TFE4171

Design of Digital Systems II

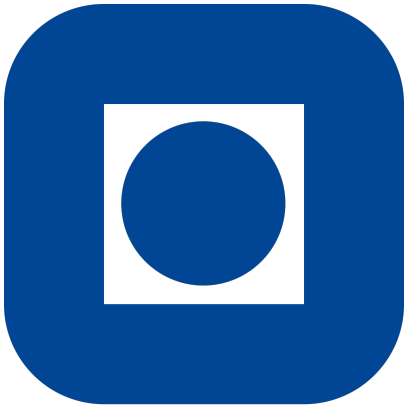
Project

Verification of a High-Level Data Link Controller module

Morten Paulsen
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science & Engineering
mortepau@stud.ntnu.no

Magnus Ramsfjell
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science & Engineering
magnuhr@stud.ntnu.no

April 7, 2020



NTNU

Contents

1 Introduction

This report goes through the verification of a High-Level Data Link Control (HDLC). We are given the RTL code for the implementation of a HDLC module, but there has not been done any verification on it, so our task is to verify the module.

The report begins by introducing the HDLC module shortly in section 2. Showing how the HDLC frames are constructed, how transmission and receiving is done, and how the module does CRC checking. In section 3, we explain the specifications given and describe the assertions added to satisfy the specifications. We also discuss the assertion and coverage reports generated by the testbench.

2 Description

The HDLC module is designed to supply a data communication protocol and provides a bit-oriented code-transparent data transmission.

2.1 Interface

The HDLC's interface is shown in Figure 1, and consists of 11 signals - 9 inputs and 2 outputs.

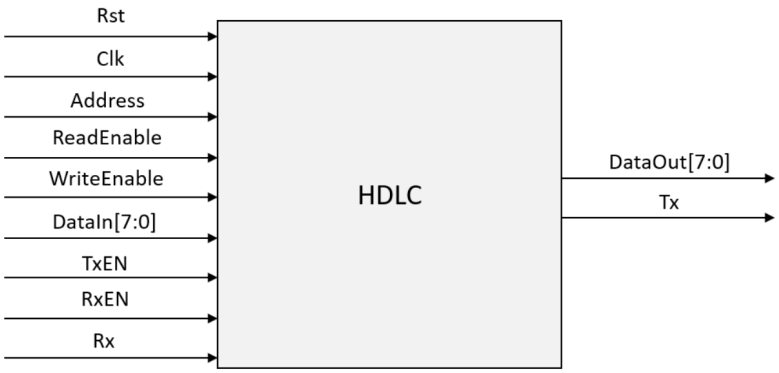


Figure 1: The HDLC interface

The module receives and transmits frames of information, as shown in Table 1. Every frame starts and ends with a flag (0111_1110). The FCS bytes are generated inside the module, using the Cyclic Redundancy Check (CRC), and are used as a safety check to determine if the frame has arrived whole, see subsection 2.3. The Address, Control and Information bytes does not need to be any specific values, but has to be byte aligned.

Flag	Address	Control	Information	FCS	Flag
8 bits	8 or more bits	8 bits	Variable length, n · 8 bits	16 bits	8 bits

Table 1: HDLC frame.

To abort a frame, an abort pattern has to be generated (1111_1110). If an abort pattern is detected the current frame is discarded and ignored.

In the scenario when 5 1's are transmitted consecutively a 0 should be inserted after them on the transmission side to prevent the unfortunate problem of transmitting 6 1's after each other, as this is the start/stop flag. On the receiving end, if 5 1's are detected the following 0 is to be removed.

When no transmission occurs, the line should be kept logical high, which is the idle pattern (1111_1111).

2.2 Internals

In Figure 2 and Figure 3 we can see the internal signaling between blocks for the receive block and the transmit block, respectively. Some of these signals are used in the verification properties.

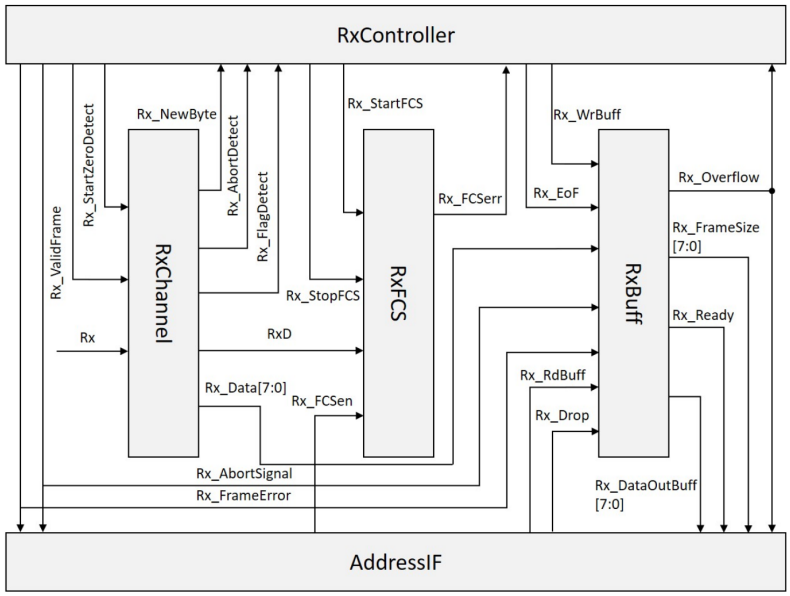


Figure 2: The receive design of the HDLC module.

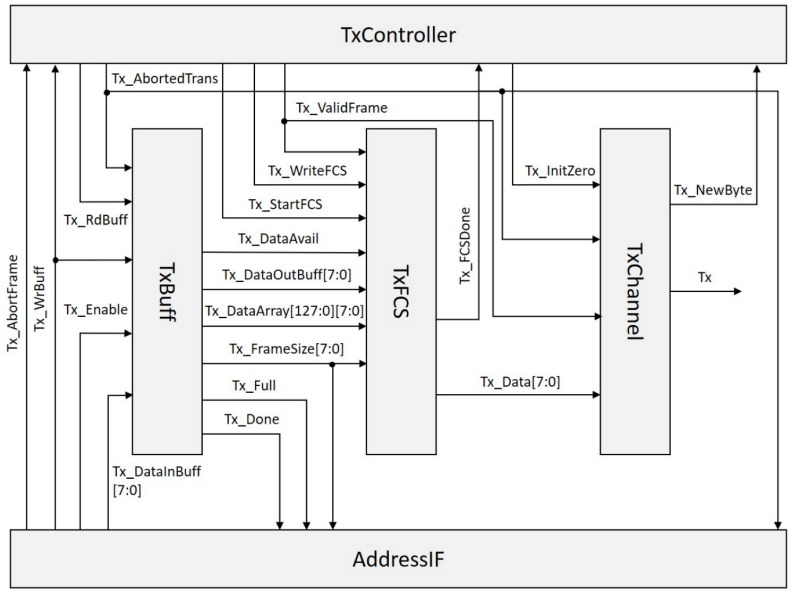


Figure 3: The transmit design of the HDLC module.

2.3 Cyclic Redundancy Check

CRC calculation is done by dividing the message by a predetermined polynomial `P`. In the HDLC module the polynomial

$$P = X^{16} + X^{15} + X^2 + 1$$

is used. The calculation is done by using a shift register and bitwise XOR-ing as shown in Figure 4.

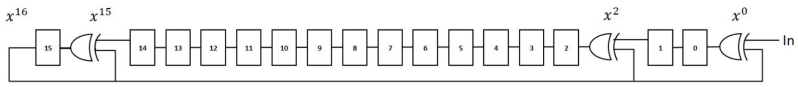


Figure 4: The hardware implementation of the CRC.

On the transmitting side, the CRC is calculated as the information bits are transmitted and appended to the end of the message, but before the stop flag. On the receiving end, the CRC should be zero after receiving the complete message. If it is not zero, a received bit has had the wrong polarization and the frame is flagged as erroneous.

3 Verification

3.1 Specifications

To verify the HDLC module we are given the following specifications:

1. Correct data in RX buffer according to RX input. The buffer should contain up to 128 bytes (this includes the 2 FCS bytes, but not the flags).
2. Attempting to read RX buffer after aborted frame, frame error or dropped frame should result in zeros.
3. Correct bits set in RX status/control register after receiving frame. Remember to check all bits. I.e. after an abort the `Rx_Overflow` bit should be 0, unless an overflow also occured.
4. Correct TX output according to written TX buffer.
5. Start and end of frame pattern generation (Start and end flag: 0111_1110).
6. Zero insertion and removal for transparent transmission.
7. Idle pattern generation and checking (1111_1111 when not operating).
8. Abort pattern generation and checking (1111_1110). Remember that the 0 must be sent first.
9. When aborting frame during transmission, `Tx_AbortedTrans` should be asserted.
10. Abort pattern detected during valid frame should generate `Rx_AbortSignal`.
11. CRC generation and checking.
12. When a whole RX frame has been received, check if end of frame is generated.
13. When receiving more than 128 bytes, `Rx_Overflow` should be asserted.
14. `Rx_FrameSize` should equal the number of bytes received in a frame (max. 126 bytes = 128 bytes in buffer - 2 FCS bytes).
15. `Rx_Ready` should indicate byte(s) in RX buffer is ready to be read.
16. Non-byte aligned data or error in FCS checking should result in frame error.
17. `Tx_Done` should be asserted when the entire TX buffer has been read for transmission.
18. `Tx_Full` should be asserted after writing 126 or more bytes to the TX buffer (overflow).

3.2 Immediate assertions

Specification number 1, 2, 4 and 11 are implemented as immediate assertions. The immediate assertions are tested by creating specific stimuli based on which behaviour we want to test. The code for the stimuli generation is shown in Appendix A line 620-715 for the Rx side and Appendix A line 717-797 for the Tx side. For the Rx side there are 6 different behaviour to verify and for the Tx side there are 2 behaviours to verify.

3.2.1 Rx, Normal behaviour

To verify the normal behaviour for the Rx side, we use the task `VerifyNormalReceive` (line 39-75, Appendix A). In the task we verify that `Rx_SC` is set correctly - Only `Rx_Ready` set, then we verify that the size is matching the value stored in `Rx_Len` and lastly we read all the data stored in `Rx_Buff` and assert that it matches the content we used as stimuli.

3.2.2 Rx, Overflow behaviour

To verify the overflow behaviour for the Rx side, we used the task `VerifyOverflowReceive` (line 80-127, Appendix A). This task is similar to `VerifyNormalReceive` except that we in `Rx_SC` to also have `Rx_Overflow` set. We also attempt to read a few extra bytes to check if the additional stimuli created on the `Rx` input is stored. We expect it to result in `Rx_Buff = 8'b0`.

3.2.3 Rx, Abort behaviour

To verify the abort behaviour for the Rx side, we used the task `VerifyAbortReceive` (line 131-153, Appendix A). In the task we check that `Rx_SC` is correct and has `Rx_Abort` asserted. The other thing we do in that task is to assert that reading from `Rx_Buff` results in `Rx_buff = 8'b0`.

3.2.4 Rx, Drop behaviour

To verify the drop behaviour for the Rx side, we used the task `VerifyDropReceive` (line 157-173, Appendix A). In the task we assert that by writing the `Rx_Drop` bit in `Rx_SC` and by then reading `Rx_Buff` results in `Rx_Buff = 8'b0`.

3.2.5 Rx, Non-byte aligned behaviour

To verify the Non-byte aligned behaviour for the Rx side, we used the task `VerifyNonByteAlignedReceive` (line 177-199, Appendix A). In the task we assert that only `Rx_FrameError` is set in `Rx_SC` and that the size of `Rx_Len = 8'b0`.

3.2.6 Rx, FCS error behaviour

To verify the FCS error behaviour for the Rx side, we used the task `VerifyFCSErrReceive` (line 203-225 Appendix A). In the task we assert that only `Rx_FrameError` in `Rx_SC` is set and that `Rx_Len = 8'b0`, just as in `VerifyNonByteAlignedReceive`.

3.2.7 Tx, Non-Abort behaviour

To verify the Non-Abort behaviour for the Tx side, we used the task `VerifyNonAbortTransmit` (line 229-282, Appendix A). In the task we first check that the start flag is generated, after that we check that all the data bytes are sent correctly (including potential zero bits after 5 consequenct 1's). Second to last, we check that the FCS bytes which are generated matches the one we calculated. And finally, we check that the end of frame flag is generated.

3.2.8 Tx, Abort behaviour

To verify the Abort behaviour for the Tx side, we used the task `VerifyAbortTransmit` (line 286-346, Appendix A). In the task we check that the start frame flag is sent, and that half the data matches (same procedure as in `VerifyNonAbortTransmit`). After checking half the data, the `Tx_AbortFrame` is enabled in `Tx_SC`. After setting that bit we check that `Tx_AbortedTrans` and `Tx_Done` is asserted in `Tx_SC`. After checking `Tx_SC` we assert that the `idle` pattern is sent on `Tx` instead of the rest of the data.

3.3 Concurrent assertions

The remaining specifications are implemented as concurrent assertions. This is done since they can be more easily verified by continuously checking that a specific pattern occurs, and not check for it at certain times as with immediate assertions.

To simplify the properties themselves we are using the sequences shown in Listing 1. The sequences `idle`, `flag`, `abort` and `zero` are sequences that are similar for both the Tx side and the Rx side.

```
80  sequence idle(signal);
81      signal [*8];
82  endsequence;
83
84  sequence flag(signal);
85      !signal ##1 signal [*6] ##1 !signal;
86  endsequence;
87
88  sequence abort(signal);
89      !signal ##1 signal [*7];
90  endsequence;
91
92  sequence zero(signal);
93      !signal ##1 signal [*5] ## !signal;
94  endsequence;
95
96  sequence Rx_DataZero;
97      ( Rx_Data ==? 8'b11111xxx) or
98      ( Rx_Data ==? 8'bx11111xx) or
99      ( Rx_Data ==? 8'bxx11111x) or
100     ( Rx_Data ==? 8'bxxx11111) or
101     ((Rx_Data ==? 8'bxxxx1111) && ($past(Rx_Data, 8) ==? 8'b1xxxxxxx)) or
102     ((Rx_Data ==? 8'bxxxxx111) && ($past(Rx_Data, 8) ==? 8'b11xxxxxx)) or
103     ((Rx_Data ==? 8'bxxxxxx11) && ($past(Rx_Data, 8) ==? 8'b111xxxxx)) or
104     ((Rx_Data ==? 8'bxxxxxxx1) && ($past(Rx_Data, 8) ==? 8'b1111xxxx));
105 endsequence
106
107 sequence Tx_DataZero;
108     ( Tx_Data ==? 8'b111110xx) or
109     ( Tx_Data ==? 8'bx111110x) or
110     ( Tx_Data ==? 8'bxx111110) or
111     ((Tx_Data ==? 8'bxxx11111) && ($past(Tx_Data, 8) ==? 8'b0xxxxxxx)) or
112     ((Tx_Data ==? 8'bxxxx1111) && ($past(Tx_Data, 8) ==? 8'b10xxxxxx)) or
113     ((Tx_Data ==? 8'bxxxxx111) && ($past(Tx_Data, 8) ==? 8'b110xxxxx)) or
114     ((Tx_Data ==? 8'bxxxxxx11) && ($past(Tx_Data, 8) ==? 8'b1110xxxx)) or
115     ((Tx_Data ==? 8'bxxxxxxx1) && ($past(Tx_Data, 8) ==? 8'b11110xxx));
116 endsequence
```

Listing 1: Sequences used in properties.

3.3.1 Specification 3

To assert that the `Rx_SC` status register behaves correctly after receiving a frame we check the status after `Rx_EoF` is asserted. When we have a `Rx_FrameError` we want only `Rx_FrameError` to be asserted, and in the other scenarios we want the correct status signal to be asserted as well as `Rx_Ready`.

```
property p_Rx_Status;
    @(posedge Clk) disable iff(!Rst) $rose(Rx_EoF) |->
            if (Rx_FrameError)
                    !Rx_Ready && !Rx_Overflow && !Rx_AbortSignal &&
                    ↪  Rx_FrameError
            else if (Rx_AbortSignal && Rx_Overflow)
                     Rx_Ready &&  Rx_Overflow &&  Rx_AbortSignal &&
                     ↪  !Rx_FrameError
            else if (Rx_AbortSignal)
                     Rx_Ready && !Rx_Overflow &&  Rx_AbortSignal &&
                     ↪  !Rx_FrameError
            else if (Rx_Overflow)
                     Rx_Ready &&  Rx_Overflow && !Rx_AbortSignal &&
                     ↪  !Rx_FrameError
            else
                     Rx_Ready && !Rx_Overflow && !Rx_AbortSignal &&
                     ↪  !Rx_FrameError
endproperty
```

Listing 2: Property used to assert correct signals in Rx_SC after receiving a frame.

3.3.2 Specification 5

To check that start and end of frame patterns are generated correctly we implemented the property `p_Tx_FramePattern`. In addition, we implemented the property `p_Rx_FramePattern` to verify that the start and end of frame patterns are detected on the receiving side as well.

The property `p_Tx_FramePattern` checks that whenever `Tx_ValidFrame` changes, and it was not because of an abort, the flag sequence is generated.

On the other hand, property `p_Rx_FramePattern` assert that whenever the flag pattern is detected the `Rx_FlagDetect` should be asserted.

```
property p_Tx_FramePattern;
    @(posedge Clk) disable iff (!Rst) !$stable(Tx_ValidFrame) ##0
    ↪  $past(!Tx_AbortFrame, 2) |-> ##[1:2] flag(Tx);
endproperty

property p_Rx_FramePattern;
    @(posedge Clk) flag(Rx) |-> ##2 Rx_FlagDetect;
endproperty
```

Listing 3: Properties verifying the flag pattern generation and checking.

