USB HW/SW Co-Simulation Environment with Custom Test Tool Integration

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

***Abstract*—This paper describes a new verification environment for USB 2.0 controller. New methodology is presented, where a co-simulation environment is used as one of the starting points for the embedded hardware/software development and as an accelerator of the overall design process. The verification environment is based on the device emulation/virtualization technique, using USB controller's real register transfer level (RTL) instead of models. This approach is functionally very close to the corresponding real-world devices and allows wider opportunities for hardware debugging. The new software utilities for USB host and device functionality testing are also presented. This tool allows generating custom tests by including various transfer types and modifying parameters such as data payload, interval, number of pipes, etc. It can be used for both hardware (HW) and software (SW) limitations characterization, as well as debugging.**

***Index Terms*—Co-simulation, FPGA, System C, QEMU, USB**

Original Research Paper
DOI: 10.7251/ELS1418023G

I. INTRODUCTION

THE Universal Serial Bus (USB) is a fast, bidirectional, low-cost, dynamically attachable communication interface consistent with the requirements of the present microelectronic platforms [1]. It is 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. Fig. 1 shows the high-level architecture diagram of USB controller within a typical system.

Field programmable gate array (FPGA) based techniques can be used for verification and functional debuging in design flow of USB IPs. In this approach, hardware core is mapped onto an emulation platform, based on an 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 testing and debugging of hardware (HW) and software (SW) components, close to system's full operational speed. In fact, FPGAs are used primarily to speed up testing and make some parts of the verification easier. 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 and the core were implemented for verification. For example, only UTMI logic can be designed using VHDL code, then simulated, synthesized and programmed to the targeted FPGA [3]. But the drawback of both approaches [2], [3] is that designed components were not fully validated in complete USB system (including host, USB SW stack, etc.).

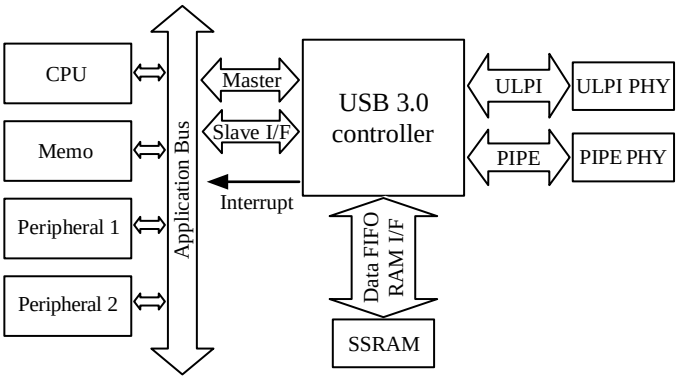

Fig.1. System-Level Block diagram including USB controller.

Another drawback of FPGA based HW verification can be relatively slow debugging of signals, when some problem in register transfer level (RTL) emerges. If the developer wants to look at some signals generated in the core, he needs to perform some modifications in RTL, connect appropriate wires to the debug pins of chip, resynthesize logic, connect analyzing osciloscope tools like Logic Analyzer and trigger the signals or use JTAG cable to connect to FPGA and store necessary signals in RAM memory supported (which requires more resources) and finaly study signals from this store point. If the developer is trying to proceed directly with RTL validation in real system using FPGAs, problems can occur with the absence of other USB system's 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 running by SoC or some other system, the software layer that encapsulates the underlying hardware and provides the application developer defined API (at minimum it is the so called hardware driver or the hardware abstraction layer; often it contains more software layers like OS modules) should be developed. Thus, only hardware subcomponents (in this case USB IP core licensed by suppliers) become a combination of hardware and software.

Manuscript received 13 April 2014. Accepted for publication 30 May 2014.

G. Y. Zargaryan, V. K. Aharonyan, N. V. Melikyan are with the Synopsys Armenia CJSC (e-mail: {grigorz, vahrama, nazeli}@synopsys.com).

M. A. Dimitrijević is with Faculty of Electrical Engineering, University of Niš, Niš, Serbia, (+381-18-529-321; e-mail: marko@venus.elfak.ni.ac.rs).

