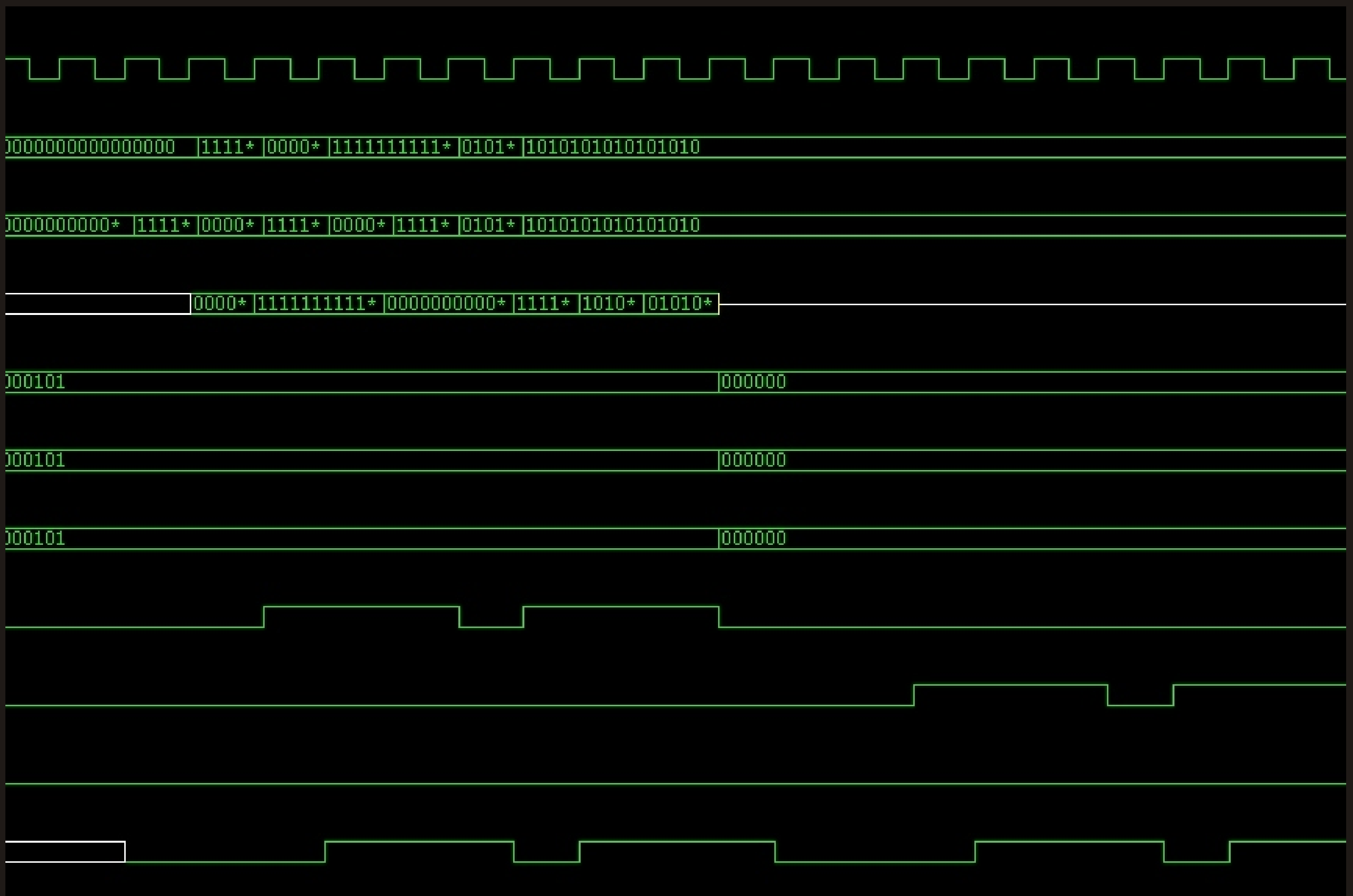
Vladimir A. Zivkovic

Testability in Hardware/Software Codesign Systems



CHAPTER 1

Introduction

1.1. Introduction

In 1997, when the research described in this thesis started, *hardware/software* codesign was a “hot topic” and was a consequence of the current trends in semiconductor industry. The semiconductor industry has historically delivered four times as many functions per chip every three years with only a moderate 1.4 times increase in chip size and 1.4 times in cost (approximately constant cost per cm^2 of silicon). The ever-increasing development in present semiconductor technology has been defined by Moore’s law as early as in 1965 [MOO65]. However, even today, Semiconductor Industry Association (SIA) issued the strong expectation that this trend will furthermore continue [SIA01], as given in Table I. This technological and economic performance is the fundamental engine behind the growth of the semiconductor industry enabling the design of complete *Systems-on-Chip* (*SoC*) at one single die. Such complex Systems-on-Chip are typically very large Integrated Circuits (IC’s) consisting of millions of transistors and contain a variety of both digital and analogue hardware modules. They are comprising of e.g. microprocessors, RAM, ROM, digital signal processing blocks (DSP), user defined blocks (UDB), A/D and D/A converters, timers, PLL, all of them integrated within one single IC. In addition, microelectromechanical systems (MEMS), electro-optical, electro-biological, and other nontraditional elements arise from and give rise to changes in technology. With this rapid rise in heterogeneity, technology advances, new product requirements, reuse of already designed intellectual properties (IP), once considered to be a factor that would ease design

productivity, become difficult. Moreover, incorporation of several separately-designed components on a chip requires significant integration and verification cost.

TABLE I SEMICONDUCTOR TECHNOLOGY DEVELOPMENT [SIA01]

Year	1997	1998	1999	2002	2005	2008	2011	2014
Feature Size [nm]	250	250	180	130	100	70	50	35

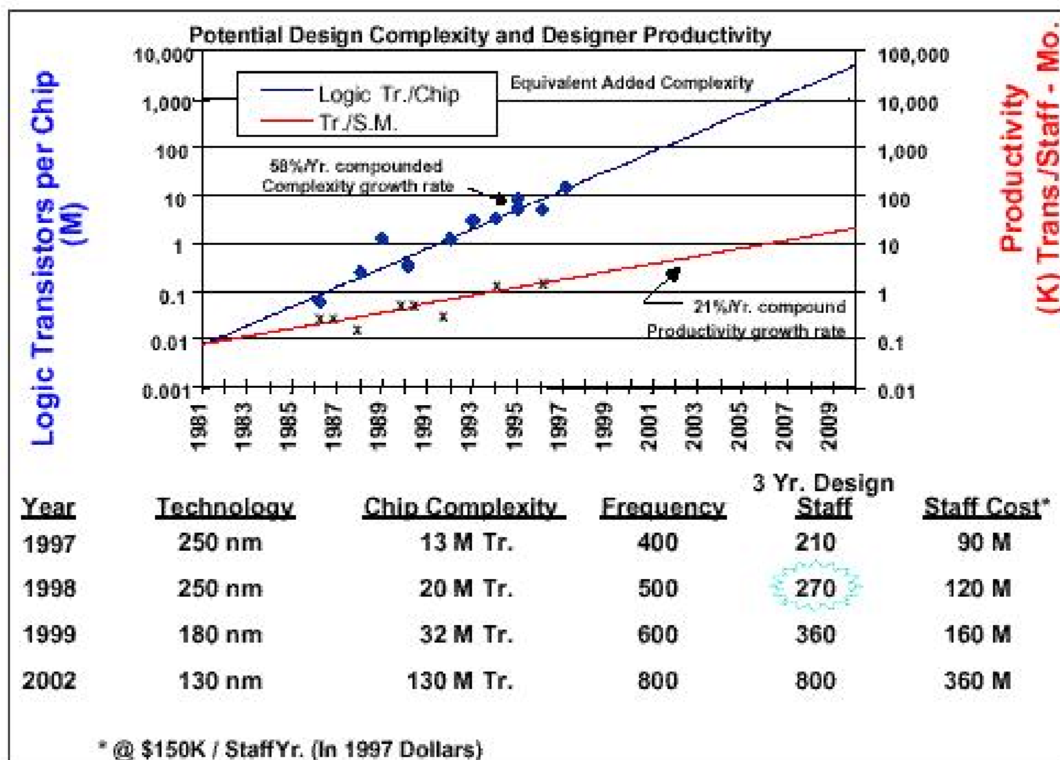
The increasing complexity of design requires high-skilled, more broadly-trained designers and Computer-Aided Design (CAD) tools that take into account factors that could be ignored in previous technology generations. For example, logic synthesis tools now must take into account interconnect and physical placement and increasingly accommodate the transistor-level properties of dynamic circuit families and the noise effects at smaller feature sizes and higher frequencies. The design flow must assure functional correctness and timing, power, reliability, manufacturability, signal integrity and testability requirements. Thus, design complexity increases exponentially, but automated tools are handicapped by having to run on the previous generation of computing equipment. This complexity has a similar effect on designers whose abilities are not always tracking the Moore curve for certain applications such as high-performance Application-Specific Integrated Circuits (ASIC), with distinctions being blurred between logic, layout, and circuit design.

As designs move from digital microprocessors and ASICs to SoCs, designers and design tools also encounter complex embedded software, and the challenges of providing a diverse range of components on a single chip. The rapidly changing technological environment also shrinks product life cycles, making time to-market one of the most critical issues for semiconductor customers. This demand is driven by growth of personal information flow through wired and wireless voice and data communication, and the Internet. There is great pressure to reduce the total time to create a chip for products that are not obsolete before they are brought to market. This time is dominated by the system design and verification phase. Investment in technology improvements has dominated product creation resources, and design productivity has not been able to keep pace with transistor density growth. Figure 1.1 indicates the growing productivity gap, defined as the ratio between available number of transistors (per chip) and number of transistors that can be designed (per man-month) [SIA99]. Design reuse addresses only part of the productivity gap. Increase in design team size has problems with respect to productivity of large groups and is limited by communication, integration and software issues.

The SoC's are most often used as so-called *embedded systems* in an increasing number of applications. The embedded systems pose new design challenges, which are believed to be the driving forces of design automation in the incoming years. These included the design of electronic circuitry, i.e., the hardware, embedded software and its integration into a heterogeneous hardware/software (HW/SW) codesign system. The goal of codesign is to find an optimal HW/SW architecture that implements the system specification and meets the constraints with regard to real-time behaviour, speed, area, memory, power consumption, flexibility etc. In codesign, the implementation decisions for hardware, software and communication interfaces are closely related: changes in one will immediately affect the other two. High-level design and the accompanying design automation tools are often considered key to design productivity improvement. However, to avoid frequent rework and iteration, decisions made at higher levels of design must be based on an accurate estimation of the effects of lower-level decisions, and/or must permit

later adjustments to logical and physical factors. As time-to-market issues and SoCs become more common, creating an embedded HW/SW system emerges as a key design problem. The associated issues include:

- high-level architectural design-space exploration
- analysis of the tradeoffs (design cycle time, performance, cost) of implementing designs in hardware and software
- high-level design planning and estimation
- HW/SW co-design at all design levels
- analysis, verification and test issues.



Year	Technology	Chip Complexity	Frequency	3 Yr. Design Staff	Staff Cost*
1997	250 nm	13 M Tr.	400	210	90 M
1998	250 nm	20 M Tr.	500	270	120 M
1999	180 nm	32 M Tr.	600	360	160 M
2002	130 nm	130 M Tr.	800	800	360 M

* @ $150K / StaffYr. (In 1997 Dollars)

Figure 1.1. The design productivity gap [SIA99]

Examples of codesign application domains are control systems, communication and telecommunication systems (GSM or DECT terminals), video-image processing (MPEG image compression), multimedia systems, automotive real-time control, domestic and aerospace systems. In principle, there are two codesign methods differing from each other with respect to approaches to reach the codesign goals. Hence, the HW/SW codesign methods can be roughly classified into:

1. Codesign of heterogeneous HW/SW systems starting from an implementation-independent specification.
2. Codesign of *Application-Specific Instruction Processors* (ASIP) such as the widely used and accepted *Very-Long Instruction Word* (VLIW) Processor.

The research efforts in this thesis are primarily focused on the investigation of implementing the optimal solution with regard to the test in the field of hardware/software codesign of VLIW architectures. Hence the appropriate introduction into the ASIP-VLIW codesign flow will be given next.

1.2. Hardware/Software codesign of an ASIP-VLIW

A VLIW processor is a dedicated processor in which the hardware architecture and the instruction set are optimised to execute algorithms in a specific application domain [GOO96]. An ASIP-VLIW combines the concept of an ASIC and a programmable processor. However, a VLIW processor is software programmable and hence, more flexible than an ASIC and at the same time a VLIW processor provides higher performance for executing software than a standard processor. The architecture and instruction set of modern VLIW processors enable their customisation to a specific application such as GSM or DECT terminals, MPEG image compression, automotive real-time control, etc. [MIC96]. The general flow of VLIW-ASIP HW/SW codesign has the form as shown in Figure 1.2.

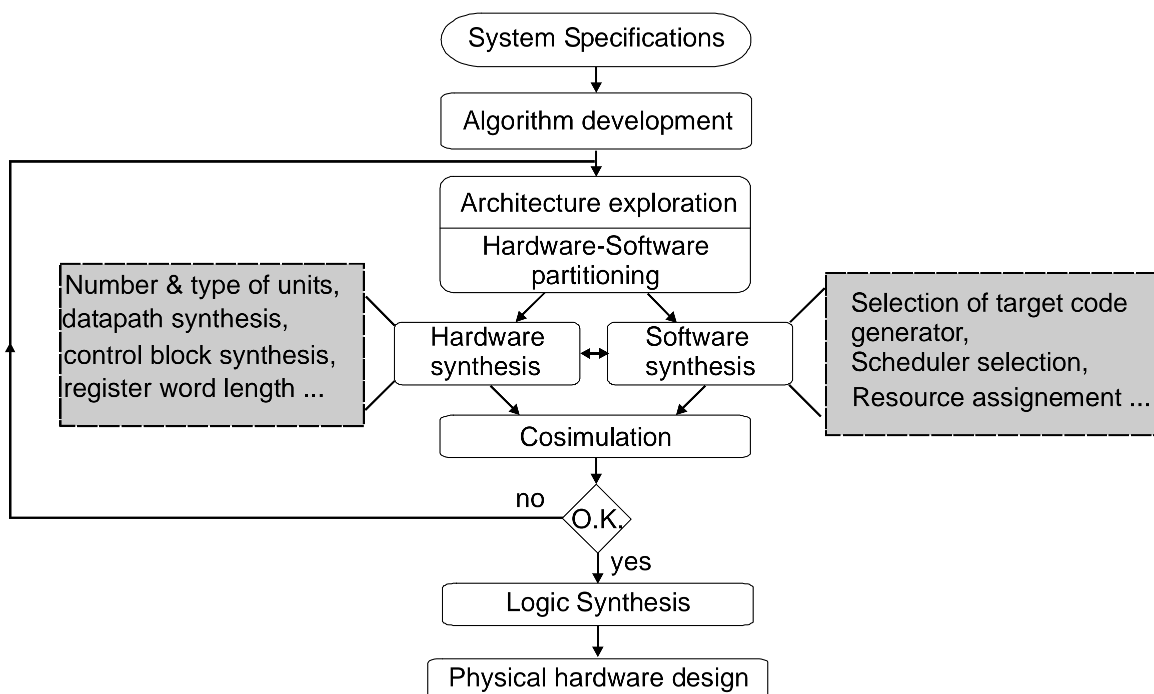


Figure 1.2. Basic HW/SW codesign flow

The flow starts from system specifications assuming that the functional behavior of the system is captured into a conceptual, formal model. The system specification already implies an implicit architectural model and has to be organized in such a way to facilitate this implication during the transition of behavior specification to HW/SW architecture. Given a system specification, an algorithm or specification is developed and described in one of the widely used programming languages such as *C* or *C++*. Next, in the architecture exploration phase, numerous alternative HW/SW architectures are explored in order to find the one that satisfies initial constraints. The exploration phase is the key issue

in HW/SW codesign as the HW/SW partitioning is also addressed in this phase. Namely, system architecture is a set of hardware and software components, each component implementing a part of the functional specification.
Partitioning is followed by hardware and software synthesis. Hardware synthesis is comprised of exploring the design space and choosing the appropriate functional modules for datapath and control block. In addition, the data and control width busses and registers are determined during the hardware synthesis. Once the architectural instantiation is defined, a *High-Leve*l synthesis tool translates a hardware system architecture into the Register-Transfer Level (RTL) resulting in HDL (Hardware Description Language) source of the initial application.

On the other hand, the software synthesis generates instruction-level object code and gives statistics on the usage of hardware resources. The information exchange between hardware and software components needs to be maintained during the synthesis. That can be achieved via cosimulation at various levels of abstraction. The final cosimulation assumes the object code of the generated software running at the hardware of the architecture described at the gate level. If the cosimulation proves a correct result, the flow proceeds with the *Logic Synthesis* converting the RTL components into the gate-level description. More accurately, the logic synthesis performs the translation of HDL source into the generic library of components followed with optimization and mapping into the gate-level components. Finally, physical hardware design performs translation of the gate-level design into the layout.

Though the codesign flow in Figure 1.2 suggests a sequential execution of tasks, it is not the case. There are several local iteration loops and concurrent work (with feedback) particularly among the algorithm development, architecture exploration and software synthesis [PUT95].

There are several CAD tools tending to employ the previously described steps and integrate them into one comprehensive package. Examples are e.g., the MOVE codesign package created at the University of Delft [COR95], CASTLE at GMD (Germany) [WIL97], Ptolemy at Berkeley University [KAL93], CoWare at IMEC [VER96], *Synopsys* Co-centric studio [SYN01] etc.

The codesign flow depicted in Figure 1.2 does not provide the designer with the possibility to assess the test quality of the final product. Testing of such systems is crucial to ensure that only good products will be shipped to the customers at reasonable test cost. However, testability must be designed and planned during the architectural design phase [SIA99]. This is easily explainable if one has in mind the complexity of embedded HW/SW ASIP-VLIW architectures, as they can be built as a part of a complex SoC design. As the number of functions/chip continues to increase, it becomes increasingly difficult and costly to test the final products. The current generation of IC technology has already the transistor per pin ration far above 1000:1. Therefore the problem of accessibility in this kind of environment arose. Depending solely on external test equipment has not been an option. If testers continue on the full-pin, at-speed, functional test route, costs will exceed $20M per tester by the year 2014. Similarly, test times on these very expensive testers will approach hours per device because of the increasing patterns required across the fixed bandwidth tester/device interface. Hence, the Design for Testability (DfT) techniques to improve the controllability and observability capabilities of embedded systems have been put in practice. Yet, SIA predicts that by the year 2010, it may cost more to test than to manufacture a transistor unless an appropriate application driven DfT technique and test style is employed for the test. This particularly includes the high-performance, dedicated applications such as VLIW ASIP architectures, typical

products of HW/SW codesign. The test parameter has become one of the key issues of their success in the market.

1.3. Objectives and outline of the thesis

The objective of this thesis is to develop a generic method towards the implementation of test that targets the faults that may occur during the chip manufacturing of the VLIW ASIP HW/SW codesign product. The method has to be incorporated earlier in the design as an additional test constraint, but with clearly defined connections with regard to the gate-level test implications.

An independent reader may assume that we try to suggest the profiling of HW/SW test and its integration into the HW/SW codesign package. However, this is not the case. We strongly believe that "hardware/software test" is not a very convincing term. There may exist hardware tests and software tests while the testing of their communication is related to debugging and design for debug (DfD) of a HW/SW system [VRA98]. As manufacturing faults are the target, the research efforts in this thesis are concentrated completely on the test of the hardware of the architecture while the software test has not been tackled. We have assumed that the software is "correct by construction", i.e., that the formal verification has already been performed. Also, system-level testing, as verifying the correctness of a system as a whole has not been addressed in this thesis. Hence, the faults that may appear during the execution of the VLIW assembler program are assumed to have an origin due to the faults in the memory where the program itself is stored. However, memory test is also part of the hardware test.

The objective of our approach is to achieve high fault coverage with minimal adverse affects on circuit throughput and area. Methodologies must ensure that simulations and analysis done earlier in the design do not have to be redone after test insertion. Any special testability cells insertion in the design has to be done early so that the effects on timing, area and test coverage may be analyzed and assessed promptly. Moreover, the test implementation at the gate-level has to be derived according to the test constraint.

There is another important objective that needs to be achieved and is related to the *test time*. The overall quality of test has inevitably to take into account the time that the Automatic Test Equipment (ATE) is occupied while testing the circuit. The test time directly influences the cost of the chip and that aspect will become even more significant in the future, according to the recent SIA report [SIA00]. As digital parts get progressively larger in the future, the external pattern volume becomes prohibitive to apply (test time = $$) while still maintaining the high fault coverage necessary for the desired quality (even with low-cost ATE). The development of new methodologies to overcome this situation is a necessity.

In summary this thesis proposes and describes the generic method that provides an efficient hardware test of ASIP-VLIW HW/SW architectures, while fulfilling the following objectives:

- Test-time minimization
- High fault coverage
- Low DfT (Design for Testability) area overhead
- Minimum throughput penalty
- Seamless integration into the HW/SW codesign packages

The thesis contributes to advancing state-of-the-art design for test of VLIW architectures. We propose an alternative test-style in order to fulfill the objectives as stated above. In addition, another added value of the thesis is the established link between the test constraints introduced early at the design stage and its gate-level implementation. This thesis is organized into five chapters and one appendix, apart from the introduction covered in this chapter.

Chapter 2 describes the state-of-the-art in modern digital test approaches. The aspects of various test styles interesting from the point of VLIW-ASIP test view will be clarified. Special attention will be paid to the new test style, the so-called "*Core-based test*", becoming widely used in modern design & test community. Also, the test cost issue will be discussed in this chapter.

Chapter 3 refers to the practical traineeship that the author of this thesis has been carrying out at the Philips Semiconductors ESTC group in Eindhoven working in the area of core-based test. A consistent Computer-Aided Test (CAT) flow is profiled based on the required core-test strategy. It generates a test-pattern set for the embedded cores with high fault coverage and low DfT area overhead. The CAT flow is applied within the Philips Core Test Pilot IC project. The gained experience has been used as a basis in initial development of the test strategy applied in subsequent chapters of the thesis.

In *Chapter 4*, high-level synthesis for test is treated. Test constraint is introduced in the early phase of codesign. The proposed method calculates the testability of the system, helps the designer to assess the obtained architectural instantiations with respect to test, area and throughput in the early phase of the codesign. The test constraint is introduced at the high-level of the design, but the link with the design instantiation at lower hierarchy levels is always maintained during the codesign process. The approach is applied in two existing HW/SW codesign packages called MOVE [COR95] and CASTLE [WIL97] while running several applications and targeting the manufacturing faults. The proposed approach reduces the size of test vectors while keeping the DfT area low and fault coverage sufficiently enough.

Chapter 5 elaborates on the implementation of the test strategy at the gate-level in a so-called Very Long Instruction Word Transport Triggered Architecture (VLIW-TTA). The complete test strategy is derived referring to the results of test synthesis as introduced in Chapter 4 and carried out during the early phase of the design. It takes the area/throughput parameters into account. The test strategy, exploiting the regularity and modularity of the VLIW-TTA structure, remains general for an arbitrary application and instantiation of the TTA processor and is based on the partial-scan approach along with the functional test. The test-time analysis, in order to justify our approach and show the superiority over the classical full-scan has been performed. The results of our strategy have shown the superiority over classical full scan test in most of the aspects that determine the overall test quality of the product.

Chapter 6 summarizes the thesis and points out the originality and added value of the approach. Also, the recommendations for future research in this area are outlined.

References:

[COR95] H. Corporaal, *Transport Triggered Architectures; Design and Evaluation*, Ph.D. thesis, Delft University of Technology, September 1995.

[GOO96] G. Goossens, J. van Praet, D. Lanneer, W. Geurts, F. Thoen, “Programmable Chips in Consumer Electronics and Telecommunications,” “*Hardware/Software Codesign*,” Kluwer Academic Publishers 1996.

[KAL93] A. Kalavade, E. A. Lee, “A Hardware/Software Codesign Methodology for DSP Applications,” *IEEE Design & Test of Computers*, September 93, vol. 10, no 3, pp. 16-28.

[MIC96] G. De Micheli, M. G. Sami, et al., “*Hardware/Software Codesign*,” Kluwer Academic Publishers, 1996.

[MOO65] G. Moore, “Cramming more components onto integrated circuits,” *Journal of Electronics*, vol. 38, no 8, April 19, 1965.

[PUT95] P. H. A. van der Putten, J. P. M. Voeten, “Object-Oriented Codesign for Hardware/Software Systems,” *IEEE Proc. of EUROMICRO*, 1995, pp. 718-726.

[SIA99] SIA semiconductor industry association, “The National Technology Roadmap for Semiconductors,” 1999 edition.

[SIA00] SIA semiconductor industry association, “The National Technology Roadmap for Semiconductors,” 2000 update.

[SIA01] SIA semiconductor industry association, “The National Technology Roadmap for Semiconductors,” 2001 update.

[SYN01] http://www.synopsys.com/products/cocentric_studio/cocentric_studio_ds.html

[VER96] D. Verkest, K. Van Rompaey, I. Bolsens, H. De Man, “CoWare – A Design Environment for Heterogeneous Hardware/Software Systems,” Design Automation for Embedded Systems 1 (1996), no 4, pp. 357 – 386.

[VRA98] H. Vranken, Design For Test & Debug in Hardware/Software Systems, Ph.D. thesis, Technical University of Eindhoven, June 1998.

[WIL97] J. Wilberg, *Codesign for Real-Time Video Appliccations*, Kluwer Academic Publishers, Dordrecht, the Netherlands, ISBN 0-7923-8006-1, 1997.

CHAPTER 2

Testing Issues in Advanced Digital ICs

2.1. Introduction

It has been common in the past to view Design-for-Testability (DfT) as a design option only. DfT has been considered as an issue that may be included or not, depending on various economic parameters. Implicit in this view was the assumption that circuit designers and test engineers were able to write extensive tests of chip functionality and that automated test equipment (ATE) was able to apply these exhaustive functional tests in a manner that matched expected chip operation and accurately distinguish between correct and faulty chips. Now, with the technology progressing to nano-scale, it is apparent that this approach does not hold any more and that major changes are underway in test methodology and application.

Test continues to be a major expense in the IC development and manufacturing chain, with up to 35 % of Non-Recurrent Engineering (NRE) costs attributed to test development and debug, while ATE cost per transistor is expected to remain flat [SIA00]. Figure 2.1 illustrates that test represents an increasing percentage of overall manufacturing cost of the device in the next decade.

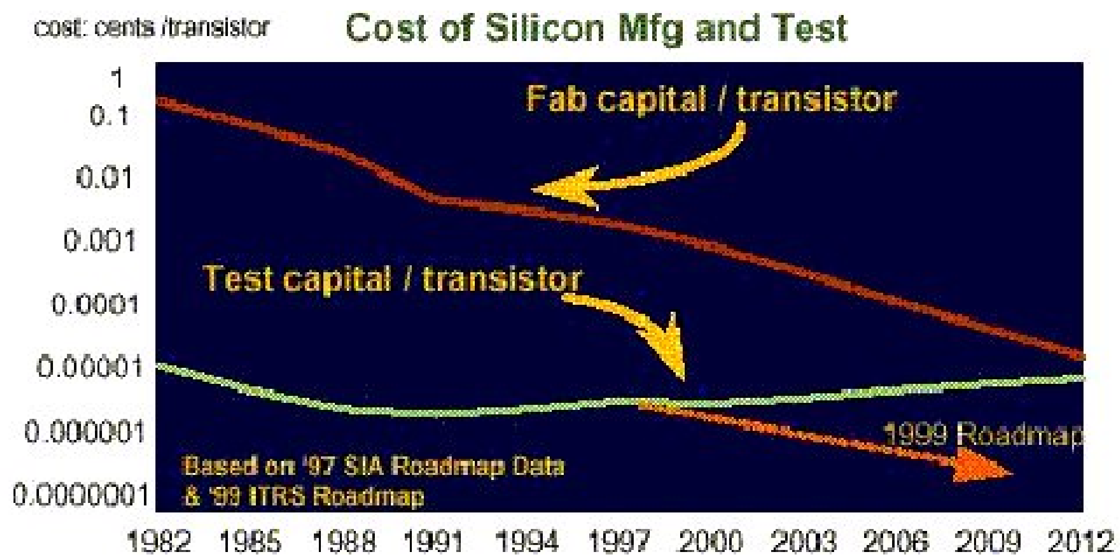


Figure 2.1. Test as a major cost concern in IC manufacturing [SIA00]

In addition, changing processes and design methods push testability beyond economic limits. Various test styles and methodologies evolved in the last period in order to improve the overall chip testability and test economics. Processors in general have primarily been tested applying functional test [ABR91]. However, as it became clear that the application and development of functional tests for processors turned out to be nonviable due to the high complexity of the modern processors, new alternative methods for complex digital systems testing had to be invented. The new methods include e.g.:

- Built-in self-test (BIST) will be needed as chip performance begins to outpace tester timing accuracy and as overall test data volume exceeds ATE and chip boundary capability.
- Divide-and-conquer techniques such as e.g. scan chains around hierarchical blocks supplemented with localized BIST.
- Core-based test is becoming widely used test-style and is a logic consequence of the Core-based design used to build complex SoCs.
- Functional test will still persist in certain form and will be used as a supplement test type to any of the above-mentioned techniques.
- Test methods targeting the faults derived according to the new fault models that became actual with the smaller feature sizes (e.g. delay test, IDDQ test etc.).

DfT must be tightly integrated into all steps of the overall design flow in order to enable the application of the test methodologies. In addition, DfT has to ease the Automatic Test Pattern Generation (ATPG) and achieve high fault coverage for all relevant fault types (such as stuck-at faults, timing, bridging). Of course, mixed-signal testing, yield improvement, failure analysis, signal integrity and electromagnetic phenomena are the issues that need to be tackled during the comprehensive test of a modern SoC design. However, they fall out of the scope of this thesis and will not be covered in the sequel.

This chapter is organised according to the following scheme. The test cost will be addressed in the second section of the chapter. It is necessary to give certain milestones in what determines the cost of the test, as the sequel of the thesis copes with the test cost

while introducing the test constraint at the high-level of the codesign. Section three tackles various approaches to test complex digital circuits. A few BIST approaches, interesting from the point of view for VLIW Application Specific Instruction Processors (ASIP), are shortly described. The fourth section addresses core-based test and the on-going P1500 standard. The chapter is concluded in section five.

2.2. Test cost

Apparently, the test cost issue becomes a hot-topic having in mind the fact that the cost of testing the circuit will become a predominant factor in the overall manufacturing and delivery process of modern ICs. Time-to-money, time-to-quality, and time-to-market are all significantly influenced by test. Hence, test has to support a cost-effective manufacturing process. This is particularly true for microprocessors, either general-purpose or dedicated types (such as VLIW-ASIP), as they become technology leader in the current situation of the market. Nevertheless, one has not to hesitate to spend considerable effort in terms of time and money while developing the test approach. The ultimate good is to ensure that the customer is happy, i.e., that a minimal number of faulty ICs is delivered to the final user. Hence, achieving the high quality of test has to be fulfilled in order to provide the success of the IC at the market.

In order to calculate the overall test cost, the following parameters need to be considered:

- test quality
- test application time
- DfT area overhead
- throughput penalty
- test development time
- pin-counts
- ATE cost.

Obviously, the analytical formula that would calculate the test cost of an IC is not feasible. Most of the parameters have a different nature and hence, cannot be easily compared, though some efforts such as [EVA99], have been done trying to combine some of the parameters into a Semiconductor Test Economic model. However, all of them need to be addressed early in the design phase, to avoid eventual redesign after the test strategy implementation. The above-mentioned parameters will be discussed in slightly more detail in the following subsections.

2.2.1. Test Quality

The test quality is usually measured according to the fault coverage achieved using the test method. The fault coverage is defined as the number of detected faults divided with the number of all considered faults within the circuits. Obviously, this ratio also depends on the assumed fault model, as the test vectors are generated having a particular type of faults as a target. The single stuck-at fault (SSF) has been the most popular fault model, widely used in the industry so far. Most of the semiconductor companies have been using ATPG tools producing the test vectors that target this type of faults. There is opinion that

the SSF will still remain in use in the industry as a dominant fault model in the 21st Century [PAN98]. However, this may not necessarily be the true, as the new modes of chip failure will inevitably come as a consequence of shrinking the transistor sizes. Even today, the stuck-at test vectors are usually supplemented with an IDDQ set of test vectors that target at detecting bridging faults. Memory blocks already employ multiple fault models requiring a comprehensive test-vector set where each set targets the faults from an individual fault-model group. There is a large variety of new fault-models, such as, bridging faults, transistor leakage, stuck-open, capacitive coupling faults, transmission line effects, gate delay, path delay, crosstalks, etc. In addition, design styles are also changing from standard static CMOS logic gates to circuits that are more dynamic and include pass transistor networks. In high-speed circuits signals waveforms do not resemble to clean logic 1 and 0 signals. Also, wires behave as transmission lines in high-speed circuits. All these phenomena require research in order to extend the set of currently used fault-models or to map the new failures mechanisms into the already existing fault models.

Another concern in the estimation of test quality is the acceptance of the level that is considered as sufficient for the fault coverage. High fault coverage is needed to make sure that the DPM (Defects Per Million parts shipped) level is satisfactory. The fault coverage level varies from one to another company. In general 100 % of fault coverage is not always possible to achieve as the redundancy can also be employed due to the functional reasons. We will adopt the fault coverage level of 98.5 % in the sequel of the thesis, as a boundary during assessment of the test quality of the proposed procedure.

2.2.2. Test Application Time

This parameter has grown in terms of its importance with pressing demands of time-to-market. It basically relates to the time required to execute the set of the test-vectors and is directly proportional to the number of test cycles. Complex digital circuits require often a large test-set. A large test-vector set implies a long test application time and therefore enhances the test cost per chip. Equally important is the fact that a particular IC can sometimes only be tested on expensive test equipment having also large pin memories to store test vectors. Because of these reasons, the size of test vectors is an important factor in the overall manufacturing process of high-volume ICs, and its reduction is a necessity. Minimization of test application time include several techniques such as test scheduling [CHA00], test compaction [KAJ98], optimisation of the test access path to embedded cores [AER98], reduction in number (and length) of scan chains [ZIV01] etc.

Directly related to the topic of test application is the power dissipation during testing. Exercising many nodes with a compacted test size and in a reduced test application time can result in burnout. Hence, the number of active transistors during test also needs to be taken into account while generating the test pattern set.