3.3.3 Specification 6

Specification 6 is the check for zero insertion and zero removal. The zero insertion is checked on the transmission side while the zero removal is checked on the receiving side.

To assert that the zero is inserted correctly we check that whenever `Tx_NewByte` is asserted and the data pattern matches one of the cases in the sequence `Tx_DataZero` the `zero` sequence should be found on `Tx` 13 to 22 cycles later. When we receive the `zero` pattern we should see it removed in a data byte 9 to 17 clock cycles later.

```
147 property p_Tx_InsertZero;
148     @(posedge Clk) disable iff (!Rst || !Tx_ValidFrame) $rose(Tx_NewByte)
        ↪  ##0 Tx_DataZero |-> ##[13:22] zero(Tx);
149 endproperty
150
151 property p_Rx_RemoveZero;
152     @(posedge Clk) disable iff (!Rst) (zero(Rx) and Rx_ValidFrame [*6])
        ↪  |-> ##[9:17] Rx_NewByte ##1 Rx_DataZero;
153 endproperty
```

Listing 4: Properties verifying the zero insertion and removal.

3.3.4 Specification 7

The properties `p_Tx_IdlePattern` and `p_Rx_IdlePattern` are used to verify that the `idle` pattern is generated and that it is detected. The `idle` pattern should be generated when no transmission occurs, which is when `Tx_ValidFrame` is deasserted and `Tx_FrameSize = 0`. We need to add `Tx_FrameSize` to the antecedent since we otherwise would check for the idle pattern when end of frame pattern is generated.

The property `Rx_IdlePattern` checks that when we detect the `idle` pattern `Rx_FlagDetect` should stay deasserted.

```
156 property p_Tx_IdlePattern;
157     @(posedge Clk) disable iff (!Rst) !Tx_ValidFrame && Tx_FrameSize ==
        ↪  8'b0 |-> idle(Tx);
158 endproperty
159
160 property p_Rx_IdlePattern;
161     @(posedge Clk) disable iff (!Rst) idle(Rx) |=> !Rx_FlagDetect;
162 endproperty
```

Listing 5: Properties verifying the idle pattern generation and checking.

3.3.5 Specification 8

The properties `p_Tx_AbortPattern` and `p_Rx_AbortPattern` are verifying that the abort pattern is generated and checked correctly. `p_Tx_AbortPattern` asserts that the `abort` pattern is generated whenever `Tx_AbortFrame` is asserted. Likewise, `p_Rx_AbortPattern` verifies that whenever we detect the `abort` pattern and `Rx_ValidFrame` is deasserted the `Rx_AbortDetect` should be generated.

```
165 property p_Tx_AbortPattern;
166     @(posedge Clk) disable iff (!Rst) $rose(Tx_AbortFrame) |-> ##4
        ↪  abort(Tx);
167 endproperty
168
169 property p_Rx_AbortPattern;
170     @(posedge Clk) disable iff (!Rst) abort(Rx) and (!Rx_ValidFrame [*7])
        ↪  |-> ##2 $rose(Rx_AbortDetect);
171 endproperty
```

Listing 6: Properties verifying the abort pattern generation and checking.

3.3.6 Specification 9

The property `p_Tx_AbortSignal` verifies that `Tx_AbortedTrans` is asserted after `Tx_AbortFrame` has been asserted.

```
174 property p_Tx_AbortSignal;
175     @(posedge Clk) disable iff (!Rst) $rose(Tx_AbortFrame) && Tx_DataAvail
        ↪  ##1 $fell(Tx_AbortFrame) |=>  $rose(Tx_AbortedTrans);
176 endproperty
```

Listing 7: Property verifying the Abort signal behaviour when aborting frame.

3.3.7 Specification 10

The property `p_Rx_AbortSignal` checks that if we are currently receiving a frame and `Rx_AbortDetect` is asserted, `Rx_AbortSignal` should be asserted.

```
179 property p_Rx_AbortSignal;
180     @(posedge Clk) disable iff (!Rst) Rx_ValidFrame && Rx_AbortDetect |=>
        ↪  Rx_AbortSignal;
181 endproperty
```

Listing 8: Property verifying generation of Rx_AbortSignal.

3.3.8 Specification 12

The property `p_Rx_EndOfFrame` verifies that whenever `Rx_ValidFrame` falls - indicating that either the frame has been received or that an error has occured - `Rx_EoF` should be asserted.

```
184 property p_Rx_EndOfFrame;
185     @(posedge Clk) disable iff (!Rst) $fell(Rx_ValidFrame) |=>
        ↪  $rose(Rx_EoF);
186 endproperty
```

Listing 9: Property verifying the generation of Rx_EoF.

3.3.9 Specification 13

The property `p_Rx_Overflow` verifies that `Rx_Overflow` is asserted if we, within a single frame, receive more than 126 bytes (more than 128 including FCS bytes). This is achieved by counting how many times `Rx_NewByte` is asserted.

```
189 property p_Rx_Overflow;
190     @(posedge Clk) disable iff (!Rst || !Rx_ValidFrame)
        ↪ $rose(Rx_ValidFrame) ##0 ($rose(Rx_NewByte) [->129]) |=>
        ↪ $rose(Rx_Overflow)
191 endproperty
```

Listing 10: Property verifying the generation of Rx_Overflow when receiving more than 126 bytes.

3.3.10 Specification 14

The property `p_Rx_FrameSize` verifiers that the number of bytes we receive actually matches the valud of `Rx_FrameSize` when the whole frame have been received.

```
194 property p_Rx_FrameSize;
195     int bytes = 0;
196     @(posedge Clk) disable iff (!Rst) $rose(Rx_ValidFrame) ##0 ((##[7:9]
        ↪ $rose(Rx_NewByte), bytes++) [*1:128]) ##5 ($rose(Rx_EoF) and
        ↪ !Rx_FrameError) |=> Rx_FrameSize == (bytes - 2);
197 endproperty
```

Listing 11: Property verifying that Rx_FrameSize matches the number of bytes received.

3.3.11 Specification 15

The property `p_Rx_Ready` verifies that when `Rx_Ready` is asserted that also `Rx_EoF` has asserted and that we are not receiving a frame anymore.

```
200 property p_Rx_Ready;
201     @(posedge Clk) disable iff (!Rst) $rose(Rx_Ready) |-> $rose(Rx_EoF)
        ↪ and !Rx_ValidFrame;
202 endproperty
```

Listing 12: Property verifying Rx_Ready signalling data ready to be read.

3.3.12 Specification 16

The property `p_Rx_FCSerr` verifies that whenever `Rx_FCSerr` and `RX_FCSen` is asserted (FCS detection enabled) that `Rx_FrameError` is asserted indicating that an error has occured.

```
105 property p_Rx_FCSerr;
106     @(posedge Clk) disable iff (!Rst) $rose(Rx_FCSerr) && Rx_FCSen |=>
        ↪ $rose(Rx_FrameError);
107 endproperty
```

Listing 13: Property verifying Rx_FrameError behavior.

3.3.13 Specification 17

The property `p_Tx_Done` verifies that when no more data is available for transmission (`Tx_DataAvail` falls) `Tx_Done` is also asserted.

```
210 property p_Tx_Done;
211     @(posedge Clk) disable iff (!Rst) $fell(Tx_DataAvail) |->
        ↪ $past(Tx_Done, 1);
212 endproperty
```

Listing 14: Property verifying the generation of Tx_Done.

3.3.14 Specification 18

The property `p_Tx_Full` verifies that when `Tx_Full` is asserted if we receive more than 126 bytes. This is done by counting the number of times `Tx_WrBuff` is asserted while reading in data. We use `[*125]` since `$fell(Tx_Done)` is caused by `$rose(Tx_WrBuff)`.

```
215 property p_Tx_Full;
216     @(posedge Clk) disable iff (!Rst) $fell(Tx_Done) ##0
        ↪ (($rose(Tx_WrBuff) [->1]) [*125]) ##0 !Tx_Done |=> Tx_Full;
217 endproperty
```

Listing 15: Property verifying the generation of Tx_Full.

3.4 Coverage

To generate the coverage report in Appendix C, we created the covergroup as shown in lines 402-483 in A. In the covergroup we create coverpoints for the data and address signals in the interface `DataIn`, `DataOut` and `Address`. `DataIn` and `DataOut` has a bin for each value, while `Address` has a bin for each valid address and uses an `ignore_bins` for the remaining values. The Rx signals which are covered are `Rx_Data`, `Rx_FrameSize`, `Rx_ValidFrame`, `Rx_AbortSignal`, `Rx_Ready`, `Rx_EoF`, `Rx_Overflow`, `Rx_FCSerr`, `Rx_FrameError` and `Rx_Drop`. Each of them, except `Rx_Data` and `Rx_FrameSize`, has two bins, covering both binary values. `Rx_Data` has a bin for each value (0 - 255) and `Rx_FrameSize` has a bin for each value in the range [0,126] and uses `ignore_bins` for the remaining values (127-255). For the Tx side we have coverpoints for the signals `Tx_Data`, `Tx_FrameSize`, `Tx_ValidFrame`, `Tx_AbortedTrans`, `Tx_Full`, `Tx_Enable` and `Tx_AbortFrame`. These signals follows the same configuration as for their respective Rx signals.

The stimuli we created obtained a total coverage of 89%. The remaining 11% comes from the uncovered bins in `DataIn`, `DataOut`, `Rx_Data`, `Rx_FrameSize`, `Tx_Data` and `Tx_FrameSize`. It is possible to achieve 100% coverage by adding executing both `Receive` and `Transmit` more times, but with different configurations on the data size.

