Wrapper Design for the Reuse of Functional Interconnect for Test

Alexandre M. Amory[2] Kees Goossens[1] Erik Jan Marinissen[1] Marcelo Lubaszewski[2] Fernando Moraes[3]

[1] Philips Research Laboratories
IC Design
Prof. Holstlaan 4
5656 AA Eindhoven, The Netherlands
{kees.goossens, erik.jan.marinissen}@philips.com

[2] Federal University of RGS – UFRGS
Instituto de Informática
Av. Bento Gonalves, 9500
Porto Alegre, RS, Brazil
amamory@inf.ufrgs.br, luba@ece.ufrgs.br

[3] Catholic University – PUCRS
Faculdade de Informática
Av. Ipiranga, 6681
Porto Alegre, RS, Brazil
moraes@inf.pucrs.br

Abstract

This paper proposes a wrapper design for interconnects with guaranteed bandwidth and latency services and on-chip protocol. We demonstrate that these interconnects abstract the interconnect details and provide predictability in the data transfer, which are desirable not only for the functional domain but also for the test application. The proposed wrapper model is implemented in VHDL and integrated to the Æthereal NoC. The results show the impact of the amount of bandwidth assigned for the test in the core test time. The wrapper area and core test time are compared with a wrapper design for dedicated TAM.

1 Introduction

The increasing complexity of Systems-on-Chip (SoCs) poses challenges to testing for manufacturing defects [1]. *Modular testing*, i.e. testing individual SoC modules as stand-alone units, effectively addresses most of these challenges. Non-logic modules such as embedded analog and memories, as well as black-box or encrypted third-party cores require modular testing. In addition, modular testing provides an attractive "divide-and-conquer" test development approach and allows for test reuse. Modular testing is enabled by on-chip hardware in the form of module test wrappers and a Test Access Mechanism (TAM) [1]. The design of wrappers and TAMs has a large impact on the required test vector memory of the ATE, the test application time, and other test-related costs of the SoC. Several design and optimization procedures have been proposed [2, 3].

As SoCs grow in complexity, functional interconnects for SoCs have evolved from a single buses, to multiple hierarchical buses, and recently to networks-on-chip (NoC) [4]. Each new generation of functional interconnect has more bandwidth, scalability, modularity, and communication services.

With such increasing number of features, the reuse of this infrastructure as TAM seems to be straightforward. However, four main challenges arise:

(*i*) *test time* - core test time minimization;
(*ii*) *area overhead* - wrapper area overhead minimization;
(*iii*) *compatibility* - backward compatibility with the existing tools and design methods. Changes are not required on test equipment, on functional interconnect, and on the core. Reuse the existing tools are taken into account;
(*iv*) *general* - general wrapper design approach. It should fit to a large number of functional interconnects;

In the sequel of the paper we refer to these challenges as *the goals* that should be met.

A functional interconnect with support for *guaranteed services and on-chip protocol* is required to have a modular design and also for reuse [5]. This paper presents a *general wrapper design to reuse of this functional interconnect as a TAM*. The proposed wrapper does the conversion between the functional protocol and the test protocol. The wrapper is implemented in VHDL and integrated to the Æthereal NoC [6]. The results for test time and area overhead are presented. The novelties presented in this paper are: (*i*) we consider the tester, and the CUT running in test mode, while the functional interconnect runs in normal mode. No modification on them is required; (*ii*) the proposed wrapper, rather than the core which is in test mode, plays the protocol with the functional interconnect, and it converts the test data format required by the CUT to/from the protocol format supported by the functional interconnect; (*iii*) we demonstrate the application and benefits of interconnects with guaranteed bandwidth and latency services for test.

This paper is organized as follows. Section 2 compares and analyses how the previous papers address the *goals*. Section 3 defines how the *goals* are met in our approach, and presents its basic background, motivation, and advantages. Section 4 and 5 show the template for both the core and wrapper, respectively. Section 6 demonstrates an operational wrapper examples integrated to the Æthereal NoC. Section 7 concludes the paper.

2 Prior Work

There are papers proposing the reuse of different functional interconnects, such as, directly connected wires, buses, and NoCs. These papers are compared in this section considering the *goals* presented in the introduction. However, this paper focus on the reuse of NoCs for test because it has the required properties to deal with the future complex SoCs, e.g. scalability, bandwidth, low power, abstraction, among others.

