

An IEEE 1500 Compatible Wrapper Architecture for Testing Cores at Transaction Level

Fatemeh Refan¹, Paolo Prinetto², Zainalabedin Navabi¹

¹*CAD Research Group, ECE Department, University of Tehran, Tehran 14399, Iran
{refan, navabi}@cad.ece.ut.ac.ir*

²*Dipartimento di Automatica e Informatica, Politecnico di Torino, I-10129 Torino, Italy
paolo.prinetto@polito.it*

Abstract

With the evolution of Electronic System Level (ESL) design methodologies, Transaction Level Modeling (TLM) is regarded as the next step in the direction of system level design. This requires definition of appropriate test strategies at this level including wrapper, scheduling and Test Access Mechanism (TAM) design. In this paper we propose a wrapper for core testing at transaction level, compatible with IEEE 1500 standard architecture. The proposed wrapper is designed to support communication with tlm_fifo as the basic primitive channel of SystemC TLM core. The wrapper is defined as a layer on top of standard IEEE 1500 architecture, including a controller to packetize and de-packetize the test stimuli and responses. Controller design, and architecture configuration for serial and parallel core test instructions are presented. Using the same strategy other instructions can be implemented.

1. Introduction

With increase in the complexity of systems, scalable system test methods are needed. Furthermore as system modeling level trends to transactional level, the capabilities of test strategies should be improved to handle new problems caused by specifications of modern approaches. On the other hand, higher levels of abstraction may offer some facilities to ease the process of system testing.

Most researches in the field of system testing have targeted System-on-Chip (SoC) testing[1]. The currently used method is modular test, where test specification for each core is provided by IP manufacturers, and system test strategy should be defined based on this information. SoC modular test includes three main parts: Wrapper design, TAM design, and test scheduling [1]. Wrapper design became standard by releasing the IEEE 1500 standard for embedded core based inte-

grated circuits [2]. Current researches define their test strategies compatible with this standard.

Nearly all papers on this topic just focus on low level physical specifications of SoCs. However as design complexity changes from wiring gates to inter-connecting complex processing elements at TLM abstraction level, new DFT approaches compatible with this level of design are required. Special strategies should be designed to test channels [3] (TLM interconnections) and modules (TLM cores).

In this paper we tackle the problem of designing a wrapper at transaction level as the first step in defining TLM-based test strategy. The proposed wrapper is designed as a two layer structure: including a transactor to communicate with TLM channels, and a low level interface to IEEE 1500 wrapper architecture. Four mandatory and optional IEEE 1500 standard instructions necessary for core testing are provided in this paper to elaborate wrapper design methodology.

The paper is organized as follows: Section 2 presents an overview of SystemC TLM, and IEEE 1500 standard as background information to understand the rest of paper. In Section 3 and 4 the problem of testing at TLM is defined and the proposed TLM wrapper is presented respectively. We evaluate cost of proposed wrapper architecture in terms of hardware complexity and execution time in Section 5. Finally Section 6 concludes the paper.

2. Background

In this section the information needed to understand the rest of paper is presented. First we introduce SystemC TLM as our modeling level and language. Then a brief introduction to IEEE 1500 standard for embedded core based integrated systems is presented.

2.1. SystemC TLM

SystemC TLM standard [4] released by OSCI has now become the most popular approach for modeling at the transaction level. SystemC as the class library for modeling at this level has a close correspondence with lower RT level descriptions, and its high level interface with C++. It is an object-oriented language based on C++, where main hardware-oriented parameters like Time, Concurrency, and Hardware data types are implemented in SystemC [5].

The main classes and methods proposed by SystemC TLM are for modeling bidirectional, unidirectional, blocking and non-blocking communications. Interface classes as the main parts of SystemC TLM, are implemented by TLM primitive channels. Core TLM interfaces presented in this standard include unidirectional blocking *tlm_blocking_get_peek_if*, *tlm_blocking_put_if* and nonblocking *tlm_nonblocking_get_peek_if*, *tlm_nonblocking_put_if*, and bidirectional blocking *tlm_transport_if* interfaces.

