Wrapper Design for Embedded Core Test

Erik Jan Marinissen

Philips Research Laboratories
Dept. Digital Design & Test
Prof. Holstlaan 4, M/S WAY-41
5656 AA Eindhoven, The Netherlands
Erik.Jan.Marinissen@philips.com

Sandeep Kumar Goel *

Indian Institute of Technology
Dept. of Comp. Science and Engineering
New Delhi, Zip 110016
India
Sandeepkumar.Goel@philips.com

Maurice Lousberg

Philips Research Laboratories
Electronic Design & Tools
Prof. Holstlaan 4, M/S WAY-31
5656 AA Eindhoven, The Netherlands
Maurice.Lousberg@philips.com

Abstract

A wrapper is a thin shell around the core, that provides the switching between functional, and core-internal and core-external test modes. Together with a test access mechanism (TAM), the core test wrapper forms the test access infrastructure to embedded reusable cores. Various company-internal as well as industry-wide standardized but scalable wrappers have been proposed. This paper deals with the design of such core test wrappers. It gives a general architecture for wrappers, and describes how a wrapper can be built up from a library of wrapper cells which are selected on basis of the terminal types of the core. We show that the ordering and partitioning of wrapper cells and core-internal scan chains over TAM chains determines the test time of the core. A heuristic approach for the $\mathcal{NP}$-hard problem of partitioning the TAM chain items for minimal test time is presented and its usage is illustrated by means of an example. Finally we sketch how wrapper generation and verification can be automated.

1 Introduction

Modern semiconductor process technologies enable the manufacturing of a complete system on one single die, the so-called system chip. Such system chips typically are very large ICs, consisting of millions of transistors, and contain a variety of hardware modules. In order to design these large and complex system chips in a timely manner and leverage from external design expertise, increasingly reusable cores are utilized. *Cores* are pre-designed and pre-verified design modules. Examples of cores are CPUs, DSPs, media coprocessors, communication modules, memories, and mixed-signal modules. Core-based IC design divides the IC design community into two groups: *core providers* and *core users*.

In order to guarantee that test development for large core-based system chips does not become the bottleneck in the overall development trajectory, it is recommended that their test development is also core-based [1]. This means that core providers should deliver a set of appropriate tests with their product, which are used by core users to create the overall IC test. Due to the distribution of tasks over the multiple parties involved, core-based test development introduces several new challenges compared to test development for traditional IC designs [2]. These include (1) the transfer of sufficient 'test knowledge' from core provider to core user, (2) test access to the often deeply embedded cores, and (3) chip-level optimization of test-related costs such as test time, silicon area, etc.

Zorian et al. [1] introduced a generic conceptual test access architecture for embedded cores as well as a, now generally used, nomenclature for its elements (cf. Figure 1). Real-time stimulus generation takes place in the test *source*, while real-time response evaluation is carried out by the test *sink*. Source and sink can either be implemented off chip (Automatic Test Equipment, ATE) or on chip (Built-In Self Test, BIST), or in a combination of both. The second element of the architecture is the *Test Access Mechanism* (TAM). The TAM serves as 'test data highway' in the sense that it bridges the physical distance between source and core, as well as between core and sink. Various alternative TAM implementations have been published, including TestRail [3], Test Bus [4], and Addressable Test Ports [5]. The third architectural element is the core test *wrapper*. The wrapper is a thin shell around the core that connects the TAM(s) to the core. The wrapper provides the switching between normal functional access and test access via the TAM. Well-designed wrappers provide test access for both core-internal testing, as well as core-external testing. Furthermore, wrappers may provide width adaptation in case of a mismatch between core I/O width

* Sandeep Goel is currently with the Dept. Digital Design & Test of Philips Research Laboratories in Eindhoven, The Netherlands.

0-7803-6546-1/00 $10.00

and TAM width, e.g., by means of serial-parallel and parallel-serial conversion. Examples of wrappers include TestShell [3], and Test Collar [4]. One of the elements of IEEE P1500 SECT, the Standard for Embedded Core Test under development [6], is a standardized, but scalable wrapper [7].

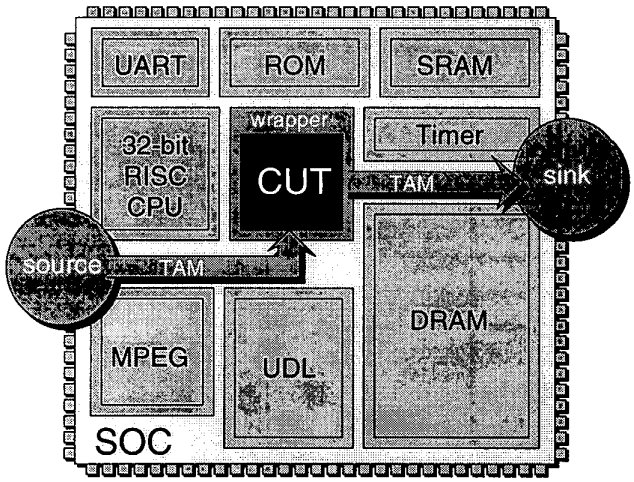

Figure 1: Generic conceptual test architecture [1] consisting of three elements: (1) source/sink, (2) test access mechanism, and (3) wrapper.

This paper addresses the issue of wrapper design for embedded cores. The wrapper is designed such that the access requirements for normal operation, core-internal testing, and core-external testing are met. Wrappers are built up from a library of wrapper elements. We specify the required minimum features of these library elements and indicate how they can be extended for additional functionality. We address the relationship between wrapper design and test time, and show how the wrapper can be optimized for reducing the test vector sets for core-internal and core-external testing. We also discuss how wrapper generation and validation can be automated, in such a way that the wrapper design can take advantage of existing features at the boundary of the core.

The sequel of this paper is organized as follows. Section 2 summarizes previous publications on core test wrappers. Section 3 describes the architecture of the wrapper considered in this paper and outlines its general features. A library of wrapper cells and a core-terminal-type-dependent selection mechanism for these cells is described in Section 4. In Section 5 we address the impact of wrapper design on test time. We give rules for the ordering of wrapper cells, show that the partitioning of the set of TAM items over TAM wires is equivalent to the well-known $\mathcal{NP}$-hard problem of Multi-Processor Scheduling, and provide a heuristic algorithm for it. The wrapper design procedures are put to work on an example core in Section 6, while Section 7 addresses the issue of automated wrapper generation and verification. Finally, Section 8 concludes this paper.

