A tailor-made checker for specific DV challenges

Tsu-Ting Shen, Yu-Juei Chen, Jack Yen

Mediatek Inc., Synopsys Inc.

Hsinchu, Taiwan

www.mediatek.com, www.synopsys.com

ABSTRACT

The importance of QOS (Quality of Service) is undeniable. As modern design grows bigger, the QOS becomes more complex. Correct communication between master and slave is crucial to every design.

We can easily detect functional issues on standard bus signals by simulation when their behaviors are incorrect or not as we expected. As for the sideband signals whose main purpose is to improve Quality on Service instead of implementing functionalities, we can do a thorough connection check by monitoring the commands sent from the master side and the data received by the slave side to guarantee their correctness. However, there still exist some bus signals that can't deploy these verification methods. In which case we can only rely on limited real scenarios or human efforts to catch them, until now.

In this paper we address how to do a tailor-made check for QOS side band signal Req/Gnt pair, which doesn't apply both of the previously mentioned verification techniques due to its nature. We developed a novel checker using VC App to solve this DV challenge. With the help of this checker, we can debug more efficiently and still guarantee the quality of our design.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The Advanced Micro controller Bus Architecture (AMBA) bus protocols is a set of interconnect specification from ARM that standardizes on-chip communication mechanisms between various functional blocks in order to facilitate us building high performance SOC designs. These designs typically have a large number of controller and peripherals. In order to construct the perfect interface structure that fits our Soc designs, we make use of various interface protocols such as APB, AHB, AXI …, etc. Each defines standard ports for functional signal transition, like address and data. ARM also allows users to define sideband signals for better QOS (Quality of Service) on these protocols.

The usage of Ultra/Flush signal pair is one example on how we utilize sideband signals to improve QOS. Figure 1-1 shows how the Ultra/Flush signal pairs are connected in our bus design. The mechanism of Ultra/Flush signal pair is quite simple. When Ultra signal arrives it indicates there is an urgent command sent from this channel, and the interface will send out a flush signal to make sure the commands queued in front of it are being executed first, so the urgent command can be executed quicker. If there are no other commands jamming the traffic before the Ultra specified command, there's no need to produce the flush signal because it will be executed right away.

However, we can't verify the connection of Ultra/Flush signal pairs by simulation. We use simulation to detect bugs on functional signals of interfaces when the verification pattern we designed had failed. But the incorrect connection of Ultra/Flush pairs can go undetected since they only have an impact on the performance, rather than the simulation results. Another way to verify bus signals is to monitor the commands sent from one side and see if it's identical to the receiving side. But the values of Ultra/Flush signals don't necessarily remain the same. Their value can only be determined by the traffic on the interface dynamically. So this is still not a proper solution. For a long time, we can only rely on human eyes to trace the connections one by one; or waiting for some real scenarios to catch them by chance.

In this paper we will demonstrate how to use VC App to develop a novel, elegant solution to solve this issue, and it can easily be deployed on other similar verification situations. VC App is a powerful tool which allows us to develop functional, customized connection checks based on Verdi engine that fits our requirements. Once we load our design, this VC App performs a thorough connection check and generates clear reports for designers to review.

The rest of this paper is organized as follows. Section 2 introduces all the connection issues that are hidden in our design, and what their impacts are. Section 3 addresses the proposed VC App solution. In section 4, we will demonstrate some of our experiment results to show the script is workable currently. Finally, section 5 concludes this paper and section 6 points to the future work.

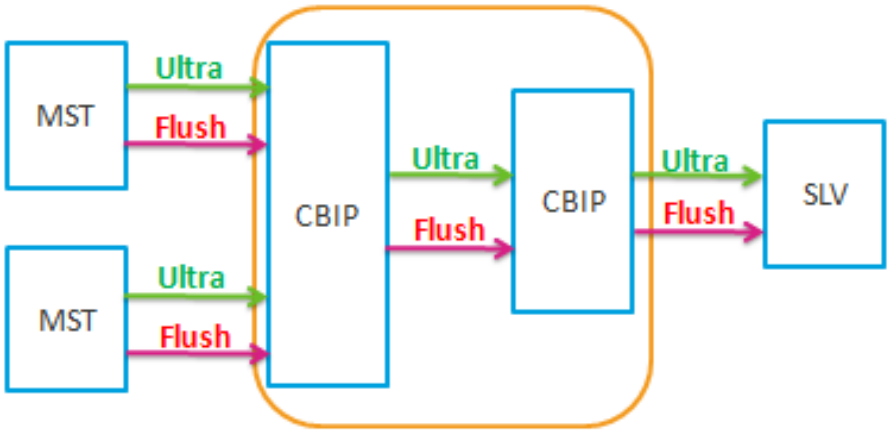

Figure 1-1 Ultra/Flush signal pairs relationship

2. Design Challenge

In this section we will introduce six scenarios we need to find in our bus design. Three of them are lethal and needs to be fixed. The other three won't necessarily cause issues but should still be reviewed by the designer.