Cota et al. [7] present a tool that reuses *directly connected network of wires*. Transparent mode is implemented in the wrapper, and mismatches between the number of function wires and test wires are solved

implementing parallel-to-serial and serial-to-parallel converters. There are also reuse approaches based on *buses*. Huang et al. [8] reuse a PCI bus with VCI interface. Hwang and Abraham [9] present a reuse model based on a wishbone bus. Harrod [10] present an industrial approach for AMBA bus, which supports functional test. Feige et al. [11] extended the approach to support scan-based test. All the above mentioned bus-based approaches have common drawbacks; they require clock gating, which means modifying the clock tree, to halt the test when there is no data (*goal iii*), buses don't support scan-in and scan-out in parallel and the cores must be tested sequentially since buses support only one transaction per time (*goal i*), and the approaches are specific to one type of bus and interface protocol (*goal iv*). Moreover, like in directly connected wires, buses are not scalable enough for future chips, and may not provide sufficient bandwidth for the functional application.

There are also some recent approaches reusing *NoC* for test. The advantages of NoCs compared to other functional interconnect are: (*i*) they use shorter global wires than buses, which reduces undesirable features in deep-submicron technologies like wiring delay, power dissipation, and signal integrity [4, 12]; (*ii*) they share the global wires for long distance interconnects, increasing their utilization [12]; and (*iii*) they provide scalability to handle more cores. NoCs also support multiple simultaneous transactions, which is important to allow parallel test cores and to allow sending data in and out simultaneously (*goal i*).

Cota et al. [13] propose preemptive test scheduling, where the test of a core can be interrupted if there is no free path between the source to CUT or CUT to sink. A test packet can also take different paths depending on their availability, but the shortest available is selected. The drawbacks are that preemptive test may reduce the parallelism between scan-in and scan-out, and clock gating is required to halt the test when there is no data (*goals i* and *iii*). Liu et al. [14] proposed non-preemptive testing. A single path from source to CUT and from CUT to sink is established in the beginning of the test and the test packets are sent one after the other on this dedicated path. Although this approach preserves the scan-in and scan-out pipeline, it does not guarantee that there will be a new test data in each clock cycle. For example, there are some clock cycles that are used to pack/unpack data, and also some clock cycles to execute the functional protocol. The exact timing depends on the packet format, that may be different in each design (*goal iv*). Then, the clock gating or holdable scan cells must be used (*goal iii*).

Despite all the advantages of NoCs for both the functional and test domains, a NoC is more complex than a dedicated TAM. A common aspect to all the previous approaches is the large amount of NoC implementation details required for the test model (*goal iv*). All kind of functional implementation details (e.g. used arbitration algorithm), temporal details (e.g. time to route a packet), and organizational details (e.g. network topology) are required, in addition to a cycle-accurate scheduling used to determine the available paths between the test source, CUT and test sink. This amount of implementation details requires major efforts to adapt the test model to different NoCs. Thus, *interconnect abstraction is the key challenge for test reuse approaches*.

Although we compare the reuse approaches based on existing functional interconnects, we claim that *the method presented in this paper can also be applied to other or even future interconnects*, as long as the interconnect supports the following features: (*i*) guaranteed bandwidth and latency services to abstract the interconnect implementation details and to provide predictability to the data transfer; (*ii*) on-chip protocol used to standardize the communication between the functional interconnect and the core, whatever the mode.

It is important to realize that these features that our model rely on were defined for the functional domain. Guaranteed services are applied to real-time system since it provides predictability to the data transfer. We propose the reuse of this kind of interconnect for test, and we present a wrapper design for it.

3 System Model

Figure 1 illustrates the overall environment and its main components: the tester, the CUT, the functional interconnect, a converter around the tester interface, and a wrapper around the CUT interface. This figure emphasizes the interaction between these components and the data format expected by each of them.

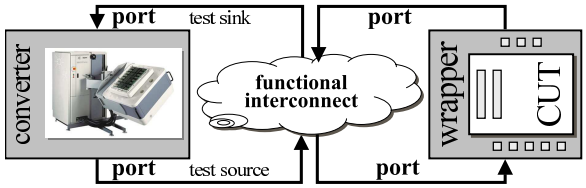

Figure 1: System model for the reuse of functional interconnect for test.

The first component in Figure 1 is the *tester*. Testers are built to assert and evaluate signals in a timing accurate manner. A tester works in a *continuous streaming* mode, which means that once the test starts, it is not interrupted until its end. Considering other type of communication is not realistic (*goal iii*).

The second component in Figure 1 is a *core* in scan-based test mode. The core, like the tester, also requires continuous test data streaming, where the internal scan chains are continuously fed with new test data every clock cycle until the end of its test. On the other hand, it is possible to modify the wrapper design in order to support temporally stalling of the test. This kind of test is called preemptive test. Preemptive test may be useful to support dividing the test of a core in several pieces in order to better fit in the chip-level test scheduling. However, preemptive test requires logic to halt the test while there is no test data. The usual approach is to implement clock gating, but it interferes in the clock tree design. The second approach is to implement hold state to all scan cells of a core. This option increases the area overhead due to the additional multiplexers required for each scan cell to hold its current value. Moreover, it is impossible to modify the scan cell design for hard and encrypted cores. In addition, every time the test of a core is interrupted in a preemptive test, the parallelism between the scan-in and scan-out of test patterns is lost, increasing the core test time. To meet the *goals i* and *ii*, this paper uses non-preemptive test, and preserve the continuous test data streaming nature of the test application.