2 Prior Work

A core test wrapper named TestShell has been proposed by Marinissen et al. [3] and is currently used within Philips (cf. Figure 2). The TestShell consists of the following components.

- A *Test Cell* for every core terminal. The test cell provides controllability as well as observability.
- An (optional) *Bypass Register*, which allows a TAM to bypass core and wrapper, in order to test another core that is connected to the same TAM.
- A *Test Control Block* (TCB). The TCB has a bit-slice nature and consists of a shift and an update register. The TCB is primarily meant to control the operation of the TestShell, through several mandatory bit slices. Additional user-defined bit slices can be added for control of core-internal test modes.

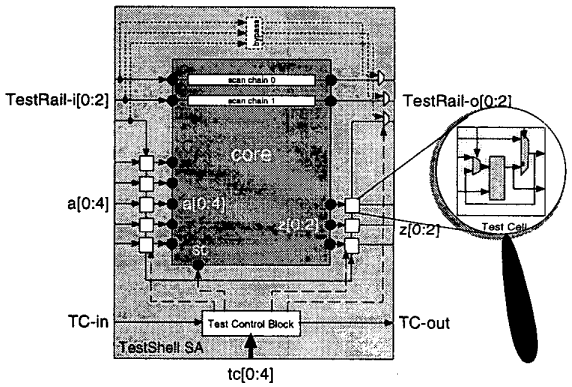

Figure 2: Conceptual view of Philips' TestShell [3].

The TestShell supports four basic modes: (1) normal functional mode, (2) IP test mode, (3) interconnect test mode, and (4) the bypass mode. In this approach, TAMs are called TestRails. In principle, a TestShell is connected to the same TestRail at both input and output. Therefore, the TAM input *plug* and the TAM output plug of a TestShell normally have the same width. The area costs of TestShell and TestRail depend on the size of the core, as well as the number of core terminals. Arendsen and Lousberg [8] reported 3.2% additional silicon area for TestShell and the associated TestRail for a 21.4 mm^2 industrial IC containing eight cores, on top of 4.5% area costs in order to make all cores fully scan testable.

Varma and Bhatia of Duet Technologies described a very similar wrapper, called *Test Collar* [4]. Apart from different naming

for basically similar features, the main difference between this and the previously described approach is that the Test Collars do not have a bypass feature. This makes that per Test Bus, only one core can be serviced at a time. With respect to area costs, Varma and Bhatia reported that "for designs containing significantly sized cores, the area overhead is small and can even be less than 1% in some cases".

The IEEE P1500 Standard for Embedded Core Test [6] is an IEEE standard under development that consists of two components, a Core Test Language to facilitate the test knowledge transfer from core provider to core user, and a Core Test Wrapper [7]. The wrapper (cf. Figure 3) is very similar to the previously described TestShell and Test Collar. The P1500 wrapper has *Wrapper Boundary Cells* and a *Wrapper Instruction Register* (WIR) with similar functionality to respectively the test cells and the Test Control Mechanism in TestShell. Nevertheless, there are some remarkable differences between TestShell and the current P1500 proposal.

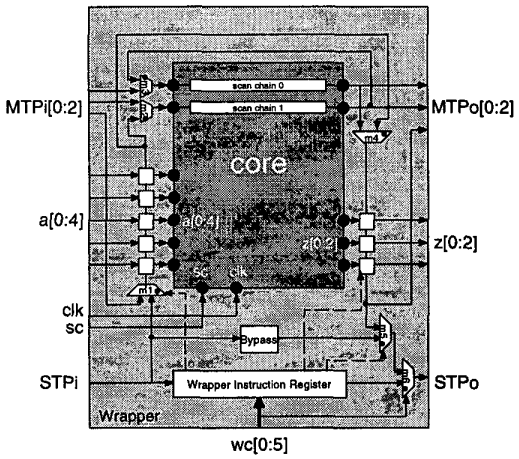

Figure 3: Conceptual view of IEEE P1500's wrapper [7].

- *Number of TAM plugs.* The P1500 wrapper connects to one mandatory one-bit wide TAM and zero or more scalable-width TAMs. A minimal compliant implementation has only the single-bit TAM plug (STP), along which both test control values for the WIR, as well as test stimuli and responses are transported. Envisaged typical usage has one multi-bit TAM plug (MTP) next to the mandatory STP. In that case, the bulk test data access is performed along the multi-bit TAM, while the single-bit wide TAM is used to program the WIR and possibly transport test data in a silicon debug scenario [9]. Multiple multi-bit wide TAMs are also allowed.
- *TAM widths.* Corresponding input and output MTPs need not to be of the same width. Hence, P1500 would allow an input MTP of n wide, while the corresponding output MTP is m wide (n and m both positive integers with not necessarily $n = m$).
- *Bypasses.* The two wrappers allow for different types of bypasses. The Philips wrapper has a TAM-wide bypass. The P1500 wrapper has a bypass for the single-bit TAM (enabled by multiplexer `m5`), next to the possibility to bypass the core-internal scan chains while accessing the wrapper boundary register (enabled by multiplexer `m4`).

The above publications on core test wrappers all provide general concepts for wrapper design. All approaches seem to assume that automated generation of their particular wrapper is possible. However, none of the above publications details the rules and algorithms required for such wrapper generator tools.

3 Wrapper Architecture

This section describes the architecture of the wrapper for which the design procedures are detailed in the sequel of this paper. This architecture is very similar to the ones described in the section above, and basically unites the features of both the Philips TestShell [3] and the IEEE P1500 Wrapper [7]. An example of our architecture is depicted in Figure 6.