2.1 Lethal Scenarios

These scenarios indicate incorrect ultra/flush connections and will lead to bad performance.

2.1.1 Ultra destination mismatches with corresponding Flush destination (most common)

Ultra and Flush signals should connected to the same destination or else they will not work.

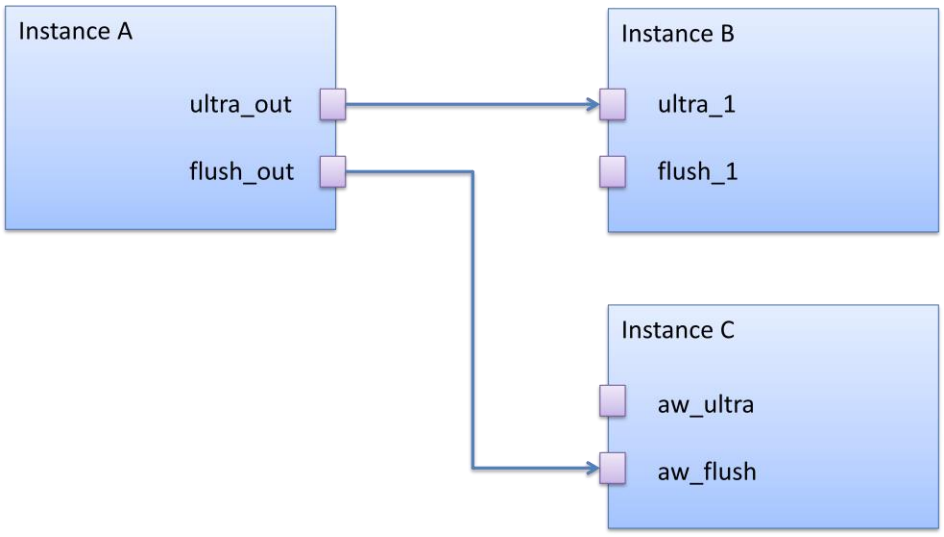

Figure 2-1 Ultra destination mismatches with corresponding Flush destination

2.1.2 Ultra destination matches corresponding Flush destination, but it is connected to a port with different naming prefix or postfix

Ultra and Flush signals should have the same naming rule.

Figure 2-2 Ultra port and Flush port is connected to different naming port

2.1.3 Ultra floating but corresponding Flush is not

When Ultra signal is floating, its corresponding Flush signal should be too.

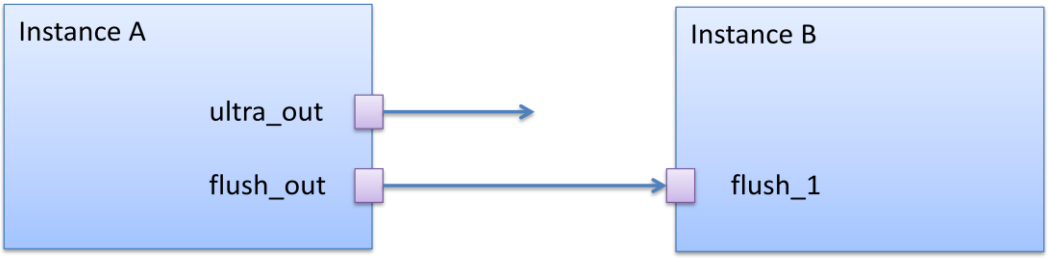

Figure 2-3 Ultra port is floating but Flush port is not

2.2 Threatening Scenarios

These scenarios are not correct ultra/flush connections but they might be constructed like this intentionally. They will not be a threat if the designer can guarantee the design.

2.2.1 Instance contains Ultra port but there's no corresponding Flush port (most common)

We have strict rules on Ultra/Flush port naming; each pair should follow the same naming rule.

Figure 2-4 Ultra port does not contain corresponding Flush port

2.2.2 Ultra port range != corresponding Flush port range

Ultra port should have the same width as the corresponding Flush port. Designer should guarantee the situation is designed intentionally.

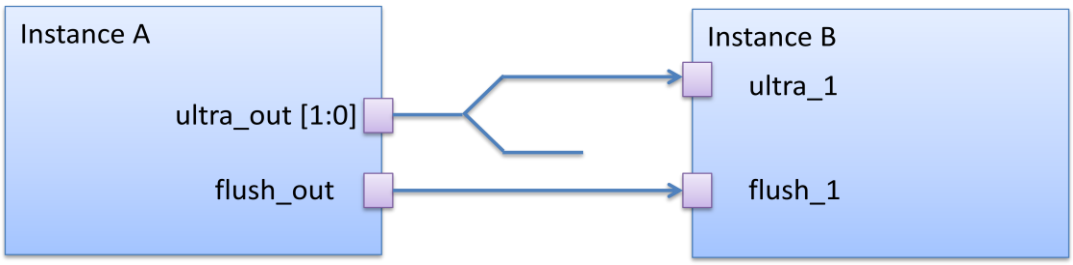

Figure 2-5 Ultra port width not equal to Flush port width

2.2.3 Input Ultra port is tied constant

Ultra port can only be tied constant when it has been reviewed by the designer.

