Chapter 2

Literature Survey

Continuous advancement in semiconductor manufacturing technology has facilitated the integration of tens or even hundreds of millions of transistors onto silicon die [1]. The need for faster and smaller products has driven the semiconductor technology recently to introduce a new generation of complex chips. Chips that allow the integration of a wide range of complex functions, which used to comprise a system, into a single die, called System-On-Chip(SOC).

System-level integration is evolving as a new paradigm in system design, allowing an entire system to be built on a single chip, using pre-designed functional blocks called cores. While SOC is proving to be very useful in meeting aggressive time-to-market, performance, and cost requirements of today's electronic products, testing such core-based SOCs poses a two-fold challenge [2]. Core-level testing involves making each core testable, inserting the necessary design for testability (DFT) structures and generating test sequences. Typically, for hard and firm cores, testability is ensured, and pre-computed test sets are provided by the core provider. For soft cores, testability can be addressed and test sets be generated by the user. When the cores are integrated into an SOC, chip-level testing needs to be addressed by the SOC designer. The main difficulty in chip-level testing is the problem of justifying pre-computed test sequences of a core, embedded deep in the design from the chip inputs, and propagating the test responses from the core outputs to the chip outputs, that is, the cores integrated in an SOC need to be tested after the manufacturing process of the device [3]. So testing SOC is more complex than conventional board test.

Today's embedded cores incorporated into system-chips cover a very wide range of functions, while utilizing an unprecedented range of technologies,

from logic to DRAM to analog [4]. They often come in hierarchical compositions. For instance, a complex core may embed one or more simple cores. An example system-chip design is shown in Fig. 2.1, where different cores are shown. These cores typically come in a wide range of hardware description levels. They spread from fully optimized layouts in GDSII format to widely flexible RTL codes. Embedded cores are categorized into three major types based on their hardware description level: *soft*, *firm* and *hard* [5]. Each type of core has different modeling and test requirements. The three types of cores offer trade-off. *Soft* cores leave much of the implementation to the designer, but are flexible and process-independent. *Hard* cores have been optimized for predictable performance, but lack flexibility. *Firm* cores offer a compromise between the two.

The emerging process of plug-and-play with embedded cores from diverse sources faces numerous challenges in the areas of system-on-chip design, integration, and test.

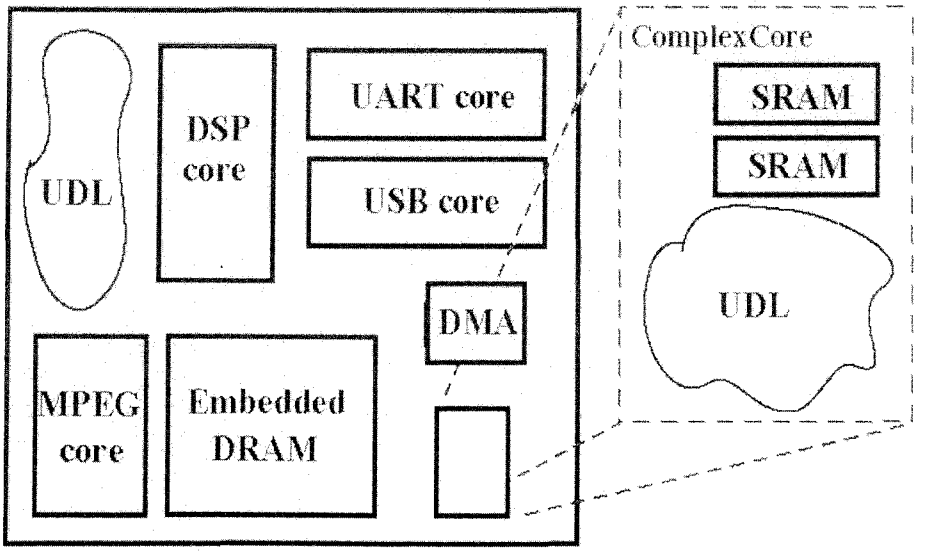


Figure 2.1: System-Chip consisting of hierarchical cores and User Defined Logic (UDL)

2.1 System Chip Test Challenges

In this section, the main challenges of testing system chips are analyzed and compared to the traditional chip test approach. Even though the design process in core-based system chips is conceptually analogous to the one used in traditional chip design, the manufacturing test processes in both cases are quite different [3]. In the traditional system-on-board approach, chip manufacturing and testing are performed by the component provider, prior to PCB assembly and test done by the system integrator, as shown in Fig. 2.2(a). On the other hand, in a core-based system chip, the reusable cores are first designed individually by the component provider. Next, the system integrator designs the *User Defined Logic (UDL)* and assembles the pre-designed cores. Only then, the manufacturing and test steps are performed. This is done for the whole system chip, as shown in Fig. 2.2(b).

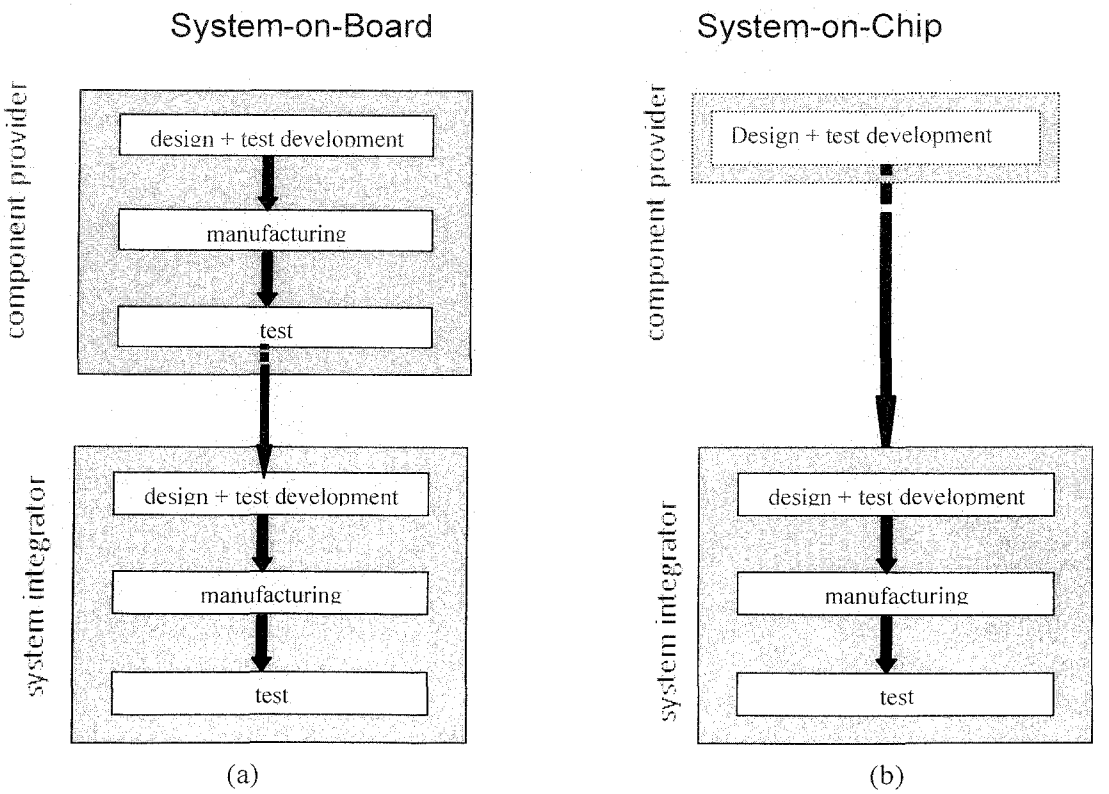


Figure2.2: System-on-Board (a) vs. System-on-Chip (b) trajectory.

2.1.1 Core Level Test

A core is typically the hardware description of today's standard ICs, e.g., DSP, RISC processor, DRAM core, etc. Even though a given core is not tested individually as in standard ICs, and instead is tested as a part of the overall system chip by the system integrator, the preparation of a core internal test, i.e. the test development, is often done by the core provider, as in Fig. 2.2(b). Because, the system integrator in most cases, except for soft cores, has very limited knowledge about the structural content of the adopted core, and hence deals with it as a black box. Therefore, he/she cannot develop the necessary test for it. That is why the core vendor has to provide the model, the DFT (e.g., scan based DFT or built-in self test (BIST) based DFT) [9, 10, 11] structures and the corresponding test vectors. This is especially true if a core is a hard one or is an encrypted Intellectual Property (IP) block. This necessitates that the core provider develops the core test, i.e., the DFT structures and the corresponding test patterns, and delivers it with the core. Another function of core provider is to determine the internal test requirements of the core without knowing the system chip environment and possibly even the target process. For instance, which test method needs to be adopted (e.g., BIST, scan, IDDQ, functional test), what type of fault(s) (e.g., static, dynamic, parametric) to target and what level of fault coverage is desired. In the traditional approach, the overall chip test method and the desired fault coverage are predetermined. Hence, the designer incorporates the requirements during test development. But in system chips, a core provider often does not have enough information about the target applications of his component and their quality requirements. Hence, the provided quality level might or might not be adequate. If the coverage is too low, the quality level of the system chip is put to a risk, and if it is too high, the test cost may become prohibitive (e.g., test time, performance, area, power). Furthermore, different processes have different defect densities and distributions.

The core internal test developed by a core provider needs to be adequately described, ported and should be ready for plug-and-play, i.e., for interoperability, with the system chip test. For an internal test to accompany its corresponding core and be interoperable, it needs to be described in a commonly accepted, i.e., standard, format. Such a standard format has been proposed in the IEEE P1500 [48] and is referred to as standardization of a Core Test description Language (CTL) [49].