The third component in Figure 1 is the *functional interconnect*. This figure splits the functional interconnect in the test input and output part to ease the test view of the system, but the interconnect is only one and it is bi-directional. Figure 2 presents a more detailed view of the functional interconnect model, which includes on-chip protocols and guaranteed services. An *on-chip protocol*, e.g. OCP [15], DTL [16], or AXI [17], standardizes the interface between cores and the interconnect. *Guaranteed services* provide high-level functionalities based on requirements of the core, abstracting how the service is implemented. The most relevant services used in the functional domain that are useful for the test domain is *guaranteed bandwidth and latency services* [6, 18, 19]. These services assure that the target core in the communication can receive a certain amount of data in a fixed time interval providing predictability for the data transfers.

The *converter around the tester interface* performs a parallel-to-serial to match the number of functional data terminals from the interconnect with the number of test terminals from the tester. A serial-to-parallel conversion is required for the output part.

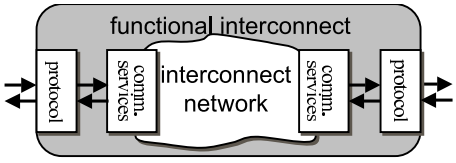

Figure 2: Functional interconnect model.

The proposed functional interconnect model helps to address the functional requirements, and the test goal *iii* and *iv*. A functional interconnect without these characteristics may hamper in some extent the functional requirements. *We claim that the presented interconnect model is not only useful for the functional domain to enable a compositional and modular chip design, but also to the test domain, for reuse of the functional interconnect for test*. Section 5 presents how the adopted functional interconnect model helps to define a simple wrapper design that performs the required conversions, and attends the test *goals*.

4 Core Model

Cores are connected to interconnect using standardized protocols. This paper proposes a new core terminals classification, which takes the protocols into account. When considering reuse of functional interconnect for test, it is important to know the role of each terminal used to connect the core to the interconnect.

Figure 3 illustrates the functionality of a DTL-like port and its protocol. The main principles can be applied to other protocols.

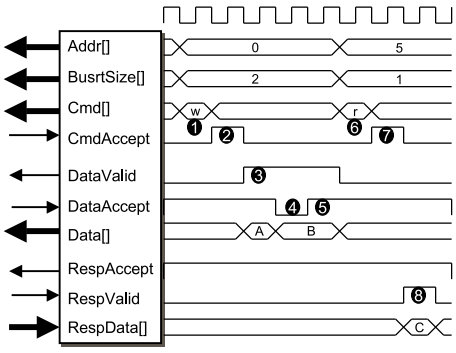

Figure 3: Functionality of a DTL-like port.

Event 1 in Figure 3 represents the request of a write command to send two words, which is accepted in the next clock cycle, during the event 2. The first word is sent during the event 3, when the data valid is high. In event 4 the target does not accept the second word, but it is accepted in event 5. The read command works in a similar way.

Definition 1 formally defines a port for test purpose, classifying the port terminals as control or data.

Definition 1 [Port].
- set DI of *data input terminals* and a set DO of *data output terminals*, such that $DI \cup DO \neq \emptyset$;
- set CI of *control input terminals*;
- set CO of *control output terminals*;
- the *port maximal input data rate* b^{in}_{max}, expressed in bytes/s;
- the *port maximal output data rate* b^{out}_{max}, expressed in bytes/s;

□

The *data terminals* DI and DO are those terminals of a port that transport actual data. A port may have data input terminals, output or both, but there must be terminals classified as data terminals. The sets of *control terminals* CI and CO are the terminals which actually implement the control protocol signaling. Typically, the minimal number of control terminals in a port is a pair of terminals to implement handshake (e.g. valid and accept terminals), but there may be other terminals used, for example, to identify the end of a data transfer, to do error signaling, among others. The *maximal data rates* b^{in}_{max} and b^{out}_{max} represent the maximal sustainable data rate that can be assigned to a port in functional mode. These values are defined by the functional application or system specification. Taking the port illustrated in Figure 3 as example. The signals *Addr*, *BurstSize*, *Cmd*, *DataValid*, and *respAccept* are classified as CO; *CmdAccept*, *DataAccept*, and *RespValid* are classified as CI; *Data* as DO, and *RespData* as DI.

A core may have a set of ports connected to the functional interconnect. The core may also have other terminals connected elsewhere, e.g. chip pins, rather than the functional interconnect. Definition 2 classifies a complete core for test considering these issues.

Definition 2 [Core].
- set FI of *functional input terminals*;
- set FO of *functional output terminals*;
- set SI of *scan input terminals*;
- set SO of *scan output terminals*;
- set S of *scan chains*, where for each $s \in S$ its length is denoted by $l(s)$;
- set P of *ports*;
- the *system test frequency* f, expressed in Hz.

