

Topic: ESL, Transaction level modelling for IP based SoC design

**Transaction Level Model of
the USB *On-The-Go* controller IP core**

(Extended abstract)

**Filip Rak, Marek Podeszwa
Evatronix S.A.
Bielsko-Biala, Poland**

**Wojciech Sakowski
Institute of Electronics, Silesian University of Technology
Gliwice, Poland**

Keywords

transaction level modeling, USB *On-The-Go*, design reuse, electronic virtual components, IP cores, system-on-chip, hardware dependent software

Abstract

The paper describes a transaction level model of the serial bus controller compliant to USB *On-The-Go* specification [1]. The model has been developed as an abstraction of an existing IP core, written in VHDL. The possible use in the development or testing of a software driver was addressed too.

Purpose of the work

Transaction Level Modeling paradigm (addressed in more detail in the next section) creates a solid foundation for an effective concurrent design of software / hardware components of System-On-Chips. On the other hand wide IP reuse has become a common practice in System-On-Chip design and a source of substantial productivity gains.

Taking an advantage of both these technologies (*Transaction Level Modeling* and *IP reuse*) calls for availability of transaction level models as a standard element of virtual components (IP cores) deliverables.

At the same time for quite a few classes of electronic virtual components (if not all of them) supporting software is perceived by the users as important value adding component of the total solution offered by a provider of these cores. Such software may range from simple hardware

abstraction layers that hide hardware details to the application programmer to multiplayer software stacks supporting complex protocols.

During development of the software supporting IP core such as the USB OTG controller availability of the transaction level model of this controller provides software developer with a virtual testing environment that enables effective debugging even before the hardware is available.

Originally the software stack supporting the USB OTG controller was developed before the work on the transaction level model was available. Experiences gained during the development of this software were a valuable source of requirements for debug support features of the TLM model.

Development effort related to this TLM is carried out as one of subtasks of the SPRINT project, funded by EU under 6 Framework Program. TLM modeling style and solutions presented in this abstract will be updated in the final version of the paper according to the recent suggestions of such members of this project consortium as Philips, STMicroelectronics and Infineon.

Transaction Level Modeling (TLM)

TLM [2,3] is gaining interest because of its flexibility in handling different abstraction levels and separating on chip communication issues from functionality of the submodules (whether they are created in hardware or in software). Supported with a new breed of hardware description languages (like SystemC [4,5] and to some extent SystemVerilog) it has becoming an important technique that enables shifting the design activities

above the RTL abstraction level.

Thanks to the proper abstraction view of the underlying hardware, simulation times of TLM models may be orders of magnitude shorter than simulation of RTL models describing equivalent functionality, written in “classic” hardware languages like Verilog or VHDL. This makes feasible to test interaction of the hardware modeled with TLM and software that is meant to control this hardware.

As SystemC is simply a C++ [6] class library it is possible to link the hardware model described in SystemC with arbitrary C++ functions.

Elements of SystemC that support TLM paradigm are *channels* and their *interfaces* as well as *ports*. Channels represent resources responsible for communication between system modules. They enable to access them by direct call of methods provided by interfaces. When this approach is used simulation in so-called native mode is possible.

TLM models may support one of the two major abstractions: PV (programmer’s view) and PVT (programmer’s view with timing). The former focus on functionality features relevant to the programmer such as accessible registers and data and control interactions between software running on embedded processor and the modeled hardware, abstracting details of hardware implementation. The latter annotates this functionality with timing properties useful in system (or module) performance analysis.

Main features of the model

The model described in this paper (named *USBHS-OTG-TLM*) is a transaction-level model of Evatronix synthesizable USBHS-OTG-MPD serial bus interface controller core. This core complies to USB2.0 with OTG Supplement to USB 2.0 standard. Model was developed as transactional PV+T (Programmer’s View with Timing) model using C++ language and SystemC library. Its main goal is to help in embedded software design and verification process, and in system architecture / performance analysis process.

USBHS-OTG-MPD features included in USBHS-OTG-TLM

USBHS-OTG-TLM functionality includes subset of USBHS-OTG-MPD core features set, enriched by additional debug and testing abilities.

