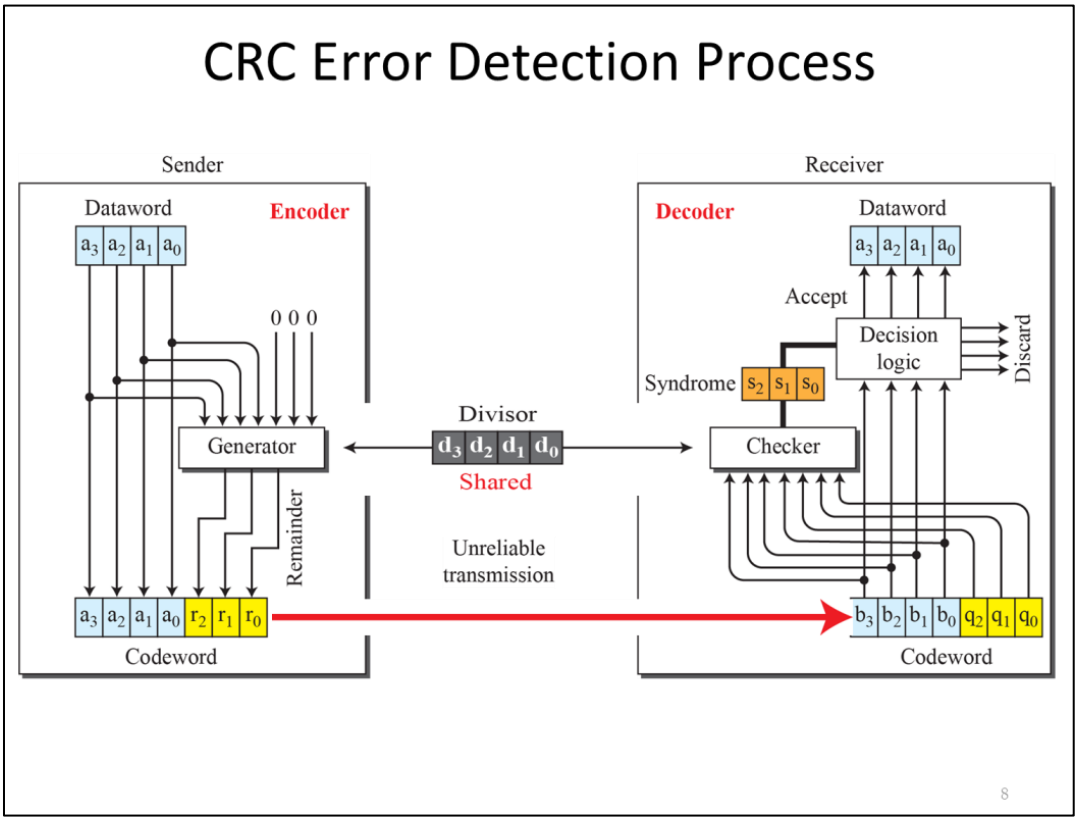
The general form of the architecture for the CRC digital logic implementation presented in the lecture is shown in the figure below.



The specific example presented in the lecture will be used to perform an initial verification of the CRC code. That example is shown below.



The VHDL code for the CRC module shall be written such that it can be used for both the code *generator* and code *checker*. The figure below shows the functionality that is to be implemented. However, because of the serial nature of the CRC digital logic implementation, it will not be necessary to use the registers that are shown for transmitting and receiving the data. In other words, the data will be transmitted as it is serially applied to the CRC generator, with the check bits (FCS) transmitted immediately after the final data bit. Then on the receiving end, the data will be immediately shifted into the CRC checker as each bit is received.



Design/Simulation Hierarchy and Files

Test1_CRC – behavior (Test1_CRC.vhd)

uut – CRC – Behavioral (CRC.vhd)

Test2_CRC – behavior (Test2_CRC.vhd)

uut – CRC – Behavioral (CRC.vhd)

The Test1_CRC.vhd, Test2_CRC.vhd, and CRC.vhd files are provided. The CRC.vhd file is a template for the CRC code design. You will be designing the VHDL code for each of the provided processes in the template. The Test1_CRC.vhd file provides test sequences for a CRC generating a 5-bit FCS:

1. A 10-bit data frame from the example in the lecture with the CRC mode set for generation of the FCS
2. The 15-bit data frame generated by the CRC in #1 with mode set for error detection
3. The same 15-bit data frame, but with one bit flipped to simulate a bit error with mode still set for error detection
4. A different 10-bit data frame with mode set for FCS generation
5. The 15-bit data frame generated by the CRC in #4 with mode set for error detection
6. The same 15-bit data frame, but with an undetectable burst error applied with mode still set for error detection
7. The same 15-bit data frame, but with a detectable burst error applied with mode still set for error detection

The Test2_CRC.vhd file provides test sequences for a CRC generating a 4-bit FCS. The test bench includes the following test functions:

1. Generates a series of 5-bit data frames of random data
2. Each data frame is passed through the CRC with the mode set for generation of the FCS
3. Captures the output of the CRC (5 bits of data plus 4 bits of FCS) to use as input to CRC
4. Applies the 9-bit frame to the CRC with mode set to error detection and checks that no errors are detected
5. Introduces an error pattern to the 9-bit frame and reapply the frame to the CRC in error detection mode
6. All possible error patterns are tested (511 patterns)
7. Non-detected error patterns are captured in a file, *CRC_undetected_errors.txt*

Procedure:

Design the CRC

1. **Create a project named “Lab4” in the ISE Project Navigator**
2. **Add the Test1_CRC.vhd, Test2_CRC.vhd, and CRC.vhd source files to your Lab4 project**

3. **Review the port map of the CRC entity in the CRC.vhd file and review the I/O definition comments that are provided so that you understand the intended operation of the module.**

4. **Review the signals and constants that are declared in the template**

   Notice that a generic parameter is used to provide the FCS length to the CRC module. This gives the CRC module the flexibility to be "scaled" according to the CRC code requirements of a given application when the module is synthesized. The FCS length value specifies the number of registers needed to form the C-register. It also specifies the number of bits required for the predetermined divisor (FCS length + 1).

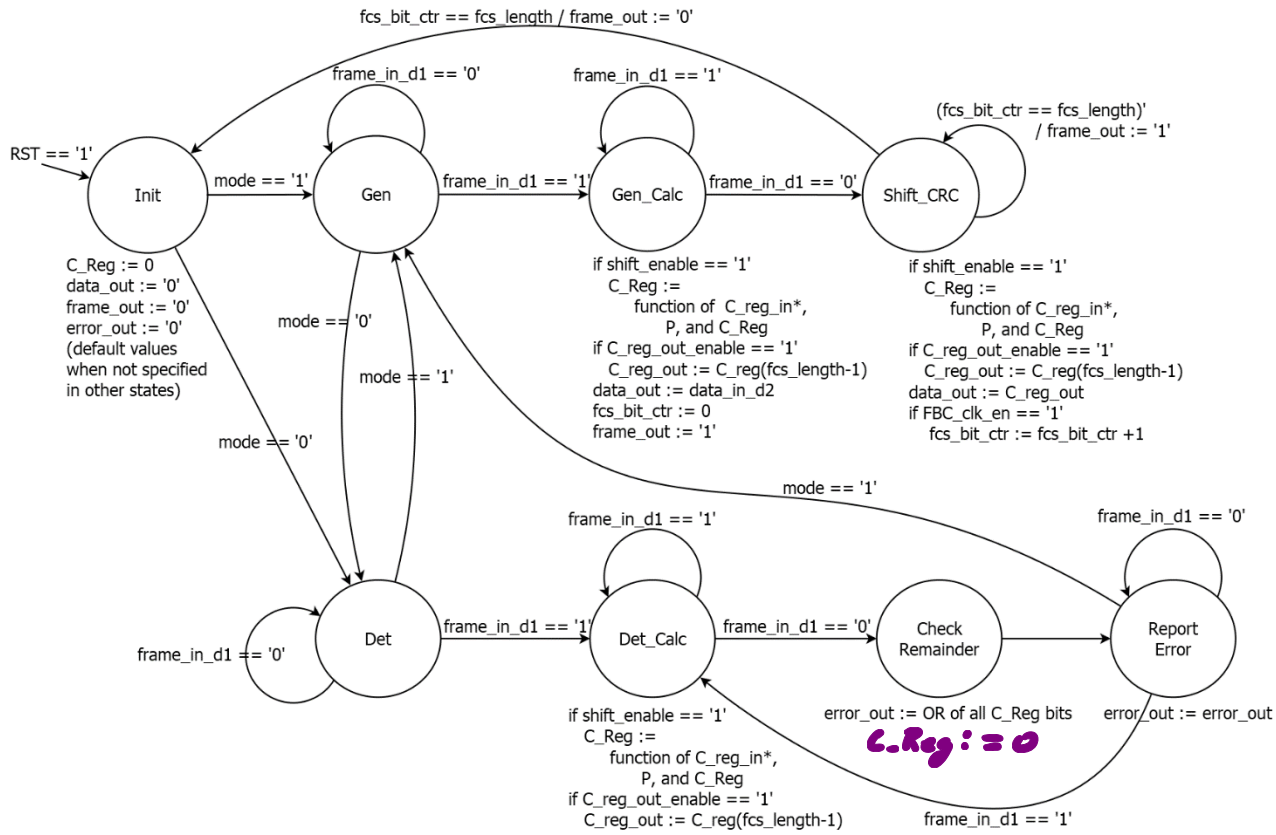
   The value of the predetermined divisor is provided as an input port to the CRC module. As long as we implement the general form of the CRC algorithm, it is not necessary to re-synthesize the CRC module when the value of P is changed. The P input is simply a constant value used to configure the CRC to operate with the particular predetermined divisor.