A testPr_hdlc.sv

```
1  ////////////////////////////////////////////////
2  // Title:         testPr_hdlc
3  // Author:
4  // Date:
5  ////////////////////////////////////////////////
6
7  program testPr_hdlc(
8      in_hdlc uin_hdlc
9  );
10
11     int TbErrorCnt;
12
13     // Addresses
14     logic [2:0] TXSC   = 3'b000,
15                 TXBUFF = 3'b001,
16                 RXSC   = 3'b010,
17                 RXBUFF = 3'b011,
18                 RXLEN  = 3'b100;
19
20     // Address bits
21     logic [7:0] TXSC_FULL         = 8'b0001_0000,
22                 TXSC_ABORTEDTRANS = 8'b0000_1000,
23                 TXSC_ABORTFRAME   = 8'b0000_0100,
24                 TXSC_ENABLE       = 8'b0000_0010,
25                 TXSC_DONE         = 8'b0000_0001,
26                 RXSC_FCSEN        = 8'b0010_0000,
27                 RXSC_OVERFLOW     = 8'b0001_0000,
28                 RXSC_ABORTSIGNAL  = 8'b0000_1000,
29                 RXSC_FRAMEERROR   = 8'b0000_0100,
30                 RXSC_DROP         = 8'b0000_0010,
31                 RXSC_READY        = 8'b0000_0001;
32
33     // Read masks
34         logic [7:0] RXSC_READ_MASK = 8'b1101_1101,
35                 TXSC_READ_MASK = 8'b1111_1001;
36
37     // VerifyNormalReceive should verify correct value in the Rx status/control register
38     // and that the output is correct
39     task VerifyNormalReceive(logic [127:0][7:0] data, int Size);
40             logic [7:0] ReadData, DataLen;
41
42         // Wait for Rx_Ready to be asserted
43             wait(uin_hdlc.Rx_Ready);
44
45             // Assert that only Rx_Ready is set in RX_SC
46             ReadAddress(RXSC, ReadData);
47         // Only check RO bits
48             ReadData = ReadData & RXSC_READ_MASK;
49         assert (ReadData == RXSC_READY) begin
50             $display("PASS: VerifyNormalReceive:: Rx_SC correct");
51         end else begin
52                 TbErrorCnt++;
53                 $error("FAIL: VerifyNormalReceive:: Expected Rx_SC = 0x01, Received Rx_SC = 0x%h", ReadData);
54             end
```

```
55
56              // Assert data length is correct
57              ReadAddress(RXLEN, DataLen);
58          assert (DataLen == Size) begin
59              $display("PASS: VerifyNormalReceive:: Rx_Len correct");
60          end else begin
61              TbErrorCnt++;
62              $error("FAIL: VerifyNormalReceive:: Expected Rx_Len = %0d, Received Rx_Len = %0d", Size, DataLen);
63          end
64
65          // Check content
66              for (int i = 0; i < DataLen; i++) begin
67                  ReadAddress(RXBUFF, ReadData);
68              assert(ReadData == data[i]) begin
69                  $display("PASS: VerifyNormalReceive:: Rx_Buff correct");
70              end else begin
71                      TbErrorCnt++;
72                      $error("FAIL: VerifyNormalReceive:: Expected Rx_Buff = 0x%h, Received Rx_Buff = 0x%h", data[i], Re
73                  end
74              end
75          endtask
76
77          // VerifyOverflowReceive should verify correct value in the Rx status/control
78          // register, that the data is correct and that the Rx data buffer is
79      // zero after overflow.
80          task VerifyOverflowReceive(logic [127:0][7:0] data, int Size);
81              logic [7:0] ReadData, DataLen;
82
83          // Wait for Rx_Ready to be asserted
84              wait(uin_hdlc.Rx_Ready);
85
86              // Assert that Rx_Ready and Rx_Overflow is set
87              ReadAddress(RXSC, ReadData);
88          // Only check RO bits
89              ReadData = ReadData & RXSC_READ_MASK;
90              assert (ReadData == (RXSC_OVERFLOW | RXSC_READY))
91              $display("PASS: VerifyOverflowReceive:: Rx_SC correct");
92              else begin
93                  TbErrorCnt++;
94                  $error("FAIL: VerifyOverflowReceive:: Expected Rx_SC = 0x11, Received Rx_SC = 0x%h", ReadData);
95              end
96
97              // Assert data length is correct
98              ReadAddress(RXLEN, DataLen);
99          assert (DataLen == Size) begin
100             $display("PASS: VerifyOverflowReceive:: Rx_Len correct");
101         end else begin
102             TbErrorCnt++;
103             $error("FAIL: VerifyOverflowReceive:: Expected Rx_Len = %0d, Received Rx_Len = %0d", Size, DataLen);
104         end
105
106         // Check content
107             for (int i = 0; i < DataLen; i++) begin
108                 ReadAddress(RXBUFF, ReadData);
109             assert(ReadData == data[i]) begin
110                 $display("PASS: VerifyOverflowReceive:: Rx_Buff correct");
```

```
111             end else begin
112                     TbErrorCnt++;
113                     $error("FAIL: VerifyOverflowReceive:: Expected Rx_Buff = 0x%h, Received Rx_Buff = 0x%h", data[i], 
114                 end
115             end
116 
117             // Read a few extra bytes and check that they are invalid
118         for (int i = 0; i < 3; i++) begin
119             ReadAddress(RXBUFF, ReadData);
120             assert(ReadData == 8'b0)
121                 $display("PASS: VerifyOverflowReceive:: Rx_Buff correct");
122             else begin
123                 TbErrorCnt++;
124                 $error("FAIL: VerifyOverflowReceive:: Expected Rx_Buff = 0x00, Received Rx_Buff = 0x%h", ReadData);
125             end
126         end
127     endtask
128 
129         // VerifyAbortReceive should verify correct value in the Rx status/control
130         // register, and that the Rx data buffer is zero after abort.
131         task VerifyAbortReceive(logic [127:0][7:0] data, int Size);
132             logic [7:0] ReadData;
133 
134             // Assert that only Rx_AbortSignal is set
135             ReadAddress(RXSC, ReadData);
136         // Only check RO bits
137             ReadData = ReadData & RXSC_READ_MASK;
138             assert (ReadData == RXSC_ABORTSIGNAL)
139             $display("PASS: VerifyAbortReceive:: Rx_SC correct");
140             else begin
141                 TbErrorCnt++;
142                 $error("FAIL: VerifyAbortReceive:: Expected Rx_SC = 0x08, Received Rx_SC = 0x%h", ReadData);
143             end
144 
145             // Assert that data is invalid
146             ReadAddress(RXBUFF, ReadData);
147             assert (ReadData == 8'b0)
148             $display("PASS: VerifyAbortReceive:: Rx_Buff correct");
149             else begin
150                 TbErrorCnt++;
151                 $error("FAIL: VerifyAbortReceive:: Expected Rx_Buff = 0x00, Received Rx_Buff = 0x%h", ReadData);
152             end
153         endtask
154 
155         // VerifyDropReceive should verify that the Rx data buffer is zero
156     // after dropping the frame.
157         task VerifyDropReceive(logic [127:0][7:0] data, int Size);
158             logic [7:0] ReadData;
159 
160         // Drop the current frame
161         WriteAddress(RXSC, RXSC_DROP);
162 
163         @(posedge uin_hdlc.Clk);
164 
165             // Assert that Rx_Buff is invalid
166             ReadAddress(RXBUFF, ReadData);
```

```
167             assert (ReadData == 8'b0)
168             $display("PASS: VerifyDropReceive:: Rx_Buff correct", ReadData);
169             else begin
170                 TbErrorCnt++;
171                 $error("FAIL: VerifyDropReceive:: Expected Rx_Buff = 0x00, Received Rx_Buff = 0x%h", ReadData);
172             end
173         endtask
174
175         // VerifyNonByteAlignedReceive should verify correct value in the Rx status/control
176         // register, and that the Rx data buffer is zero after Non-Byte-Alignment.
177         task VerifyNonByteAlignedReceive(logic [127:0][7:0] data, int Size);
178             logic [7:0] ReadData;
179
180             // Assert that only Rx_FrameError is set
181             ReadAddress(RXSC, ReadData);
182         // Only check RO bits
183             ReadData = ReadData & RXSC_READ_MASK;
184             assert (ReadData == RXSC_FRAMEERROR)
185             $display("PASS: VerifyNonByteAlignedReceive:: Rx_SX correct");
186             else begin
187                 TbErrorCnt++;
188                 $error("FAIL: VerifyNonByteAlignedReceive:: Expected Rx_SC = 0x04, Received Rx_SC = 0x%h", ReadData);
189             end
190
191             // Assert that Rx_Buff is invalid
192             ReadAddress(RXBUFF, ReadData);
193             assert (ReadData == 8'b0)
194             $display("PASS: VerifyNonByteAlignedReceive:: Rx_Buff correct", ReadData);
195             else begin
196                 TbErrorCnt++;
197                 $error("FAIL: VerifyNonByteAlignedReceive:: Expected Rx_Buff = 0x00, Received Rx_Buff = 0x%h", ReadDat
198             end
199         endtask
200
201         // VerifyFCSErrReceive should verify correct value in the Rx status/control
202         // register, and that the Rx data buffer is zero after frameError.
203         task VerifyFCSErrReceive(logic [127:0][7:0] data, int Size);
204             logic [7:0] ReadData, DataLen;
205
206             // Assert that only Rx_FrameError is set
207             ReadAddress(RXSC, ReadData);
208             // Only check RO bits
209             ReadData = ReadData & RXSC_READ_MASK;
210             assert (ReadData == RXSC_FRAMEERROR)
211             $display("PASS: VerifyFCSErrReceive:: Rx_SC correct");
212             else begin
213                 TbErrorCnt++;
214                 $error("FAIL: VerifyFCSErrReceive:: Expected Rx_SC = 0x04, Received Rx_SC = 0x%h", ReadData);
215             end
216
217             // Assert that Rx_Buff is invalid
218             ReadAddress(RXBUFF, ReadData);
219             assert (ReadData == 8'b0)
220             $display("PASS: VerifyFCSErrReceive:: Rx_Buff correct", ReadData);
221             else begin
222                 TbErrorCnt++;
```

```
223                 $error("FAIL: VerifyFCSErrReceive:: Expected Rx_Buff = 0x00 Received Rx_Buff = 0x%h", ReadData);
224             end
225         endtask
226 
227     // VerifyNonAbortTransmit should verify correct output stream from Tx
228     // and that CRC is correct
229     task VerifyNonAbortTransmit(logic [128*8+204:0] fData, int Size, int FCSSize);
230         // Using +3 as this takes potential zero insertions into consideration
231         // floor(16/5)=3
232         logic [15+3:0] FCSBytes,
233                        FCSBytes_calc;
234         logic  [7:0] flag;
235 
236         flag = 8'b0111_1110;
237 
238         // Check flag
239         for (int f = 0; f < 8; f++) begin
240             if (f != 0) begin
241                 @(posedge uin_hdlc.Clk);
242             end
243             assert(uin_hdlc.Tx == flag[f]) else begin
244                 $error("FAIL: VerifyNonAbortTransmit:: Expected Tx_flag = 0b%b, Received Tx_flag = 0b%b", flag[f], uin
245                 TbErrorCnt++;
246             end
247         end
248 
249         // Check data
250         for (int i = 0; i < Size; i++) begin
251             @(posedge uin_hdlc.Clk);
252                 assert (fData[i] == uin_hdlc.Tx) else begin
253                     $error("FAIL. VerifyNonAbortTransmit:: Expected Tx = 0b%b, Received Tx = 0b%b", fData[i], uin_hdlc
254                     TbErrorCnt++;
255                 end
256         end
257 
258         // Check FCS bytes
259         FCSBytes = '0;
260         FCSBytes_calc = '0;
261         for (int i = 0; i < FCSSize; i++) begin
262             @(posedge uin_hdlc.Clk) begin
263                 FCSBytes[i] = uin_hdlc.Tx;
264                 FCSBytes_calc[i] = fData[Size + i];
265             end
266         end
267         assert(FCSBytes == FCSBytes_calc) begin
268             $display("PASS: VerifyNonAbortTransmit:: FCSBytes correct");
269         end else begin
270             $error("FAIL: VerifyNonAbortTransmit:: Expected FCSBytes = 0x%5h, Received FCSBytes = 0x%5h", FCSBytes_cal
271             TbErrorCnt++;
272         end
273 
274         // Check flag
275         for (int f = 0; f < 8; f++) begin
276             @(posedge uin_hdlc.Clk);
277                 assert(uin_hdlc.Tx == flag[f]) else begin
278                     $error("FAIL: VerifyNonAbortTransmit:: Expected Tx_flag = 0b%b, Received Tx_flag = 0b%b", flag[f],
```

```
279                     TbErrorCnt++;
280                 end
281             end
282         endtask
283
284         // VerifyAbortTransmit should verify correct output stream from Tx
285         // and that the abort sequence is generated and idle after that
286         task VerifyAbortTransmit(logic [128*8+204:0] fData, int Size);
287             // Using +3 as this takes potential zero insertions into consideration
288             // floor(16/5)=3
289             logic [15+3:0] FCSBytes,
290                            FCSBytes_calc;
291             logic  [7:0] flag,
292                          ReadData;
293
294             flag = 8'b0111_1110;
295
296             // Check flag
297             for (int f = 0; f < 8; f++) begin
298                 if (f != 0) begin
299                     @(posedge uin_hdlc.Clk);
300                 end
301                 assert(uin_hdlc.Tx == flag[f]) else begin
302                     $error("FAIL: VerifyAbortTransmit:: Expected Tx_flag = 0b%b, Received Tx_flag = 0b%b", flag[f], uin_hd
303                     TbErrorCnt++;
304                 end
305             end
306
307             // Check data
308             for (int i = 0; i < Size / 2; i++) begin
309                 @(posedge uin_hdlc.Clk);
310                     assert (fData[i] == uin_hdlc.Tx) else begin
311                         $error("FAIL: VerifyAbortTransmit:: Expected Tx = 0b%b, Received Tx = 0b%b", fData[i], uin_hdlc.Tx
312                         TbErrorCnt++;
313                     end
314             end
315
316             // Abort the frame
317             WriteAddress(TXSC, TXSC_ABORTFRAME);
318
319             // Wait 2 clock cycles
320             @(posedge uin_hdlc.Clk);
321             @(posedge uin_hdlc.Clk);
322
323             // Check that Tx_AbortedTrans is asserted
324             ReadAddress(TXSC, ReadData);
325                 // Only check RO bits
326             ReadData = ReadData & TXSC_READ_MASK;
327             assert (ReadData == (TXSC_ABORTEDTRANS | TXSC_DONE)) begin
328                 $display("PASS: VerifyAbortTransmit:: Tx_SC correct");
329             end else begin
330                 $error("FAIL: VerifyAbortTransmit:: Expected Tx_SC = 0x09, Received Tx_SC = 0x%h", ReadData);
331                 TbErrorCnt++;
332             end
333
334             // Wait 2 clock cycles for Tx_AbortFrame to propagate
```

```
335         @(posedge uin_hdlc.Clk);
336         @(posedge uin_hdlc.Clk);
337 
338         // Check that Tx is set back to idle
339         repeat(10) begin
340             @(posedge uin_hdlc.Clk);
341                 assert (uin_hdlc.Tx == 1'b1) else begin
342                     $error("FAIL: VerifyAbortTransmit:: Expected Tx = 0b1");
343                     TbErrorCnt++;
344                 end
345         end
346     endtask
347 
348         /*****************************************************************************
349          *                                                                           *
350          *                              Simulation code                              *
351          *                                                                           *
352          *****************************************************************************/
353 
354     initial begin
355             $display("************************************************************");
356             $display("%t - Starting Test Program", $time);
357             $display("************************************************************");
358 
359             Init();
360 
361             // Receive: Size, Abort, FCSerr, NonByteAligned, Overflow, Drop, SkipRead
362             Receive(      10,     0,      0,              0,        0,    0,        0); // Normal
363             Receive(      40,     1,      0,              0,        0,    0,        0); // Abort
364             Receive(     126,     0,      0,              0,        1,    0,        0); // Overflow
365             Receive(      45,     0,      0,              0,        0,    0,        0); // Normal
366             Receive(     126,     0,      0,              0,        0,    0,        0); // Normal
367             Receive(     122,     1,      0,              0,        0,    0,        0); // Abort
368             Receive(     126,     0,      0,              0,        1,    0,        0); // Overflow
369             Receive(      25,     0,      0,              0,        0,    0,        0); // Normal
370             Receive(      47,     0,      0,              0,        0,    0,        0); // Normal
371             Receive(      58,     0,      1,              0,        0,    0,        0); // FCSerr
372             Receive(      90,     0,      0,              1,        0,    0,        0); // NonByteAligned
373             Receive(      74,     0,      0,              0,        0,    1,        0); // Drop
374             Receive(     101,     0,      0,              0,        0,    0,        1); // SkipRead
375 
376         //Transmit: Size, Abort, Overflow
377         Transmit(     10,     0,        0); // Normal
378         Transmit(    122,     1,        0); // Abort
379         Transmit(    126,     0,        1); // Overflow
380         Transmit(     40,     0,        0); // Normal
381         Transmit(    126,     0,        0); // Normal
382         Transmit(     40,     1,        0); // Abort
383         Transmit(    126,     0,        1); // Overflow
384         Transmit(    122,     0,        0); // Normal
385         Transmit(     47,     0,        0); // Normal
386 
387             $display("************************************************************");
388             $display("%t - Finishing Test Program", $time);
389             $display("************************************************************");
390             $stop;
```

```
391         end
392
393         final begin
394             $display("*********************************");
395             $display("*                                 *");
396             $display("* \tAssertion Errors: %0d\t  *", TbErrorCnt + uin_hdlc.ErrCntAssertions);
397             $display("*                                 *");
398             $display("*********************************");
399         end
400
401         // Covergroup
402     covergroup hdlc_cg() @(posedge uin_hdlc.Clk);
403             Address: coverpoint uin_hdlc.Address {
404             bins Tx_SC = {0};
405             bins Tx_Buff = {1};
406             bins Rx_SC = {2};
407             bins Rx_Buff = {3};
408             bins Rx_Len = {4};
409             ignore_bins Invalid = {[5:7]};
410             }
411             DataIn: coverpoint uin_hdlc.DataIn {
412                 bins DataIn[] = {[0:255]};
413             }
414             DataOut: coverpoint uin_hdlc.DataOut {
415                 bins DataOut[] = {[0:255]};
416             }
417             RxData: coverpoint uin_hdlc.Rx_Data {
418                 bins RxData[] = {[0:255]};
419             }
420             RxFrameSize: coverpoint uin_hdlc.Rx_FrameSize {
421                 bins RxFrameSize[] = {[0:126]};
422             ignore_bins Invalid = {[127:255]};
423             }
424             RxValidFrame: coverpoint uin_hdlc.Rx_ValidFrame {
425                 bins InvalidFrame = { 0 };
426                 bins ValidFrame = { 1 };
427             }
428             RxAbortSignal: coverpoint uin_hdlc.Rx_AbortSignal {
429                 bins Keep = { 0 };
430                 bins Abort = { 1 };
431             }
432             RxReady: coverpoint uin_hdlc.Rx_Ready {
433                 bins NotReady = { 0 };
434                 bins Ready = { 1 };
435             }
436             RxEoF: coverpoint uin_hdlc.Rx_EoF {
437                 bins NotEoF = { 0 };
438                 bins EoF = { 1 };
439             }
440             RxOverflow: coverpoint uin_hdlc.Rx_Overflow {
441                 bins NoOverflow = { 0 };
442                 bins Overflow = { 1 };
443             }
444             RxFCSErr: coverpoint uin_hdlc.Rx_FCSerr {
445                 bins NoError = { 0 };
446                 bins Error = { 1 };
```

```
447             }
448             RxFrameError: coverpoint uin_hdlc.Rx_FrameError {
449                 bins NoFrameError = { 0 };
450                 bins FrameError = { 1 };
451             }
452             RxDrop: coverpoint uin_hdlc.Rx_Drop {
453                 bins Keep = { 0 };
454                 bins Drop = { 1 };
455             }
456         TxValidFrame: coverpoint uin_hdlc.Tx_ValidFrame {
457             bins InvalidFrame = { 0 };
458             bins ValidFrame = { 1 };
459         }
460         TxAbortedTrans: coverpoint uin_hdlc.Tx_AbortedTrans {
461             bins Kept = { 0 };
462             bins Aborted = { 1 };
463         }
464         TxData: coverpoint uin_hdlc.Tx_Data {
465             bins TxData[] = {[0:255]};
466         }
467         TxFull: coverpoint uin_hdlc.Tx_Full {
468             bins NotFull = { 0 };
469             bins Full = { 1 };
470         }
471         TxFrameSize: coverpoint uin_hdlc.Tx_FrameSize {
472             bins TxFrameSize[] = {[0:126]};
473             ignore_bins Invalid = {[127:255]};
474         }
475         TxEnable: coverpoint uin_hdlc.Tx_Enable {
476             bins Disabled = { 0 };
477             bins Enabled = { 1 };
478         }
479         TxAbortFrame: coverpoint uin_hdlc.Tx_AbortFrame {
480             bins Keep = { 0 };
481             bins Abort = { 1 };
482         }
483         endgroup
484
485         // Instantiate the covergroup
486         hdlc_cg inst_hdlc_cg = new();
487
488         task Init();
489             uin_hdlc.Clk         =   1'b0;
490             uin_hdlc.Rst         =   1'b0;
491             uin_hdlc.Address     = 3'b000;
492             uin_hdlc.WriteEnable =   1'b0;
493             uin_hdlc.ReadEnable  =   1'b0;
494             uin_hdlc.DataIn      =     '0;
495             uin_hdlc.TxEN        =   1'b1;
496             uin_hdlc.Rx          =   1'b1;
497             uin_hdlc.RxEN        =   1'b1;
498
499             TbErrorCnt = 0;
500
501             #1000ns;
502             uin_hdlc.Rst         =   1'b1;
```

```
503         endtask
504
505         task WriteAddress(input logic [2:0] Address ,input logic [7:0] Data);
506             @(posedge uin_hdlc.Clk);
507                 uin_hdlc.Address     = Address;
508                 uin_hdlc.WriteEnable = 1'b1;
509                 uin_hdlc.DataIn      = Data;
510             @(posedge uin_hdlc.Clk);
511                 uin_hdlc.WriteEnable = 1'b0;
512         endtask
513
514         task ReadAddress(input logic [2:0] Address ,output logic [7:0] Data);
515             @(posedge uin_hdlc.Clk);
516                 uin_hdlc.Address    = Address;
517                 uin_hdlc.ReadEnable = 1'b1;
518                 #100ns;
519                 Data                = uin_hdlc.DataOut;
520           @(posedge uin_hdlc.Clk);
521                 uin_hdlc.ReadEnable = 1'b0;
522         endtask
523
524         task InsertFlagOrAbort(int flag);
525             @(posedge uin_hdlc.Clk);
526                 uin_hdlc.Rx = 1'b0;
527             @(posedge uin_hdlc.Clk);
528                 uin_hdlc.Rx = 1'b1;
529             @(posedge uin_hdlc.Clk);
530                 uin_hdlc.Rx = 1'b1;
531             @(posedge uin_hdlc.Clk);
532                 uin_hdlc.Rx = 1'b1;
533             @(posedge uin_hdlc.Clk);
534                 uin_hdlc.Rx = 1'b1;
535             @(posedge uin_hdlc.Clk);
536                 uin_hdlc.Rx = 1'b1;
537             @(posedge uin_hdlc.Clk);
538                 uin_hdlc.Rx = 1'b1;
539             @(posedge uin_hdlc.Clk);
540                 if(flag)
541                     uin_hdlc.Rx = 1'b0;
542                 else
543                     uin_hdlc.Rx = 1'b1;
544         endtask
545
546         task MakeRxStimulus(logic [127:0][7:0] Data, int Size);
547             logic [4:0] PrevData;
548             PrevData = '0;
549             for (int i = 0; i < Size; i++) begin
550                 for (int j = 0; j < 8; j++) begin
551                     if(&PrevData) begin
552                         @(posedge uin_hdlc.Clk);
553                             uin_hdlc.Rx = 1'b0;
554                             PrevData = PrevData >> 1;
555                             PrevData[4] = 1'b0;
556                     end
557
558                     @(posedge uin_hdlc.Clk);
```

```
559                         uin_hdlc.Rx = Data[i][j];
560
561                     PrevData = PrevData >> 1;
562                     PrevData[4] = Data[i][j];
563                 end
564             end
565         endtask
566
567     task MakeTxStimulus(input logic [127:0][7:0] Data, input int Size, input logic [3:0][7:0] OverflowData, input int
568         // Write data that is actually inserted
569         for (int i = 0; i < Size; i++) begin
570             WriteAddress(TXBUFF, Data[i]);
571         end
572
573         // Write overflow data
574         for (int i = 0; i < OverflowSize; i++) begin
575             WriteAddress(TXBUFF, OverflowData[i]);
576         end
577     endtask
578
579     // Flatten the 2D array of data + FCS bytes and insert zeros when 5 consequent 1's
580     task ConvertToTxStream(input logic [127:0][7:0] Data, input int Size, output logic [128*8 + 200:0] fData, output i
581         logic [4:0] prevData;
582
583         prevData = '0;
584         newSize = 0;
585         FCSSize = 0;
586
587         // Insert zeros if necessary
588         fData = '0;
589         for (int i = 0; i < Size + 2; i++) begin
590             for (int j = 0; j < 8; j++) begin
591                 // If 5 1's in a row
592                 if (&prevData) begin
593                     if (i <= Size) begin
594                         fData[newSize] = 1'b0;
595                         newSize++;
596                     end else begin
597                         fData[newSize + FCSSize] = 1'b0;
598                         FCSSize++;
599                     end
600
601                     prevData = prevData >> 1;
602                     prevData[4] = 1'b0;
603                 end
604
605                 if (i <= Size) begin
606                     fData[newSize] = Data[i][j];
607                     newSize++;
608                 end else begin
609                     fData[newSize + FCSSize] = Data[i][j];
610                     FCSSize++;
611                 end
612
613                 prevData = prevData >> 1;
614                 prevData[4] = Data[i][j];
```

```
615             end
616         end
617
618     endtask
619
620         task Receive(int Size, int Abort, int FCSerr, int NonByteAligned, int Overflow, int Drop, int SkipRead);
621             logic [127:0][7:0] ReceiveData;
622             logic       [15:0] FCSBytes;
623             logic   [2:0][7:0] OverflowData;
624             string msg;
625
626             if(Abort)
627                 msg = "- Abort";
628             else if(FCSerr)
629                 msg = "- FCS error";
630             else if(NonByteAligned)
631                 msg = "- Non-byte aligned";
632             else if(Overflow)
633                 msg = "- Overflow";
634             else if(Drop)
635                 msg = "- Drop";
636             else if(SkipRead)
637                 msg = "- Skip read";
638             else
639                 msg = "- Normal";
640
641         $display("****************************************************************");
642             $display("%t - Starting task Receive %s", $time, msg);
643             $display("****************************************************************");
644
645             for (int i = 0; i < Size; i++) begin
646                 ReceiveData[i] = $urandom;
647             end
648             ReceiveData[Size]   = '0;
649             ReceiveData[Size+1] = '0;
650
651             //Calculate FCS bits;
652             GenerateFCSBytes(ReceiveData, Size, FCSBytes);
653
654         if (FCSerr) begin
655             FCSBytes[7:0]  = FCSBytes[7:0] ^ 8'b11111111;
656             FCSBytes[15:8] = FCSBytes[15:8] ^ 8'b11111111;
657         end
658
659         ReceiveData[Size] = FCSBytes[7:0];
660         ReceiveData[Size+1] = FCSBytes[15:8];
661
662             //Enable FCS
663             if(!Overflow && !NonByteAligned)
664                 WriteAddress(RXSC, 8'h20);
665             else
666                 WriteAddress(RXSC, 8'h00);
667
668             //Generate stimulus
669             InsertFlagOrAbort(1);
670
```

```
671             MakeRxStimulus(ReceiveData, Size + 2);
672
673             if(Overflow) begin
674                 OverflowData[0] = 8'h44;
675                 OverflowData[1] = 8'hBB;
676                 OverflowData[2] = 8'hCC;
677                 MakeRxStimulus(OverflowData, 3);
678             end
679
680         if (NonByteAligned) begin
681             @(posedge uin_hdlc.Clk);
682                 uin_hdlc.Rx = 1'b1;
683             @(posedge uin_hdlc.Clk);
684                 uin_hdlc.Rx = 1'b0;
685             @(posedge uin_hdlc.Clk);
686                 uin_hdlc.Rx = 1'b0;
687         end
688
689             if(Abort) begin
690                 InsertFlagOrAbort(0);
691             end else begin
692                 InsertFlagOrAbort(1);
693             end
694
695             @(posedge uin_hdlc.Clk);
696                 uin_hdlc.Rx = 1'b1;
697
698             repeat(8)
699                 @(posedge uin_hdlc.Clk);
700
701             if(Abort)
702                 VerifyAbortReceive(ReceiveData, Size);
703             else if(Overflow)
704                 VerifyOverflowReceive(ReceiveData, Size);
705             else if(Drop)
706                 VerifyDropReceive(ReceiveData, Size);
707             else if(FCSerr)
708                 VerifyFCSErrReceive(ReceiveData, Size);
709             else if(NonByteAligned)
710                 VerifyNonByteAlignedReceive(ReceiveData, Size);
711             else if(!SkipRead)
712                 VerifyNormalReceive(ReceiveData, Size);
713
714             #5000ns;
715         endtask
716
717         task Transmit(int Size, int Abort, int Overflow);
718             logic [127:0][7:0] TransmitData;
719             logic   [2:0][7:0] OverflowData;
720             logic       [15:0] FCSBytes;
721         logic     [1228:0] fData;
722         int                NewSize;
723         int                FCSSize;
724             string msg;
725
726             if(Abort)
```

```
727                 msg = "- Abort";
728             else if(Overflow)
729                 msg = "- Overflow";
730             else
731                 msg = "- Normal";
732 
733         $display("************************************************************");
734             $display("%t - Starting task Transmit %s", $time, msg);
735             $display("************************************************************");
736 
737         // Generate random data
738             for (int i = 0; i < Size; i++) begin
739                 TransmitData[i] = $urandom;
740             end
741         // Set FCS bytes to 0
742         TransmitData[Size] = '0;
743         TransmitData[Size + 1] = '0;
744 
745 
746             //Calculate FCS bits;
747             GenerateFCSBytes(TransmitData, Size, FCSBytes);
748         // Insert the updated FCS bytes
749             TransmitData[Size] = FCSBytes[7:0];
750             TransmitData[Size+1]   = FCSBytes[15:8];
751 
752         // Flatten data and insert zeros
753         ConvertToTxStream(TransmitData, Size, fData, NewSize, FCSSize);
754 
755             if(Overflow) begin
756                 OverflowData[0] = 8'h44;
757                 OverflowData[1] = 8'hBB;
758                 OverflowData[2] = 8'hCC;
759             MakeTxStimulus(TransmitData, Size, OverflowData, 3);
760         end else begin
761             MakeTxStimulus(TransmitData, Size, OverflowData, 0);
762         end
763 
764         // Assert that Tx_Overflow is asserted
765         if (Overflow) begin
766             logic [7:0] ReadData;
767             ReadAddress(TXSC, ReadData);
768             assert (ReadData == 8'h10) begin
769                 $display("PASS: Tx_Overflow asserted");
770             end else begin
771                 $error("FAIL: Tx_Overflow not asserted");
772                 TbErrorCnt++;
773             end
774         end
775 
776         // Start transmission
777         WriteAddress(TXSC, 8'h02);
778 
779         // Wait for the beginning of the first flag
780         @(negedge uin_hdlc.Tx);
781 
782             if(Abort)
```

```
783                 VerifyAbortTransmit(fData, NewSize);
784             else
785                 VerifyNonAbortTransmit(fData, NewSize, FCSSize);
786 
787         #5000ns;
788 
789         @(posedge uin_hdlc.Clk);
790         uin_hdlc.Rst = 1'b0;
791 
792         repeat(100)
793             @(posedge uin_hdlc.Clk);
794 
795         uin_hdlc.Rst = 1'b1;
796         @(posedge uin_hdlc.Clk);
797     endtask
798 
799         task GenerateFCSBytes(logic [127:0][7:0] data, int size, output logic[15:0] FCSBytes);
800             logic [23:0] CheckReg;
801             CheckReg[15:8]  = data[1];
802             CheckReg[7:0]   = data[0];
803             for(int i = 2; i < size+2; i++) begin
804                 CheckReg[23:16] = data[i];
805                 for(int j = 0; j < 8; j++) begin
806                     if(CheckReg[0]) begin
807                         CheckReg[0]    = CheckReg[0] ^ 1;
808                         CheckReg[1]    = CheckReg[1] ^ 1;
809                         CheckReg[13:2] = CheckReg[13:2];
810                         CheckReg[14]   = CheckReg[14] ^ 1;
811                         CheckReg[15]   = CheckReg[15];
812                         CheckReg[16]   = CheckReg[16] ^1;
813                     end
814                     CheckReg = CheckReg >> 1;
815                 end
816             end
817             FCSBytes = CheckReg;
818         endtask
819 
820 endprogram
```