For test data access, our architecture has one or more multi-bit TAM plug pairs, and corresponding plugs have equal width. For test control access, our architecture has one single-bit test control plug pair, through which instructions are loaded into the WIR. This plug can be multiplexed on top of a TAM plug. A minimal implementation of the above equals the minimal implementation as proposed by IEEE P1500. However, other wrappers as allowed by either Philips or P1500 also fall under our architecture.

The wrapper contains a wrapper cell per core terminal. The wrapper cell provides the required (test) access for the core terminal in question. In Section 4, a minimum library of wrapper cells is proposed that meets the (test) access requirements only. Optionally, the minimal (test) access functionality of the wrapper cells can be enhanced with additional functionality. Additional (de)multiplexers might be added in case multiple TAM chains share wrapper cells.

At the expense of a small amount of additional hardware, bypasses can significantly contribute to reducing test time. In our architecture, we allow for optional bypasses between any pair of corresponding TAM plugs. This includes the optional bypass between multi-bit TAM plugs in the Philips' TestShell, as well as the bypass between the single-bit TAM plug, which is mandatory in IEEE P1500. Furthermore, we allow for optional bypasses of core-internal scan chains, as is the case for

IEEE P1500. Following the Philips and P1500 wrapper examples, our bypasses contain not just wires and/or buffers, but also a register, as this allows for concatenating an arbitrarily large number of cores in the same TAM chain.

The wrapper cells and core-internal scan chains are connected into TAM chains in between the TAM plugs in order to meet the access requirements. Section 5 of this paper describes how to minimize the core's test time by exploiting the degrees of freedom with respect to the exact ordering and partitioning of the TAM chain items.

A *Wrapper Instruction Register* (WIR), equivalent to P1500's WIR and Philips' TCB, provides pseudo-static control signals to the wrapper itself. These signals control the mode of the wrapper by setting control signals of wrapper cells and bypass multiplexers. If desirable, core-internal test control signals can also be derived from the WIR. The WIR is implemented with a shift and update register. Via a single-bit interface, a new test control instruction is shifted into the WIR, which becomes active only afer clocking it into the update register. The update register prevents that invalid instructions are given to wrapper and core while shifting in a new instruction.

4 Library of Wrapper Cells

In our approach, a core test wrapper contains elements selected from a library of wrapper cells. Per core terminal, a wrapper cell is added to the wrapper. Which wrapper cell is used for a core terminal depends primarily on the type of core terminal and its corresponding (test) access requirements. These attributes can be found in the *test protocol* that comes with the core [10].

Section 4.1 presents a classification of core terminals and their access requirements. Subsequently, in Section 4.2 a wrapper cell library is presented of which the elements match these requirements. Finally, Section 4.3 sketches how the wrapper cells might be extended with additional functionality to support other features beyond (test) access only.

4.1 Classification of Core Terminals

Our classification of core terminals is based on the type (functional-only, test data, combined functional and test data, test control) and direction (input, output) of the terminal. In total ten classes are identified.

- *Functional-Only Terminals (F)*
  - *Fi*: functional-only input
  - *Fo*: functional-only output
  - *Fz*: functional-only tri-state enable output
- *Test Data Terminals (T)*
  Examples of test data terminals are the terminals of core-internal scan chains. We assume here that tri-stateable test data terminals do not exist.
  - *Ti*: test data input
  - *To*: test data output
- *Combined Functional / Test Data Terminals (FT)*
  It is possible to time-multiplex functional and test data terminals onto the same core terminals (as is also often done with functional and test data pins at chip level).
  - *FTi*: functional + test data input
  - *FTo*: functional + test data output
  - *FTz*: functional + test data tri-state enable output
- *Test Control Terminals (C)*
  We assume here that test control terminals are always inputs to the core.
  - *Cs*: pseudo-static test control input terminal, i.e., test control signals that change very infrequently, e.g., once per complete test. An example of a pseudo-static test control signal is a BIST-enable signal.
  - *Cd*: dynamic test control input terminal, i.e., test control signals that require instantaneous change. An example of a dynamic test control input is a scan-enable signal.

In this paper we address digital synchronous terminals only. Hence, analog signals and asynchronous signals such as clocks are excluded from our discussion. For these type of terminals, both Philips [3] as well as IEEE P1500 [7] have proposed 'direct test access', i.e., these signals pass the wrapper unhindered and can for example be directly connected to IC pins. Furthermore, we exclude bidirectional core terminals from our discussion. It is recommended not to have bidirectional terminals for embedded cores, but instead split them into separate input, output, and direction control terminals. In case certain (legacy) cores have bidirectional terminals in spite of this recommendation, it is best to create 'direct test access' for them as well.

The pseudo-static test control signals can be obtained from the WIR. The primary function of the WIR is to generate the pseudo-static signals necessary to control the operation of the wrapper itself. However, the bit-sliced WIR is extendable with user-defined bits, and the pseudo-static test control signals for the core can be such user-defined additional bits.

With respect to data access, we consider five different types.

- *Functional access*, i.e., from a core terminal to a part of the system chip outside the core itself.
- *Controllability* from the TAM for internal test.
- *Observability* from the TAM for internal test.
- *Controllability* from the TAM for external test.
- *Observability* from the TAM for external test.

Table 1 shows which access types are required for which terminal type. The functional-only terminals require functional access, as well as test access for core-internal and core-external testing. In addition, tri-state enable outputs require ripple-through protection to prevent bus conflicts while shifting test

data. Test terminals do not have a functional connection, and hence, the corresponding wrapper cells does not need functional access nor core-external test access. Combined functional/test data terminals (type *FT*) can operate as functional and test data terminal, and hence require the union of the access types that are required by the corresponding functional-only and test data terminal types.

Terminal Type	Access Type				
	Functional Access	Controllability from TAM		Observability from TAM	
		InTest	ExTest	InTest	ExTest
Fi	√	√			√
Fo + *Fz*	√		√	√	
Ti		√			
To				√	
FTi	√	√			√
FTo + *FTz*	√		√	√	

Table 1: Access requirements for the various terminal types.