- Supports Host Negotiation Protocol and Session Request Protocol
- Supports Low-Speed, Full-Speed, or High-Speed peripheral devices and USB Hubs in Host mode
- Supports Full-Speed and High-Speed data transfer in Peripheral mode

- Supports 16 IN and 16 OUT fully programmable endpoints
- Supported transmission modes: bulk/control, interrupt and isochronous
- Suspend and resume power managements functions
- Remote Wakeup function
- Interrupts
- USB protocol-aware multichannel DMA

Programmer’s View USB HS-OTG-TLM features

Since TLM PV+T model should meet particular requirements, which are supported by USBHS-OTG-TLM :

- Register-bit accuracy to achieve seamless software connection. USBHS-OTG-TLM
- implements full USBHS-OTG-MPD register set and provides set of convenient user methods to access them.
- Communication and functionality separation to ease reusing and further refinements.
- USBHS-OTG-TLM can use blocking communication at model boundary to meet data consistency requirements, while using non-blocking communication between
- submodules to gain high simulation speed.
- Microarchitectural details decreased to minimal for increase simulation speed – USBHS-OTG-TLM is developed as algorithm model. SystemC constructions are used to allow concurrency and communication (between distinct devices and within single device between submodules).

Additionally, as OTG functionality is strictly related to time behavior on USB bus, there is a separated module, which generates all necessary events for OTG algorithm.

This sub-module is used only when environment doesn't support own functions for event generation needed by OTG algorithm. When `usb_timers` is not used then USBHS-OTG-TLM is fully PV model, but it loses some functionality of USBHS-OTG-MPD that is part of timeout detection.

Model architecture

Division into submodules

USBHS-OTG-TLM is divided into two submodules to differentiate PV algorithms from time relations on USB bus.

Architecture overview of the model is shown in the Figure 1 below.

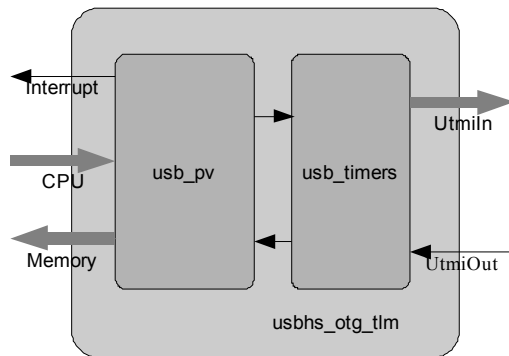


Figure 1. Top level view of USBHS-OTG-TLM

usb_pv consists:

- SFR registers and convince functions for communication with HOST device
- Forming and decomposing packets
- USB, DMA, Wakeup interrupt control functions
- OTG algorithm.
- Host Transaction Scheduler
- Device and Host transfer controllers
- CRC checking/generation
- Basic testing abilities
- Wakeup detection
- Suspend mode

usb_timers consists

- Upstream and downstream timeout events generation and USB bus reset generation/detection.
- SOF events generator
- OTG timeout events generation

Communication between modules/submodules

Communication between modules/submodules is done by using `sc_port` to `sc_export` binding. Each submodule has corresponding interface with methods that are available to others and this interface is published using `sc_export` port. Communication inside of each sub-module is done through events.

Interaction with environment

Model uses non-blocking communication to exchange data with model and ULPI / UTMI part.

Control information are transferred via functions through port with `UtmiOut` interface. These functions are used to control indirectly USB bus.

Packets are transferred with use of the following two functions.

```
status Send (const TPackage &tPackage)
```

This function transfers structure of one packet from controller to USB bus.

```
status Receive(TPackage &tPackage)
```

This function transfers one packet in direction from USB bus to model.

Status of USB bus are transferred via port with `UtmiIn` interface, example functions that are used in controlling OTG algorithm are shown below.

```
status SetIddig (int iValue)
```

Function sets `Iddig` signal to choose controller between Host or Peripheral

```
status SetbValid (int iValue)
```

Function informs that USB Host starts session and model goes to Peripheral.

The USBHS-OTG-TLM provides three interrupt request controllers, one for USB and OTG algorithms, one for DMA and one for wakeup. These interrupts are provided to CPU as port with a set of convenience functions.

CPU can use to communication with USBHS-OTG-TLM AHB slave interface. Another interface is AHB Master interface to system RAM for data transfers. This interface is driven by internal very flexible and useful DMA controller. The DMA controller can work in several modes:

- Ringpointer – In this mode DMA treats reserved memory area in the system RAM as a circular buffer.
- Incremental – burst transmission with incremental address (typical AHB transmission)

Once initialized endpoint, then controller automatically service it until all bytes are transferred.