5. **Complete the design of the CRC module.**

   The high-level state machine (HLSM) diagram below describes the CRC design.

CRC HLSM

HLSM to control CRC generation and error detection
Inputs: fcs_length (generic integer parameter)
P (fcs_length+1 bits),
mode, frame_in_d1, data_in_d1, data_in_d2, RST, shift_enable, FBC_clk_en, C_reg_out_enable (bit)
Outputs: data_out, frame_out, error_out(bit)
Internal Storage: fcs_bit_ctr(6 bits), C_Reg(fcs_length-1 bits), C_reg_out (bit), error_out (bit)



*C_reg_in = function of data_in_d1, frame_in_d1, and C_reg(fcs_length-1)

Using the HLSM and following the descriptions in the comments provided in the CRC code template file, write the VHDL code for the CRC module, which will contain the code for both the datapath and the controller (FSM). The details of the C_Reg operation have been left for you to determine from the CRC block diagram provided above that shows the general form for the CRC.

Sketch the datapath components as needed to understand its operation and connections needed to the controller. Draw a diagram of the controller FSM. A template for the FSM is provided at the end of this document.

Write the VHDL code for each of the datapath components (a process statement or signal assignment statement for each one) as well as the VHDL code for the controller FSM.

Submit an image of your FSM diagram to the lab assignment (preferred) or a hardcopy to the TA.

6. Synthesize the design and fix any errors

Run synthesis and review all the error and warning messages. You may get warnings that the P<0> and/or P<5> are never used. That is because the MSB of the predetermined divisor is not used in the CRC and the LSB is assumed to be a '1'.

	Synthesis Messages - Errors, Warnings, and Infos	New
WARNING	Xst:647 - Input <P<5>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.	New
WARNING	Xst:647 - Input <P<0>> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.	New

You must address all issues causing other error and warning messages.

CRC Test #1

1. Observe and debug the CRC operation using the simulator with the provided test bench, Test1_CRC.vhd

Set the constants clk_period and dclk_period in the test bench to the values appropriate for your FPGA board.

Run the simulation for 150 microseconds.

Group the test bench signals.

Add the internal signals of the CRC module to the waveform view, group them, and save your waveform configuration.

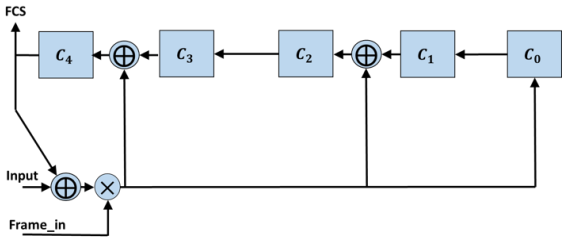
2. Zoom into the C_Reg for the first data frame and check the data for each step of the process of generating the FCS

What is the size (number of bits) of C_Reg? _____

Compare the C_Reg contents for each step of the process of generating the FCS to the values in the figure below:

CRC using Digital Logic

$P = X^5 + X^4 + X^2 + 1$



	C_4	C_3	C_2	C_1	C_0	I=input	$I \oplus C_4 \oplus C_3$	$I \oplus C_4 \oplus C_1$	$I \oplus C_4$
Initial	0	0	0	0	0	1	1	1	1
1	1	0	1	0	1	0	1	1	1
2	1	1	1	1	1	1	1	1	0
3	1	1	1	1	0	0	0	0	1
4	0	1	0	0	1	0	1	0	0
5	1	0	0	1	0	0	1	0	1
6	1	0	0	0	1	1	0	0	0
7	0	0	0	1	0	1	1	0	1
8	1	0	0	0	1	0	1	1	1
9	1	0	1	1	1	1	0	1	0
10	0	1	1	1	0				

5

(Hint: click on the C_Reg waveform and then use the left and right arrow keys to scroll through the values of the register, while viewing the "value" column to the right of the signal "name" column.)

Does your C_Reg contents match the figure? _____

What is the FCS that is generated? _____

What is the complete pattern of data that is output on "data_out" when the CRC is generating the FCS?

_____

3. Look at all seven data frames in the waveform viewer.

Does the error_out signal indicate that any of the frames has an error?_____

Identify all of the frames that have errors (1 – 7) _____

Are there any errors reported in the simulator console? _____

at 412500 ps: Note: CRC TEST 1 STARTED (/test1_crc/).

Any errors reported here?

at 142500 ns: Note: CRC TEST 1 COMPLETED (/test1_crc/).

ISim>

If there are, determine the problem and fix your CRC.

4. **<u>Submit a PDF of your simulation waveforms</u>**

   Your waveforms should show all seven data frames.

   Expand both the Test Bench and the CRC groups.

   Expand the C_reg waveform to show the individual bits of the register.

CRC Test #2

1. **Now, select the test bench in Test2_CRC.vhd which will set the value of fcs_length to 4 and the predetermined divisor to "10011".**

   This test bench will first put a 5 bit data frame into the CRC with mode=1 to generate a 4-bit FCS. The test bench will capture the output data of the CRC including the generated FCS and then send the 9-bit data frame to the CRC with mode=0 to have the CRC check it for errors. Of course, there should be no errors detected. Then the test bench will generate 511 (2^9 - 1) more data frames using the same 9-bit frame, but sequencing through and applying all the possible error combinations to that same frame. Each is passed through the CRC to see if it detects the error. The test bench will save all the error patterns that were not detected by the CRC into a file named *CRC_undetected_errors.txt.* Where the patterns have a '1', the corresponding bit in the frame of data was flipped, forcing an error. A '0' means the corresponding data bit in the frame was left unchanged.

2. **Run the simulation**

   Run the test bench for 25 milliseconds. Check the console to be sure the simulation is run long enough to complete. The console should have a series of "notes" that look something like the following:

```
ISim>
# run 25ms
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 412500 ps: Note: CRC TEST 2 STARTED (/test2_crc/).
at 412500 ps: Note: TESTING FOR UNDETECTED ERROR PATTERNS (/test2_crc/).
at 912 us: Note: ERROR NOT DETECTED (/test2_crc/).
at 1824 us: Note: ERROR NOT DETECTED (/test2_crc/).
        ⋮
at 23232 us: Note: ERROR NOT DETECTED (/test2_crc/).
at 24144 us: Note: ERROR NOT DETECTED (/test2_crc/).
at 24529 us: Note: CRC TEST 2 COMPLETED (/test2_crc/).
ISim>
```

   !!!!! Be sure that you get the "CRC TEST 2 COMPLETED" message !!!!!

   Also, if you get the console message, "Error: UNEXPECTED ERROR DETECTED BY CRC", your CRC is setting the error output signal when a frame has no errors. You should not see this message if your CRC is working correctly.

3. Open the file, *CRC_undetected_errors.txt*, and analyze the results:

This file captured the error patterns that were not detected by the CRC out of all of the possible error patterns where a '0' represents no error and a '1' represents an error that was induced into the transmitted data. An error burst is counted from the first '1' in the error pattern to the last '1' in the error pattern. For example, "001011100" has a 5-bit burst error pattern.

1. How many error patterns were not detected by the CRC? _____
2. How many 4-bit burst error patterns were not detected? _____
3. How many 5-bit burst error patterns were not detected? _____
4. How many 6-bit burst error patterns were not detected? _____
5. How many 7-bit burst error patterns were not detected? _____
6. How many 8-bit burst error patterns were not detected? _____
7. How many 9-bit burst error patterns were not detected? _____
8. What percentage of the 511 error patterns were not detected? _____

Submit your answers to this set of questions to the Lab4 Submission assignment on myCourses

4. Demonstrate your CRC simulation with the *Test2_CRC.vhd* test bench to a proctor and get checked off.

Show the waveforms for the first few frames generated with errors.