2.2.3. DfT area overhead and throughput penalty

Achieving high fault coverage is seldom possible without inclusion of circuitry to enhance the accessibility to certain chip areas. Each Design for Testability (DfT) technique implies a certain overhead of the chip die size, as the circuit is in that case equipped with parts that are used only for the testing purposes. Obviously, the DfT area overhead has a direct impact to the overall production cost of the chip. ICE corporation [THO96] calculated a cost per unit and estimated the incremental cost with 1% additional area on a side. The cost increase is \$1.27 per IC or as much as \$63.5 million per year

more. DfT area overhead for the microprocessor is even more critical than the DfT for most other high-volume products, as microprocessor volumes are enormous, and the die sizes are very large. Hence, it is not an easy task to calculate a good return on investement on DfT.

Furthermore, the DfT inclusion is tightly coupled to another concern – the performance degradation, i.e., the throughput penalty. For example, the insertion of DfT circuitry in a critical path of the component will definitely lead to a decrease of the throughput. Of course, this can make the circuit less appealing in the market and hence, the throughput penalty has to be kept minimal. Moreover, the total power consumption of the chip may also be increased due to the DfT features, an issue that also becomes hot-topic in modern low-power circuit design.

2.2.4. Test development time

Complex SoCs require a comprehensive test approaches and strategies, usually employing more than one person while developing and integrating the test set. Experience shows that roughly one third of the manpower in the design and test trajectory is spent on test development, programming and debugging. The test development time is usually referred as a Non-Recurring Engineering (NRE) [EBS01] cost. Also, the data management needs to be included in the total test development time. The introduction of standards in test can save time in the overall test development cycle. An example of such a test standard is e.g., the IEEE 1149.1 Boundary-Scan Test Standard [IEEE90]. Its purpose is to make the testing of boards incorporating chips easier and less costly. Boundary scan is a special form of scan-design, as boundary-scan cells are inserted between the input/output ports of the digital circuit under test and chip I/O pads. Another example is on-going IEEE P1500 core-based test standard [IEEE01] defining the test-access mechanism to embedded cores. However, neither of the two deals with the internal test of the component in question. Moreover, it looks like that one will never be able to introduce any standard for chip or core internals with given the ever growing development in semiconductor technology. Each application, built as a complex System-on-Chip demands peculiar test styles to be employed, developed and integrated. Therefore, test development is not a "push-button" approach and will probably never become [EBS01].

2.2.5. Pin-counts and ATE cost

The SIA International Technology Roadmap for Semiconductors (ITRS) shows that the test cost per pin for high-end ATE amounts to around $8K per pin. As the number of pins of advanced digital ICs exceeds 1000, one may easily derive the conclusion that the ATE cost can reach 10 million $. However, it is expected that future requirements resulting in higher performances and higher pin-counts will increase the ATE cost. Incorporating more DfT into an IC design is a possibility to reverse the trend of rising ATE costs and to enable the use of low-cost ATE for structural testing while achieving high fault coverage. Some semiconductor companies have even stated that introduction of low-cost DfT testers have even improved their test quality as they were enforced to enhance their DfT methods [PAN00]. Low-cost ATE is typically used for production testing, though it may also be used for characterization and diagnostic purposes.

One of the DfT techniques that would enable the lower ATE cost is the reduction in number of pins used during the test application. For example, an approach proposed in [VRA01] provides access to the internal scan-chains via IEEE 1149.1 compatible

boundary-scan chains performing serial/parallel conversion of test data, instead of direct access via the IC pins. Another approach toward the reduction in number of pins used for test is optimal balancing and distribution of scan chains combined with the use of several optional bypass circuitries [MAR00].

Developing a test method with optimal parameters is seldom (or never) feasible. Various trade-offs needs to be made in the process of test strategy implementation. In any case, the test strategy needs to be adopted early in the design process, and the design & test engineers should have in mind all above-mentioned factors that determine the overall test cost. The next section will discuss the test styles that may be used in modern digital design.

2.3. Modern digital test approaches

Major testing problems are appearing in all test areas because of the steadily increasing number of pins, frequencies, number of logic gates, and the introduction of analog/RF and memory in SOC devices. ATE manufacturers are struggling to provide equipment to meet these challenges. ATE costs are approaching $20M and wafer yields are beginning to fall because tester accuracy cannot be maintained at 5% of the test period as frequencies rise above 1 GHz. A common theme in most of the testing reports is the increasing need for DfT and BIST to avoid the problems being introduced as ATE technology fails to keep up with shrinking IC feature sizes. In the 1970s, ATE manufactured with bipolar ECL technology were capable of testing at frequencies an order of magnitude higher than the devices under test that were fabricated with 5 μm CMOS technology. In 1999, ATE is struggling to test high pincount devices at 1 Gbit/s, while off-chip data rates of an IC manufactured in 0.18μ technology are poised to advance up to much higher frequencies [SIA99]. Potential solutions therefore lie in areas of developing new DFT and BIST techniques for complex SoC.

The different approaches can be split into two distinctly different manners of testing:

- functional test
- structural test.

Functional test is based on the functional model of the system and considers the unit to be tested as a black-box. A functional model is, to a great extent, independent of the implementation. Therefore, the functional test can be used not only to check whether physical faults are present in the manufactured system, but also as design verification test for checking that the implementation is free of design errors. Its objective is to validate the correct operation of a system with respect to its functional specification. Another advantage of functional testing is that, usually, no extra testing hardware is required. However, functional tests that completely exercise fault-free behaviour are seldom applicable in practice due to the test length. The functional test uses a significantly shorter test than a pseudo-exhaustive test, taking into account some knowledge of the circuit structure.

On the other hand, a structural test is based on specific fault models and tests the proper construction and interconnections of each element in the structure. Therefore, a structural

tests check the netlist of the circuit and may be applied at each hierarchy level (using different fault models). Structural tests usually require DfT hardware, as this kind of testing cannot be done via the normal functional mode of the circuit.

It is very important at this point to separate the concept of functional verification testing from manufacturing test as those two have different objectives. While a functional test can be accomplished for design verification at the system-prototype level as well as for design validation, it is rarely adequate in production test of modern complex SoC designs. The reason for this is that functional tests do not test circuit structure in a complete way, as they are not based on the specific fault model. Hence, the functional test may not provide a sufficient level of fault coverage as required in IC manufacturing testing. A good structural solution comprising of efficient structural test method is what is needed for high-test quality of manufacturing test.

This thesis is concerned with implementation of structural testing for manufacturing faults in dedicated microprocessors. Hence, the testing will be tackled once the software and hardware parts are completed and the system prototype at high level becomes available.

2.3.1. BIST

BIST is the capability of circuits to test them self, and has become an alternative in IC testing. The concept itself is not new, there has been on-going research in this area during the past 30 years. In general, BIST provides various advantages and fulfils several requirements with respect to the test parameters that determine the test cost. Many BIST architectures have been proposed in the past, e.g. [BEN75], [KON80], [BAR87], [SAL88], [ABR91], [HEL96], [RAJ98], [KIE00], etc. All of them strive to find a trade-off with regard to the additional DfT area, number of test cycles and achieved fault coverage. However, the general principle of a BIST architecture has the generic form according to Figure 2.2.

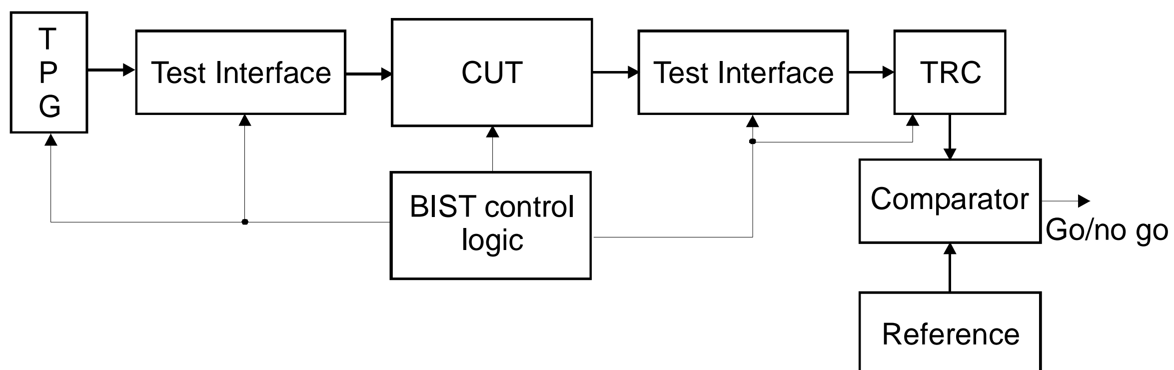


Figure 2.2. The general BIST template

The test vectors are applied to the Component Under Test (CUT) from the Test-Pattern Generator (*TPG*) and test responses are captured in the Test-Response Compactor (*TRC*) which output is being compared with reference values. The *BIST control logic* block controls the test sequence. Of course, there are different ways to implement the *TPG*, *TRC*, to store the reference signal, etc. Most implementations use a linear feedback shift register

(LFSR) as the generator of pseudo-random test stimuli and a multiple-input signature register (MISR) to compress the test response [ABR91]. However, using only pseudo-random test stimuli cannot guarantee sufficiently high fault coverage that would comply with industry demands; this is because the majority of the circuits is not completely random testable. Hence, the pseudo-random test-patterns have to be extended with deterministic test patterns using more sophisticated pattern generator. For example, it has been proven [HEL95] that combining random and efficiently encoded deterministic patterns can provide a complete fault coverage of non-redundant faults. That can be done in various ways: by insertion of test points into the CUT, by applying additional (external) deterministic test vectors, using a weighted random pattern generator [STR91], pseudo-exhaustive pattern generator [AKE85] and a deterministic pattern generator [KIE98]. Obviously, all these approaches require that a certain price in terms of the circuit area and number of test sequences has to be paid with respect to the achieved fault coverage.

Two advanced BIST schemes will be briefly illustrated in the sequel of the chapter, both of them trying to balance the additional silicon area, fault coverage and test vector size. One uses the conversion of pseudo-random test patterns into a deterministic set to test random logic. The other uses the concept of the so-called Soft BIST. The two approaches have serious potential in future industrial applications and our approach to test VLIW-ASIP has interference with the two approaches. Hence, they are worthwhile to be mentioned here in slightly more detail.

2.3.1.1 Deterministic Logic BIST

BIST for random logic (Logic BIST) has evoked its significance with recent advances in deep-submicron IC process technology, as external testing is becoming more difficult and costly. The use of logic BIST for digital cores and memory BIST for memory test, results in the advantage that a low-cost external tester is sufficient.

Several commercial CAT tools currently support logic BIST for industrial applications [MUK98], [HET99]. Deterministic logic BIST requires that the CUT is scanable, i.e., that the CUT contains inserted scan chains. Moreover, a scan chain (or bypass logic) has to be added around each circuit allowing all primary inputs to be controlled by the LFSR and all primary outputs of the CUT have to be connected to the MISR. In addition, all clock domains are required to run at the same frequency when logic BIST is active. The architecture has the form as shown in Figure 2.3.

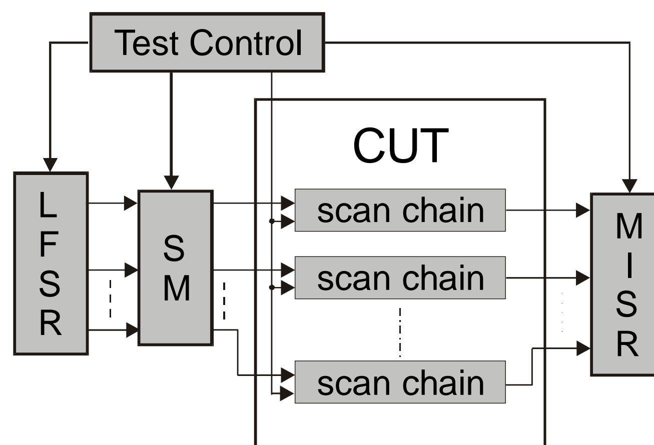


Figure 2.3. Deterministic logic BIST scheme

The pattern generator consists of an LFSR and a small combinational block called sequence modifier (SM). The SM block has to modify the pseudo-random patterns obtained from the LFSR into deterministic ones by changing certain bit positions. The sequence modifications provide that deterministic test patterns are generated and applied to the CUT via scan chains. The outputs of the scan chains are connected to a MISR, which compresses the test responses. A test control unit controls the test flow setting the LFSR, SM, MISR and scan chains into the appropriate modes.

The main concern with deterministic logic BIST is to provide the appropriate deterministic test patterns to the CUT. Hence, special attention has to be paid to a careful design of the SM block. That block can have the structure as proposed in [KIE00] and as shown in Figure 2.4. Here, the SM block contains a set of XOR circuits and the so-called Bit-Flipping Function (BFF). The BFF is constructed using an iterative algorithm that starts with an empty BFF and terminates when sufficient fault-coverage is achieved. The inputs of the BFF receive signals from the current state of the LFSR and the test control unit. If all BFF outputs are equal to *0*, the scan chains will be loaded with a sequence of pseudo-random test patterns. However, the BFF is designed in such a way that some ("useless") pseudo-random test patterns are converted into deterministic test-patterns obtained by applying ATPG to the CUT. The number of converted patterns is relatively small as observed in [WUN96], so that BFF will not be very complex.

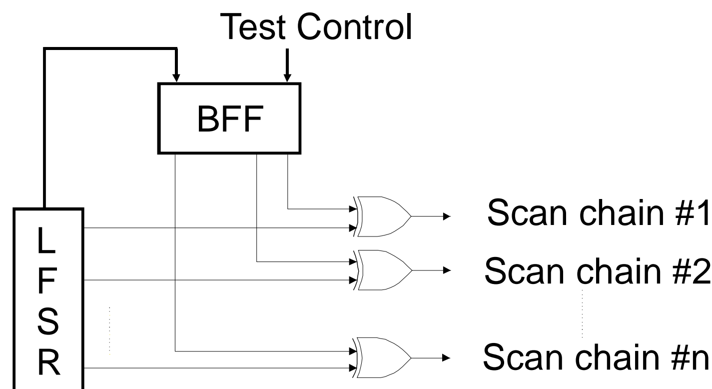


Figure 2.4. The sequence modification block

A CAT tool inserts the DfT circuitry required for logic BIST at the gate-level netlist of the CUT. After that, no further modifications of CUT and surrounding circuitry are required. The results of deterministic logic BIST application on industrial circuits [KIE00] have been performed on circuits with a complexity in the range between 2K and 90K gates. The reported DfT area overhead ranges between 5-35 %. A total number of 10000 pseudo-random test patterns has been applied to all circuits and it was embedded with 20-1000 deterministic test patterns, depending on the circuit size and inherent random-testability of the circuits. The total number of the test cycles is obtained after that the number of applied patterns to the CUT is multiplied with the length of the scan chains. That number can still be excessively large for a reasonable amount of the scan chains within the circuit, increasing in that way the overall cost of the chip.

2.3.1.2. Soft BIST

Embedded systems with processors (either dedicated or general purpose) can be used to run software routines for the purpose of testing. Namely, the software program may contain the test patterns of all blocks of the embedded system. Consequently, the test responses may also be stored into the embedded data memory using the test programs. The test patterns may be either pseudo-random or deterministic. If the pseudo-random patterns are used, the structure of the test program can be kept very simple, as some elementary processor instruction can be used. Moreover, even a LFSR can be emulated very efficiently in software to generate a fixed number of state transitions.

As already explained, pseudo-random patterns usually have to be supplemented with deterministic patterns in order to achieve sufficient fault coverage. These patterns can also be applied using the software routines or a hardware-based deterministic BIST scheme can be emulated by the test software [HEL95]. In general, the Soft-BIST scheme has a form as depicted in Figure 2.5. It shows the general Soft-BIST scheme without any assumption with regard to the nature of the generated test patterns.

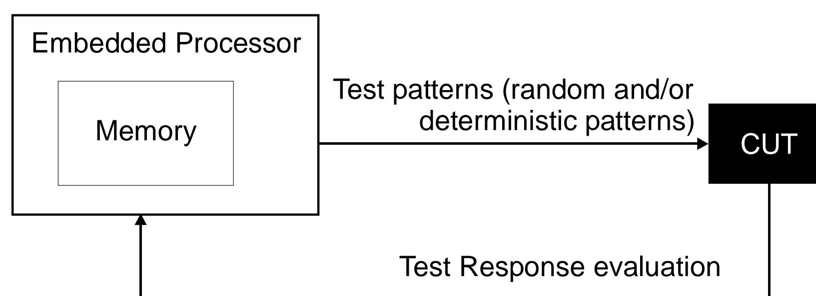


Figure 2.5. The general Soft BIST scheme

Usually, pseudo-random test pattern generation requires long run times of the software routine that generates the sequence. On the other hand, the deterministic test patterns may impose high memory requirements, depending on the test vector size. There are various approaches tending to combine the two test pattern generation procedures and find a balance, such as e.g. in the mixed-mode BIST [HEL96]. The method use software routines to implement a random test scheme (emulating multiple-polynomial LFSR) and a reseeding scheme for the deterministic test patterns. The deterministic patterns are software encoded, based on reseeding of the multiple-polynomial LFSR. The test program has run on an Intel 80960CA processor and achieved a fault coverage of 100 % for several benchmark circuits for a random sequence of 10000 patterns. The memory requirements have been reduced applying this approach. Nevertheless, the test time can still be significantly long, especially in the case when CUT has long scan chains. Moreover, the proposed approach is not suited for manufacturing test of the embedded system as it assumes that the embedded processor and its memory are fault-free. However, in an embedded complex SoC design, that assumption is not valid.

The above-presented methods can be used to test embedded ASIP-VLIW codesign processor. Nevertheless, it is not that easy to predict the impact of BIST insertion at high-level of codesign. The approach that we tend to propose differ from the previous two,

though our approach does not exclude the test access mechanism of the two BIST approaches. This will be explained in fourth and fifth chapter.

2.4. Core-based testing

Sub-micron technologies allow the integration of a complex functions in a single chip. SoCs offers advantages such as better performance, lower power consumption and smaller volume and weight, when compared to their traditional multi-chip/PCB counterparts. SoCs are typically heterogeneous, containing modules of different functionality such as random logic, memories in various flavours, mixed-signal and analogue blocks. To efficiently use design resources and improve time-to-market, there is a trend to embed reusable (parameterised) versions of large modules, so-called *cores* (or IPs, modules, blocks). Examples of cores are e.g., CPU's, DSPs, MPEG and JPEG modules, memories, A/D, D/A converters, PLLs, etc. Cores might come as hard (layout), firm (netlist) or soft (register-transfer level – RTL) descriptions. Core-based design divides the IC design community into two groups: *core providers* and *core users*. The entire product creation process for core-based designs consists of three steps: IP development, core customisation and core integration and packaging. The core-based design takes place both within the companies as well as between different companies.

These complex SoCs, comprising of millions of transistors, require a comprehensive structured solution for DfT and test-pattern generation. Namely, the various circuit modules performing different functions may require a particular test style. For example, the ROM and RAM modules depicted in Figure 2.6 in a typical SoC environment have to be tested with several memory test algorithms. MPEG, UDL (User Defined Logic) and other combinational and sequential blocks can be tested with e.g., logic BIST or full-scan. On the other hand, analogue and mixed-signal blocks (PLL, A/D, D/A converters) require mixed-signal testing. Therefore, the only solution for testing such a design is to apply divide and conquer approach: the various modules are divided and placed into different groups determined upon the test style that will be used for their test.

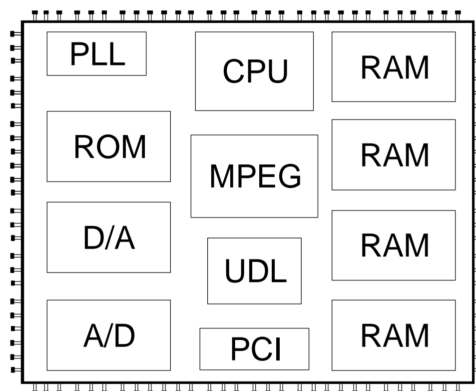


Figure 2.6. A typical SoC design

The core user is responsible for manufacturing and testing the entire chip, i.e., not only the interconnection between the cores, but also the cores themselves. Test development for the cores requires knowledge of the core internals, which, typically, a core user does not have available. The core user wants to use the core, without spending too much time and

efforts on the core implementation. Therefore, the assumption must be taken into account that the core provider will deliver with the core design itself a set of tests for the core. In that case, the reduction of time-to-market for new versions and redesigns is feasible not only by reusing core designs, but also core DfT and test patterns. This statement and integration of test patterns of various blocks of SoC design as well as interconnection test in such environment has lead a new test style to evolve – *core-based* testing.

The cores are tested as stand-alone units in which the core user has to assemble a chip-level test out of the pre-defined tests for the various cores and additional tests for non-core chip circuitry. However, there is a crucial difference between the embedded core test and the individual chip case. As became clear from the previous text, the core is just an element of the complete SoC design, provided to the customer, i.e., core user. He is free to use a core in a manner as he wants, implement and integrate it with other cores and then manufacture the complete system. That is fundamentally different than selling already manufactured chips. With an embedded core, it becomes necessary to fully consider the test type, test description and accompanying test vectors. This brings several new challenges [MAR99a]:

- Core-internal tests

  This feature addresses the harnesses in generating an efficient test pattern set suited for all faults from the set of specified fault models. As already explained, deep sub-micron and nano-technology requires new failure mechanisms and fault models to be investigated. Consequently, the traditional ATPG tools need to be extended with the new tools targeting the faults from these new groups of fault models.

- Core-test knowledge transfer

  The total chip-test development is a distributed effort requiring the information transfer from core provider to core user including test-patterns (eventually core internal DfT as well), test modes and corresponding test protocols, fault coverage, etc.

- Test access to embedded cores

  A core is typically deeply embedded in the system-chip design. In order to run its tests that are defined at the core terminals, a Test Access Mechanism (TAM) needs to be defined, as shown in Figure 2.7. It consists of test source that generates the test stimuli (implemented either by off-chip ATE or on-chip BIST) to the core inputs, a test sink (also either off- or on- chip) that evaluates the test responses and a *wrapper* to connect the TAMs to the core [ZOR98]. The wrapper is a thin shell around the core and provides the switching between normal functional access and test access to the core via the TAM. The wrapper has to be able to wrap all kinds of cores with potentially different test access requirements for core-internal as well as core-external testing.

- Test integration and optimisation

  The core user has the responsibility to expand the core internal test-patterns into chip level tests. The overall set of test patterns at the chip level has to cover all cores

within the SoC design, user defined logic (UDL) and interconnections, and glue logic in between. Hence, the core user has to deal with many optimisation issues and carry out various trade-offs with respect to the factors that determine the test cost. If the SoC has been designed within one company, the test pattern integration can be done with a team of test engineers in parallel as soon as the design of an individual core has been finished. This saves time at the end of the design project and avoids the detection of DfT problems in the last phase of the design.

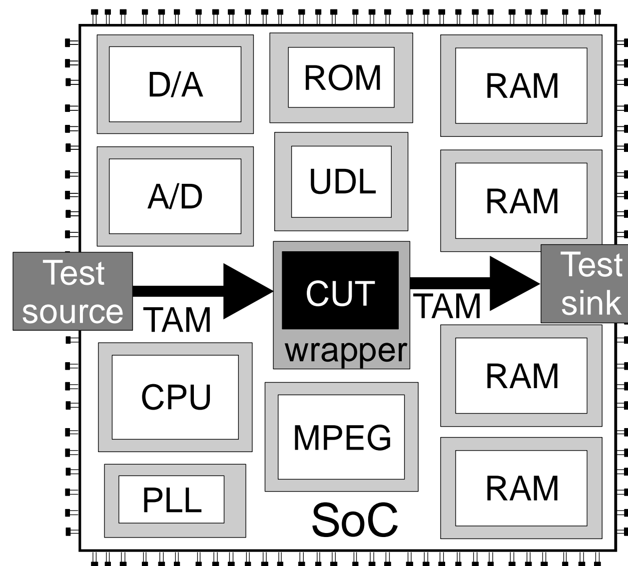


Figure 2.7. The access architecture for embedded cores

Improving the efficiency of both core providers and core users will facilitate the interoperability with regard to testing if cores from different sources come to one SoC (so-called plug-and-play), requires a high degree of standardization in a SoC environment [ZOR97]. In June 1997, the IEEE Standard Board granted permission to start a standard activity in profiling the *Standard for Embedded Core Test* called *IEEE P1500* [IEEE01]. Several semiconductor companies are actively participating in the on-going discussion that should result in a final approve of the standard by the IEEE board by the end of 2001. P1500 does not deal with core-internals test method or DfT, nor SoC test integration and optimisation. These issues are not suited for standardization as their requirements differ for different cores and SoC's. Instead, P1500 focuses on standardizing the interface between core providers and core users (core test knowledge transfer) and these are the test access to embedded cores and core test knowledge transfer.

The test access standardization to embedded cores deals with the wrappers, only. It was decided not to standardize source, sink or TAM. They are completely left in the hands of the core user. The wrapper, although standardized, still retains flexibility [MAR99b] because its architecture is scalable, depending on the width of the TAM. In general, the number of TAM outputs does not have to be equal to the number of core input terminals and likewise holds for TAM inputs and core outputs. In such cases, the wrapper may provide width adaptation through serial-parallel and parallel-serial conversion. The controllability of the core internal test is provided from the TAM outputs connected through the wrapper to the appropriate core input ports. The observability is obtained from the appropriate core output terminals to the TAM inputs connected at the wrapper boundary. On the other hand, controllability of the core external test is obtained from the

TAM outputs connected to the wrapper functional outputs, and observability is provided via the wrapper functional inputs connected to the TAM inputs at the wrapper boundary. The design of wrapper content and its interface circuitry will be elaborated in more details in next chapter, which will, among other things, illustrate one implementation of scalable, flexible wrapper design.

Capturing and expressing test-related information for reusable cores is done using a Core-Test Language (CTL). CTL is a part of the standard intended to maintain the core test knowledge transfer [KAP99]. CTL describes aspects of the test data (such as type of data, rate of data, stability of values), test mode information, connectivity information (such as attributes and protocols), test methodology related information and connectivity information. The reader can find more details of the P1500 standard and its current update at its website [IEEE01]. Related to this item is also a Virtual Socket Interface Alliance (VSIA) that also discusses standardization on core-based design and test [VSI00].

The previous introduction to core-based testing was necessary, because according to the technology trends, there is a realistic expectation that VLIW ASIP will be designed and implemented in a core-based environment. The VLIW ASIP can be designed as a SoC or even more probable, the VLIW ASIP will be a part of larger SoC design. Either way, the design & test engineer of VLIW ASIP (in core based design, he is in role of the core provider) has to have in mind the basic aspects of core-based test while developing the test strategy. Of course, if the VLIW ASIP is one core within a complex SoC, its internal test does not depend on the surrounding circuitry, as the core-based test should provide an infrastructure for test stimuli application and test response evaluation independent on the test style.

Chapters 4 and 5 of this thesis address the test development of two particular VLIW ASIP processors that can be used in a core-based environment with the emphasis put on their internal test generation and optimisation. This is justified, as in the typical case, the core internal circuitry is much larger than the circuitry that is used to interconnect the cores. This is particularly true for large designs such as ASIP-VLIW processors. Therefore, the test data volume involved in core-internal testing is much larger than the test data set for a core external test. Moreover, in many cases, the core provider designs the wrapper at the time when the environment where the core will be used is not known yet, and hence, the test patterns related to core-external test. Therefore, we have given the priority to test synthesis of efficient test pattern generation striving to satisfy as many as possible the criteria that determine the low cost of the test.

2.5. Conclusion

This chapter has introduced emerging trends, problems and solutions and developments of modern digital test techniques. Testing is becoming a dominant factor that cost most of the time and money during the high-volume production of modern ICs organized as complex Systems-on-Chip (SoC). The factors that determine the test cost have been identified and explained. These include the test quality, i.e., high fault coverage, test time, DfT area overhead, throughput penalty, test-development time, number of pins used for test and ATE use. The test styles in modern digital design have been briefly discussed. A few techniques striving to find the optimal balance of the cost parameters are have been indicated. Advanced BIST schemes seem to be very promising with respect to a high test-quality. The basis of Core-based testing is also presented, as System-on-Chips will also

need to be tested in an embedded-core environment. The development of the test strategy for manufacturing test of VLIW ASIP will be inevitably constrained with all these emerging trends and problems in complex SoC design.

Literature:

[ABR91] M. Abramovici, M. A. Breuer, A. D. Friedman, *Digital Systems Testing and Testable Design,* Computer Science Press, New York, 1991

[AER98] J. Aerts, E. J. Marinissen, "Scan Chain Design for Test Time Reduction in Core-Based Ics," *Proc. of the International Test Conference,* October 1998, Washington D.C., pp. 448-457.

[AKE85] S. B. Akers, "On the use of Linear Sums in Exhaustive Testing," *Proc. of the IEEE International Symposium on Fault-Tolerant Computing*, 1985, pp. 148-153.

[BAR87] P. Bardell, W. H. McAnney, J. Savir, *Built-in Test for VLSI*, Willey-Interscience, New York, 1987.

[BEN75] N. Benowitz, D. F. Calhoun, G. E. Alderson, J. E. Bauer, C. T. Joeckel, "An Advanced Fault Isolation System for Digital Logic," *IEEE Trans. on Computers*, vol. C-24, No. 5, May 1975, pp. 489-497.

[CHA00] K. Chakrabarty, "Test Scheduling for Core-Based Systems using Mixed-Integer Linear Programming," *IEEE Trans. on Computer-Aided Design of IC and Systems*, vol. 19, No. 10, October 2000, pp. 1163-1174.

[EBS01] Evening Breakout Session, "Low-Cost DfT Enabled Testers – Why Haven't We Used Them Before?," *The IEEE European Test Workshop (ETW 01)*, May-June 2001, Stockholm, Sweden,

[EVA99] A. C. Evans, "Applications of Semiconductor Test Economics, and Multisite Testing to Lower Cost of Test," *Proc. of the IEEE International Test Conference,* September 1999, Atlantic City, NJ, pp. 113-123.

[HEL95] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, B. Courtois, "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Trans. on Computers*, vol. 44, No. 2, February 1995, pp. 223-233.

[HEL96] S. Hellebrand, H.-J. Wunderlich, A. Hertwig, "Mixed-Mode BIST Using Embedded Processors," *Proc. of the IEEE International Test Conference*, Washington DC., October 1996, pp. 1-10.

[HET99] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kasab, A. Hassan, J. Rajski, "Logic BIST for Large Industrial Designs: Real Issues and Case Studies," *Proc. of the IEEE International Test Conference,* September 1999, Atlantic City, NJ, pp. 358-367.

[IEEE90] IEEE Computer Society, *IEEE Standard Test Access Port and Boundary-Scan Architecture – IEEE Std 1149.1 – 1990*, IEEE, New York, 1990.

[IEEE00] IEEE P1500 Web Site, http://grouper.ieee.org/groups/1500/.

[KAJ98] S. Kajihara, I. Pomeranz, K. Kinoshita, S. M. Reddy, "Cost effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits," *Proc. of the IEEE Design Automation Conference*, June 1998, pp. 102-106.

[KAP99] R. Kapur, "P1500-CTL: Towards a Standard Core Test Language," *Proc. of the IEEE VLSI Test Symposium (VTS),* Dana Point, CA, April 1999, pp. 489-490.

[KIE98] G. Kiefer, H.-J. Wunderlich, "Deterministic BIST with Multiple Scan Chains," *Proc. of the IEEE International Test Conference*, Washington DC., October 1998, pp. 1057-1064.

[KIE00] G. Kiefer, H. Vranken, E. J. Marinissen, H.-J. Wunderlich, "Application of Deterministic Logic BIST on Industrial Circuits," *Proc. of the IEEE International Test Conference,* October 2000, Atlantic City, NJ, pp. 105-114.

[KON80] B. Konemann, J. Mucha, G. Zwiehoff, "Built-In Test for Complex Digital Integrated Circuit," *IEEE Journal of Solid State Circuits*, vol. SC-15, No. 3, June 1980, pp. 315-318.

[MAR99a] E. J. Marinissen, Y. Zorian, "Challenges in Testing Core-Based System ICs," *IEEE Communication Magazine*, vol. 37, No 6, June 1999, pp. 104-109.

[MAR99b] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, L. Whetsel, "Towards a Standard for Embedded CoreTest: An Example," *Proc. of the IEEE International Test Conference,* September 1999, Atlantic City, pp. 616-627.

[MAR00] E. J. Marinissen, S. K. Goel, M. Lousberg, "Wrapper Design for Embedded Core Test," *Proc. of the IEEE International Test Conference*, October 2000, Atlantic City, NJ, pp. 1119-1122.

[MUK98] S. Mukherji, L. Nguyen, D. Burek, S. Baird, "IP/VC-Based Test Methodology (Part 1): A Case Study," IEEE Digest of Papers International Workshop on Testing Embedded Core-Based Systems, October 1998, pp. 1.2-1.9.

[PAN98] Panel session, "Stuck-at Fault: The fault model of choice for the third millennium!?," *Proc. of the IEEE International Test Conference*, Washington DC., October 1998, pp. 1165-1167.

[PAN00] Panel session, "DfT-focused chip testers: what can they really do?," *Proc. of the IEEE International Test Conference*, October 2000, Atlantic City, NJ, pp. 1119-1122.

[RAJ98] J. Rajski, J. Tyszer, *Arithmetic Built-In Self-Test*, Prentice-Hall PTR, 1998.

[SAL88] K. K. Saluja, R. Sharma, C. R. Kime, "A concurrent Testing Technique for Digital Circuits," *IEEE Trans. on Computer-Aided Design*, vol. 7, No. 12, December 1988, pp. 1250-1259.

[SIA99] SIA semiconductor industry association, "The National Technology Roadmap for Semiconductors," 1999 edition.

[SIA00] SIA semiconductor industry association, "The National Technology Roadmap for Semiconductors," 2000 update.

[STR91] A. Strole, H.-J. Wunderlich, "TESTCHIP: A chip for weighted random pattern generation, evaluation and test control," *IEEE Journal of Solid State Circuits*, vol. 26, No. 7, 1991, pp. 1056-1063.

[THO96] K. M. Thompson, "Intel and the Myths of Test," *IEEE Design & Test of Computers*, Spring 1996, pp. 79-81.

[VRA01] H. Vranken, T. Waayers, H. Fleury, D. Lelouvier, "Enhanced Reduced Pin-Count Test for Full-Scan Design," *Proc. of the IEEE European Test Workshop (ETW 01)*, May-June 2001, Stockholm, Sweden, pp. 155-163.

[VSI00] VSI Alliance Web site: http://www.vsi.org/

[WUN97] H.-J. Wunderlich, G. Kiefer, "Bit-Flipping BIST," *IEEE Proc. of the International Conference on Computer-Aided Design*, 1997, pp. 337-343.