□

The *functional terminals* FI and FO consist of those terminals not connected to the interconnect. The functional terminals can also consist of ports connected to the interconnect, but are not used for test. The *scan terminals* SI and SO consist of terminals used to connect the wrapper cells to the set of internal scan chains S. The set of *ports used for test* P is defined according to Definition 1.

5 Wrapper Model

Figure 4 presents the proposed wrapper model for interconnect reuse. It has three main parts: test wires, wrapper cells, and control part. The wrapper also has the CUT specified as in Definition 2. We are assuming that the core may be a hard-core or encrypted core, and no modifications are allowed on it. Figure 4 has one port ($port_1$) and its terminals are classified in the sets CI_1, DI_1, CO_1, DO_1 terminals. More ports are allowed. Then, the $port_n$ would have the sets CI_n, DI_n, CO_n, DO_n terminals. The sets FI, FO, SI, and SO are unique for the whole core. The wrapper cells are connected by test wires. The control and protocol signals are generated in the control logic. Each core port is connected to one interconnect port. The interconnect port is split in the input and output parts to ease the figure.

5.1 Wrapper Cells

All functional terminals of a core require wrapper cells. The terminals classified as DI, DO, and CI use the standard wrapper cell illustrated in Figure 4(c) (the CI terminals use the standard cell because, as they are inputs for the wrapper, the wrapper does not control these signals). The terminals classified as CO use the proposed wrapper cell shown in Figure 4(b). Both cells have functional/scan input/output terminals as

well as control terminals to operate the muxes. The proposed cell has an additional multiplexer. The terminal *prot_in* receives from the control logic the required value to play the protocol (the actual protocol is implemented in the control logic, Section 5.3, not in the wrapper cell). During test mode, the terminal *prot_mode* is asserted to '1' to assure that test signaling does not interfere with the functional protocol.

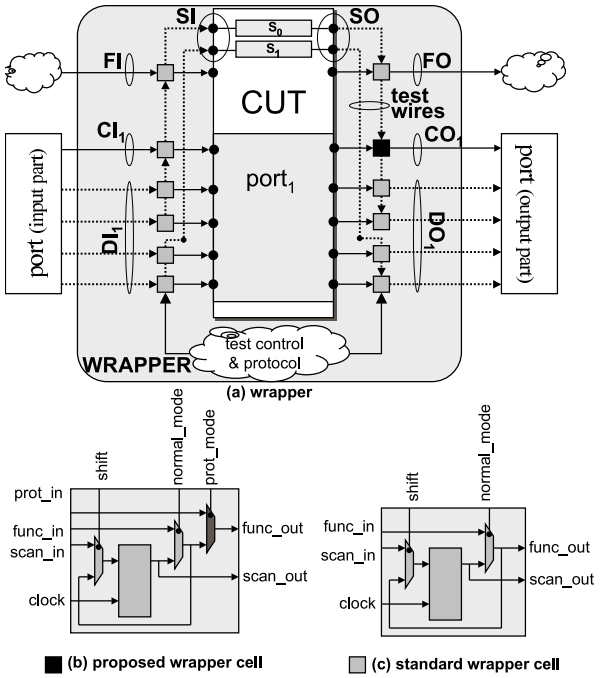

Figure 4: Wrapper model.

5.2 Defining the Test Wires

We define *test wires*, illustrated in Figure 4(a) in dotted lines, as those used to connect the wrapper cells and internal scan chains. A wrapper must have at least one test wire. In general, the more test wires a wrapper has, the shorter is the test time. The order of elements in a test wires, illustrated in Figure 5, must be first one or more wrapper cells connected to data input terminals. Secondly, all the remaining input wrapper cells. Third, zero or more internal scan chains. Then, all output wrapper cells, such that the last elements are connected to data terminals. This order enables the scan-in and scan-out pipelining effect to reduce the core test time (*goal i*).

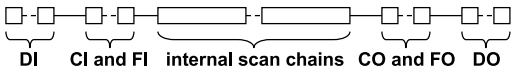

Figure 5: Ordering of test wires elements.

The complete specification of test wires is done in three steps: defining the number of test wires, defining the number of data terminals per test wire, and balancing the scan-in and scan-out.

