# Functional Verification of xHCI (extensible host controller Interface) for USB 3.1 Using HDL

[1]Mr. Dipesh Gehani, [2]Prof. Ketan N. Patel,

M.E. Student, Assistant Professor
Vishwakarma Government Engineering College,
Chandkheda, Ahmedabad

***Abstract:*** **As the requirement has been increased, the complexity of the electronic device design also has increased. Functional verification has become one of the major aspects in the design and verification flow. In order to respond to modern requirements, new devices are made of standard pre-verified reusable IP blocks. The same way, to verify the design the testbench should be created such that it should be reusable and self-checking. The paper proposes a method that should be used for verification based on a reusable test bench. The approach is demonstrated on an xHCI (extensible host controller Interface) which is the computer interface specification that defines how the communication happens between software and hardware in USB 3.x.**

***Key Words:*** **USB, xHCI**

## 1. Introduction

Now a day's electronic devices already combine a lot of different functions, the market incessantly demands new functionalities. In future, functionality of any single device shall have to be improved, meaning that its complexity should be drastically increased that requires more effort in its design and verification.

Verification consumes around 50% to 70% effort of the total design time. It has become very essential part in the design flow process of large ASICs containing multimillion logic gates. Hence it has become the bottleneck in the design flow.

A main purpose of the functional verification is to detect the failures so that bugs can be identified and corrected before it gets handed over to the customer. If RTL designer makes a mistake in designing or coding, that will result into the bug in the design and will lead to wrong results and hence failure. It may be possible that every bug may not result in failure because of the coding related to the one which is not used in the fabrication. Sometimes the bugs may also be in the specification or may come due to the miss communication in the design team. So functional verification is very important in the design process.

A **Universal Serial Bus (USB)** is "*a common interface that enables communication between devices and a host controller such as a personal computer (PC)*". It connects peripheral devices such as digital cameras, mice, keyboards, printers, scanners, media devices, external hard drives and flash drives. A USB is intended to enhance plug-and-play and allow hot swapping. Plug-and-play enables the operating system (OS) to spontaneously configure and discover a new peripheral device without having to restart the computer. As well, hot swapping allows removal and replacement of a new peripheral without having to reboot. USB host controller is structured as a layered architecture that contains 4 layers. xHCI is the top most layer in that structure. [1]

The paper is organised as follows. It will first give the basic idea about the layered architecture of USB host controller. Then will give the communication flow between software and hardware. That will be followed by the methodology used for verification. Then the discussion about one of the testcases has been done in results section.

## 2. USB 3.x Host Controller

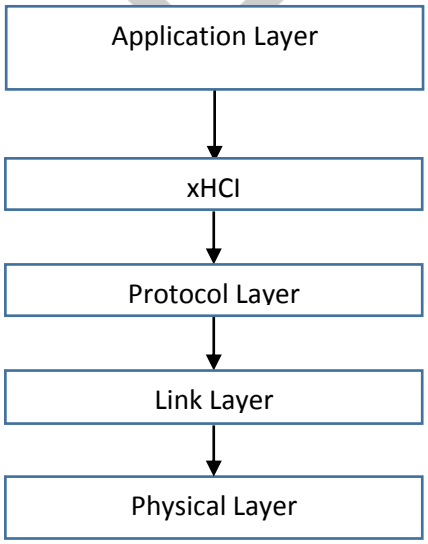


**Fig.1 USB 3.x Layered Architecture**

Fig.1 shows the diagram for layered architecture of USB 3.x. As shown here, the USB 3.x host controller consists of five layers of functionality which can be presented in many different ways. We will define layers as follows. The first layer takes care of the USB electrical part like serialization and new device detection (Physical Layer). The second layer composes the packet where the data is added the synchronization field (Sync) and "end of packet" (EOP) (Link Layer). Also responsible for LTSSM related activities. The third, Packet layer generates the packet data and makes the cyclic redundancy check (CRC) when required (Protocol Layer). The fourth, Transfer layer provides different types of transfers (xHCI). The fifth layer is the functionality layer for connection to the endpoints and application layers.
When certain application requires some data from a certain USB function, a pipe is established. The Transfer layer then calls a specific transfer type. The Packet layer prepares the data required for the operation and adds CRC. The packet is completed with Sync and EOP in the second layer. The first layer serializes the complete data and sends it to the slaves [3].