[ZIV01] V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff, “An Implementation for Test-Time Reduction in VLIW Transport-Triggered Architectures,” *Proc. of the IEEE European Test Workshop (ETW 01)*, May-June 2001, Stockholm, Sweden, pp. 255-262.

[ZOR97] Y. Zorian, “Test Requirements for Embedded Core-Based Systems and IEEE P1500,” *Proc. of the IEEE International Test Conference*, Washington DC., November 1997, pp. 191-199.

[ZOR98] Y. Zorian, E. J. Marinissen, S. Dey, “Testing Embedded Core-Based System Chips,” *Proc. of the IEEE International Test Conference*, Washington DC., October 1998, pp. 130-143.

CHAPTER 3

Core-Based Testing in PI-Bus Based Processor Design

3.1. Introduction

In order to become competitive in today's highly demanding electronics market, the worldwide-leading IC suppliers and EDA vendor companies constantly add new functionality, increase performance and reduce costs of new products. These, often conflicting, goals can be achieved by exploiting the increase in semiconductor process technology as defined by Moore's Law. Hence, driven by the high complexity of modern Systems-on-Chip (SoC) and reduced time-to-market, *core-based* design is becoming a widely accepted design style in the electronic industry community. Of course, there is a need for standardization reflected by:

- The establishment of an efficient communication among the cores or between the cores and the outside world.
- The cores have to incorporate standard views in order that EDA tools can handle them efficiently.

- EDA tools must properly support the core-based design and test flow.

These three requirements have also been acknowledged by industry and there is an effort to standardize these requirements via the Virtual Socket Interface Alliance (VSIA) [VSI00]. Many important companies are participating in VSIA, whose task it is to establish a unifying vision for the System-on-Chip industry and the technical standards required to enable the most critical part of that vision: the mix and match of Virtual Components (IP, cores) from multiple sources. Much like standardized physical components, which are incorporated on a printed circuit board, IP in standardized "Virtual Component" forms need to be rapidly mixed and matched into system chips. In order to facilitate this, VSIA specifies "open" interface standards, which will allow Virtual Components to fit quickly into "Virtual Sockets", at both the functional level (e.g., interface protocols) and the physical level (e.g., clock, test, and power structures). This will allow core providers to product and maintain a uniform set of IP deliverables, rather than have to support numerous sets of deliverables required for the many unique customer design flows. Most semiconductor companies are participating in VSIA giving their contribution in establishing the various standards. For example, Philips Semiconductors has developed an on-chip standard, called Peripheral Interconnect (PI) bus [ARE98]. It allows an easy and efficient integration of various cores within a flexible architectural framework, essential for SoC design. A simplified view of the PI-bus based architecture is shown in Figure 3.1.

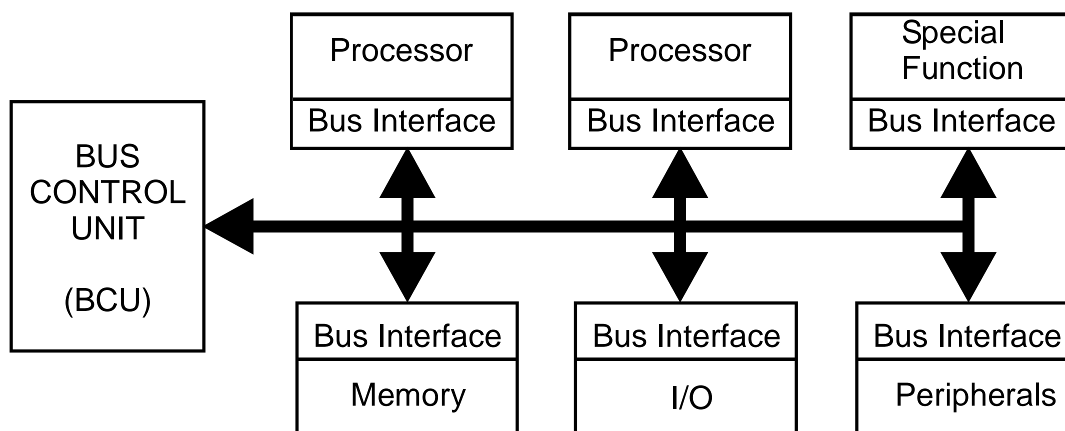


Figure 3.1: Generic architecture overview of a PI-Bus based system.

A complex system can be more easily built by combining the various modules performing the required function. The cores are allowed to be designed independently of each other, which is crucial for core-based design. The interfaces, where a master and a slave type can be distinguished, and communication protocols are standardized. A master can initiate communication via the PI-bus by issuing operations on the bus once the Bus Control Unit (BCU) has granted bus access to the master block. A slave responds to bus operations as soon as it is selected via its address on the PI-bus. Typical cores that may appear within these types of architectures are processors (RISC, CPUs, DSPs, VLIWs), memory and I/O interfaces, peripheral functions (UART, counters, timers), and system specific functions, like e.g. MPEG decoders, data compression, data encryption, etc., or simply User-Defined Logic (UDL). The cores do not necessarily have to be designed by

third parties; often, semiconductor companies want to develop their own cores by their design teams.

Since core-based design within this architecture is quite straightforward, the same must hold for its production test. However, the main concern of VSIA is not to specify the test concept in core-based environment. Hence, an appropriate core-based test strategy has to be developed being capable to handle the test in an arbitrary embedded environment. The IEEE 1500 [IEEE00] is a standard-under-development focused to ease of reuse and interoperability with respect to testing the cores in an embedded environment.

The remaining part of this chapter focuses on this issue and is based on the traineeship the author has carried out at the Philips Semiconductors ASG ESTC group in Eindhoven. It is organized as follows. The second section introduces the basics of the core test strategy as proposed by the Philips Core Test Action Group (CTAG). It also deals with the DfT concept in an embedded core environment and explains the problem regarding test-pattern generation of embedded cores. The DfT implementation fulfilling the requirements of the DfT concept is outlined in the third section. The section four proposes a Computer-Aided Test (CAT) integration flow applicable in an arbitrary embedded environment and tuned to the production test. Special attention is paid to the problem of realistic fault coverage determination of an embedded core within this flow. The CAT flow itself is illustrated in section five and applied to generate the test patterns and determine the fault coverage of the cores within the Philips Core-Test pilot project. The design used the PI-bus architectural framework from Figure 1. These results are also reported in the fifth section. Finally, the conclusion and required future work in this area is given in the last section. The experience gained has been used to develop the methodology for testing the architecture in the MOVE codesign framework.

3.2. The CTAG Testing Strategy

3.2.1. Core testing

As has already been mentioned, the major reason for core-based design and test is a reduction of the time-to-market. Hence, the application of the core-based style will be only justified if the time and effort required to develop a working IC are much shorter than the development from scratch. Since the cores have to be reusable (so called “plug and play”) in different environments, a certain level of standardization between the core provider and the core user is required. This also relates to their test. Since the standardization of P1500 is not completed yet, Philips established its own standardization within CTAG (Core Test Action Group) targeting the test in an embedded core environment. The test in core-based design style within Philips has evolved from the Macro test concept [BEE86, BEE95, MAR97]. All activities of CTAG are highly compliant with the IEEE P1500 on-going discussions towards a standard in core-based test.

Since the time-to-market reduction is the principal factor in core-based design, the pre-computed tests for manufacturing defects are attached to the cores in order not only to shorten the design time, but also the test-development trajectory. A sound core-test strategy has to fulfil the following requirements [MAR98]:

1. Reuse of pre-computed tests as delivered with the core by the core provider.
2. Easy integration with respect to arbitrary test-style.

3. Cost efficiency with respect to commonly adopted factors such as extra DfT area, number of pins, test time, throughput deterioration, power consumption, etc.

The cost efficiency during the design phase could be contradictory with the basic idea of the core-based design style: time-to-market. Therefore, various tradeoffs in that sense are necessary. However, core-based design is driven by time-to-market motives, and hence, this requirement should prevail. An area or performance-driven project has to choose another design style, not the core-based one. An ideal core-based test strategy is one that provides the flexibility and space to make trade-offs between the various requirements and ability to tackle the new challenges such as:

- The test-data interface between the core provider and the core user, i.e., a description of all test data aspects of the core such as test protocols, test patterns, test modes etc.,
- Design for testability techniques adapted to this new design style,
- Automatic test expansion of core-level test into IC-level tests,
- IC-level test scheduling of the various core tests in order to minimize the test time.

Also, a core-test strategy has to support:

- All types of core tests such as functional test, scan test, BIST, memory test, IDDQ test, interconnection tests, etc.,
- Hierarchy in every form since the cores very often contain multiple modules of various circuit structures that require different tests,
- Provisions for testing systems with multiple-clock domains,
- Clock-skew tolerant design,
- Cost effectiveness (in terms of silicon area, number of pins, test time, power consumption).

One can describe the basic steps in core-based test in the following way:

1. A test of an actual core is split into test patterns and a test protocol. The test protocol describes the way in which the actual test patterns detecting the faults within the core are to be applied to the inputs, outputs and I/O pins of the core under consideration. Also, the testing of the interconnections as well as the glue logic between the cores is carried out via the test patterns defined in the test protocols.
2. Translation of the core-level tests into the IC-level tests is realized by means of the test protocol expansion in the CAT tool [MAR97]. The tool expands only the test protocols, not the actual test patterns. This tool also supports multiple levels of hierarchy in a design.
3. Test scheduling, in order to test as many cores as possible at the same time (in parallel), is applied. This step is optional and depends on the actual design, i.e., which type of test-access mechanism is employed.

It is worthwhile to mention at this point that there are strong similarities between ICs on a printed circuit board and cores (virtual components) within an IC. Core-based design

can be considered as the next step in on-chip integration. This is because state-of-the-art IC's are tomorrow's cores. The most popular and commonly accepted worldwide standardization for board-test is IEEE 1149.1 ('JTAG') [IEEE90]. The main task of this standard is testing the interconnections between IC's (components). It defines the additional hardware for the IC, in order to solve the problem of access and test protocols. However, despite all similarities, major differences between the two standards are expressed with the next two issues:

- The design and test engineers have already tested the IC's (components) at board level for manufacturing. Thus, they can be expected to be fault-free. However, this is not the case for the cores, since the core supplier does only deliver the description of cores in hard, firm or soft form. Therefore, the cores are not yet manufactured and the core provider cannot test them for manufacturing faults; it is a matter for the core user to carry out that task. Hence, testing of the interconnections between the cores is of the same importance as is the testing of the internals of the core.
- Inherent to boundary scan test lies the reduction of the number of extra pins requested for test. This is not the case for the cores since these are virtual components so they have only virtual pins. One is (more or less) free to use as many virtual pins as required.

Therefore, a boundary scan-test has limited capabilities when applied in the core-based environment, primarily because of the access problem. The core-test standard requires different solutions and many research has been performed in both industry and academia to overcome this problem, such as e.g. [GHO97, WHE97, BHA98, IMA90, VAR97, JAS98]. This work exploits the test access mechanism using the so-called *test shells* [MAR98] as wrappers around the cores, as described in previous chapter. This approach is also becoming a substantial part of the future IEEE P1500 standard. The concept of the test shell and its influence on the overall fault coverage will be explained in the next subsection.

3.2.2 Test shells and their influence on test-pattern generation

In order to ease the testing of embedded cores, the CTAG introduces the concept of the test shell [MAR98]. The test shell, which can be considered as DfT circuitry, is wrapped around every core. The test-shell introduces the three levels of hierarchy: the IP module, the test shell and the host (IC, chip). In the remaining part of the thesis, the inner part of the core will be named as an IP - Intellectual Property blocks, while the term core will be used for an IP equipped with the test shell. This situation is shown in the example as illustrated in Figure 3.2.

The core provider delivers the design of the test shell. The role of the test shell is to ensure a proper execution of the pre-computed test patterns for each IP module surrounded with that test-shell and in addition to provide the infrastructure to test the interconnections and glue logic between the IP modules. No assumptions are made with regard to any particular test method or DfT technique that is going to be used for the test of the cores. This allows the handling of the IP modules that do not contain any DfT structure but only predefined structural or functional tests, as well as the IP modules with DfT structures such as scan or built-in self-test. In addition, it is up to the test shell to facilitate the test-expansion task for the core user up to the IC level, (in the sequel of the thesis also – host or chip level), and to enable the execution of the actual test patterns of the IP by creating a

standardized interface. The test shells are connected to each other via the test access mechanism (TAM), which is defined at the host level. The TAM transports the stimuli from the source host connectors to the core under test (CUT) and from the core to sink host connectors to evaluate the response. The design of the TAM is left to the system chip designer. Now, the test development consists of generating the IP test, expanding all IP-level tests to the host-level, introducing the interconnect test and finally, the test scheduling of various cores at host level. The test shells (and the TAM) have to be constructed in such a way that the test expansion becomes straightforward.

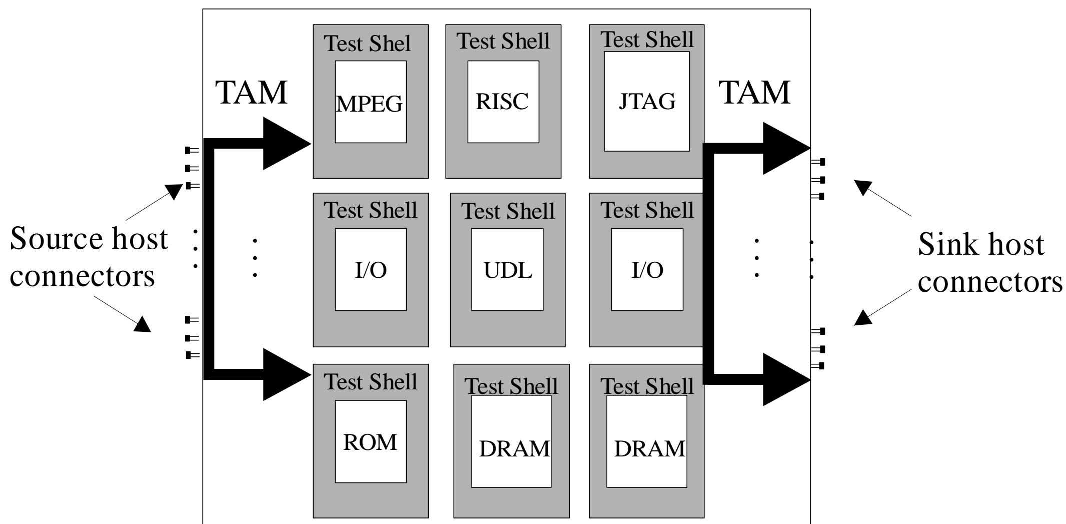


Figure 3.2: The test shells equipping the IP’s in an embedded cores environment.

The external connectors of the test shell (core-to-chip interface) in embedded cores environment are depicted in Figure 3.3.

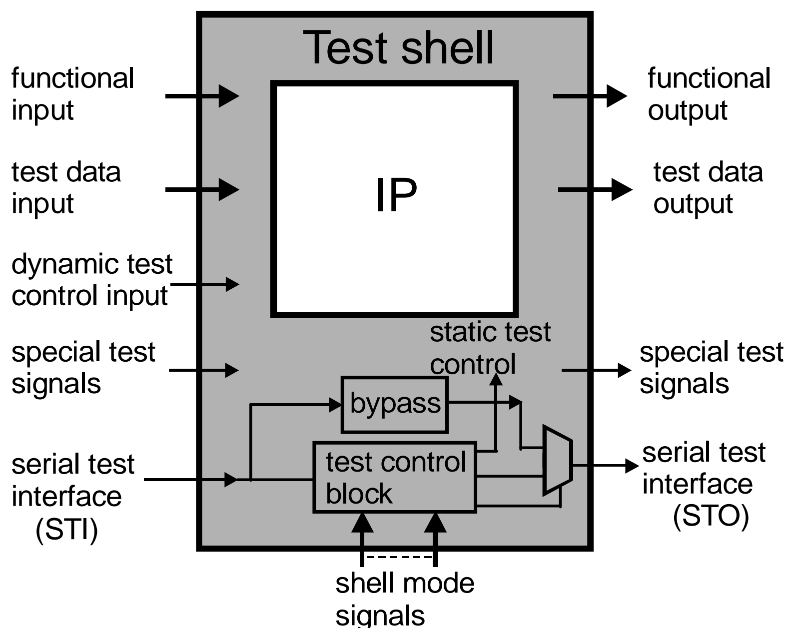


Figure 3.3: External test shell connectors.

As shown in this figure, there are six types of basic external connectors:

- Functional inputs/outputs correspond to the normal functional inputs and outputs of the IP module.
- Test-data input/output serve for the test-data transport for all synchronous digital tests via TAM, such as scan tests, memory tests, built-in self-tests, functional tests (this consideration holds for BIST and functional test that employ synchronous signals), etc. Their width is scalable, i.e., number of inputs and outputs may be changed due to the factors such as host pins, test time and silicon area. However, it has to be fixed upon instantiation of the specific test shell.
- Dynamic test control inputs such as the *scan enable* signal.
- "Special" test signals that cannot be transported via the test data input/outputs such as test clocks or non-synchronous signals (analog signals and all kind of asynchronous signals).
- Serial test interface (*STI* and *STO*) to support IC-level test and debug and to shift the instruction in and out of the test control block.
- Shell-mode signals to determine the operational mode of the test control block.

Figure 3.4 shows the interface between the test shell and IP block itself and a conceptual view of the realization. An actual implementation of the test shell varies from one IP to another. It also must cope with the design constraints and leaves space for optimisation. One may distinguish the following signals:

- Functional input and output signals (*d_in*, *d_out*).
- Test-data inputs and outputs, e.g. inputs and outputs of core-internal scan chains such as *s1, s2, r1, r2*.
- Dynamic test control signal, e.g. scan-enable, not shown in this figure.
- Static test control signal, e.g. BIST-enable, signals determining the test shell mode *m1* and *m2*.
- Special test signals, the same as in the external test-shell interface, not shown in this figure.

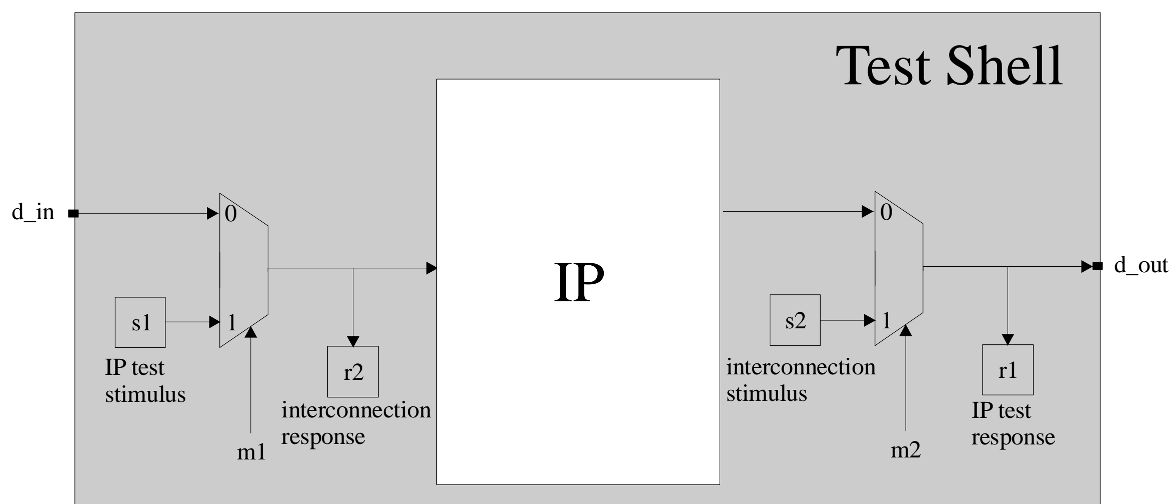


Figure 3.4: The IP-shell interface.

Note that due to the multiplexing of different signals at the boundary of the core, some of these categories may have common wires. For example, functional inputs/outputs, can be reused during scan operation as scan-chain inputs/outputs.

Looking at the figures 3.3 and 3.4, one may observe the four basic modes of operations of the test shell, being the functional mode, IP test, Interconnect test, and bypass mode.

A. Function mode

The test shell is transparent in this mode and this mode is reserved for the normal operating condition of the core, i.e., when the IC is not tested. It can be achieved, for example, by setting the multiplexer control inputs *m1* and *m2* to a logic zero. The function inputs and outputs are then directly connected to the IP module. The propagation delay of the multiplexers should be taken into account at this point since it could deteriorate the performance.

B. IP test mode

In this case, the core wrapped by the test shell is being tested. The test shell ensures that the test stimuli coming from the host are transported from the host (IC) source input connectors via TAM to the IP module under test and vice versa, i.e., that the test responses are transported from the IP module via TAM to the sink host output connectors (see Figure 2). This can be achieved by setting *m1* to logic "*1*" and *m2* to a logic "*0*". The test stimuli for the IP module will come from the test-data input *s1* that could be either a scan chain input or a special test input, depending on the type of the signal, providing the controllability in this mode. Likewise, the observability is achieved by the test-data output *r1* that captures the test responses. *r1* can also be either a scan chain output or special test signal output.

C. Interconnect test mode

In this mode, the multiplexers are set in states opposite with respect to the IP test mode. The test stimuli, coming from the host level via TAM, targeting the faults in interconnections and possible glue logic succeeding the outputs of the core under test are propagated via the test shell outputs. Therefore, the controllability in this mode is provided with *s2* (the test-data input) that will apply the test stimuli for succeeding cores or logic behind the test shell outputs. On the other hand, the test responses that come from the interconnection or glue logic in front of the inputs of the core under test are captured and transported to the host level via the TAM. Hence, the test-data output *r2* will capture the test responses coming from the test shell input, i.e. coming from the interconnection or glue logic preceding the test shell inputs, providing the observability in this mode.

D. Bypass mode

The test stimuli and/or responses for the other cores are transported via the test shell in this mode, independent whether the IP module has its own transparent modes or not. This mode is optional and is used if the cores are connected serially to provide an access path via cores that are not tested to the core-under-test.

The previously described modes are applied to the test shell, only. They are completely independent of any test mode of the IP module itself surrounded by that test shell. Of course, test modes of the IP can be extended with the modes of the test shell if required.

The modes of the test shell are determined via the so-called test control block (TCB). The steps of introducing the standardization of this block are also part of CTAG. The next section outlines the design of the test shells and the test control block in more detail.

3.3. The DfT implementation

3.3.1. Test Shell Implementation

The test shell is designed in such a way that an execution of the IP test is independent of the other IPs. Therefore, the IP under test has to be isolated from the other IPs. However, this condition is already fulfilled and is inherent property of the test-shell since both control and observation of the test data is done in the shell that surrounds the IP under test (see Figure 3.4 and explanation in previous section). The test shell is designed to contain the so-called *surrounding* scan chains, referring to *s1*, *s2*, *r1* and *r2* modules in the Figure 3.4, in order to enable the access to the ports of the core. The surrounding scan-chains connect all flip-flops at the boundary of the core. These can be either functional or added for testing reasons. Each type of core port (input, output, I/O) requires appropriate circuitry for isolation. The hardware described in this section is only needed when the inputs, outputs and I/Os are not connected directly to the IC pins. For the core terminals that are connected to the chip pins, no additional hardware is needed, since these terminals are directly controllable and observable.

The DfT circuitry placed within the test shell that is used to isolate input pins of the cores is shown in the figure 3.5.

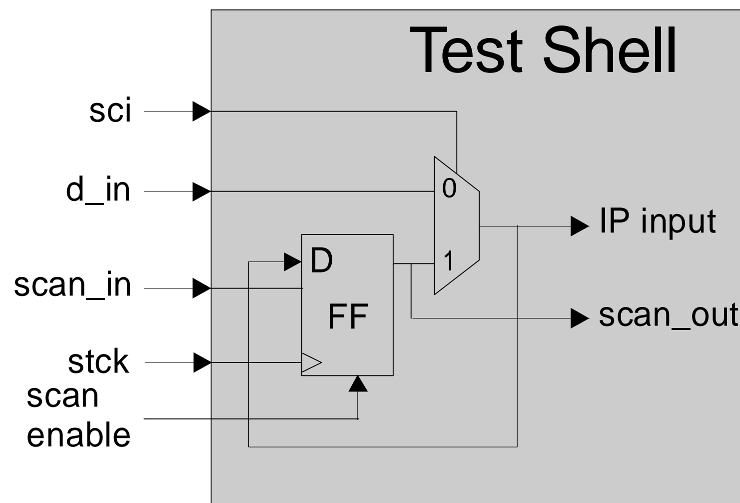


Figure 3.5: DfT circuitry for input port isolation.

A scan flip-flop from the surrounding scan-chain and a multiplexer need to be added to enable the control of the functional data input *d_in* in the IP-test mode and to observe the data from the preceding core during the interconnect test. The static test control signal, shell-control input (*sci*), sets the IP-test mode and is provided by the TCB. The flip-flop is

clocked (preferably) with an external test-clock (*stck*) for the reason of power reduction, since in this case the flip-flop will be inactive during the functional mode. Nevertheless, using an external clock is not mandatory. The *IP input* signal refers to the functional input of the IP.

A similar consideration holds for the isolation of the output ports of the core. The DfT hardware shown in Figure 6 provides observation of the IP functional output during the IP-test and provides control for the succeeding core during the interconnect test, by applying the stimuli to the core functional data output, *d_out*. The test mode is controlled by the shell-control output signal, labelled *sco*, also provided by the TCB. The same consideration holds for the circuit clock signal, *stck*, as it does for the core input isolation circuitry. The feedback loop, besides its test purpose, has another important contribution because it assures that data transfer is kept stable during the interconnect test. It provides the data transfer independent of the clock skew that may exist between the flip-flop in Figure 3.6 and the flip-flop in the succeeding core, i.e., flip-flop from the DfT circuitry for core-input isolation. Moreover, the feedback loop prevents the clock-skew problems that may appear if the sequential gates in the IP are clocked with phase mismatch with regard to the test-shell clock *stck*.

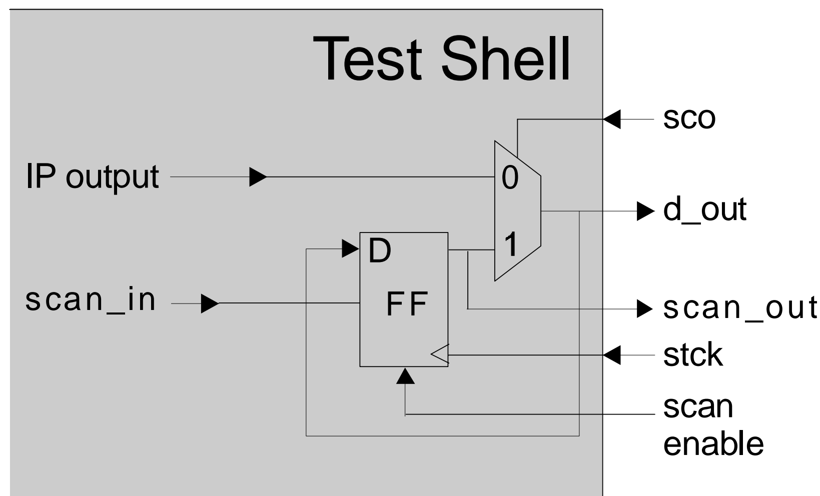


Figure 3.6: The DfT circuitry for output port isolation.

The I/O ports require a slightly more complex DfT for isolation. Basically, an I/O (input/output) is a combination of the three terminals: input, output and enable signal that controls a tri-state buffer. The DfT for input and output ports are not different from the DfT hardware that is required for unidirectional inputs and outputs. However, the enable signal requires special care as Figure 3.7 depicts. The additional logic in this Figure is necessary since during the IP-test mode, the on-chip bus should not influence the cores and vice-versa. Otherwise there would be a concern of having opposite values at the on-chip bus, which in worst case may lead to permanent destruction of the complete SoC. Hence, the flip-flop, multiplexer and DfT logic gate control the behaviour of the *enable* signal for IP and the interconnect test modes. During the IP test the signal *I/O_test* is set to 1. That will force the tristate buffer into the high-impedance state. The flip-flop and multiplexer combination can observe the output enable signal from the IP, while the test of the tristate buffer itself is tackled during the interconnect test. During the interconnect test mode, *I/O_test* is low, *sco* is high, and hence, the flip-flop/multiplexer combination can

control the enable signal. The core functional *d_I/O* bidirectional port is controlled and observed by the core input and output DfT circuitry for isolation, as previously explained. The clock skew is also handled and prevented as in the previous two DfT structures.

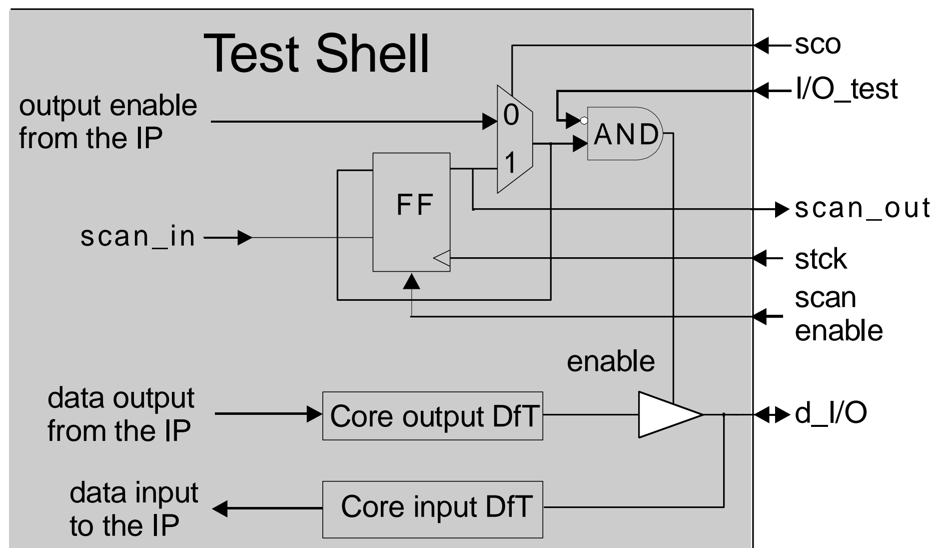


Figure 3.7: The Core I/O port isolation circuit.

The flip-flops in the previous drawings are connected into surrounding scan chains and they can be used for the test-stimuli application or the test-response capturing. All scan flip-flops within one scan chain must belong to the same clock domain. Multiple clock domains are also supported. However, in that case, there will be at least as many surrounding scan-chains, as is the number of clock domains. The flip-flops in surrounding scan-chains may be clocked with an external test clock *stck*, as already mentioned. Surrounding scan chains may also contain flip-flops from the core but then, they require clocking with a functional clock.

The implemented DfT circuitries within the test shells enable the test-data access to and from the cores within the embedded system environment. In addition, a straightforward implementation of the CAT flow becomes feasible as will be explained later in this chapter.

3.3.2. The Test Control Block (TCB)

The test shell contains a standardized test-control mechanism according to Figure 3.3. The test control block in that figure, controls the modes of operations of the test shell. Also, looking at the Figure 3.3, it can be seen that there are two types of test-control signals at the boundary between the test shell and the IP. For the sake of clarity, they will be repeated at this point:

1. A (pseudo) static test-control signal that remains stable during the entire test (e.g. test shell modes, BIST enable).
2. Dynamic test-control signals, which may change the value even during the execution of a single test pattern (e.g. scan enable).

Hence, dynamic test-control signals may need to change their values from one clock cycle to the next. For example, switching the scan chains from shift to normal mode and vice versa cannot take several clock cycles. Bearing these facts in mind, the dynamic test-control signals are to be treated as normal test-data signals. However, this will not be the case with a new set of (pseudo) static test control signals, which may take up to a few clock cycles. Fast access is not required, since these signals change their values very infrequently. In order to reduce the burden of the test-control block signals and save the number of host pins, the (pseudo) static test control signals may be applied via a serial access mechanism. The test control block is equipped with a register pair (*shift* and *update* register) similar to the structure used in the Boundary Scan test standard [IEEE90]. Both registers contain multiple-bit outputs whose number is equal to the number of (pseudo) static test control signals. Thus, the structure consists of so-called *bit slices*, as shown in Figure 3.8 [MAR98].

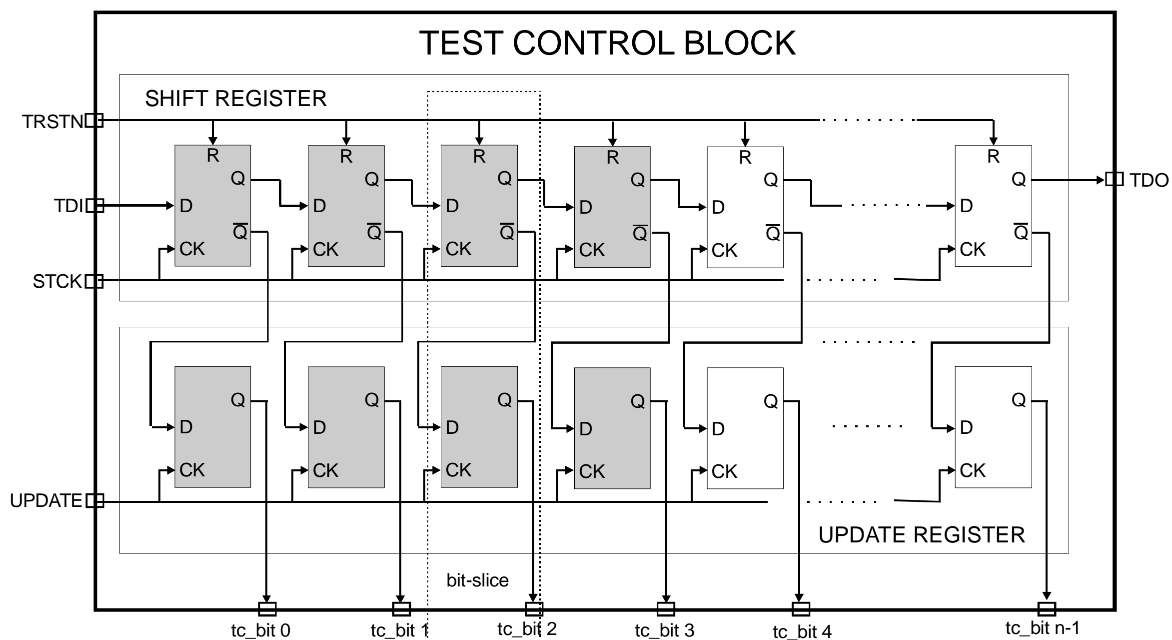


Figure 3.8: Structural view of the test shell control block.