B assertions_hdlc.sv

```
1  ////////////////////////////////////////////////
2  // Title:   assertions_hdlc
3  // Author:
4  // Date:
5  ////////////////////////////////////////////////
6 
7  /* The assertions_hdlc module is a test module containing the concurrent
8     assertions. It is used by binding the signals of assertions_hdlc to the
9     corresponding signals in the test_hdlc testbench. This is already done in
10    bind_hdlc.sv
11 
12    For this exercise you will write concurrent assertions for the Rx module:
13    - Verify that Rx_FlagDetect is asserted two cycles after a flag is received
14    - Verify that Rx_AbortSignal is asserted after receiving an abort flag
```

```
15  */
16
17  module assertions_hdlc (
18      output int                 ErrCntAssertions,
19      input  logic               Clk,
20      input  logic               Rst,
21      input  logic [2:0]         Address,
22      input  logic               WriteEnable,
23      input  logic               ReadEnable,
24      input  logic [7:0]         DataIn,
25      input  logic [7:0]         DataOut,
26      input  logic               Rx,
27      input  logic               RxEN,
28      input  logic               Rx_Ready,
29      input  logic               Rx_ValidFrame,
30      input  logic               Rx_WrBuff,
31      input  logic               Rx_EoF,
32      input  logic               Rx_AbortSignal,
33      input  logic               Rx_StartZeroDetect,
34      input  logic               Rx_FrameError,
35      input  logic               Rx_StartFCS,
36      input  logic               Rx_StopFCS,
37      input  logic [7:0]         Rx_Data,
38      input  logic               Rx_NewByte,
39      input  logic               Rx_FlagDetect,
40      input  logic               Rx_AbortDetect,
41      input  logic               RxD,
42      input  logic               Rx_FCSerr,
43      input  logic [7:0]         Rx_FrameSize,
44      input  logic               Rx_Overflow,
45      input  logic [7:0]         Rx_DataBuffOut,
46      input  logic               Rx_FCSen,
47      input  logic               Rx_RdBuff,
48      input  logic               Rx_Drop,
49      input  logic               Tx,
50      input  logic               TxEN,
51      input  logic               Tx_Done,
52      input  logic               Tx_ValidFrame,
53      input  logic               Tx_AbortedTrans,
54      input  logic               Tx_WriteFCS,
55      input  logic               Tx_InitZero,
56      input  logic               Tx_StartFCS,
57      input  logic               Tx_RdBuff,
58      input  logic               Tx_NewByte,
59      input  logic               Tx_FCSDone,
60      input  logic [7:0]         Tx_Data,
61      input  logic               Tx_Full,
62      input  logic               Tx_DataAvail,
63      input  logic [7:0]         Tx_FrameSize,
64      input  logic [127:0][7:0] Tx_DataArray,
65      input  logic [7:0]         Tx_DataOutBuff,
66      input  logic               Tx_WrBuff,
67      input  logic               Tx_Enable,
68      input  logic [7:0]         Tx_DataInBuff,
69      input  logic               Tx_AbortFrame
70  );
```

```
71
72      initial begin
73              ErrCntAssertions = 0;
74          end
75
76          /*****************************************
77          *                Sequences                *
78          *****************************************/
79
80          sequence idle(signal);
81              signal [*8];
82          endsequence;
83
84          sequence flag(signal);
85              !signal ##1 signal [*6] ##1 !signal;
86          endsequence
87
88          sequence abort(signal);
89              !signal ##1 signal [*7];
90      endsequence
91
92          sequence zero(signal);
93              !signal ##1 signal [*5] ##1 !signal;
94          endsequence
95
96          sequence Rx_DataZero;
97          ( Rx_Data ==? 8'b11111xxx) or
98          ( Rx_Data ==? 8'bx11111xx) or
99          ( Rx_Data ==? 8'bxx11111x) or
100         ( Rx_Data ==? 8'bxxx11111) or
101         ((Rx_Data ==? 8'bxxxx1111) && ($past(Rx_Data, 8) ==? 8'b1xxxxxxx)) or
102         ((Rx_Data ==? 8'bxxxxx111) && ($past(Rx_Data, 8) ==? 8'b11xxxxxx)) or
103         ((Rx_Data ==? 8'bxxxxxx11) && ($past(Rx_Data, 8) ==? 8'b111xxxxx)) or
104         ((Rx_Data ==? 8'bxxxxxxx1) && ($past(Rx_Data, 8) ==? 8'b1111xxxx));
105     endsequence
106
107         sequence Tx_DataZero;
108         ( Tx_Data ==? 8'b111110xx) or
109         ( Tx_Data ==? 8'bx111110x) or
110         ( Tx_Data ==? 8'bxx111110) or
111         ((Tx_Data ==? 8'bxxx11111) && ($past(Tx_Data, 8) ==? 8'b0xxxxxxx)) or
112         ((Tx_Data ==? 8'bxxxx1111) && ($past(Tx_Data, 8) ==? 8'b10xxxxxx)) or
113         ((Tx_Data ==? 8'bxxxxx111) && ($past(Tx_Data, 8) ==? 8'b110xxxxx)) or
114         ((Tx_Data ==? 8'bxxxxxx11) && ($past(Tx_Data, 8) ==? 8'b1110xxxx)) or
115         ((Tx_Data ==? 8'bxxxxxxx1) && ($past(Tx_Data, 8) ==? 8'b11110xxx));
116     endsequence
117
118         /*****************************************
119         *                Properties               *
120         *****************************************/
121
122         // 3. Correct bits set in RX status/control register after receiving frame.
123         property p_Rx_Status;
124             @(posedge Clk) disable iff(!Rst) $rose(Rx_EoF) |->
125                     if (Rx_FrameError)
126                             !Rx_Ready && !Rx_Overflow && !Rx_AbortSignal &&  Rx_FrameError
```

```
                else if (Rx_AbortSignal && Rx_Overflow)
                        Rx_Ready &&  Rx_Overflow &&  Rx_AbortSignal &&  Rx_FrameError
                else if (Rx_AbortSignal)
                        Rx_Ready && !Rx_Overflow &&  Rx_AbortSignal && !Rx_FrameError
                else if (Rx_Overflow)
                        Rx_Ready &&  Rx_Overflow && !Rx_AbortSignal && !Rx_FrameError
                else
                        Rx_Ready && !Rx_Overflow && !Rx_AbortSignal && !Rx_FrameError
    endproperty

        // 5. Start and end of frame pattern generation.
        property p_Tx_FramePattern;
            @(posedge Clk) disable iff (!Rst) !$stable(Tx_ValidFrame) ##0 $past(!Tx_AbortFrame, 2) |-> ##[1:2] flag(Tx
        endproperty

        property p_Rx_FramePattern;
            @(posedge Clk) flag(Rx) |-> ##2 Rx_FlagDetect;
        endproperty

        // 6. Zero insertion and removal of transparent transmission.
        property p_Tx_InsertZero;
        @(posedge Clk) disable iff (!Rst || !Tx_ValidFrame) $rose(Tx_NewByte) ##0 Tx_DataZero |-> ##[13:22] zero(Tx);
        endproperty

        property p_Rx_RemoveZero;
            @(posedge Clk) disable iff (!Rst) (zero(Rx) and Rx_ValidFrame [*6]) |-> ##[9:17] Rx_NewByte ##1 Rx_DataZer
        endproperty

        // 7. Idle pattern generation and checking
        property p_Tx_IdlePattern;
            @(posedge Clk) disable iff (!Rst) !Tx_ValidFrame && Tx_FrameSize == 8'b0 |-> idle(Tx);
        endproperty

        property p_Rx_IdlePattern;
              @(posedge Clk) disable iff (!Rst) idle(Rx) |=> !Rx_FlagDetect;
        endproperty

        // 8. Abort pattern generation and checking.
        property p_Tx_AbortPattern;
            @(posedge Clk) disable iff (!Rst) $rose(Tx_AbortFrame) |-> ##4 abort(Tx);
        endproperty

        property p_Rx_AbortPattern;
            @(posedge Clk) disable iff (!Rst) abort(Rx) and (!Rx_ValidFrame [*7]) |-> ##2 $rose(Rx_AbortDetect);
        endproperty

        // 9. When aborting frame during transmission, Tx_AbortedTrans should be asserted
        property p_Tx_AbortSignal;
            @(posedge Clk) disable iff (!Rst) $rose(Tx_AbortFrame) && Tx_DataAvail ##1 $fell(Tx_AbortFrame) |=>  $rose
        endproperty

        // 10. Abort pattern detected during valid frame should generate Rx_AbortSignal
        property p_Rx_AbortSignal;
            @(posedge Clk) disable iff (!Rst) Rx_ValidFrame && Rx_AbortDetect |=> Rx_AbortSignal;
        endproperty