4.2 Minimal Wrapper Cell Library

Figure 4 depicts a library of wrapper cells. Cell `wInput` offers transparency, controllability, and observability, and hence fulfils the needs of terminal types *Fi* and *FTi*. Cell `wOutput` is the mirror image of this cell and suits the needs of *Fo* and *FTo*. Cell `wZOutput` has an additional gate to prevent ripple-through of unwanted control values and matches the needs of *Fz* and *FTz*. Cells `wTInput` and `WTOutput` match the simple test access requirements of *Ti* and *To* respectively. Cells `wMux` and `wDemux` provide the capability to combine and decombine multiple TAM chains through a subset of wrapper cells.

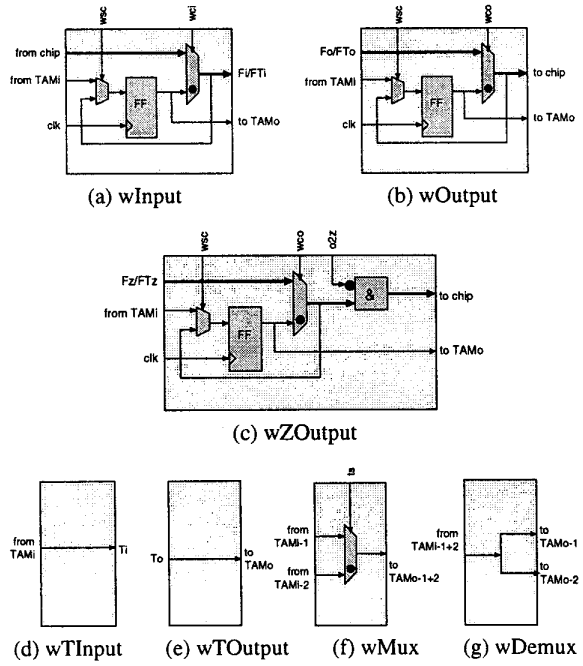

Figure 4: Minimal wrapper cell library.

This library of wrapper cells has the following properties.

1. Access for functional mode, as well as core-internal and core-external testing.
2. Ripple-through protection for outputs that require this, such as tri-state enable outputs.
3. Capability to combine and decombine multiple TAM chains through subsets of wrapper cells.
4. Full testability of all wrapper cells. In cells `wInput`, `wOutput`, and `wZOutput`, this is achieved by observing values after the multiplexer, instead of in front of it.

We consider this a minimal set of requirements and hence call the corresponding wrapper cell library minimal.

4.3 Enhanced Wrapper Cells

The library can be extended with cells that provide additional functionality beyond the minimal requirements. Some possible extensions are the following.

- Non-rippling wrapper cells implemented with a second flip flop, instead of a gate. This allows the wrapper user to program the value that is presented at the wrapper cell output, rather than have it hard-wired.
- Wrapper cells implemented with multiple flip flops, say k, which allow to deliver stimuli and/or capture responses in k consecutive (at-speed) clock cycles.
- Wrapper cells with Built-In Self Test (BIST) capabilities, i.e., wrapper cells that generate stimuli and/or compare or compact responses.

5 TAM Chain Design

Once the appropriate library cells are selected for a given core, the remaining task in order to complete the wrapper design is to make the interconnections between the wrapper cells, the core-internal scan chains, and the TAM plugs. We refer to this activity as *TAM chain design*, and call the elements that make up a TAM chain the *TAM chain items*. Test access is already guaranteed if all TAM chain items are accessible from the TAM plugs. In this section, we show that the partitioning over and ordering within TAM chains of the items has a large impact on the size

of the resulting test vector set. As pointed out in [11], the size of the test vector set is an important cost factor in high-volume testing of ICs. A large test vector set implies that the test application time is long and, often even more importantly, that the IC in question can only be tested on expensive test equipment having equally large pin memories to store those test vectors. Therefore, reduction of the test vector set size is desirable.

Wrappers are used both for core-internal and core-external testing. Optimizing the wrapper w.r.t. the test vector set for core-internal testing might lead to conflicting requirements w.r.t. optimization for the core-external test vector set. In the typical case, the core-internal circuitry is much larger than the circuitry that is used to interconnect the cores, and hence the test data volume involved in core-internal testing is much larger than the test data volume for core-external testing. Moreover, in many cases, the wrapper is designed by the core provider to whom the circuit environment in which the core will be used is not known, and hence data about the core-external test is not available at wrapper design time. Therefore, we give priority to optimizing the core-internal test vector set in the sequel of this paper.

5.1 Ordering of TAM Chain Items

The test time T for a core is defined as

$$T = \{1 + max(s_i, s_o)\} \cdot p + min(s_i, s_o), \qquad (5.1)$$

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. Note that this formula is valid even for non-scan-testable cores, for which $s_i = s_o = 0$.

From the set of all TAM chain items, two non-disjunct subsets are involved in loading and unloading of test patterns. The wrapper input cells and the core-internal scan chains (we refer to these as the *input items*) participate in loading of test patterns, while the wrapper output cells and the core-internal scan chains (we refer to these as the *output items*) participate in unloading of test patterns. In order to reduce s_i and s_o, it is best to order the items in any TAM chain such that the input items are at the head, and the output items are at the tail of the TAM chain. Given the fact that core-internal scan chains are in both sets, they should be in the middle of a TAM chain.

Figure 5 shows a generic template for a single TAM chain. The items are ordered such that the TAM chain contains subsequently (1) wrapper input cells, (2) core-internal scan chains, and (3) wrapper output cells. Optionally we can provide a *bypass* for the core-internal scan chains. These scan chains do not take part in the core-external testing, and at the cost of a multiplexer and an additional control wire, we can reduce the length of the access chain by bypassing them during the core-external tests. Also optionally, we can provide a bypass for the entire TAM chain in this wrapper. Such a bypass is particularly useful if multiple cores are concatenated into a single TAM, such as is the case in the Daisychain Architecture as described in [11]. Cores which are not tested, can be bypassed in order to reduce the access length to cores which are tested. As multiple cores are daisychained into one TAM, this might lead to long TAM wires and hence to long propagation delays. In order to prevent propagation delays from becoming too long and to contribute to the *plug-n-play* character of our wrapper, we propose to equip the wrapper bypass with a register.

