Prototyping System for USB3.0 Link Layer Using Synthesizable Assertions and Partial Reconfiguration

Harutyun, Krrikyan
Synopsys Armenia CJSC
Yerevan, Armenia
e-mail: harutk@synopsys.com

Taron, Hovhannisyan
Synopsys Armenia CJSC
Yerevan, Armenia
e-mail: taronhov@synopsys.com

Sergey, Manukyan
Synopsys Armenia CJSC
Yerevan, Armenia
e-mail: msergey@synopsys.com

ABSTRACT
There are several researches, which identify the following two problems as main bottlenecks of post-silicon validation: bug detection/localization and coverage calculation. Although FPGA prototyping is considered as pre-silicon verification the coverage calculation and bug localization are as much challenging as for post-silicon phase. This article presents a novel approach to solve the above mentioned issues by embedding synthesizable assertions into prototype. The experiment has been performed on USB3.0 link layer prototype. During the experiment all injected errors have been successfully detected and functional coverage has been calculated. The main drawback of the proposed approach is unacceptable resource utilization which, was solved by adding reconfigurable regions into FPGA. Synthesis results are presented for group of assertions intended to test entry/exit functionality of low power mode which show that the FPGA resource overhead is less than 2% of USB3.0 link prototype. This approach gives us ability to implement USB3.0 design and all assertions for single test in one FPGA with acceptable timing results and minor area overhead.

Keywords
Field Programmable Gate Array (FPGA), Partial Reconfiguration (PR), Universal Serial Bus (USB), System Verilog Assertion (SVA)

1. INTRODUCTION
[1] presents the overview of post-silicon validation problem and how it differs from traditional pre-silicon verification and manufacturing testing. The following challenges of post-silicon validation introduced in research exist in FPGA prototyping: bug detection/localization and coverage calculation.

Although there are multiple tools and methods for bug detection provided by USB IF, there is a lack of bug localization tools. Insufficient controllability and observability due to limited access to internal signals during prototyping is a bottleneck of fast bug detection and fixing.

Several methods exist for measuring pre-silicon coverage including code coverage, Finite State Machine (FSM) coverage, assertion coverage and mutation coverage (e.g., checking if the injected bug is caught during verification). Although FPGA prototyping is considered as pre-silicon verification, the coverage calculation is as much challenging as for post-silicon. Most of researches try to find an acceptable solution for this issue with respect to use of resources [2]. Only few articles are related to coverage measurement for FPGA prototyping. Some of the proposed methodologies could be used for both cases. Performed researches use only small group of assertions intended to test corner cases or known issues and don't take into account full verification of digital core. As a result there is no data related to resource utilization which is very important for prototyping. As [7] suggests more than 70% utilization of FPGA resources can cause performance issues.

Embedding synthesizable assertion into prototype could be an effective solution for the above mentioned issues. The article presents an approach for optimization of assertions to solve the resource utilization issue by adding reconfigurable regions into FPGA. All verification and coverage calculations are done for USB3.0 link layer. The synthesizable assertions are developed for transceiver (TX), receiver (RX) and Link Training and Status State Machine (LTSSM) sub-modules and for link interfaces (e.g., pipe interface, which is used between USB3.0 link and physical layers). The proposed approach can be used during FPGA prototyping of any digital core and will facilitate error detection/localization. The approach will make possible the calculation of functional coverage for FPGA prototyping which is a very important metric for quality of digital core.

2. USB3.0 LINK PROTOTYPE ARCHITECTURE
The architecture of USB3.0 link layer prototype is presented in Figure 2.1. Assertions are added for: TX controller, RX controller, LTSSM and for internal/external interfaces. These are most important and error prone modules of USB3.0 link and most of the scenarios described in [3] can be detected and verified by monitoring processes within these modules. Assertions for coverage and for checking correct behavior are used. The first type is used for coverage calculation as its name suggests. If the expected scenario does not happen after the start of assertion the state of the first type becomes IDLE and status "Not covered". The second type is used to detect the protocol or other functional violations, therefore, if the expected scenario doesn't happen the state of assertion becomes "ERROR". This assertions match with "cover" and "assert" statements in SystemVerilog Assertion (SVA) language. In the following sections only two assertions are presented for each module (one for "assert" type and one for "cover" type) because of size limitations.

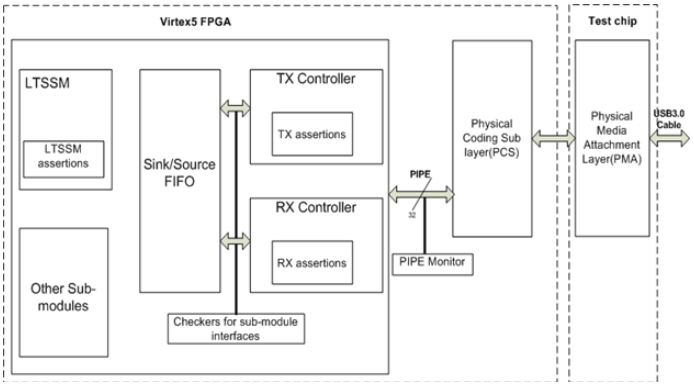

Figure 2.1 Prototype of USB3.0 link layer.