```

```
183         // 12. When a whole RX frame has been received, check if end of frame is generated
184         property p_Rx_EndOfFrame;
185             @(posedge Clk) disable iff (!Rst) $fell(Rx_ValidFrame) |=> $rose(Rx_EoF);
186         endproperty
187 
188         // 13. When receiving more than 128 bytes, Rx_Overflow should be asserted
189         property p_Rx_Overflow;
190             @(posedge Clk) disable iff (!Rst || !Rx_ValidFrame)  $rose(Rx_ValidFrame) ##0 ($rose(Rx_NewByte) [->129])
191         endproperty
192 
193     // 14. Rx_FrameSize should equal the number of bytes received in a frame.
194         property p_Rx_FrameSize;
195             int bytes = 0;
196             @(posedge Clk) disable iff (!Rst) $rose(Rx_ValidFrame) ##0 ((##[7:9] $rose(Rx_NewByte), bytes++) [*1:128])
197         endproperty
198 
199         // 15. Rx_Ready should indicate byte(s) in RX Buffer is ready to be read
200         property p_Rx_Ready;
201             @(posedge Clk) disable iff (!Rst) $rose(Rx_Ready) |-> $rose(Rx_EoF) and !Rx_ValidFrame;
202         endproperty
203 
204         // 16. Non-byte aligned data or errors in FCS checking should result in frame error
205         property p_Rx_FCSerr;
206             @(posedge Clk) disable iff (!Rst) $rose(Rx_FCSerr) && Rx_FCSen |=> $rose(Rx_FrameError);
207         endproperty
208 
209     // 17. Tx_Done should be asserted when entire TX buffer has been read for transmission
210         property p_Tx_Done;
211             @(posedge Clk) disable iff (!Rst) $fell(Tx_DataAvail) |-> $past(Tx_Done, 1);
212         endproperty
213 
214         // 18. Tx_Full should be asserted after writing 126 or more bytes to the TX buffer (overflow)
215         property p_Tx_Full;
216         @(posedge Clk) disable iff (!Rst) $fell(Tx_Done) ##0 (($rose(Tx_WrBuff) [->1]) [*125]) ##0 !Tx_Done |=> Tx_Ful
217         endproperty
218 
219         /*****************************************
220         *                Assertions                *
221         *****************************************/
222 
223         Rx_Status_Assert : assert property (p_Rx_Status) begin
224             $display("PASS: Rx_SC correct");
225         end else begin
226                 $error("FAIL: Rx_SC incorrect");
227                 ErrCntAssertions++;
228         end
229 
230         Tx_FramePattern_Assert : assert property (p_Tx_FramePattern) begin
231             $display("PASS: Flag sent");
232         end else begin
233             $error("FAIL: Flag not sent");
234             ErrCntAssertions++;
235         end
236 
237         Rx_FramePattern_Assert : assert property (p_Rx_FramePattern) begin
238             $display("PASS: Flag received");
```

```
        end else begin
            $error("FAIL: Flag not received");
            ErrCntAssertions++;
        end

        Tx_InsertZero_Assert : assert property (p_Tx_InsertZero) begin
            $display("PASS: Zero inserted");
        end else begin
            $error("FAIL: Zero not inserted");
            ErrCntAssertions++;
        end

        Rx_RemoveZero_Assert : assert property (p_Rx_RemoveZero) begin
        $display("PASS: Zero removed");
        end else begin
        $error("FAIL: Zero not removed");
            ErrCntAssertions++;
        end

        Tx_IdlePattern_Assert : assert property (p_Tx_IdlePattern) else begin
            $error("FAIL: Idle pattern not transmitted");
            ErrCntAssertions++;
        end

        Rx_IdlePattern_Assert : assert property (p_Rx_IdlePattern) else begin
            $error("FAIL: Idle pattern not received");
            ErrCntAssertions++;
        end

        Tx_AbortPattern_Assert : assert property (p_Tx_AbortPattern) begin
            $display("PASS: Abort transmitted");
        end else begin
            $error("FAIL: Abort not transmitted");
            ErrCntAssertions++;
        end

        Rx_AbortPattern_Assert : assert property (p_Rx_AbortPattern) begin
            $display("PASS: Abort received");
        end else begin
            $error("FAIL: Abort not received");
            ErrCntAssertions++;
        end

        Tx_AbortSignal_Assert : assert property (p_Tx_AbortSignal) begin
            $display("PASS: Tx_AbortedTrans asserted");
        end else begin
            $error("FAIL: Tx_AbortedTrans not asserted");
            ErrCntAssertions++;
        end

        Rx_AbortSignal_Assert : assert property (p_Rx_AbortSignal) begin
            $display("PASS: Rx_AbortSignal asserted");
        end else begin
            $error("FAIL: Rx_AbortSignal not asserted");
            ErrCntAssertions++;
        end
```

```
295
296         Rx_EoF_Assert : assert property (p_Rx_EndOfFrame) begin
297             $display("PASS: Rx_EoF asserted");
298         end else begin
299             $error("FAIL: Rx_EoF not asserted");
300             ErrCntAssertions++;
301         end
302
303         Rx_Overflow_Assert : assert property (p_Rx_Overflow) begin
304             $display("PASS: Rx_Overflow asserted");
305     end else begin
306             $error("FAIL: Rx_Overflow not asserted");
307             ErrCntAssertions++;
308         end
309
310         Rx_FrameSize_Assert : assert property (p_Rx_FrameSize) begin
311             $display("PASS: Rx_FrameSize correct");
312         end else begin
313             $error("FAIL: Rx_FrameSize incorrect");
314             ErrCntAssertions++;
315         end
316
317         Rx_Ready_Assert : assert property (p_Rx_Ready) begin
318             $display("PASS: Data ready");
319         end else begin
320             $error("FAIL: Data not ready");
321             ErrCntAssertions++;
322         end
323
324         Rx_FCSerr_Assert : assert property (p_Rx_FCSerr) begin
325             $display("PASS: Rx_FrameError asserted");
326         end else begin
327             $error("FAIL: Rx_FrameError not asserted");
328             ErrCntAssertions++;
329         end
330
331         Tx_Done_Assert : assert property (p_Tx_Done) begin
332             $display("PASS: Tx_Done asserted");
333         end else begin
334             $error("FAIL: Tx_Done not asserted");
335             ErrCntAssertions++;
336         end
337
338         Tx_Full_Assert : assert property (p_Tx_Full) begin
339             $display("PASS: Tx_Full asserted");
340         end else begin
341             $error("FAIL: Tx_Full not asserted");
342             ErrCntAssertions++;
343         end
344
345 endmodule
```

C Coverage Report

COVERGROUP COVERAGE:

Covergroup	Metric	Goal	Status
TYPE /test_hdlc/u_testPr/hdlc_cg	89.0%	100	Uncovered
covered/total bins:	991	1309	
missing/total bins:	318	1309	
% Hit:	75.7%	100	
Coverpoint hdlc_cg::Address	100.0%	100	Covered
covered/total bins:	5	5	
missing/total bins:	0	5	
% Hit:	100.0%	100	
Coverpoint hdlc_cg::DataIn	92.1%	100	Uncovered
covered/total bins:	236	256	
missing/total bins:	20	256	
% Hit:	92.1%	100	
Coverpoint hdlc_cg::DataOut	87.8%	100	Uncovered
covered/total bins:	225	256	
missing/total bins:	31	256	
% Hit:	87.8%	100	
Coverpoint hdlc_cg::RxData	98.0%	100	Uncovered
covered/total bins:	251	256	
missing/total bins:	5	256	
% Hit:	98.0%	100	
Coverpoint hdlc_cg::RxFrameSize	7.8%	100	Uncovered
covered/total bins:	10	127	
missing/total bins:	117	127	
% Hit:	7.8%	100	
Coverpoint hdlc_cg::RxValidFrame	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
Coverpoint hdlc_cg::RxAbortSignal	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
Coverpoint hdlc_cg::RxReady	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
Coverpoint hdlc_cg::RxEoF	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
Coverpoint hdlc_cg::RxOverflow	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	

```
        % Hit:                                    100.0%        100
    Coverpoint hdlc_cg::RxFCSErr                  100.0%        100    Covered
        covered/total bins:                            2          2
        missing/total bins:                            0          2
        % Hit:                                    100.0%        100
    Coverpoint hdlc_cg::RxFrameError              100.0%        100    Covered
        covered/total bins:                            2          2
        missing/total bins:                            0          2
        % Hit:                                    100.0%        100
    Coverpoint hdlc_cg::RxDrop                    100.0%        100    Covered
        covered/total bins:                            2          2
        missing/total bins:                            0          2
        % Hit:                                    100.0%        100
    Coverpoint hdlc_cg::TxValidFrame              100.0%        100    Covered
        covered/total bins:                            2          2
        missing/total bins:                            0          2
        % Hit:                                    100.0%        100
    Coverpoint hdlc_cg::TxAbortedTrans            100.0%        100    Covered
        covered/total bins:                            2          2
        missing/total bins:                            0          2
        % Hit:                                    100.0%        100
    Coverpoint hdlc_cg::TxData                     90.6%        100    Uncovered
        covered/total bins:                          232        256
        missing/total bins:                           24        256
        % Hit:                                     90.6%        100
    Coverpoint hdlc_cg::TxFull                    100.0%        100    Covered
        covered/total bins:                            2          2
        missing/total bins:                            0          2
        % Hit:                                    100.0%        100
    Coverpoint hdlc_cg::TxFrameSize                4.7%        100    Uncovered
        covered/total bins:                            6        127
        missing/total bins:                          121        127
        % Hit:                                     4.7%        100
    Coverpoint hdlc_cg::TxEnable                  100.0%        100    Covered
        covered/total bins:                            2          2
        missing/total bins:                            0          2
        % Hit:                                    100.0%        100
    Coverpoint hdlc_cg::TxAbortFrame              100.0%        100    Covered
        covered/total bins:                            2          2
        missing/total bins:                            0          2
        % Hit:                                    100.0%        100
Covergroup instance \/test_hdlc/u_testPr/inst_hdlc_cg
                                                   89.0%        100    Uncovered
    covered/total bins:                              991       1309
    missing/total bins:                              318       1309
    % Hit:                                        75.7%        100
    Coverpoint Address                            100.0%        100    Covered
        covered/total bins:                            5          5
        missing/total bins:                            0          5
        % Hit:                                    100.0%        100
```

ignore_bin Invalid	0		ZERO
bin Tx_SC	13116	1	Covered
bin Tx_Buff	1530	1	Covered
bin Rx_SC	8685	1	Covered
bin Rx_Buff	1152	1	Covered
bin Rx_Len	14	1	Covered
Coverpoint DataIn	92.1%	100	Uncovered
covered/total bins:	236	256	
missing/total bins:	20	256	
% Hit:	92.1%	100	
bin DataIn[0]	3523	1	Covered
bin DataIn[1]	4	1	Covered
bin DataIn[2]	12874	1	Covered
bin DataIn[3]	12	1	Covered
bin DataIn[4]	264	1	Covered
bin DataIn[5]	0	1	ZERO
bin DataIn[6]	2	1	Covered
bin DataIn[7]	2	1	Covered
bin DataIn[8]	2	1	Covered
bin DataIn[9]	6	1	Covered
bin DataIn[10]	4	1	Covered
bin DataIn[11]	4	1	Covered
bin DataIn[12]	0	1	ZERO
bin DataIn[13]	6	1	Covered
bin DataIn[14]	14	1	Covered
bin DataIn[15]	4	1	Covered
bin DataIn[16]	2	1	Covered
bin DataIn[17]	4	1	Covered
bin DataIn[18]	4	1	Covered
bin DataIn[19]	14	1	Covered
bin DataIn[20]	2	1	Covered
bin DataIn[21]	10	1	Covered
bin DataIn[22]	0	1	ZERO
bin DataIn[23]	10	1	Covered
bin DataIn[24]	0	1	ZERO
bin DataIn[25]	0	1	ZERO
bin DataIn[26]	2	1	Covered
bin DataIn[27]	6	1	Covered
bin DataIn[28]	8	1	Covered
bin DataIn[29]	14	1	Covered
bin DataIn[30]	4	1	Covered
bin DataIn[31]	0	1	ZERO
bin DataIn[32]	6326	1	Covered
bin DataIn[33]	10	1	Covered
bin DataIn[34]	10	1	Covered
bin DataIn[35]	4	1	Covered
bin DataIn[36]	8	1	Covered
bin DataIn[37]	6	1	Covered
bin DataIn[38]	8	1	Covered
bin DataIn[39]	6	1	Covered

bin DataIn[40]	2	1	Covered
bin DataIn[41]	0	1	ZERO
bin DataIn[42]	2	1	Covered
bin DataIn[43]	8	1	Covered
bin DataIn[44]	8	1	Covered
bin DataIn[45]	2	1	Covered
bin DataIn[46]	8	1	Covered
bin DataIn[47]	6	1	Covered
bin DataIn[48]	8	1	Covered
bin DataIn[49]	4	1	Covered
bin DataIn[50]	2	1	Covered
bin DataIn[51]	8	1	Covered
bin DataIn[52]	0	1	ZERO
bin DataIn[53]	6	1	Covered
bin DataIn[54]	4	1	Covered
bin DataIn[55]	4	1	Covered
bin DataIn[56]	4	1	Covered
bin DataIn[57]	2	1	Covered
bin DataIn[58]	2	1	Covered
bin DataIn[59]	6	1	Covered
bin DataIn[60]	10	1	Covered
bin DataIn[61]	4	1	Covered
bin DataIn[62]	10	1	Covered
bin DataIn[63]	6	1	Covered
bin DataIn[64]	2	1	Covered
bin DataIn[65]	8	1	Covered
bin DataIn[66]	10	1	Covered
bin DataIn[67]	4	1	Covered
bin DataIn[68]	12	1	Covered
bin DataIn[69]	2	1	Covered
bin DataIn[70]	16	1	Covered
bin DataIn[71]	4	1	Covered
bin DataIn[72]	2	1	Covered
bin DataIn[73]	6	1	Covered
bin DataIn[74]	4	1	Covered
bin DataIn[75]	10	1	Covered
bin DataIn[76]	4	1	Covered
bin DataIn[77]	12	1	Covered
bin DataIn[78]	6	1	Covered
bin DataIn[79]	2	1	Covered
bin DataIn[80]	10	1	Covered
bin DataIn[81]	10	1	Covered
bin DataIn[82]	8	1	Covered
bin DataIn[83]	10	1	Covered
bin DataIn[84]	8	1	Covered
bin DataIn[85]	0	1	ZERO
bin DataIn[86]	2	1	Covered
bin DataIn[87]	8	1	Covered
bin DataIn[88]	10	1	Covered
bin DataIn[89]	6	1	Covered

bin DataIn[90]	8	1	Covered
bin DataIn[91]	10	1	Covered
bin DataIn[92]	4	1	Covered
bin DataIn[93]	4	1	Covered
bin DataIn[94]	10	1	Covered
bin DataIn[95]	8	1	Covered
bin DataIn[96]	8	1	Covered
bin DataIn[97]	6	1	Covered
bin DataIn[98]	2	1	Covered
bin DataIn[99]	6	1	Covered
bin DataIn[100]	6	1	Covered
bin DataIn[101]	2	1	Covered
bin DataIn[102]	4	1	Covered
bin DataIn[103]	2	1	Covered
bin DataIn[104]	4	1	Covered
bin DataIn[105]	6	1	Covered
bin DataIn[106]	0	1	ZERO
bin DataIn[107]	6	1	Covered
bin DataIn[108]	4	1	Covered
bin DataIn[109]	8	1	Covered
bin DataIn[110]	4	1	Covered
bin DataIn[111]	4	1	Covered
bin DataIn[112]	2	1	Covered
bin DataIn[113]	0	1	ZERO
bin DataIn[114]	8	1	Covered
bin DataIn[115]	10	1	Covered
bin DataIn[116]	2	1	Covered
bin DataIn[117]	6	1	Covered
bin DataIn[118]	10	1	Covered
bin DataIn[119]	10	1	Covered
bin DataIn[120]	6	1	Covered
bin DataIn[121]	6	1	Covered
bin DataIn[122]	4	1	Covered
bin DataIn[123]	4	1	Covered
bin DataIn[124]	2	1	Covered
bin DataIn[125]	8	1	Covered
bin DataIn[126]	6	1	Covered
bin DataIn[127]	0	1	ZERO
bin DataIn[128]	8	1	Covered
bin DataIn[129]	2	1	Covered
bin DataIn[130]	4	1	Covered
bin DataIn[131]	10	1	Covered
bin DataIn[132]	6	1	Covered
bin DataIn[133]	4	1	Covered
bin DataIn[134]	4	1	Covered
bin DataIn[135]	8	1	Covered
bin DataIn[136]	4	1	Covered
bin DataIn[137]	6	1	Covered
bin DataIn[138]	6	1	Covered
bin DataIn[139]	4	1	Covered

bin DataIn[140]	8	1	Covered
bin DataIn[141]	4	1	Covered
bin DataIn[142]	4	1	Covered
bin DataIn[143]	2	1	Covered
bin DataIn[144]	18	1	Covered
bin DataIn[145]	4	1	Covered
bin DataIn[146]	8	1	Covered
bin DataIn[147]	14	1	Covered
bin DataIn[148]	6	1	Covered
bin DataIn[149]	12	1	Covered
bin DataIn[150]	16	1	Covered
bin DataIn[151]	6	1	Covered
bin DataIn[152]	8	1	Covered
bin DataIn[153]	4	1	Covered
bin DataIn[154]	6	1	Covered
bin DataIn[155]	2	1	Covered
bin DataIn[156]	14	1	Covered
bin DataIn[157]	10	1	Covered
bin DataIn[158]	8	1	Covered
bin DataIn[159]	8	1	Covered
bin DataIn[160]	12	1	Covered
bin DataIn[161]	8	1	Covered
bin DataIn[162]	4	1	Covered
bin DataIn[163]	2	1	Covered
bin DataIn[164]	6	1	Covered
bin DataIn[165]	0	1	ZERO
bin DataIn[166]	4	1	Covered
bin DataIn[167]	8	1	Covered
bin DataIn[168]	6	1	Covered
bin DataIn[169]	4	1	Covered
bin DataIn[170]	4	1	Covered
bin DataIn[171]	8	1	Covered
bin DataIn[172]	12	1	Covered
bin DataIn[173]	12	1	Covered
bin DataIn[174]	8	1	Covered
bin DataIn[175]	2	1	Covered
bin DataIn[176]	6	1	Covered
bin DataIn[177]	6	1	Covered
bin DataIn[178]	8	1	Covered
bin DataIn[179]	8	1	Covered
bin DataIn[180]	10	1	Covered
bin DataIn[181]	10	1	Covered
bin DataIn[182]	8	1	Covered
bin DataIn[183]	8	1	Covered
bin DataIn[184]	10	1	Covered
bin DataIn[185]	6	1	Covered
bin DataIn[186]	12	1	Covered
bin DataIn[187]	10	1	Covered
bin DataIn[188]	10	1	Covered
bin DataIn[189]	8	1	Covered