5.2 Partitioning of TAM Chain Items

The TAM width is the result of a trade-off between its transport capacity and associated costs w.r.t. additional IC pins, silicon area, etc. [3]. Therefore, in many practical cases, the total number of TAM items is much larger than the width of the TAM plugs. If this is the case, it is required that the set of TAM items is partitioned into a number of subsets equal to the number of available TAM chains.

The partitioning of TAM items over TAM wires determines the scan-in time s_i and scan-out time s_o for the core, and hence determines its test time. As can be derived from Equation 5.1, the test time is minimal if the maximum of s_i and s_o is minimal. Hence, we are looking for a partitioning of the TAM items that achieves this minimal test time.

The partitioning problem can be formulated as finding an

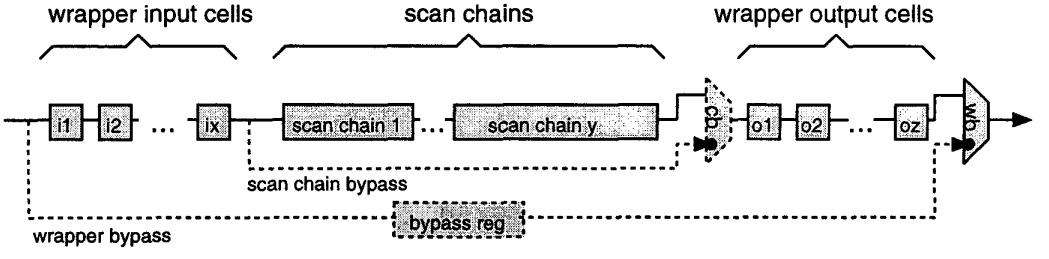

Figure 5: Ordering of TAM chain items (optional items are dashed).

assignment of all TAM items to one of the available TAM chains such that the maximum of scan-in and scan-out times is minimized. This problem can be formalized as follows [12].

Problem 1 [Partitioning of TAM Chain Items (PTI)]
Given a set $\mathcal{WI} = \{WI_1, WI_2, \ldots, WI_x\}$ of *wrapper input cells*, each wrapper input cell having a *length* $l(WI_i) = 1$. Given a set $\mathcal{S} = \{S_1, S_2, \ldots, S_y\}$ of *core-internal scan chains*, where scan chain S_i has *length* $l(S_i)$. Given a set $\mathcal{WO} = \{WO_1, WO_2, \ldots, WO_z\}$ of *wrapper output cells*, each wrapper cell having a *length* $l(WO_i) = 1$. Given a set of m identical TAM chains. We define for any $X \subseteq \mathcal{WI} \cup \mathcal{S} \cup \mathcal{WO}$, $l(X) = \sum_{x \in X} l(x)$. A *TAM partition* is a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ of $\mathcal{WI} \cup \mathcal{S} \cup \mathcal{WO}$ into m disjoint sets, one for each TAM chain. We define *input set* $IN_i = P_i \backslash \mathcal{WO}$. Likewise, we define *output set* $OUT_i = P_i \backslash \mathcal{WI}$. The *scan-in length* for TAM partition $\mathcal{P}$ is defined by $si(\mathcal{P}) = max_{1 \leq i \leq m} l(IN_i)$. The *scan-out length* for TAM partition $\mathcal{P}$ is defined by $so(\mathcal{P}) = max_{1 \leq i \leq m} l(OUT_i)$. Find an *optimal* TAM partition $\mathcal{P}^*$, i.e., one that satisfies $max(si(\mathcal{P}^*), so(\mathcal{P}^*)) \leq max(si(\mathcal{P}), so(\mathcal{P}))$ for all partitions $\mathcal{P}$ of $\mathcal{WI} \cup \mathcal{S} \cup \mathcal{WO}$ into m subsets. □

To solve the PTI problem, we use a three-step approach.

1. Assign the core-internal scan chains in $\mathcal{S}$ to TAM chains, such that the maximum sum of scan lengths assigned to a TAM chain is minimized. The resulting partition is named $\mathcal{P}_\mathcal{S}$.
2. Assign the wrapper input cells in $\mathcal{WI}$ to TAM chains on top of $\mathcal{P}_\mathcal{S}$, such that the maximum scan-in time of all TAM chains is minimized.
3. Assign the wrapper output cells in $\mathcal{WO}$ to TAM chains on top of $\mathcal{P}_\mathcal{S}$, such that the maximum scan-out time of all TAM chains is minimized.

Note that wrapper input cells and wrapper output cells have length 1. Therefore, Steps 2 and 3 of our approach can yield an optimal solution in linear compute time, if Step 1 was solved to optimality. The first step is the problem of partitioning of scan chains over TAM chains, which can be formalized as follows.

Problem 2 [Partitioning of Scan Chains (PSC)]
Given a set $\mathcal{S} = \{S_1, S_2, \ldots, S_y\}$ of *core-internal scan chains*, where scan chain S_i has *length* $l(S_i)$, and a set of m identical TAM chains. A *scan partition* is a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ of $\mathcal{S}$ into m disjoint sets, one for each TAM chain. TAM chain i, $1 \leq i \leq m$, contains all scan chains in P_i. The *scan length* for scan partition $\mathcal{P}$ is defined by $s(\mathcal{P}) = max_{1 \leq i \leq m} l(P_i)$, where for any $X \subseteq \mathcal{S}$, $l(X) = \sum_{S \in X} l(S)$. Find an *optimal* scan partition $\mathcal{P}^*$, i.e., one that satisfies $s(\mathcal{P}^*) \leq s(\mathcal{P})$ for all partitions $\mathcal{P}$ of $\mathcal{S}$ into m subsets. □

The PSC problem is equivalent to the well-known problem of Multi-Processor Scheduling (MPS), sometimes referred to as Bin Design [13]. In the MPS problem, n independent tasks have to be non-preemptively scheduled on m identical parallel processors with the objective of minimizing the 'makespan', i.e., the total time span required to process all given tasks. A formal version of the MPS problem is given below.