**3. Communication Between Hardware and Software**

The diagram in fig 2 shown below gives the basic understanding of how communication takes system software and xHC controller for different types of transfers which are explained in the following sections [2].

The no.s (1 through 9) indicate the stepwise flow of communication that is discussed below.

Every communication starts from system software side only. Here the communication between xHC drivber and xHC controller can be via AMBA AXI or AHB bus. And also communication with DRAM is also via these buses.

Understanding the flow:

1) Application layer tells the software layers below it for the different types of communication and about their properties.
2) These drivers will understand the request by the application layer and give according to that establish the environment and gives information to xHC driver.
3) According to the received information from the above software layers xHC driver will form the TRBs and put them on the DMA at required location.
4) Then xHC driver will inform the xHC controller that it has put TRBs on the DMA by ringing respective doorbell ring.
5) xHC controller than reads the TRB, decodes it and performs the operations required as per specification.
6) After performing the operations, xHC controller will put respective responses in the form of TRB at the required location in DMA.
7) Then xHC controller will let know the system software that it has put something in DMA by giving interrupt to it.
8) & 9) xHC driver will pass that response to upper software layers.

**4. Verification Environment**

Fig 3 shows the block diagram of verification environment. As shown in the block diagram it contains mainly three blocks [4].

1) USSP SSP TOP:
This is the top module of the environment where the instances of other two modules have been taken.
Testcase inputs are applied in this block, which are applied to the DUT via AMBA AXI-Master BFM.
AXI-Master BFM is used for the DMA related operations both by software and DUT for read and write operations. AXI-Slave BFM is used for MMIO related operations in xHC controller.
Also clock and reset signals are generated here and System memory is also there.

2) USB SSP:
This module contains the instances of AXI to communicate with AXI master and Slave BFM of xHC core, xHC core, xDC controller and other required for data transfer between xHC core and xDC controller.
Here is the data transfer takes place between host and device via PIPE (for USB 3.1) or ULPI (for USB 2.0) interface.

3) USB SSP Emulator:
This module is the emulator that is used for generation and transmission of data from device to host and also give the responses.
It contains two blocks. one is used for communication with xDC and second is used for generation of data.

Finally, the response received by environment is checked and according to that response is printed.

**5. Verification Methodology**

For verification purpose, two methods have been combined, task and function based verification and self-checking testbench based verification [6].

- **Task and function based verification**

-All the operations in are done using takes and functions. The task based BFM is extremely efficient if the device under test performs many calculations. Each task or function focuses on one single functionality. Verification of DUT using the task based testbench is faster. Using tasks makes it possible to describe structural testbenches. These tasks can be ported without much effort.

- **Self-checking testbench based verification**

Two important aspects of today's functional verification are quality and re usability. Design engineers have made design reuse to reduce development time and effort in designing an ASIC. Significant design blocks are reused from one project to the next. The lack of flexible verification environments that allow verification components reuse across ASIC design projects keep the verification cost very high. Considering the fact that verification consumes more resources than design does, it would be of great value to build verification components that are modular and reusable. When a design is passing all the tests in the verification environment, it has not been possible to know whether the design under verification is correct, and may be safely taped-out, or whether the verification environment is just incapable of finding any bugs that may still remain in DUT.

