Verification of USB 2.0 Functional Core

Parth R. Zalawadia

M. E.

VLSI & Embedded System Design

GTU PG School Gujarat Technological University, Ahmedabad, Gujarat, India

Abstract---A verification environment to verify USB 2.0 Functional core using System Verilog is implemented in this paper. The verification IP can be reused to verify any USB protocol. Verification Environment is built to check the behaviour of DUT and to check whether we meet desired functionality of the core so that we can cover all the aspects of the design during verification. Verification of UTMI, Protocol Layer, Wishbone Interface and Memory Arbiter block is implemented in this paper. The Simulation Results are generated using QuestaSim 10.0b.

Keywords: Verification Environment, System Verilog, USB 2.0

I. INTRODUCTION

The Universal Serial Bus (USB) has evolved to the standard interconnect between computers and peripherals. Everything from a mouse to a camera can be connected via USB. With the new USB 2.0 specification, data rates of over 480 Mb/s are possible. The Universal Serial Bus is a point to point interface. Multiple peripherals are attached through a HUB to the host. This core provides a function (peripheral device) interface. This core fully complies with the USB 2.0 specification and can operate at USB Full and High Speed rates (12 and 480 Mb/s). [1]

SystemVerilog is a special hardware verification language to be used in verification. SystemVerilog provides high-level data structures and the notion of dynamic data types. SystemVerilog provides an object-oriented programming model. In this Paper, components of verification environment are implemented for UTMI, Protocol Layer, Wishbone Interface and Memory Arbiter block.

II. ARCHITECTURE OF USB 2.0 FUNCTIONAL CORE

In this Paper, for verification of USB core drawn an USB 2.0 design from opencores.org and the architecture is shown in Fig.1. The USB 2.0 Functional Core consists of 6 modules: Protocol Layer Module, UTMI Module, Memory Arbiter Module, SSRAM module, Physical layer interface module and Wishbone module.

UTMI block consist of Interface State Engine and Speed Negotiation Engine. The Interface state engine block tracks the interface state. It controls suspend/resume modes and Full Speed/High Speed switching. An internal state machine keeps track of the state and the switching of the operating modes and the Speed Negotiation block negotiate the speed of the USB interface and handles suspend and reset detection. [1]

Protocol Layer block consist of packet assembly, packet disassembly, and protocol engine block. Packet assembly block assembles packets and places them in to the output FIFO. It first assembles the header, inserting a proper PID and check sums, then adds a data field if requested. Packet Disassembly block decodes all incoming packets and forwards the decoded data to the appropriate blocks. The decoding includes extracting of PID and sequence numbers, as well as header check sum checking. Protocol engine block handles all the standard USB protocol handshakes and control correspondence. Those are SOF tokens, acknowledgment of data transfers (ACK, NACK, NYET), replying to PING tokens. [1]

The memory interface and arbiter arbitrates between the USB core and host Interface for memory access. This block allows the usage of standard single port Synchronous SSRAM. [1]

The SSRAM is a single ported Synchronous SRAM block that is used to buffer the input and output data. [1]

The host interface block provides a consistent core interface between the internal functions of the core and the function-specific host or micro controller. The host interface is WISHBONE SoC bus specification Ver. compliant. [1]

The Physical layer to complete the data transmission.

III. VERIFICATION ENVIRONMENT

The purpose of Verification Environment is to generate stimulus with the help of stimulus generator, which are sent to DUT (Design Under Test) by driver. And scoreboard compares the actual and expected output to verify that the function is correct.

IV. INTRODUCTION OF VERIFICATION ENVIRONMENT

Fig.3 shows the verification environment. The Environment include following components: Generator, Driver, Monitor, Scoreboard and DUT written by System Verilog language, System Verilog test bench which include interface, top module, and test program.

All rights reserved by www.ijsrd.com 253
Verification of USB 2.0 Functional Core

In System Verilog test bench the generator component generates stimulus which are sent to DUT by driver, and then the driver translate the operations produced by the generator into the actual inputs for the design under verification. The Monitor reports the protocol violation and converts the pin level activities into high level. Also the results can be checked by the scoreboard.

The fig.1 is divided into three parts of the verification environment. The first is the DUT, which is the design top module to be verified. The second is the system Verilog test bench. The third is the interface part used to joining the DUT and the System Verilog test bench. All this parts are instanced in the test top.

According to wb_we_i level, the system determines whether the data is reading or writing. When wb_we_i is high, main port writes data to WISHBONE interface, but when wb_we_i is low, port reads out data from WISHBONE interface. Data Communication rely on answering signal wb_ack_o: at the rising edge of the wb_ack_o signal the main port reads out data from data bus or writes data to data bus.

V. SIMULATION RESULTS

A. Simulation of Wishbone Interface

It first transmits data by using the WISHBONE master port’s function. When started working, the system is reset by the reset signal rst, when wb_stb_i and wb_cyc_i signal asserts it determines an effective transmission cycle by sending wb_cyc_i and wb_stb_i signals to WISHBONE interface.

