Getting Rid of False Errors when Verifying LSI Designs Including Non-Determinism

Matthieu Parizy
parizy.matthieu@jp.fujitsu.com

Hiroaki Iwashita
iwashita.hiroak@jp.fujitsu.com

Design Innovation Laboratory
FUJITSU LABORATORIES LTD.
Kawasaki, Japan

***Abstract*— The recent years have seen LSI design complexity continuing to rise sharply. This phenomenon translates itself in the design specifications as they include non-deterministic parts more and more frequently. For example, in cases of designs using packets for data transmission, packets transfer order is determined by precise rules. But, depending on the timing of the transactions, the final order might be hard to predict. It represents a big challenge in the verification field because traditional methodologies could lead to detect false errors which might actually be an early misunderstanding of the specs by the verification engineer or a failure to model the non-determinism in a Golden Reference Model (GRM). We present a new method of verifying designs including non-determinism in their specifications specifically the ones which use packet based communications. We create customizable plugins to provide reusability to the user. Furthermore, we introduce a margin parameter to control the degree of strictness of the verification.**

1. INTRODUCTION

On one hand, making a GRM that translates perfectly the non-determinism of the design specs is very time consuming. False errors will be often reported i.e. the GRM will output something different from the Design Under Test(DUT) whereas when we look at the design specs, the DUT's output is also correct. On the other hand, in the case of designs using packet based communications protocol, existing tools focus on verifying if a DUT's outputs are correct in regard to the protocol specifications. But the problem is: what can we do when the design specs include stricter rules than the protocol ones? For example, concerning performances in data transactions, a delay could be allowed in a particular transaction when we look at the protocol specs; whereas when we look at the design specs, this delay would be unaffordable because it is vital that this data arrives at a certain timing in another part of the design or else it would lead to consequent failures.

In this paper, we will talk about the solution we developed for the case of the verification of a design using the USB2.0 Enhanced Host Controller Interface (EHCI) [1]. When we conceived the methodology at the core of our solution, as we abstracted the problem, we realized it could also be used for any designs using packet based communications. Therefore, to improve productivity of verification engineers, we set our goal to develop a reusable and customizable monitor that will not output false errors. The core of the monitor consists of a verification process that can decide if packet order is in accordance to both the protocol in use and the design's specifications. It is based on customizable binary functions called plugins which represent the protocol and the design's specifications' rules concerning packet order of transfer which can be easily inserted or removed as the verification team gets familiar with the specifications. To implement our monitor, we chose OVM [2] [3] as its capabilities for adaptability and reusability made it a natural choice.

2. "REAL" AND "FALSE" ERRORS

During the functional verification process, we often encounter the following problems:

- By trading the signal/timing accuracy of the DUT with a more abstract model, less time consuming to realize, the possible outputs of the GRM might be within the protocol range but outside of the DUT possible operations [4].
- Protocol specific verification tools might allow a behavior of the DUT that could also be outside of the DUT possible operations.
- Because of the Non-Deterministic aspect of the design's specifications, the GRM and the DUT's output could be different although both being correct in regard to the protocol and the design's specifications. Thus it could mislead the verification team, when comparing the outputs of the GRM and the DUT for the same input vector, into finding false errors. (Figure1)

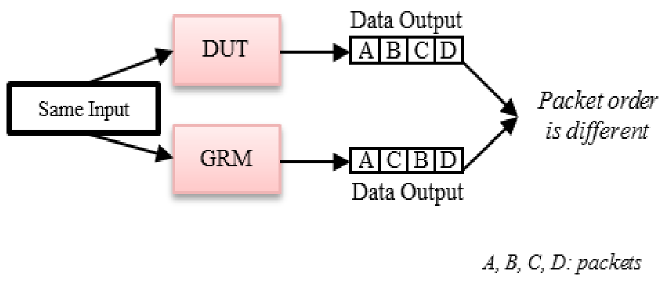

Figure 1. Using GRM with Non-Deterministic DUT

3. OUR APPROACH WITH OVM

Before considering using OVM for our verification needs, we started by trying to identify the source of the non-determinism in the EHCI specifications. After finding some clues, namely on the timing of packet sending in regard to USB frames and micro-frames, we started to model them using SystemC. The resulting module gave birth to what later became the "Smart Checker".

At the beginning, the SystemC module was only checking if the USB controller was sending packets in a predefined order set by us. We then improved our module by implementing a verification process able to judge by itself if the packet order is correct by setting the plugins according to transfers rules present in the protocol and in the design's specifications. These rules take the form of logical functions, later explained and named as plugins in this paper. Once it had reached that state, we decided to put it into practice with OVM on an already started verification effort. We chose SystemVerilog [5] +OVM because the language construct of the former combined with the methodology and set of classes provided by the later seemed to us the ideal candidate.