## 6. Results

45 Test case were made for verifying DUT using above method. One result has been shown in figure 4. This result is for checking whether chain bit set is supported for bulk OUT transfer or not. To check this, in Testcase first the endpoint was selected as bulk OUT. Then 4 kB of data was generated which needed to be transferred to the device. Here SCSI [5] protocol was used for transaction, so first command transfer for 31Bytes is done, then data is transmitted and then the status IN transaction is done to confirm that the data has been successfully transmitted. For data transfer TD (Transfer Descriptor) is formed which contains 4 TRBs (Transfer Request Block). Out of these, first 3 TRBs have the chain bit set and ioc bit is clear and the last TRB has chain bit clear and ioc bit set. So, interrupt of data transfer to software is generated for last TRB only which gives the idea about total data transfer.

Here, as shown in the result all the 4kB of data (shown by dark on both host and device side) is transmitted by host and received by device successfully. And also the event generated for data transfer contains the transfer length equals to zero, TRB type as transfer event and completion code as success, which is expected from xHCI specification [2]. So we can say that the DUT supports chain bit set condition for Bulk OUT transfer successfully.

## 7. Conclusion

The above method and verification environment was used for the verification using QuestaSIM as the tool. This environment was used for BULK transfer related test cases and found to be very much useful and efficient.

The following advantages were observed while using above method.

1) Speeds up verification and results in early tape out of the chip.
2) Less man power is required by which the overall cost of the project will be low.
3) Environment is reusable.
4) Easy tracking of verification progress.

## References

[1] Donovan (Don) Anderson, Vice President, MindShare, Inc.," Introduction to USB 3.0", Jan 14 -2014
[2] Intel Inc., "eXtensible Host Controller Interface for Universal Serial Bus (xHCI) specification" revision 1.1 , December 21, 2013
[3] "Universal Serial Bus 3.1 Specification", usb.org, Revision 1.0, July 26 2013
[4] Sibridge Technologies pvt. Ltd., "Orion_TestPlan.pdf" revision 1.003, December 1, 2015
[5] "Universal Serial Bus Mass Storage Class Bulk Only Support", usb.org, revision 1.0, September 31, 1999
[6] Sameer Palnitkar, A book on "*Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition*", Prentice Hall PTR, February 21, 2003