There are three TLM primitive channels: *tlm_fifo*, *tlm_req_rsp_channel* based on *tlm_fifo*, and *tlm_transport_channel*. *Tlm_fifo* class implements all the unidirectional interfaces mentioned above, while bidirectional interfaces are implemented in *tlm_transport_channel*. *Tlm_fifo* is defined on the basis of SystemC *sc_fifo* with the additional capability of having a zero or infinite size [6]. Among described access methods in *tlm_fifo* class, we just focus on get and put methods for reading data from and writing data to channel.

2.2. IEEE 1500 standard

IEEE 1500 standard defines a mechanism for the test of cores within a system on chip (SoC), including a wrapper hardware architecture. It also uses core test language (CTL) to facilitate communication between core designers and integrators.

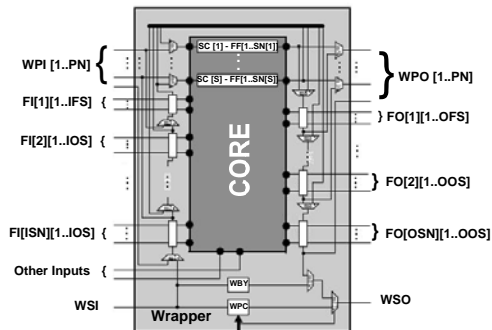


Figure 1. IEEE 1500 wrapper architecture [2]

Figure 1 illustrates IEEE 1500 standard components and general wrapper architecture; where WPI, WPO, FI, FO, WSI, and WSO represent wrapper parallel input port, wrapper parallel output port, core functional inputs, core functional outputs, wrapper serial input, and wrapper serial output respectively. WPI and WPO include PN 1-bit ports, and FIN, and FON are equal to the bit width of FI, and FO respectively. The input (output) side WBR is segmented into ISN (OSN) segments, where first segment is IFS (OFS) bits wide and other segments has bit width of IOS (OOS). Therefore FIN, and FON are equal to $IFS + IOS \times (ISN - 1)$ and $OFS + OOS \times (OSN - 1)$ respectively.

3. Core testing at the transaction level

A system consists of a number of cores and a set of connections between them. Based on the type of interconnections, systems can be divided into three main groups: point-to-point systems, bus-based SoCs, and Network based systems (NoCs). Bus-based and network based systems support more general structures. Point-to-point systems offer better performance but require longer design times.

However as design methodology evolves toward higher levels of abstraction like TLM, communication becomes more and more important. At this level the simplest communication channel is a FIFO, the other two primitive channels being defined based on it. Using channel based communication necessitates use of a transactor to enable communication between high level channel functionality and low-level test requirements. On the other hand, the more complex behavior of channels can be used to enhance test strategy, without introducing additional hardware cost.

To define test strategy, its three parts (Wrapper, TAM, and Scheduling) should be determined. Since scheduling and TAM design are defined based on the capabilities of Wrapper, in this paper we just tackle wrapper design. We suppose that input test data stimuli and responses are transferred by channels (in this paper only *tlm_fifo*). Stimuli patterns are encapsulated in a packet, including values of functional inputs, and internal scan chains. Packets are broken into smaller parts to fit in *tlm_fifo*. The sequence of placement of test bits in each packet is determined based on the type of instruction. The same packets are used for delivering test responses.

4. Wrapper design

In this section we propose the principal idea of designing a TLM based core wrapper. It is designed as a transactor on top of IEEE 1500 standard wrapper. We

reuse functional connections as TAM, and consider *tlm_fifo* as the only communication channel. The most important issues about the proposed wrapper will be discussed in this section: The capability of wrapper to communicate with channels, and its configuration for parallel and serial core test instructions.

4.1. Communicating with channels

Channels provide a more intelligent mechanism than simple wires for communication. They have memory, and controller. They can even be considered as a module and isolated using wrappers. However, this strategy is expensive in terms of area, because of the large number of channels in a complicated system. Therefore we consider that each channel has its own self test method and does not need wrappers for testing [4]. This means that when the system is in test mode, channels are still functional. Therefore the wrapper should be capable of communicating with different channels.

tlm_fifo as the only communication channel discussed in this paper, has many functions. But it is enough for wrapper to be able to read from input channel, process the test stimuli data based on the current instruction, and finally write data on the output channel. Therefore the only interface that ports of wrapper channel need to support are *tlm_blocking_get_peek_if*, and *tlm_blocking_put_if*. Figure 2 shows the wrapper schematic.

