Verification of an untested High-level Data Link Control module

Øystein Smith and Eivind Gamst

May 3, 2016

Abstract

In this report an untested HDLC module will be verified with the usage of SystemVerilog Assertions and constrained random stimuli.

The language for verifying the HDLC module was chosen as system verilog (SV). System verilog assertions (SVA) was used to verify the output. The input is generated with constrained randomisation.

The assertions were written from the specification in (1), although it was lacking in parts describing the protocol of hdlc were the hdlc standart was used to specify correct behaviour (2). But these two specifiactions were not completely covering, so when an assertion failed where these specifications were not absolute it was assumed that the authors guesswork was at fault and the hdlc module was functioning properly, and such the assertions were rewritten as to pass. This is a large error source, wich can be improved by a better specification, and/or more experienced authors.

As the work on implementing the testbench and subsequent revisions of the assertions was not completed, no conclusions as to the correct behaviour of the hdlc module can be made. The implementation of the testbench, data generation, and validation by assertions seems to be on the right track but not completed.

Preface

This report describes the work done in the term project in the subject TFE4171 Design of Digital Systems 2 spring of 2016. This project was cancelled one week before the deadline, and this report is heavily influenced by this. This report is written after the cancellation of the project to document the work done on the project, because the authors felt a lot of work has been put into the project and did not want it to go to waste. This report is to be regarded as a first draft, and not a complete work.

When reading the methodology chapter section 2 it is recommended to keep the source code appended side by side to quicly reference it.

Contents

1 Introduction

An High-Level Data Link Control (2), HDLC, needs to be verified. It's implemented in VHDL (1) and has to be bounded to SystemVerilog assertions and a constrained random testbench, to assertion that the design meets the specification before it is integrated with other modules.

1.1 HDLC module by J. Khatib

The hdlc module to be verified is written by J. Khatib. This module consists of a rx part and a tx part, as well as a control module. For more information see the documentation of the module (1).

1.2 Source code

The source code for this project can be found in the attached zip file. Or on github (3).

2 Methodology

When reading this section it is recommended to view the source code at the same time.

The language for verifying the HDLC module was chosen as system verilog (SV). System verilog assertions (SVA) was written in a separate file and included in the testbench. The only task the testbench has, is to generate input to the HDLC module. The input is generated with constrained randomisation.

2.1 Assertions

To verify correct behaviour of the hdlc module the output and some internal signals needs to be verified. This is done with SVA, see appendix 'hdlc_props.v'. These assertions check whether the signals has the correct sequence in the correct sequence. These are covered more in depth in section 4.

2.2 Testbench

The testbench consists of two files; 'hdlc_packet.sv' 'and hdlc_tb.sv'. The first contains a class used to generate the packets sent into the rx line of the dut, and the second is the top level of the test bench.

In the class a struct (line 1) was made to hold the required data fields. Certain input fields are predefined while others are randomly generated, these fields are defined in the struct. A class (line 10) is used to create functions to generate the random data, and to assure correct data is arriving at the correct time. The class also contains a crc generating function (line 80) from (4). The integration of this function was not completed, and thus it does not function as intended.

In the test bench top level; before beginning to generate random data, a initialisation sequence has to be input (line 34-38). This consists of waiting a bit, releasing *reset* and then enabling tx and rx. To generate the random data, an array is initialized, and for each entry the constructor for the packet class is run, then the randomize function is run to randomize everything exept the *data* field. To randomize the data field the 'randomize_foreach' (custom) function is called. Then the getbits function unpacks the data and inputs one bit at a time into rx of the dut, a small delay is inserted between each packet (line 39-47).

Due to change in the project priority, certain features where not implemented. Future necessary improvements includes proper coverage reporting through adding coverpoints and/or coverage groups that can assure that all types of cases have been covered and that corner cases can be discovered. Also missing from the testbench is data generation and input for the transmitting part of the hdlc module. This would be acheived in much the same way, but obviously a completely different data structure.

2.3 Wrapper

To bind the VHDL design to systemverilog a wrapper (5) in verilog was used ('tb_wrapper.sv'). The wrapper contains the ports of the VHDL module and the Verilog module, and then the wrapper is instantiated in systemverilog (by being included in the test bench top level), were it is connected with systemverilog assertions and testbench.

3 Results

Random data generation was successful, except the crc function not being functional (not finished integration). And it seems to be a powerful and flexible tool. All the finished assertions behaved as expected, and no errors were discovered.

4 Discussion

4.1 Assertions

The assertions were written from the specification in (1), although it was lacking in parts describing the protocol of hdlc were the hdlc standart was used to specify correct behaviour (2). But these two specifiactions were not completely covering, so when an assertion failed where these specifications were not absolute it was assumed that the authors guesswork was at fault and the hdlc module was functioning properly, and such the assertions were rewritten as to pass. This is a large error source, wich can be improved by a better specification, and/or more experienced authors.

4.2 Data generation

The random data generation seems to be correct, but only covers half of the functionality of the module. This is due to the work being cancelled before the other half was implemented.

5 Conclusion

As the work on implementing the testbench and subsequent revisions of the assertions was not completed, no conclusions as to the correct behaviour of the hdlc module can be made. The implementation of the testbench, data generation, and validation by assertions seems to be on the right track but not completed.

References

[1] J. Khatib, "HDLC controller core," tech. rep., Apr. 2001.

[2] "Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures," standard, International Organization for Standardization, Geneva, CH, Dec. 2007.

[3] E. Gamst and Øystein Smith, "Tfe4171 semester project." `https://github.com/cable89/tfe4171`, 2016.

[4] S. Gandhi, "System-verilog-packet-library." `https://github.com/sach/System-Verilog-Packet-Library`, 2011.

[5] A. Crone, *Binding SystemVerilog to VHDL Components Using Questa*. Mentor Graphics Corporation, 2005.