According to the CTAG, the test control block within the test shell has four mandatory bit slices corresponding to the four basic operational modes of the test shell: functional, IP-test, interconnect test and bypass modes (shaded area in the Figure 3.8). However, the structure can easily be extended with other bit slices according to the required additional test modes of the particular IP. The test control block has following terminals:

- Test data input (*TDI*)
- Test data output (*TDO*)
- Test clock (*STCK*, may also be the functional clock)
- Test reset (*TRSTN*)
- Test update (*UPDATE*)

TRSTN and *UPDATE* terminals correspond to the “shell-mode signals” connectors from Figure 3.3, while the “tc_bit_i” output connectors from the figure above refer to the “static test control” signals from the Figure 3.3.

Since all cores have the same interface, the core user does not need to worry about all different test control signals in various cores. The TCBs of various cores may be concatenated via *TDI* and *TDO* ports that are referring to the serial test interfaces *STI* and *STO* from Figure 3.3, respectively. Due to the update register, the TCB can be programmed with the TCB output in any state. When the *UPDATE* signal appears, all test-mode control values switch instantaneously to their new values. This property, changing from one arbitrary mode to another is also useful for the functional debug and diagnosis since the test shell is also equipped with the bypass circuitry (see Figure 3.3).

The core user is of course free to connect the Test Control Blocks of various cores, in an arbitrary manner at IC level. There has to be a test controller at host-level to control the TCBs of the cores. This host-level TCB can be controlled from multiplexed chip pins directly or via e.g. a TAP controller [IEEE90]. However, that choice is left to the final SoC designer. Also, the core user is responsible for the actual implementation of the test access mechanism for the test data. That can be for example, a test bus [VAR98], an addressable test-port mechanism [WHE99] or a TestRail [MAR98].

To conclude this section, the following properties of the core-based test will be stated, organised with above explained concept of the test shell and test control block:

- Each embedded core is wrapped in a test shell providing the isolation from the system during the test and enabling appropriate infrastructure for that purpose.
- Each type of the core's port (input, output, I/O) requires a special DfT structure for isolation
- The test shell provides clock skew tolerance between the cores during test.
- The test control is coming from the register-type test control block (TCB) consisting of shift and update registers, which is placed in every core.
- The test in an embedded core environment is performed in two steps enabled by various modes of the Test Shell: the IP test and interconnect test.

The remaining part of this chapter will now focus on the Core-Test flow development from the point of view of the core provider.

3.4. Computer-Aided Test Flow Development

Independent of whether the cores appear in hard, firm or soft form, they are design descriptions only. They still have to be manufactured and subsequently tested for production faults. Hence, one of the challenges facing design engineers in a core-based design environment is the elaboration of a comprehensive test strategy. In order to enable a straightforward test and facilitate the job for the core user by means of the above stated advantages of the CTAG, the core provider needs to assign the appropriate CAT views to each core, besides its design views. The CAT view of the core consists primarily of the test patterns and test-protocols. With these available views, the task of the core user is to assemble chip-level test patterns from the pre-computed tests for the various cores. Besides the test patterns, the CAT view of the core has also to include information concerning the DfT overhead of each core and real fault coverage of the core in an embedded environment. Each company has a certain level of fault coverage that needs to be achieved during the production test, so that the core provider must give the actual information on the fault coverage of the core valid in an arbitrary environment.

The classical test-flow during IC design, once the netlist is ready, is shown in Figure 3.9. The flow starts from the gate-level netlist of the circuit obtained after logic synthesis and optimisation using one of the conventional EDA tools (*Synopsys*, *Mentor Graphics*, …). Most of these tools have also the capability of *DfT Insertion* (scan flip-flops, BIST), and its subsequent check for testability. Next step is *ATPG* (Automatic Test Pattern Generation) either within or outside the EDA tool, resulting in a set of the test patterns guaranteeing a certain fault coverage. These patterns need to have an appropriate protocol at chip-level, e.g. obeying the IEEE boundary-scan standard. For this purpose the *Protocol Expansion* tool is used. The Test Assembler has to create a file for the simulation of the circuit with its test patterns. If the simulation, using e.g. the *Verilog* gate-level simulator proves correct results, the design is ready to be delivered with the files in the tester format that may be applied to the real tester and manufactured device.

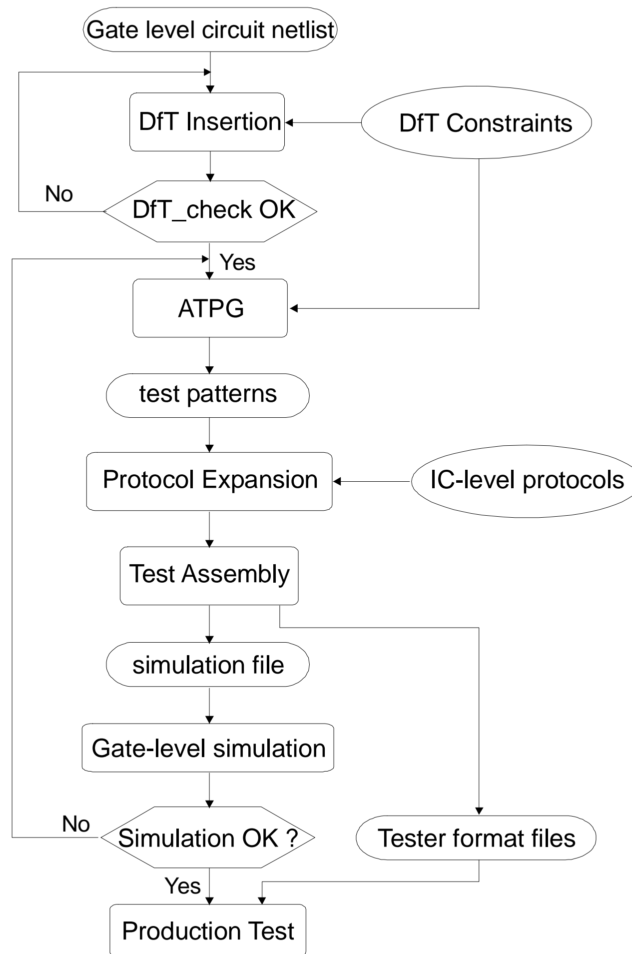


Figure 3.9: The conventional IC test flow.

However, this flow cannot be directly applied during the core-test development, at least not for the CAT view development of the individual cores. As already mentioned, the pins of the core will not necessarily be the host-level pins and thus not accessible in an embedded environment (see Figure 3.2). The test patterns generated in this way cannot be applied to a real core-based design, so the fault coverage obtained at the end of this flow cannot be considered accurate. Therefore, an alternative CAT flow had to be derived, according to the test access mechanism that is using the test shell, which is the situation in

our case. For the sake of clarity, it will be repeated that the test of the core consists of two steps:

1. IP test when the inner part of the cores is accessed via the TAM. The TAM is connected to the surrounding scan chains that enable the access to the logic/sequential elements inside the core. The test shell itself is not tested, at least not completely with this test mode.
2. Interconnections between the cores, glue logic and the remaining faults inside the test shell are tested during interconnect test. In this step, the test shells around the cores are used to drive and observe the interconnection, instead of the data from and to the cores.

Therefore, it is necessary to split the ATPG run according to the different test-shell modes. The test shell ensures the application of the test patterns to the IP, but its control signals must be in that case forced into predefined statements. Figure 3.10 illustrates this situation graphically in the example of the input DfT isolation circuit. The test generation for the inner part of the core (IP) will be carried out when the test-shell has a controlled value set for the IP-test mode (*sci = 1* in Figure 3.10a). IP test patterns will not cover all faults in the test-shell. This is because the primary inputs, outputs and I/Os of the core have to be skipped during the ATPG since the test patterns will come from the test access mechanism, e.g. scan-chain flip-flops within the test shell.

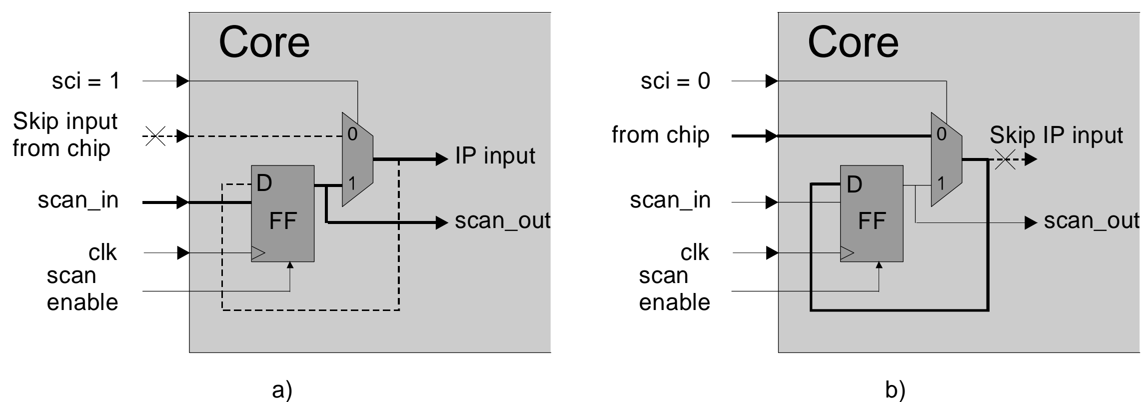


Figure 3.10: The test pattern generation for embedded cores:
a) IP test generation b) Interconnect test.

Subsequently, another ATPG run is required when the control signals of the test-shell are fixed for the interconnection test mode (*sci = 0*, Figure 3.10b) that will detect remaining uncovered faults. The inner part of the core is treated as a skipped block for this purpose. Of course, the interconnections between the cores and glue logic are minor part of the design and they will require fewer test patterns as compared to the IP test.

The fault coverage of the core is calculated at the IP equipped with the test shell. The test patterns obtained for the shell can be actually merged with the test patterns for the interconnect test. Consequently, combining and (fault) simulating the two pattern sets resulting from these two ATPG runs will result in an accurate fault coverage of the embedded core. These consist of the combination of the test patterns for the core and test patterns for the shell.

Therefore, it is obvious that a modified CAT flow had to be developed in order to obtain the core-level test patterns and accurate fault coverage. The core-based test generation flow that results in the patterns for the core and determines the real fault coverage for any environment in which the core can be embedded is shown in Figure 3.11 [VAR98], [ZIV98], [ZIV00]. The new flow proposed in this chapter is in full agreement with the internal CTAG standard. All CAT tools in this flow are developed with features to ease the core-based test flow. Hence, the flow is highly automated.

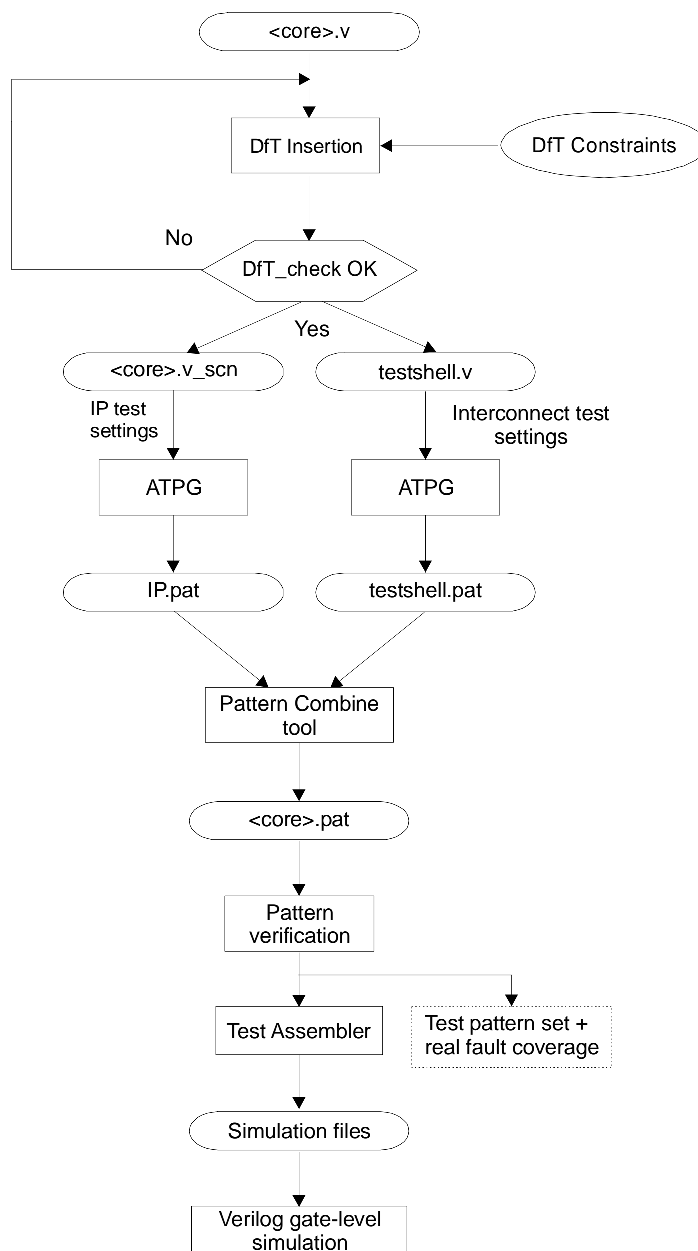


Figure 3.11: The proposed computer-aided test flow [ZIV98, ZIV00].

The flow starts with the gate-level netlist file named <core>.v obtained by means of synthesis using, e.g. the Synopsys synthesiser tool. The file <core>.v contains the IP accompanied with the flip-flops placed at the boundary of the core, i.e., within the test shell. However, the test shell is not configured yet, since the flip-flops at the boundary are

not connected into scan chains. Hence, DfT insertion is performed in order to make all flip-flops scannable, and next they are placed into the scan chains and form the test shell. The design and test engineer can direct this procedure by imposing a variety of constraints such as the ordering of flip-flops in chains, limiting the number of chains, etc. The DfT insertion results in a "scannable" verilog netlist <core>.v_scn and the *testshell.v* being the netlist extracted after DfT insertion.

If the DfT check is correct, the flow proceeds with ATPG. According to the previous explanation, two ATPG runs are required, in order to generate the test patterns for the IP and the interconnect test. In the latter case, the IP may be treated as a skipped block in the design in order to save CPU time for generation. The two sets of patterns are then combined with an appropriate tool (part of the complete test-kit) and passed to the pattern verification tool that calculates the real fault coverage of the core by applying the patterns to the core. A test assembler can subsequently generate simulator test benches for many simulators with both stimuli and response information. If the gate-level simulation provides correct results, the core is ready to be delivered containing its CAT views. The remaining part of the core test concerns the expansion of the core-test patterns and the implementation of the top-level infrastructure by the (existing) test-pattern expansion tool [MAR97]. However, that is the task of the core user. This is another good characteristic of the proposed method, i.e. the clear separation in tasks between core provider and the core user who does not need to worry about the test of a core of which he has not sufficient knowledge.

Once the top-level System-on-Chip netlist is available and all core-level test patterns are developed, integrated and expanded up to the chip-level, the proposed CAT flow from figure 3.11 can be concatenated with the second half of the flow from figure 3.9. This starts from the protocol expansion task concerning board test, to finalize the job and generate the patterns in format that can be applied to a real tester.

The CAT flow depicted in Figure 3.11, carried out by the core provider, results in an efficient set of test patterns and accurate and high fault coverage for each core for the arbitrary embedded environment. A fault simulation can be another method for the calculation of the real fault coverage. For that purpose, the *Verifault* simulator within the *Cadence* environment has been used. However, that is a problem since the simulation time was excessively long and it was practically useless to obtain the fault coverage in that way. The reason why the simulation had taken so much time is that the fault injection within the fault simulator was done not only at the moment of test stimuli application itself but also during the shifting of the patterns (stimuli or response) in the scan chains. However, it is of interest to know which faults are undetected during the actual test pattern application only, and not during shifting. Therefore, from a practical point of view, it was obvious that the usage of Verifault simulator has not been justified. The fault coverage calculation using the methodology proposed in this thesis is much faster. Another reason to use our derived approach is maintaining the consistency with Philips CAT flow.

3.5. Results

The execution of the resulting set of test patterns obtained by means of the proposed CAT flow itself will be illustrated in an example that starts as a "bare core", given in Figure 3.12. This core is of course very simple and cannot be taken as a representative in terms of the complexity concerning the cores that have been used in the PI-Bus design.

However, its simplicity should help reader to understand the steps one has to follow in order to make the core "CTAG compliant".

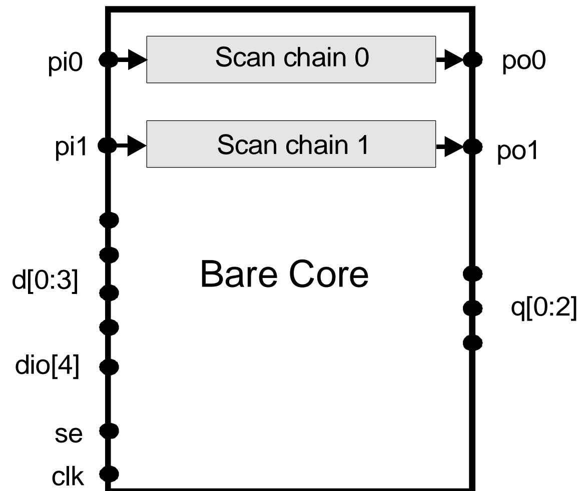


Figure 3.12: The example "Bare Core".

The core has four functional input terminals *d[0:3]*, one I/O terminal *dio[4]*, and three functional output terminals, *q[0:2]*. In addition, the "bare core" has two core-internal scan chains with arbitrary length and running between terminals *pi0* and *po0* and *pi1* and *po1*, respectively. The scan chains are shifting the data if a logic "*1*" is assigned to a terminal scan enable (*se)*. The clock *clk* clocks internal circuitry of the core.

In accordance to the flow depicted in Figure 3.11, DfT circuitry in the form of the test shell needs to be inserted first. Therefore, the terminals of the core are isolated with circuitries given in figures 3.5, 3.6 and 3.7 corresponding to their functionality. The flip-flops of the DfT circuitries are connected to form a surrounding scan chain, and a new scan chain input/output ports are inserted. Also, the test clock (s*tck*) has also been introduced to the test-shell. Upon the test-shell instantiation, the test control block is added, together with the bypass circuitry for core serial connection. After the DfT insertion is done, the core obtains the form as depicted in Figure 3.13.

The pi[0:2] and po[0:2] may be the parts of the TestRail providing the test data access mechanism to the bare core. The multiplexers MU1 and MU2 enable the functional debug of the core by making the connection of the surrounding scan chains and serial interface. It is also possible to connect internal scan chains by inserting an additional multiplexers, (not shown in this figure) thus having only one chain running through the complete core. The purpose of the multiplexer MU3 is to shift the instruction in and out of the test control block. This task is used also for the functional debug of the test control. The wires coming from the test control block are also not shown in this figure for the sake of readability. These signals are referring to the control signals of the test-shell ports isolation circuitries.

In Figure 3.14, the active wires of the test data that is referring to the four modes of the test shell are bold highlighted, so that one may have clear vision which paths are active in a particular mode. The different modes of the test shell are set using *static test control signals*. They set the test control signals of the multiplexers (*sci*, *sco*) within the DfT isolation circuitries given in Figures 3.5, 3.6 and 3.7 into the appropriate state. Also, the multiplexers enabling the functional debug, MU1, MU2 and MU3, are set via these control signals. The *static test control signals* are controlled through the test control block

(TCB), i.e., the *shell-mode* control signals. The *shell-mode* signals refer to the *TRSTN* and *UPDATE* signals from the Figure 3.9. Of course, there is an option to extend the set of the control signals within the TCB. The core activity during the test is summarized in the text that follows. All signals correspond to the notation given in Figures 3.5, 3.6 and 3.7. If the control signals are not mentioned, it means they have a *don't care* value with the exception of the scan enable signal that may changes its value during the execution of one single pattern.

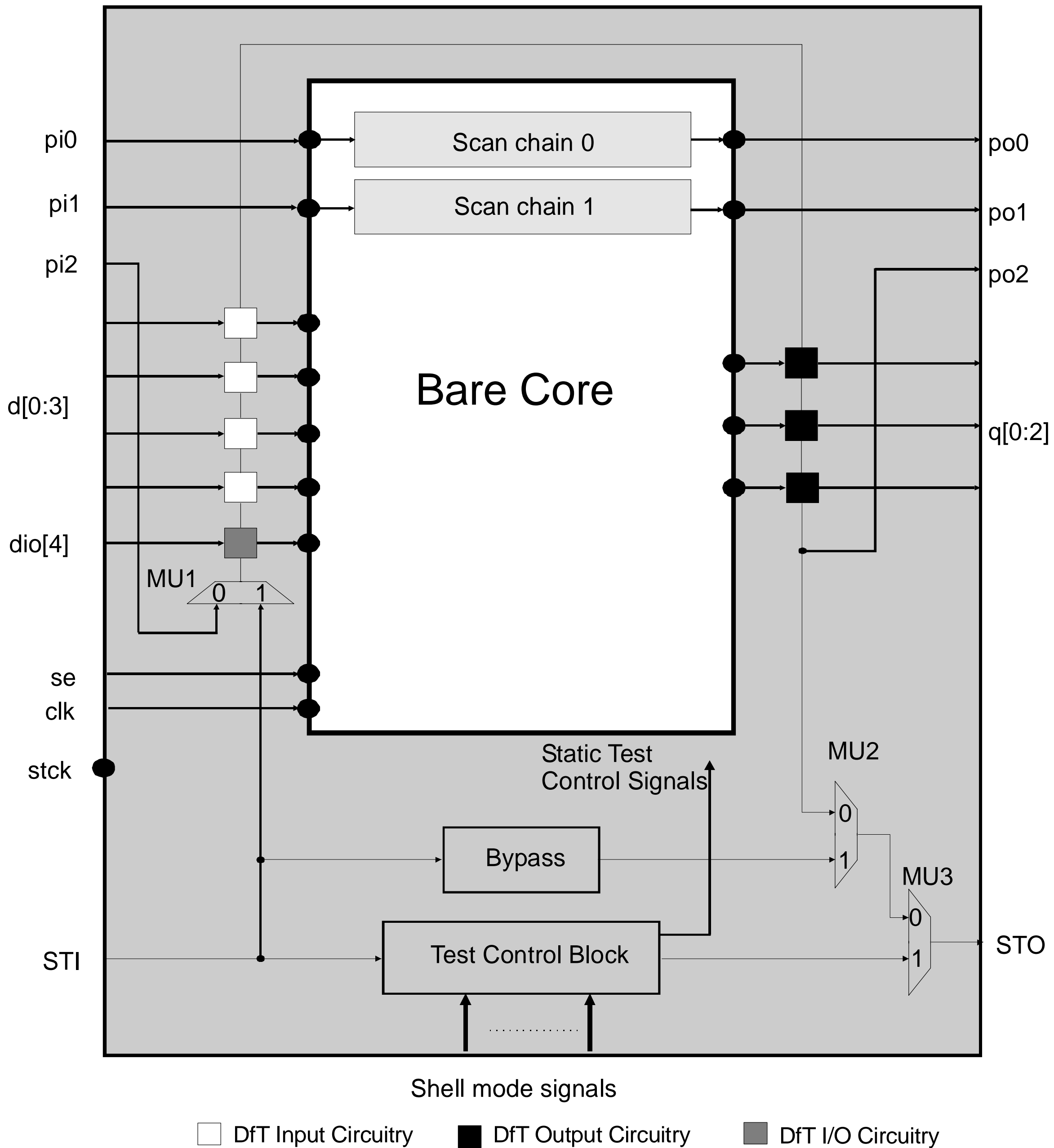


Figure 3.13: Example "Bare Core" equipped with the Test Shell Circuitry and the Test Control Block.

A. Function mode (*sci = 0, sco = 0, se =0*)

No test is performed, the core carries out its functional duties.

B. IP test mode (*sci =1, sco = 0, i/o_test =1, MU1 =0*)

The test patterns for the IP are scanned into the surrounding and internal scan-chains via *pi[0:2]* and applied to the core. The response is captured and shifted out via *po[0:2]*.

C. Interconnect test mode (*sci =0, sco = 1, i/o_test =0, MU1 =0*)

The patterns are scanned only to surrounding scan chain, *pi2* and applied via the outputs of the core *q[0:2]* and I/O port *dio[4].* The response coming from another core is captured in core inputs *d[0:3]*, I/O port *dio[4]* and scanned out to *po[2].*

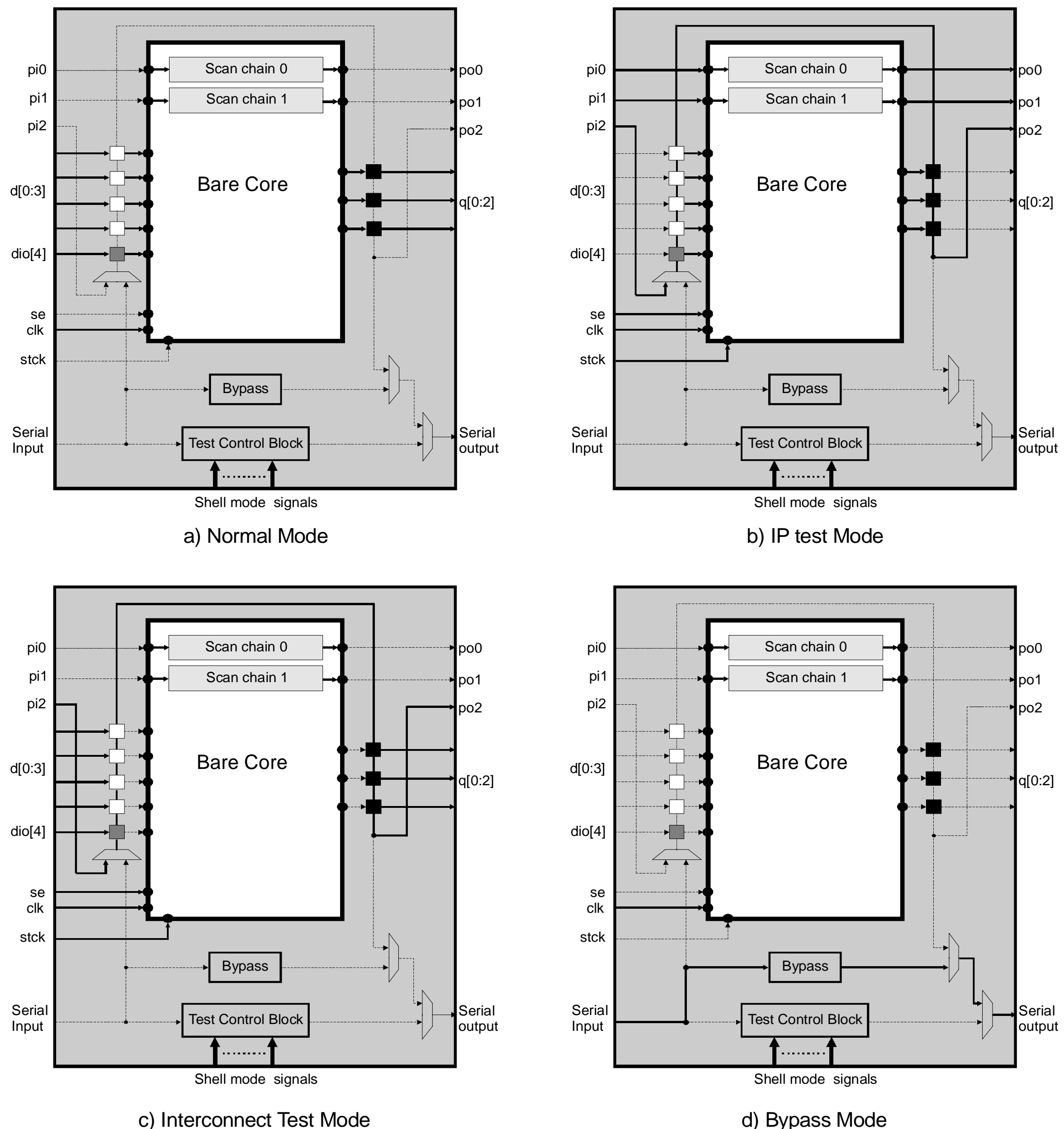


Figure 3.14: The active wires in various test-shell modes.

D. Bypass mode (*sci = 0, i/o_test =1, sco =1, se =0, MU2=1, MU3 =0*)

The core is transparent and the signal is coming from the serial input, *si*, and transported without any alterations to the serial output, *so*. Note that, with *MU1 = 1*, and *MU2 = 0*, the core functional debug becomes feasible. However, the debugging issues have not been the part of this study and thus, will not be considered.

The described patterns are combined with a specially derived tool and verified with another tool from the SoC test kit, according to the proposed CAT flow from the figure 3.12. The remaining tasks are to generate the simulation files and the test pattern formats that the tester device may use.

Figure 3.15 outlines the concatenation of the cores and the already mentioned test access mechanism (TAM) at the host-level. This System-on-Chip is, with respect to test, organized with one serial and one parallel test access mechanism per core. The Philips Pilot project has used the TestRail [MAR98] concept for the parallel TAM design, meaning that the cores can be accessed in parallel during the test. The TestRail is capable of handling all synchronous digital test signals from IC pins to the core under test and vice versa. In addition, the TestRail is scalable, i.e., its width is adaptable to the number of test bits per core. The concrete implementation of the TestRail is, however, not the topic of this work and will not be considered.

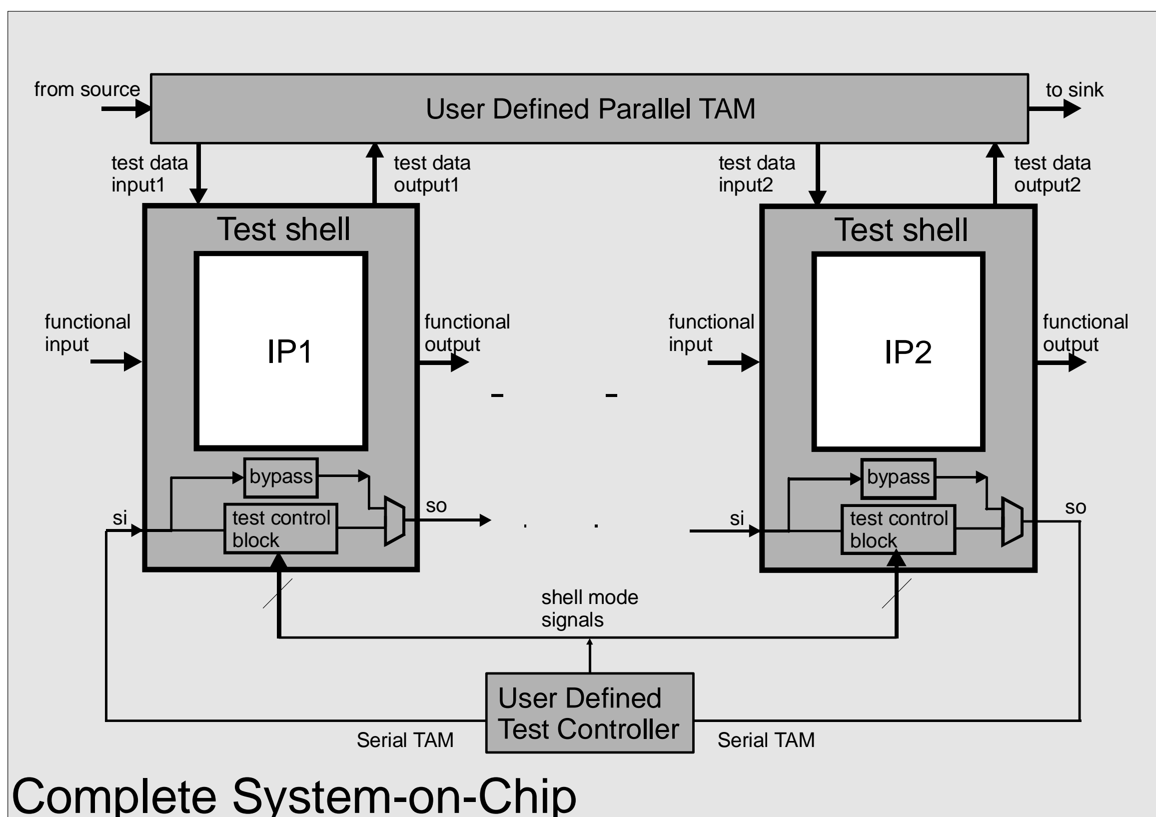


Figure 3.15: Core integration for core-based test at the host-level.

The core-level TCBs are connected in series while the test controller at IC-level is user defined. The serially connected TCBs together with bypass circuitry (placed also in each core) enable the functional debug.

The proposed CAT flow for the test-pattern generation and exact fault coverage calculations of the cores has been applied to the Philips pilot IC project chip of which the main characteristics are shown in Table I. The pilot IC contains two RISC microprocessor cores, a core that enables microprocessor software debug, several memory interfaces, an UART and several other peripherals as well as user defined logic (UDL). The overall architecture has been implemented using the PI-bus concept shown in Figure 1. The various design centres developed the cores while the core-based test strategy itself has been performed within Philips Semiconductors. The cores came in soft and firm form and the proposed CAT flow assigned them the test views, that can also be found in the Table 1, concerning the fault coverage and number of the test patterns of the complete System-on-Chip.

TABLE I. THE MAIN CHARACTERISTICS OF PILOT IC.

Process	CMOS 5-metal layer
Feature size	0.35μm
Total die area	50 mm^2
# of cores	8
Complexity	$\approx 20\times10^6$ transistors
# of test-patterns	3500
Size of the largest core	21.4 mm^2
Size of the smallest core	0.45 mm^2
Total extra DfT area	7.7 %
Fault Coverage (%)	98.5 –99.94

The assumed fault model is based on the stuck-at fault modelling. The fault coverage of the cores varies between 98.5 and 99.94 %. The obtained results are much better as compared to a chip of similar complexity and functionality as reported in [SAX98] with an average fault coverage of 91% and more than 4000 patterns. Note that the number of test patterns is obtained after simple addition of the test patterns of each core. Hence, the test patterns have still to be expanded and compressed. However, it will not increase the number of test patterns, since only the test-protocols will be changed, not the test data. The test expansion task will be also straightforward due to the presence of the test shell and the parallel test mechanism i.e., TestRail, unlike the macro test approach where the successful protocol expansion depends on the signal paths through the cores [BEE86], [MAR97]. The results concerning one particular core from this design are summarized in Table II. It reports the number of faults inside the core, the fault coverage and number of generated test patterns depending on the test mode of the core. Therefore, the second column contains the results with “unrealistic fault coverage” when the complete core is treated as a non-embedded core with all terminals free to be accessed from the external world. The third column contains the results when the test-shell is set into IP test mode, while fourth one contains the results when the test-shell is configured for interconnect test patterns. Finally, the fifth column contains real fault coverage when both IP followed with interconnect set of test patterns are applied to the core.