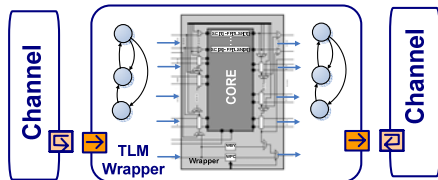


Figure 2. TLM wrapper general architecture

4.2. Testing cores

In order to test a core, a set of test pattern stimuli should be applied to the core under test, and the responses should be got from it. The input stimuli include functional and scan test input bits, which are supplied by input channel. The total number of bits of input stimuli is equal to the functional input bit width, in addition to the total number of scan flip-flops. Therefore, the total number of needed input test packets can be determined by equation 1. An input controller is needed to handle input packets and feed test data into main wrapper architecture, and an output controller is required to do the reverse process and collect responses.

$$TNIE = \left\lceil \frac{\text{Patterns Num} \times \text{Pattern Length}}{\text{Element Length}} \right\rceil$$

Equation 1.

According to IEEE 1500 standard, WS_INTEST is a mandatory core test instruction. We have also designed the optional parallel WP_INTEST_SCAN instruction. In WS_INTEST instruction, all bits of packet should be serialized and shifted into scan flip-flops and core inputs. Elements are taken from *tlm_fifo*, then they are shifted into WBR and internal scan chains from WSI, and responses are shifted out from WSO. After shifting FIN bits, next element is got. After complete load of a pattern, core executes for one clock cycle. During stimuli shift-in, responses will be shifted out automatically, but for last pattern response, an extra OPL (output pattern length) shifts are needed. Figure 3(a) demonstrates architecture configuration extracted from standard. Input and output controllers and appropriate packet structure are shown in Figure 3(a) and Figure 3(b).

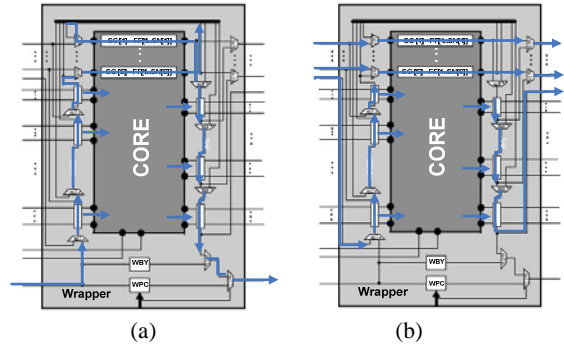


Figure 3. Instruction architecture configuration
(a) WS_INTEST - (b) WP_INTEST_SCAN

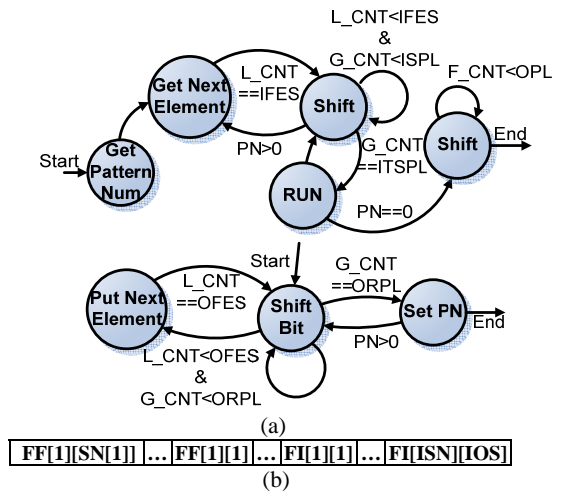


Figure 4. WS_INTEST instruction
(a) above: Input controller, below: Output controller - (b) Packet structure

For WP_INTEST_SCAN instruction, WPP is used to parallelize applying stimuli. We use the architecture extracted from standard, where PN is equal to S plus one: One input for shifting functional input bits into input WBRs, and other inputs to initialize scan FFs. The controller starts by getting two elements (A, B) from input *tlm_fifo*. Then WBR and scan chains are loaded with PN bits from A, and set of A and B registers are shifted to left for PN bits. When contents of B are completely shifted into A, new element is loaded. The same as WS_INTEST, final response should be shifted out manually. The reverse process is done in output controller. According to the architecture of Figure 4(b), controller and packet structure are shown in figures 5(a), and 5(b).