2.1.2 Test Access

Another key difference between the traditional approaches and the ones for system chip is the accessibility to component peripheries, i.e., accessing primary input/outputs of chips and cores, respectively. With a system-on-board, direct physical access to chip peripheries, i.e., pins, is typically available for probing during manufacturing test; whereas for cores, which are often deeply embedded in a system chip, direct physical access to its peripheries is not available by default, hence, an electronic access mechanism is needed. This access mechanism requires additional logic, such as a *wrapper* around the core and wiring, such as a test access mechanism to connect core peripheries to the test sources and sinks, defined in the next section. The wrapper performs switching between normal mode and the test mode and the wiring is meant to connect the wrapper surrounding the core to the test source and sink. The wrapper can also be utilized for core isolation. Typically, a core needs to be isolated from its surroundings in certain test modes. Core isolation is often required on the input side, the output side, or both. If it is on the input side, it can put the core into a safe-state by protecting the core under test from external interference (from preceding cores or UDL). On the output side, it protects the inputs of the superseding blocks (cores or UDL) from undesired values (e.g., random patterns applied to tri-state buffers creating bus conflicts).

2.1.3 System Chip Level Test

One of the major challenges in the system chip realization process is the integration and coordination of the on-chip test and diagnosis capabilities. Compared to the conventional scheme, the system chip test requirements are far more complex than the PCB assembly test, which for instance in digital chips consists of interconnect and pin toggling tests. The system chip test is a single composite test. This test is comprised of the individual tests of each core, the UDL test, and the test of their interconnects. As discussed earlier, each individual core or UDL test may involve surrounding components. Certain peripheral constraints (e.g., safe mode, low power mode, bypass mode) are often required. This necessitates access and isolation modes. In addition to the test integration and interdependence issues, the system chip composite test requires adequate test scheduling. This is needed to meet a number of chip-level requirements, such as total test time, power dissipation [6], area overhead, etc. [7]. Also, test scheduling is necessary to run intra-core and inter-core tests [8] in certain order not to impact the initialization and final contents of individual cores. With the above scheduling constraints the schedule of the composite system chip test is created.

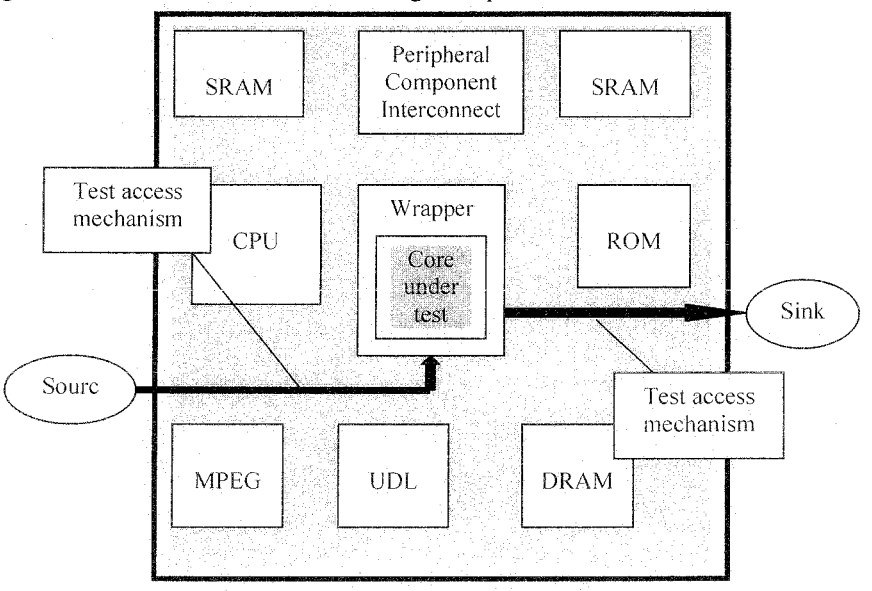
In addition to the above differences between testing traditional chips and system chips, we have to note that system chips do also have the typical testing challenges of the very deep-submicron chips, such as defect/fault coverage, overall test cost, and time-to-market.

Driven by the above challenges a vast body of research has endeavored to provide a better understanding of this research area. Numerous test strategies and algorithms in SOC test architecture design and optimization, test scheduling and test resource partitioning have been proposed in order to reduce test cost of the SOC. Iyenger *et al.* [12] presented an overview of modular SOC test planning techniques that addresses the problems of test architecture design and optimization and constrained test scheduling algorithm.

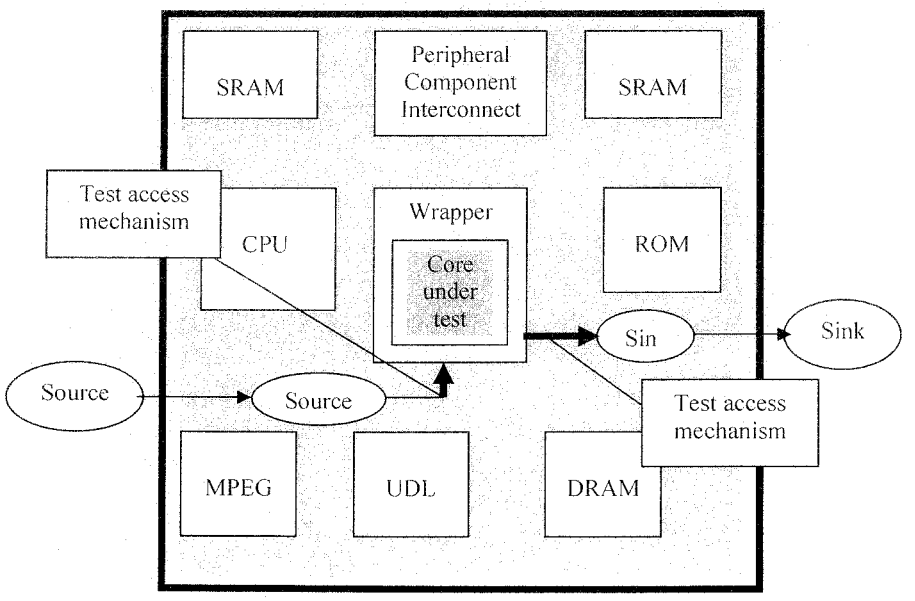
2.2 SOC Test Access

Conceptual Architecture for Core Test

Zorian *et al.* [3] proposed a conceptual infrastructure for SOC test, as shown in Fig. 2.3. It consists of the following components.



(a)



(b)

Figure 2.3. This architecture applies to test patterns generated by automatic test equipment (ATE) from (a) outside the chip. It also applies to built-in self-test (BIST) configurations, which have source and sink (b) inside the chip.

- *Test pattern source and sink.* The source generates the test stimuli for the embedded core, and the sink compares the response(s) to the expected values.
- *Test access mechanism.* It transports test patterns. It provides on-chip transport of test stimuli from a test pattern source to the core under test. It also transports test responses from the core under test to a test pattern sink.
- *Core test wrapper.* The wrapper forms the interface between the embedded core and its environment. It connects the terminals of the embedded core to the rest of the IC and to the test access mechanism.

All three elements can be implemented in various ways, such that a whole palette of possible approaches to testing embedded cores emerges. We review the various alternatives and classify the current approaches.

Test Pattern Source and Sink

Designers can implement test pattern sources and sinks either off-chip, using external automatic test equipment (ATE), or on-chip, using built-in-self-test (BIST), or a combination of both or even an embedded microprocessor [19]. Source and sink need not be of the same type; that is, an embedded core's source can be off-chip, while its sink is on-chip. Three factors influence the choice of a certain type of source or sink:

- The type of circuitry in the core,
- The predefined tests that come with the core,
- Quality, test time, and cost considerations.

Core circuitry

Today, system chips use three main types of circuitry: digital logic, memory, and analog. Simple cores consist of only one type; complex cores combine multiple simple cores, possibly of different circuitry types.

These three types of circuitry exhibit different defect behavior and require different tests [13]. The various tests require different types of sources to

generate the stimuli and sinks to compare the responses. Typically, distinct ATE systems as well as BIST schemes are used for logic, memory, and analog circuitry. System chips, which often incorporate all three types of circuitry into one IC, are encouraging ATE vendors and BIST providers to integrate their traditionally separate solutions for logic, memory, and analog into combined product offerings. Hence, instead of using a separate ATE for the logic part of the system chip, a second ATE for the embedded memory and a third for the analog circuitry, ATE vendors are offering "super" ATE systems to combine the test capabilities of all three types.

Core tests

The variety of core tests is much larger than the three circuitry types. Tests are classified not only by the type of circuit they test, but also by the measurements they require (voltage or current), by the way they are generated (based on the IC's function or structure), by the amount of core-internal adaptation they require (scan or test points), and so on.

Examples of current measurement tests are IDDQ and DDT [14], which measure quiescent and transient currents. Current can be measured both by sinks on-chip (current monitors) [16] as well as off-chip (current monitors in an ATE system).

Not all test patterns can be generated on-chip in a cost-effective manner. The test patterns of cores that come with function tests and/or ATPG-generated tests are often irregular in structure. It is difficult to generate such irregular deterministic test patterns on-chip at acceptable area costs.

Quality and cost

Off-chip sources and sinks often require large capital investment and, because they are built using yesterday's technology, suffer from various problems. Faster ICs require increasing accuracy to detect timing signals at the IC pins. Although tester accuracy has improved by 12% annually, IC speeds have improved by about 30% per year. This growing gap reduces off-chip testers'

ability to properly identify bad chips, leading to yield losses and cost increases. Furthermore, it is becoming increasingly difficult for off-chip ATE to keep up with the very high frequencies needed to sufficiently test performance-related defects in today's ICs.

In addition to these test-quality-related issues, the increased pin count and the mixing of diverse technologies in system chips will cause ATE costs - for running the tests as well as the equipment itself - to rise toward $20 million, according to the *1997 SIA National Technology Roadmap for Semiconductors [16]*. These problems with the quality and cost of external ATE will only become worse for high-speed, high-density, and mixed-technology system chips, rendering external ATE unacceptably inaccurate and prohibitively expensive.

Test Access Mechanism

A test access mechanism takes care of on-chip test pattern support. It can be used to transport

- test stimuli from the test pattern source to the core under test, and
- test responses from the core under test to the test pattern sink.

By definition, the test access mechanism is implemented on-chip. Although one core often uses the same type of test access mechanism for transporting both stimulus and response, such consistency is not required and various combinations may coexist.