5.2.1 defining the maximal number of test wires

The number of *test wires* tw is defined in Equation 5.1. It refers to the maximal number of test wires a core can have considering the maximal bandwidth for data input and data output. b^{in}_{max} and b^{out}_{max} are, respectively, the maximal input bandwidth and the maximal output bandwidth, expressed in bytes/s, that can be assigned to the port. The factor 8 is used since the bandwidth is expressed in bytes/s. f is the test frequency. The floors are required to have discrete number of test wires. In addition, equal number of test wires for scan-in and scan-out is required for test, then the minimal is used.

$$tw = min(\lfloor \frac{b^{in}_{max} \times 8}{f} \rfloor, \lfloor \frac{b^{out}_{max} \times 8}{f} \rfloor) \quad (5.1)$$

5.2.2 defining the number of data terminals per test wire

We define the *number of data terminals per test wire* for data input and output in Equations 5.2 and 5.3 respectively. Equal number of data terminals is important to do a serial-to-parallel conversion of di wires to tw wires without corrupting tw. For example, consider $|DI| = 32$ and $tw = 2$. Then, each of the two test wires would have 16 data input terminals. On the other hand, if this constraint were not applied, it would be possible to have one test wires with 20 input data terminals, and the other with 12 data terminals. Problems would arise during the parallel-to-serial conversion. Either the second test wire would have eight bits corrupting the test every word read from the interconnect, or the first test wire would have eight unused bits. There is a second issue regarding this constraint. Consider, for example, $|DI| = 32$ and $tw = 3$. Then, the three test wires would have ten data input terminals. The data in two of the $|DI|$ terminals would be ignored. The last issue is that, as presented in Section 6.3, this constraint may imply in a slight increase of test time compared to a standard test wrapper. Analogously, there is a parallel-to-serial conversion to send responses to the interconnect.

$$di = \lfloor \frac{|DI|}{tw} \rfloor \quad (5.2)$$

$$do = \lfloor \frac{|DO|}{tw} \rfloor \quad (5.3)$$

From the interconnect point of view, the number of data terminals di and do means the period data is read or sent. Every di clock cycles a word is read from the interconnect; meanwhile, this word is shifted into the wrapper cells. The interconnect must be able to provide data in these intervals. A violation in the data request would corrupt the test wires. The guaranteed services play an important role for test since it assures this periodicity. Previous approach could not assume it. Usually they assume clock gating or large buffers in the wrapper. Thus, reusing guaranteed throughput service for test enables the creation of a simpler wrapper. Just a simple parallel-to-serial conversion is required.

5.2.3 balancing the scan-in and scan-out

This step distributes the core internal scan chains and wrapper cells (except those related to the data terminals, which were already defined) among the test wires in order to minimize the core test time. The test time T is defined as

$$T = \{1 + \max(s_i, s_o)\} \times p + \min(s_i, s_o)\} \quad (5.4)$$

where p denotes the number of test patterns, and s_i and s_o denote respectively the scan-in and scan-out time for a core.

Since the number of test patterns is fixed, in order to reduce the test time, the maximal scan-in and scan-out time should be reduced. Algorithms like LPT [2] have been proposed to minimize the core test time by balancing test wires. The same algorithm can be used in our approach, but a modification is required to support the constraint related to the equal number of data terminals per test wire. Due to this constraint, the resulting $\max(s_i, s_o)$ of the proposed approach may be slightly bigger then the one used in wrappers with dedicated TAMs. Examples are presented in Section 6.3.

5.3 Wrapper Control and Protocol

When the core is in test mode, it cannot play the protocol with the interconnect to keep the data flowing. In the proposed approach, a subset of the protocol is implemented in the wrapper.

All the CO terminals require a protocol signal. However, the large majority of the CO terminals requires only a hard coded '0' or '1'. For a few terminals, it is necessary to change its value over the time. For these terminals a small finite state machine have to be implemented. The exact logic to be implemented depends on the protocol used, and the role of the terminal in the protocol.

Lets take the port illustrated in Figure 3 as an example. Let us consider that this port is used to send test responses. In this way, the responses are sent via terminal *Data*; terminal *RespData* is not used, hence the CO terminal *RespAccept* is tied to '0'; the CO terminal *Addr* is not relevant for test and it can be asserted to zero; the CO terminal *BurstSize* must be assigned with the number of words to be transfer in each burst;

The CO terminals *DataValid* and *Cmd* require a FSM. *DataValid* must be asserted high every di clock cycles to send a new word; the CO terminal *Cmd* must be asserted high when the number of words specified in *BurstSize* were sent. As one can see, the FSM required to these two terminals is simple and depends only on two internal counters.

The typical steps performed by the control logic during internal test mode are: load data into the input data wrapper cells; shift it to the test wires to fill remaining wrapper cells and core internal scan chains; apply test patterns to the CUT; capture CUT outputs; shift the responses; send the responses in the output data wrapper cells to the interconnect.