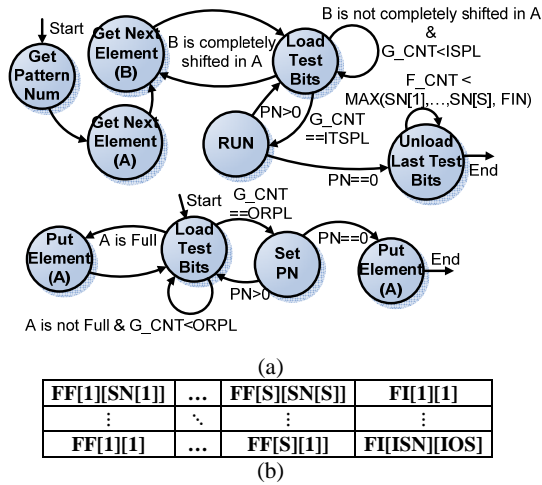


Figure 5. WP_INTEST_SCAN instruction
(a) Above: Input controller, Below: Output controller - (b) Packet structure

If the core under test has no scan chains, we use WS_INTEST_RING and WP_INTEST_FUNC instructions. The serial instruction is the same as WS_INTEST. In WP_INTEST_FUNC instruction, each element gotten from *tlm_fifo* is loaded in parallel into WBR, and output is unloaded in parallel too. In other words, core functional ports are used for testing. Therefore, input controller includes three states for getting element, loading and core execution. Output controller in the same manner has two states. WS_INTEST architecture configuration of WS_INTEST_RING, and WP_INTEST_FUNC are the same as WS_INTEST, and normal mode of core, respectively.

5. Results

To have an estimation of cost of proposed wrapper, we define hardware complexity and time as our cost

factors. We estimate hardware cost by number of controller state machines, and time as the clock cycles needed for loading and unloading wrapper registers and test execution. Table 1 demonstrates cost of different instructions in terms of area and complexity, where NP represents number of patterns. Among scan based instructions, WS_INTEST is simpler but slower, while in functional test instructions the parallel version is both faster and simpler.

Table 1. Cost of wrapper for each instruction

Instruction	Time Cost (clock cycles)	State Num	
		IN	OUT
WS_INTEST	$NP \times (FIN + \sum_{i=1}^S SN[i] + 1) + FON + \sum_{i=1}^S SN[i]$	5	3
WP_INTEST_SCAN	$NP \times [1 + MAX(FIN, SN[1], \dots, SN[S])] + MAX(FON, SN[1], \dots, SN[S])$	6	4
WS_INTEST_RING	$NP(FIN + 1) + FON$	5	3
WP_INTEST_FUNC	$2NP + 1$	4	2

6. Conclusions

In this paper we proposed a TLM wrapper architecture compatible with standard IEEE 1500 architecture. The wrapper is designed to support reusing functional connections (here *tlm_fifo* channel). The proposed wrapper is defined as a transactor layer on top of standard architecture. We demonstrated implementation of four core test instructions with this strategy. Finally instructions are compared in terms of time cost, and design complexity.

The proposed method can be generalized for other instructions, other types of channels, and other TAMs. Furthermore, the estimated costs can be used as input information to evaluate test scheduling algorithms.

7. References

- [1] Q. Xu and N. Nicolici, "Resource-Constrained System-on-a-Chip Test: A Survey," *IEE Proc. Computers and Digital Techniques*, vol. 152, no. 1, pp. 67-81, Jan. 2005.
- [2] IEEE Std 1500, *IEEE Standard for Embedded Core Test—IEEE Std. 1500-2004*. New York: IEEE, 2004.
- [3] H. Alemzadeh, S. D. Carlo, F. Refan, P. Prinetto, Z. Navabi, "Plug & Test at System Level via Testable TLM Primitives," To be appeared in *Proc. of International Test Conference (ITC'08)*
- [4] OSCI SystemC TLM 2.0 Standard, http://www.systemc.org/projects/tlm/document/TLM_2.0_Overview/en/1
- [5] S. Mirkhani, Z. Navabi, *System Level Design Languages, The VLSI Handbook*, Chapter 86, CRC Press, 2nd Edition, Dec. 2006.
- [6] A. Rose, S. Swan, J. Pierce, J.-M. Fernandez, "Transaction Level Modeling in SystemC", *OSCI white-paper*, 2004.