2.1 TX CONTROLLER ASSERTIONS

The number of developed assertions for coverage is 34 and there are 6 assertions used as checker for this module. First one is used to check if the length of a packet payload is not greater than 1024. It is "assert" type and will fail if data length is greater than 1024. The implementations of synthesizable RTL for these assertions are straightforward and is not presented here.

```
sva_u3link_tctrl_data_gt_1024:
assert property( @(posedge link_clk)
!(data_length > 'd1028))
else sva_link_error("The data payload is greater than 1024");
```

This assertion checks whether the port has "Loopback Master Capability" and is directed to go to Loopback. If so then it transmits identical TS2 ordered sets with the Loopback bit set upon entry to Recovery->Configuration.

```
sequence RecovConfg_TxTS2_SetLPBK; @(posedge link_clk)
(lt_link_state != RECOV) ##1 (lt_link_state == RECOV)
##1 (tx_state == TS_BLK) [->1]
##1 lndb_lpbken;
endsequence
svc_u3link_tctrl_RecovConfg_TxTS2_SetLPBK: cover
property (RecovConfg_TxTS2_SetLPBK)
```

2.2 RX CONTROLLER ASSERTIONS

The number of developed assertions for coverage is 31 and there are 5 assertions used as checker for this module. First one checks whether the port receiving a header packet sends an LBAD when after a valid HPSTART is detected and any K-symbol occurrence or any 8b/10b error is detected. This assertion is implemented as an assert type because according to protocol the link must respond with LBAD when there is a CRC5 error in link control work. Not doing so will cause a loss of data or corrupted packet header will be treated as correct.

```
sequence CRC5_error_TxLBAD_recovery; @(posedge mac3_clk)
(pkt_state != HDR3) ##1 (pkt_state == HDR3) ##0 !crc5_ok
##1 que_lbad
##1 replay_wait
##1 (lt_link_state == RECOV) [->1]
##0 replay_wait;
endsequence
svc_u3link_rpkt_CRC5_error_TxLBAD_recovery: assert property
(CRC5_error_TxLBAD_recovery)
```

The second assertion checks the scenario when one of end packet payload framing symbols is corrupted. As [5] states if one of four framing symbols is corrupted the link partner still must be able to recognize the end payload framing. This type of scenarios must be verified during hardware validation and this assertion can be used to check whether it is covered.

```
sequence dpp_endf_err_1; @(posedge mac3_clk)
(dpp_length[1:0] != 2'h0) ##1 (dpp_length[1:0] == 2'h0) ##0
(pkt_state == DPP_XFR)
 ##0 (match4(pkt_window_9b, {EPF, END, END, END}) == 3)
 ##1 ({lnmcr_data_last, lnmcr_data_err} == 2'b10);
endsequence
svc_u3link_rpkt_dpp_endf_err_1:
cover property (dpp_endf_err_1)
```

2.3 LTSSM ASSERTIONS

These assertions check whether the behavior of the LTSSM is according specification and if different functionalities are covered. The number of developed assertions for coverage is 332 and there are 12 assertions used as checker for this module. The first assertion checks if the link error counter is nullified after reset. This belongs to "assert" group because [5] defines that reset is one of the conditions when error count should become zero.

```
sequence reset_link_error_cnt_after_reset;  @(posedge link_clk)
ltmcs_link_err_ctr!=0
##1  !link_reset_n
##1 link_reset_n [->1]
##1 ltmcs_link_err_ctr==0;
endsequence
```

The second assertion checks if port upon receiving U1/U2 EXIT or U3 wakeup LFPS handshake signal,

then starts U1/U2 exit or U3 wakeup by responding with LFPS signal. This assertion belongs to "cover" group because other scenarios are possible.

3. SYNTHESIS RESULTS

Synopsys's Synplify Premier is used for synthesis. The synthesis is done for Virtex5 FPGA. Table 3.1 shows the utilized resources for each of the above described groups of assertions not including the logic for result retrieval. The number of slice registers is 91% and the number of slice LUTs is 33% of USB3.0 link design. This utilization is not acceptable as we have implemented assertions for only three modules. If we implement all possible assertions, the resource overhead will be bigger than Device under Test (DUT). Assertions for RX and TX controllers can be included without any change. The main overhead is due to LTSSM assertions.

Resources	TX assertions	RX assertions	LTSSM assertions	USB3.0 Link + PCS
Slice Registers	214	195	2087	2715
Slice LUTS	209	190	2036	7182
RAMB18X2SDP	-	-	-	1
RAMB36_EXP	-	-	-	5
RAMB18X2	-	-	-	1

Table 3.1 Utilized resources in Virtex5 (XC5VLX330T) FPGA

4. USING PARTIAL RECONFIGURATION TO SOLVE RESOURCE UTILIZATION ISSUE

FPGA technology provides ability to modify design without going through re-fabrication. Partial Reconfiguration (PR) takes this flexibility one step further, allowing the modification of design during FPGA operation by loading a partial configuration file. After a full bit file configures the FPGA, partial bit files can be downloaded to modify reconfigurable regions in the FPGA without change of the applications running on those parts of the device that are not being reconfigured [4].