Once the wrapper is completely specified, the required *deliverables* for the system integration are the wrapper itself and the bandwidth the interconnect should deliver to the port during the core test. The *actual bandwidth* assigned for test, b_{act}^{in} and b_{act}^{out}, is defined in Equation 5.5. It represents the minimal bandwidth that can sustain tw test wires. Thus, this model guarantees the maximal number of test wires using the minimal amount of interconnect bandwidth (*goal i*).

$$b_{act}^{in} = b_{act}^{out} = \lceil \frac{f \times tw}{8} \rceil \tag{5.5}$$

6 Experimental Results

6.1 Experimental Setup

Figure 6 shows the system described in VHDL to validate our wrapper. It has three cores: the test source; the test sink; and the CUT which is involved by the wrapper further described in Section 6.2. The *interface* between the cores and the interconnect is via a DTL protocol configured with 32 data bits. The *interconnect* is a Æthereal NoC automatically generated with four network interfaces (NI) and one router (R). The Æthereal instance matches the interconnect model illustrated in Figure 2: the protocol box in the model matches with the DTL port; the communication service box in the model matches with the NI; and the interconnect network matches with the router network. The numbers in Figure 6(1 to 4) identify the group of signals in Figure 8(1 to 4).

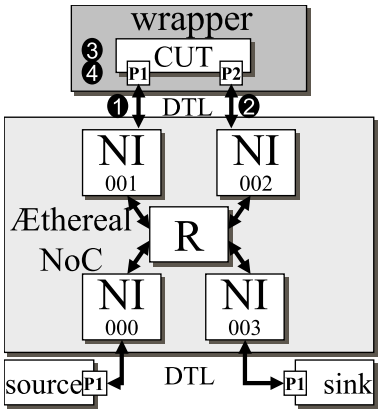

Figure 6: Target system.

Although the system used as example is simple, we chose it just for sake of readability and understanding. The guaranteed communication services implemented in the Æthereal NI abstract the internal implementation completely, for example, the topology of the interconnect network.

6.2 Wrapper Architecture

The *CUT* has two complete DTL ports; one to receive test stimuli from the source (port1), and the other to send responses to the sink (port2). The CUT is configured with five internal scan chains with 123, 123, 50, 50 and 23 cells, respectively. Each DTL port has 133 terminals. Both of them have 32 DI terminals, as well as 32 DO terminals. Port1 has 62 CI and 7 CO terminals, while port2 has 7 CI and 62 CO. The core has no terminals classified as FI and FO. The maximal bandwidths are $b_{max}^{in} = 200$ MB/s and $b_{max}^{out} = 300$ MB/s. The test frequency is $f = 500$ MHz. The number of test patterns is $p = 10$.

Equation 5.1 results in $tw = 3$ test wires. The number of input and output data terminals per test wire is $di = 10$ and $do = 10$, respectively. After balancing the test wires, the maximal scan-in and scan-out time is $\max(s_i, s_o) = 168$, while the minimal scan-in and scan-out time is $\min(s_i, s_o) = 167$. Then, the core test time is $T = 1857$ clock cycles.

The implemented wrapper for this core is presented in Figure 7. The wrapper has three test wires identified by the dotted lines. The test wires start with DI_1 (DI of port1) terminals used to receive test stimuli from the interconnect, followed by CI_1, DI_2 (not used to receive test stimuli), CI_2, internal scan chains, DO_1 (not used to send test responses), CO_1, CO_2, and DO_2 used to send test responses to the interconnect. Note that the terminals $DI_{1_{10}}$ (terminal 10 in the set DI of port1), $DI_{1_{21}}$, $DO_{2_{10}}$, and $DO_{2_{21}}$ belong to the test wires, but are not being used to carry test data (they have darker lines). This effect happens because the division $\frac{|DI|}{tw}$ has remainder different than zero.

6.3 Results

As presented in Section 5, more bandwidth may result in more test wires for the wrapper, which may reduce the core test time. The test time and wrapper area are compared with an approach for dedicated TAMs [3], since it provides enough information to reproduce their

wrapper.

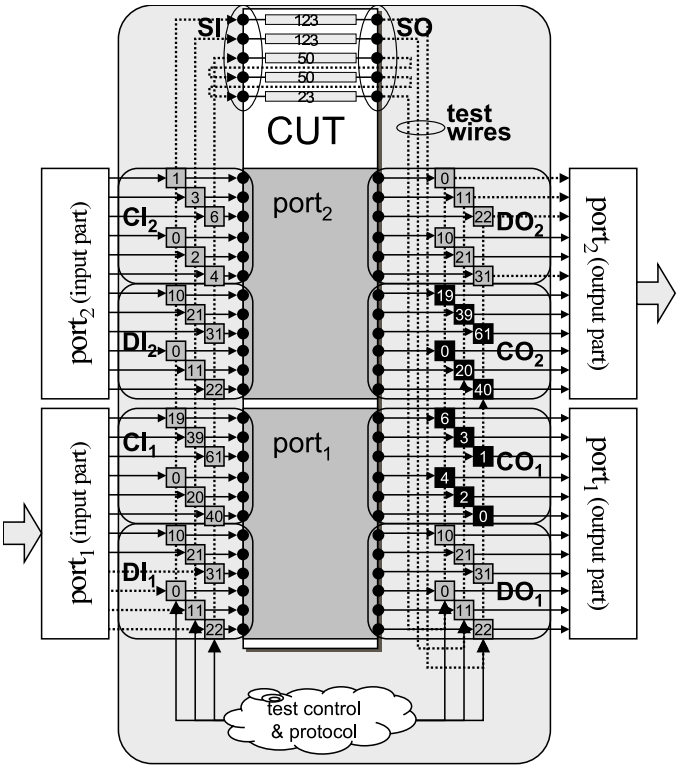

Figure 7: Wrapper instance with three test wires.