In this case, bare USB hardware IP is 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 the above mentioned problems, the necessity arises to have a complete emulation and simulation environment for USB coreduring controller design flow, .

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

SystemC [4], [5] has solved many of the software-hardware co-simulation issues today, and software based verification perfectly supports the new IP packaging requirements (i.e. requirement to deliver hardware dependent software along with the hardware component). Therefore, to design a USB controller HW in a faster manner and to begin the development of necessary drivers in the first stage of the design, various modeling and verification methods have been suggested. In [6] USB device simulation using Microsoft Device Simulation Framework (DSF) is presented, which allows quick construction of a virtual mass storage USB device having final RTL architecture, but DSF does not provide sufficient capabilities for USB simulation and validation, 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 using SmartModel tools which include functions of USB 2.0 host and UTMI. This environment tests main functions of device peripheral without SW stack (Start of Frame packet generation, handling of Split transfers, ping protocol, etc.). Paper [8] presents the USB host controller verification environment using Transaction Level modeling (TLM), but this environment actually does not cooperate with RTL, nor USB host software, delivered as the final package to the user. Only in [9] an example is given of design flow that includes real HW (described in RTL language) validation with SW stack. Here, SystemC TLM is used for hardware architecture definition and preliminary SW development, in the first step. RTL verification and prototyping is the next step. User-level test is done only after USB RTL prototyping to the FPGA.

In this paper, a new approach to HW/SW co-simulation and verification technique is proposed, using two entries of USB Super-Speed (SS) core acting as a host and device. It allows testing and validation of Verilog HDL code of USB core together with depending software (SW) drivers, whole Linux USB stack, running real user level applications (for either smoke or stress tests, or specific protocol testing means [10]), which makes simulation very close to real testing and debugging. The main role of the simulation environment is to make the USB controller work in real physical world conditions prior to implementation on FPGA or ASIC. The USB host and device controllers are attached to the emulated x86 PCs running 3.6.3 Linux kernel using PCI bus. The simulation environment provides full access to all signals of RTL, during and after simulation. The possibility of capturing and analyzing data packets sent and received over USB bus is also provided. PC systems are emulated using QEMU open source virtualization software which works straightforwardly 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 the other USB controller (EHCI, OHCI, USB 3.0 xHCI) RTL codes can be easily integrated into verification environment as well, after specific changes in Verilog source code of USB PHY model. As Linux OS runs on emulated PC, the validation of these controllers with corresponding drivers included in Linux kernel becomes suitable and does not require any specific changes in the SW stack.

The testing of complete USB hardware and software environment is performed using certification tools – USB CV, USB GoldenTree environment, WHCK, etc. [11], [12]. A majority of these tools run on the final product and cannot be used during early stages of the USB controller design. For Linux systems there is a combination of *ustbtest* class and *g_zero* gadget drivers [10], which allow generating traffic on the various pipes of the host and device controllers either on final product or using co-simulation environment. Actually, this tool is for USB 2.0 mode only and does not allow testing various configurations and interfaces, changes of test parameters, etc. This paper also presents new testing tool integrated within suggested co-simulation environment that provides flexible interface for generating custom tests running in USB 2.0 and 3.0 modes, gathering statistics on the test results and observing failures.

III. QEMU virtualization software

QEMU [11] is a processor emulator that relies on dynamic binary translation in order to achieve a reasonable speed. QEMU is easy to port on new host CPU architectures. In conjunction with CPU emulation, it also provides a set of device models, ensuring possibility 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 [12], [13]:

- User mode emulation - 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 - 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 to build 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 also 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 which is necessary to connect multiple USB devices. This is based on QEMU internal models of UHCI controller, and it allows to validate real 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 the real RTL implementation of the core (not TLM or any other model of the controller) is co-simulated using virtualization, giving a chance to perform complete functionality tests on USB IP.

The simplified block diagram of co-simulation environment is presented in the Fig 2. Two entries of SS RTL core are interconnected via PHY model, written in Verilog. Advanced High-performance Bus Verification IP (AHB VIP) performs memory read/write actions between the system and the 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 emulated as host and device, and both work 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 as well as Linux OS make the software layer of environment. User level USB testing application is registered on the driver and installed in Linux.