Designing a test access mechanism involves making trade-offs between the mechanism's transport capacity (bandwidth) and its test application cost. Bandwidth is limited by the bandwidth of source and sink and the silicon area that we want to spend on the test access mechanism itself. A wider test access mechanism provides more bandwidth, but consumes more wiring area. For example, if the test pattern source is an external ATE, it does not make much sense to provide a mechanism wider than there are IC pins available to connect it to. In this case, the IC pins are the bandwidth bottleneck, and a wide

mechanism costs more silicon area without adding to that bandwidth.

Test time is a result of the test data volume of the individual cores and the bandwidth of the test access mechanism. How expensive test time is per unit of time depends on the type of source and sink. There is a wide range of external ATE with similarly wide-ranging associated test costs, and these again differ from the cost of test application time for BIST.

There are several options for implementing a core test access mechanism, which can

- reuse existing functionality to transport test patterns or be formed by dedicated test access hardware;
- go through other modules on the IC - including other cores - or pass around those other modules;
- provide access for only one core or for multiple cores; or
- be a plain signal transport medium or may contain certain intelligent-test-control functions.

Core Test Wrapper

The core test wrapper is the interface between the embedded core and its system chip environment. It connects the core terminals both to the rest of the IC as well as to the test access mechanism. By definition, the core test wrapper is implemented on-chip and should have the following mandatory modes:

- *Normal operation* (non test). In this mode, the core is connected to its system IC environment, and the wrapper is transparent.
- *Core-internal test*. The test access mechanism is connected to the core such that a source can apply stimuli at the core's inputs, and a sink can observe responses at the core's outputs.
- *Core-external test*. The test access mechanism is connected to the interconnect wiring and logic such that a source can apply stimuli at the core's outputs, and a sink can observe responses at the core's inputs.

Apart from these mandatory modes, a core test wrapper can also have several optional modes. For example, a wrapper can include a detach mode to disconnect the core from its system chip environment and the test access mechanism.

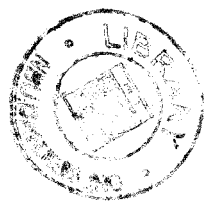
Depending on the test access mechanism's implementation, some of these modes can coincide. For example, if the test access mechanism uses existing functionality, normal and core test modes can coincide.

Predesigned cores have their own internal clock distribution system. Different cores have different clock propagation delays, which might result in clock skew for intercore communication. The system-IC designer should take care of this clock skew in the functional communication between cores. However, clock skew might also corrupt the data transfer over the test access mechanism, especially if multiple cores share this mechanism. The core test wrapper is the best place in the test access paths between cores to implement clock skew prevention.

The *test collar* [17] and the *test shell* [18] are examples of core test wrappers. Details of wrapper design have been given in Section 2.6.

2.3 Testing Strategies

The test access mechanism takes care of on-chip test pattern transport. It can be used (1) to transport test stimuli from the test pattern source to the core-under-test, and (2) to transport test responses from the core-under-test to the test pattern sink [6]. The test access mechanism is, by definition, implemented on-chip. In this section an overview of the various testing strategies are discussed. The embedded cores can be accessed from the chip's I/O pins in four different ways [3, 20]: (i) direct parallel access via chip inputs; (ii) serial access and core isolation through a boundary scan-like architecture (called isolation ring access mechanism); (iii) functional access through functional busses or transparency



of embedded cores; and (iv) access through a combination of core wrappers and dedicated test busses.

2.3.1 Direct Access

An obvious mechanism to make embedded cores testable from the IC pins makes the core-under-test directly and parallely accessible from the IC pins. In this mode, no previously existent connection is reused. A direct connection between the CUT interface and the system interface is established.

This approach is commonly practiced for embedded memories, but also many block based ASICs use this test access strategy [21]. Additional wires are connected to the core's terminals and multiplexed onto existing IC pins. The multiplexer control is coded as a dedicated test mode. In case of multiple embedded cores, this leads to an equal number of test modes. Fig. 2.4 gives a schematic view of the approach.

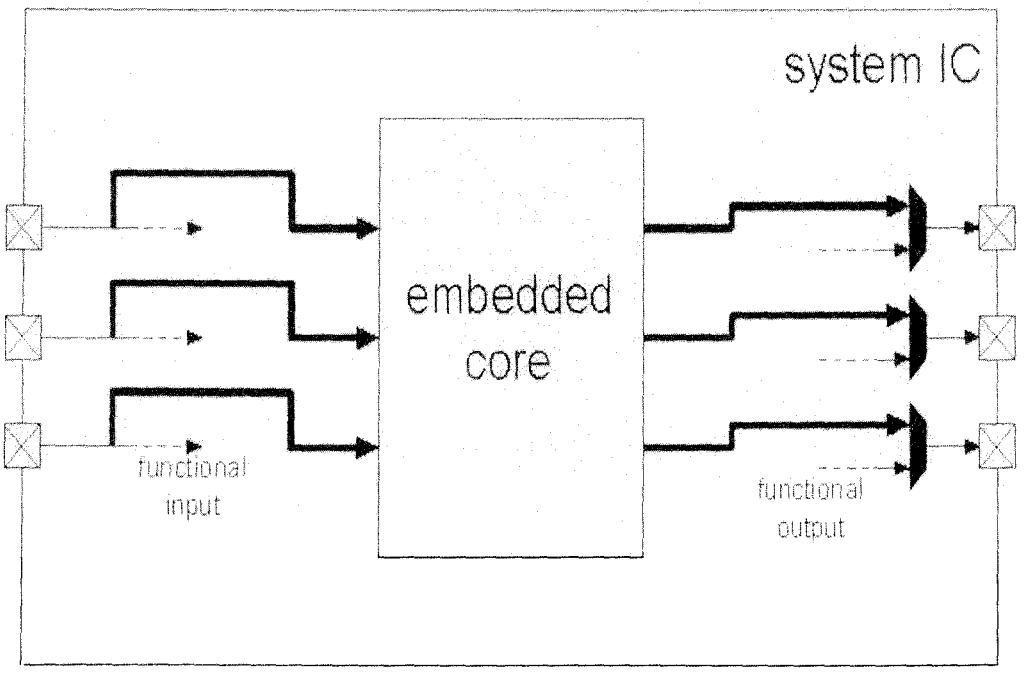


Figure 2.4: Multiplexed direct parallel access.

The advantage of this approach is in its simplicity and in testing the core as if it is the only circuit on the IC *i.e.* the embedded core can be tested as if it were a stand-alone device. This also simplifies silicon debug and diagnosis. The disadvantage of this technique is that it is not very scalable. If a system IC contains many embedded cores, this approach leads to high area costs for connecting and multiplexing all core terminals to IC pins, and the control circuitry for these multiplexers will grow more complex. The bit width of this connection is assumed to be as large as the number of bits that need to be propagated to the interface. This implies an overhead of n in the number of pins, for n the number of test signals being propagated. The area overhead is proportional to n and to the routing distance from CUT to the system interface.
The test access mechanism as proposed by Varma & Bhatia [22, 23] also accesses the core-under-test directly from the IC pins, but provides the option to share one access mechanism with multiple cores. Such a shared access structure is called a *test bus*. Per IC, there are one or more test buses of varying width. Multiple cores can be connected to one tri-stateable test bus; all cores on the same bus are tri-stated, except for the core-under-test. This approach allows the system IC integrator to choose his optimal mix of number and width of test buses and number of cores per test bus and hence provides the opportunity to trade off silicon area for test time. Although presented as a dedicated test access mechanism, it is relatively easy in this approach to reuse existing on-chip buses as test bus implementation, as is done in ARM's AMBA system (Fig. 2.5) [24]. The main disadvantage of this approach is that per test bus, only one core can be connected at a time. This limits the possibilities to test multiple cores at the same time, which is sometimes desired (test time reduction through test scheduling, IDDQ testing) or even required (testing interconnect wiring and glue logic in between cores).

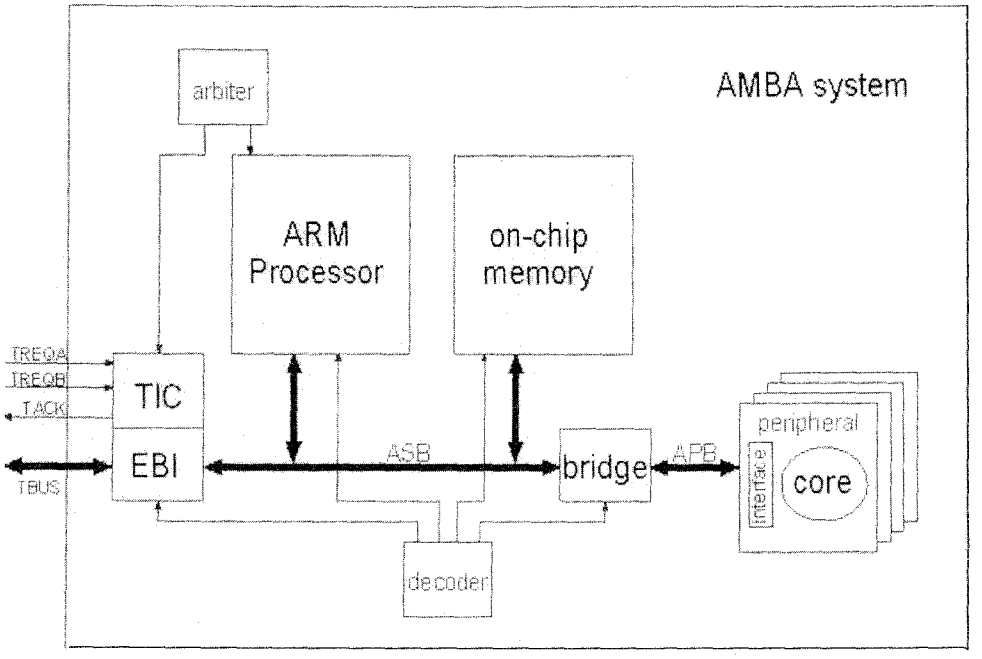


Figure 2.5:ARM's AMBA system

2.3.2 Isolation ring access (Use of Boundary-scan)

The Boundary Scan Test (BST) approach was initially a DFT technique for printed circuit boards (PCBs). This technique requires that ICs include extra hardware to facilitate communication between the board and the various ICs on which they are mounted during testing. Also with the present trend of a system-on-chip, the ICs themselves consist of several embedded complex and diverse modules and start to beg for a DFT technique that facilitates their testing. In 1990, the methodology became IEEE/ANSI Standard 1149.1 [25].

