A New Approach of USB HW/SW Co-simulation and Verification

Grigor Y. Zargaryan, Vahram K. Aharonyan, Nazeli V. Melikyan and Marko Dimitrijević

***Apstrakt*—With everyday growing demands, complexity of electronic devices has been constantly increasing. Thus, simulation of register transfer level (RTL) model of controllers has become major aspect in the hardware design flow. Also hardware validation with software means is one of the main stages of verification. Development of this software usually starts when the hardware part is finished (or almost finished) and can be prototyped in an FPGA device or tapped out on other chip. So early software development and hardware validation based on virtual prototyping is getting more popular and reasonable. This paper describes the new approach of verification environment for USB 3.0 controller. The new methodology, where a co-simulation environment is used as one of the starting points for the embedded hardware/software development and as an accelerator to overall design process, is presented. Verification environment is based on device emulation/virtualization technique rather then using models instead of USB controller's real RTL, which makes it functionally very close to the corresponding real-world device and allows having more wide opportunities for hardware debug.**

***Keywords*—USB controller, simulation, verification, virtualization**

I. Introduction

The Universal Serial Bus (USB) is a fast, bidirectional, low-cost, dynamically attachable communication interface that is consistent with the requirements of the present microelectronic platforms [1]. It has been widely used in the huge number of instruments and devices, which include personal computers, digital cameras, scanners, image devices, printers, keyboards, mice, telephones, embedded systems, systems on a chip (SoC), etc. Figure 1 shows the high-level composing diagram of USB controller within a typical system.

During design flow of USB IPs for verification and functional debugging, field programmable gate array (FPGA) based techniques can be used. In this case hardware core is mapped onto an emulation platform based on a FPGA that mimics the behavior of the final chip, and the software modules are loaded into the memory of the emulation platform. Once programmed, the emulation platform enables the hardware and software combination to be tested and debugged at close to its full operational speed. In fact, FPGAs are used primarily to speed up and make easier some parts of the verification. Hence various components of USB system can be prototyped and verified using FPGAs. In [2] the development of USB peripheral core on the FPGA is described. USB protocol analyzer along with core itself was implemented to have a mean for verification of core. Also, for example, only UTMI logic can be designed by using VHDL code, then simulated, synthesized and programmed to the targeted FPGA [3]. But shortcoming of both [2,3] is that designed components were not fully validated in complete USB system (including host, USB SW stack, etc.).

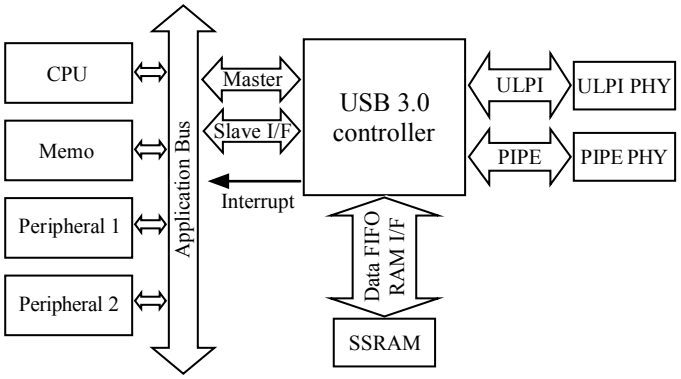

Fig. 1. System-Level Block diagram including USB controller

Another drawback of FPGA based HW verification could be the relatively slow debugging of signals when some problem in RTL pops up. If developer wants to look at some signals coming out from the core or inside it, he should do some modifications in RTL each time, connect appropriate wires to the debug pins of chip, re-synthesize logic, connect analyzing-oscillograph tools like Logic Analyzer and trigger the signals, or use JTAG cable for connect to FPGA and store necessary signals in RAM memory supported on FPGA (which requires more recourses) and then study signals from this store point. Also if try to go ahead directly with RTL validation in real system using FPGAs, problems can occur with absence of USB system's other components (USB PHY chip, programmable processor of SoC, memory access unit, etc.).

On the other hand, in order to actually validate and use the implemented USB hardware in applications run by SoC or any other system, the software layer that encapsulates underlying hardware and provides the application developer will be defined.

API (at minimum it is so called hardware driver or hardware abstraction layer; it often contains more software layers like modules of OS) should be developed. Thus only hardware subcomponents (in this case USB IP core licensed

Grigor Y. Zargaryan, Vahram K. Aharonyan, Nazeli V. Melikyan – Synopsys Armenia CJSC, 41 Arshakunyats Avenue Via Sphere Technopark, 0026 Yerevan, Armenia (e-mail: {grigorz, vahram, nazeli}@synopsys.com).