One of the main components of designed verification environment is the DPI layer, serving as a bridge between SVerilog/Vera and C code domains. DPI consists of two layers: A SVerilog/Vera Layer and a C language layer. Both 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, such functions are called Export functions. DPIs allow data transfer between two domains through function arguments. The main functions of DPI are 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 Fig. 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 requests from SVerilog/Vera domain to 'C' domain have highest priority and are implemented as blocking statements. They must be executed in '0' simulation time.

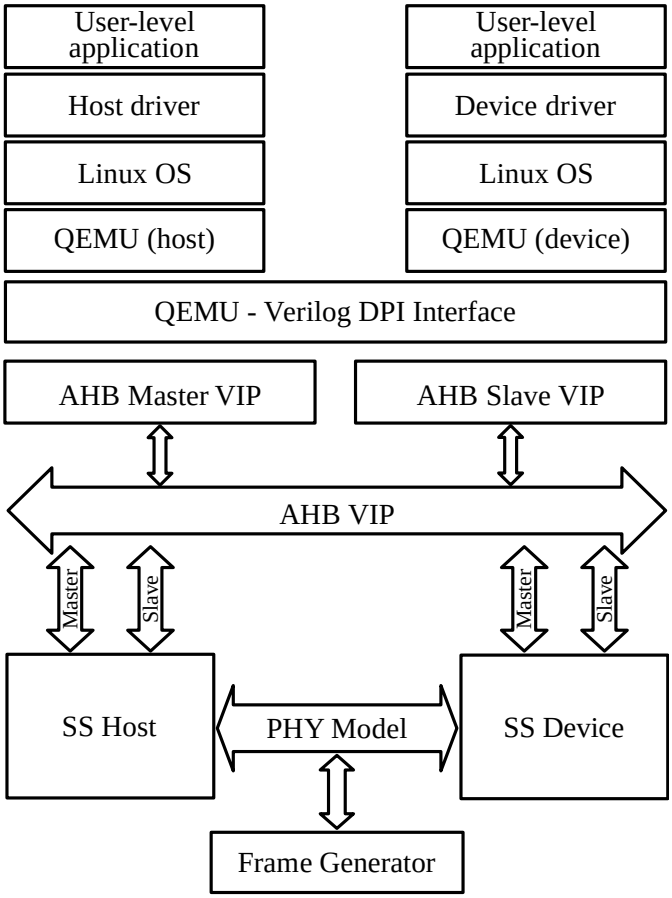

Fig. 2 Block diagram of co-simulation environment.

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 runs in three processes simultaneously: VCS *simv*, QEMU host, and QEMU device.

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

• Compiling Linux kernel, QEMU C-sources using gcc. Compile RTL sources of USB controller, SS PHY model using VCS tool, integrate DPI (result will be *simv* simulation executable).

• Compiling and installing the developed driver for USB controller, setup USB user level testing utilities.

• Running three threads on Linux machine: QEMU Host, QEMU Device and *simv* executable (HW simulation)

• Loading SW modules on QEMU emulated devices, run user-specified tests,

• Debugging with the use of VPD file (signal diagrams from RTL simulation are shown in Fig. 5), console logs.