The use of Boundary Scan has been extended beyond boards to circuits [26]. For example, it is useful for internal testing and to run BIST. In addition to testing, Boundary Scan was adopted in debugging and diagnosis using in-circuit emulation protocols [27].

In this approach, a serial scan chain is laid around the terminals of the core. Through this scan chain both the core itself, as well as the interconnect wiring and logic in between cores can be tested. The obvious benefit of this approach is that an existing test standard is reused. Many ICs are equipped with BST, often augmented with multiple private instructions which represent all kinds of test, debug, and emulation modes. If these ICs become cores in large system ICs, it seems logical to reuse the test infrastructure as provided by BST. A technical problem is that a scheme has to be developed to control multiple TAP controllers on one IC; various solution approaches have been proposed [28, 29]. A more serious drawback of this approach is that the BST standard defines a combined test control and test data access path of only one bit wide via TDI and TDO. Therefore, the BST approach does not allow to make a trade-off between bandwidth and test time. For small test pattern sets, the small bandwidth does not pose a problem. This is, for example, the case for board-level interconnect testing, for which BST was developed, or for cores equipped with BIST. However, for large scan-testable cores, providing many scan test patterns via only one serial access wire results in prohibitively long test times.

Touba & Pouya [30] have presented a variation on the Boundary Scan Test approach by using a partial boundary scan ring around the core. The ATPG techniques are used to find out which of a given set of stimuli can be justified from the (assumed) user-defined logic (UDL) in front of the core, such that the corresponding core inputs can be excluded from the partial boundary scan ring. In [31] the same authors even allow modification of the UDL such that the required test patterns can be generated. In addition to its function as test data and control signal transportation medium, the Test Access Mechanism presented in Zorian [7] has an embedded intelligence to execute the system chip test in a predetermined schedule. The test sequencing of all the cores is embedded in a network of distributed modular controllers, which is an integral part of the Test Access Mechanism. This network called, the Universal BIST Scheduler, is either customized to execute a predetermined BIST schedule for

manufacturing test or is programmed on the fly through the JTAG port to execute the test of individual cores during silicon debug and diagnosis.

2.3.3 Functional Access

The third technique for peripheral access is based on using the surrounding logic to propagate and justify the necessary test patterns [33]. This can be either based on existing normal mode logic put in transparency for core test purposes [32], or be based on using existing DFT mode, such as scan chains in the surrounding logic. Even though, the cost of extra hardware is very low, this approach becomes complicated if the surrounding logic has deep functional paths and requires sophisticated protocols for test pattern translation.

The timing impact of a peripheral access mechanism is critical. The path delay created by peripheral access logic and the skews introduced in the access path need to meet the requirements of core test timing constraints.

The ring based technique remains the most realistic option for peripheral access. The ring approach provides access to run internal BIST and internal scan for each core. It also helps testing the surrounding logic using the ring registers.

Ghosh *et. al.* [34] assumed that every core has a transparent mode in which data can be propagated from inputs to outputs in a fixed number of cycles. Their approach is based on the concept of finding functional test path (F-path) [35] through test control /data flow extraction. Although the proposed method successfully lowered the test area and delay overheads, it requires functional description of each core to make it transparent, which in most cases is not available or it is hard to extract. In addition, because test data cannot be pipelined through a core in this method, the potentially large transparency latency of each core (number of cycles needed to propagate test data from inputs to outputs of the core) may incur a relatively large test application time. To address the above issue, in [2] the same authors extended their approach by describing a trade-off between the silicon area consumed by the design-for-

transparency hardware and propagation latency. They suggested that the core providers offer a catalogue of area/latency versions for their cores, from which the system integrator can select one in order to meet the test access needs of its neighboring cores. By associating a user-defined cost function with the transparent test paths in the core connectivity graph, the system integrator is able to select the version of cores that achieve an optimized test solution at the system level. This approach, however, requires a large design effort at the core provider side. Another limitation of [34,2] stems from the difficulty in handling other popular DFT schemes, such as scan or BIST, in the SOC.

Solutions from [34,2] require that all the I/Os of a core be simultaneously, though indirectly, controllable/observable. Ravi *et. al.* [36] showed that this complete controllability and observability is unnecessary and proposed to provide them on an "as needed basis". Their approach allows for complex core transparency modes and is able to transfer test data to and from the CUT in a more aggressive and effective manner.

In [37], Nourani and Papachristou presented a similar technique to core transparency, in which cores are equipped with a bypass mode using multiplexers and registers. They can model the system as a directed weighted graph, in which the accessibility of the core input and output ports is solved as a shortest path problem. While this approach eases the problem of finding paths from the SOC inputs to the CUT, it requires packetisation of test data (to match the bit width of input and output ports), and the help of serialization/de-serialisation bit-matching circuits.

2.3.4 Test Rail

A single test rail provides access to one or more cores and an IC may contain one or more test rails of varying width. Each core is wrapped in a test shell. Per core, there is also a test rail bypass mode. This allows the core user to test each core sequentially or multiple cores in parallel. A test rail provides flexible and scalable test access mechanism and helps to trade-off between test time and area overhead by varying test data width. Fig. 2.6 shows typical test rail

architecture. In the figure there are 5 cores which are embedded in their wrappers and test buses of total width 16. Total bus width is divided into three buses of widths 5, 5 and 6 respectively. Cores 1 and 2 are assigned to first bus, cores 3 and 4 are assigned to second bus and core 5 is assigned to third bus. The testing time for a SOC is determined to a large extent by the design of test wrappers and the TAM.

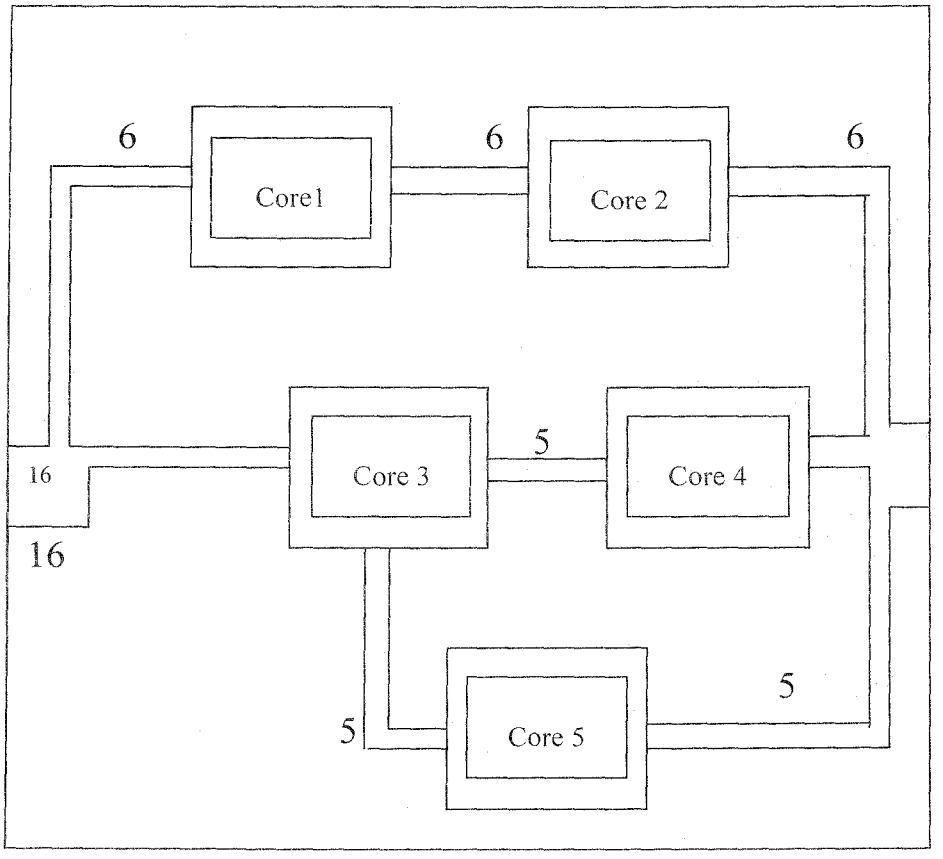


Figure 2.6: Test Rail example

Wrapper/TAM co-optimization is therefore necessary for minimizing SoC testing time. Iyengar *et al.* in [34, 35] proposed an exact technique for co-optimization based on a combination of integer linear programming (ILP) and exhaustive enumeration. However, this approach is computationally expensive for large SOCs, and it is limited to fixed-width test buses. Rectangle packing, also referred to as two-dimensional packing [5] for wrapper/TAM co-

optimization decreases the testing time by reducing the mismatch between a cores test data needs and the width of the TAM to which it is assigned. One of the more recent works on the reduction of test cost for System-on-Chip uses virtual TAMs to match high speed ATE channels to slower scan chains [4]. This method also presented a new TAM optimization framework based on Lagrange multipliers.

2.4 Test Scheduling and Test Architecture Optimization

Test access mechanisms (TAMs) and test wrappers have been proposed as important components of SOC test access architecture [38]. TAMs deliver pre-computed test sequences to cores on the SOC, while test wrappers translate these test sequences into patterns that can be applied directly to the cores. Test wrapper and TAM design are of critical importance in SOC system integration since these directly impact the vector memory depth required on the ATE, as well as testing time, and thereby affect test cost. A TAM and wrapper design that minimizes the idle time spent by TAMs and wrappers during test directly reduces the number of don't-care bits in vectors stored on the tester, thereby reducing vector memory depth. The design of efficient test access architectures has become an important focus of research in core test integration [40, 41, 42, 43, 44, 45, 46, 47]. This is especially timely and relevant, since the proposed IEEE P1500 standard provides a lot of freedom in optimizing its standardized, but scalable wrapper, and leaves TAM optimization entirely to the system integrator [48, 49].