TABLE II THE TEST RESULTS FEATURED AT ONE OF THE CORES USED IN PI-BUS DESIGN

	"Non-embedded" core test	IP test	Interconnect Test	IP + Interconnect Test
# of total faults	19990	19566	4797	19990
# of detected faults	19963	17378	2915	19955
# of undetected faults	27	2188	1882	35
Fault Coverage (%)	99.86	88.82	60.78	99.82
# of stuck-at patterns	210	207	26	233

According to the obtained results expressed in this table, one may conclude the following important issues:

- The realistic fault coverage given in the last column is close to 100 % justifying the proposed approach.
- The percentage of undetected faults after IP run depends on the size of the IP. It means that if the IP is smaller, the fault coverage will be lower, since the test shell size becomes more significant within the total core.
- The Interconnect test comprises of the small number of the test patterns.
- The total number of faults during IP test only is lower than in both embedded and non-embedded core test and is the consequence of "skipped" functional terminals of the core.
- The Interconnect test is supplement to the IP test concerning the detected/undetected faults.
- The total fault coverage after IP + Interconnect runs is slightly lower comparing to the one obtained for the non-embedded core. Therefore, in the case of that core, there was a small fraction of faults that could not be detected even after both sets of the test patterns are combined and applied to the core. This appeared to be due to the particular design implementation peculiarities of the test shell. However, these faults have been subsequently detected by the IDDQ test patterns, later in the phase of testing.

The DfT overhead, another constraint that has to be taken into account is also quite low. It amounted to 7.7% of the total SoC area. It increased only 3.2% due to the test-shells around the core and the rest is originating from the core internal DfT. It has been noticed during the design that the small sized cores require a relatively larger area for DfT. This occurs since the test shell has the most impact on a small core with many ports. The DfT area overhead for the smallest core is 30.2% while the area overhead due to the test shell for the largest core was only 0.5 %. Therefore, further improvement of the core test strategy will focus on reducing the DfT area overhead for the smaller cores. Of course, that reduction may not influence the already achieved efficiency in terms of the test patterns and fault coverage.

3.6. Conclusions

This chapter describes the methodology used for the test pattern generation and realistic fault coverage calculation in embedded core-based design. The core-test strategy according to Philips CTAG group has been explained. The so-called test shell is introduced providing the infrastructure for the test and isolation of embedded cores. The test of the embedded core consists of the test of the inner part of the core (IP test) and the test of its interconnection together with the test shell. The problem of getting the accurate fault coverage due to the introduction of the test shell is explained. The analysis of the DfT structures used within test shell in core-based design is performed. A new automatic test-pattern generation flow according to the CTAG core test strategy has been proposed and implemented. It results in an efficient test pattern set and accurate fault coverage of the core that can be used in an arbitrary environment. The proposed flow has been used in the Philips pilot IC project. The IC is based on the Peripheral Interconnect (PI) bus concept. The obtained results with respect to the number of the test patterns and the fault coverage were better compared to another similar design that has been done at another place. Also, the total DfT area overhead was quite acceptable. Another important benefit of the proposed core-test flow is the clear separation of responsibility between core provider and core user with respect to test. The future work in this field has to study the reduction of the DfT overhead for smaller cores since their area becomes quite excessive due to the introduction of the test shell. This problem may be overcome by leaving the possibility to combine the smaller cores into a large one such that an impact of the Test Shell is relatively smaller. Another possibility is to treat smaller cores as user defined logic (UDL) and test them during the interconnect test without equipping them with a test shell.

The experience gained during the elaboration of this work has been used to derive the test synthesis of another bus-based processor. The next two chapters will outline that method.

Literature:

[ARE98] R. Arendsen, M. Lousberg, "Core Based Test for a System on Chip Architecture Framework," *Proc. of the 2nd IEEE Workshop on Testing Embedded Core-based Systems*, October 1998, paper 5.1, Washington D.C.

[BEE86] F. Beenker, K. van Eerdewijk, R. Gerritsen, F. Peacock, M. van der Star, "*Macro Test: Unifying IC and Board Test,"* IEEE Design and Test of Computers, vol. 3, no 4, pp. 26-32, December 1986.

[BEE95] F. Beenker, B. Bennetts, L. Thijssen, *Testability Concepts for Digital Ics – The Macro Test Approach,* volume 3 of *Frontiers in Electronics Testing*, Kluwer Academic Publishers, Boston 1995.

[BHA98] D. Bhattacharya, "Hierarchical Test Access Architecture for Embedded Cores in an Integrated Circuit," *Proc. IEEE VLSI Test Symposium,* pp. 8-14, April 1998.

[GHO97] I. Ghosh, N. K. Jha, S. Dey, "A Low Overhead Design for Testability and Test Generation Technique for Core-Based Systems," *Proc. Int. Test Conf.*, November 1997, Washington D.C, pages 50-59.

[IEEE90] IEEE Computer Society, *IEEE Standard Test Access Port and Boundary-Scan Architecture – IEEE Std 1149.1 – 1990*, IEEE, New York, 1990.

[IEEE00] IEEE P1500 Web Site, http://grouper.ieee.org/groups/1500/.

[IMM90] V. Immaneni, S. Raman, "Direct Access Test Scheme – Design of Block and Core Cells for Embedded ASICs," *Proc. Int. Test Conf.*, September 1990, pages 488-492.

[JAS98] A. Jas, N. Touba, "Test Vector Decompression via Cyclical Scan Chains and its Application to Testing Core-Based Designs," *Proc. Int. Test Conf.*, October 1998, Washington D.C, pages 458-467.

[MAR97] E. J. Marinissen, M. Lousberg, "Macro Test: A liberal Test Approach for Embedded Reusable Cores", *Proc. of the 1st IEEE Workshop on Testing Embedded Core-based Systems*, Washington D.C, November 1997, paper 1.2.

[MAR98] E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, C. Wouters, "A structured and Scalable Mechanism for Test Access to Embedded Reusable Cores", *Proc. Int. Test Conf.*, October 1998, Washington D.C, pages 284-293.

[SAX98] J. Saxena, P. Policke, K. Cyr, A. Benavides, H. Malpass, F. Ngoh, "Test Strategy for TI's TMS320AV7100 Device", *Proc. of the 2nd IEEE Workshop on Testing Embedded Core-based Systems*, October 1998, paper 3.2, Washington D.C.

[VAR97] P. Varma, S. Bhatia, "A Structured Test Re-Use Methodology for Systems on Silicon," *Proc. of the 1st IEEE Workshop on Testing Embedded Core-based Systems*, Washington D.C, November 1997, paper 3.1

[VAR98] Various authors, "*Core-based Test: Cookbook*", Philips Internal publication, 1998.

[VSI00] VSI Alliance Web site: http://www.vsi.org/

[WHE97] L. Whetsel, "An IEEE 1149.1 Based Test Access Architecture for Ics with Embedded Cores," *Proc. Int. Test Conf.*, November 1997, Washington D.C, pages 69-78.

[ZIV98] V.A. Zivkovic, R.G. J. Arendsen, "Accurate Fault Coverage Determination in Core-based Test," ASG/ESTC/98.7156, October 1998, Philips Internal Documentation.

[ZIV00] V. A. Zivkovic, R.J.W.T. Tangelder, H.G. Kerkhoff, "Computer-Aided Test Flow in Core Based Design", 22nd International Conference on Microelectronics, Nis, Yugoslavia, 14-17 May 2000, pp. 715-718, ISBN 0-7803—5235-1.

CHAPTER 4

Testability Constraints in the High-Level Phase of Hardware/Software Codesign

4.1. Introduction

Codesign, as concurrent design engineering of hardware and software components, employs a variety of tools for the design of heterogeneous systems, as explained in the preceding chapters. A framework to assist designers with the transformation of an algorithmic specification into a suitable hardware implementation and to allow the possibility for easy integration of various tools is a clear necessity. There have been numerous academicals efforts, e.g., [WIL95, COR95, BUC94] to compose such a tool and carry out the design of a mixed hardware/software (HW/SW) implementation. All these frameworks should have the following properties in common:

- They have to start with an algorithmic description in a high-level language such as *C* or *C++*. A high-level specification is also important for successful maintenance and documentation of a HW/SW product.

- They have to be able to handle a wide variety of hardware instantiations. For example, the designer should be able to compare an implementation with certain modules and interconnections between them inside the datapath with other implementations containing other types of components and interconnections.
- They should be interactive, giving the designer complete influence on the flow of the execution at any stage.
- They have to provide the relevant statistical data concerning the input application to the designer. This is performed within the so-called *profiling* tools to identify the functions and operations that are the most heavily used.
- An estimation of the performance of the final product with respect to e.g., area, throughput, power, etc. has to be provided.
- Reuse of the hardware and software components from a library should be supported. Hence, the codesign framework should be able to accept the new component that the designer imposes, i.e., the library of the component has to be easily extendable with respect to the type and attributes of the components.

However, none of the above mentioned tools including the most recent, powerful *Synopsys* CoCentric System Studio [SYN01] does consider the testability of the system as an additional parameter during the high-level synthesis. Hence, they do not offer the possibility to assess the overall test cost before the actual hardware implementation. This may have serious consequences with regard to the resulting product. For example, even though the optimal solution in terms of area/timing/power has been found, its cost after the test synthesis could increase. This can be reflected in the large number of test patterns necessary to test the resulting structure. The test time, which is directly proportional to the number and depth of the test patterns, is the parameter that is becoming increasingly important in the overall cost of modern System-on-Chip (SoC) implementations. Moreover, the introduction of DfT circuitry like BIST inside the final implementation may lead to the degradation of already achieved performances with respect to area and throughput. In addition, it may complicate the design and/or increase its design time. In extreme cases, the test harness may implicate the usage of complete Codesign flow not being justified anymore over the ASIC full-custom design style. This, when the resulting circuitry after the DfT insertion becomes cumbersome compared to the same (testable) circuitry obtained using the full-custom design. Therefore, it is more convenient if the designer of an embedded HW/SW system has a notion of the test implications during the early phase of the design. Our proposal is to introduce an additional test constraint during the high-level phase of the codesign. This constraint will allow the designer to assess the number of the test patterns that the resulting implementation will require for its test, while minimizing the influence of test at the achieved area-throughput ratio.

This chapter is organized as follows. The second section introduces the so-called *MOVE* codesign framework [HOO96], together with the explanation of the concept of the so-called *Transport-Triggered Architecture* (TTA) [COR95]. The third section copes with adding the testability criterion into the *MOVE-TTA* template. The test constraint is derived while exploiting the structure and properties of the TTA itself. A new test style, the so-called *functional-structural* test that fully exploits the instruction-level scheduling of VLIW TTA will be proposed. The analytical expressions for the test cost are given in the fourth section. The fifth section elaborates the test-scheduling introduced to decrease the number of test-cycles during the application of functional-structural test patterns. The CAD implementation of our approach is outlined in the sixth section together with

examples of our method applied to various applications. Section seven illustrates our approach applied to the *CASTLE* framework, being another hardware/software codesign tool. The conclusion is given in section eight.

4.2. The MOVE Codesign Framework

The MOVE framework [COR98a] consists of a set of tools for hardware and software synthesis of the Transport-Triggered Architectures (TTA). TTA's are derived from VLIW (Very Long Instruction Word) Processor architectures. The difference between the two architectures is that the TTA have implicit data control, i.e. they are programmed by specifying the data transports instead of the operations. They support a high degree of instruction-level parallelism and offer a possibility to parameterize the architecture, thus leaving the designer more degrees of freedom in the design. This will result in a more efficient use of hardware resources and a less complex hardware structure [COR95]. The simple hardware structure is paid through a complex, sophisticated compiler structure. Hence, the TTA relies heavily on compiler support for achieving high performances. There are also other additional advantages of using the TTA, such as e.g. cycle minimization, implementation flexibility, performance scalability, etc. They are all explained in more detail in [COR95, HOO96, COR98b]. Hence, these architectures have serious potential in important applications in the future [COR98b].

A typical TTA template is shown in Figure 4.1. It is obvious that a TTA consists of the datapath, interconnection network and control part. FU denotes functional units such as adders, comparators, multipliers, ALUs etc., while RF denotes register files. The interconnection network is comprised of busses and sockets. Each functional unit and register file from the datapath is directly accessible from/to the bus via the sockets that connect their input or output ports to the busses. The control block contains Load/Store unit (LD/ST), Instruction Fetch Unit (IFU) with an embedded instruction cache and Guard Unit (not shown in Figure 4.1). The Memory Interface Unit (MIU) connects the Instruction Fetch and the Load/Store unit to an external memory.

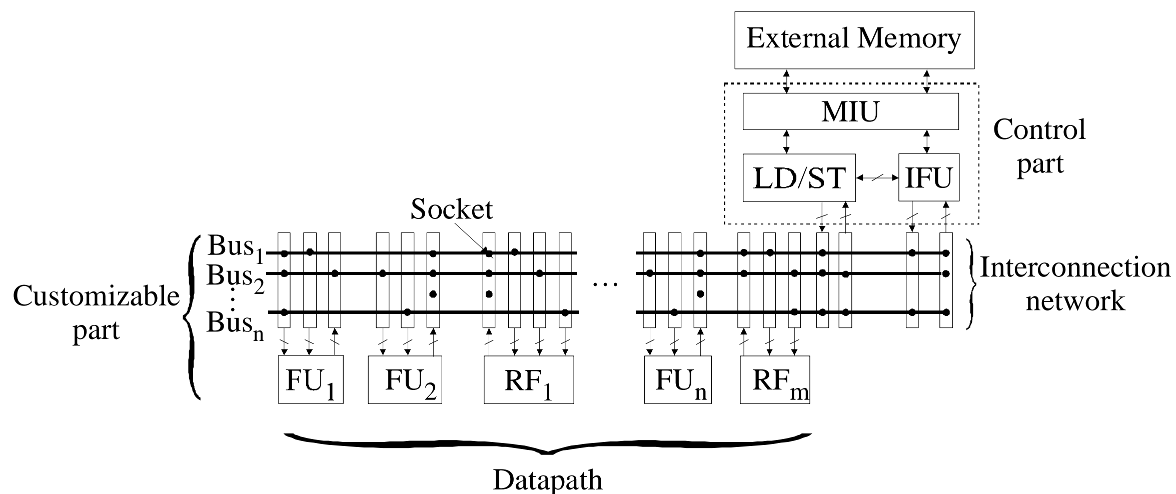


Figure 4.1. The TTA structure

The sockets perform the control of the data flow through the network. There is only one type of operation, being the "*move*" operation through the network, from one FU (or RF)

to another one. The number of possible move operations in parallel is equal to the number of busses implemented within the TTA. The functional units and register files are triggered with the arrival of data at their inputs, performing their function(s) subsequently, i.e., the operations are triggered with the transport of the data through the network. In the sequel of the text, both functional units and register files will also be referred to as components. Actually, the register files may be considered as functional units performing the “identity” operation.

Figure 4.1 also shows that the TTA has a relatively simple hardware structure with a lot of regularity in the datapath. The architecture is fully pipelined, meaning that not only a few different operations in parallel are possible, but also the cycles of consecutive instructions are scheduled in the same timing slots. During the execution of an operation, the scheduled, pipelined data transport from a MOVE bus via the sockets and components to another (or the same) MOVE bus takes place (see Figure 4.1). The function units and registers files are implemented using the so-called *hybrid-pipelining* mechanism [COR95]. The MOVE processor generator supports two types of pipeline schemes, the two-stages (the preferred one, because it is faster) and three-stages pipeline. Figure 4.2 shows both of these situations.

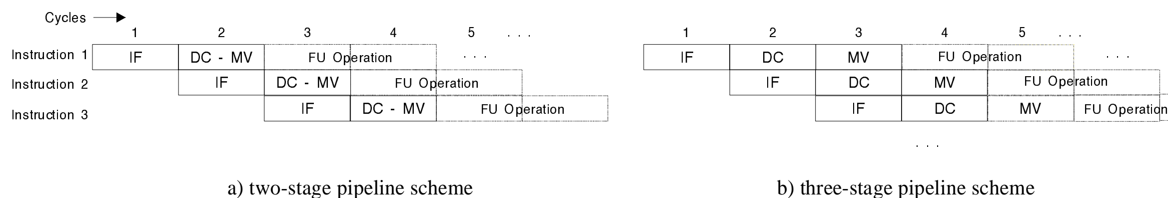


Figure 4.2. Pipeline schemes in the TTA

IF, DC and MV denote the instruction fetch, decoding and the actual data transport (move) phase of the instructions respectively. Note that the decoding and transport stages are combined in the first case. According to Figure 4.2, an operation of the functional unit may be executed in one or few cycles. The number of cycles necessary to complete the execution of the operation determines the *latency* of the component. Further details of the architecture and execution of the program can be found in [COR98a].

The main characteristics of the TTA’s are as following:

- Regularity and modularity. FU’s are independent of each other and the interconnection network. They can be all designed and optimised separately. There are virtually no constraints on how to design the interconnection network and the functional units; they just need to fulfil the interface specifications prescribed by the used sockets.
- Flexibility and scalability. Due to the fact that interconnection network and FU’s are separated from each other, the architecture can be easily expanded by adding extra functional units or increasing the transport network capacity, i.e., extending the number of bus lines and sockets. Both can be done without affecting each other, so the instruction format is not changed.
- Performance increase. The processor is optimised for the operation throughput instead of latency. These measures result in an architecture where the cycle time is limited by the time needed that datapath components perform their internal operations.

- Low connectivity. Because of reduced register-file traffic, the number of registers inside the datapath is decreased. Moreover, using the partitioning algorithm described in [JAN95], the large register files can be split up into smaller ones containing one input and two output ports.
- Trivial decoding. The decoding logic is significantly reduced (similar to RISC machines) and it is performed within the sockets, as it will be shown later. Hence, the TTA supports only one operation format.

These characteristics will also be used while deriving the optimal test-strategy, as will be explained later.

In order to generate the TTA template, the MOVE hardware/software codesign system has been used. The codesign flow itself is drawn in Figure 4.3.

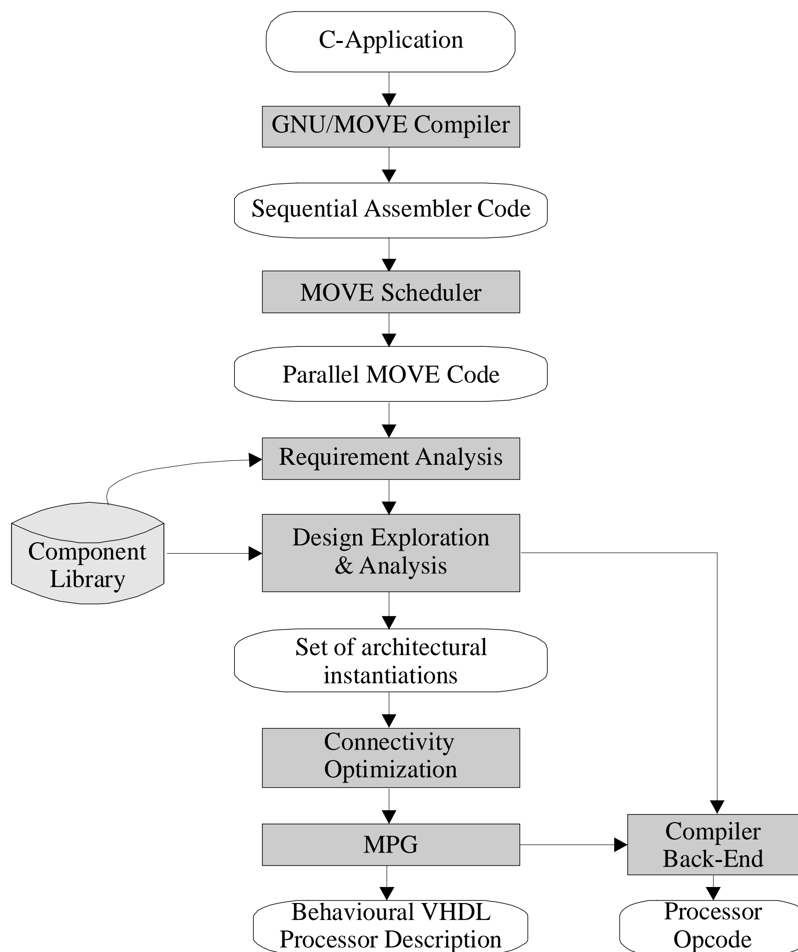


Figure 4.3. MOVE codesign flow

The flow starts with the application written in the C programming language. The C-code is compiled and subsequently scheduled and profiled in order to make the parallel code suitable for the specific assembler language imposed by the TTA. In addition, the *MOVE* tool sets the minimal hardware requirements necessary to execute the basic operations driven by the generated parallel assembler code. Consequently, the tool offers the designer the freedom to influence the final outcome of the instantiation of the application by choosing the structure and connectivity of the datapath. Hence, the

architecture is highly parameterizable with respect to the type and number of the functional units, register files and buses within the datapath (see Figure 4.1). The exact match of the datapath parameters is the subject of *design-space exploration* provided within this environment. In addition, the designer may influence the final result concerning the sockets, i.e. the mutual connections among the blocks in the datapath during the so-called *connectivity-optimisation* phase. So far, the main design evaluation criteria within MOVE have been circuit area and performance. The exploration is performed with iterative generation of different datapath instantiations performing the input applications. Hence, for each instantiation, the area cost and time necessary to execute the given application are generated. The library of the components, adjoined to the codesign kit, has been used for this purpose. It contains information with respect to the area and delay of each component used in the datapath. The solution space is bound by local optimal solutions, the so-called *Pareto points* [BRA80]. Let $\vartheta^2(a, t)$ denotes the two-dimensional area-execution time solution space. A point $(a_p, t_p) \in \vartheta^2(a, t)$ is a Pareto points if and only if $\forall\ (a, t) \in \vartheta^2(a, t)$, $(a \geq a_p \vee t \geq t_p)$. For example, Figure 4.4 depicts the result of the design space exploration for the "Crypt" application [PAT87]. The Pareto points on the graph refer to a particular architectural instantiation. Of course there exist a number of other solutions above the Pareto point curve that are not shown in this Figure.

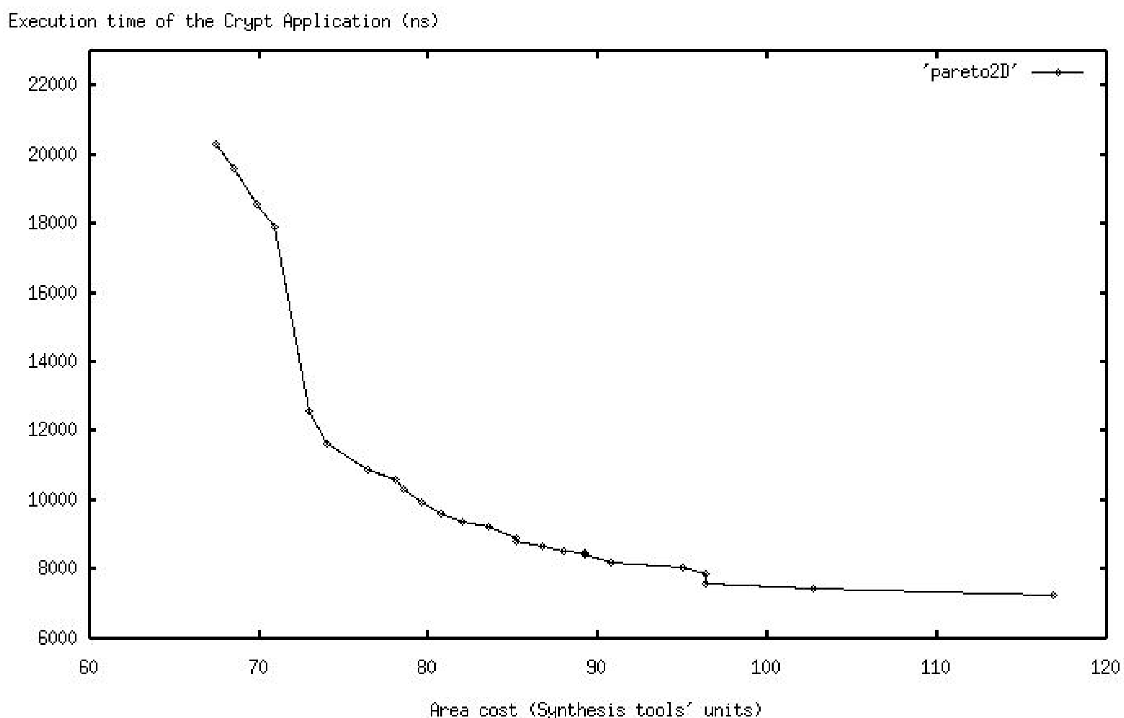


Figure 4.4 The solution space limited with Pareto points for Crypt application

The result of the design space and connectivity exploration is a set of TTA instantiations performing the input application in combination with the applied profiled assembler code. Hence, the designer may choose the instantiation that satisfies his initial specifications according to the area and throughput criteria. The TTA instantiations are given at the algorithmic level of abstraction. They are converted to behavioural VHDL code using the MPG (MOVE Processor Generator) [VAR00], the back-end tool of the

MOVE codesign kit. MPG also generates the files for the simulation while interacting with the compiler back-end.

The previously described high-level design approach, so far, does not offer the possibility to assess the overall test cost of the resulting TTA before the actual hardware implementation. Moreover, the resulting behavioural VHDL code of the application is not testable and our experience has shown that, in our case, the time spent for DfT (Design for Testability) largely exceeded the time for its design. The optimal solution in terms of area/timing does not necessarily remain optimal after the DfT strategy is introduced and profiled. It can be reflected with a large number of test patterns necessary to test the resulting structure or an increase in area or throughput degradation, in case additional circuitry for test is introduced in the critical path of the circuit. Therefore, it has been decided to extend the characterization of the architectural instantiations assigning the test constraint within the design space exploration during the high-level design, while preserving the already achieved area-performance (execution time) ratio as much as possible. Addressing the test in the early design phase of these architectures will definitely lower their cost, making them more robust and easier to test. The testability constraint is introduced in the form of the test cost. The next section clarifies the method.

4.3. Adding the Testability Constraint in the MOVE Framework

The concept of the test cost as a merit of the testability of one system has also been addressed in some recent papers in the area of the high-level test synthesis, such as [RIE92, CHE93, LeT96, SHA96, GIZ96]. However, none of them was acceptable or applicable in the TTA environment. For example, the testability costs described in [RIE92] and [CHE93] are appropriate for random logic, not for the regular structure that characterizes the TTA. [LeT96] introduces the test cost during the tradeoff in hardware/software codesign partitioning. However, the hardware test patterns were developed according to the high-level hardware description, not its actual implementation. They check the functionality of the hardware/software system and hence, not applicable as such at the gate-level. The approach given in [SHA96] copes with the test cost introduced for the sake of the testable ALU design and it might be useful as a test cost of a particular FU within TTA. Reference [GIZ96] copes with the implementation of BIST in the datapath. The references [HOE99] and [BEA00] indicate interesting approaches to test the VLIW processors based on heuristic functional testing. However, the drawback of functional testing methods in general is their unknown test quality. Experience shows that the fault coverage of typical heuristic functional test is likely to be in the range of 50 – 70 % [ABR91]. Moreover, the functional testing is tightly coupled with specific functional modelling techniques. Deriving the functional model used in test generation is often a manual, time consuming and error-prone process [BOT81].

Obviously, the MOVE codesign flow requires a testing approach different from any of the above mentioned. The appropriate test cost needs to be defined while bearing in mind the characteristics of the TTA structure and the flow itself. The idea behind the proposed approach is to preserve the area-performance ratio as much as possible, obtained after the design-space exploration upon the test-strategy implementation. Hence, the intention is to use as little extra circuitry for the test of the datapath components as possible. Nevertheless, area saving is not the only driving factor; also the fact that the datapath is the most critical part of the TTA in terms of the throughput plays an important role. The datapath components determine the TTA cycle time and it is desirable not to introduce any

additional circuitry inside the datapath as the critical path may be affected. The test cost is introduced according to definition 1:

Definition 1:

Given the two-dimensional design space solution $\upsilon^2(\underline{a}, \underline{t})$, the test cost is the result of function f that translates this design space into a three-dimensional $\Re^3$ $(\underline{a}, \underline{t}, \underline{f_t})$ vector space where the parameters $\underline{a}$, $\underline{t}$ and $\underline{f_t}$ are the vectors of the area, execution time and test cost values, respectively, of the set of resulting instantiations obtained after the design-space exploration:

$$f\left(\vartheta^2\left(\underline{a}\ ,\underline{t}\ \right)\right)=\Re^3\left(\underline{a}\ ,\underline{t}\ ,\underline{f_t}\ \right). \qquad (4.1)$$

Hence, the test constraint brings up another degree of freedom in the design space solutions and is introduced during the high-level design. The test cost of the *i-th* architectural instantiation f_{ti} depends on the vector of the architectural parameters $\underline{x}$, and is expressed in the form of its unique test cost function $f_{ti}(\underline{x})$ as:

$$\underline{x} \rightarrow f_{ti}(\underline{x}) \qquad \underline{x} = x(\underline{c}, \underline{n_p}, \underline{n_b}, \underline{n_{conn}}), \qquad (4.2)$$

where $\underline{c}$ denotes the vectors of component types used in the resulting instantiation, $\underline{n_p}$ is the vector of the number of the test patterns targeting the stuck-at faults of the components, $\underline{n_b}$ is the vector of the number of busses connected to at least one port of the component and $\underline{n_{conn}}$ is the vector of the number of the ports (connectors) of the component. Therefore, the test cost is a function of the datapath parameters only, i.e., it depends on the match of the functional units, register files, busses and sockets.

In principle, the test cost may be calculated for all instantiations in the design space and not only the ones that correspond to the Pareto points. However, the designer may restrict that space giving the lower bounds to the values of area, values of throughput or both of them.

The characteristic of regularity and modularity of the architecture, i.e., the fact that each FU and RF from the datapath is accessible via a MOVE bus and the sockets, is exploited during the calculation of the test cost. As already indicated, the components may already be predesigned up to the gate-level, so that the actual area and delay of each component are back-annotated into the library of the components and used during the design-space exploration phase. The same holds for its actual stuck-at test patterns that can be generated using an automatic test pattern generation (ATPG) tool that determines the test cost of the instantiation. Before the analytical expressions for the test cost of the components are given, the implementation details of the interface between components and sockets need to be revealed and discussed. Also, the data transport mechanism via the components and sockets has to be explained.

4.3.1 Data-transport scheduling within the TTA

The block-diagram in Figure 4.5 shows the interface between the datapath component and interconnection network. The TTA design itself is synchronous. This Figure shows an arbitrary two-input, one-output component and its additional blocks controlling the data

flow in the component. In general, components can have an arbitrary number of input and output ports followed with the appropriate registers and sockets. However, there has to be one trigger register (*T*) that actually triggers the operation of the component one cycle after the data arrival at its input and in combination with *enable1* signal. The *O* and *R* are the operand and the result registers, respectively. In the recent version of the MOVE architectures, the result register may be omitted, increasing the overall throughput. The component itself may have multiple numbers of pipelined stages and may take multiple clock-cycles to execute the operation. The *Pipeline Control* block is responsible for providing the correct time scheduling of the data flow through the components. The components from the control block are connected to the interconnection network in the same fashion as their datapath counterparts. They may be communicating among themselves using the *local input/output port* connectors.

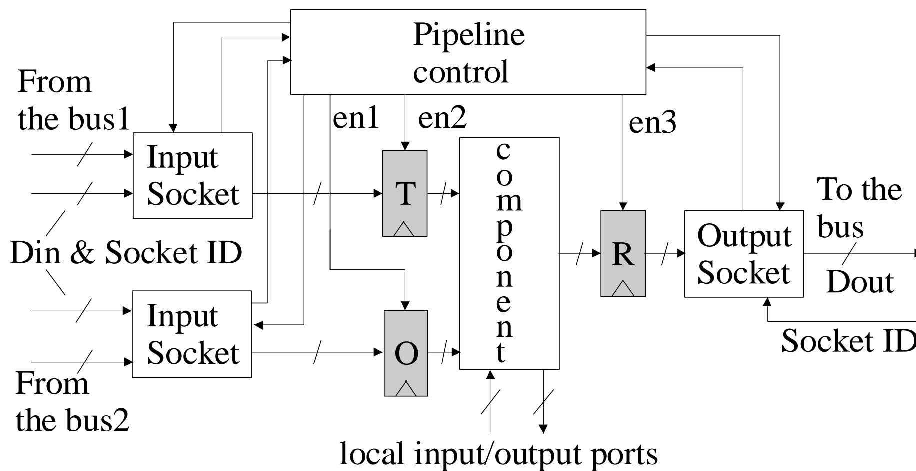


Figure 4.5. The interface between a component and the interconnection network

The principal schematic of the input-socket implementation is shown in Figure 4.6. The exact implementation will be dealt with in the next chapter. Now, it is important to observe the role of the sequential element F_{in} that enables the data (*Din*) storage to the input register, if the component is selected via its corresponding socket ID.

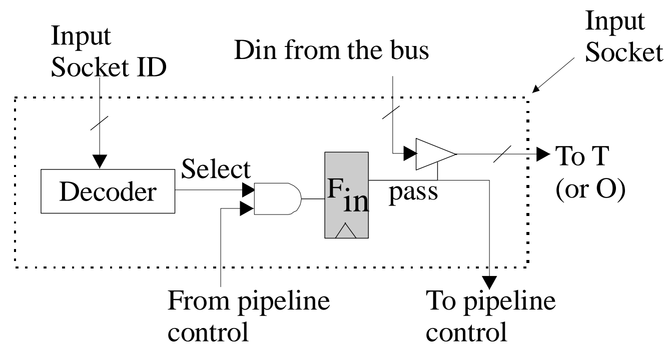


Figure 4.6. Scheme of the input socket

The output socket has to provide the reading (moving) of data from the component to the bus. The flip-flop F_{out} in the output socket controls this move, as outlined in the Figure 4.7.