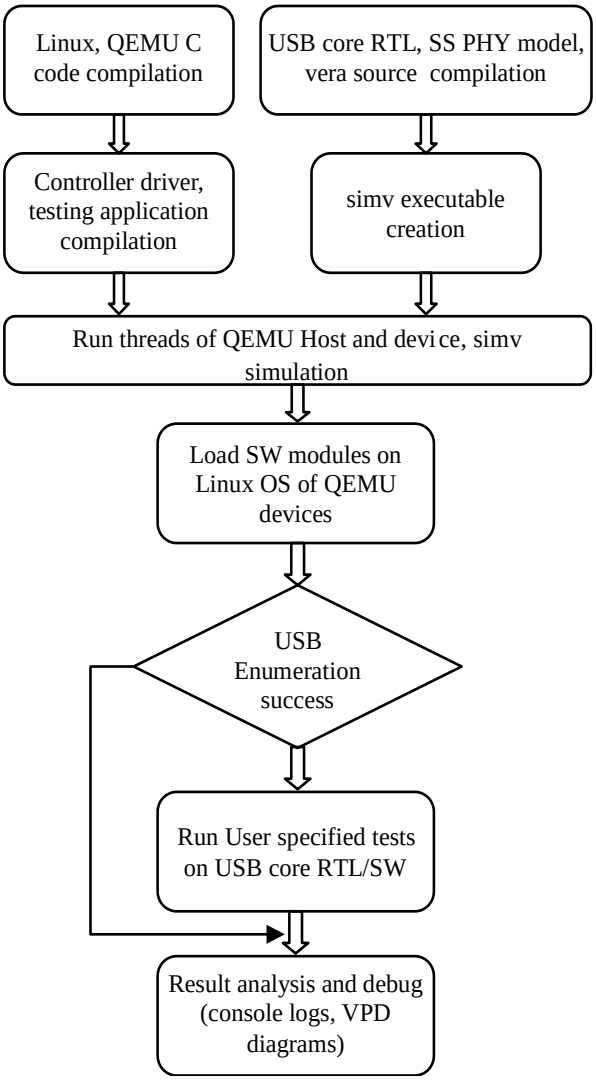

Fig. 4 Co-simulation environment setup and run flow.

It is important to note that after running QEMU host and device threads, third thread responsible for running *simv* executes simulation of the whole Verilog/Vera code domain including USB core RTL. 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 on their disposal. While loading core drivers on both sides, OS are requesting recourses using communication interface with PCI registered devices (mapping memory, interrupt line allocation, driver registering in OS, etc.) and proceeds with USB cores initialization. Afterwards USB host detects device connection, notifies upper layers of Linux USB stack and initiates USB bus reset to begin enumeration. After successfully passing enumeration, user level USB testing application can be executed in this environment to validate USB core RTL/SW combination.

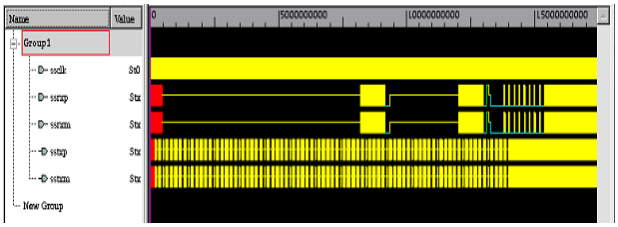

Fig.5. Signal diagrams from RTL simulation on USB core.

V. User level USB testing application

Implemented test tool consists of three modules – test application, class driver and gadget driver. The last one is loaded on the device side of the USB environment and the first two are executed on the host side. The architecture of the implemented test environment is shown in Fig. 6.

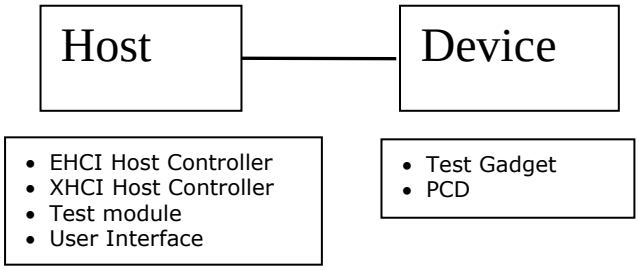

Fig. 6 Testing application architecture.

Test application provides the command line user interface for test cases execution. Test application parses the command line options, analyzes the test parameter values, checks availability of the corresponding pipes and sends appropriate commands to the Linux test class module for further test execution and final results reporting.. The mentioned class driver is the interface between test application and USB Host controller driver. Class driver creates USB transfer blocks, submits USB requests to the USB core with defined communication protocol, and report results to test application after completion. The implemented testing gadget driver submits transfer requests to the underlying Peripheral Controller Driver (PCD) and handles their completion. The gadget driver is responsible for data verification for OUT transfers and for data generation in IN transfer mode. There are also loop back mode tests, when the host writes the data to

the device first, then reads it back and verifies its accuracy.

For communication between host and device, vendor specific protocol is developed (Table 1) in addition to standard USB requests.

