Implementation of a Functional Verification System using SystemC
Myoung-Keun You, Young-Jin Oh, Gi-Yong Song
School of Electrical and Computer Engineering
Chungbuk National University, Cheongju Chungbuk, 361-763, Korea
E-mail : mkyou77@chungbuk.ac.kr, goodmen913@chungbuk.ac.kr, gysong@chungbuk.ac.kr

Abstract: The implementation of a functional verification system using SystemC, system-level design language, is presented in this paper. SystemC is used in system-level design methodology because of the capability of system architectural model description and hardware/software design. The implemented verification system, which consists of various SystemC modules, in this paper can explore design space using SystemC and verify functional correction of progressive refined module in RTL HDL. The implemented system can explore design space using SystemC and verify functional correction of progressive refined module in RTL HDL. The functional verification is performed on a simple device-under-test, the transposed FIR filter. Connections between SystemC simulation kernel and HDL simulator are achieved through user-defined system function of HDL simulator and communication channel.

1. Introduction

As the design of a system gets larger and more complex, the exploration of a design on a high abstraction level becomes more important than ever. The system level design language such as SystemC which models the system on a high abstraction level is drawing attention. SystemC is a system modeling language which consists of a simulation kernel and C++ class library for hardware modeling. SystemC provides co-design environment for hardware units and software modules and enables the design on a high abstraction level to proceed to a synthesizable RT-level design through a progressive refinement [1-3].

In the chip design flow, verification of a newly introduced hardware module is of a paramount importance in the design phase. Roughly 70 to 80 percent of the design cycle is spent in functional verification [4-5]. Functional verification of hardware is recently performed through the BFM(bus functional model) of target module, and SystemVerilog which is combined hardware description language and hardware verification language takes attentions. However these functional verification methods are proper for only DUT(device-under-test), not for system-level design.

This paper describes an implementation of functional verification system of hardware in system-level using SystemC simulation kernel and HDL simulator. The implemented system can explore design space using SystemC and verify functional correction of progressive refined module in RTL HDL. The functional verification is performed on a simple DUT, the transposed FIR filter. The FIR filter which processes convolution sum is a typical operation being involved in various applications regarding DSP.

This paper is organized as follows; section 2 briefly introduces SystemC, VPI (Verilog procedural interface), and transposed FIR filter, and section 3 describes implementation of function verification system. Section 4 is for simulation of functional verification on the transposed FIR filter, and section 5 discusses the conclusion.

2. Preliminary

2.1 SystemC

System designs consist of hardware parts and software parts. Traditionally the hardware parts are described in HDLs such as Verilog or VHDL while the software parts are coded in a programming language such as C or C++. Mixed-language system level design flows do not allow a rapid exploration of the design space [2-3]. In the beginning phase of the system design, the hardware and software partitioning of target system is not made a decision clearly. If there is any change on the partitioning of the system later, expensive and time consuming re-design will become necessary.

The system-level languages allow a flexible partitioning in the design of the hardware and software. Moreover, many properties depend on the combination of hardware and software and cannot be verified on either part alone. SystemC [1-3] is a language for system level design at multiple abstraction levels. Being built on standard C/C++ language, the SystemC describes functionality and communication at various levels of abstraction since it supports concepts of time, hardware data type, concurrency, and hierarchy. Abstraction levels of system models are shown in Fig. 1.
A SystemC model described at higher level can successively be refined down to synthesizable SystemC RTL model. SystemC supports hardware-software co-design and a description of the architecture of complex systems consisting of both hardware and software.

2. 2 Verilog procedural interface, VPI

The VPI provides a mechanism for defining system tasks and functions that communicate with the simulator through a C procedural interface. Today’s hardware design paradigms typically involve creating models of an intended design before the hardware is actually built. Simulations are then run on these models to verify that the design behavior is as intended [7-8].

The Verilog PLI is a simulation interface. It provides run-time access to the simulation data structure, and allows user-supplied C functions a way to access information within the simulation data structure. This user-supplied C function can both read and modify certain aspects of the data structure. The Verilog PLI does not work with Verilog source code. It only works with a simulation data structure.

The Verilog PLI allows programmers to extend the Verilog language though the creation of system tasks and system functions. The names of these user-defined system tasks and functions must begin with a dollar sign ($). A task in Verilog is analogous to a subroutine. When a task is called, the simulator’s instruction flow branches to a subroutine. Upon completion of the task, the instruction flow returns back to the instruction following the task call.

2.3 Transposed FIR Filter

Digital filters [9] are typically used to modify or alter the attributes of a signal in the time or frequency domain. An FIR with constant coefficients is a linear time-invariant (LTI) digital filter. The output sequence, y(n), of an FIR of order or length L, to an input sequence, x(n), is given by the finite version of convolution sum as follows:

\[ y[n] = x[n] * f[n] = \sum_{k=0}^{L-1} x[k] f[n-k] \]

where \( f[0] \neq 0 \) through \( f[L-1] \neq 0 \) are the filter’s L coefficients.