bin DataIn[190]	8	1	Covered
bin DataIn[191]	8	1	Covered
bin DataIn[192]	6	1	Covered
bin DataIn[193]	2	1	Covered
bin DataIn[194]	0	1	ZERO
bin DataIn[195]	4	1	Covered
bin DataIn[196]	8	1	Covered
bin DataIn[197]	6	1	Covered
bin DataIn[198]	10	1	Covered
bin DataIn[199]	8	1	Covered
bin DataIn[200]	6	1	Covered
bin DataIn[201]	2	1	Covered
bin DataIn[202]	4	1	Covered
bin DataIn[203]	8	1	Covered
bin DataIn[204]	8	1	Covered
bin DataIn[205]	6	1	Covered
bin DataIn[206]	12	1	Covered
bin DataIn[207]	4	1	Covered
bin DataIn[208]	6	1	Covered
bin DataIn[209]	10	1	Covered
bin DataIn[210]	8	1	Covered
bin DataIn[211]	10	1	Covered
bin DataIn[212]	12	1	Covered
bin DataIn[213]	0	1	ZERO
bin DataIn[214]	0	1	ZERO
bin DataIn[215]	8	1	Covered
bin DataIn[216]	8	1	Covered
bin DataIn[217]	10	1	Covered
bin DataIn[218]	2	1	Covered
bin DataIn[219]	2	1	Covered
bin DataIn[220]	6	1	Covered
bin DataIn[221]	4	1	Covered
bin DataIn[222]	4	1	Covered
bin DataIn[223]	8	1	Covered
bin DataIn[224]	6	1	Covered
bin DataIn[225]	8	1	Covered
bin DataIn[226]	4	1	Covered
bin DataIn[227]	8	1	Covered
bin DataIn[228]	4	1	Covered
bin DataIn[229]	0	1	ZERO
bin DataIn[230]	10	1	Covered
bin DataIn[231]	4	1	Covered
bin DataIn[232]	4	1	Covered
bin DataIn[233]	6	1	Covered
bin DataIn[234]	2	1	Covered
bin DataIn[235]	4	1	Covered
bin DataIn[236]	14	1	Covered
bin DataIn[237]	2	1	Covered
bin DataIn[238]	2	1	Covered
bin DataIn[239]	0	1	ZERO

bin DataIn[240]	0	1	ZERO
bin DataIn[241]	4	1	Covered
bin DataIn[242]	0	1	ZERO
bin DataIn[243]	10	1	Covered
bin DataIn[244]	8	1	Covered
bin DataIn[245]	8	1	Covered
bin DataIn[246]	4	1	Covered
bin DataIn[247]	2	1	Covered
bin DataIn[248]	8	1	Covered
bin DataIn[249]	10	1	Covered
bin DataIn[250]	8	1	Covered
bin DataIn[251]	8	1	Covered
bin DataIn[252]	8	1	Covered
bin DataIn[253]	2	1	Covered
bin DataIn[254]	6	1	Covered
bin DataIn[255]	6	1	Covered
Coverpoint DataOut	87.8%	100	Uncovered
covered/total bins:	225	256	
missing/total bins:	31	256	
% Hit:	87.8%	100	
bin DataOut[0]	23972	1	Covered
bin DataOut[1]	4	1	Covered
bin DataOut[2]	1	1	Covered
bin DataOut[3]	1	1	Covered
bin DataOut[4]	5	1	Covered
bin DataOut[5]	1	1	Covered
bin DataOut[6]	2	1	Covered
bin DataOut[7]	0	1	ZERO
bin DataOut[8]	1	1	Covered
bin DataOut[9]	4	1	Covered
bin DataOut[10]	5	1	Covered
bin DataOut[11]	3	1	Covered
bin DataOut[12]	0	1	ZERO
bin DataOut[13]	2	1	Covered
bin DataOut[14]	1	1	Covered
bin DataOut[15]	4	1	Covered
bin DataOut[16]	4	1	Covered
bin DataOut[17]	4	1	Covered
bin DataOut[18]	1	1	Covered
bin DataOut[19]	2	1	Covered
bin DataOut[20]	1	1	Covered
bin DataOut[21]	3	1	Covered
bin DataOut[22]	3	1	Covered
bin DataOut[23]	2	1	Covered
bin DataOut[24]	2	1	Covered
bin DataOut[25]	2	1	Covered
bin DataOut[26]	0	1	ZERO
bin DataOut[27]	0	1	ZERO
bin DataOut[28]	1	1	Covered
bin DataOut[29]	3	1	Covered

bin DataOut[30]	2	1	Covered
bin DataOut[31]	3	1	Covered
bin DataOut[32]	3	1	Covered
bin DataOut[33]	7	1	Covered
bin DataOut[34]	2	1	Covered
bin DataOut[35]	0	1	ZERO
bin DataOut[36]	3	1	Covered
bin DataOut[37]	2	1	Covered
bin DataOut[38]	4	1	Covered
bin DataOut[39]	4	1	Covered
bin DataOut[40]	4	1	Covered
bin DataOut[41]	1	1	Covered
bin DataOut[42]	0	1	ZERO
bin DataOut[43]	1	1	Covered
bin DataOut[44]	3	1	Covered
bin DataOut[45]	3	1	Covered
bin DataOut[46]	2	1	Covered
bin DataOut[47]	2	1	Covered
bin DataOut[48]	1	1	Covered
bin DataOut[49]	0	1	ZERO
bin DataOut[50]	3	1	Covered
bin DataOut[51]	2	1	Covered
bin DataOut[52]	1	1	Covered
bin DataOut[53]	2	1	Covered
bin DataOut[54]	3	1	Covered
bin DataOut[55]	0	1	ZERO
bin DataOut[56]	1	1	Covered
bin DataOut[57]	1	1	Covered
bin DataOut[58]	2	1	Covered
bin DataOut[59]	1	1	Covered
bin DataOut[60]	3	1	Covered
bin DataOut[61]	1	1	Covered
bin DataOut[62]	0	1	ZERO
bin DataOut[63]	2	1	Covered
bin DataOut[64]	3	1	Covered
bin DataOut[65]	1	1	Covered
bin DataOut[66]	3	1	Covered
bin DataOut[67]	0	1	ZERO
bin DataOut[68]	1	1	Covered
bin DataOut[69]	2	1	Covered
bin DataOut[70]	5	1	Covered
bin DataOut[71]	2	1	Covered
bin DataOut[72]	2	1	Covered
bin DataOut[73]	2	1	Covered
bin DataOut[74]	0	1	ZERO
bin DataOut[75]	1	1	Covered
bin DataOut[76]	2	1	Covered
bin DataOut[77]	3	1	Covered
bin DataOut[78]	1	1	Covered
bin DataOut[79]	2	1	Covered

bin DataOut[80]	2	1	Covered
bin DataOut[81]	4	1	Covered
bin DataOut[82]	1	1	Covered
bin DataOut[83]	4	1	Covered
bin DataOut[84]	6	1	Covered
bin DataOut[85]	1	1	Covered
bin DataOut[86]	3	1	Covered
bin DataOut[87]	1	1	Covered
bin DataOut[88]	2	1	Covered
bin DataOut[89]	2	1	Covered
bin DataOut[90]	0	1	ZERO
bin DataOut[91]	2	1	Covered
bin DataOut[92]	2	1	Covered
bin DataOut[93]	0	1	ZERO
bin DataOut[94]	0	1	ZERO
bin DataOut[95]	0	1	ZERO
bin DataOut[96]	3	1	Covered
bin DataOut[97]	2	1	Covered
bin DataOut[98]	3	1	Covered
bin DataOut[99]	3	1	Covered
bin DataOut[100]	3	1	Covered
bin DataOut[101]	2	1	Covered
bin DataOut[102]	1	1	Covered
bin DataOut[103]	3	1	Covered
bin DataOut[104]	0	1	ZERO
bin DataOut[105]	4	1	Covered
bin DataOut[106]	3	1	Covered
bin DataOut[107]	1	1	Covered
bin DataOut[108]	4	1	Covered
bin DataOut[109]	3	1	Covered
bin DataOut[110]	1	1	Covered
bin DataOut[111]	3	1	Covered
bin DataOut[112]	2	1	Covered
bin DataOut[113]	1	1	Covered
bin DataOut[114]	2	1	Covered
bin DataOut[115]	3	1	Covered
bin DataOut[116]	2	1	Covered
bin DataOut[117]	3	1	Covered
bin DataOut[118]	4	1	Covered
bin DataOut[119]	1	1	Covered
bin DataOut[120]	2	1	Covered
bin DataOut[121]	1	1	Covered
bin DataOut[122]	1	1	Covered
bin DataOut[123]	0	1	ZERO
bin DataOut[124]	1	1	Covered
bin DataOut[125]	2	1	Covered
bin DataOut[126]	6	1	Covered
bin DataOut[127]	0	1	ZERO
bin DataOut[128]	1	1	Covered
bin DataOut[129]	1	1	Covered

bin DataOut[130]	1	1	Covered
bin DataOut[131]	0	1	ZERO
bin DataOut[132]	2	1	Covered
bin DataOut[133]	2	1	Covered
bin DataOut[134]	4	1	Covered
bin DataOut[135]	0	1	ZERO
bin DataOut[136]	1	1	Covered
bin DataOut[137]	3	1	Covered
bin DataOut[138]	1	1	Covered
bin DataOut[139]	8	1	Covered
bin DataOut[140]	3	1	Covered
bin DataOut[141]	1	1	Covered
bin DataOut[142]	2	1	Covered
bin DataOut[143]	1	1	Covered
bin DataOut[144]	4	1	Covered
bin DataOut[145]	3	1	Covered
bin DataOut[146]	3	1	Covered
bin DataOut[147]	1	1	Covered
bin DataOut[148]	0	1	ZERO
bin DataOut[149]	2	1	Covered
bin DataOut[150]	4	1	Covered
bin DataOut[151]	2	1	Covered
bin DataOut[152]	0	1	ZERO
bin DataOut[153]	1	1	Covered
bin DataOut[154]	5	1	Covered
bin DataOut[155]	2	1	Covered
bin DataOut[156]	2	1	Covered
bin DataOut[157]	2	1	Covered
bin DataOut[158]	1	1	Covered
bin DataOut[159]	1	1	Covered
bin DataOut[160]	2	1	Covered
bin DataOut[161]	5	1	Covered
bin DataOut[162]	2	1	Covered
bin DataOut[163]	0	1	ZERO
bin DataOut[164]	2	1	Covered
bin DataOut[165]	2	1	Covered
bin DataOut[166]	2	1	Covered
bin DataOut[167]	2	1	Covered
bin DataOut[168]	3	1	Covered
bin DataOut[169]	4	1	Covered
bin DataOut[170]	2	1	Covered
bin DataOut[171]	3	1	Covered
bin DataOut[172]	1	1	Covered
bin DataOut[173]	2	1	Covered
bin DataOut[174]	3	1	Covered
bin DataOut[175]	4	1	Covered
bin DataOut[176]	0	1	ZERO
bin DataOut[177]	0	1	ZERO
bin DataOut[178]	4	1	Covered
bin DataOut[179]	1	1	Covered

bin DataOut[180]	4	1	Covered
bin DataOut[181]	4	1	Covered
bin DataOut[182]	3	1	Covered
bin DataOut[183]	4	1	Covered
bin DataOut[184]	3	1	Covered
bin DataOut[185]	0	1	ZERO
bin DataOut[186]	1	1	Covered
bin DataOut[187]	1	1	Covered
bin DataOut[188]	3	1	Covered
bin DataOut[189]	1	1	Covered
bin DataOut[190]	3	1	Covered
bin DataOut[191]	1	1	Covered
bin DataOut[192]	1	1	Covered
bin DataOut[193]	3	1	Covered
bin DataOut[194]	1	1	Covered
bin DataOut[195]	2	1	Covered
bin DataOut[196]	4	1	Covered
bin DataOut[197]	1	1	Covered
bin DataOut[198]	3	1	Covered
bin DataOut[199]	2	1	Covered
bin DataOut[200]	4	1	Covered
bin DataOut[201]	1	1	Covered
bin DataOut[202]	2	1	Covered
bin DataOut[203]	1	1	Covered
bin DataOut[204]	3	1	Covered
bin DataOut[205]	3	1	Covered
bin DataOut[206]	1	1	Covered
bin DataOut[207]	6	1	Covered
bin DataOut[208]	3	1	Covered
bin DataOut[209]	2	1	Covered
bin DataOut[210]	3	1	Covered
bin DataOut[211]	3	1	Covered
bin DataOut[212]	4	1	Covered
bin DataOut[213]	2	1	Covered
bin DataOut[214]	1	1	Covered
bin DataOut[215]	2	1	Covered
bin DataOut[216]	0	1	ZERO
bin DataOut[217]	2	1	Covered
bin DataOut[218]	2	1	Covered
bin DataOut[219]	2	1	Covered
bin DataOut[220]	3	1	Covered
bin DataOut[221]	4	1	Covered
bin DataOut[222]	1	1	Covered
bin DataOut[223]	2	1	Covered
bin DataOut[224]	4	1	Covered
bin DataOut[225]	4	1	Covered
bin DataOut[226]	2	1	Covered
bin DataOut[227]	1	1	Covered
bin DataOut[228]	0	1	ZERO
bin DataOut[229]	1	1	Covered

bin DataOut[230]	0	1	ZERO
bin DataOut[231]	1	1	Covered
bin DataOut[232]	1	1	Covered
bin DataOut[233]	0	1	ZERO
bin DataOut[234]	3	1	Covered
bin DataOut[235]	1	1	Covered
bin DataOut[236]	2	1	Covered
bin DataOut[237]	1	1	Covered
bin DataOut[238]	1	1	Covered
bin DataOut[239]	3	1	Covered
bin DataOut[240]	3	1	Covered
bin DataOut[241]	1	1	Covered
bin DataOut[242]	1	1	Covered
bin DataOut[243]	2	1	Covered
bin DataOut[244]	1	1	Covered
bin DataOut[245]	5	1	Covered
bin DataOut[246]	2	1	Covered
bin DataOut[247]	1	1	Covered
bin DataOut[248]	1	1	Covered
bin DataOut[249]	1	1	Covered
bin DataOut[250]	1	1	Covered
bin DataOut[251]	1	1	Covered
bin DataOut[252]	4	1	Covered
bin DataOut[253]	0	1	ZERO
bin DataOut[254]	4	1	Covered
bin DataOut[255]	1	1	Covered
Coverpoint RxData	98.0%	100	Uncovered
covered/total bins:	251	256	
missing/total bins:	5	256	
% Hit:	98.0%	100	
bin RxData[0]	14455	1	Covered
bin RxData[1]	48	1	Covered
bin RxData[2]	25	1	Covered
bin RxData[3]	49	1	Covered
bin RxData[4]	48	1	Covered
bin RxData[5]	24	1	Covered
bin RxData[6]	24	1	Covered
bin RxData[7]	9	1	Covered
bin RxData[8]	33	1	Covered
bin RxData[9]	32	1	Covered
bin RxData[10]	49	1	Covered
bin RxData[11]	56	1	Covered
bin RxData[12]	8	1	Covered
bin RxData[13]	171	1	Covered
bin RxData[14]	16	1	Covered
bin RxData[15]	89	1	Covered
bin RxData[16]	16	1	Covered
bin RxData[17]	32	1	Covered
bin RxData[18]	24	1	Covered
bin RxData[19]	32	1	Covered

bin RxData[20]	16	1	Covered
bin RxData[21]	48	1	Covered
bin RxData[22]	57	1	Covered
bin RxData[23]	139	1	Covered
bin RxData[24]	32	1	Covered
bin RxData[25]	41	1	Covered
bin RxData[26]	17	1	Covered
bin RxData[27]	41	1	Covered
bin RxData[28]	24	1	Covered
bin RxData[29]	48	1	Covered
bin RxData[30]	56	1	Covered
bin RxData[31]	324	1	Covered
bin RxData[32]	40	1	Covered
bin RxData[33]	24	1	Covered
bin RxData[34]	57	1	Covered
bin RxData[35]	24	1	Covered
bin RxData[36]	24	1	Covered
bin RxData[37]	40	1	Covered
bin RxData[38]	64	1	Covered
bin RxData[39]	48	1	Covered
bin RxData[40]	24	1	Covered
bin RxData[41]	8	1	Covered
bin RxData[42]	8	1	Covered
bin RxData[43]	8	1	Covered
bin RxData[44]	33	1	Covered
bin RxData[45]	33	1	Covered
bin RxData[46]	16	1	Covered
bin RxData[47]	8	1	Covered
bin RxData[48]	8	1	Covered
bin RxData[49]	16	1	Covered
bin RxData[50]	41	1	Covered
bin RxData[51]	40	1	Covered
bin RxData[52]	17	1	Covered
bin RxData[53]	24	1	Covered
bin RxData[54]	32	1	Covered
bin RxData[55]	40	1	Covered
bin RxData[56]	40	1	Covered
bin RxData[57]	16	1	Covered
bin RxData[58]	33	1	Covered
bin RxData[59]	41	1	Covered
bin RxData[60]	32	1	Covered
bin RxData[61]	65	1	Covered
bin RxData[62]	16	1	Covered
bin RxData[63]	32	1	Covered
bin RxData[64]	64	1	Covered
bin RxData[65]	8	1	Covered
bin RxData[66]	24	1	Covered
bin RxData[67]	0	1	ZERO
bin RxData[68]	56	1	Covered
bin RxData[69]	24	1	Covered