TABLE 1 VENDOR SPECIFIC PROTOCOLS

Request Mnemonic	Description/Purpose
DEV_CAP	The device will return the device capabilities.
CONF_EP_IF	Allows the host to change the parameters of endpoints and configure an interface.
TEST_SETUP	The host sets the test parameters.
TEST_STAT	The device will return the Test Case status.
CONTROL_WRITE	Writes data to the control pipe.
CONTROL_READ	Reads data from the control pipe.
SET_TR	Provides to device for setting up a transfer for an Endpoint.
RET_RES	The device will return the results of the most recent transfer.
DEV_RESET	Resets the Device or Interface to a known state.

User interface module is a command line interface. Each test case (or group of test cases) needs to be configured individually with flexible input arguments through user interface module. Each configuration parameters can be divided into the following groups: endpoints parameters which represent endpoint descriptors main fields and endpoint's pair number (interface, number, direction, type, and max packet size), test parameters (mode, transfer size and iteration count).

A special algorithm is developed to handle errors. The transfer buffers are filled with data using implemented algorithm. This algorithm is common for the test gadget and the host test module. Depending on the direction of the test, the test module or test gadget will fill the transfer buffer. The other part will verify the received data.

The USB test gadget provides functionality for implementing the Communication protocol. The test gadget handles all vendor specific requests, controlling endpoints and interface.

VI. STATISTICS AND TEST RESULTS

Co-simulation environment requires significant resources from host Linux machine. The analysis shows that major CPU load is caused by *simv* executable running. Simulation profiling shows that about 50% 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 – time 7.2 min (performing once during setup)
- QEMU Host and Device booting – 2.5 min
- Driver loading done – time 1.3 min
- Core Initialization by driver done – time 1.2 min
- Simulating 1mSec – time 3.6 sec
- CPU Utilization Profile [%]: total time 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 shown in statistics analysis, it takes about 3–4s of real time to simulate 1 ms of hardware 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 performed with 46-60MB/s speed.

After driver loading phase and USB cores' initialization by SW on emulated host and device systems, various tests are performed using the suggested test utility – control, bulk, isochronous and interrupt traffic generation in both IN and OUT directions. Also, test parameters like maximum packet size, interval for periodic endpoints, burst for USB 3.0 transfers, stream count, etc. were modified during tests. Obtained results are captured in Table 2.

TABLE 2. LINUX USBTEST UTILITY TEST RESULTS IN CO-SIMULATION ENVIRONMENT

Test Description	Max packet size	Max burst size	Tr. size	Interval	Duration	Result
Bulk in	1024	0	0.1M	-	0.328s	Pass
Bulk in	1024	10	0.1M	-	0.18s	Pass
Bulk out	1024	0	0.1M	-	0.331s	Pass
Bulk out	1024	10	0.1M	-	0.19s	Pass
Control in	512	-	1M	-	2.083s	Pass
Control out	512	-	1M	-	2.063s	Pass
Isochronous in	1024	0	10M	1	1.5s	Pass
Isochronous out	1024	0	16K	14	19.09s	Pass

VII. CONCLUSIONS

In this paper a new approach to USB controller RTL and driver co-simulation and verification method is proposed. The suggested method is based on virtualization techniques (QEMU emulator) using USB SS core. The method is based on the idea of using a real USB controller RTL model, instead of TLM or other abstract models. The designed environment allows identifying/debugging USB HW design easily, debugging device driver before real tests on FPGA or SoC,

reproducing and debugging issues emerging in the real circumstances using signal diagrams obtained from simulation, beginning device driver development and testing when hardware is not available yet. A user level USB testing application is also developed, which allows running custom tests with various test parameters using different speeds. The implemented application has the ability to ensure four types of transfer running in virtual environment. The method is flexible and does not depend on the USB controller or any other USB IP's RTL, and can be easily integrated into the environment after minor changes in the 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, pp. 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, pp. 966-970.

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

[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, pp. 220-226.

[7] Xiaoping, B.; Shenlei, The Study on System Verification of USB2.0 Interface Protocol Control Chip Hardware Design // ICEE '07. International Conference on Electrical Engineering, 2007, pp. 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, pp. 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, pp. 343-346..

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

[11] SuperSpeed USB Compliance Testing http://www.usb.org/developers/ssusb/testing

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

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