TFE4171 - Term Project Part A

Bjørg Solem
Emanuela Tran
Stine Olsen

Spring 2022

1 Introduction

Part A of the term project is to get to know HDLC design and add assertions in the modules in order to ensure that the design meets given specifications. During the project there is written immediate and concurrent assertions.

2 Immediate assertions

For the immediate assertions we had to write assertions to verify correct values in Rx Status/Control register (RX_SC) and the Rx data buffer (RX_Buff). In order to read the status/control register and the Rx data buffer we had to read the address using the ReadAddress-function and the address given HDLC Module Design Description.

2.1 VerifyAbortReceive

VerifyAbortReceive verifies correct value in the Rx status/control register, and checks that the Rx data buffer is zero after abort. We check that the rx buffer has no data, that we have no frame error, abort signal is asserted and that we have no overflow signal. Additionally we asserts that rx_buff is empty ('b0000 0000). If one assertion fails we will be given a FAIL-message and increase the error count with one.

```
task VerifyAbortReceive(logic [127:0][7:0] data, int Size);
logic [7:0] ReadData;

ReadAddress('h02,ReadData); //Rx_SC address is 0x2

//assert that rx_sc is correct value for abort, all RO bits except abort should
↪ be 0. We dont know what value the WO bits will have

a_Correct_Val_0: assert (ReadData[0]==0)
    $display("PASS: VerifyAbortReceive PASSED, Rx_Buff has no data ");
else begin
    $display("FAIL: VerifyAbortReceive FAILED,  wrong value in ReadData:
    ↪ %h",ReadData);
    TbErrorCnt++;
```

```
end

a_Correct_Val_2: assert ( ReadData[2]==0)
    $display("PASS: VerifyAbortReceive PASSED, No frame error ");
else begin
    $display("FAIL: VerifyAbortReceive FAILED,  wrong value in ReadData:
    ↪  %h",ReadData);
    TbErrorCnt++;
end

a_Correct_Val_3: assert (ReadData[3]==1)
    $display("PASS: VerifyAbortReceive PASSED, Abort signal asserted ");
else begin
    $display("FAIL: VerifyAbortReceive FAILED,  wrong value in ReadData:
    ↪  %h",ReadData);
    TbErrorCnt++;
end

a_Correct_Val_4: assert (ReadData[4]==0)
    $display("PASS: VerifyAbortReceive PASSED, No overflow signal ");
else begin
    $display("FAIL: VerifyAbortReceive FAILED,  wrong value in ReadData:
    ↪  %h",ReadData);
    TbErrorCnt++;
end


//assert that rx_buff is empty
ReadAddress('h03,ReadData); //Rx_Buff address is 0x3

a_DataBuf_zero: assert (ReadData=='b00000000)
    $display("PASS: VerifyAbortReceive PASSED, Rx_Buff is empty");
else begin
    $display("FAIL: VerifyAbortReceive FAILED, Rx databuf is not zero:
    ↪  %h",ReadData);
    TbErrorCnt++;
end

endtask
```

2.2 VerifyNormalReceive

VerifyNormalReceive verifies the correct value in Rx status/control. Here it should verify that Rx_ready is true, and that all other RO (read only) is false. VerifyNormalReceive also asserts that Rx_buff has the correct data. It's correct if the Rx data buffer is not full. This is solved using a for-loop, where it checks every bit in the Rx data buffer.

```
task VerifyNormalReceive(logic [127:0][7:0] data, int Size);
logic [7:0] ReadData;
wait(uin_hdlc.Rx_Ready);

ReadAddress('h02, ReadData);

a_VerifyNormalReceive_0: assert(ReadData[0] == 1 )
```

```
    $display("PASS: VerifyNormalReceive PASSED. Rx_Buff has data to read");
else begin
    $display("FAILED: VerifyNormalReceive FAILED. Value in Rx_SC is %h",
     ↪  ReadData);
    TbErrorCnt++;
end

a_VerifyNormalReceive_2: assert(ReadData[2] == 0 )
    $display("PASS: VerifyNormalReceive PASSED. No Frame error");
else begin
    $display("FAILED: VerifyNormalReceive FAILED. Value in Rx_SC is
     ↪  %h",ReadData);
    TbErrorCnt++;
 end

a_VerifyNormalReceive_3: assert(ReadData[3] == 0 )
    $display("PASS: VerifyNormalReceive PASSED. No Abort signal");
else begin
    $display("FAILED: VerifyNormalReceive FAILED. Value in Rx_SC is %h",
     ↪  ReadData);
    TbErrorCnt++;
 end

a_VerifyNormalReceive_4: assert(ReadData[4] == 0 )
    $display("PASS: VerifyNormalReceive PASSED. No overflow signal");
else begin
    $display("FAILED: VerifyNormalReceive FAILED. Value in Rx_SC is %h",
     ↪  ReadData);
    TbErrorCnt++;
 end

//        ReadAddress('h03,ReadData);

for (int i =0; i< Size; i++) begin
    ReadAddress('h03,ReadData);
    a_Rx_Buff_not_full: assert (ReadData == data[i])
            $display("PASS: VerifyNormalReceive PASSED. Rx_Buff has correct
             ↪  data");
    else begin
        $display("FAIL: VerifyNormalReceive FAILED. Value in Rx_Buff is %h",
         ↪  ReadData);
        TbErrorCnt++;
        end
end
endtask
```