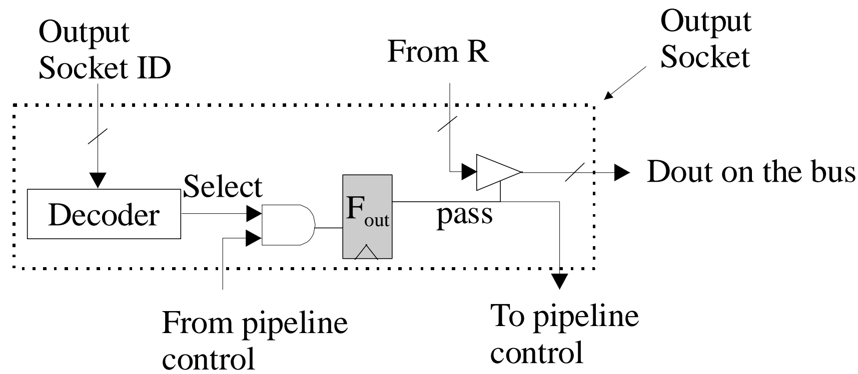


Figure 4.7. Scheme of the output socket

Definition 2:

Let $C_i(r)$ denotes the cycle when the actual data transport of the *i-th* operation to the register *r*, $r \in \{O, T, R, F_{in}, F_{out}\}$ takes place, where *O*, *T* and *R* are N-bit registers (with N as a data bus width) and F_{in} and F_{out} are considered as single bit-registers.

Definition 3:

The operations (*i* and *j*) are called to be mapped in order, if and only if:

$$i < j \Leftrightarrow C_i(r) < C_j(r). \quad (4.3)$$

Using the above definitions, one is able to define the scheduling restrictions that will also be used for the test. The timing-transport relations among the registers can now be expressed as:

$$C_i(T) - C_i(O) \geq 0, \quad (4.4)$$
$$C_i(R) - C_i(T) \geq 1, \quad (4.5)$$

In words, this means that the data cannot appear in the trigger register before the data is scheduled in the operand register (4.4). In addition, the data processing inside the component itself requires at least one cycle (4.5), depending on the number of pipeline stages. Actually, the relation (4.5) determines the internal latency (l_c) of the component, which is expressed as [COR94]:

$$l_c = (s - 1)d + 1 > 1 \quad (4.6)$$

where *d* denotes distance (in clock cycles) between pipeline stages, while *s* is the number of pipeline stages.

Furthermore:

$$C_i(T) > C_j(T) \Leftrightarrow C_i(R) > C_j(R) \quad (4.7)$$
$$C_i(T) > C_j(T) \Leftrightarrow C_i(O) > C_j(T) \quad (4.8)$$

The instructions in the same functional unit are executed sequentially and the operand value must not be overwritten unless it has been previously used, as the relations (4.6) and (4.7) reveal. Finally, the instruction decoding takes also at least one cycle, according to the inequalities (4.9 - 4.11):

$$C_i(O) - C_i(F_{in}) \geq 1 \quad (4.9)$$
$$C_i(T) - C_i(F_{in}) \geq 1 \quad (4.10)$$
$$C_i(F_{out}) - C_i(R) \geq 1 \quad (4.11)$$

The control block, being instruction fetch and load/store unit of the processor together with pipeline controller mechanism from Figure 4.5, ensure that these conditions are fulfilled.

Definition 4:

Let $CD_c(t_i, t_j)$ denotes the difference in clock cycle between the transports in the timing slots t_i and t_j for component *c*.

The minimum number of clock cycles necessary for the completion of the *i-th* operation of the component in Figure 4.5, in the case the data arrives at the same time in operand and trigger register is equal to 3, referring to relations (4.4), (4.5), (4.10), (4.11):

$$CD_c(t_{Din}, t_{Dout}) = (C_i(T) - C_i(F_{in})) + l_c + (C_i(F_{out}) - C_i(R)) \geq 3. \quad (4.12)$$

where t_{Din} and t_{Dout} denote the cycles in which the data is applied to or read from the MOVE bus, respectively. Hence, the *n-th* operation of the same component may be completed in the timing slot *n+2*, expressed with (4.5) and (4.10-4.11) if the hybrid-pipelining scheme from the Figure 4.2a is applied. However, the execution time increases if the operand and trigger registers are connected to the same bus since the data cannot appear at the same time in the operand and the trigger register in that case. The architecture demands an additional timing slot for data fetching into the operand register before the data may be set into the trigger register (relations (4.4) and (4.8)) resulting in:

$$CD_c(t_{Din}, t_{Dout}) = (C_i(O) - C_i(F_{in})) + (C_i(T) - C_i(O)) + l_c + (C_i(F_{out}) - C_i(R)) \geq 4. \quad (4.13)$$

For example, Figure 4.8 shows two identical components ($FU_1 = FU_2$) where the $CD_{fu1} < CD_{fu2}$ due to their different port connections to the interconnection network. More particular, two input ports of FU2 are connected to the same bus.

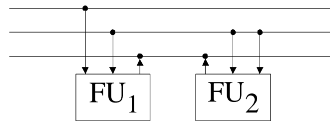


Figure 4.8. Two identical functional units with different execution times

The execution time will be also increased in case when the result register is tied to the bus to which either operand or trigger registers are also connected. In addition, the number of cycles for the execution will further increase if all registers are tied to the same bus.

This analysis is important for determining of the test time per component if a functional test is to be applied. Before the analytical expressions for the test cost are given, it is important to explain the test style as proposed in our method.

4.3.2 Functional-structural test style of TTA

As already explained, the TTA processor has one or more busses that are programmed by assigning a data transport (move) to each of them. The instruction format has the same form as a VLIW, i.e. each instruction consists of multiple operations. In the case of the TTA, it is only the “move” operation. Hence, an instruction for a TTA with n-busses has the format according to Figure 4.9.

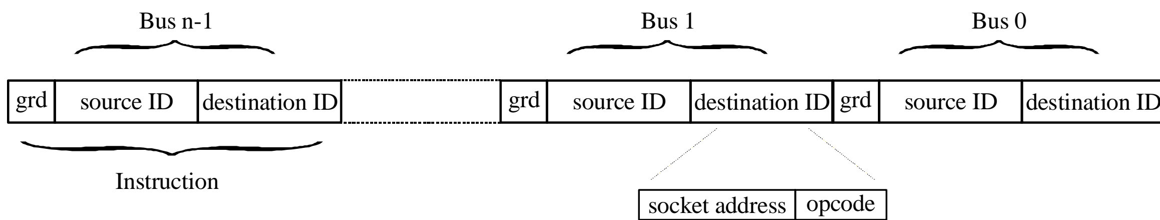


Figure 4.9. Instruction format of the TTA

Each operation specifies the data transport from a source register to a destination register. The *source ID* selects a component from which the result register content will be read, i.e., transferred to the bus, while the *destination ID* selects a component to receive the data from the bus, i.e. the component whose trigger or operand register will be written. *Grd* denotes the so-called “guard expression” [COR98a] in order to determine whether the move is executed or not (if the *grd* field evaluates to 0, the corresponding move will not be executed). Both the source and destination IDs contain a socket physical address. In addition, if the component requires, a number of bits may be used as an opcode to select the operation that component has to execute. It is obvious from Figure 4.9 that the number of control busses has to be equal to the number of data busses in the design and they all originate form the instruction fetch unit. Of course, it is clear that the instruction fetch unit needs to be connected to all existing sockets in order for them to perform the decoding step.

The structure of the pipelined component and its instruction format shown in Figure 4.5 offers a convenient environment for the application of the functional tests for the components without adding any extra DfT circuitry. Hence, the pipelining of the operation can be used to schedule the subsequent test patterns of the components into the first possible consecutive timing slots, according to the inequalities from the previous subsections. Figure 4.10 clarifies this situation in the case of a component with two inputs and one output if the two-stage pipelined scheme is used. The boxes with letters inside denote the relative position of the test patterns within the registers (flip-flops) for a particular timing slot.

Figure 4.10 can be interpreted in the following way. At the beginning, in the instruction fetch phase, the instruction cache lookup is performed and the instruction containing the test patterns is transported to the bus. It means that the data from the result register of Load/Store unit will be transferred to an appropriate bus, corresponding to the timing slot $R \rightarrow F_{in}$ where *Fin* denotes the flip-flop in the input sockets of the components connected to that bus. If the instruction carries the appropriate ID-code, the "pass" signal will be

issued in the corresponding input socket by its decoder; hence, the pattern is transported and stored into its trigger (T) or operand (O) register. This corresponds to the decoding phase ($F_{in} \rightarrow T(O)$). Next, after one or more timing slots, depending on the internal latency of the component, the response of the application of the patterns shows up in the result register. The pattern response can be read in the same way, with the difference that the data has to be transported to the LD/ST unit. Hence, the pipelining mechanism of the execution of an operation can also be applied to pipeline the test patterns. Figure 4.10 (dashed box) shows that while the first pattern is stored in the trigger (operand) register, the second one can already be in front of the input registers waiting for the "pass" signal.

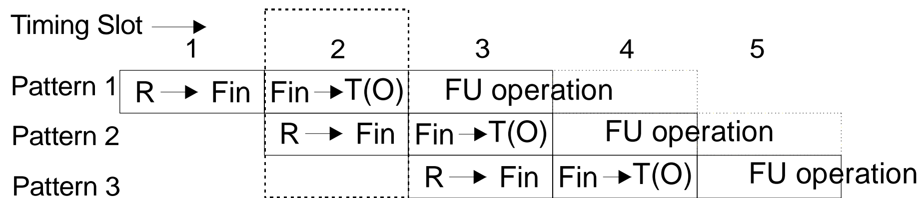


Figure 4.10. The pipelined execution of a functional test for the datapath component

Even though the test is performed in a functional fashion, it is not completely functional because the test patterns of each component within the data path are structural. They may be precalculated and assigned to the library of the components that is used during the high-level synthesis. This is feasible because of the flexibility of the TTA; the inputs and outputs of the datapath components are all registered (see Figure 4.5). It allows the designer to design them separately, optimise them, run ATPG and generate structural test patterns that target stuck-at faults.

The test pattern generation for various components is feasible after their synthesis using any commercial high-level synthesis tool. In our case, the *Synopsys Design Compiler* (DC) tool has synthesized the various components. As ATPG tool, the already mentioned Philips in-house industrial tool, applied during the core-based testing explained in the previous chapter, has been used for this purpose. The test-pattern generation results are stored in a so-called “canned” test set per component. Moreover, the library of MOVE has been extended with different architectures per component, each with a different area, delay and number of test vectors. This provides the designer with the possibility to use and analyse different implementations.

Therefore, the test attribute is adjoined to each component in the library, expressed as the number of test vectors that are necessary to test that unit. For instance, the Adder functional unit has two different implementations: Ripple-Carry and Carry-Look-Ahead (CLA) implementation, each with various attributes adjoined as Table I illustrates in the case of their 16-bit implementation. The multiplier has also various implementations built either in one of the iterative cellular array fashions (Brown’s, Pezaris, Baugh-Wooley’s or logarithmic) or standard and recoded multipliers. The same holds for all other components used in the library. Hence, the designer can make trade-offs with slower and less expensive solutions in terms of area and test cost or faster adder implementations using more silicon and having higher test cost.

The concept of the so-called *C-testability* [FRI73] has been applied in the case of the Ripple-Carry adder, thus providing a low test-cost. Normally, the number of the test patterns increases if the number of bits per component increases. However, if the circuit is C-testable there exist a certain set of test patterns whose number is independent with respect to the number of bits. These components can be adders (if built as ripple-carry),

subtractors, barrel-shifters and multipliers (if built as modified iterative cellular arrays such as the Baugh-Wooley multiplier [SHE84]). Note that the number of the test patterns for a ripple-carry adder equals to 7, not 8 as presented in some earlier publications such as [FRI73], [GIZ96]. Additional implementation details of a number of components within MOVE will be given in the next chapter that discusses the gate-level test implementation of our strategy.

TABLE I TWO 16 BIT-ADDER IMPLEMENTATIONS AND THEIR ATTRIBUTES

Type	Area (μm^2)	Delay (ns)	Power (mW)	# test vectors	FC (%)	Latency (# clock cycles)
Ripple-Carry	6.912	8.06	7.2598	7	100	1
CLA	11.232	3.79	10.168	18	100	1

The structural test patterns will be applied using the data buses. However, the surrounding circuitry of the component i.e. the functional signals of the sockets has to be set in an appropriate way as Figure 4.11 reveals.

According to that Figure, the sockets ID will select the component to receive the data, i.e. test patterns from the bus or the component whose response will be placed onto the bus. Note that the DfT circuitry is not introduced in any form inside the data path, thus preserving the area/execution time ratio from the design space exploration phase completely.

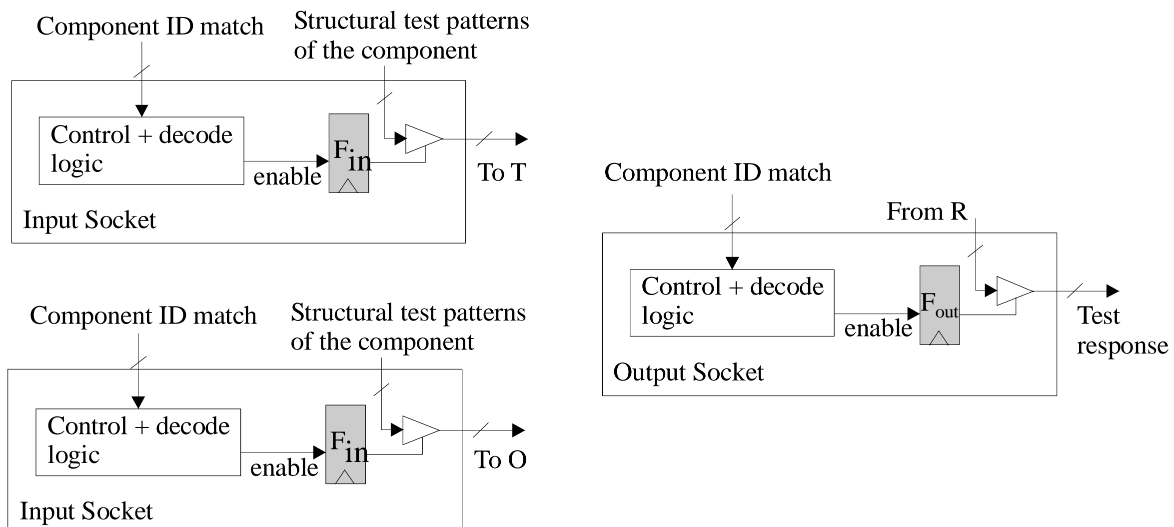


Figure 4.11. The functional test path via the sockets of the components

Therefore, the datapath of the processor and interconnection network will be tested using the assembler program of the processor. Running this type of test will decrease, in most cases of the practical implementation of the TTA, the total number of test cycles compared to the other test styles. For example, suppose there are three data buses in the design, the latency of the adder unit is equal to 1 and the two-stage pipeline scheme has been used. In that case the assembler program that tests the Adder block of the processor has the form shown bellow in the Table II:

TABLE II THE MOVE ASSEMBLER

Slot	Bus1	Bus2	Bus3
1	$R_1(LDST)_1 \rightarrow O(Add)_1$;	$R_2(LDST)_1 \rightarrow T(Add)_1$;	
2	$R_1(LDST)_2 \rightarrow O(Add)_2$;	$R_2(LDST)_2 \rightarrow T(Add)_2$;	
3	$R_1(LDST)_3 \rightarrow O(Add)_3$;	$R_2(LDST)_3 \rightarrow T(Add)_3$;	$R(Add)_1 \rightarrow T_1(LDST)_1$;
4	$R_1(LDST)_4 \rightarrow O(Add)_4$;	$R_2(LDST)_4 \rightarrow T(Add)_4$;	$R(Add)_2 \rightarrow T_1(LDST)_2$;
.			
.			
.			
n_p	$R_1(LDST)_{np} \rightarrow O_1(Add)_{np}$;	$R_2(LDST)_{np} \rightarrow T(Add)_{np}$;	$R(Add)_{np-2} \rightarrow T_1(LDST)_{np-2}$;
n_p+1			$R(Add)_{np-1} \rightarrow T_1(LDST)_{np-1}$;
n_p+2			$R(Add)_{np} \rightarrow T_1(LDST)_{np}$;

where $R_i(c)_j$, $O_i(c)_j$ and $T_i(c)_j$ denote the *i-th* result, operand and trigger registers of *c-th* component, respectively, containing *j-th* pattern or pattern response. n_p is the number of structural patterns of the adder. Note that the proposed functional-structural test style enables at-speed test of the datapath part. Hence, the timing problems in the final implementation can also become transparent using the proposed test method, as the execution of a test corresponds to the execution of an assembler program.

It takes three cycles in total for completing the adder operation with the two-stage pipeline scheme employed in the design. However, attention should be paid that another condition needs to be fulfilled, in order to accomplish that the execution of one test pattern takes a minimal number of cycles: the Load/Store unit requires two result registers. Moreover, they have to be connected in such a way to allow the simultaneous application of the input patterns for both adder input terminals. The next section will discuss this issue in more detail while providing the analytical expressions for the test cost of the datapath components.

4.4. The Test Cost Function

Bearing in mind there will be no additional DfT circuitry within the datapath, the test cost function will be expressed with respect to the number of cycles necessary to execute the tests, i.e. the cost is related to the testing time. The analytical expression for the test cost (f_{tc}) of the component, valid both for functional units and register files is defined by:

$$f_{tc} = n_p \bullet \max\left(\left\lceil \frac{n_{port}}{n_b} \right\rceil, \left\lceil \frac{n_{in}}{n_{out}(LD/ST)} \right\rceil, \left\lceil \frac{n_{out}}{n_{in}(LD/ST)} \right\rceil\right) + CD_c(t_{Din}, t_{Dout}) \qquad (4.14)$$

where n_p is the number of structural test patterns, with n_{port} denoting the number of component ports. n_b denotes the number of busses, which have at least one connection to the ports of the functional unit. Therefore, the test cost of a functional unit will increase with the ratio n_{port}/n_b in the case when the number of component ports exceeds the number of busses they are connected to (see Figure 4.8). However, the connectivity of the Load/Store unit needs to be taken into account as well, as this unit applies the structural

data patterns to the bus and reads the response. Hence, in order to apply a minimum required number of test cycles to test the component, the number of input ports of the functional unit (n_{in}) needs to be smaller than or equal to the number of output ports of the Load/Store unit ($n_{out}(LD/ST)$). The same relation holds in the case if the functional unit has multiple outputs (n_{out}). In that case, the inequality $n_{out} \leq n_{in}(LD/ST)$ holds. It is assumed that the connectivity of the Load/Store unit does not create the problem, i.e. that each bus in the design has at least one connection to the input and output port of the unit. Figure 4.12 clarifies this situation, showing the physical path of the test patterns for two different designs of a Load/Store unit (two outputs and one input in Figure 4.12.a and one input and one output in Figure 4.12.b). The ports and connectivity of the Load/Store unit in Figure 4.12.a enable one test pattern being applied in one single timing slot. On the other hand, due to the presence of only one output port (consequently: one result register) in the configuration of Load/Store in Figure 4.12.b, the test pattern needs to be divided and distributed over two timing slots (slot I and slot II in Figure 4.12b).

The term CD_c in equation *(4.13)* is defined according to *(4.12)* and, despite the dependencies on the component internal latency and employed pipeline mechanism, has a constant value for particular component.

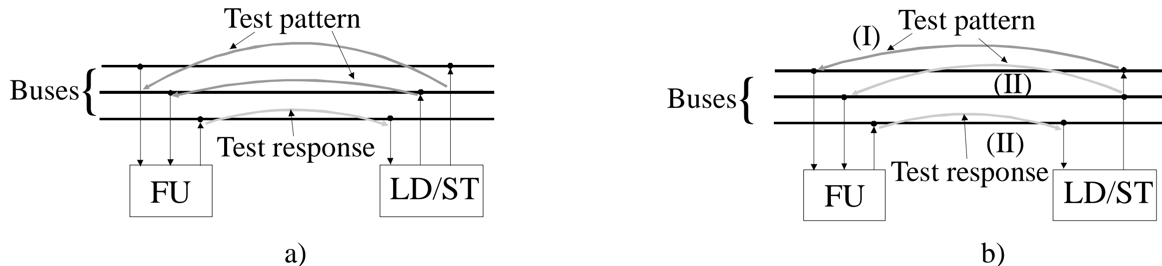


Figure 4.12. The physical test path

The same formula *(4.13)* is valid for the register files where n_{in} and n_{out} denote the number of input and output ports, respectively. Nevertheless, the latest version of MOVE architecture uses register files with not more than two input/output ports. Large register files are partitioned into two input/output-port registers [JAN95]. For register files, n_p in the equation *(4.13)* denotes the number of the marching test patterns [GOO91], which depends on the number of locations within the register file. The same statements with respect to the connectivity to the buses and mutual relations with Load/Store unit, valid for functional units, are applicable in this case, as well. It is assumed that the register files are implemented as a multi-port memory, not as a set of flip-flops.

===

Examples

Consider the component FU given in Figure 4.12.a. Assuming that the number of back-annotated test-patterns amounts to 36, while $CD_{fu} = 3$, its architectural parameters equal to:

$n_p = 36,\ n_{port} = 3,\ n_b = 3,\ n_{in} = 2,\ n_{out} = 1,\ n_{in}(LD/ST) = 1,\ n_{out}(LD/ST) = 2.$

The test cost function then equals to:

$$f_{t_{fu}} = 36 \cdot \max\left(\left\lceil \frac{3}{3} \right\rceil, \left\lceil \frac{2}{2} \right\rceil, \left\lceil \frac{1}{1} \right\rceil\right) + 3 = 39$$

The test cost of the same component however changes if a different Load/Store unit as given in Figure 4.12.b has been used, because $n_{out}(LD/ST) = 1$:

$$f_{t_{fu}} = 36 \cdot \max\left(\left\lceil \frac{3}{3} \right\rceil, \left\lceil \frac{2}{1} \right\rceil, \left\lceil \frac{1}{1} \right\rceil\right) + 3 = 75$$

The component FU_2 from Figure 4.8 has $n_b = 3$. With the Load/Store Unit from Figure 4.12.a, (assuming the same number of the test patterns n_p) the test cost is determined as:

$$f_{t_{fu}} = 36 \cdot \max\left(\left\lceil \frac{3}{2} \right\rceil, \left\lceil \frac{2}{2} \right\rceil, \left\lceil \frac{1}{1} \right\rceil\right) + 4 = 76\,.$$

==

The sockets, controlling the data flow in the interconnection network, have not been tackled structurally during the proposed test method. Nevertheless, they are tested in a functional way, by setting the control signals carrying the socket IDs, according to Figure 4.11. In that case, they will pass (or not pass) the structural test patterns to the components. Of course that will decrease the fault coverage of the overall processor, as the structural test patterns of the components do not target the faults inside the sockets, but that is the price paid by not introducing any DfT inside the datapath. In principle, it is possible to introduce scanable flip-flops in the sockets and component's trigger, operand and results registers of the component and subsequently connect them into scan-chains, [ZIV00a, ZIV00b]. This is only useful if a high fault coverage is the principal target. That will of course, increase the overall cost of the processor, not only in terms of testing but also in terms of area and throughput, as well. This will be discussed in the next chapter.

Before the total test cost is given, it is desirable to tackle another item related to the connectivity optimization. That is the test scheduling, i.e., the possibility to test more than one component at the same time.

4.5. Test Scheduling

The fully connected interconnection network, in which all components' ports in the datapath are connected to every bus, is obviously overkill. Reducing the connectivity is important for processors that exploit instruction-level parallelism. It may not only reduce the chip area, but may also shorten the processor cycle time, thus increasing the speed performance. The transformation from the fully connected structure into a partially connected configuration also takes place during the high-level phase of the codesign, after the design-space exploration. The MOVE environment offers the possibility to remove the connections and associated sockets in an iterative way. The connectivity optimisation is carried out with respect to the architecture with a constant number of components and busses. Each time when a connection is removed, the throughput is recalculated. Hence, the designer can clearly see how many connections may be dismissed without degradation in performance. However, this can have an impact on testing, as well. Even the test cost of one particular component may be changed as the following example demonstrates.

Figure 4.13 depicts an arbitrary architecture of two functional units FU1 and FU2 with test cost f_{t1} and f_{t2} respectively. It is obvious that the test patterns of the two components in the architecture "*a*" can be scheduled together since each connector of the components has at least one socket assigned to a separate bus. Therefore, the test cost for the two units is equal to max(f_{t1}, f_{t2}) since the two sets of patterns can be scheduled in parallel. However, that will not be possible for architecture "*b*". In this case, FU2 misses one connection (or has one socket at a "wrong" position) so that the test patterns of the two components can not be scheduled into the same cycles. They have to be applied sequentially and the test cost for "*b*" is equal to (f_{t1} + f_{t2}). The architecture "*c*" has the worst testability: not only that the test patterns cannot be scheduled with the other component, but also the testing time and in turn the test cost for FU2 itself has increased since the component connectors are connected to the same bus (input and output socket on same bus). Hence, for architecture "*c*" the test cost is larger than the test cost for architecture "*b*" since f_{t2} increased due to the fact that one single pattern can not be scheduled within one cycle.

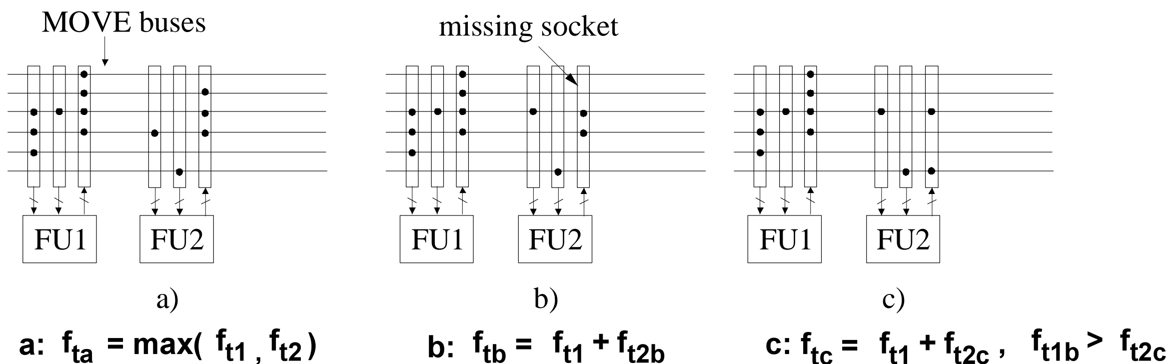


Figure 4.13. Structures with decreased testability (a, b, c)

Obviously, according to this small example, the calculation of the test cost for each component has to take place after the connectivity-optimisation phase. Moreover, in order to minimise the test time, an algorithm to find the minimal test-scheduling scheme for a given architecture needs to be developed.

There are many algorithms devoted to the problem of scheduling in high-level synthesis, such as for example [WAL95, STO91]. A good overview of scheduling techniques has been given in [STO94]. However, none of the previous problem presentations fit in our case. For instance, while deriving the test-scheduling scheme, one has to assign test patterns for the component along the buses in such a way that each bus transports the test patterns for different connectors of components. Therefore, the binding problem as indicated in reference [GAJ94] also appears in our case. The exact formulation of the problem is as follows:

> For a given architecture and recalculated test costs of each component c_i $(i= 1,\dots,n_c)$ with the given set of test patterns, assign the socket-bus pairs (s_j, b_j), $(j= 1,.. n_{conn,i})$ to the components, such that the test time of the whole architecture is minimal. n_c and $n_{conn,i}$ are the numbers of the components in the architecture and the number of the connectors of the *i-th* component.

A heuristic algorithm, of which the pseudo-code is given in Figure 4.14, tries to minimise the number of test cycles for a given architectural instantiation in a "greedy" way [ZIV00c]. The algorithm is creating sub-lists of the components for the parallel test

while trying to minimize their number by having as many components as possible within one sub-list. Finally, it tries to put the components with the highest test cost into the same sub-list. While determining which components may be scheduled in parallel, the algorithm may use backtracking and change the binding of the sockets and busses involved. The result of the algorithm is a list of components whose test-vectors can be scheduled together.

```
Begin
  Input: The Datapath architecture network
    D_l becomes the list of components c_i i = 1,…,n_c in
    decreasing order of their test costs;
    j := 1;
    while D_l ≠ Ø do
      List [j] := Ø;
      n_l := j;
      for c ∈ D_l do
        if c can be scheduled in parallel with other
        components in List [j] then
          List [j] := List [j] ∪ c;
          D_l := D_l - {c};
        end if;
      end for;
      j := j + 1;
    end while;
  Output: List [j], j = 1,…n_l
end
```

Figure 4.14. The test-scheduling algorithm

Even though not optimal, the algorithm indicates the upper bound of the test time, and afterwards, an additional optimisation may take place. Obviously, the additional optimisation requires full use of the interconnection network in time. It means the simultaneous application of test patterns to all buses, whenever possible. Furthermore, it would mean that the test patterns of various components should be split and distributed over the different timing slots. However, it has been estimated (and it proved correctly, as will be demonstrated in the next chapter) that further optimisation will not significantly decrease the test time and hence it has not been tackled.

The total test cost f_t of the whole datapath is now equal to the sum of the highest test-cost component from each list. There are as many terms in that sum as there are scheduling lists, so the total sum of the architecture can be expressed as:

$$f_t(\underline{x}) = \sum_{i=1}^{n_l} \max_i (f_{tc_k}(\underline{x}), k = 1,...,n_{ki}) \qquad (4.15)$$

where n_l denotes the number of scheduling lists, n_{ki} the number of components within *i-th* list and $\underline{x}$ is the vector of the architectural parameters, as described at the beginning of this

chapter. Note that the function f_t refers to the function f given by equation (4.1) that translates the 2D into 3D vector space when applied to each of the instantiations of the architecture.

If it is decided that the datapath test will be supplemented with the individual tests of the sockets, the total test cost needs to be reformulated by adding another term:

$$f_t(\underline{x}) = \sum_{i=1}^{n_l} \max_i (f_{tc_k}(\underline{x}), k = 1,...,n_{ki}) + f_{ts} \qquad (4.16)$$

The function f_{ts} depends on the particular test style that is used to test the sockets (full-scan, BIST, etc.). However, note that in this case a certain displacement of area-execution time points in the design space solutions may occur if the test style requires DfT insertion in any form, as both the area and throughput change in that case.

4.6. CAD Implementation

The proposed methodology to insert the test constraint in the high-level phase of the codesign has been incorporated into the MOVE framework according to Figure 4.15 showing the modified codesign and test flow.

The first part of the approach until the design space exploration remains the same as in Figure 4.3. The changes become apparent during the design-space exploration when the designer has a multiple choice of various components concerning the area, throughput and test. The resulting set of architectural instantiations has a fully connected interconnection network, i.e., each port of the components is connected to all busses in the architecture. The CAD flow proceeds with the connectivity optimization phase in which the connections are iteratively removed, while keeping the throughput the same or even better [COR98a]. The connectivity optimisation is not being performed for each of the instantiations in the design space obtained after the design-space exploration, but only for the instantiations lying on the Pareto curve. One may object that in that case, the resulting structure is not necessarily optimal after the connectivity optimisation. Another structure above the Pareto curve may have better overall performances after the connectivity optimisation. Actually, the connectivity optimisation should have been tackled as an additional parameter during the design space exploration. However, that has not been implemented inside the MOVE package, because it would further extend the already long computational time. For example, the CPU-time required to perform the design-space exploration followed with connectivity optimisation of an average *C*-application is several days. The machine that has been used in our case is Hewlett Packard HP 9000-785/J 5000 workstation (processor type 2 × PA-8500, 440 MHz + 512 Mb RAM) running on UNIX operating system HP-UX 10.3.

The CAD flow resumes with the test cost calculation applying the analytical formulas *(4.12-4.16)*. It is carried out for each of the architectural instantiations and it is followed with the test scheduling. The selection of resulting instantiations can be carried out using any of the standard weighted norm techniques within the vector space $\Re^3$ [MIL91]:

$$\|\underline{u}\|_q = \left(\sum_{k=1}^{3} |w_k u_k|^q \right)^{\frac{1}{q}} \qquad (1 \le q < \infty) \qquad (4.17)$$

where $\underline{u} = u(\underline{a}, \underline{t}, \underline{f_t})$, w_k is the weight function expressing the significance of the area, execution time and test constraints in the overall design and q is the space dimension. Of course if $q=2$, the norm is the Euclid one, $|\underline{u}|_E$.

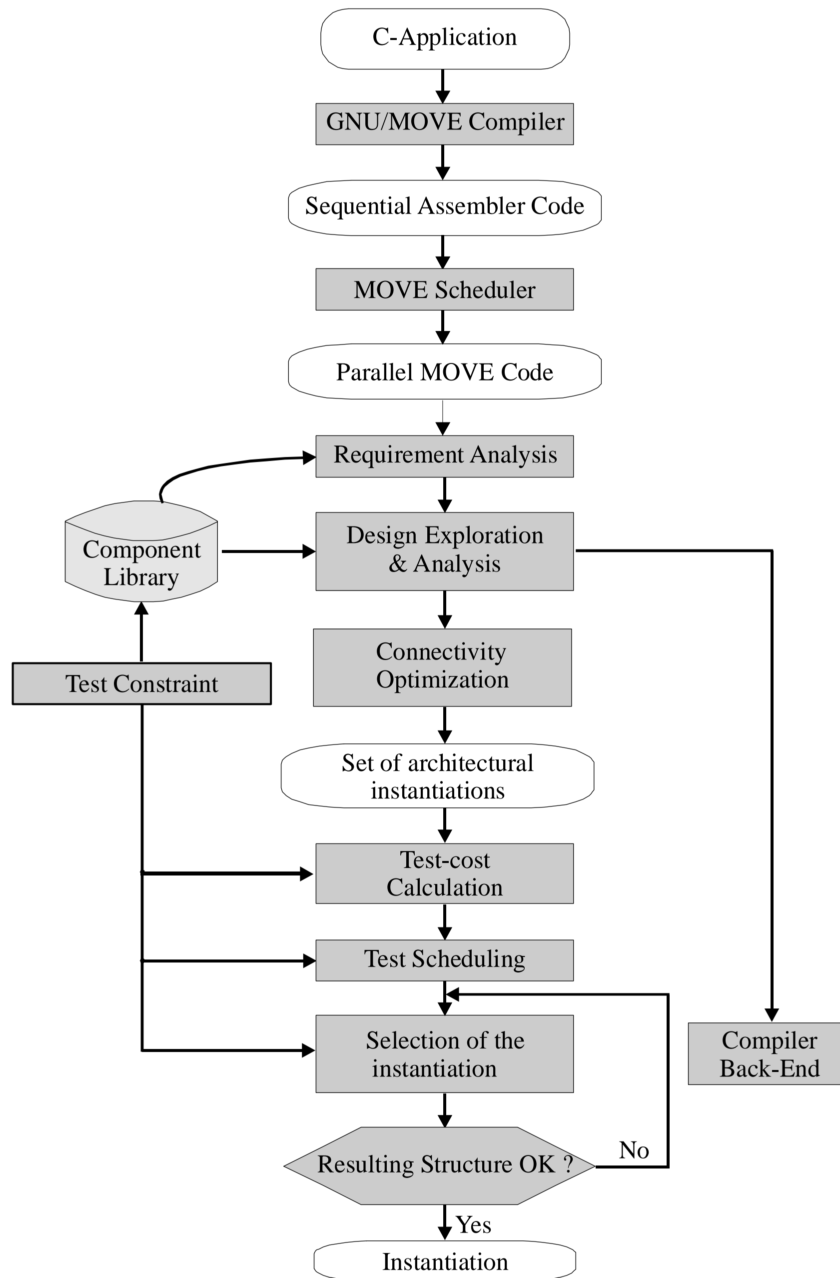


Figure 4.15. The test constraint inserted into the MOVE codesign package

===

Example

Consider the set of M instantiations inside the solution space performing the same application consisting of different number and types of functional units and register files and their mutual connectivity, as given in Figure 4.16. Let us assume there are n different

types of functional units and m types of register files available during the design space exploration. Assume also that the number of busses (n_b) also varies from one to another instantiation. Therefore, the test cost of each component for each instantiation is first calculated. These values are given in Table III. The test costs of each instantiation f_t will be obtained after the test scheduling. The total test cost for each of the instantiations is given in the last row of the Table III.