The general problem of SOC test integration includes the design of TAM architectures, optimization of core wrappers, and test scheduling. The goal is to minimize the testing time, area costs, and power consumption during testing. The wrapper/TAM co-optimization problem addressed by Iyengar *et. al.* in [50] is as follows. Given the test set parameters for the cores on the SOC, as well as the total TAM width, determine an optimal number of TAMs for the SOC, an

optimal partition of the total TAM width among the TAMs, an optimal assignment of cores to each TAM, and an optimal wrapper design for each core, such that the overall system testing time is minimized. In order to solve this problem, authors examine a progression of three incremental problems structured in order of increasing complexity, such that they serve as stepping-stones to the more general problem of wrapper/TAM design. The first problem P_W is related to test wrapper design. The next two problems P_{AW} and P_{PAW} are related to wrapper/TAM co-optimization.

1. $\boldsymbol{P_W}$: Design a *wrapper* for a given core, such that (i) the core testing time is minimized, and (ii) the TAM width required for the core is minimized.
2. $\boldsymbol{P_{AW}}$: Determine (i) an *assignment* of cores to TAMs of given widths and (ii) a *wrapper* design for each core, such that SOC testing time is minimized. (Item (ii) equals P_W.)
3. $\boldsymbol{P_{PAW}}$: Determine (i) a *partition* of the total TAM width among the given number of TAMs, (ii) an *assignment* of cores to the TAMs, and (iii) a *wrapper* design for each core, such that SOC testing time is minimized. (Items (ii) and (iii) together equal P_{AW}.)

These three problems lead up to P_{NPAW}, the more general problem of wrapper/TAM co-optimization described as follows.

4. $\boldsymbol{P_{NPAW}}$: Determine (i) the *number* of TAMs for the SOC, (ii) a *partition* of the total TAMwidth among the TAMs, (iii) an *assignment* of cores to TAMs, and (iv) a *wrapper* design for each core, such that SOC testing time is minimized. (Items (ii), (iii) and (iv) together equal P_{PAW}.)

Authors [50] assume the "test bus" model for TAMs and that the TAMs on the SOC operate independently of each other; however, the cores on a single TAM are tested sequentially. This can be implemented either by (i) multiplexing all the cores assigned to a TAM, or (ii) by testing one of the cores on the TAM, while the other cores on the TAM are bypassed.

Test wrappers provide a variety of operation modes, including normal operation, core test, interconnect test, and (optional) bypass [51]. In addition, test wrappers need to be able to perform test width adaptation if the width of the TAM is not equal to the number of core terminals. The IEEE P1500 standard addresses the design of a flexible, scalable wrapper to allow modular testing [48, 49]. This wrapper is flexible and can be optimized to suit the type of TAM and test requirements for the core. A "test collar" was proposed in [23] to be used as a test wrapper for cores. However, test width adaptation and interconnect test were not addressed. The issue of efficient de-serialization of test data by the use of balanced wrapper scan chains was discussed in [44]. Balanced wrapper scan chains, consisting of chains of core I/Os and internal scan chains, are desirable because they minimize the time required to scan in test patterns from the TAM. However, no mention was made of the method to be used to arrive at a balanced assignment of core I/Os and internal scan chains to TAM lines. The TESTSHELL proposed in [51] has provisions for the IEEE P1500 required modes of operation. Furthermore, heuristics for designing balanced wrapper scan chains, based on approximation algorithms for the well-known Bin Design problem [52], were presented in [46]. However the issue of reducing the TAM width required for a wrapper was not addressed. A number of TAM designs have also been proposed which include multiplexed access [53], partial isolation rings [30], core transparency [54], dedicated test bus [55], reuse of the existing system bus [56], and a scalable bus-based architecture called TESTRAIL [51]. Bus-based TAMs, being flexible and scalable, appear to be the most promising. However, their design has largely been ad hoc and previous methods have seldom addressed the problem of minimizing testing time under TAM width constraints. While [57] presents several novel TAM architectures (i.e., multiplexing, daisy chaining and distribution), it does not directly address the problem of optimal sizing of TAMs in the SOC. In particular, only internal scan chains are considered in [57], while wrappers and functional I/Os are ignored. Moreover, the lengths of the internal scan chains

are not considered fixed, and therefore [57] does not directly address the problem of designing test architectures for hard cores.

More recently, integrated TAM design and test scheduling has been attempted in [58, 59]. However, in [58, 59], the problem of optimizing test bus widths and arbitrating contention among cores for test width was not addressed. In [59], the cost estimation for TAMs was based on the number of bridges and multiplexers used; the number of TAM wires was not taken into consideration. Furthermore, in [58] the impact of TAM widths on testing time was not included in the cost function. The relationship between testing time and TAM widths using ILP was examined in [60, 61], and TAM width optimization under power and routing constraints was studied in [62]. However, the problem of effective test width adaptation in the wrapper was not addressed. This led to an overestimation of testing time and TAM width. Iyenger *et. al.* in [50] present a new wrapper/TAM co-optimization methodology that overcomes the limitations of previous TAM design approaches that have addressed TAM optimization and wrapper design as independent problems. The new wrapper design algorithm improves upon previous approaches by minimizing the core testing time, as well as reducing the TAM width required for the core. It uses an approach based on ILP to solve the problems of determining an optimal partition of the total TAM width and determining an optimal core assignment to the TAMs. They also address a new problem, that of determining the optimal number of TAMs for an SOC. This problem gains importance with increasing SOC size.

2.4.1 Constraint-driven test scheduling

Test scheduling is the process that allocates test resources (i.e. TAM wires) to cores at different time. Primary objective of test scheduling is to minimize testing time, while addressing one or more of the following issues: resource conflicts between cores arising from the use of shared TAMs [115], precedence constraints among tests [116], and power dissipation constraints [117]. Furthermore, testing time can be decreased further through the selective use of

test pre-emption [115]. As discussed in [115,116] most problems related to test scheduling for SOCs are also NP-hard.

Test scheduling was formulated as a combinatorial optimization problem. Reordering tests to maximize defect detection early in the schedule was explored in [117]. The authors used a polynomial time algorithm to reorder tests based on the defect data as well as execution time of the tests [117]. A test scheduling technique based on the defect probabilities of the core has been reported in [118]. In [119], a heuristic algorithm based on pair wise composition of test protocol was presented.

SOCs in test mode can dissipate up to twice the amount of power they do in normal mode, since cores, that do not operate in parallel may be tested concurrently [120]. Hence power constrained test scheduling is important. In [121], a method based on approximate vertex cover of a resource constrained test compatibility graph was presented. In [122], the use of a list scheduling and tree growing algorithm for power constrained scheduling was discussed. The authors presented a greedy algorithm to overlay tests such that the power constraint is not violated. The issue of re-organizing scan chains to trade-off testing time with power consumption was investigated in [123]. The authors presented an optimal algorithm to parallelize tests under power constraints. In [115], an integrated approach for test scheduling with precedence constraints was presented. In [124], a new approach wrapper/TAM co-optimization and constraint driven test scheduling using rectangular packing was described. Flexible widths TAMs that are allowed to fork and merge were designed. Rectangle packing was used to develop test schedules that incorporate precedence and power constraints, which allowing the SOC integrator to designate group of tests as pre-emptable. The work reported in [115] was extended in [125] to address the minimization of ATE buffer reload and include multisite testing. The mapping between core I/Os and SOC pins during the test schedule was investigated in [126]. TAM design and test scheduling was modeled as 2D bin packing in which each core test is presented by a rectangle. The authors next formulated constraint driven pin mapping and test

scheduling as the chromatic number problem from the graph theory and as a dependency matrix partitioning problem. In [127], SOC test problem is solved with power constraints as a 3D bin packing problem and provided a heuristic to handle the problem. Zou et al. proposed in [81] a method using simulated annealing (SA) to handle the problem. The problem has also been addressed by Harmanani [177]. In [81], TAM lines are not partitioned; rather, a rectangle fitting problem has been solved. It utilized a special data structure called *sequence-pair*. In this formulation, a TAM line needs to be switched individually, rather than the whole bus at a time between the cores. Thus, the test controller becomes complex. It borrowed concepts from floor-planning problem. In [177], an integrated approach for wrapper design, TAM assignment and test scheduling for SOC has been reported. Xia et al. in [86] presented an algorithm for co-optimizing test scheduling and wrapper design under power constraints by using an evolutionary algorithm and the sequence pair representation. Same authors also presented an algorithm for assigning non-consecutive TAM wires to core tests. In [128], authors proposed test scheduling solution using B-tree based floor-planning and using simulated anneling techniques. Recent work on TAM optimization has focused on the use of ATEs with port scalability features [129, 90,130]. Optimization techniques have been developed to ensure that the high-data-rate tester channels are efficiently used during SOC testing [130]. The availability of dual-speed ATEs was also exploited in [129,90], where a technique was presented to match ATE channels with high data rate to core scan chain frequencies using virtual TAMs. In [130], the hardware overhead is reduced over [129] through the use of a smaller number of on-chip TAM wires; ATE channels with high data rates directly drive SOC TAM wires, without requiring frequency division hardware. Most recently test methodology concerns about the hierarchical SOCs. A hierarchical SOC is designed by integrating heterogeneous technology cores at several layers of hierarchy [1]. The ability to reuse embedded cores in a hierarchical manner implies that 'today's SOC is tomorrow's embedded cores'. Two broad design transfer model: interactive and non-interactive, are emerging

in hierarchical SOC design flow. Recently test design methodologies for hierarchical SOCs were proposed in [131,132]. The problem of designing test wrappers and TAMs of multilevel TAMs for the cores within core's design paradigm [133,134,135,136] has been addressed. Two design flows have been considered for the scenario in which "megacores" are wrapped by the core vendor prior to delivery. In an alternative scenario, the megacores can be delivered to the system integrator appropriately designs the megacore wrappers and SOC-level TAM architecture to minimize overall testing time.

2.5 Test Wrapper Design and Optimization

The IEEE P1500 wrapper as shown in Fig. 2.7, is a thin shell around a core that allows the core and its environment to be tested independently[63].