On both AHB interfaces have basic functions for communication are:

```
status read (unsigned long ugAddress,
            unsigned long *pugData,
            unsigned long *pControl)
```

```
status write(unsigned long ugAddress,
            unsigned long *pugData,
            unsigned long *pControl)
```

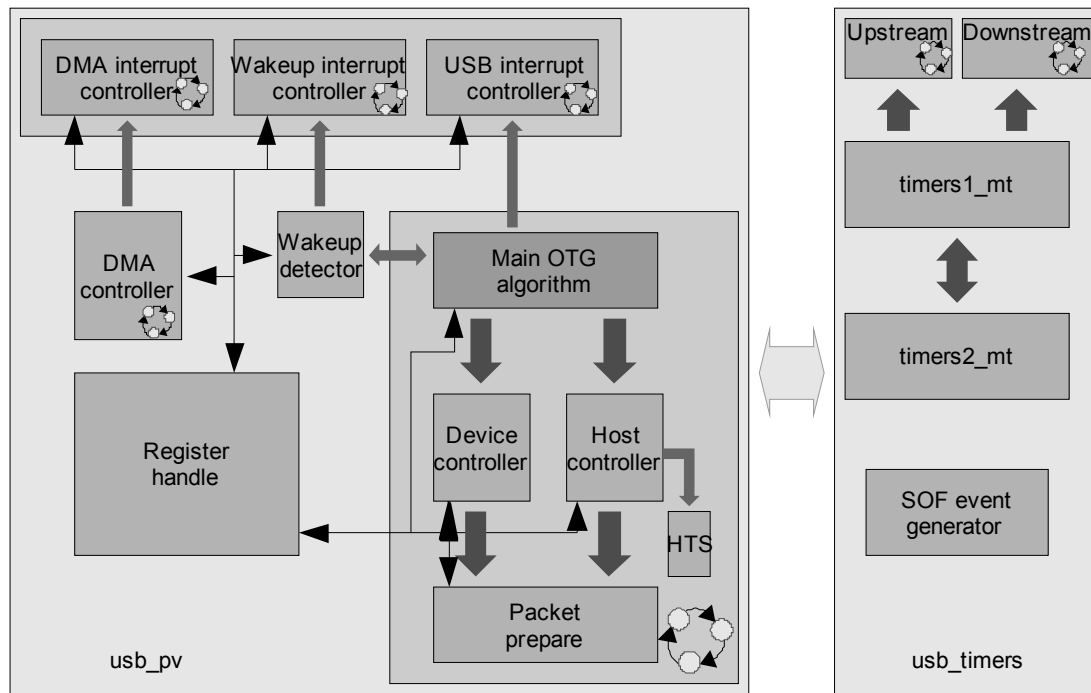


Figure 2. USBHS-OTG-TLM submodule architecture

Testing environment for the model

In order to ease testing and verification process appropriate environment was developed – it consists of stimulus module that contains:

- Parser with Bison grammar analyzer and accompanying Flex lexer for parsing test files with commands for CPU, UTMI, software stack, memory and complex test generator
- CPU submodule which decodes CPU commands and applies proper stimuli to the USBHS-OTG-TLM model
- UTMI submodule, whose role is similar to CPU submodule – that is, UTMI part receives
- commands from parser, decodes them and passes to model
- Software stack, which acts as ANSI-C-compliant implementation of higher protocol layers to provide easy API for embedded software developers
- Complex test generator, which is responsible for verification process automation by generating
- complex stimuli sequence from given constraints
- Checker component for monitoring and checking data flow on model-environment boundary against protocol correctness

CUSB_OTG_TLM model at PV abstraction level was designed to ease embedded software development and verification process. Along with USBHS-OTG-MPD software stack was being developed and from the beginning software engineers took advantage from hardware-software cosimulation.

Testing environment with TLM model is shown in the Fig. 3. The main environment submodule is parser, which task is to take input script (written in an ordinary text file), parse and decompose into sequence of commands, then distribute them among CPU, UTMI, software stack and test generator.

Commands are stored in array and are processed in sequence. Some of them (for example the ones from test generator) are generated dynamically during simulation, without being stored. CPU and UTMI submodules have similar tasks – they take commands from parser, change it into stimuli sequence that is applied to USBHS-OTG-TLM model. They also provide basic verification functionality by checking conditions given in script.

Software stack (presented in more detail in the following section) is a pure-software implementation of higher protocol layers. It is developed in ANSI-C and it is integrated with environment using native cosimulation, which main advantage is very high simulation speed – much higher than possible to achieve when using ISS; that allows verification engineers to develop more complex and longer tests that in case of RTL model. Not less important

is bus scenario generation flexibility – there are no limits on events order declared in bus activity script, so there is possibility to develop non-standard sequence in order to check software behavior in incorrect situations.

To avoid problems with errors in testing scripts, a test generator module was included; it's goal is to generate complex test scenarios with randomly generated data, taking only basic configuration parameters from script. Such approach minimizes

risk of errors in tests, that are common for such a repetitive event sequences.

With `usb_timers` sub-module and wrappers at interfaces tests can be very precise designed with time accuracy, and with not much lost in simulation speed. In the future simulation environment in connection with an ISS will be possible (not only functional, but also its interaction with its destination CPU).