We based our research on a provided verification environment, developed for the verification of a designed based on the USB2 EHCI, by the verification team in charge at the time. The provided environment contains a monitor written in SystemC which assumed the task of checking the correctness of the DUT at a functional level with the help of transactors converting the signal from a RTL to Transaction Level. Wanting to use OVM as well as using the provided environment without rewriting it in SystemVerilog, we chose to use TLM to ensure communication between the SystemC monitor and our OVM scoreboard containing the Smart Checker. It is connected via TLM to the existing bus monitor. (Figure 2)

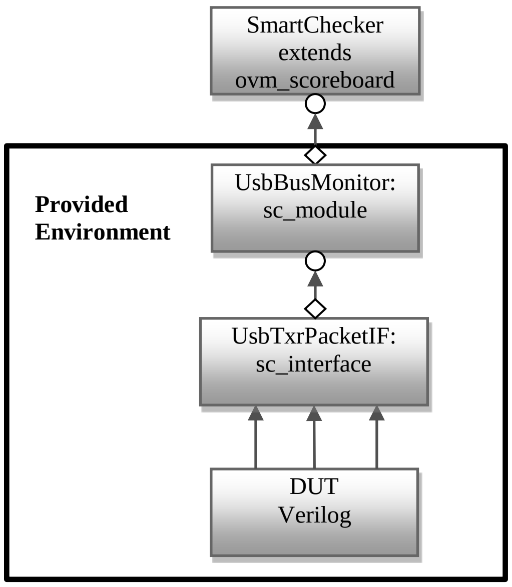
SmartChecker extends ovm_scoreboard
Provided Environment
UsbBusMonitor: sc_module
UsbTxrPacketIF: sc_interface
DUT Verilog

Figure 2. Smart Checker's position in the verification Environment

4. PLUGINS FOR ADAPTABILITY

Figure 3 shows the flow of the Smart Checker inner process:

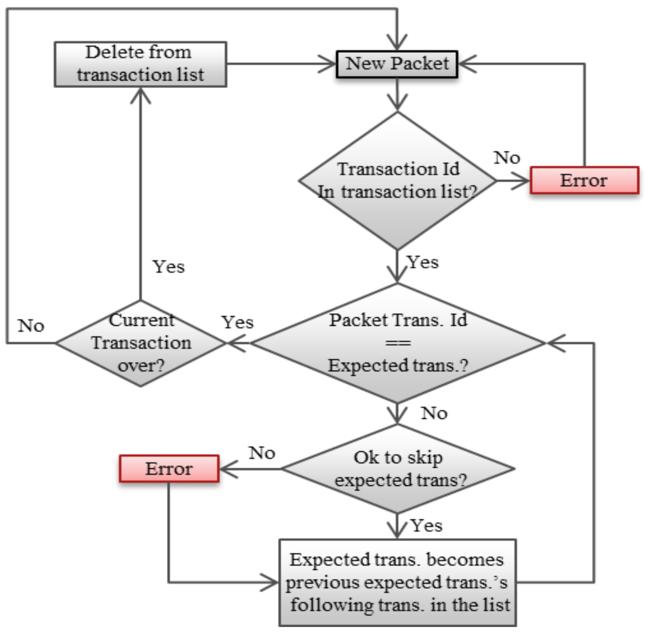
New Packet
Delete from transaction list
Transaction Id In transaction list?
No
Error
Yes
Yes
No
Current Transaction over?
Yes
Packet Trans. Id == Expected trans.?
No
No
Error
Ok to skip expected trans?
Yes
Expected trans. becomes previous expected trans.'s following trans. in the list

Figure 3. Smart Checker flow

- The Smart Checker first receives the information that a new packet came on the bus from the UsbBusMonitor.
- It then performs a check to see if the packet corresponds to an existing tracked transaction inside the Smart Checker.
- It then checks if the transaction identification of the packet corresponds to the transaction Id. that was expected. If it does, it waits for the next packet.
- If the received packet's transaction Id. was not the one expected, it calls the plugins to check if the expected transaction's packet can be skipped. If the skip is possible, no problem, it updates the expected transaction Id. with the next transaction Id. in the assumed order of transaction (set by the user).
- If the plugins did not allow for the expected transaction to be skipped, an error is signaled with packet information and we update in the same way as previously the expected transaction Id.

The plugins are the key components of the Smart Checker to solve the false error problem by answering, in our case, the "Ok to skip expected transaction?" in figure3. They are a set of logical functions in the monitor which represents the rules the design has to follow to perform transactions. These rules are part of the design's specifications above the protocol's one and represent conditions on which the monitor will base its judgment to tell if an error is real or not. (Figure 4)

```
skip_ok(packet_id) = not_enough_time_to_send(packet_id)
OR
user plugin
```

Figure 4. skip_ok logical function

In Figure 4, if in regard to the USB Host Controller (HC) specifications there is not enough time to send a packet before the end of a mircro-frame, the Smart Checker allows that packet to be skipped. The *user plugin* represent other rules that could be set to allow the packet skipping, for which an example is given after. By giving the plugins the form of logical functions, they are easy to remove or had in the smart checker program.