Figure 2-6 Ultra port is tied constant

3. Proposed Solution

The main goal of our proposed solution is to establish an automatic, reliable, and general flow that can be applied to different interface designs and generate a straight forward report to alert designers the existence of the aforementioned scenarios. Human effort will be reduced from having

to check the thousands of connections one by one to mere report review.

The concept is to treat every Ultra signal as golden and verify whether the corresponding Flush signal is connected to the same destination, because Ultra can be guaranteed by system-level verification.

Using VC App, we're able to do achieve this goal through four stages: (1) finding target modules, (2) Ultra/Flush signal pair identification, (3) connectivity checking, and (4) report generation. Figure 3-1 shows the overall flow of our VC App. The script we develop also supports "waive list" when there's some modules we don't want to include in the checking.

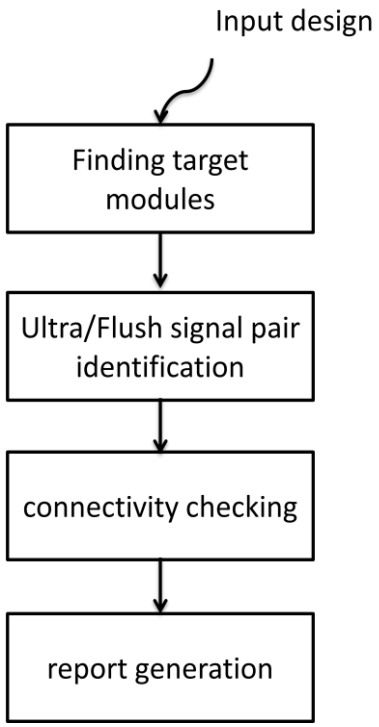

Figure 3-1 Ultra/Flush checker flow chart

- Finding target modules

  First we list all the modules that have ports with the keyword "ultra". If the module doesn't contain any ultra port, there's no need to include it. Then we exclude the ones that are specified by the waive list to avoid false alarms. Usually the waive list will be empty at first few runs, then after designer review, they can add the reviewed modules in the list.

- Ultra/Flush signal pair identification

  We use naming rule to identify the ultra/flush signal pairs. First select the ports that contain keyword "ultra" to be the signal pair candidate. Then we substitute the "ultra" with "flush", making the target flush port name with the same prefix and postfix as the ultra port. We search for this flush port in the module while making sure their port width and I/O direction also match. Figure 3-2 shows some of the signal pair examples. Uncovered ultra and flush port will be listed in the output report.

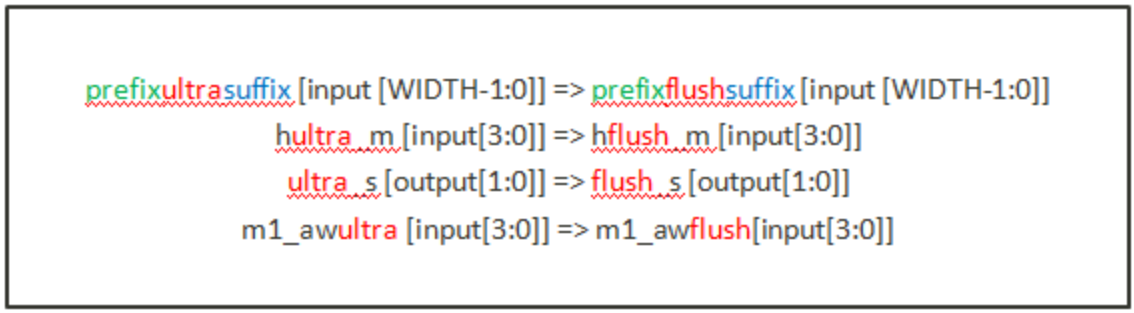

Figure 3-2 Ultra/Flush pair identifying mechanism

- Connectivity checking

  For all the pairs we found in the last stage, we trace the ultra port's destination to see if its corresponding ultra port also connects the same place. The connection of Ultra port is considered as golden, and the Flush port should follow its destination. Any mismatch will result in a record in the output report. Also, since the Ultra connection is expected to be clean, the tracing will stop when encounter any logic or DFFs and report this in the output. VC App provides convenient and efficient tracing functions to achieve this.