Marko Dimitrijević – Elektronski fakultet, Univerzitet u Nišu, Aleksandra Medvedeva 14, 18000 Niš, Srbija (e-mail: marko.dimtirijevic@elfak.ni.ac.rs).

by suppliers) slowly but steadily become a combination of hardware and software. Bare USB hardware IP is just not enough. Development of this software usually starts when the hardware part is finished (or almost finished) and can be prototyped in an FPGA device. This may considerably extend the project timeframe. Early software development based on virtual prototyping is getting more popular at system level but in the IP development it has not become a common practice yet. To overcome above mentioned problems during controller design flow the necessity of having complete emulation and simulation environment for USB core rises.

II. Background of known methods for simulation and validation of USB controller

However today, when SystemC [4, 5] has solved many of the software-hardware co-simulation issues, software based verification perfectly supports the new IP packaging requirements (i.e. requirement to deliver hardware dependent software along with the hardware component). So to have opportunity to see USB controller HW designed in faster manner and begin development of necessary drivers in first stage of design, various methods of modeling and verification has been suggested. In [6] USB device simulation using Microsoft Device Simulation Framework (DSF) is presented, which allow to quickly construct a virtual mass storage USB device, and have architecture for final RTL, but DSF did not provide perfect simulation and validation of USB and offers only smoke tests (i.e. file copy from and to). Paper [7] describes the way of simulation and validation system design for USB peripheral device's Verilog HDL code's testing using SmartModel tools which include functions of USB 2.0 host and UTMI. This environment tests main functions of device peripheral without SW stack (like Start of Frame packet generation, handling of Split transfers, ping protocol, etc.). Reference [8] presents a way of USB host controller verification environment using Transaction Level modeling (TLM) but actually is not cooperating with RTL and USB host software that are being passed as final package to the user. An example of design flow that includes real HW (described in RTL language) validation with SW stack is given only in [9]. Here as a first point SystemC TLM is used for hardware architecture definition and preliminary SW development, after that goes ahead with RTL verification and prototyping. But user-level test are being done only after USB RTL prototyping to the FPGA.

In this paper the new approach of HW/SW co-simulation and verification technique using two entries of Synopsys USB Super-Speed core acting as a host or device is proposed. It allows to have Verilog HDL code of USB core tested and validated with its SW drivers, whole Linux USB stack, run user level real-life applications (for either smoke or stress tests, specific protocol testing means [10]) which makes simulation very close to the real-life test and debug. Simulation environment's main idea is to make USB controller working in real-world circumstances before its tape out on FPGA or final chip. Here the USB host and device controllers using PCI bus are being integrated to the emulated x86 PCs running 3.6.3 Linux kernel. It provides developer a full access to all wires of RTL and look at any signal after simulation. Also chance of capturing and analyzing real sent and received packets on USB bus is given. PC systems are emulated using QEMU open source virtualization software which allows to easily working with verification environment considering it as a combination of two real-world Linux systems connected to each other via USB. Another important advantage of this method is that other USB controller (EHCI, OHCI, Synopsys UDC, USB 3.0 xHCI) RTL codes can be easily integrated into verification environment as well after specific changes in USB PHY model Verilog source code. Also, as Linux OS is running on emulated PC devices, the validation of these controllers with their drivers included in Linux kernel becomes suitable and does not require any specific change in SW stack.

III. QEMU virtualization SW

QEMU [11] is a processor emulator that relies on dynamic binary translation to achieve a reasonable speed while being easy to port on new host CPU architectures. In conjunction with CPU emulation, it also provides a set of device models, allowing it to run a variety of unmodified guest operating systems; it can thus be viewed as a hosted virtual machine monitor. It also provides an accelerated mode for supporting a mixture of binary translation (for kernel code) and native execution (for user code). It has two main modes of operation [11, 12]:

- User mode emulation - In this mode, QEMU can launch processes on CPUs that are compiled on another CPU. It is used to ease cross compilation or cross-debugging.
- Full system emulation - In this mode, QEMU emulates a full system (for example PC), including processors and several peripherals.

The second mode is used to launch different kinds of operating systems on the same PC, as virtual machines. This mode is used for building of co-simulation environment. One of the main advantages of QEMU is the fact that it runs in user space. There is no kernel module and therefore no adherence with the host system. It is executed like any other application. QEMU is a layer oriented application. The highest layer in QEMU is the lowest layer for the system inside the emulator (i.e. the hardware).