Problem 3 [Multi-Processor Scheduling (MPS)]
Given a set $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ of *tasks*, where task T_i has execution *length* $l(T_i)$, and a set of m identical *processors*. A *schedule* is a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ of $\mathcal{T}$ into m disjoint sets, one for each processor. Processor i, $1 \leq i \leq m$, executes the tasks in P_i. A *finishing time* for schedule $\mathcal{P}$ is defined by $f(\mathcal{P}) = max_{1 \leq i \leq m} l(P_i)$, where for any $X \subseteq \mathcal{T}$, $l(X) = \sum_{T \in X} l(T)$. Find an *optimal* processor schedule $\mathcal{P}^*$, i.e., one that satisfies $f(\mathcal{P}^*) \leq f(\mathcal{P})$ for all partitions $\mathcal{P}$ of $\mathcal{T}$ into m subsets. □

There is a direct one-to-one mapping between the PSC and MPS problems. In wrapper design, tasks are formed by the scan chains. The execution length of a tasks is equivalent to the length of a scan chain. The set of identical processors corresponds to the set of identical TAM chains. Note that we have tried to emphasize the equivalence of the two problems by the way we formalized them above.

The MPS problem is $\mathcal{NP}$-hard [14]. Because of the one-to-one mapping between PSC and MPS, we claim that the PSC problem is also $\mathcal{NP}$-hard.

In the literature, various polynomial-time algorithms have been proposed for MPS that yield near-optimal schedules. Graham [15] proposed the Largest Processing Time (LPT) algorithm, that first sorts the tasks such that $l(T_1) \geq l(T_2) \geq \ldots \geq l(T_n)$ and then assigns the tasks in succession to the minimally loaded processor. LPT has a time complexity of $\mathcal{O}(n \log n + n \log m)$. Graham proved that the worst-case performance ratio is $\frac{4}{3} - \frac{1}{3m}$. Algorithm 1 gives the pseudo-code of LPT, expressed in the variables of PSC.

Algorithm 1 [LPT]

(assumes $m < y$)
  sort $\mathcal{S}$ such that $l(S_1) \geq l(S_2) \geq \ldots \geq l(S_y)$;
  for $i := 1$ **to** m **do** $P_i := S_i$ **od**;
  for $i := m + 1$ **to** y
  do select $k \in \{j \mid l(P_j) = min_{1 \leq x \leq m} l(P_x)\}$;
    $P_k := P_k \cup \{S_i\}$;
  od;
  return $max_{1 \leq x \leq m} l(P_x)$;

The Bin Packing problem can be seen as the dual version of the Bin Design (= MSP) problem. In Bin Design, a fixed num-

ber of bins is given, for which the minimum capacity needed to pack a set of given items has to be determined. In Bin Packing, the capacity of the bins is fixed, and the number of bins needed to pack all items has to be determined. An alternative approach to solve MPS is to utilize a bin-packing heuristic in conjunction with a search over the bin capacity C to find the minimum capacity such that all n items (tasks) will fit into (onto) the m bins (processors). For each fixed bin capacity, the First Fit Decreasing (FFD) heuristic is used to fit the tasks to the processors. Assume that the tasks have been sorted such that $l(T_1) \geq l(T_2) \geq \ldots \geq l(T_n)$. The FFD heuristic assigns the tasks in succession to the lowest indexed processor which can complete the task within its capacity. Algorithm 2 gives the pseudo-code of FFD, expressed in the variables of PSC.

Algorithm 2 [FFD(C)]

(assumes $\mathcal{S}$ sorted such that $l(S_1) \geq l(S_2) \geq \ldots \geq l(S_y)$
and assumes initially $P_j = \emptyset$ for all j)
  for $i := 1$ **to** y
  do $j := 1$
    while $l(P_j) + l(S_i) > C$
    do $j := j + 1$ **od**;
    $P_j := P_j \cup T_i$;
  od;
  return $max\{j \mid P_j \neq \emptyset\}$;

To decide on the capacity of the processor, the algorithm should use some search method. The MULTIFIT method, proposed by Coffman et al. [13], uses a bisection search over the bin capacity. Starting with known upper and lower bounds on the capacity C, at each step FFD is ran for a value of C midway between the current upper and lower bounds. If FFD(C) $> m$, C becomes the new lower bound; if FFD(C) $\leq m$, C becomes the new upper bound. Initial lower and upper bounds on C are given as $C_L = max\left(\frac{l(\mathcal{T})}{m}, max_{1\leq i\leq n} T_i\right)$ and $C_U = max\left(\frac{2\cdot l(\mathcal{T})}{m}, max_{1\leq i\leq n} T_i\right)$ respectively [16]. Compared to LPT the worst-case performance of MULTIFIT is better, at the expense of additional computation time. The time complexity of MULTIFIT is $\mathcal{O}(n \log n + kn \log m)$, where k denotes the number of iterations in the binary search. Coffman et al. [13] showed that MULTIFIT has a worst-case performance ratio of $1.22 + (\frac{1}{2})^k$. Later, Friesen [17, 16] proved that the worst-case performance ratio is even better, viz. $1.20 + (\frac{1}{2})^k$. Algorithm 3 gives the pseudo-code of MULTIFIT, expressed in the variables of PSC.

Algorithm 3 [MULTIFIT]

(assumes $\mathcal{S}$ sorted such that $l(S_1) \geq l(S_2) \geq \ldots \geq l(S_n)$)
  $C_L := max\left(\frac{l(\mathcal{S})}{m}, l(S_1)\right)$;
  $C_U := max\left(\frac{2\cdot l(\mathcal{S})}{m}, l(S_1)\right)$;
  for $i := 1$ **to** k
  do **if** $C_L \neq C_U$
    then $C := \lfloor \frac{C_L + C_U}{2} \rfloor$;
      if FFD(C) $> m$ **then** $C_U := C$;
        else $C_L := C$;
      fi;
    fi;
  od;
  return $\mathcal{P}$;

A problem with the binary search of MULTIFIT is that it can have the following anomalous behavior. If the tasks do not fit onto m processors with capacity C, they may still fit onto m processors with a capacity smaller than C. Hence, whereas binary search is useful to search quickly in a large search space, at the expense of additional computation time, linear search might obtain better results. Note that in many practical cases of wrapper design, the additional computation time required for a linear search will be acceptable.