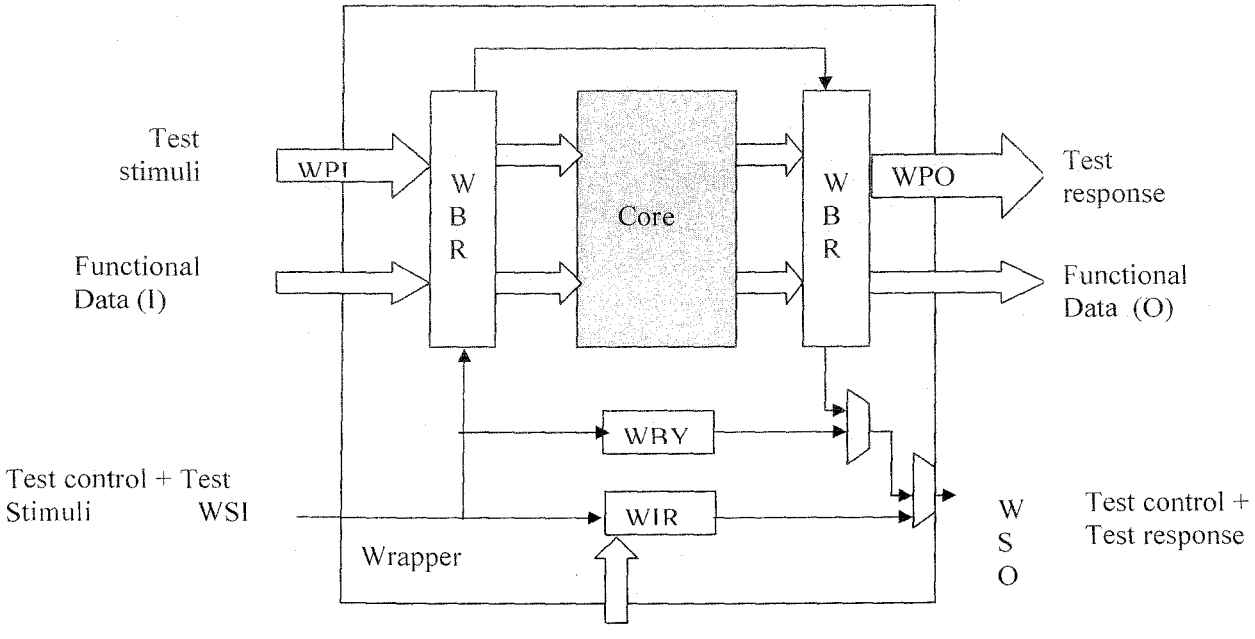


Figure 2.7: IEEE P1500 wrapper architecture [63]

The P1500 core wrapper is a shell which allows various configurations at its inputs and outputs, i.e., various operating modes. An IEEE P1500 core wrapper is illustrated in Fig. 2.7. The wrapper has as inputs the wrapper parallel input

(WPI), the wrapper serial input (WSI) and the functional inputs (I), and the wrapper interface port (WIP).

The outputs are the wrapper parallel output (WPO), the wrapper serial output (WSO) and the functional outputs (O). The core wrapper comprises two wrapper boundary registers (WBR), the wrapper bypass register (WBY) and the wrapper instruction register (WIR). The WBRs provide the interface between the test environment and the core, facilitating the test modes normal operation, core-internal test and core external test. The WBR is a register formed out of wrapper boundary scan cells. The wrapper cell assigned to the inputs are referred to as wrapper boundary input cells, while those assigned to the outputs are referred to as wrapper boundary output cells. The WIP facilitates the load of instructions corresponding to the above modes into the WIR. For each of the above modes there can be different configurations. For example, there can be a serial internal test, when the test stimuli are provided through the WSI (serial access), and the test responses are observed at the WSO; or a parallel internal test when the test stimuli are provided through the WPI (parallel access), and the test responses are observed at the WPO. A standardized, but scalable test wrapper is an integral part of the IEEE P1500 working group proposal [48]. A test wrapper is a layer of DFT logic that connects a TAM to a core for the purpose of test [64]. Test wrappers have four main modes of operation. These are (i) *Normal* operation, (ii) *Intest*: core-internal testing, (iii) *Extest*: core-external testing, i.e., interconnect test, and (iv) *Bypass* mode. Wrappers may need to perform test width adaptation when the TAM width is not equal to the number of core terminals. This will often be required in practice, since large cores typically have hundreds of core terminals, while the total TAM width is limited by the number of SOC pins. Iyenger *et. al.* in [50] address the problem of TAM design for *Intest*. A core usually contains several core I/Os as well as several internal scan chains consisting of flip-flops connected in serial within the core for the purpose of scanning test data in and out of the core. To perform test width adaptation, wrapper scan chains are constructed by connecting core I/Os and internal scan

chains in serial. The number of wrapper scan chains constructed is equal to the TAM width provided to the core; hence each wrapper scan chain is assigned to a single unique TAM line. Thus the test data width (number of core terminals) of the core is adapted to the TAM width. The problem of designing an effective width adaptation mechanism for *Intest* can be broken down into three problems [65]: (i) partitioning the set of wrapper scan chain elements (internal scan chains and wrapper cells) into several wrapper scan chains, which are equal in number to the number of TAM lines, (ii) ordering the scan elements on each wrapper chain, and (iii) providing optional bypass paths across the core. The problems of ordering scan elements on wrapper scan chains and providing bypass paths were shown to be simple in [65], while that of partitioning wrapper scan chain elements was shown to be NP-hard.

Recent research on wrapper design has stressed the need for balanced wrapper scan chains [44, 65]. *Balanced* wrapper scan chains are those that are as equal in length to each other as possible. Balanced wrapper scan chains are important because the number of clock cycles to scan in (out) a test pattern to (from) a core is a function of the length of the longest wrapper scan-in (scan-out) chain. Let s_i (s_o) be the length of the longest wrapper scan-in (scan-out) chain for a core. The time required to apply the entire test set to the core is then given by $T = (1 + \max\{s_i, s_o\}).p + \min\{s_i, s_o\}$, where p is the number of test patterns. This time T decreases as both s_i and s_o are reduced, i.e., as the wrapper scan-in (and scan-out) chains become more equal in length.

Figure 2.8 illustrates the difference between balanced and unbalanced wrapper scan chains; *Bypass* and *Extest* mechanisms are not shown. In Figure 2.8(a), wrapper scan chain 1 consists of two input cells and two output cells, while wrapper scan chain 2 consists of three internal scan chains that contain 14 flip-flops in total. This results in unbalanced wrapper scan-in/out chains and a scan-in and scan-out time per test pattern of 14 clock cycles each. On the other hand, with the same elements and TAM width, the wrapper scan chains in Figure 2.8(b) are balanced. The scan-in and scan-out time per test pattern is now 8 clock cycles. The problem of partitioning wrapper scan chain elements into

balanced wrapper scan chains was shown to be equivalent to the well-known Multiprocessor Scheduling and Bin Design problems in [65]. In this paper, the authors presented two heuristic algorithms for the Bin Design problem to solve the wrapper scan chain element partitioning problem. Given k TAM lines and sc internal scan chains, the authors assigned the scan elements to m wrapper scan chains, such that $\max\{s_i, s_o\}$ was minimized. This approach is effective if the goal is to minimize only $\max(s_i, s_o)$.

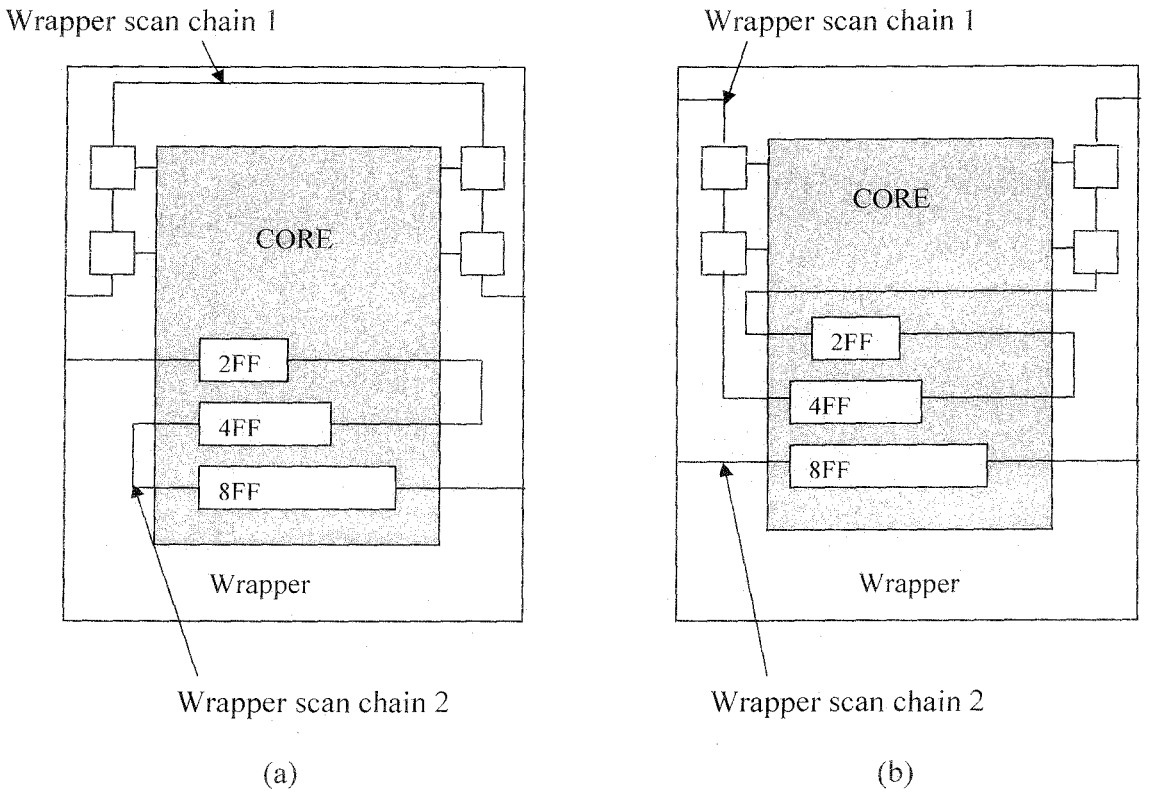


Figure 2.8: Wrapper chains (a) Unbalanced (b) Balanced

This can be explained as follows. Consider a core that has four internal scan chains of lengths 32, 8, 8, and 8, respectively, 4 functional inputs, and 2 functional outputs. Let the number of TAM lines provided be 4. The algorithm in [65] will partition the scan elements among four wrapper scan chains as shown in Fig. 2.9(a), giving $\max\{s_i, s_o\} = 32$. However, the scan elements may also be assigned to only 2 wrapper scan chains as shown in Fig. 2.9(b), which also gives $\max\{s_i, s_o\} = 32$. The second assignment, however, is clearly more

efficient in terms of TAM width utilization, and therefore would be more useful for a wrapper/TAM co-optimization strategy.