Table 1 shows the core test time as function of the assigned bandwidth (b_{act}). The used core is the same presented in Section 6.2.

b_{max} (MB/s)	b_{act} (MB/s)	test wires	our test time	[3] test time	additional test time (%)
63 to 124	63	1	5532	5532	0
125 to 199	125	2	2771	2771	0
200 to 249	200	3	1857	1857	0
250 to 333	250	4	1429	*1395*	2.4
334 to 399	334	5	1306	*1250*	4.3
400 to 499	400	6	1295	*1244*	3.9

Table 1: Test time (in clock cycles) as function of bandwidth.

As the bandwidth is increased, the number of test wires increases, and the core test time decreases. Last column shows the test time for [3]. Different test time between our approach and [3] were observed for wrappers with more then three test wires. Let us take the wrapper with four test wires as an example. Using Equations 5.2 and 5.3, we derive $di = do = 8$. The balanced the scan chains is presented in Table 2 for our approach and for [3].

The previous approach [3] does not require constrains when balancing the test wires. When there is the situation where a single test wire is the bottleneck for the scan time (as the test wires tw[0] and tw[1] in the example), the previous approach present slight better test time since their approach can distribute better the wrapper cells, resulting in smaller scan-in and scan-out times. However, the difference in the scan times is small, since the our constraint increases the scan time in only $\frac{|DI|}{tw}$ clock cycles for the scan-in and $\frac{|DO|}{tw}$ clock cycles for the scan-out time.

test wire	DI_1	$CI_1 + DI_2 + CI_2$	scan chains	$DO_1 + CO_1 + CO_2$	DO_2
tw[0]	8	0	{123}	0	8
tw[1]	8	0	{123}	0	8
tw[2]	8	50	{50,23}	50	8
tw[3]	8	51	{50}	51	8
tw[0]	2	0	{123}	0	2
tw[1]	2	0	{123}	0	2
tw[2]	3	50	{50,23}	50	3
tw[3]	25	51	{50}	51	25

Table 2: Comparing test wires between our approach and [3].

We compare the area to implement a wrapper, for the core presented in Section 6.2, using our approach and [3]. The comparison evaluates the area to implement the wrapper cells and the control logic. The CUT requires 266 wrapper cells. In the proposed wrapper, 69 out of 266 use the new wrapper cell. The area to implement all the 266 cells is 3000 equivalent gates. On the other hand, the area to implement 266 wrapper cells for the previous wrapper is 2910 gates. Thus, the proposed approach requires +3% of area to implement the wrapper cell then the previous approach. The comparison of control logic is not straight-forward. The previous wrapper has system-level control logic shared between all cores. Thus, in on hand it requires fewer gates to implement the control, on the other hand it has longer wires from the CUTs to the centralized control. The wire length overhead depends on the size of the chip, and place-and-route. The proposed approach has distributed and independent control logic. Each wrapper has its own control logic. While the wire length overhead is minimal, the gate count is higher compared to the previous wrapper. The control logic presented in Section 6.2 requires 489 equivalent gates, but it may change slightly for other cores due to the size of internal counters.

On one hand, our approach has marginally higher test time and area overhead. On the other hand, the wrapper presented in [3] cannot be applied for functional interconnect reuse for test because they ignore the protocol and don't have support to convert data terminals to test wires.

Figure 8 presents a waveform with some relevant signals of the system presented in Figure 6. The waveform is divided into four parts: (1) signals between the NoC and input port $port_1$ of the CUT, used to show periodic data timing and the protocol activity. The signal *dtl_cmd_valid_target* represents the period the CUT receive a new burst of data. The signal *dtl_wr_accept_target* represents the fixed intervals when a word is read from the interconnect. The signal *dtl_wr_valid_target* shows when the interconnect has data available; (2) signals between the output port $port_2$ of the CUT and the NoC. This port can send data to the NoC whenever it has data to send (the signal *dtl_wr_valid_initiator* is not periodic). The interconnect makes sure that the destination receives them in a periodic manner. (3 and (4) internal wrapper signals used to control scan-in and scan-out times. They represent the moment test stimuli are applied to the core input terminals, and the moment the output terminals are captured into the output wrapper cells. Both ports work simultaneously to enable parallel scan-in and scan out.