The logic in the FPGA design is divided into two different types, reconfigurable and static logics. The USB3.0 link layer is implemented by static logic and the groups of assertions in each sub-module are implemented within reconfigurable region (Figure 4.1). The static logic remains functioning and is completely unaffected by the loading of a partial bit file. The reconfigurable logic is replaced by the contents of the partial bit file. There are many reasons why the ability to time multiplex hardware dynamically on a single FPGA device is advantageous. One of the benefits of PR is the reduction of size of the FPGA device required to implement a given function, with consequent reductions in cost and power consumption. In addition to reducing size, weight, power and cost, Partial Reconfiguration enables new types of FPGA designs that are impossible to implement without it. The assertion can be divided into groups taking as criteria the state of USB3.0 link or the test when they are applied. There are assertions for SS.Disable, Rx.Detect, Polling, Recovery, U0 and power saving state (U1, U2 and U3). It is possible to create a partial bit file for assertions of these states and swap the content of reconfigurable regions during USB3.0 link connect phase or low power mode entry/exit. The size of the reconfigurable region should be equal to the size of the biggest assertion group. This approach requires very high reconfiguration speed to be able to swap the partial bit file during transition of USB3 link from one state to another, but is not possible to reach with JTAG configuration mode, which we are using for current prototype. It is possible to do with different configuration modes which we are going to implement in future researches (e.g., embedding additional hardware to be able to do self-reconfiguration during operation).

We have chosen the second approach to create groups of assertions. Before running every test partial bit file of test related assertions are loaded into reconfigurable part of the FPGA via JTAG interface. Test developer defines the set of assertions, which should be covered by the test. Table 4.1 shows resource utilization for low power mode test which includes 13 LTSSM, 2 TX and 2 RX assertions for U1/2/3 states plus the overhead of logic for result retrieval.

Resources	TX assertions	RX assertions	LTSSM assertions	USB3.0 Link + PCS
Slice Registers	27	29	132	2715
Slice LUTS	23	16	202	7182
RAMB18X2SDP	-	-	-	1
RAMB36_EXP	-	-	-	5
RAMB18X2	-	-	-	1

Table 4.1 Results for low power mode test assertions.

CONCLUSION

Table 4.1 shows resource utilization of assertions for single test. The FPGA resource overhead for low power mode test is less than 2%. Proposed methodology gives opportunity to implement all needed assertions not only for link layer but also for all other USB3.0 modules, interfaces for USB3.0 physical layer and other cores. Partial Reconfiguration of FPGA enables dynamic switching of test specific assertions during USB3.0 operation.

In future research Internal Configuration Access Port (ICAP) controller will be implemented to automatically switch the content of reconfigurable parts of FPGA based on the state of USB3.0 (operational mode -U0, low power mode-U1/2/3 etc.), which will reduce human intervention during hardware validation.

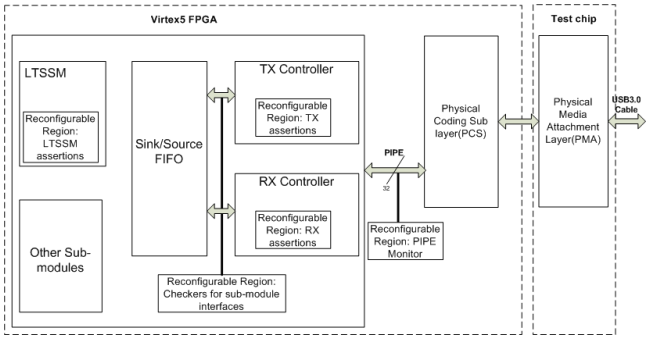

Figure 4.1 USB3.0 link prototype with reconfigurable regions

REFERENCES

[1] Mitra, Subhasish, Sanjit A. Seshia, and Nicola Nicolici. "Post-silicon validation opportunities, challenges and recent advances". In Proceedings of the 47th Design Automation Conference, pp. 12-17. ACM, 2010.

[2] Karimibiuki, Mehdi, Kyle Balston, Alan J. Hu, and Andre Ivanov. "Post-silicon code coverage evaluation with reduced area overhead for functional verification of SoC" In High Level Design Validation and Test Workshop (HLDVT), pp. 92-97, IEEE 2011.

[3] Universal Serial Bus 3.0 Link Layer Test Specification. August 2, 2013, www.usb,org

[4] Partial Reconfiguration User Guide (v14.5) April 26, 2013, www.xilinx.con

[5] USB3.1 specification July 26, 2013, www.usb.org

[6] F. I. Haque, J. Michelson, K. A. Khan. The Art of Verification With SystemVerilog Assertions, Verification Central, Fermont California 2006.

[7] D. Amos, A. Lesea, R. Richter. FPGA-Based Prototyping Methodology Manual, Synopsys Inc. 2011.