- Report generation

  We've divided the illegal scenarios into two groups, lethal scenarios as "ERROR" and threatening scenarios as "Warning". Each group includes 3 kinds of scenarios. There will be two final reports, one containing errors and the other one contains warnings that needed designer's checking. After reviewing the reports, designer can decide which action to take next. Figure 3-3 shows the usage model for designers.

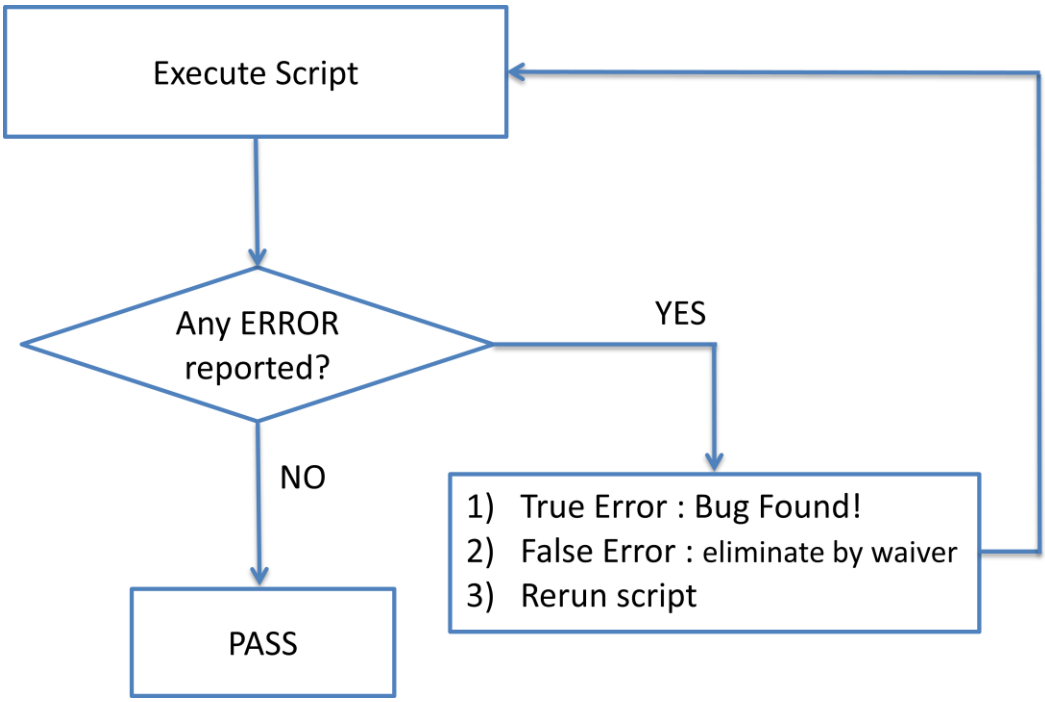

Figure 3-3 Ultra/Flush checker usage model

4. Experiment Result

Table 4-1 shows all the scenarios we found using this solution. We can see the number of cases found is quite large for human checking. Without the automatic solution it'll be almost impossible to catch these scenarios using the old ways.

Severity	Scenario	Cases Found
Error	UFE1 – Ultra/Flush destination mismatch	774
	UFE2 – Ultra/Flush naming mismatch	324
	UFE3 – Ultra floating but Flush not	2
Warning	UFW1 – Ultra not having corresponding Flush	5238
	UFW2 – Ultra/Flush range mismatch	4
	UFW3 – Ultra tied constant	50

Table 4-1 Illegal scenarios found by Ultra/Flush checker

5. Conclusion

Reliable and efficient interconnect architecture is essential to complex SOC designs that contain various functionality blocks. We need to design a thorough verification plan for all the bus signals in order to guarantee the communication between controller and peripheral is right. However, for QOS signals that their impact is subtle comparing to other functional signals, we need a different approach to verify them. Traditional simulation is not enough because they can only judge from the final results instead of the performance of each pattern. Nor can monitoring and comparing the send commands and received commands do the trick, because Ultra/Flush signals won't necessarily stay the same value at all times. Their value will depend on the situation of the interface at that time. Therefore, we make use of the very powerful tool VC App and Verdi engine to facilitate this troubling issue, to relief us from devoting too much human effort into this check and get a straight-forward report to review. And finally we are also working on deploying this checker to other tricky verification challenges.

6. Future Work

Using this proposed VC App solution to check the connections of special SOC signals is very powerful and thorough. But there's still some enhancement that can give simpler usage and better our user experience. Our future work is to combine this script with other checking mechanisms to complete an automatic flow; and to add verbose functionality to list all the paths that has been traced, giving a clearer idea of what has been checked by the Script.

7. References

[1] Yu-Juei Chen, Shang-Wei Tu, Tsu-Ting Shen, Himanshu Bhatt "Low Power Verification for Complex Analog Power Relation", SNUG 2016

[2] Verdi Interoperability Apps (VIA)