7 Conclusion

We proposed the reuse of interconnects with on-chip protocol and guaranteed bandwidth and latency services as a TAM. These properties abstract the interconnect implementation and provide the predictability in the data transfer required by the test application. On top of this interconnect, the paper presented a general wrapper model for reuse

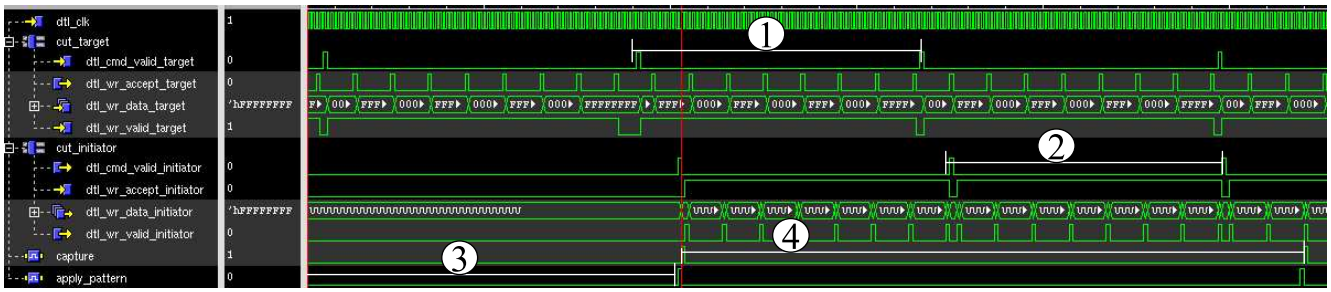

Figure 8: Waveform.

of functional interconnect as a TAM. The wrapper design method is compatible with the well-defined and existing tools (e.g. interconnect design tools), equipments (e.g. tester), standards (e.g. P1500, DTL, OCP, and AXI), and concepts (e.g. separation between the computation and the communication). The proposed wrapper was implemented in VHDL and integrated with Æthereal NoC. Results for core test time and area overhead were compared with a wrapper design for dedicated TAM.

References

[1] Y. Zorian, E.J. Marinissen, and S. Dey. Testing Embedded-Core Based System Chips. In *Proc. ITC*, pages 130–143, 1998.

[2] E.J. Marinissen, S.K. Goel, and M. Lousberg. Wrapper Design for Embedded Core Test. In *Proc. ITC*, pages 911–920, 2000.

[3] S.K. Goel and E.J. Marinissen. SOC Test Architecture Design for Efficient Utilization of Test Bandwidth. *ACM TODAES*, 8(4):399–429, October 2003.

[4] L. Benini and G. De Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, 35(1):70–80, 2002.

[5] K. Keutzer et al. System-Level Design: Orthogonalization of Concerns and Platform-Based Design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.

[6] A. Rădulescu et al. An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Programming. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 24(1):4–17, January 2005.

[7] E. Cota et al. Test Planning and Design Space Exploration in a Core-Based Environment. In *Proc. DATE*, pages 478–485, 2002.

[8] J-R. Huang et al. A Self-Test Methodology for IP Cores in Bus-Based Programmable SoCs. In *Proc. VTS*, pages 198–203, 2001.

[9] S. Hwang and J. A. Abraham. Reuse of Adrdressable System Bus for SOC Testing. In *IEEE ASIC/SOC*, pages 215–219, 2001.

[10] P. Harrod. Testing Reusable IP - A Case Study. In *Proc. ITC*, pages 493–498, 1999.

[11] C. Feige et al. Integration of the Scan-Test Method into an Architecture Specific Core-Test Approach. In *Proc. ETW*, pages 241–245, 1998.

[12] W. J. Dally and B. Towles. Route Packets, not Wires: on-Chip Interconnection Networks. In *Proc. DAC*, pages 684–689, 2001.

[13] E. Cota, L. Carro, and M. Lubaszewski. Reusing an On-Chip Network for the Test of Core-based Systems. *ACM TODAES*, 9(4):471–499, 2004.

[14] L. Chunsheng et al. Power-Aware Test Scheduling in Network-on-Chip Using Variable-Rate On-Chip Clocking. In *Proc. VTS*, pages 349–354, 2005.

[15] OCP International Partnership. *Open Core Protocol Specification. Release 1.1.1*, 2003.

[16] Philips Semiconductors. *Device Transaction Level (DTL) Protocol Specification. Version 2.2*, July 2002.

[17] ARM. *AMBA AXI Protocol Specification*, June 2003.

[18] T. Bjerregaard and J. Sparso. Scheduling Discipline for Latency and Bandwidth Guarantees in Asynchronous Network-on-Chip. In *11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 34–43, 2005.

[19] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip. In *Proc. DATE*, volume 2, pages 890–895, 2004.