2.3 VerifyOverflowReceive

VerifyOverflowReceive verifies that rx_buff has data, no frame error, no abort signals and overflow signal is asserted. In this task there is only focus on the Rx status/control, not the Rx data buffer. If one of the assertion fails we are displaying a fail-message and increase the error count.

```
task VerifyOverflowReceive(logic [127:0][7:0] data, int Size);
logic [7:0] ReadData;
```

```
wait(uin_hdlc.Rx_Ready);

ReadAddress('h02, ReadData);

a_Verify_Overflow_0: assert ( ReadData[0]==1)
    $display("PASS: VerifyOverflowRecive PASSED. Rx_Buff has data to read");
else begin
    $display("FAIL: VerifyOverflowReceived FAILED.");
    TbErrorCnt++;
end

a_Verify_Overflow_2: assert (ReadData[2]==0)
    $display("PASS: VerifyOverflowReceive PASSED. No frame error.");
else begin
    $display("FAIL: VertifyOverflowReceived FAILED.");
    TbErrorCnt++;
end

 a_Verify_Overflow_3: assert (ReadData[3]==0)
    $display("PASS: VerifyOverflowReceive PASSED. No abort signal.");
else begin
    $display("FAIL: VertifyOverflowReceived FAILED.");
    TbErrorCnt++;
end

 a_Verify_Overflow_4: assert (ReadData[4]==1)
    $display("PASS: VerifyOverflowReceive PASSED. Overflow signal asserted.");
else begin
    $display("FAIL: VertifyOverflowReceived FAILED.");
    TbErrorCnt++;
end

endtask
```

3 Concurrent assertions

3.1 Rx_FlagDetect

Adds the sequence Rx_flag we are verifying, given in the HDLC Module Design Description (should be 0111 1110).

```
sequence Rx_flag; //Rx_flag should become 01111110
    !Rx ##1  Rx [*6] ##1 !Rx;
endsequence

// Check if flag sequence is detected
property RX_FlagDetect;
    @(posedge Clk) Rx_flag |-> ##2 Rx_FlagDetect;
endproperty

RX_FlagDetect_Assert : assert property (RX_FlagDetect) begin
    $display("PASS: Flag detect");
end else begin
    $error("Flag sequence did not generate FlagDetect");
    ErrCntAssertions++;
end
```

3.2 Rx_AbortSignal

Verifies correct Rx_AbortSignal behavior. If abort is detected during valid frame, then abort signal should go high. If Rx_AbortSignal is asserted then displays a pass message, if not gives an error message and increase error count.

```
property RX_AbortSignal;
    @(posedge Clk) (Rx_ValidFrame && Rx_AbortDetect) |=> Rx_AbortSignal;
endproperty

RX_AbortSignal_Assert : assert property (RX_AbortSignal) begin
    $display("PASS: Abort signal");
end else begin
    $error("AbortSignal did not go high after AbortDetect during validframe");
    ErrCntAssertions++;
end
```

4 RXSC

When running the simulation for the first time we get an error with RXSC in *TestPr_hdlc.sv*. In order to fix this error we changed RXSC to 'h02, which is the actual address for RXSC.

```
//Enable FCS
if(!Overflow && !NonByteAligned)
    WriteAddress('h02, 8'h20);
else
    WriteAddress('h02, 8'h00); //changed RXSC to its actual address
```

When the RXSC error is fixed we get the proper results, and have completed part A. There are codes that could be improved, but for not it is sufficient.

5 Discussion

When comparing our simulation results to the provided simulation log, we ended up with the same amount of assertion errors, and the simulation messages were the same, except for the differences in our own display-messages. This shows that the written assertions work the way they were supposed to.