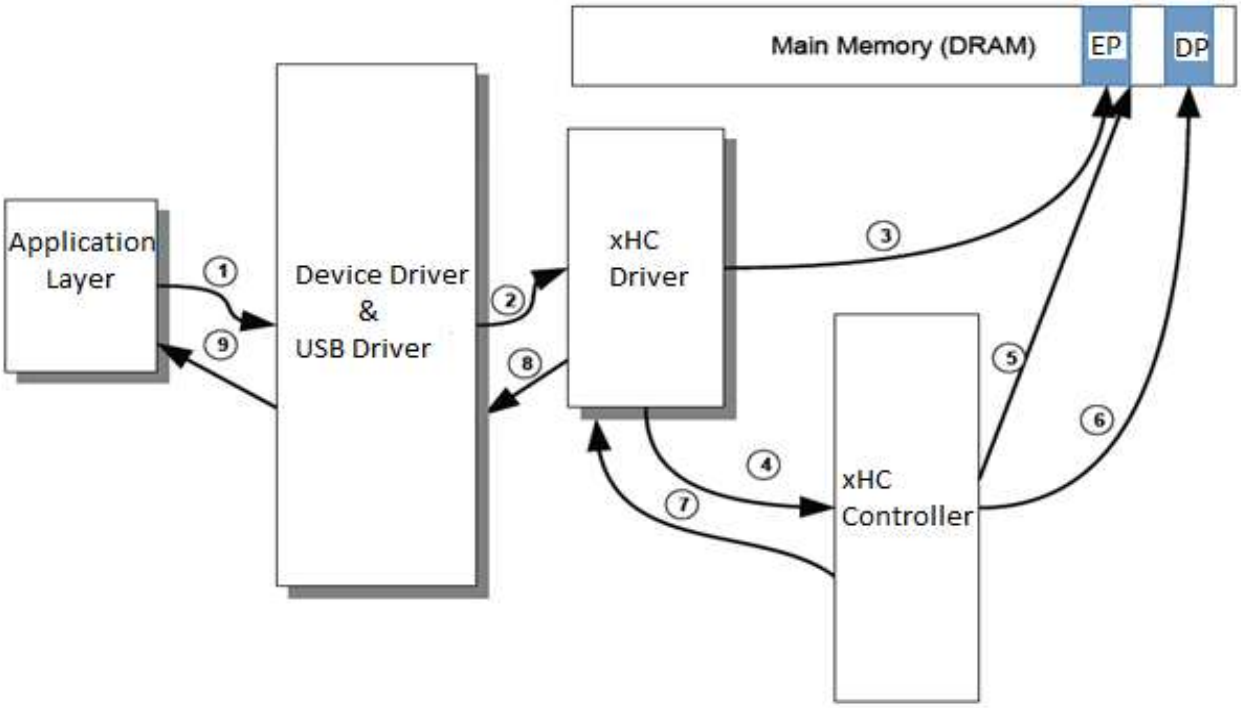


**Fig.2 Communication between Hardware and Software**

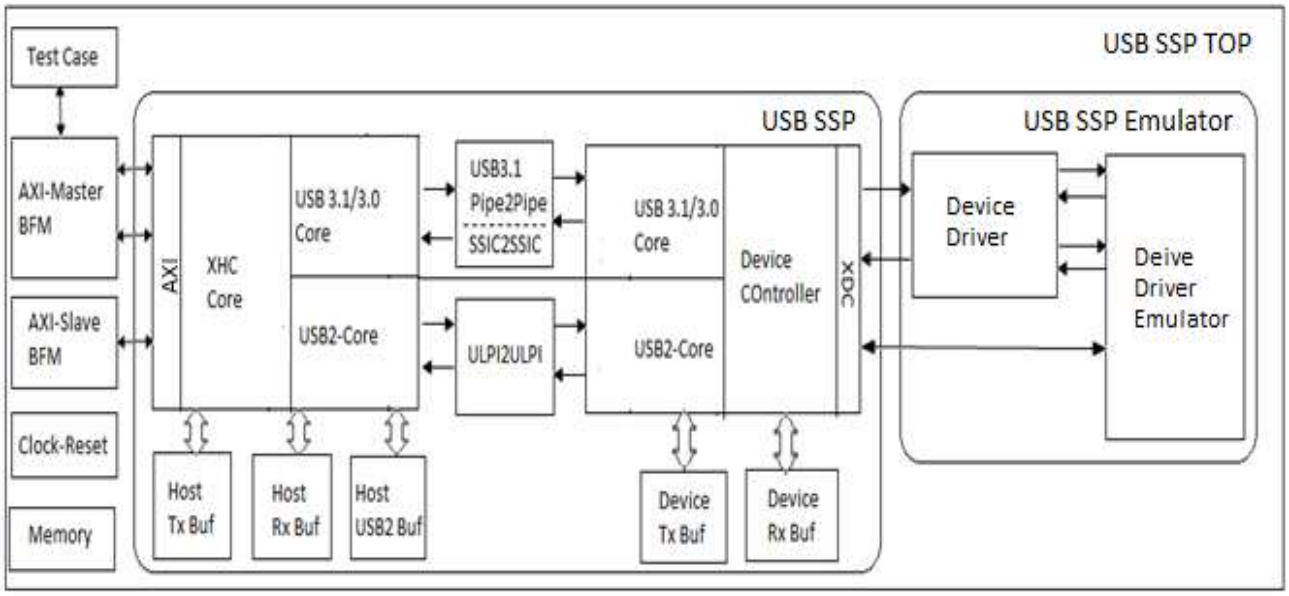