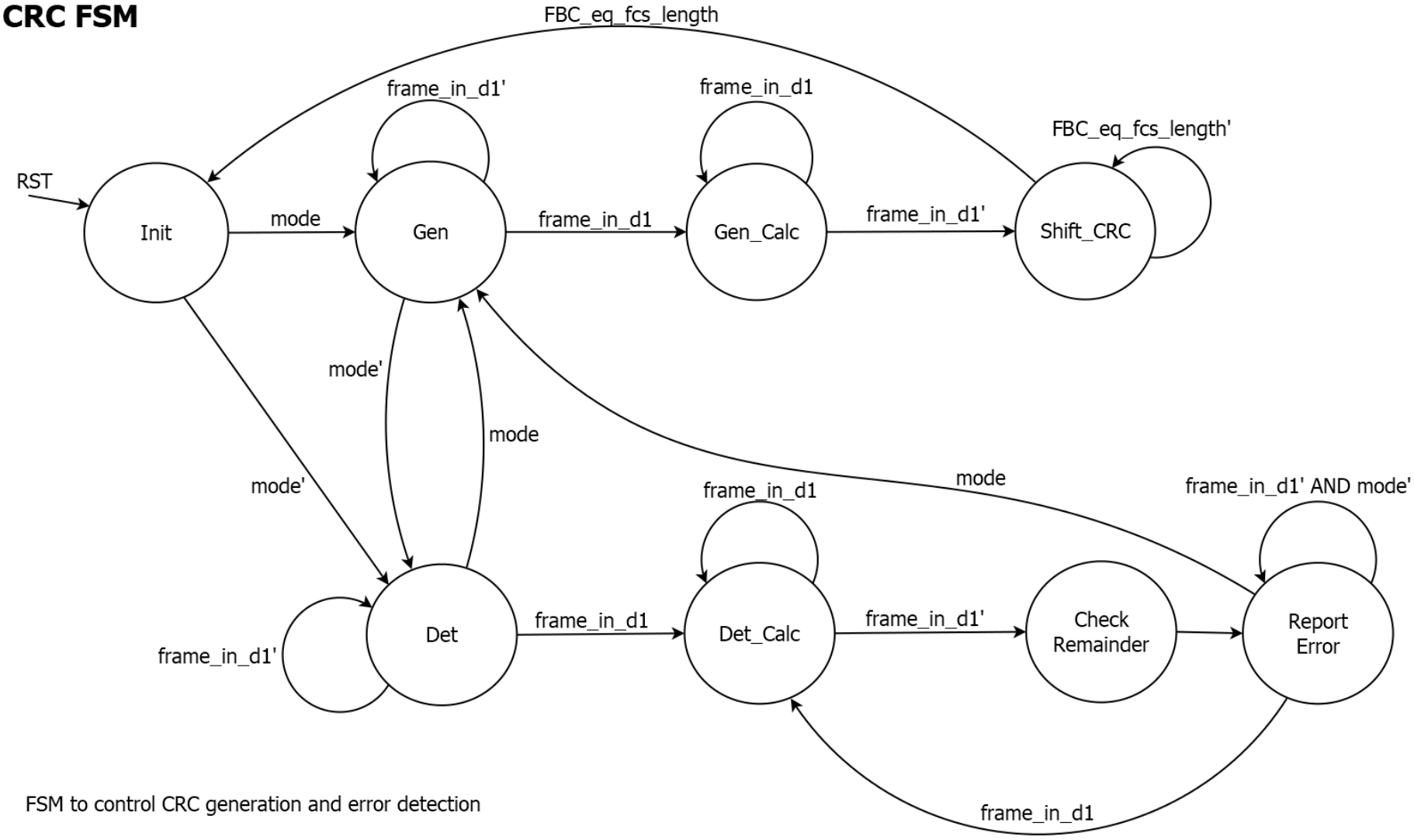
Show that the console messages are what is expected.

Show the contents of the *CRC_undetected_error.txt* file.

5. Submit your VHDL code for the CRC (*CRC.vhd*) to the Lab4 assignment

FSM Template on next page

CRC FSM



FSM to control CRC generation and error detection