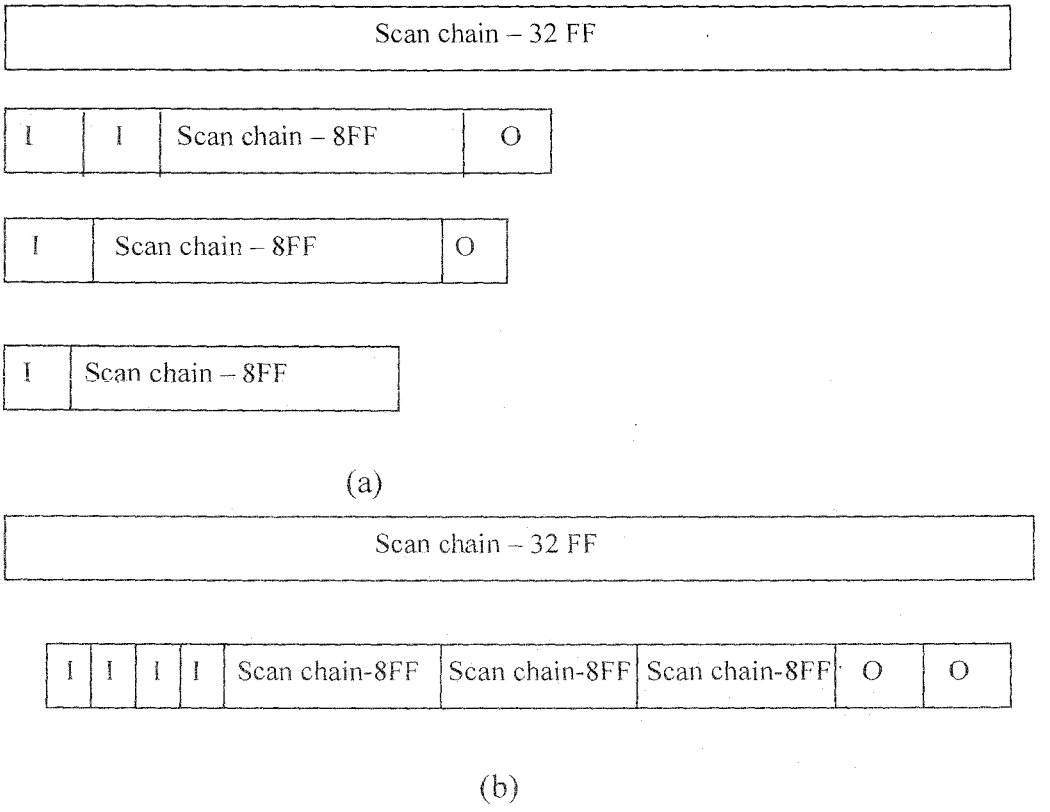


Figure 2.9: Wrapper design example using (a) four wrapper scan chain, and (b) two wrapper scan chains

Koranne [66, 67] addressed a design of reconfigurable core wrapper which allows a dynamic change in the TAM width while executing the core test. This is achieved by placing extra multiplexers at the input and output of each reconfigurable scan chain. He also described a procedure for the automatic derivation of these multiplexers using a graph representation of core wrappers. Instead of connecting the outputs of each scan chain to multiplexers, Larsson and Peng [68] presented a reconfigurable power-conscious core wrapper by connecting the inputs of each scan chain to multiplexers. This approach

combines the reconfigurable wrapper [66,67] with the scan chain clock gating [69] concepts. The reconfigurable core wrapper is useful for cores with multiple tests, where each of the tests has different TAM width requirements.
In [70], Vermaak and Kerkhoff presented a P1500 compatible wrapper design for delay fault testing, based on the digital oscillation test method. To be able to use this method, they introduced extra multiplexers and a cell address register to each wrapper cell. This approach for delay testing is only suitable for combinational cores, because paths between the core's inputs and outputs are tested. Xu and Nicolici [71] proposed a novel core wrapper that addressed the testability problems raised by embedded cores with multiple clock domains. By partitioning core I/Os and scan cells from different clock domains into different virtual cores, they proposed to build wrapper SCs within virtual cores to solve the clock skew problem during the shift phase. To avoid clock skew during the capture phase, a capture window design technique similar to [72] was proposed that supports multi-frequency at-speed testing. The authors also described wrapper optimization algorithms that can tradeoff between the number of tester channels, test application time, area overhead and power dissipation. In [73], Goel described a wrapper architecture for hierarchical cores, which allows for parallel testing of the parent and child cores, at the cost of an expensive wrapper cell design.

2.6 Design-Wrapper Algorithm

V.Iyenger, K. Chakraborty, and E.J. Marinissen [50] have developed an approximation algorithm based on the Best Fit Decreasing (BFD) heuristic [52] to solve P_W efficiently. The algorithm has three main parts, similar to [65]: (i) partition the internal scan chains among a minimal number of wrapper scan chains to minimize the longest wrapper scan chain length, (ii) assign the functional inputs to the wrapper scan chains created in part (i), and (iii) assign the functional outputs to the wrapper scan chains created in part (i). To solve part (i), the internal scan chains are sorted in descending order. Each internal scan chain is then successively assigned to the wrapper scan chain, whose

length after this assignment is closest to, but not exceeding the length of the current longest wrapper scan chain. Intuitively, each internal scan chain is assigned to the wrapper scan chain in which it achieves the *best* fit. If there is no such wrapper scan chain available, then the internal scan chain is assigned to the current *shortest* wrapper scan chain. Next the process is repeated for part (ii) and part (iii), considering the functional inputs and outputs as internal scan chains of length l. The pseudocode for this *Design wrapper* algorithm is as follows.

Procedure *Design- wrapper*

Part (i)

1. **Sort** the internal scan chains in descending order of length
2. **For** each internal scan chain l
3. **Find** wrapper scan chain S_{max} with current maximum length
4. **Find** wrapper scan chain S_{min} with current minimum length
5. **Assign** l to wrapper scan chain S, such that {length(S_{max}) - (length(S)+length(l))} is minimum
6. *If* there is no such wrapper scan chain S *then*
7. Assign l to S_{min}

Part (ii)

8. Repeat steps 1 through 7 to add the primary inputs to the wrapper chains created in part (i)

Part (iii)

9. Repeat steps 1 through 7 to add the primary inputs to the wrapper chains created in part (i)

Algorithm 2.1: Design_wrapper algorithm

	Wrapper scan chains			
	1	2	3	4
Internal scan chains	12+6	12+6	8+6+6	8+8
Wrapper input cells	2	2	0	4
Wrapper output	3	3	0	5
Scan-in length	20	20	20	20
Scan-out length	21	21	20	21

Table 2.1: Wrapper scan chains for Core A.

The algorithm designed based on the BFD heuristic mainly because BFD utilizes a more sophisticated partitioning rule than First Fit Decreasing (FFD), since each scan element is assigned to the wrapper scan chain in which it achieves the *best* fit [52]. FFD was used as a subroutine to the wrapper design algorithm in [65]. In this algorithm, a new wrapper scan chain is created only when it is not possible to fit an internal scan chain into one of the existing wrapper scan chains without exceeding the length of the current longest wrapper scan chain. Thus, while the algorithms presented in [65] always use k wrapper scan chains, *Design wrapper* uses as few wrapper scan chains as possible, without compromising test application time. The worst-case complexity of the *Design wrapper* algorithm is $O(sc.\log sc + sc \cdot k)$, where sc is the number of internal scan chains and k is the limit on the number of wrapper scan chains.

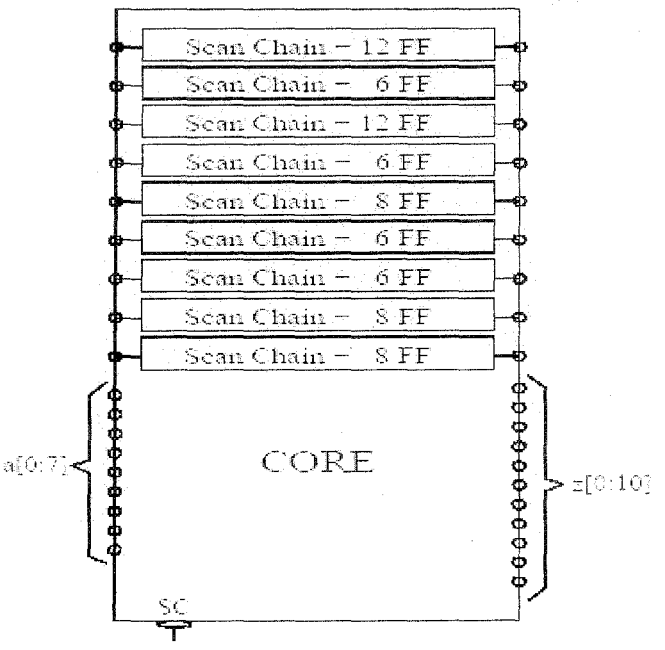


Figure 2.10(a): Example core A[65]

This algorithm is used to design wrapper for the example Core A in [65] shown in figure 2.10(a). Core A has 8 functional inputs `a[0:7]`, 11 functional

outputs z[0:10], 9 internal scan chains of lengths 12, 12, 8, 8, 8, 6, 6, 6, and 6 flip-flops, and a scan enable control *sc*. The test wrapper for A is to be connected to a 1-bit TAM STP (as mandated by the IEEE P1500 standard [48]) as well as to a 4-bit TAM MTP[0:3]. Furthermore, the test wrapper is to contain a bypass from TAM inputs to TAM outputs [65]. The scan elements in the core were partitioned among four wrapper scan chains using the *Design wrapper* algorithm as shown in Table 2.1. This partition yields a longest scan-in chain of length 20, and a longest scan-out chain of length 21, both of which are optimal values for a 4-bit TAM. The wrapper designed for Core A is illustrated in Fig. 2.10(b).