The FIR filter which processes convolution sum is a typical operation being involved in various applications regarding DSP. When implementing FIR filter, the transposed structure is preferred as the FIR filter with transposed structure does not need either an extra shift register for input sequence or an extra pipeline stage for adder of the products. The block diagram of the FIR filter in transposed structure is shown in Fig. 2.

3. Implementation of a Functional Verification System using SystemC

3.1 Structure of a functional verification system

The described system in this section uses interaction of between SystemC modules and Verilog HDL modules by using communication channel which will be mentioned in following section. SystemC modules consisting of the implemented system provide infrastructure of functional verification, and HDL simulator performs simulation of progressive refined module in HDL. The main objects and connections of the functional verification system are simply shown in Fig. 3. Data transmissions between SystemC modules or between processes are performed using channels provided by SystemC, and those between SystemC module and HDL module are through communication channel which has an instance of vreg class in Teal library [4-5].
The operations performed on each module are simply explained as follows:

- **SystemC::driver module**

  This module randomly generates test_vector which will be sent to HDL simulator through communication channel which makes SystemC code accessible to wires and registers of DUT. Also this module initializes verification process by sending reset signal to testbench.v.

- **HDL testbench**

  This module generates clock signal and sends it to both driver and monitor module in order to synchronize. And this sends results of the DUT operation on the received test_vector to monitor module through communication channel.

- **SystemC::monitor**

  This module receives the output from HDL simulator, and compares it to the output of golden file described in SystemC. The verification result on the operation of DUT is reported to the console monitor of HDL simulator.

### 3.2 SystemC function and Verilog system function

The VPI is used to integrate SystemC models with Verilog simulations. The testbench function called sc_main() of SystemC model is registered to the system functions of HDL simulator using VPI. The registered system function is called in an initial block at the top-level testbench, and SystemC simulation kernel starts in the result.

As shown in Fig. 4, when Verilog parser meets a statement which begins with `S`, Verilog simulator finds system function’s name in the external symbol table and calls the corresponding function. The tfname member of the s_vpi_systf_data structure contains the name called from Verilog, and the calltf member contains the function pointer of user-defined system function. Since vpi_register_systf function calls with the assigned s_vpi_systf_data structure type, the corresponding system function can be used in Verilog.

#### 3.3 Communication channel

Architectural exploration and early system integration require the ability to connect modules of different abstraction levels into a system model. Architectural exploration involves refinement and decomposition of modules, and repartitioning of the model. Refinement of functionality is best accomplished in parallel with a refinement of communication. SystemC provides constructs such as interfaces, ports, and channels.

Teal is an open-source verification utility and connection library that fits at the lowest abstraction level. The vreg class is probably the most commonly used class in Teal, as it connects C++ code to the HDL. This class provides mechanisms to connect wires and registers in the HDL simulation. The at() function in Teal allows a thread to pause until any of the HDL signals has changed.

Communication channel described in this section is a custom primitive channel which provides connection between SystemC module and HDL module. This channel contains an instance of vreg class in Teal, so it is connected to driver module and monitor module as shown in Fig. 3.

A channel must have an interface or interfaces to implement. The interface methods of communication channel are read(), write(), default_event(), value_changed_event() and etc. To allow and simplify use in static sensitivity lists, we’ll specify a default_event. Read() and write() method are finally implemented by reading and writing internal instance of vreg class. Default_event() and value_changed_event() methods play a role of mapping wait() in SystemC to at() in Teal. These methods wait until an HDL signal connected to internal instance of vreg has changed and then return the default_event to be recognized by sensitive.

### 4. Simulation and Result

To simplify the result observation, we set coefficients of 4-stage transposed FIR filter as 0x1, 0x2, 0x4, and 0x8, and specify the range constraint of test vector between 0x0 and 0x5. The result of a functional verification is shown in Fig. 5.
when driver module generates test vector as $0x04$, $0x1$, $0x3$, and $0x4$. Messages are reported by monitor module after comparing output from HDL simulator to the output from C-simulator of the transposed FIR filter.

The exploration of design space in system-level is possible as the infrastructure of the described verification system is provided by SystemC modules, and as SystemC’s great strength is the ability to provide higher levels of abstraction for all components of a design.

5. Conclusions

This paper describes an implementation of a functional verification system using SystemC. Hardware modules are designed with Verilog HDL in order to make SystemC code accessible to wires and registers of DUT through VPI. Connections between SystemC simulation kernel and HDL simulator are achieved through user-defined system function of HDL simulator.

References