**Fig.3 Verification Environment**

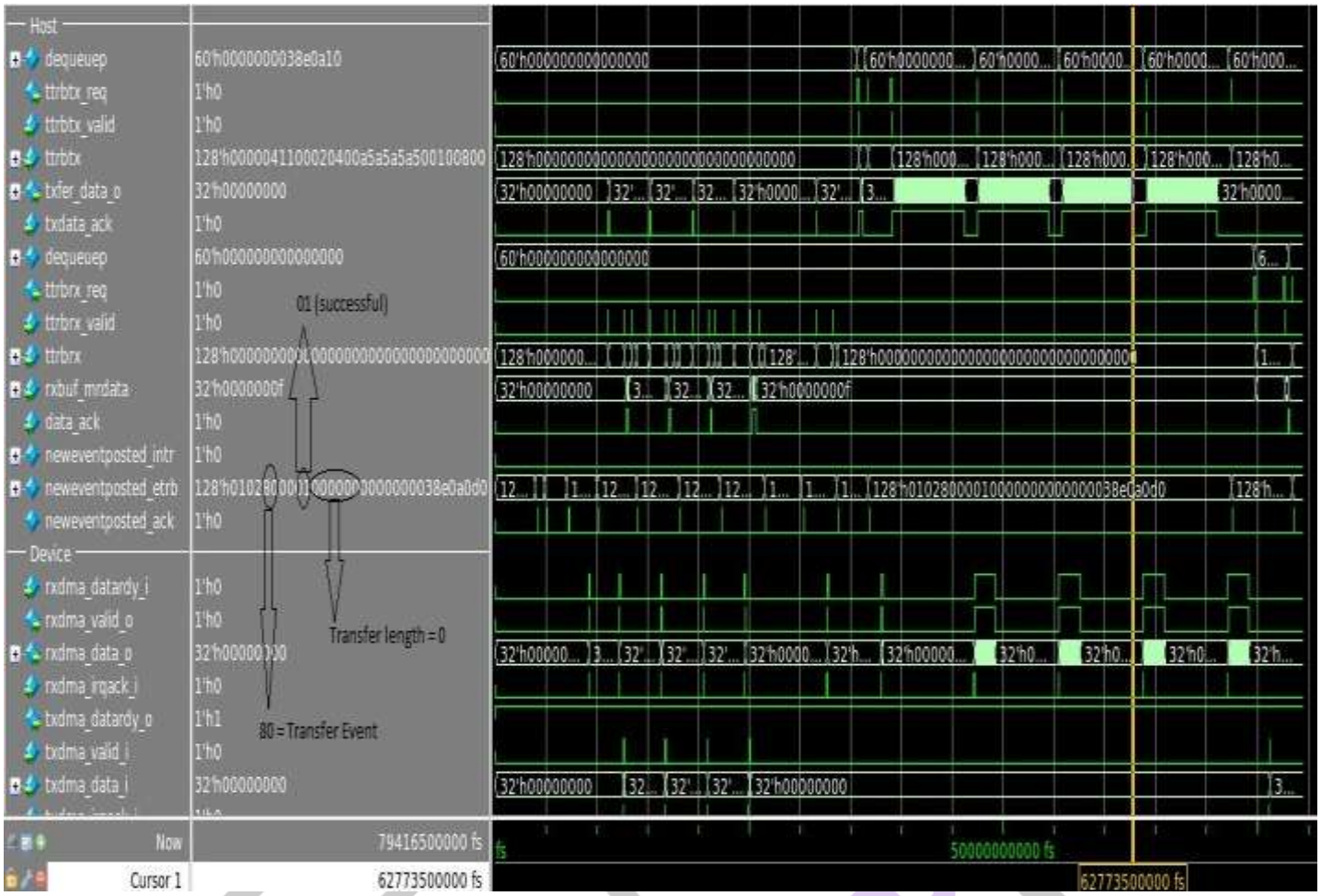


**Fig.4 Result for Bulk OUT transfer with chain bit set**