The built-in plugin *not_enough_time_to_send(packet_id)* has been implemented this way:

- In the USB2.0 EHCI specifications, it is stated that if not enough time is left in a micro-frame, a transaction A that could not be finished should not be started. Therefor allowing skipping it if there is a transaction B which could fit in the remaining time.

- To control the verification's degree of strictness with the Smart Checker, we introduced in this plugin a time margin parameter to ensure packets would be sent within micro-frame boundaries by checking the remaining time in the micro-frame minus the time margin is longer than the time needed to send a packet.

- As an example of use of the Smart Checker, we can start with a large margin and gradually reduce it to allow less and less packet skipping. We can then analyze errors and fix bugs that would increasingly show on the monitor as we progressively reduce the margin.

not_enough_time_to_send(packet): $T_{remain} - T_{margin} < T_{send}$(packet_id)

In the use scenario of figure 5 we have a design including the USB2.0 HC fetching data from the memory to send it over the USB to a device. Two transactions A and B are queued in that order but the fetching of data for transaction A is not over yet while B's fetching is. Thus in that case we want to allow skip of packets from transaction A and transfer packets from transaction B instead. Therefor we create a user plugin *packet_not_ready* which returns "true" if data fetching for a transaction is not complete.

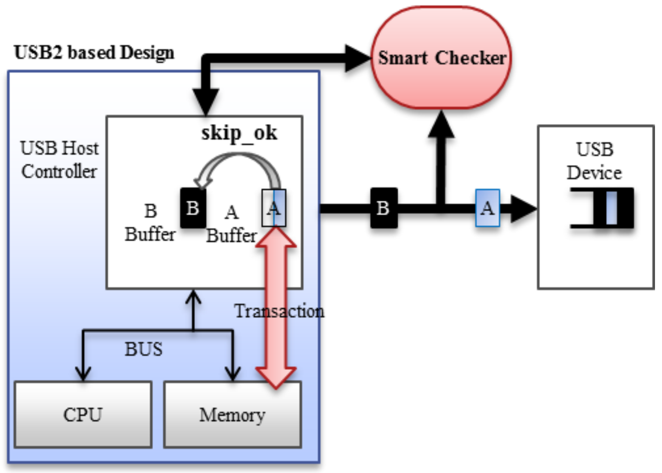

Figure 5. Example of utilization of Smart Checker

5. RESULTS

We solved the problem of getting rid of false error when verifying designs using packet based communications including non-deterministic parts in their specifications. We achieved creating a reusable verification monitor for that purpose. But some difficulties remain:

- Translate the protocol and design specifications rules concerning packet transfer correctly.

- Find all the rules in the specifications.

One problem we are facing right now is how to be able to set the plugins properly as a mistake in the translation of the specifications could lead to false errors or even worse: some errors being undetected. Especially, the difficulty is to answer the question "how do we know if the plugins are well set?"

6. ACKNOWLEDGEMENTS

We would like to thank Minoru Shoji from Fujitsu Semi-Conductors for the support he provided us.

7. CONCLUSION

With the rise of complexity in LSI designs, some of it due to non-determinism in its specs, the need for checkers to alleviate this problem is getting stronger. As developing these checkers is becoming harder and more time consuming, we naturally want to achieve reusability to save time in future verifications. The versatility of OVM helps us in achieving it by giving us a solid methodology we could rely on along with a great API.

Our Smart Checker is a customizable solution adaptable for the verification of any DUT using packet based communications. It efficiently removes false errors in the verification of packet communication based designs, specifically in our example, a USB2.0 based design, if the plugins are well set. But there is still a need for defining proper guidelines on how to translate efficiently the non-deterministic parts of the design and the protocol specifications into plugins for the Smart Checker to be fully operational.

8. REFERENCES

[1]Enhanced Host Controller Interface Specifications. http://www.intel.com/technology/usb/ehcispec.htm
[2] Glasser, Mark. Open Verification Methodology Cookbook. Mentor Graphics, 2009
[3]Open Verification Methodology. http://www.ovmworld.org
[4] Rensch, Josh., Prussi, Jesse. Effects of Abstraction in Stimulus Generation of Layered Protocols within OVM. DVCon, 2010
[5] IEEE-Computer Society. IEEE Standard for SystemVerilog Unified Hardware Design, Specification, and Verification Language, IEEE-Std 1800, 2005