Basically, QEMU supports various processor architectures but in terms of this task Intel x86 is used. QEMU can emulate several hardware controllers such as CD-ROM, IDE hard disk interface, parallel and serial port, sound card, USB controller, etc. QEMU emulates a PCI UHCI USB controller. User is able to virtually plug virtual USB devices or real host USB devices and emulator will automatically create and connect virtual USB hubs as necessary to connect multiple USB devices. This is based on QEMU internal models of UHCI controller and is not allowing validation real-world USB controller.

IV. PROPOSED USB RTL/SW CO-SIMULATION APPROACH

The main difference and novelty of this approach from known verification/simulation methods is that using virtualization the real RTL implementation of the core (not TLM or any other model of the controller) is being co-simulated with SW and give chance to perform complete functionality tests on USB IP.

In Figure 2 the simplified block diagram of co-simulation environment is presented. Two entries of SS RTL core are connected to each other via PHY model that is written on Verilog. Advanced High-performance Bus Verification IP (AHB VIP) is there for memory read/write actions between the system and USB controller (handling any action regarding data reception/sending, direct memory access (DMA) actions, control and status registers' (CSR) read write, etc.). Using QEMU two x86 based PC systems are being emulated for host and device, and both are working with implemented Direct Programming Interface (DPI) which allows higher level Linux/QEMU C code to communicate with Verilog/Vera implementation and vice versa. User-Level application, device and host drivers and Linux OS make the software layer of environment. Under user-level application can be any testing tool that is being registered on the driver and installed in Linux per user's request. For example, USB test class driver and zero gadgets for host and device sides correspondingly can be used. Standard mass-storage stack is a good way as well. Implemented drivers are C-coded Linux USB driver controlling the work of SS cores.

One of the designed main components of verification environment is DPI layer that is bridge between SVerilog/Vera and C code domains. DPI consists of two layers: A SVerilog/Vera Layer and a C language layer. Both the layers are isolated from each other. DPI allows direct inter language function calls between the SVerilog/Vera and C language. The functions implemented in C language are called from SVerilog/Vera and such functions are called Import functions. Similarly functions implemented in SVerilog/Vera code domain can be called from C language and such functions are called Export functions. DPI allows transfer of data between two domains through function arguments and return. The main functions of DPI are responsible Forward DMA reads and writes initiated by RTL to memory in 'C' domain, report interrupt status to 'C' domain, execute processor ('C' domain) read/write to registers inside the core, control advancing simulation time while performing one of these actions. In Figure 3 the common mechanism of DPI implementation is shown – for example dma_read function that is implemented in C code domain should be imported to SVerilog/Vera code, and correspondingly after implementing ahb_read_vr task in SVerilog/Vera file, it should be exported there and declared as extern function in C code. The main specific thing here is that request from SVerilog/Vera domain to 'C' domain have highest priority and are implemented as blocking statements. They must be executed in '0' simulation time.

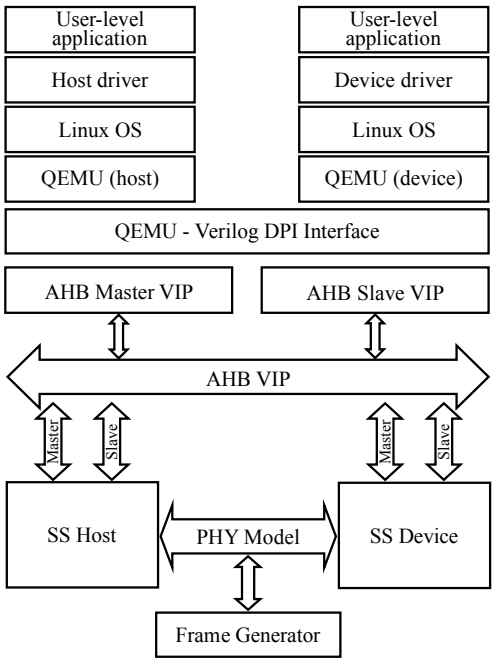

Fig. 2. Block diagram of co-simulation environment

C domain

- void dma_read(int addr, int* value) { };
- extern void ahb_read_vr(int*data, int addr, int length)

SVerilog/Vera domain

- import "DPI" context task dma_read(integer addr, var integer value);
- export "DPI" task ahb_read_vr { };

Fig.3. Direct Programming Interface

Co-simulation environment needs 3 processes running simultaneously: VCS simv, QEMU host, and QEMU device.