bin RxData[70]	40	1	Covered
bin RxData[71]	64	1	Covered
bin RxData[72]	32	1	Covered
bin RxData[73]	32	1	Covered
bin RxData[74]	0	1	ZERO
bin RxData[75]	33	1	Covered
bin RxData[76]	49	1	Covered
bin RxData[77]	25	1	Covered
bin RxData[78]	16	1	Covered
bin RxData[79]	24	1	Covered
bin RxData[80]	32	1	Covered
bin RxData[81]	160	1	Covered
bin RxData[82]	16	1	Covered
bin RxData[83]	40	1	Covered
bin RxData[84]	64	1	Covered
bin RxData[85]	17	1	Covered
bin RxData[86]	65	1	Covered
bin RxData[87]	8	1	Covered
bin RxData[88]	40	1	Covered
bin RxData[89]	24	1	Covered
bin RxData[90]	24	1	Covered
bin RxData[91]	40	1	Covered
bin RxData[92]	32	1	Covered
bin RxData[93]	16	1	Covered
bin RxData[94]	24	1	Covered
bin RxData[95]	16	1	Covered
bin RxData[96]	40	1	Covered
bin RxData[97]	16	1	Covered
bin RxData[98]	48	1	Covered
bin RxData[99]	56	1	Covered
bin RxData[100]	33	1	Covered
bin RxData[101]	40	1	Covered
bin RxData[102]	8	1	Covered
bin RxData[103]	32	1	Covered
bin RxData[104]	0	1	ZERO
bin RxData[105]	40	1	Covered
bin RxData[106]	48	1	Covered
bin RxData[107]	41	1	Covered
bin RxData[108]	41	1	Covered
bin RxData[109]	32	1	Covered
bin RxData[110]	24	1	Covered
bin RxData[111]	48	1	Covered
bin RxData[112]	16	1	Covered
bin RxData[113]	56	1	Covered
bin RxData[114]	24	1	Covered
bin RxData[115]	40	1	Covered
bin RxData[116]	24	1	Covered
bin RxData[117]	40	1	Covered
bin RxData[118]	57	1	Covered
bin RxData[119]	17	1	Covered

bin RxData[120]	32	1	Covered
bin RxData[121]	8	1	Covered
bin RxData[122]	24	1	Covered
bin RxData[123]	25	1	Covered
bin RxData[124]	24	1	Covered
bin RxData[125]	41	1	Covered
bin RxData[126]	40	1	Covered
bin RxData[127]	16	1	Covered
bin RxData[128]	25	1	Covered
bin RxData[129]	8	1	Covered
bin RxData[130]	24	1	Covered
bin RxData[131]	8	1	Covered
bin RxData[132]	16	1	Covered
bin RxData[133]	48	1	Covered
bin RxData[134]	48	1	Covered
bin RxData[135]	8	1	Covered
bin RxData[136]	16	1	Covered
bin RxData[137]	41	1	Covered
bin RxData[138]	32	1	Covered
bin RxData[139]	80	1	Covered
bin RxData[140]	50	1	Covered
bin RxData[141]	32	1	Covered
bin RxData[142]	24	1	Covered
bin RxData[143]	24	1	Covered
bin RxData[144]	48	1	Covered
bin RxData[145]	56	1	Covered
bin RxData[146]	40	1	Covered
bin RxData[147]	16	1	Covered
bin RxData[148]	0	1	ZERO
bin RxData[149]	16	1	Covered
bin RxData[150]	40	1	Covered
bin RxData[151]	48	1	Covered
bin RxData[152]	8	1	Covered
bin RxData[153]	81	1	Covered
bin RxData[154]	48	1	Covered
bin RxData[155]	49	1	Covered
bin RxData[156]	32	1	Covered
bin RxData[157]	24	1	Covered
bin RxData[158]	16	1	Covered
bin RxData[159]	32	1	Covered
bin RxData[160]	16	1	Covered
bin RxData[161]	56	1	Covered
bin RxData[162]	24	1	Covered
bin RxData[163]	16	1	Covered
bin RxData[164]	17	1	Covered
bin RxData[165]	48	1	Covered
bin RxData[166]	58	1	Covered
bin RxData[167]	24	1	Covered
bin RxData[168]	25	1	Covered
bin RxData[169]	50	1	Covered

bin RxData[170]	32	1	Covered
bin RxData[171]	25	1	Covered
bin RxData[172]	16	1	Covered
bin RxData[173]	32	1	Covered
bin RxData[174]	134	1	Covered
bin RxData[175]	41	1	Covered
bin RxData[176]	0	1	ZERO
bin RxData[177]	24	1	Covered
bin RxData[178]	56	1	Covered
bin RxData[179]	58	1	Covered
bin RxData[180]	50	1	Covered
bin RxData[181]	48	1	Covered
bin RxData[182]	32	1	Covered
bin RxData[183]	57	1	Covered
bin RxData[184]	49	1	Covered
bin RxData[185]	33	1	Covered
bin RxData[186]	17	1	Covered
bin RxData[187]	40	1	Covered
bin RxData[188]	40	1	Covered
bin RxData[189]	16	1	Covered
bin RxData[190]	32	1	Covered
bin RxData[191]	33	1	Covered
bin RxData[192]	25	1	Covered
bin RxData[193]	40	1	Covered
bin RxData[194]	25	1	Covered
bin RxData[195]	24	1	Covered
bin RxData[196]	32	1	Covered
bin RxData[197]	42	1	Covered
bin RxData[198]	40	1	Covered
bin RxData[199]	32	1	Covered
bin RxData[200]	184	1	Covered
bin RxData[201]	48	1	Covered
bin RxData[202]	25	1	Covered
bin RxData[203]	17	1	Covered
bin RxData[204]	646	1	Covered
bin RxData[205]	48	1	Covered
bin RxData[206]	40	1	Covered
bin RxData[207]	65	1	Covered
bin RxData[208]	24	1	Covered
bin RxData[209]	48	1	Covered
bin RxData[210]	33	1	Covered
bin RxData[211]	66	1	Covered
bin RxData[212]	366	1	Covered
bin RxData[213]	25	1	Covered
bin RxData[214]	16	1	Covered
bin RxData[215]	25	1	Covered
bin RxData[216]	16	1	Covered
bin RxData[217]	25	1	Covered
bin RxData[218]	24	1	Covered
bin RxData[219]	40	1	Covered

bin RxData[220]	26	1	Covered
bin RxData[221]	43	1	Covered
bin RxData[222]	32	1	Covered
bin RxData[223]	33	1	Covered
bin RxData[224]	60	1	Covered
bin RxData[225]	48	1	Covered
bin RxData[226]	33	1	Covered
bin RxData[227]	32	1	Covered
bin RxData[228]	9	1	Covered
bin RxData[229]	25	1	Covered
bin RxData[230]	42	1	Covered
bin RxData[231]	8	1	Covered
bin RxData[232]	33	1	Covered
bin RxData[233]	16	1	Covered
bin RxData[234]	53	1	Covered
bin RxData[235]	16	1	Covered
bin RxData[236]	33	1	Covered
bin RxData[237]	25	1	Covered
bin RxData[238]	17	1	Covered
bin RxData[239]	34	1	Covered
bin RxData[240]	52	1	Covered
bin RxData[241]	9	1	Covered
bin RxData[242]	35	1	Covered
bin RxData[243]	34	1	Covered
bin RxData[244]	17	1	Covered
bin RxData[245]	50	1	Covered
bin RxData[246]	18	1	Covered
bin RxData[247]	18	1	Covered
bin RxData[248]	18	1	Covered
bin RxData[249]	9	1	Covered
bin RxData[250]	27	1	Covered
bin RxData[251]	18	1	Covered
bin RxData[252]	40	1	Covered
bin RxData[253]	8	1	Covered
bin RxData[254]	56	1	Covered
bin RxData[255]	17	1	Covered
Coverpoint RxFrameSize	7.8%	100	Uncovered
covered/total bins:	10	127	
missing/total bins:	117	127	
% Hit:	7.8%	100	
ignore_bin Invalid	0		ZERO
bin RxFrameSize[0]	14516	1	Covered
bin RxFrameSize[1]	0	1	ZERO
bin RxFrameSize[2]	0	1	ZERO
bin RxFrameSize[3]	0	1	ZERO
bin RxFrameSize[4]	0	1	ZERO
bin RxFrameSize[5]	0	1	ZERO
bin RxFrameSize[6]	0	1	ZERO
bin RxFrameSize[7]	0	1	ZERO
bin RxFrameSize[8]	0	1	ZERO

bin RxFrameSize[9]	0	1	ZERO
bin RxFrameSize[10]	400	1	Covered
bin RxFrameSize[11]	0	1	ZERO
bin RxFrameSize[12]	0	1	ZERO
bin RxFrameSize[13]	0	1	ZERO
bin RxFrameSize[14]	0	1	ZERO
bin RxFrameSize[15]	0	1	ZERO
bin RxFrameSize[16]	0	1	ZERO
bin RxFrameSize[17]	0	1	ZERO
bin RxFrameSize[18]	0	1	ZERO
bin RxFrameSize[19]	0	1	ZERO
bin RxFrameSize[20]	0	1	ZERO
bin RxFrameSize[21]	0	1	ZERO
bin RxFrameSize[22]	0	1	ZERO
bin RxFrameSize[23]	0	1	ZERO
bin RxFrameSize[24]	0	1	ZERO
bin RxFrameSize[25]	488	1	Covered
bin RxFrameSize[26]	0	1	ZERO
bin RxFrameSize[27]	0	1	ZERO
bin RxFrameSize[28]	0	1	ZERO
bin RxFrameSize[29]	0	1	ZERO
bin RxFrameSize[30]	0	1	ZERO
bin RxFrameSize[31]	0	1	ZERO
bin RxFrameSize[32]	0	1	ZERO
bin RxFrameSize[33]	0	1	ZERO
bin RxFrameSize[34]	0	1	ZERO
bin RxFrameSize[35]	0	1	ZERO
bin RxFrameSize[36]	0	1	ZERO
bin RxFrameSize[37]	0	1	ZERO
bin RxFrameSize[38]	0	1	ZERO
bin RxFrameSize[39]	0	1	ZERO
bin RxFrameSize[40]	1110	1	Covered
bin RxFrameSize[41]	0	1	ZERO
bin RxFrameSize[42]	0	1	ZERO
bin RxFrameSize[43]	0	1	ZERO
bin RxFrameSize[44]	0	1	ZERO
bin RxFrameSize[45]	1167	1	Covered
bin RxFrameSize[46]	0	1	ZERO
bin RxFrameSize[47]	2068	1	Covered
bin RxFrameSize[48]	0	1	ZERO
bin RxFrameSize[49]	0	1	ZERO
bin RxFrameSize[50]	0	1	ZERO
bin RxFrameSize[51]	0	1	ZERO
bin RxFrameSize[52]	0	1	ZERO
bin RxFrameSize[53]	0	1	ZERO
bin RxFrameSize[54]	0	1	ZERO
bin RxFrameSize[55]	0	1	ZERO
bin RxFrameSize[56]	0	1	ZERO
bin RxFrameSize[57]	0	1	ZERO
bin RxFrameSize[58]	0	1	ZERO

bin RxFrameSize[59]	0	1	ZERO
bin RxFrameSize[60]	0	1	ZERO
bin RxFrameSize[61]	0	1	ZERO
bin RxFrameSize[62]	0	1	ZERO
bin RxFrameSize[63]	0	1	ZERO
bin RxFrameSize[64]	0	1	ZERO
bin RxFrameSize[65]	0	1	ZERO
bin RxFrameSize[66]	0	1	ZERO
bin RxFrameSize[67]	0	1	ZERO
bin RxFrameSize[68]	0	1	ZERO
bin RxFrameSize[69]	0	1	ZERO
bin RxFrameSize[70]	0	1	ZERO
bin RxFrameSize[71]	0	1	ZERO
bin RxFrameSize[72]	0	1	ZERO
bin RxFrameSize[73]	0	1	ZERO
bin RxFrameSize[74]	880	1	Covered
bin RxFrameSize[75]	0	1	ZERO
bin RxFrameSize[76]	0	1	ZERO
bin RxFrameSize[77]	0	1	ZERO
bin RxFrameSize[78]	0	1	ZERO
bin RxFrameSize[79]	0	1	ZERO
bin RxFrameSize[80]	0	1	ZERO
bin RxFrameSize[81]	0	1	ZERO
bin RxFrameSize[82]	0	1	ZERO
bin RxFrameSize[83]	0	1	ZERO
bin RxFrameSize[84]	0	1	ZERO
bin RxFrameSize[85]	0	1	ZERO
bin RxFrameSize[86]	0	1	ZERO
bin RxFrameSize[87]	0	1	ZERO
bin RxFrameSize[88]	0	1	ZERO
bin RxFrameSize[89]	0	1	ZERO
bin RxFrameSize[90]	0	1	ZERO
bin RxFrameSize[91]	0	1	ZERO
bin RxFrameSize[92]	0	1	ZERO
bin RxFrameSize[93]	0	1	ZERO
bin RxFrameSize[94]	0	1	ZERO
bin RxFrameSize[95]	0	1	ZERO
bin RxFrameSize[96]	0	1	ZERO
bin RxFrameSize[97]	0	1	ZERO
bin RxFrameSize[98]	0	1	ZERO
bin RxFrameSize[99]	0	1	ZERO
bin RxFrameSize[100]	0	1	ZERO
bin RxFrameSize[101]	263	1	Covered
bin RxFrameSize[102]	0	1	ZERO
bin RxFrameSize[103]	0	1	ZERO
bin RxFrameSize[104]	0	1	ZERO
bin RxFrameSize[105]	0	1	ZERO
bin RxFrameSize[106]	0	1	ZERO
bin RxFrameSize[107]	0	1	ZERO
bin RxFrameSize[108]	0	1	ZERO

bin RxFrameSize[109]	0	1	ZERO
bin RxFrameSize[110]	0	1	ZERO
bin RxFrameSize[111]	0	1	ZERO
bin RxFrameSize[112]	0	1	ZERO
bin RxFrameSize[113]	0	1	ZERO
bin RxFrameSize[114]	0	1	ZERO
bin RxFrameSize[115]	0	1	ZERO
bin RxFrameSize[116]	0	1	ZERO
bin RxFrameSize[117]	0	1	ZERO
bin RxFrameSize[118]	0	1	ZERO
bin RxFrameSize[119]	0	1	ZERO
bin RxFrameSize[120]	0	1	ZERO
bin RxFrameSize[121]	0	1	ZERO
bin RxFrameSize[122]	1100	1	Covered
bin RxFrameSize[123]	0	1	ZERO
bin RxFrameSize[124]	0	1	ZERO
bin RxFrameSize[125]	0	1	ZERO
bin RxFrameSize[126]	2505	1	Covered
Coverpoint RxValidFrame	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin InvalidFrame	16146	1	Covered
bin ValidFrame	8351	1	Covered
Coverpoint RxAbortSignal	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin Keep	24431	1	Covered
bin Abort	66	1	Covered
Coverpoint RxReady	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin NotReady	23175	1	Covered
bin Ready	1322	1	Covered
Coverpoint RxEoF	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin NotEoF	24484	1	Covered
bin EoF	13	1	Covered
Coverpoint RxOverflow	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin NoOverflow	23941	1	Covered
bin Overflow	556	1	Covered
Coverpoint RxFCSErr	100.0%	100	Covered
covered/total bins:	2	2	

missing/total bins:	0	2	
% Hit:	100.0%	100	
bin NoError	24496	1	Covered
bin Error	1	1	Covered
Coverpoint RxFrameError	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin NoFrameError	24433	1	Covered
bin FrameError	64	1	Covered
Coverpoint RxDrop	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin Keep	24496	1	Covered
bin Drop	1	1	Covered
Coverpoint TxValidFrame	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin InvalidFrame	18776	1	Covered
bin ValidFrame	5721	1	Covered
Coverpoint TxAbortedTrans	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin Kept	24447	1	Covered
bin Aborted	50	1	Covered
Coverpoint TxData	90.6%	100	Uncovered
covered/total bins:	232	256	
missing/total bins:	24	256	
% Hit:	90.6%	100	
bin TxData[0]	192	1	Covered
bin TxData[1]	16	1	Covered
bin TxData[2]	32	1	Covered
bin TxData[3]	27	1	Covered
bin TxData[4]	26	1	Covered
bin TxData[5]	0	1	ZERO
bin TxData[6]	0	1	ZERO
bin TxData[7]	8	1	Covered
bin TxData[8]	8	1	Covered
bin TxData[9]	25	1	Covered
bin TxData[10]	17	1	Covered
bin TxData[11]	16	1	Covered
bin TxData[12]	0	1	ZERO
bin TxData[13]	25	1	Covered
bin TxData[14]	55	1	Covered
bin TxData[15]	16	1	Covered
bin TxData[16]	8	1	Covered
bin TxData[17]	25	1	Covered