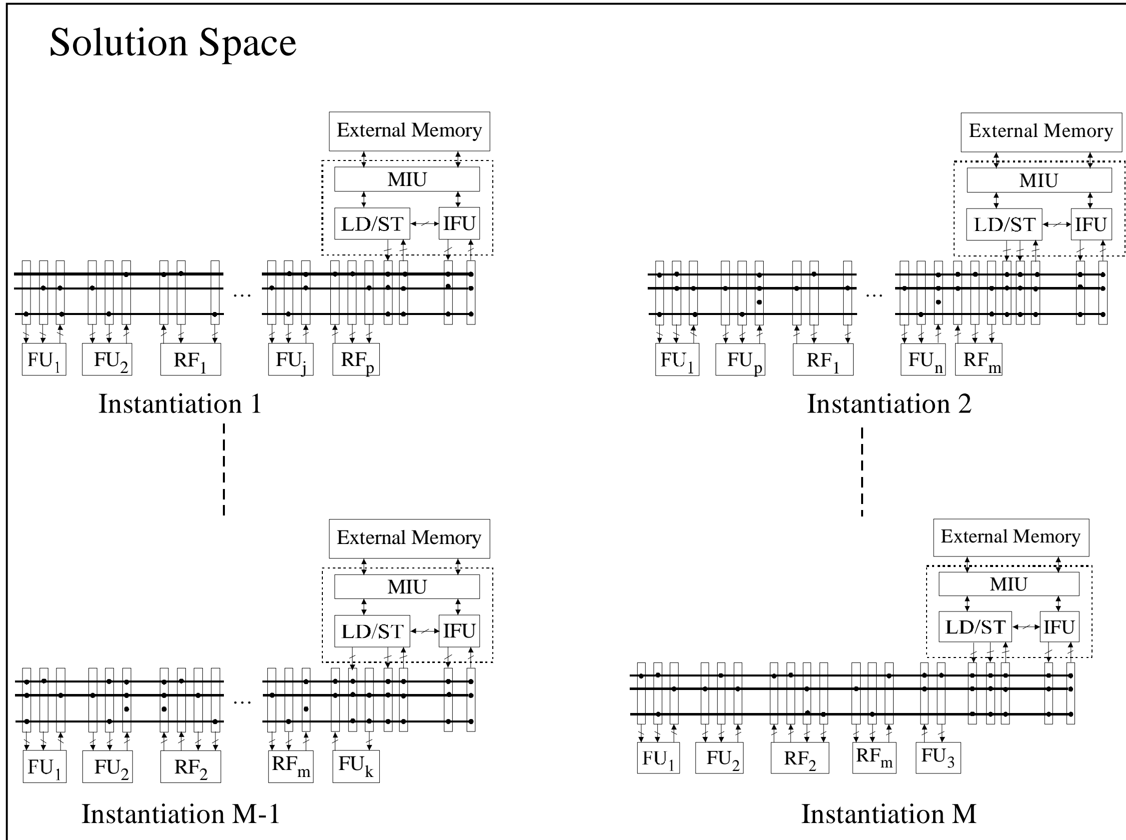


Figure 4.16. The instantiations in solution space

TABLE III THE TEST COST FUNCTION OF DIFFERENT INSTANTIATIONS

Test cost	Instantiation_1	Instantiation_2	. . .	Instantiation_M
$f_t(fu_1)$	137	69		69
$f_t(fu_2)$	70	-		70
. .				
$f_t(fu_n)$	-	11		-
$f_t(rf_1)$	150	74		
$f_t(rf_2)$	-	-		194
. .				
$f_t(rf_m)$	-	98		48
n_b	8	6		3
f_t	1278	383		498

The instantiation with the best performance ratio (area-execution time-test) was chosen for further hardware implementation while the weight values are left to the designer. For example, if the highest priority is given to the speed, and the lowest priority to the area, their weight coefficients have values such that $w_t >> w_f >> w_a$ holds (where w_t, w_f and w_a denote the weight factors of execution time, test and area, respectively). Of course, if the designer is not completely satisfied with the resulting instantiation, he may again perform the selection process with different weight factors.

The extremely long computational time has resulted in the fact that the test constraint has been implemented after the design-space exploration and not during this step. Introducing additional constraints into the current implementation of the design exploration would inevitably lead to an impractical solution from the point of view of the computing time. All the procedures concerning the testability constraint are "external" with respect to MOVE. They are written in the *GNU C++* programming language and are adjoined to the MOVE codesign package with the *Gawk* script language to convert the formats of the files whenever necessary.

Our test strategy is illustrated by several examples of various *C*-applications such as "*Crypt*", "*Compress*" and "*Eight-Queen*" applications [PAT87, WIN89]. The *C*-code of the applications is applied to the MOVE tool, the code analysis, compilation, profiling and scheduling within the tool are all performed and then the design-space exploration and connectivity optimization are carried out on the set of given types and number of components. This results in a set of Pareto points (see e.g. Figure 4.4). Next, the test-cost calculation followed with the test scheduling is performed, which results in a set of Pareto points in the $\Re^3$ vector space. The Figures 4.17, 4.18 and 4.19 show the circuit area, execution time and test cost ratio for the "*Crypt*", "*Compress*" and "*Eight-Queens*" applications, respectively.

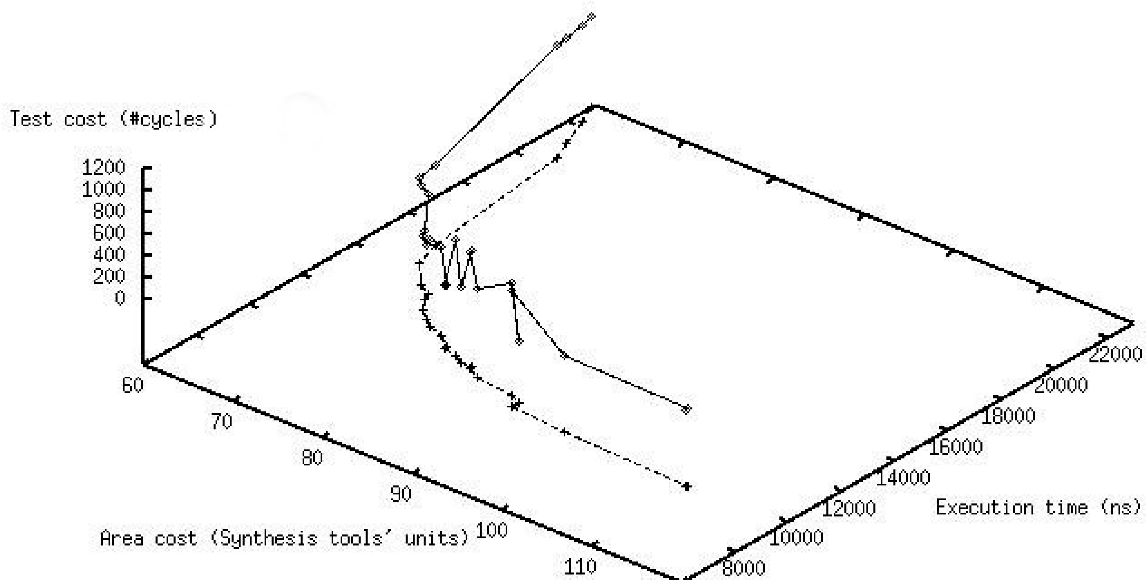


Figure 4.17. The 3D Pareto points for the "Crypt" application

Note that the already achieved area-execution time ratio is preserved since the first projection of the 3D curve of the "Crypt" application form Figure 4.17 into the area-execution-time plane is still the curve from Figure 4.4.

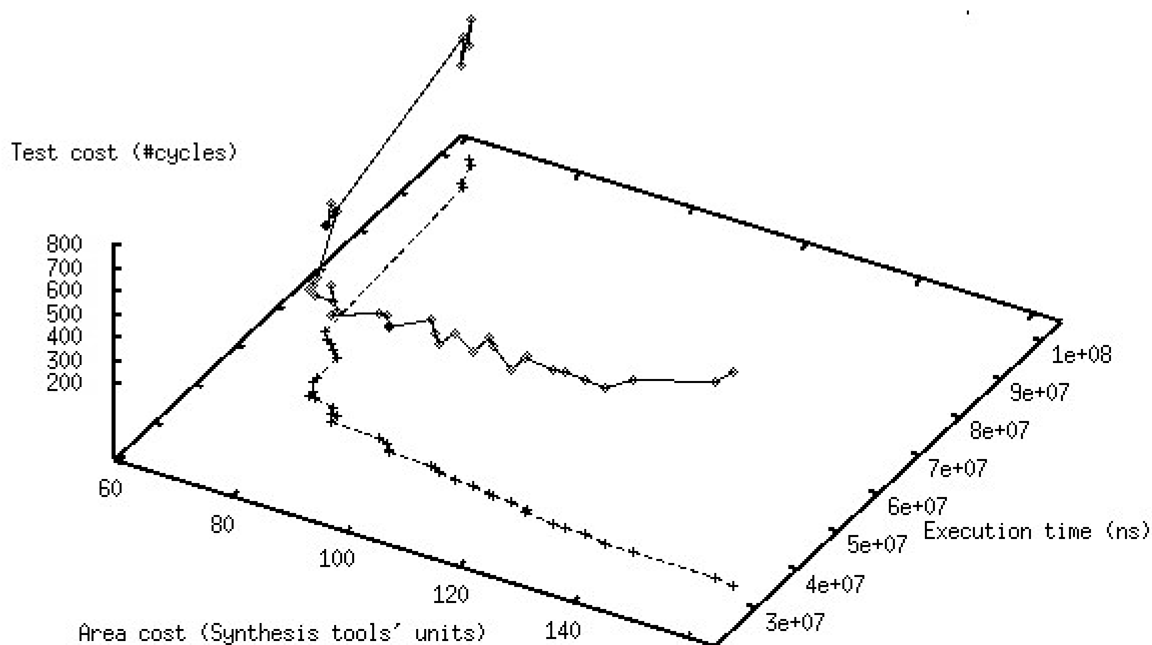


Figure 4.18. The 3D Pareto points for the "Compress" application

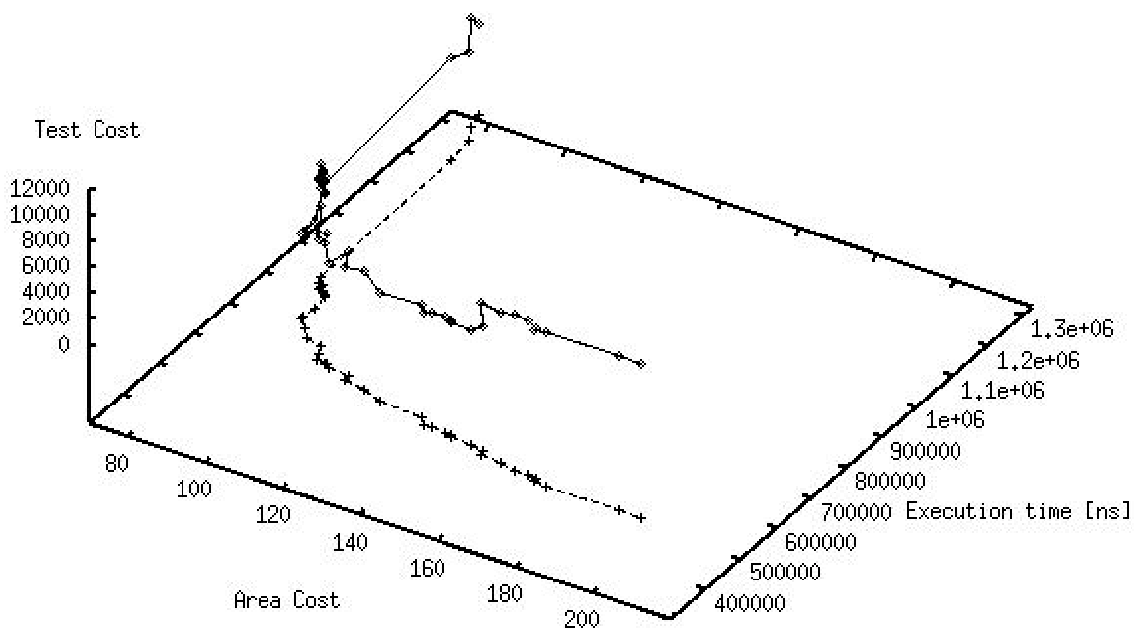


Figure 4.19. The 3D Pareto points for the "8-Queen" application

The standard Euclid norm $|\underline{u}|_E$, $\underline{u} \rightarrow u(a, t, f_l)$ with equal constraint weights has been used, i.e. no preferences have been given to the minimum test, area, and execution time in order to select the instantiation with minimal norm in the case of "Crypt" and "Compress". Both structures lead to a trivial situation with respect to the test scheduling – the number of busses in the design does not allow the simultaneous test of two components at the same time. Therefore, their test cost is equal to the sum of the individual test costs of the components used in the datapath. Figures 4.20 and 4.21 show the resulting instantiations after the complete CAD flow is executed in case of "Crypt" and "Compress" programs.

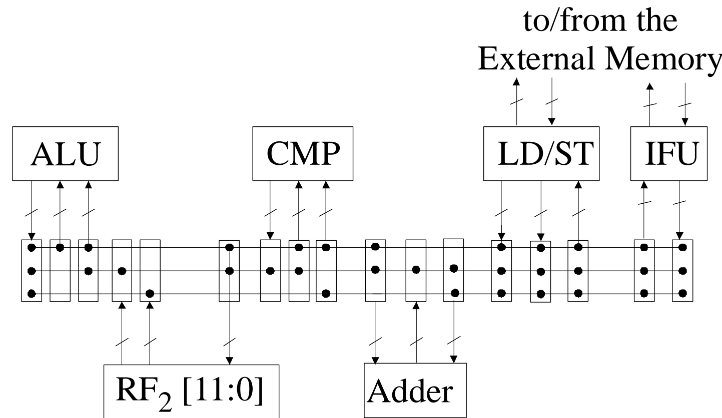


Figure 4.20. The instantiation with minimal circuit area, execution time and test cost (equal weighting factors) for the “Crypt” application

As Figure 4.20 shows, there are three buses in the interconnection network of the “Crypt” application, each having a width of 16 bits. The control busses are not shown. The ALU unit is capable of performing the operations of addition, subtraction, shifting and basic logical operations (AND, OR, XOR). CMP refers to the Compare unit capable of comparing the values of two input operands. The register file in the resulting design has one input and two outputs and has 12 ([11:0]) addressable locations. The Load/Store unit is fully connected to the network.

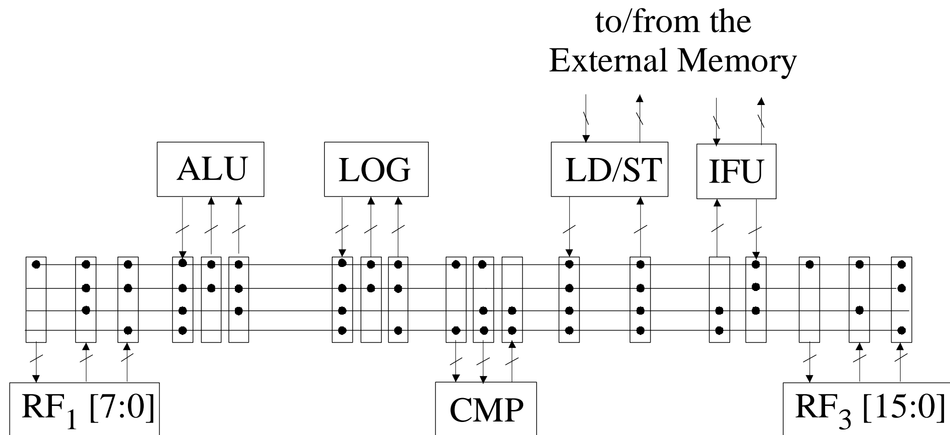


Figure 4.21. The instantiation with minimal circuit area, execution time, test cost (equal weighting factors) for the “Compress” application

However, note that in the case of the “Compress” application, this unit has one input and output port, causing a longer time for the functional-structural test application. The design itself is slightly more complex having 4 data busses (also 16 bits wide) and two register files with 8 and 16 locations, respectively. Of course, both Load/Store and Instruction Fetch Unit are connected to the external memory via the Memory Interface Unit (not shown in the Figures 4.20 and 4.21).

The “Eight-Queens” example has even more complex structure, as the application throughput got higher priority in the process of selection, as depicted in Figure 4.22.

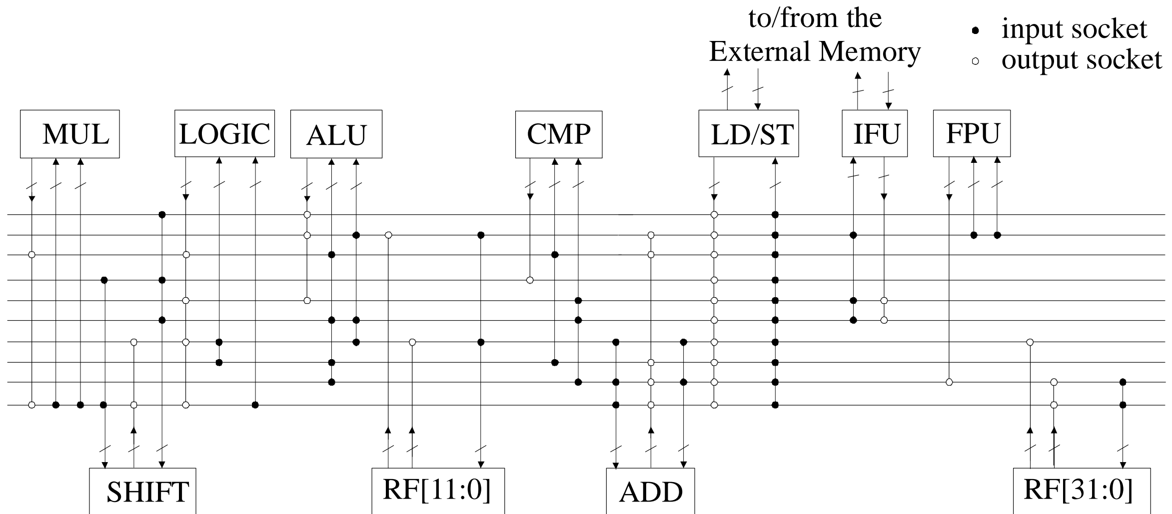


Figure 4.22. The resulting "Eight-Queens" instantiation

The application contains a number of components and busses, being a good example for the proposed approach of test scheduling. The connectivity optimization results in a set of instantiations on which the proposed test-scheduling algorithm is carried out. The test cost has been recalculated for each of the components and ordered according to their decreasing values. For example, for the instantiation given in Figure 4.22, the components are ordered according to the following relation:

$$f_{fpu} > f_{divide} > f_{reg2} > f_{cmp} > f_{alu} > f_{reg1} > f_{add} > f_{shift} > f_{logic} \qquad (4.18)$$

where reg1 and reg2 refer to the register files with depth of 12 and 32 registers, respectively. Having applied the test-scheduling algorithm shown in Figure 4.14, the following four scheduling lists are obtained:

List [1] = (fpu, divide, cmp, alu),
List [2] = (reg2),
List [3] = (reg1, add),
List [4] = (shift, logic).

Therefore, the total test cost of the components is equal to:

$$f_c = f_{fpu} + f_{reg2} + f_{reg1} + f_{shift} \; . \qquad (4.19)$$

Table IV depicts the test-cost function of the components for the three applications, as well as the fault coverage obtained using our functional-structural method. The fault coverage refers to the components without sockets.

Our approach does not take explicitly the Load/Store and Instruction Fetch unit into account during the test-cost calculation, i.e., it does not calculate their test cost. Both units must be present for any application and the design-space exploration does not cope with them. However, it does take their connectivity into account since the number of ports of the Load/Store and Instruction Fetch unit does influence the testing time. For example, the test cost of the register file in "Crypt" application is lower than the test cost of the register file in "Compress" application despite the fewer number of locations in the latter one. The

reason for this is that the Load/Store unit in “Compress” application has only one output port. Also, the link of the Load/Store unit with the memory interface unit has a large influence on the overall test time, but this will be the topic of the next chapter.

TABLE IV THE TEST RESULTS OF THE FUNCTIONAL-STRUCTURAL TEST APPLIED TO THREE TTA APPLICATIONS

Application / Component	Crypt	Compress	Eight-Queen	FC(%)
	Test Cost			
ALU	72	142	-	99.72
ADDER	10	-	-	100
COMPARE	75	150	-	100
SHIFTER	-	14	-	100
RF1[7:0]	-	142	-	100
RF2[11:0]	75	-	-	100
RF3[15:0]	-	1544	-	100
LIST1	-	-	1569	99.45
LIST2	-	-	1208	100
LIST3	-	-	610	100
LIST4	-	-	576	100

An advantage of our approach is that the functional test of the components can also assist during the delay fault tests, since the test basically checks not only the structure of the components but also their timing relations (4.3 - 4.10). In addition, our approach can be extended to any type of regular bus-oriented VLIW Application Specific Instruction Processor (ASIP) architectures. Figure 4.23 shows the general structure of such architectures [TRI98]. The TTA is a subset of these architectures.

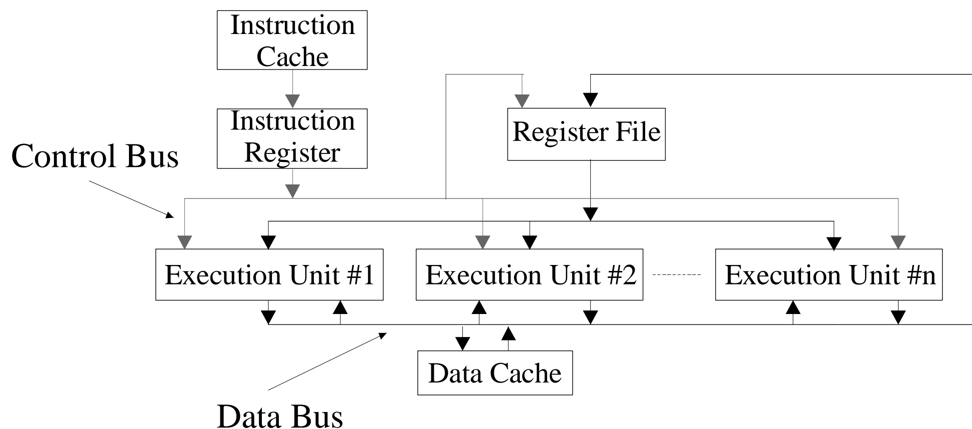


Figure 4.23. The bus-oriented VLIW ASIP template

Since most of the components are directly accessible from the bus, it is obvious that their test can be done by means of the functional application of structural test patterns. A few modifications are required however if the components are not directly accessible with regard to the memory ports. For example, Figure 4.23 shows the VLIW architecture where

the output of the register file is connected to the Data Cache via one or more functional units. In these situations, the order of testing of the components becomes also relevant. This will be explained in more detail in the next section that addresses the test-constraint insertion into another Hardware/Software codesign system, called "CASTLE".

4.7. Test-Cost Estimation in the CASTLE Codesign Flow

4.7.1. The CASTLE Codesign Environment

CASTLE is a co-synthesis environment developed at GMD SET [WIL95]. The CASTLE environment allows one to configure the various architectural solutions for a specific application domain. It copes with the architecture specification and co-synthesis of high-performance VLIW processors (an MPEG decoder in our case). The CASTLE design methodology is mainly based on the analysis of different VLIW processor instantiations from which the most suitable one, according to certain criteria, is selected for the final hardware implementation. CASTLE supports a partially automatic design flow where the designer has to take the main decisions with respect to the architecture structures. The time-consuming development processes are semi-automated. Before the basic design flow imposed by CASTLE is explained, the fundamentals of a VLIW processor framework will be discussed, (as the test cost estimation has that architecture as a target).

The generic structure of the VLIW processor together with a typical instruction word is given in Figure 4.24 [RAU93]. According to this Figure, the processor can be subdivided into two main parts: the control unit (sequencer) and the datapath. The control unit fetches one long instruction per cycle. One part of the long instruction word controls the fetch of the next instruction. This part includes the jump condition (*jumpcond*) and the jump address (*jumpaddr*). They define the flags that are used as the condition of the branch and the address displacement of a taken branch. When the branch is not taken, the next instruction is fetched. The *immediates* part controls the transport of constants from the control unit to the datapath.

The remaining part of the instruction word controls the datapath. Each component in the datapath has its own field in the instruction word, i.e. the instructions are completely orthogonal. Instructions from different units do not interfere with each other. The data is read from and written to the data cache located externally from the CASTLE framework and adapted to the VLIW architecture. The datapath is directly controlled by the instruction word. If one wants to increase the number of components in the datapath, this would be reflected as a new field addition in the instruction word corresponding to the newly added component. This is one of the advantages that makes a VLIW processor highly parametrisable and very well suited as a generic architecture template. It leaves the designer more degrees of freedom, i.e., the possibilities to assess various tradeoffs between throughput and area. The parameters that can be specified are:

- number and type of functional units such as adders (+), multipliers (*), shifters, etc., together with their properties such as e.g. latency of the component.
- registers, i.e., the size and the number of read/write ports in the register files
- number of ports; the datapath has different types of ports according to Figure 4.24: there is the port used to transfer the data from/to the data cache (*Datain1,*

Datain2), address port specifying the component location (*control),* and ports for communication with the control part (*control, flags*)

- interconnection network; usually, a restricted network connection will be used in the datapath. Hence, not all functional units are connected to each other, which saves chip area. A single driver is assumed for each connection, i.e., tri-state busses with multiple sources and destinations are not used. It simplifies the hardware synthesis, but the area overhead increases due to the higher number of busses and multiplexers used in the design.

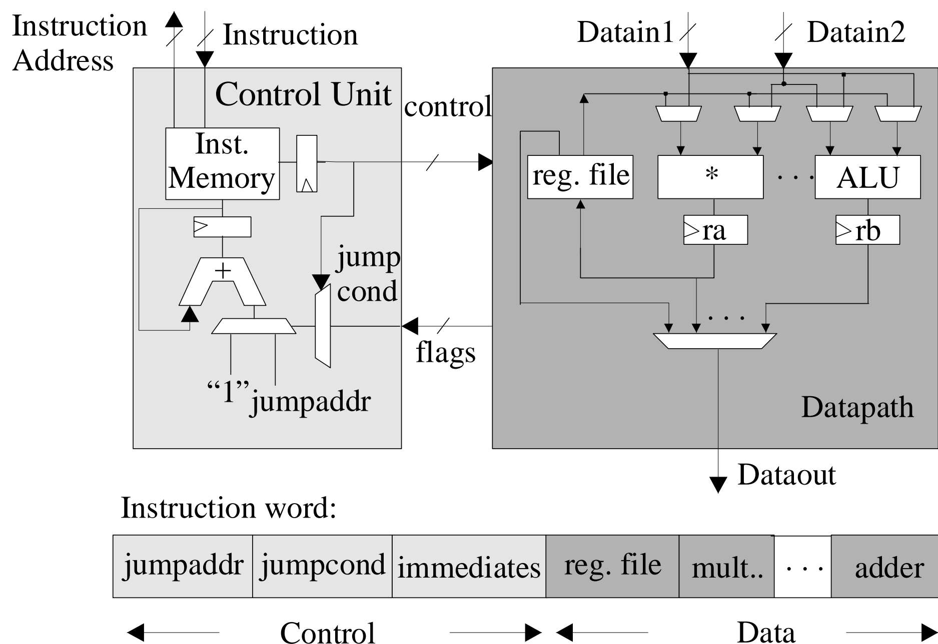


Figure 4.24. Structure of VLIW processor and typical instruction word

Figure 4.25 shows the codesign flow as used within the CASTLE tool. The design process starts with the development or reuse of an input application. That can be, for example, a video compression program based on the Discrete Cosine Transform (DCT) [RAO90] as used in standards such as Jpeg and Mpeg. The programs, usually in C or C++, are translated into a sequential final instruction-set assembler code using the MOVE/GNU compiler. The sequential assembler code is the input for the analysis and profiling of the application.

The quantitative analysis of assembler code is performed next. It consists of the requirement analysis followed with an iterative design-space exploration [CAM96]. The requirement analysis calculates the frequency of execution of basic components, which in turn determines the minimum number of functional units that given processor structure requires. It is also useful for the restriction of the design space. For instance, Figure 4.26 illustrates the result of the requirement analysis for various video-compression applications like Jpeg and Mpeg [WIL97]. The operation type is shown on the horizontal axe, while the vertical axe is the percentage of operation executions for a given application. This Figure shows that the addition operation is the most critical function in these applications of the VLIW processor. Therefore, the designer may favour to use a larger number of adders in order to improve the performance. That is the task of design-space exploration, to determine the throughput bound of the processor with respect to the

instantiation and properties of the compiler back-end. Based on these results, the designer can make decisions with respect to the number and type of components that he aims to use in the design.

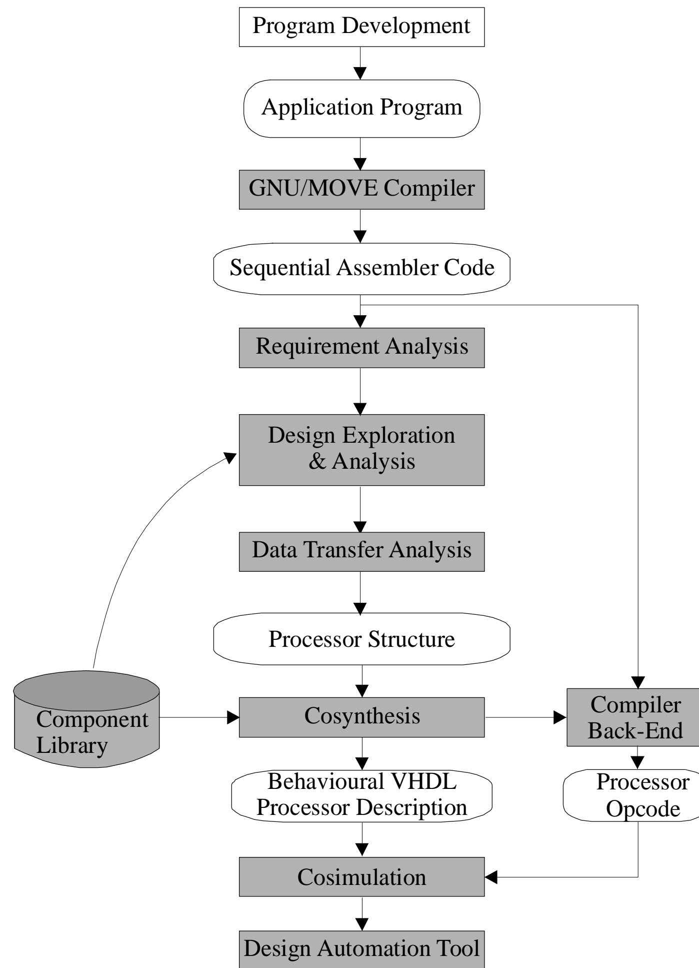


Figure 4.25. The CASTLE Codesign Flow

Figure 4.27 reflects the results where the number of cycles per unit is given with respect to the number of components used in the design. This calculation is carried out during the design-space exploration.

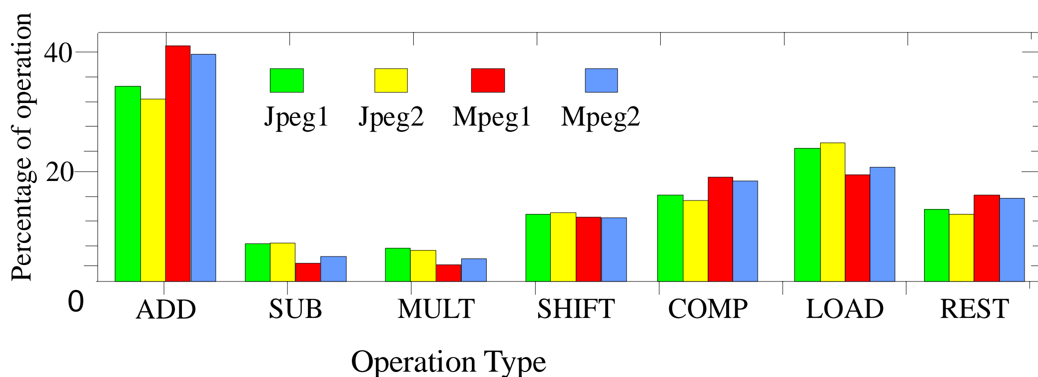


Figure 4.26. The operation execution frequency versus component types

Figure 4.27 also shows that an increase in the number of components will not contribute significantly to the throughput improvement after some limit. Therefore, the designer is capable to further restrict the range of components where performance changes are more significant. Hence, the next step will be the design exploration with a subset of, e.g., 5 adders, 3 shifters, 4 compare components, etc.