Based on a proposal by Lee & Massey [18], who use LPT to obtain an incumbent schedule for MULTIFIT, we use a combination of LPT and LINEARSEARCH to design our wrappers, if the range of C values is of acceptable size. Algorithm 4 gives the pseudo-code of the resulting COMBINE algorithm, expressed in the variables of PSC.

Algorithm 4 [COMBINE] [1]

  $A := \sum_{i=1}^{n} \frac{S_i}{m}$;
  $X :=$ LPT;
  if $X < 1.5 \cdot A$
  then $C_U := X$;
    $C_L := \lfloor max(X/(\frac{4}{3} - \frac{1}{3m}), S_1, A) \rfloor$;
    $i := C_L$;
    while $i \leq C_U \wedge$ FFD(i) $> m$
    do $i := i + 1$ **od**;
  fi;
  return $\mathcal{P}$;

6 Wrapper Design Example

Core A has 38 terminals; eight functional inputs `a[[0:7]`, eleven functional outputs `z[0:10]`, nine core-internal scan chains of lengths resp. 12, 6, 8, 6, 6, 12, 6, 8, and 8 flip flops, and a scan-enable control input `sc`. Its wrapper will be connected to one single-bit TAM as well as to a three-bit wide TAM. Hence, the wrapper requires two TAM plug pairs of respectively one (STP) and three bits (MTP) wide. Furthermore, the wrapper should be equipped with a wrapper bypass, but not with scan

[1] Lee and Massey [18] proved that X is optimal if $X \geq 1.5 \cdot A$. Hence, if in the COMBINE algorithm, LPT already yields this result, LINEARSEARCH does not need to be executed.

chain bypasses.

Terminal Type	Number	Wrapper Cell
Fi	8	wInput
Fo	11	WOutput
Ti	9	wTInput
To	9	wTOutput

Table 2: Required wrapper cells for Core A.

For the PSC problem, the LPT algorithm yields $P_1 = \{12, 8, 6\}$, $P_2 = \{12, 6, 6\}$, and $P_3 = \{8, 8, 6\}$. Hence, the longest scan chain concatenation has 26 bits. The COMBINE algorithm improves this to the optimal partition $P_1' = \{12, 12\}$, $P_2' = \{8, 8, 8\}$, and $P_3' = \{6, 6, 6, 6\}$ with a maximum length of 24 bits.

Table 3 shows an optimal ordering and partitioning of TAM items for Core A. Figure 6 shows the corresponding wrapper for Core A, including the optional wrapper bypass.

TAM Wire	TAM Input Items	TAM Output Items	
TAM[0]	{1, 1, 1}	{12, 12}	{1, 1, 1, 1}
TAM[1]	{1, 1, 1}	{8, 8, 8}	{1, 1, 1, 1}
TAM[2]	{1, 1}	{6, 6, 6, 6}	{1, 1, 1}

Table 3: Optimized ordering and partitioning of TAM items for Core A.

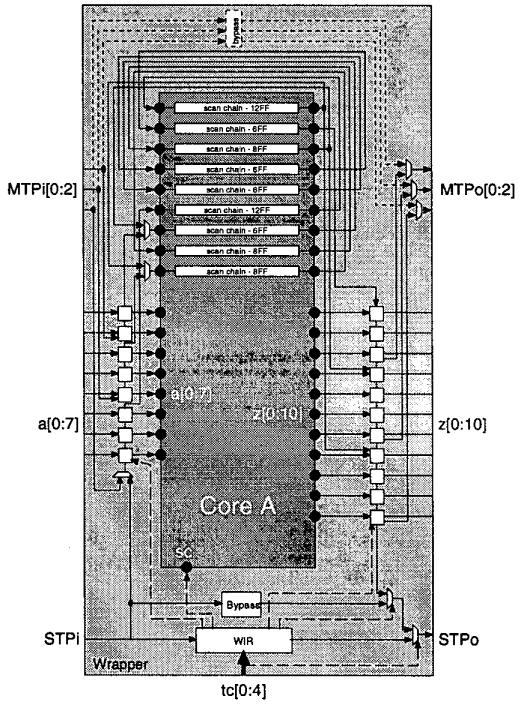

Figure 6: Core A with wrapper.

7 Wrapper Generation and Verification

In order to obtain plug-n-play interoperability between cores from various sources, core test wrappers should be standardized. On the other hand, cores differ in their test access requirements and system chip designs cannot all offer the same access facilities. Therefore, any wrapper standard needs certain scalability and flexibility. Examples of such standards are Philips' TestShell [3] and P1500's wrapper [7]. These standards typically do not define the implementation of a wrapper. Rather, they define the behavior that is required for interoperability, leaving the actual implementation to the wrapper designer.

As any DfT-insertion task, wrapper generation is suited for automation. An automated wrapper generator that is built on top of a wrapper standard, should guarantee that the resulting wrapper indeed conforms to the standard. A *smart* wrapper generator is able to take advantage of the implementation of the core itself, if available. In order to reduce the silicon area costs and performance impact of the wrapper, one would like to reuse the resources that are available at the core boundary and that can be of help. For example, a scan flip flop immediately after (before) a core input (output) terminal can be reused as wrapper input (output) cell, and therefore make that an additional dedicated-wrapper cell for those terminals is not necessary.

Next to automated wrapper generation, there is also a need for automated wrapper verification. A wrapper verifier might be used for incoming inspection of cores that already come with wrappers. Furthermore, a wrapper verifier can also play a role in automatic wrapper generation. The report on missing wrapper functionality can serve as the specification for what needs to be added by the wrapper generator.

Within Philips, we have created a wrapper generator named TESTSHELLGEN and a wrapper verifier named CHECKCORE. The wrapper standard for which these tools work is encoded in a rule set file. The current rule set supports the Philips TestShell; once IEEE P1500 becomes adopted, the same tools should be able to support this standard by means of an alternative rule set. The two tools cooperate together as described above for smart wrapper generation. Together, they offer various use scenarios.