bin TxData[18]	16	1	Covered
bin TxData[19]	56	1	Covered
bin TxData[20]	0	1	ZERO
bin TxData[21]	32	1	Covered
bin TxData[22]	0	1	ZERO
bin TxData[23]	43	1	Covered
bin TxData[24]	0	1	ZERO
bin TxData[25]	0	1	ZERO
bin TxData[26]	8	1	Covered
bin TxData[27]	114	1	Covered
bin TxData[28]	33	1	Covered
bin TxData[29]	1042	1	Covered
bin TxData[30]	16	1	Covered
bin TxData[31]	0	1	ZERO
bin TxData[32]	16	1	Covered
bin TxData[33]	41	1	Covered
bin TxData[34]	32	1	Covered
bin TxData[35]	16	1	Covered
bin TxData[36]	23	1	Covered
bin TxData[37]	24	1	Covered
bin TxData[38]	27	1	Covered
bin TxData[39]	26	1	Covered
bin TxData[40]	9	1	Covered
bin TxData[41]	0	1	ZERO
bin TxData[42]	0	1	ZERO
bin TxData[43]	32	1	Covered
bin TxData[44]	33	1	Covered
bin TxData[45]	8	1	Covered
bin TxData[46]	32	1	Covered
bin TxData[47]	16	1	Covered
bin TxData[48]	32	1	Covered
bin TxData[49]	16	1	Covered
bin TxData[50]	8	1	Covered
bin TxData[51]	32	1	Covered
bin TxData[52]	0	1	ZERO
bin TxData[53]	24	1	Covered
bin TxData[54]	8	1	Covered
bin TxData[55]	17	1	Covered
bin TxData[56]	24	1	Covered
bin TxData[57]	8	1	Covered
bin TxData[58]	9	1	Covered
bin TxData[59]	25	1	Covered
bin TxData[60]	34	1	Covered
bin TxData[61]	16	1	Covered
bin TxData[62]	41	1	Covered
bin TxData[63]	24	1	Covered
bin TxData[64]	8	1	Covered
bin TxData[65]	33	1	Covered
bin TxData[66]	40	1	Covered
bin TxData[67]	16	1	Covered

bin TxData[68]	34	1	Covered
bin TxData[69]	8	1	Covered
bin TxData[70]	50	1	Covered
bin TxData[71]	8	1	Covered
bin TxData[72]	8	1	Covered
bin TxData[73]	16	1	Covered
bin TxData[74]	16	1	Covered
bin TxData[75]	41	1	Covered
bin TxData[76]	16	1	Covered
bin TxData[77]	36	1	Covered
bin TxData[78]	32	1	Covered
bin TxData[79]	8	1	Covered
bin TxData[80]	40	1	Covered
bin TxData[81]	40	1	Covered
bin TxData[82]	33	1	Covered
bin TxData[83]	41	1	Covered
bin TxData[84]	32	1	Covered
bin TxData[85]	0	1	ZERO
bin TxData[86]	9	1	Covered
bin TxData[87]	8	1	Covered
bin TxData[88]	32	1	Covered
bin TxData[89]	24	1	Covered
bin TxData[90]	32	1	Covered
bin TxData[91]	33	1	Covered
bin TxData[92]	8	1	Covered
bin TxData[93]	9	1	Covered
bin TxData[94]	16	1	Covered
bin TxData[95]	32	1	Covered
bin TxData[96]	26	1	Covered
bin TxData[97]	24	1	Covered
bin TxData[98]	8	1	Covered
bin TxData[99]	26	1	Covered
bin TxData[100]	24	1	Covered
bin TxData[101]	8	1	Covered
bin TxData[102]	16	1	Covered
bin TxData[103]	0	1	ZERO
bin TxData[104]	17	1	Covered
bin TxData[105]	16	1	Covered
bin TxData[106]	0	1	ZERO
bin TxData[107]	24	1	Covered
bin TxData[108]	8	1	Covered
bin TxData[109]	27	1	Covered
bin TxData[110]	16	1	Covered
bin TxData[111]	16	1	Covered
bin TxData[112]	8	1	Covered
bin TxData[113]	0	1	ZERO
bin TxData[114]	25	1	Covered
bin TxData[115]	40	1	Covered
bin TxData[116]	8	1	Covered
bin TxData[117]	25	1	Covered

bin TxData[118]	47	1	Covered
bin TxData[119]	40	1	Covered
bin TxData[120]	1043	1	Covered
bin TxData[121]	1042	1	Covered
bin TxData[122]	8	1	Covered
bin TxData[123]	16	1	Covered
bin TxData[124]	8	1	Covered
bin TxData[125]	28	1	Covered
bin TxData[126]	24	1	Covered
bin TxData[127]	8	1	Covered
bin TxData[128]	24	1	Covered
bin TxData[129]	8	1	Covered
bin TxData[130]	16	1	Covered
bin TxData[131]	41	1	Covered
bin TxData[132]	24	1	Covered
bin TxData[133]	16	1	Covered
bin TxData[134]	18	1	Covered
bin TxData[135]	25	1	Covered
bin TxData[136]	16	1	Covered
bin TxData[137]	1043	1	Covered
bin TxData[138]	26	1	Covered
bin TxData[139]	16	1	Covered
bin TxData[140]	33	1	Covered
bin TxData[141]	15	1	Covered
bin TxData[142]	16	1	Covered
bin TxData[143]	8	1	Covered
bin TxData[144]	63	1	Covered
bin TxData[145]	17	1	Covered
bin TxData[146]	24	1	Covered
bin TxData[147]	47	1	Covered
bin TxData[148]	16	1	Covered
bin TxData[149]	48	1	Covered
bin TxData[150]	65	1	Covered
bin TxData[151]	24	1	Covered
bin TxData[152]	32	1	Covered
bin TxData[153]	8	1	Covered
bin TxData[154]	16	1	Covered
bin TxData[155]	8	1	Covered
bin TxData[156]	56	1	Covered
bin TxData[157]	41	1	Covered
bin TxData[158]	25	1	Covered
bin TxData[159]	26	1	Covered
bin TxData[160]	48	1	Covered
bin TxData[161]	32	1	Covered
bin TxData[162]	16	1	Covered
bin TxData[163]	8	1	Covered
bin TxData[164]	33	1	Covered
bin TxData[165]	0	1	ZERO
bin TxData[166]	16	1	Covered
bin TxData[167]	25	1	Covered

bin TxData[168]	16	1	Covered
bin TxData[169]	17	1	Covered
bin TxData[170]	8	1	Covered
bin TxData[171]	25	1	Covered
bin TxData[172]	49	1	Covered
bin TxData[173]	57	1	Covered
bin TxData[174]	33	1	Covered
bin TxData[175]	9	1	Covered
bin TxData[176]	24	1	Covered
bin TxData[177]	23	1	Covered
bin TxData[178]	24	1	Covered
bin TxData[179]	27	1	Covered
bin TxData[180]	41	1	Covered
bin TxData[181]	47	1	Covered
bin TxData[182]	8	1	Covered
bin TxData[183]	24	1	Covered
bin TxData[184]	33	1	Covered
bin TxData[185]	25	1	Covered
bin TxData[186]	369	1	Covered
bin TxData[187]	25	1	Covered
bin TxData[188]	41	1	Covered
bin TxData[189]	25	1	Covered
bin TxData[190]	32	1	Covered
bin TxData[191]	27	1	Covered
bin TxData[192]	28	1	Covered
bin TxData[193]	8	1	Covered
bin TxData[194]	0	1	ZERO
bin TxData[195]	16	1	Covered
bin TxData[196]	34	1	Covered
bin TxData[197]	26	1	Covered
bin TxData[198]	32	1	Covered
bin TxData[199]	32	1	Covered
bin TxData[200]	24	1	Covered
bin TxData[201]	8	1	Covered
bin TxData[202]	16	1	Covered
bin TxData[203]	32	1	Covered
bin TxData[204]	0	1	ZERO
bin TxData[205]	26	1	Covered
bin TxData[206]	41	1	Covered
bin TxData[207]	16	1	Covered
bin TxData[208]	24	1	Covered
bin TxData[209]	24	1	Covered
bin TxData[210]	33	1	Covered
bin TxData[211]	41	1	Covered
bin TxData[212]	49	1	Covered
bin TxData[213]	0	1	ZERO
bin TxData[214]	0	1	ZERO
bin TxData[215]	32	1	Covered
bin TxData[216]	34	1	Covered
bin TxData[217]	43	1	Covered

bin TxData[218]	8	1	Covered
bin TxData[219]	7	1	Covered
bin TxData[220]	402	1	Covered
bin TxData[221]	16	1	Covered
bin TxData[222]	346	1	Covered
bin TxData[223]	33	1	Covered
bin TxData[224]	18	1	Covered
bin TxData[225]	32	1	Covered
bin TxData[226]	8	1	Covered
bin TxData[227]	32	1	Covered
bin TxData[228]	16	1	Covered
bin TxData[229]	0	1	ZERO
bin TxData[230]	41	1	Covered
bin TxData[231]	16	1	Covered
bin TxData[232]	16	1	Covered
bin TxData[233]	11	1	Covered
bin TxData[234]	8	1	Covered
bin TxData[235]	16	1	Covered
bin TxData[236]	1035	1	Covered
bin TxData[237]	8	1	Covered
bin TxData[238]	16	1	Covered
bin TxData[239]	0	1	ZERO
bin TxData[240]	0	1	ZERO
bin TxData[241]	16	1	Covered
bin TxData[242]	0	1	ZERO
bin TxData[243]	34	1	Covered
bin TxData[244]	24	1	Covered
bin TxData[245]	24	1	Covered
bin TxData[246]	25	1	Covered
bin TxData[247]	9	1	Covered
bin TxData[248]	32	1	Covered
bin TxData[249]	35	1	Covered
bin TxData[250]	32	1	Covered
bin TxData[251]	32	1	Covered
bin TxData[252]	24	1	Covered
bin TxData[253]	8	1	Covered
bin TxData[254]	17	1	Covered
bin TxData[255]	12539	1	Covered
Coverpoint TxFull	100.0%	100	Covered
covered/total bins:	2	2	
missing/total bins:	0	2	
% Hit:	100.0%	100	
bin NotFull	24475	1	Covered
bin Full	22	1	Covered
Coverpoint TxFrameSize	4.7%	100	Uncovered
covered/total bins:	6	127	
missing/total bins:	121	127	
% Hit:	4.7%	100	
ignore_bin Invalid	0		ZERO
bin TxFrameSize[0]	12323	1	Covered

bin TxFrameSize[1]	0	1	ZERO
bin TxFrameSize[2]	0	1	ZERO
bin TxFrameSize[3]	0	1	ZERO
bin TxFrameSize[4]	0	1	ZERO
bin TxFrameSize[5]	0	1	ZERO
bin TxFrameSize[6]	0	1	ZERO
bin TxFrameSize[7]	0	1	ZERO
bin TxFrameSize[8]	0	1	ZERO
bin TxFrameSize[9]	0	1	ZERO
bin TxFrameSize[10]	229	1	Covered
bin TxFrameSize[11]	0	1	ZERO
bin TxFrameSize[12]	0	1	ZERO
bin TxFrameSize[13]	0	1	ZERO
bin TxFrameSize[14]	0	1	ZERO
bin TxFrameSize[15]	0	1	ZERO
bin TxFrameSize[16]	0	1	ZERO
bin TxFrameSize[17]	0	1	ZERO
bin TxFrameSize[18]	0	1	ZERO
bin TxFrameSize[19]	0	1	ZERO
bin TxFrameSize[20]	0	1	ZERO
bin TxFrameSize[21]	0	1	ZERO
bin TxFrameSize[22]	0	1	ZERO
bin TxFrameSize[23]	0	1	ZERO
bin TxFrameSize[24]	0	1	ZERO
bin TxFrameSize[25]	0	1	ZERO
bin TxFrameSize[26]	0	1	ZERO
bin TxFrameSize[27]	0	1	ZERO
bin TxFrameSize[28]	0	1	ZERO
bin TxFrameSize[29]	0	1	ZERO
bin TxFrameSize[30]	0	1	ZERO
bin TxFrameSize[31]	0	1	ZERO
bin TxFrameSize[32]	0	1	ZERO
bin TxFrameSize[33]	0	1	ZERO
bin TxFrameSize[34]	0	1	ZERO
bin TxFrameSize[35]	0	1	ZERO
bin TxFrameSize[36]	0	1	ZERO
bin TxFrameSize[37]	0	1	ZERO
bin TxFrameSize[38]	0	1	ZERO
bin TxFrameSize[39]	0	1	ZERO
bin TxFrameSize[40]	1254	1	Covered
bin TxFrameSize[41]	0	1	ZERO
bin TxFrameSize[42]	0	1	ZERO
bin TxFrameSize[43]	0	1	ZERO
bin TxFrameSize[44]	0	1	ZERO
bin TxFrameSize[45]	0	1	ZERO
bin TxFrameSize[46]	0	1	ZERO
bin TxFrameSize[47]	823	1	Covered
bin TxFrameSize[48]	0	1	ZERO
bin TxFrameSize[49]	0	1	ZERO
bin TxFrameSize[50]	0	1	ZERO

bin TxFrameSize[51]	0	1	ZERO
bin TxFrameSize[52]	0	1	ZERO
bin TxFrameSize[53]	0	1	ZERO
bin TxFrameSize[54]	0	1	ZERO
bin TxFrameSize[55]	0	1	ZERO
bin TxFrameSize[56]	0	1	ZERO
bin TxFrameSize[57]	0	1	ZERO
bin TxFrameSize[58]	0	1	ZERO
bin TxFrameSize[59]	0	1	ZERO
bin TxFrameSize[60]	0	1	ZERO
bin TxFrameSize[61]	0	1	ZERO
bin TxFrameSize[62]	0	1	ZERO
bin TxFrameSize[63]	0	1	ZERO
bin TxFrameSize[64]	0	1	ZERO
bin TxFrameSize[65]	0	1	ZERO
bin TxFrameSize[66]	0	1	ZERO
bin TxFrameSize[67]	0	1	ZERO
bin TxFrameSize[68]	0	1	ZERO
bin TxFrameSize[69]	0	1	ZERO
bin TxFrameSize[70]	0	1	ZERO
bin TxFrameSize[71]	0	1	ZERO
bin TxFrameSize[72]	0	1	ZERO
bin TxFrameSize[73]	0	1	ZERO
bin TxFrameSize[74]	0	1	ZERO
bin TxFrameSize[75]	0	1	ZERO
bin TxFrameSize[76]	0	1	ZERO
bin TxFrameSize[77]	0	1	ZERO
bin TxFrameSize[78]	0	1	ZERO
bin TxFrameSize[79]	0	1	ZERO
bin TxFrameSize[80]	0	1	ZERO
bin TxFrameSize[81]	0	1	ZERO
bin TxFrameSize[82]	0	1	ZERO
bin TxFrameSize[83]	0	1	ZERO
bin TxFrameSize[84]	0	1	ZERO
bin TxFrameSize[85]	0	1	ZERO
bin TxFrameSize[86]	0	1	ZERO
bin TxFrameSize[87]	0	1	ZERO
bin TxFrameSize[88]	0	1	ZERO
bin TxFrameSize[89]	0	1	ZERO
bin TxFrameSize[90]	0	1	ZERO
bin TxFrameSize[91]	0	1	ZERO
bin TxFrameSize[92]	0	1	ZERO
bin TxFrameSize[93]	0	1	ZERO
bin TxFrameSize[94]	0	1	ZERO
bin TxFrameSize[95]	0	1	ZERO
bin TxFrameSize[96]	0	1	ZERO
bin TxFrameSize[97]	0	1	ZERO
bin TxFrameSize[98]	0	1	ZERO
bin TxFrameSize[99]	0	1	ZERO
bin TxFrameSize[100]	0	1	ZERO

```
        bin TxFrameSize[101]                                0          1    ZERO
        bin TxFrameSize[102]                                0          1    ZERO
        bin TxFrameSize[103]                                0          1    ZERO
        bin TxFrameSize[104]                                0          1    ZERO
        bin TxFrameSize[105]                                0          1    ZERO
        bin TxFrameSize[106]                                0          1    ZERO
        bin TxFrameSize[107]                                0          1    ZERO
        bin TxFrameSize[108]                                0          1    ZERO
        bin TxFrameSize[109]                                0          1    ZERO
        bin TxFrameSize[110]                                0          1    ZERO
        bin TxFrameSize[111]                                0          1    ZERO
        bin TxFrameSize[112]                                0          1    ZERO
        bin TxFrameSize[113]                                0          1    ZERO
        bin TxFrameSize[114]                                0          1    ZERO
        bin TxFrameSize[115]                                0          1    ZERO
        bin TxFrameSize[116]                                0          1    ZERO
        bin TxFrameSize[117]                                0          1    ZERO
        bin TxFrameSize[118]                                0          1    ZERO
        bin TxFrameSize[119]                                0          1    ZERO
        bin TxFrameSize[120]                                0          1    ZERO
        bin TxFrameSize[121]                                0          1    ZERO
        bin TxFrameSize[122]                             3565          1    Covered
        bin TxFrameSize[123]                                0          1    ZERO
        bin TxFrameSize[124]                                0          1    ZERO
        bin TxFrameSize[125]                                0          1    ZERO
        bin TxFrameSize[126]                             6303          1    Covered
    Coverpoint TxEnable                                100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin Disabled                                    24488          1    Covered
        bin Enabled                                         9          1    Covered
    Coverpoint TxAbortFrame                            100.0%        100    Covered
        covered/total bins:                                 2          2
        missing/total bins:                                 0          2
        % Hit:                                         100.0%        100
        bin Keep                                        24495          1    Covered
        bin Abort                                           2          1    Covered

TOTAL COVERGROUP COVERAGE: 89.0%  COVERGROUP TYPES: 1
```