The flow of building and running developed environment is shown in Figure 4. It consists of following major steps:

- Compiling Linux kernel, QEMU C-sources using gcc. Compile RTL sources of USB controller, SS PHY model using Synopsys's VCS tool, integrate DPI (result will be simv simulation executable).
- Compile and install developed driver for USB controller, setup user-specified testing utilitys.
- Run three threads on Linux machine: QEMU Host, QEMU Device and simv executable(HW simulation)
- Load SW modules on QEMU emulated devices, run user-specified tests,
- Debugging using VPD file (figure 5 shows signal diagrams from RTL simulation), console logs.

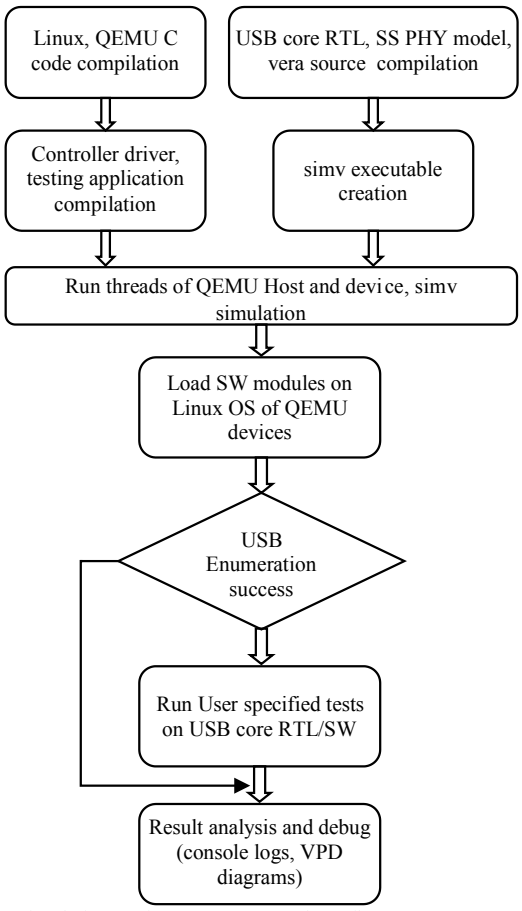

Fig. 4. Co-simulation environment setup and run flow

It is important to note that after running threads of QEMU host and device third thread responsible for simv running executes simulation of whole Verilog/Vera code domain including USB core RTL, then on both systems SS USB core is registered on PCI bus and system times are synchronized with simulation time to ensure absence of concurrencies between them and RTL simulation time – at this point virtually emulated PCs have fully functional SS USB controller under their services. While loading core drivers, both sides are requesting recourses from OS using communication interface with Linux PCI registered devices (mapping memory, interrupt line allocation, driver registering in OS, etc.) and go ahead with USB cores initialization. Afterwards USB controller of host detects device connection, notifies upper layers of Linux USB stack and initiates USB reset on bus to begin enumeration. After successfully passing enumeration, various user-specified tests can be run in environment to validate USB core RTL/SW combination.

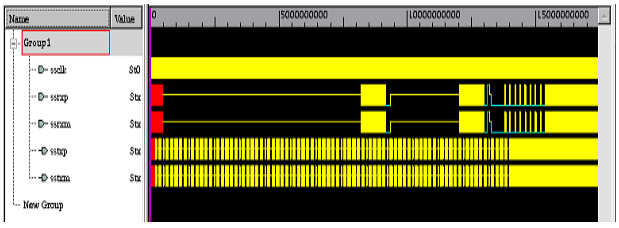

Fig.5. signal diagrams from RTL simulation on USB core

V. Statistics and Test Results

Co-simulation environment requires strong resources from host Linux machine. The analysis shows that major CPU load is caused by simv executable running. Profiling during simulation shows that about 50 percent of CPU utilization by simv is due to Verilog modules (core RTL design itself). The basic timing statistics of co-simulation and CPU utilization are follows:

- Compilation done (performing once during setup) – 7.2 min
- QEMU Host and Device booting – 2.5 min
- Driver loading –1.3 min
- Core Initialization by driver – 1.2 min
- Simulating 1mSec –3.6 sec

CPU Utilization Profile [%]:	20 min
Direct Programming Interface (DPI):	1.53
Programming language interface (PLI):	24.45
VCS for writing VCD and VPD files:	6.82
VCS for internal operations (KERNEL):	15.86
Verilog Modules:	50.69
A SystemVerilog testbench program block:	0.38
Garbage Collector (PROGRAM GC):	0.27

As it is clear from statistics analysis it takes around 3-4 seconds of real time to simulate 1ms of HW work. Hence basic file copy test between host and device systems via USB, with data size, for example, 1MB takes about 3 minutes of real-time, but based on simulation time copy is being performed with 46-60MB/s speed.