1. Wrapper design for a core of which the gate-level netlist is available is done by first executing CHECKCORE, followed by TESTSHELLGEN. The verifier finds out to which extent the wrapper functionality is already existent in the core itself. Based on the report which functionality at which terminal is still missing in order to be compliant

to the wrapper standard, TESTSHELLGEN adds only the required hardware.

2. If the implementation details of a core are not available, wrapper design is done by executing TESTSHELLGEN only. In this case, we cannot take advantage of hardware inside the core, but the core can really be treated as a black box.

3. A wrapper designer can use CHECKCORE to check compliance of a wrapper that was designed either manually or automatically by one of the above scenarios.

4. The same compliance check can be done as 'incoming inspection' by the receiver of a core that is claimed to be compliant.

8 Conclusion

Standardized, but scalable core test wrappers play an important role in the test interoperability of embedded cores from distinct sources. We have described a wrapper architecture that unites the features of Philips' TestShell and P1500's wrapper. It provides facilities for functional access, as well as access for core-internal and core-external testing. The wrapper consists of wrapper cells and a Wrapper Instruction Register (WIR). We have presented the access requirements of the various types of core terminals. A simple library of wrapper cells is sufficient to meet the access requirements and can be extended to offer additional functionality. The interconnections of wrapper cells and core-internal scan chains determine the test time of the core. We described how ordering these TAM chain items, together with the use of several optional bypasses, can reduce the test time. We showed that the partitioning of TAM items over TAM chains such that test time is minimized is equivalent to the $\mathcal{NP}$-hard problem of Multi-Processor Scheduling, and described a heuristic algorithm for this problem. The results of our wrapper design approach are illustrated by means of an example. Finally, we have indicated how wrapper generation and verification can be automated and how a wrapper verifier can be used to minimize the area and performance impact in wrapper design.

Acknowledgements

Part of the work in this paper was done in Sandeep Goel's Master of Technology graduation project in VLSI Design Tools & Technology at the Indian Institute of Technology in Delhi, India. This project was entitled 'Test Access Planning for Embedded-Core-Based System ICs' and was partly carried out at Philips Research in Eindhoven, The Netherlands.

The authors acknowledge the valuable contributions of C.P. Ravi Kumar of the Electrical Engineering Department of IIT Delhi and our Philips colleagues Jan Korst, Gert Jan van Rootselaar, and Harald Vranken. Furthermore we thank the members of the Philips' Core Test Action Group (CTAG) and the IEEE P1500 SECT Working Group for many stimulating discussions on this topic.

References

[1] Yervant Zorian, Erik Jan Marinissen, and Sujit Dey. Testing Embedded-Core Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 130–143, Washington, DC, October 1998. IEEE Computer Society Press.

[2] Erik Jan Marinissen and Yervant Zorian. Challenges in Testing Core-Based System ICs. *IEEE Communications Magazine*, 37(6):104–109, June 1999.

[3] Erik Jan Marinissen, Robert Arendsen, Gerard Bos, Hans Dingemanse, Maurice Lousberg, and Clemens Wouters. A Structured And Scalable Mechanism for Test Access to Embedded Reusable Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 284–293, Washington, DC, October 1998. IEEE Computer Society Press.

[4] Prab Varma and Sandeep Bhatia. A Structured Test Re-Use Methodology for Core-Based System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 294–302, Washington, DC, October 1998. IEEE Computer Society Press.

[5] Lee Whetsel. Addressable Test Ports: An Approach to Testing Embedded Cores. In *Proceedings IEEE International Test Conference (ITC)*, pages 1055–1064, Atlantic City, NJ, September 1999. IEEE Computer Society Press.

[6] IEEE P1500 Web Site. http://grouper.ieee.org/groups/1500/.

[7] Erik Jan Marinissen, Yervant Zorian, Rohit Kapur, Tony Taylor, and Lee Whetsel. Towards a Standard for Embedded Core Test: An Example. In *Proceedings IEEE International Test Conference (ITC)*, pages 616–627, Atlantic City, NJ, September 1999. IEEE Computer Society Press.

[8] Robert Arendsen and Maurice Lousberg. Core Based Test for a System on Chip Architecture Framework. In *Digest of Papers of IEEE International Workshop on Testing Embedded Core-Based Systems (TECS)*, pages 5.1–1–8, Washington, DC, October 1998.

[9] Gert Jan van Rootselaar and Bart Vermeulen. Silicon Debug: Scan Chains Alone Are Not Enough. In *Proceedings IEEE International Test Conference (ITC)*, pages 892–902, Atlantic City, NJ, September 1999. IEEE Computer Society Press.

[10] Erik Jan Marinissen and Maurice Lousberg. The Role of Test Protocols in Testing Embedded-Core-Based System ICs. In *Proceedings IEEE European Test Workshop (ETW)*, pages 70–75, Konstanz, Germany, May 1999. IEEE Computer Society Press.

[11] Joep Aerts and Erik Jan Marinissen. Scan Chain Design for Test Time Reduction in Core-Based ICs. In *Proceedings IEEE International Test Conference (ITC)*, pages 448–457, Washington, DC, October 1998. IEEE Computer Society Press.

[12] Sandeep Kumar Goel. Test Access Planning for Embedded Core-Based System ICs. Master's thesis, Indian Institute of Technology Delhi, New Delhi, India, December 1999.

[13] E.G. Coffman Jr., M.R. Garey, and D.S. Johnson. An Application of Bin-Packing to Multiprocessor Scheduling. *SIAM Journal of Computing*, 7(1):1–17, February 1978.

[14] M.R. Garey and D.S. Johnson. *Computers and Intractability - A guide to the theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[15] R.L. Graham. Bounds on Multiprocessing Anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429, 1969.

[16] Donald K. Friesen and M.A. Langston. Evaluation of a MULTIFIT-Based Scheduling Algorithm. *Journal of Algorithms*, 7:35–59, 1986.

[17] Donald K. Friesen. Tighter Bounds for the Multifit Processor Scheduling Algorithm. *SIAM Journal of Computing*, 13(1):170–181, February 1984.

[18] Chung-Yee Lee and J. David Massey. Multiprocessor Scheduling: Combining LPT and MULTIFIT. *Discrete Applied Mathematics*, 20:233–242, 1988.