B. Simulation of UTMI Interface
Initially when started working, the system is reset by the reset signal \( rst \), when \( \text{resume}_\text{req} \) signal and \( \text{rx}_\text{Active} \) signal is asserted then it transmit the data. It will check data was valid or not by \( \text{Rx}_\text{valid} \) signal. It will also check \( \text{tx}_\text{ready} \) signal that transmitter is ready to receive data. Here \( \text{Xcvr}_\text{select} \) signal is low which determines FS transceiver is enabled. \( \text{SuspendM} \) signal is low which determines Macro cell circuitry drawing suspended current. \( \text{LineState} \) signal is 01 which determines J state is selected so value of DP and DM is 0 and 1 respectively. Here the \( \text{OpMode} \) signal is 00 which determines core is operating in normal operating modes. \( \text{Mode}_\text{hs} \) signal is low which determines USB core is currently working in FS mode.

```
# *****SCOREBOARD : EXPECTED OUTPUT*****
# fc
#
# *****SCOREBOARD : OUTPUT FROM THE DUT*****
# fc
#
#
# *****SCOREBOARD : RESULT MATCHED*******
# 1
```

Fig. 7: Scoreboard output of UTMI Interface

### C. Simulation of Packet Assembly

Initially System is reset by the \( \text{rst} \) signal. The first step is to send the token and then data when \( \text{tx}_\text{ready} \) signal is asserted. Here corresponding \( \text{token}_\text{pid}_\text{sel} \) signal and \( \text{data}_\text{pid}_\text{sel} \) shows 00 and 01 which determines if it is 00 then token packets and if 01 then data packet. The second step is to check whether the token and data is valid or not using \( \text{tx}_\text{valid} \) signal. It also checks whether last token and data are valid or not using \( \text{tx}_\text{valid} \) last. And finally after completion of cycle it will read next token and data by asserting \( \text{rd}_\text{next} \) signal.

```
# *****SCOREBOARD : EXPECTED OUTPUT*****
# d2
#
# *****SCOREBOARD : OUTPUT FROM THE DUT*****
# d2
#
#
# *****SCOREBOARD : RESULT MATCHED*******
# 1
```

Fig. 8: Packet Assembly

```
# *****SCOREBOARD : EXPECTED OUTPUT*****
# fc
#
# *****SCOREBOARD : OUTPUT FROM THE DUT*****
# fc
#
#
# *****SCOREBOARD : RESULT MATCHED*******
# 1
```

Fig. 9: Scoreboard output of Packet Assembly

### D. Simulation of Packet Disassembly

When started working, the system is reset by the reset signal \( \text{rst} \). Initially it receives the data when \( \text{rx}_\text{active} \) signal is asserted. Here \( \text{rx}_\text{valid} \) is high which determines that data is valid. Signal \( \text{rx}_\text{err} \) is low which determines there is no error in the packet. Now it will decode all the packet and checks if it is token packet, data packet, handshake packet, special packet. Here \( \text{pid}_\text{SPLIT} \) signal is high which indicates it is special packet.

```
# *****SCOREBOARD : EXPECTED OUTPUT*****
# fc
#
# *****SCOREBOARD : OUTPUT FROM THE DUT*****
# fc
#
#
# *****SCOREBOARD : RESULT MISMATCHED*******
# 1
```

Fig. 10: Packet Disassembly

### E. Simulation of Memory Arbiter

```
# *****SCOREBOARD : EXPECTED OUTPUT*****
# fc
#
# *****SCOREBOARD : OUTPUT FROM THE DUT*****
# fc
#
#
# *****SCOREBOARD : RESULT MISMATCHED*******
# 1
```

Fig. 11: Scoreboard output of Packet Disassembly

```
# *****SCOREBOARD : EXPECTED OUTPUT*****
# fc
#
# *****SCOREBOARD : OUTPUT FROM THE DUT*****
# fc
#
#
# *****SCOREBOARD : RESULT MATCHED*******
# 1
```

Fig. 12: Memory Arbiter
In memory arbiter simulation result, there are three sub blocks: SSRAM, Memory and Wishbone. Fig. 7 shows the output of both memory block and wishbone block act as input to SSRAM so both will send req signal at the same time to sram to write their address and data at SSRAM location, but out of both block sram will select one block to process at a time. Here sram selects memory block to transfer their address and data. Here mwe signal is high which determines memory writes data and address to sram, and also sram_re signal is high which determines it reads the data and address from memory.

```#
###SCOREBOARD : EXPECTED OUTPUT####
# 1011101
#
###SCOREBOARD : OUTPUT FROM THE DUT#####
# 1011101
#
###SCOREBOARD : RESULT MATCHED#####
# 1
```

Fig. 13: Scoreboard output of Memory Arbiter

VI. CONCLUSION

The individual components of the verification environment have been written and verified using System Verilog Language and by performing verification of USB 2.0 using System Verilog we have cover all the test cases and seen the behaviour of our system and we have met the functional requirements of the core. And cover each and every aspects of the design during verification. The simulation has been successfully carried out.

ACKNOWLEDGEMENT

The submission of the paper gives me an opportunity to convey my gratitude to all those who have helped me in completion of my paper. I hereby would like to acknowledge the invaluable support of my project guide Mr. Ashish Prabhu, Staff Engineer at LSI Corporation Ltd Pune, and have been an unending source of inspiration for me. I am also thankful to other Technical Assistant members who have directly or indirectly helped me whenever it was required by me. I owe my special thanks to friends for supporting and encouraging me in all possible ways and also all faculty members of the Centre for Development Advanced Computing (C-DAC) and Gujarat Technological University (GTU) for providing facilities to access to IEEE Library which helped me in findings of my project.

REFERENCES

[2] Ku, Yi-Nan; Yang, Yu-Sen; Han, Yang, “Device interface protocol implementation based on USB2.0”, Jilin Daxue Xuebao (Gongxueban), vol. 3 pp. 170-173, March 2005