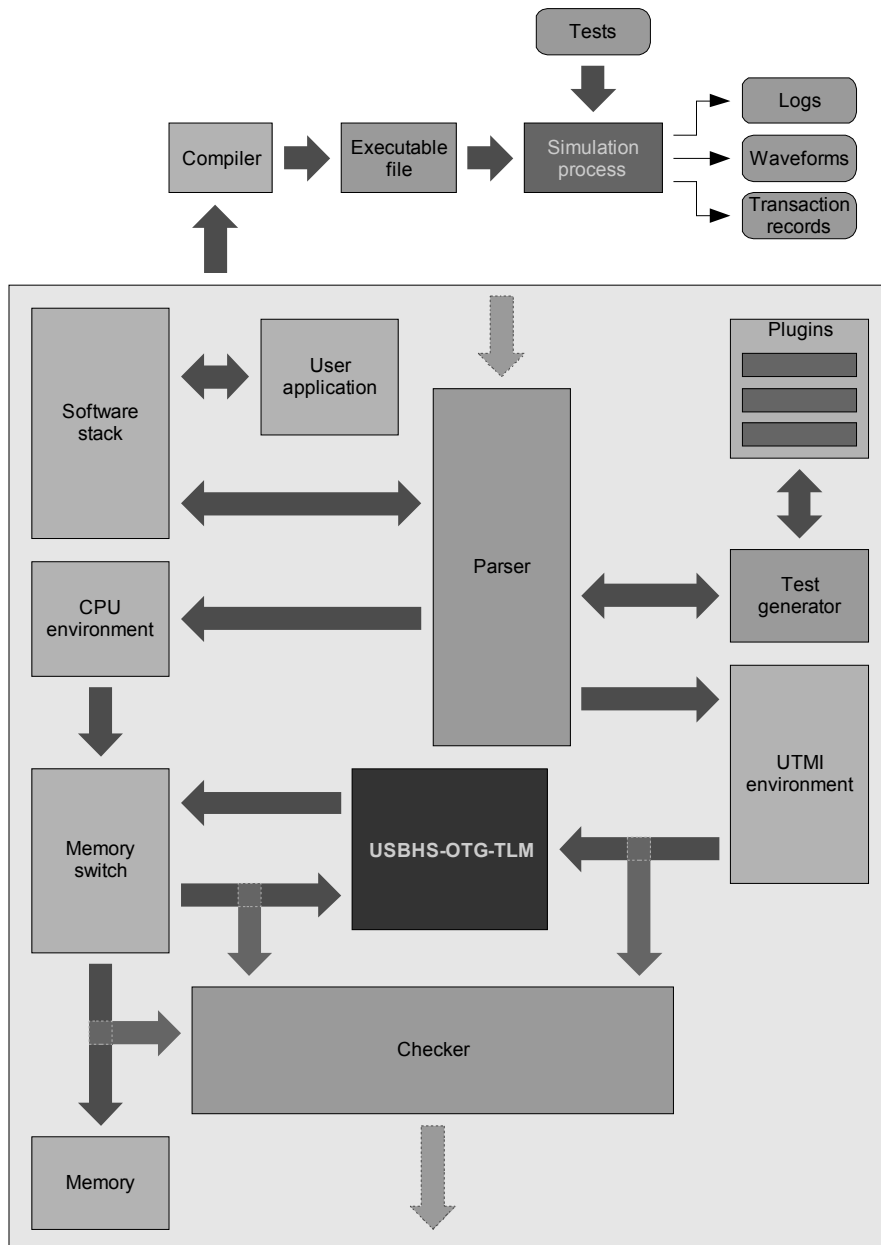


Figure 3. Simulation environment for software development

Conclusions

Transaction model of USBHS-OTG-TLM controller makes initial testing of the USB related software very convenient thanks to model debugging features (in particular: possibility of checking the state of each submodule).

Achievable simulation speed of the model makes such approach a viable alternative to testing the software with the hardware prototype. This enables parallel development of software and hardware.

Literature

1. *Universal Serial Bus Revision 2.0 specification with 1.0a revision to USB On-The-Go Supplement*, 2003
2. *OSCI Standard for SystemC TLM*, available at <http://www.systemc.org>
3. Frank Ghenassia (Editor), *Transaction-Level Modeling with SystemC : TLM Concepts and Applications for Embedded Systems*, Springer, 2005
4. *SystemC Language Reference Manual*, ver, 2.1 © Open SystemC International, May 2005
5. David C. Black, Jack Donovan, *SystemC: From the Ground Up*, Kluwer Academic Publishers, 2004
6. Bjarne Stroustrup, *The C++ Programming Language*, Addison Wesley, 2000