2.7 Test Data Compression

Test Data Compression (TDC) is a test resource partitioning scheme whose main target is the reduction of volume of test data sent from the ATE to the CUT. TDC implies the usage of a compression scheme during test set partitioning and an on-chip decompression unit during test set application. By introducing an on-chip hardware it reduces the load on the ATE, and therefore it simplifies the ATE channel capacity and speed requirements.

There exists wide number of data compression schemes proposed in the literature, some of which could give high compression on test data for SOCs but most of them are too complex for their corresponding decoder to be realized efficiently in hardware. Compression techniques mentioned in literature can broadly categorize in two categories. These are:

- Non-Dictionary based compression scheme.
- Dictionary based compression scheme.

In non-dictionary based compression techniques, total test data is decomposed into either fix length or variable length blocks and a code word, also of either fixed or variable length, is assigned to each block. The basic idea is to assign frequent blocks to a comparably small code word. Iyenger *et al.* [102] proposed a BIST approach for testing non-scanned circuits based on statistical coding. Jas et al. [103] presented a compression technique for scanned circuits by

dividing the test vectors into fixed length blocks and using the Huffman coding technique. In [104], Jas and Touba described another TDC scheme using run-length coding. By exploiting the capabilities of present ATEs to assign groups of inputs to ports and to perform vector repeat per port. Jas *et al.* in [104] employed variable-to-fixed length run-length encoding in which the runs of zeros are encoded by a fixed number of bits.

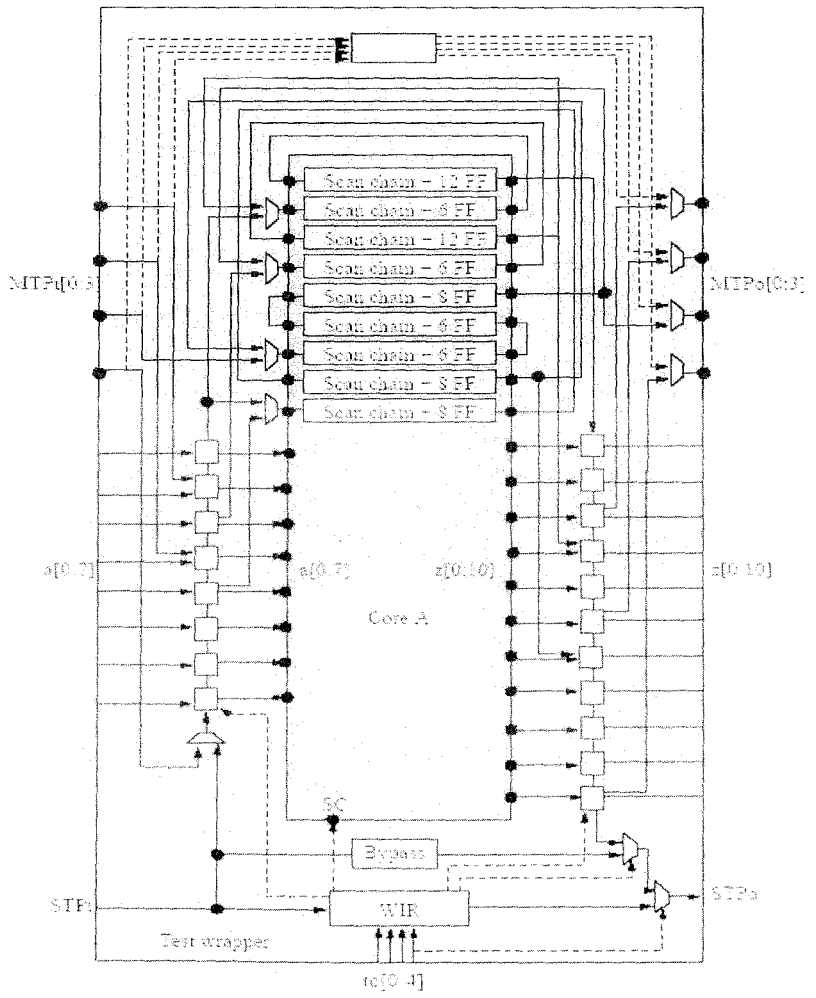


Figure 2.10(b): Wrapper design for example core A from [65].

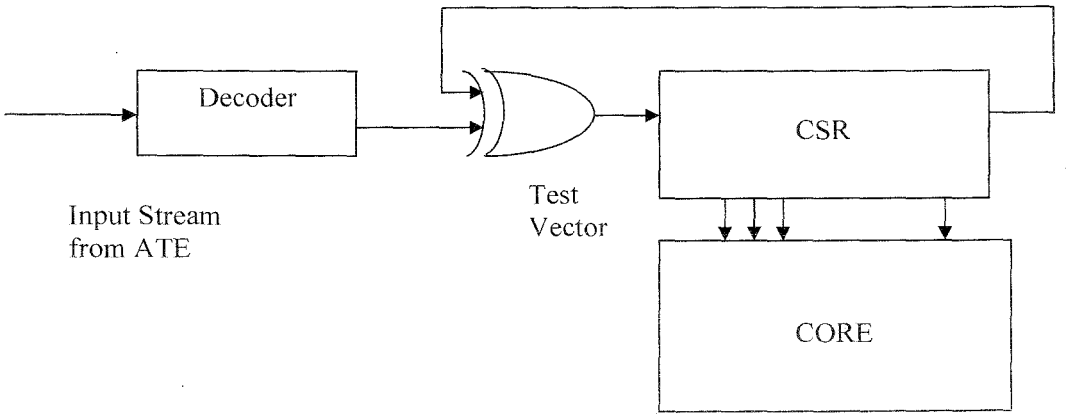


Figure 2.11: Decompression architecture based on a Cylindrical Scan Chain Register (CSR)

This method takes advantage of the fact that the consecutive vectors in the test set tend to be similar as the exercising the faults in the CUT that are structurally related require similar input value assignments. Hence instead of encoding the test vectors, this scheme encodes the difference vectors formed by taking the XOR of the consecutive test vectors. The test set is reordered to maximize the length of zeros in the difference vectors and the decompression hardware utilizes a cyclical scan chain register (CSR), as shown in Fig. 2.11 to obtain back the test vectors from the difference vectors before they are shifted into the scan chain of the CUT. The main drawback of this scheme is additional hardware area overhead that the CSR incurs.

Vranken et al. [105] implemented a similar run-length coding scheme, although the decompression is achieved of-chip on the ATE instead of on-chip. The above coding techniques are based on "variable-to-fixed-length" codes. Chandra and Chakrabarty [106] first proposed a "variable-to-variable-length" test data coding scheme, based on Golomb coding [107]. When using Golomb coding the saving in scan-in power are trade-off against improvement in compression ratio. Rosinger et al. described a Minimum Transition Count (MTC) coding scheme, which can simultaneously reduce test data and power. Frequency-directed run-length coding [108] technique proposed by Chandra and Chakrabarty, further increased the compression ratio. It is designed based

on the observation that the frequency of runs decreases with the increase of their length. While the original FDR coding was based on encoding runs of 0's, El-Maleh and Al-Abaji [109] extended it with EFDR coding, by encoding both runs of 0's and 1's. Gonciari et al. [110] analysed the three main test data compression environment parameters: compression ratio, decoder area overhead and test application time, and introduced a variable-length input Huffman (VIHC) coding scheme to efficiently trade them off. Finally, Tehranipour et al. [111] presented a 9C coding scheme that supports mapping "don't care" bits to random values instead of dedicated long runs of 0's and 1's, which is able to detect more non-modeled defects. On-chip decoders for most of these techniques are parallel in nature. These parallel on-chip decoders will require less amount of area to implement compare to those of serial decoders used for dictionary based methods. But as it is parallel extra synchronization circuitry is needed to synchronize two units with ATE. It is possible to reduce the TAT in parallel decoders by using them at optimum operating frequency.

In dictionary based compression schemes, some selected entries are taken into the dictionary. These entries are encoded using techniques like statistical coding to get a statistical model of the test data and encode blocks according to their frequencies of occurrence. Dictionary based methods select strings of the blocks to establish a dictionary, and then encode them into equal size tokens [107]. The dictionary may be either static or dynamic(adaptive). The static dictionary is permanent, whereas the dynamic dictionary permits additions and deletions of strings as new input is processed. Wolf and Papchristou [112] described a method that employs the well-known LZ77 algorithm, which uses dynamic dictionary. Knieser et al. [113] presented another TDC technique based on LZW algorithm. Li and Chakrabarty [114] proposed a TDC approach using dictionaries with fixed-length indices. One advantage of this method is the elimination of the synchronization problem between the ATE and the SOC, because it does not require multiple clock cycles to determine the end of a compressed data packet. Decoders for the dictionary based compression

techniques are serial in nature, as a result there is no synchronization between ATE and CUT. Usually a RAM or combinational circuitry is used to recover the dictionary encoded patterns and a shift register based circuitry is used to transfer non-dictionary entries. This will lead to a large amount of area overhead of on-chip decoder compare to non-dictionary methods.

2.8 Conclusion

In this chapter a survey report have been presented regarding the methodologies adopted for SOC testing. The conceptual architecture for core test and test access and test wrapper design methodologies have been discussed here. Different method of optimal test architecture, including assigning cores to test busses, distributing a given test data width among multiple test busses and test data compression technique have also been reviewed. From the next chapter onwards, we will enumerate our works. To start with, we will present a simulated annealing based approach for test scheduling