TABLE I
Linux usbtest utility test results in co-simulation environment

Test Description	Duration in Co-Simulation Environment		Duration in real FPGA based system (s)
	QEMU time (s)	Real time (m)	
Bulk write 1024 bytes 1000 times	0.010	0.58	0.014
Bulk read 512 bytes 1000 times	0.008	0.5	0.013
USB ch9 (subset) control tests	0.014	0.84	0.02
Queue 32 control calls, 1000 times	0.013	0.76	0.016
Set/clear 1000 EP halts	0.02	1.24	0.027
Isochronous write 1 MB 10 times	0.17	10	0.267
Isochronous read 1 MB 10 times	0.163	9.3	0.24

After driver loading and USB cores' initialization by SW on emulated host and device systems, various tests were run – control, bulk, isochronous and interrupt traffic generation in both IN and OUT directions. Also tests were performed using Linux's usbtest testing utility; some of obtained results are

captured in Table 1. For comparison real-world tests were performed in environment consisting of host and device cores prototyped on FPGAs installed on different systems with Intel i5 processors and connected via USB 3.0 cable. In the second column the duration of tests in co-simulation environment in terms of virtualization time is given and in the next column corresponding values in real-world time duration of it is given (it proves that 1ms of simulation time corresponds to 3.6s in real-world). But on the other hand, while comparing values of column 2 and 4 it is clear that in case of virtualization test duration is about 30 percent less than on the FPGA. This is because on the QEMU emulator there is no real PHY layer.

VI. CONCLUSIONS

In this paper new approach of USB controller RTL and driver co-simulation and verification method is proposed. Suggested method was realized based on virtualization techniques (QEMU emulator) using Synopsys SS core. Proposed method is based on the idea to use real RTL model of USB controlled instead of TLM or other abstract models. Designed environment allows to easily Identify/debug USB HW design and driver bugs before real tests on FPGA or SoC, reproduce and debug issues seen in the real circumstances using signal diagrams obtained from simulation, begin SW driver development and testing specially when no HW is available. The method is flexible and doesn't depend on USB controller and any other USB IP's RTL can be easily integrated to the environment after minor changes in USB PHY model.

REFERENCES

[1] Universal Serial Bus 3.0 Specification. Rev 1.0. - June 2011, www.usb.org

[2] De Maria, E.A.A., Gho, E., Maidana, C.E., Szklanny, F.I., Tantignone, H.R., "A Low Cost FPGA based USB Device Core", IEEE 4th Southern Conference on Programmable Logic, 2008, P. 149-154.

[3] Babulu, K., Rajan, K.S., "FPGA Implementation of USB Transceiver Macrocell Interface with USB2.0 Specifications", ICETET '08. First International Conference on Emerging Trends in Engineering and Technology, July 2008, P. 966-970.

[4] IEEE Computer Society, *IEEE Standard System C Language Reference Manual*, 2005, 423p.

[5] OSCI, Transaction Level Modeling (TLM) Library, Release 1.0, 2005

[6] Anderson, R.B., Borowczak, M., Wilsey, P.A., "The Use of Device Simulation in Development of USB Storage Devices", IEEE 41st Annual symposium, 2008, P. 220-226.

[7] Xiaoping, B.; Shenlei, "The Study on System Verification of USB 2.0 Interface Protocol Control Chip Hardware Design", ICEE '07. International Conference on Electrical Engineering, 2007, P. 1-5.

[8] Puhar, P., Zemva, A., "Functional Verification of a USB Host Controller", IEEE 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, 2008, P. 735-740.

[9] Sobański, I.; Sakowski, W., "Hardware/software co-design in USB 3.0 mass storage application", IEEE International Conference on Signals and Electronic Systems (ICSES), 2010, P. 343-346..

[10] USB Testing on Linux, March 2007, http://www.linux-usb.org/usbtest/.

[11] QEMU open source processor emulator, http://www.qemu.org, 2013.

[12] Ribière, A., "Emulation of obsolete hardware in open source virtualization software", IEEE 8th IEEE International Conference on Industrial Informatics (INDIN), 2010, P. 354-360.