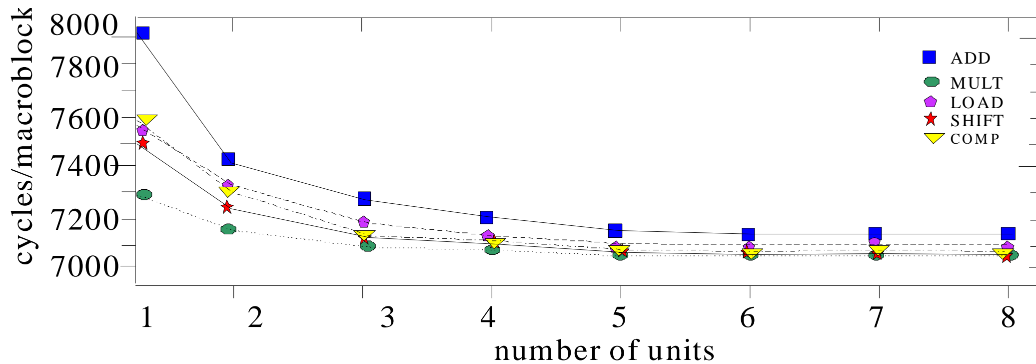


Figure 4.27. Results of the design-space exploration for various components

Next, the data-transfer analysis is performed. The analysis is carried out with respect to the number of busses and the connectivity of datapath components. For example, the most frequently used transfers between the components will be supported by direct connections between them. Otherwise, if there are not so many transfers between two components, they will be connected via a register file. However, much interconnection would occupy a large part of the chip area leading to large delays and hence, slowing down the processor speed. Therefore, the designer can make the tradeoff between the execution frequency of the components and an increase of the network size.

The results of the previous steps define the basic structure of the processor. The CASTLE cosynthesis environment will now generate a behavioral VHDL description of the components of the processor (using a VHDL library of components) and a corresponding compiler back-end with processor opcode (using the MOVE retargetable compiler). The next step should be the cosimulation of the generated hardware of the processor that runs the processor opcode in order to validate the design. After that, the final synthesis of the processor hardware should be feasible by commercial high-level design automation tools such as *Synopsys*.

The explained codesign approach aims at a systematic refinement of the design space spanned by hardware and software in a wide application range. The designer can influence the flow and adopt the solution to a particular application. However, the proposed codesign approach does not offer the possibility to assess the overall test cost of the final processor before the hardware implementation. The next subsection proposes a modified codesign approach in order to enable the designer to have an insight into the testability of the designed VLIW architecture.

4.7.2 The Test Approach in the CASTLE Flow

As already stated, the test strategy has to be considered and adopted early in the design phase if one considers complex designs such as VLIW processors. More particular, the datapath is the most critical part of the VLIW processor in terms of its performance. Therefore, the testability criterion is included in the quantitative analysis together with

speed/area constraints. The testability of a system is assessed using the fault coverage and test time, and hence, test vectors are required to test the VLIW processor. The test vectors target stuck-at faults that may appear in the hardware implementation of the datapath components. Hence, a test attribute is adjoined to each component, expressed as the number of test vectors that are necessary to test that unit with the highest possible fault coverage. All components of the datapath are described with behavioural VHDL code and placed in the library of components (see Figure 4.25), including the multiplexers that may be used to connect them to the buses. The test-pattern generation for various components is feasible after their synthesis using a high-level synthesis tool. In our case, the *Cadence Synergy* HDL tool was used to synthesize components. The same Philips in-house industrial tool applied during the core-based test explained in the previous chapter has been used as ATPG tool. The test-pattern generation results in a "canned" test set per component and hence the library of CASTLE is extended in a similar fashion as it has been done within the MOVE codesign package. The library now consists of different architectures per component, each with a different area, throughput and number of test vectors. Hence, the designer has the opportunity to analyse different implementations concerning the test constraint as well. The Table V represents the number of test vectors and fault coverage of some of the 16-bit datapath components that are used in the CASTLE library.

TABLE V THE NUMBER OF TEST PATTERNS FOR THE COMPONENTS IN THE CASTLE LIBRARY

	Adder	Multiplier	Compare	Shifter	ALU	Reg. File
# of vectors	7	55	70	6	84	112
FC(%)	100	100	100	100	99.67	100

The concept of C-testability has also been employed for some components (e.g., the Adder, Multiplier and Shifter in Table V are C-testable). The library contains 4, 8, 16, 32 and 64-bit implementations of each component. The register file has one input and two outputs, and its number of test patterns also depends on its depth, i.e., the number of registers. The register file in Table 4.5 has 16 locations.

Therefore, the choice of the designer of the architectural template with respect to the number and type of functional units during the design exploration phase is now also based on the test attribute. The test cost (*TC*) of the complete datapath is estimated after the co-synthesis and generation of the RTL description of the processor. If a proposed functional-structural test is accomplished, the test time will be directly proportional to the following parameters:

- the number of the test patterns
- the latency of the component under test
- the latency of components used to apply the test stimuli
- the latency of components used propagate the test response

Of course, the two latter latencies are applicable only in the situations when the component does not have the ports directly accessible from the processor primary inputs/outputs (Datain1, Datain2, Dataout). Hence, the test time is expressed as sum of the number of the test vectors per component:

$$TC = \sum_{i=1}^{n_c} \sum_{j=1}^{n_i} C(i,j) p_i \qquad (4.20)$$

where p_i represents the number of test vectors of *i-th* component, n_c is the number of types of different components used in the design, while coefficient *C(i,j)* denotes the number of cycles necessary to apply and read one test pattern of a *j-th* instantiation of the *i-th* component. This can be seen in Figure 4.24 where the primary inputs of an ALU are directly connected to the primary inputs of the processor. However, its output is connected to the input of register *rb*. Therefore, the propagation of the test response requires that the latency of the register rb is also taken into account. However, for some components, it may be required to search for the test path through other components in order to set the stimuli or read the response. That is the case with the register file in Figure 4.24. The inputs of the register file need to be set via the multiplier. Also, the test response appearing at one of the outputs of the register file needs to be propagated via the multiplier or ALU, thus implying a certain test order. The components that will be used to set or read the response of other components need to be tested first. In general, the coefficients C_{CUT} of the component under test (CUT) are determined according to the following formula:

$$C_{CUT} = \sum_{k=1}^{n_{set}+n_{prop}+1} L(seq) + L(CUT) + \sum_{l=1}^{n_{set}} L(set)_l + \sum_{q=1}^{n_{prop}} L(prop)_q \qquad (4.21)$$

where *L(seq)* denotes the latency of the control part (sequencer), *L(CUT)* is the latency of the component under test, while *L(set)* and *L(prop)* refer to the latencies of the components used to set the test stimuli or read the test response, respectively. The parameters n_{set} and n_{prop} indicate the total amount of these components. The sequencer has to set the control signals for each of the components that are active. Hence, the latency of sequencer has also to be taken $(n_{set} + n_{prop} + 1)$ times into account (*"+ 1"* term refers to the component under test).

Fortunately, all datapath components used in our library have the property of "neutral element" so they can easily propagate the signals. For example, forcing the "*0*" logical value at one input of an adder will result in propagating the second input to the output. However, it requires additional cycles to set the propagating elements to the desired state before the outputs of register files are propagated to the primary output via that element. The Figure 4.28 shows this situation. Here, the component under test is performing the operation *OP*. Suppose its latency *L(OP)* = 2. In addition, let *L(seq)* = 2, i.e., the number of timing slots required to bring the input *control* signals to the multiplexer. Obviously, there are not components that need to transport the test patterns, as the adder inputs are connected to the primary inputs of the VLIW processor, and hence $n_{set} = 0$ and *L(set)* = 0. The adder and register behind the outputs of the adder will propagate the test response (n_{prop} = 2). If both of them have the latency equal to 1, the coefficient C_{op} of the component under test equals to:

$$C(OP) = 3 \times L(seq) + L(OP) + L(set) + L(add) + L(reg) = 6 + 0 + 2 + 1 + 1 = 10 \qquad (4.22)$$

In general, the coefficients C_i do not necessarily have the same value for each of the instantiations of one type of component in the architecture, as they can be connected in different way with surrounding components. That is why the additional sum in equation

(4.20) is required where n_i refers to the number of instantiation of the *i-th* component type so that the term *C(i,j)* points to a unique component.

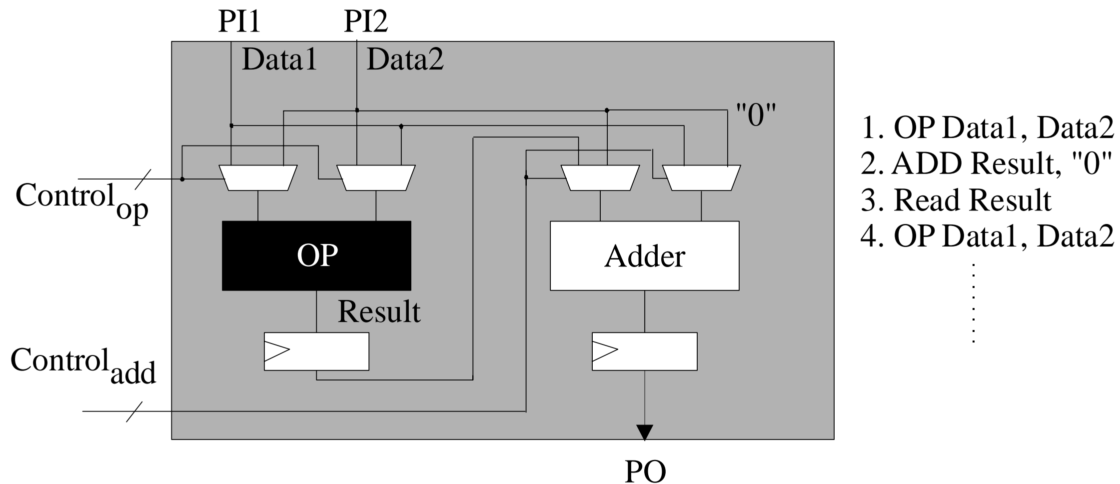


Figure 4.28. The test response propagation via datapath elements

Our approach is valid independent of the way the test pattern stimuli are applied to the primary inputs and the responses are read at the primary outputs. Hence, it may be implemented in a core-based environment that “by-default” supports any type of test. Also, there are no restrictions with regard to any type of BIST architecture.

The test cost in the case of the CASTLE codesign tool may be interpreted as the upper “bound” in the number of the test vectors. Certain components may be tested in parallel if test scheduling and test vector compression is applied. The fault coverage of the complete processor can be calculated after fault simulation of the overall processor. This is carried out after the synthesis and mapping of the functional units into a gate-level implementation.

The CASTLE codesign flow shown in Figure 4.25 needs a certain modification to include the test constraint, as is illustrated in Figure 4.29. Note that the Figure shows the modified part of the flow as well as the parts affected by that modification.

The library of the functional units is extended with the previously mentioned components and their adjoined attributes and canned test sets. After the design and connectivity exploration is performed, the designer selects the most appropriate hardware processor structure of the datapath based on the area/execution time/ criteria and test cost estimation. Namely, the designer has available number of test patterns of each component in this stage of the flow, so that he may estimate roughly the test cost for each of the instantiations. After cosynthesis and generation of the RTL structure, the values of the coefficients *Ci* are subsequently calculated, based on the datapath structure and the formula (4.21). Next, the test cost is calculated according to the formula (4.20) and if the designer is still satisfied with the outcome, the flow proceeds in a standard way, i.e., as in any other HDL-based design. If not, he has to repeat the actions concerning the selection of the hardware structure within the design-space exploration. Of course, the fault simulation with the test patterns at the gate-level is necessary to determine the fault coverage and verify the approach.

After the design of a testable datapath part of the VLIW processor, one has to define the test floorplan for the rest of the processor together with the (external) data memory. The

next section illustrates the proposed strategy for the test-cost estimation with respect to various examples of video-compression applications.

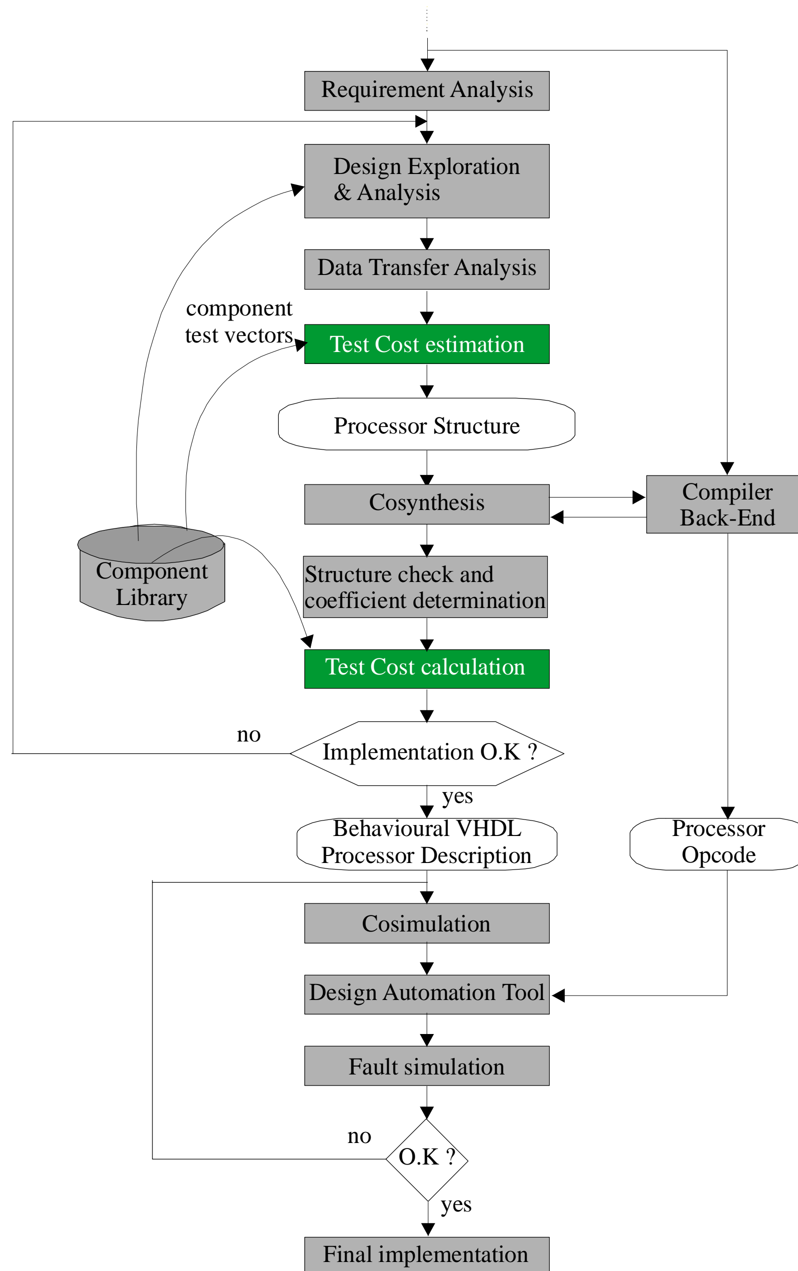


Figure 4.29. Modified codesign flow in terms of testability as additional criterion

4.7.3 Test Floorplan and Results of a Video-Processor

Figure 4.30 shows one possible implementation of the test floorplan of a complete VLIW processor without an external memory and peripheral devices capable to execute the already mentioned video-compression algorithms, Jpeg1, Jpeg2, Mpeg1 and Mpeg2 [WIL97]. The testing approach is derived according to the already proposed functional-structural test style as applied in the MOVE codesign flow.

Similarly to what has been done in MOVE, it is necessary to set the control signals at the interface between the sequencer and the datapath into the predefined statements, in order to select the component for test. The control signals are not shown for the sake of figure readability. The test response is captured at the primary outputs of the datapath. Certain components such as the Compare unit (CMP in Figure 4.30) produce also the control flags at their outputs. They require DfT insertion of observability points, as Figure 4.30 illustrates. However, the insertion has been done outside the datapath and outside the critical path of the overall processor.

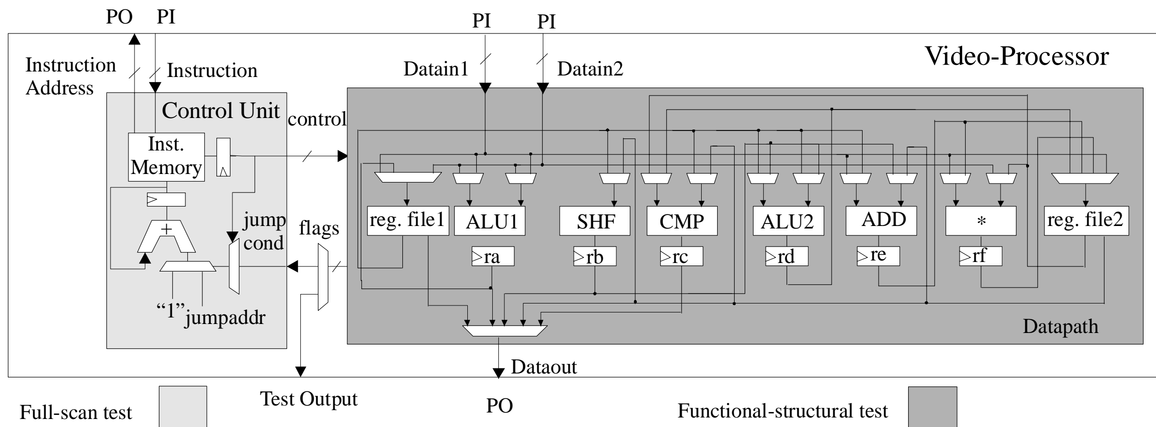


Figure 4.30. Test floorplan of the embedded video-compression system

The control block containing sequencer with embedded instruction cache requires adifferent test approach. The instruction cache is tested with the 6N plus marching test algorithm [SAC97] that detects most of the academically adopted memory faults [GOO91] plus open-defects in the memory address decoder. However, these test patterns must be combined with the test patterns for the sequencer. The sequencer consists of random logic (see Figure 4.30). It is tested by means of the full-scan technique. In order to obtain test patterns for this core, we have used again the Philips in-house CAT tool.

The test patterns for the sequencer and marching test for the instruction cache are then expanded. Of course, only the test protocols are converted, i.e. the instructions what to do with patterns and not the actual patterns. Note that the test ordering of the two blocks is also important, as the sequencer needs to be tested before the datapath.

Table VI shows the number of test-vectors expressed as test cost (*TC*) together with the fault coverage (FC) of various datapath hardware-structures.

TABLE VI THE TEST RESULTS OF DIFFERENT IMPLEMENTATIONS OF VIDEO-PROCESSOR

Adder	Shifter	Multiplier	ALU1	ALU2	Comp	RF	TC	FC (%)
Number of units								
1	1	1	1	1	1	2	1245	98.7
2	1	1	1	1	1	3	2874	98.3
3	1	1	2	1	1	1	1990	98.1
2	1	1	1	1	1	2	1224	98.9
2	2	1	2	1	1	3	3461	98.5

Hence, the designer has clear insight with respect to the number of test vectors required to test various implementations and fault coverage achieved with those test patterns. A certain fault coverage drop appeared due to the fixed states in the multiplexers at the inputs of the components and enable signals of the registers (ra, rb, rc, rd, re and rf). The fault coverage is sufficiently high to justify the use of our approach, bearing in mind that the DfT has not been introduced within the datapath in any form.

4.8. Conclusion

An original method of the test constraint insertion into existing Hardware/Software codesign systems has been elaborated in this chapter. The testability constraint is added during the high-level phase within the MOVE and CASTLE codesign packages. The MOVE codesign package generates the so-called Transport-Triggered Architectures starting from the *C*-application. Our approach that assesses the instantiation of the architecture with respect to the area, throughput and test has been explained. It is based on a functional test with structurally generated test patterns. The method does not degrade already achieved area/execution-time ratio obtained after the high-level design space exploration. The fact that allows one to minimize the extra circuitry is the regularity of the MOVE-TTA structure, i.e. the direct accessibility of the components in the datapath via busses. The test cost, for the sake of the analytical calculation of the architectural testability suitable for TTA structures, has been established. The analytical test cost functions for the functional test of the components are derived according to the transport-timing relations and they are dependent on the architectural parameters only. A heuristic test-scheduling algorithm to further optimize the test time has also been explained. Our approach has been captured in the procedures written in *C* programming language and integrated within the MOVE codesign package using *Gawk* script files. The method is validated with respect to the example of "Crypt", “Compress” and “Eight-Queens” applications. The examples has also pointed out that the test costs may vary quite significantly, even for the architectures that have similar characteristics with respect to the area and throughput. That is another hint in the favor of applying the high-level test synthesis in this environment, before the actual hardware implementation.

A similar test-constraint has been added into the CASTLE codesign tool during the design space exploration, enabling the designer to estimate different instantiations of the application. The test method is arbitrary and is validated using the slightly modified functional-structural test in a video-processor example.

In general the MOVE template has better overall testability characteristics than CASTLE due to the regularity of its datapath. Each component of the datapath is accessible from the interconnection network, while this was not the case with CASTLE where some of the datapath components were accessible only through the other components, thus implying certain test order.

In both cases, the proposed method does not introduce any DfT circuitry within the datapath. In addition, the method can achieve a high-fault coverage of the components used in the datapath. Furthermore, the method has proven to be highly efficient with respect to the number of test vectors that need to be applied in order to test the components. The next chapter will illustrate the implementation of functional-structural test at the gate-level of the TTA processor.

References:

[ABR91] M. Abramovici, M. A. Breuer, A. D. Friedman, *Digital Systems Testing and Testable Design, Computer Science Press*, New York, 1991.

[BEA00] M. Beardo, F. Bruschi, F. Ferrandi, D. Sciuto, “An Approach to Functional Testing of VLIW Architectures,” *Proc. Of the International High-Level Design Validation and Test Workshop (HLDVT)*, November 2000, Berkeley, CA, pp. 29-33.

[BRA80] R. Brayton, R. Spence, *Sensitivity and Optimisation*, Elsevier 1980.

[BOT81] P. S. Bottorf, “Functional Testing Folklore and Fact,” Digest of Papers 1981, International Test Conference, October 1981, pp. 463-464.

[BUC94] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, “Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems,” *International Journal of Computer Simulation*, April 1994, pp. 155-182.

[CAM96] R. Camposano, J. Wilberg, “Embedded System Design,” *Journal of Design Automation for Embedded Systems*,” 1(1/2), 1996, pp. 5-50.

[CHE93] C. H. Chen, D. G. Saab, "A Novel Behavioral Testability Measure," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 12, December 1993, pp. 1960 - 1970.

[COR94] H. Corporaal, “MOVE32INT, Architecture and Programmer’s Reference Manual,” Faculty of Electrical Engineering, Delft University of Technology, Computer Architecture Department, April 1994.

[COR95] H. Corporaal, *Transport Triggered Architectures; Design and Evaluation*. Ph.D. thesis, Delft University of Technology, September 1995.

[COR98a] H. Corporaal, “*Microprocessor Architectures from VLIW to TTA,*” ISBN 0-471-97157-X, John Wiley, 1998.

[COR98b] H. Corporaal, M. Arnold, “Using Transport-Triggered Architectures for Embedded Processor Design,” *Integrated Computer-Aided Engineering,* vol. 1998, no. 1, 1998, ISSN: 1069 – 2509, pp. 19-37.

[FRI73] A. D. Friedman, “Easily Testable Iterative Systems”, *IEEE Transactions on Computers*, Vol. C-22, No. 12, December 1973, pp. 1061-1064.

[GAJ94] D. D. Gajski, L. Ramachandran, "Introduction to High-Level Synthesis," *IEEE Design & Test of Computers*, winter 1994, pp. 44-54.

[GIZ96] D. Gizopoulos, A. Paschalis, Y. Zorian, “An Effective BIST Scheme for Datapaths,” *Proceedings of the International Test Conference*, October 20-25, 1996, Washington DC, USA, pp. 76-85.

[GOO91] A. J. van de Goor, *Testing of the Semiconductor Memories*, Kluwer Publishers, 1991.

[HOE99] M. R. Hoek, “DfT and BIST Techniques for the MOVE Architecture,” M.Sc Assignment, Faculty of Electrical Engineering, Delft University of Technology, 1999.

[HOO96] J. Hoogerbrugge, H. Corporaal, “Cosynthesis with the MOVE Framework,” CESA’96, IMACS Multiconference, Lille, France, 1996.

[JAN95] J. Janssen, H. Corporaal, “Partitioned Register Files for TTAs,” *Proc. of the 28th Annual International Workshop on Microprogramming* (Ann Arbor, Michigan, November 1995).

[LeT96] Y. Le Traon, G. Al-Hayek, C. Robach, "Testability-oriented Hardware/Software partitioning," *Proceedings of the International Test Conference,* October 20-25, 1996, Washington DC, USA, pp. 727-731.

[MIL91] G. Milovanovic, *Numerical Analysis, part I*, Scientific Book Edition, 1991.

[PAT87] W. Patterson, "UNIX password and DES encryption", ref. *Mathematical Cryptology for Computer Scientists and Mathematicians*, J. Willey & Sons, 1987.

[RAO90] K. R. Rao, *Discrete Cosine Transform*, Academic Press, New York, 1990.

[RAU93] B. R. Rau, J. A. Fisher, "Instruction-Level Parallel Processing; History, Overview and Perspective," *Journal of Supercomputing*, vol. 7, no. 2, May 1993, pp. 9-50.

[RIE92] R. P. van Riessen, *Module Generation for Self-Testing Integrated Systems*, Ph.D. thesis, University of Twente, January 1992.

[SAC97] M. Sachdev, "Open Defects in CMOS RAM Address Decoders," *IEEE Design & Test of Computers*, June 1997.

[SHA96] R. D. Shawn, J. P. Hayes, "Design of a fast, Easily Testable ALU," *Proc. of the 14th IEEE VLSI Test Symposium*, 1996, pp. 9-16.

[SHE84] J. P. Shen, F. J. Ferguson, "The Design of Easily Testable VLSI Array Multipliers," IEEE Transactions on Computers, Vol. C-33, No. 6, June 1984, pp. 428-431.

[STO91] L. Stok, "*Architectural Synthesis and Optimisation of Digital Systems*," Ph.D. thesis, Eindhoven University of Technology, April 1991.

[STO94] L. Stok, "Data Path Synthesis", *Integration*, *the VLSI Journal*, 1994, pp. 1 - 71.

[SYN01] http://www.synopsys.com/products/cocentric_studio/cocentric_studio_ds.html

[TRI98] http://www.semiconductors.com/trimedia/news/index.html#articles, "Tri Media Update"; Volume 1, Issue 1, 1st Quarter 1998.

[VAR00] Various authors, "The MPG manual", the documentation of Laboratory of Computer Architecture and Digital Techniques (CARDIT), Faculty of Electrical Engineering, Delft University of Technology, 2000.

[WAL95] R. A. Walker, S. Chaudhuri, "Introduction to the Scheduling Problem," *IEEE Design & Test of Computers*, summer 1995, pp. 60-69.

[WIL95] J. Wilberg, R. Camposano, M. Langevin, P. Plöger, T. Vierhaus "Cosynthesis in CASTLE," *Novel Approaches in Logic and Architecture Synthesis*, G. Saucier and A. Mignote editors, pp. 355-366, Chapman & Hall, London 1995.

[WIL97] J. Wilberg, *Codesign for Real-Time Video Appliccations*, Kluwer Academic Publishers, Dordrecht, the Netherlands, ISBN 0-7923-8006-1, 1997.

[WIN89] R. H. Winston, B. K. P. Horn, "*LISP, 3rd edition*," Addison-Wesley *ISSBN* 08329, 1989, pp. 289-290.

[ZIV00a] V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff, "Design and Test Space Exploration in Transport-Triggered Architectures," *Proc. of the Design, Automation and Test in Europe Conference (DATE)*, Le Palais des Congres, Paris, 27 – 30 March 2000, pp. 146 – 151.

[ZIV00b] V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff, "Computer-Aided Test Flow in Core-Based Design," *Elsevier Journal of Microelectronics*, vol. 31, issue 11/12, November-December 2000, ISSN 0026-2692/00, pp. 999-1008.

[ZIV00c] V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff, “The Test-Cycle Minimisation in Parameterized Bus-Oriented Datapath Design,” *Proc. of the 1st IEEE Latin American Test Workshop (LATW00)*, Rio de Janeiro, Brazil, 13-15 March 2000, pp. 215-220.

CHAPTER 5

Computer-Aided Test Implementation in the MOVE HW/SW Codesign System

5.1. Introduction

This chapter discusses the Computer-Aided test strategy in the MOVE hardware/software codesign system. The gate-level test implementation is derived by referring to the proposed test constraint that is introduced during the early phase of the codesign. This has been explained in the previous chapter. The test approach of the processor itself has been carried out in a structural top-down fashion by full exploitation of the processor's structure using a divide-and-conquer strategy. The method is based on the functional-structural test style along with full-scan. The second section of this chapter outlines the structural view of the VLIW TTA. The starting point for the gate-level implementation is VHDL source described at the Register-Transfer Level (RTL). The steps in order to convert the behavioral VHDL code towards an efficient synthesis using the *Synopsys* Design Compiler (DC) will be explained. The third section copes with the

test pattern generation of the VLIW TTA. It addresses the actual application of the functional-structural test patterns to the components of the datapath and interconnection network. They are tested with minimal extra DfT circuitry for the sake of preserving the area/execution-time ratio achieved during the earlier phase of design. In addition Design for Testability (DfT) inside the control block of the processor is explained being tested using the full-scan technique and memory test. The test-time analysis is carried out in section four. Our approach, consisting of the full-scan and memory test for the control block together with the functional-structural test of the datapath versus the conventional full-scan test applied to the whole processor structure, is compared in terms of speed. This analysis is performed by taking the design parameters into account. In section five, the results and discussion are treated: the proposed approach to test the VLIW TTA is illustrated by several examples and compared with the full-scan technique. It is shown that our approach achieves better results than full-scan in most of the categories that describe the overall quality of test. Next, the complete CAD/CAT flow that one has to use within the MOVE codesign package in order to build the efficient set of the test patterns that tests the VLIW TTA is given. Finally, section six summarizes the chapter.

5.2. Implementation Details and Synthesis of the VLIW-TTA

The test constraint introduced during the high-level phase of the codesign enables the designer to perform the test-quality assessment along with the area and throughput parameters and to select the particular instantiation for further hardware implementation. The resulting instantiation is given at architectural level of abstraction e.g., see Figure 4.20. This is subsequently translated to the RTL using the MOVE Processor Generator (MPG) [VAR00], the back-end tool of the MOVE codesign kit.

Figure 5.1 shows the structural block diagram of the complete TTA processor. The structure itself consists of the following three main parts:

- Datapath
- Interconnection network
- Control Unit

As already explained, the datapath is comprised of the functional units and the register files. The functional units of the datapath may be as complex as a Floating Point Unit (FPU) or an ALU or as simple as a full-adder. Their design is left free to the developer of the application as long as they obey the interface standard specified by the sockets. Therefore, the functional units may have an arbitrary number of input and output ports with both data and control properties and an arbitrary number of pipeline stages. The register files are organized as one or more registers with multiple input/output ports. The same statement holds for the register file design within the MOVE architecture as it holds for the functional units. Hence, one is free to design and tune the register files while taking into account the boundary conditions imposed by the socket circuitries. The current version of the MOVE codesign package uses two different implementations of register files that are described in the MOVE library of the components. Their implementations are based on the pipeline model described in e.g. [COR98].

The interconnection network consists of the input and output sockets and the buses. Their design will be described in more details in the sequel of the chapter.

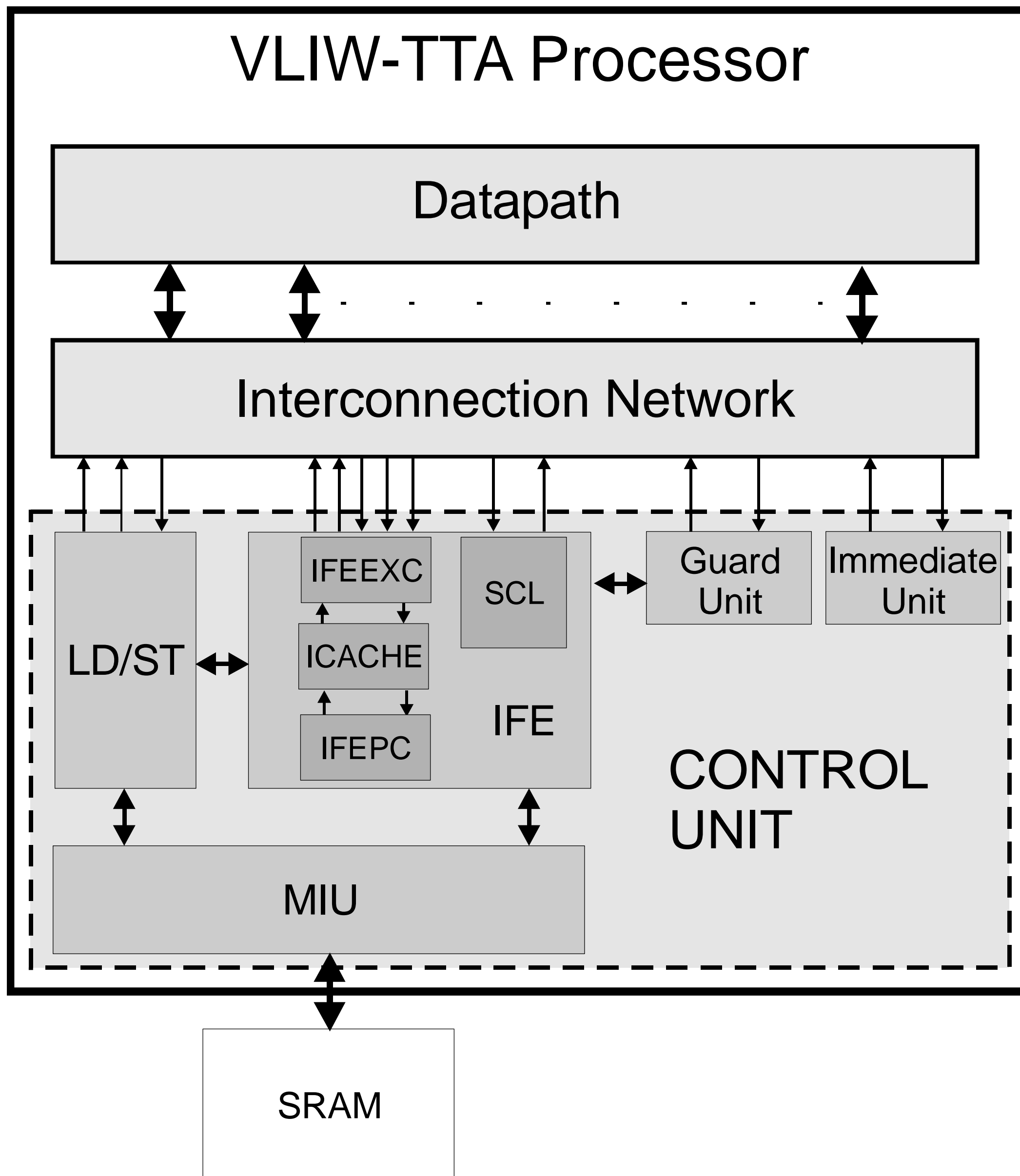


Figure 5.1. The structural view of the TTA

The control part incorporates the Instruction Fetch Unit (*IFE*), Load/Store Unit (*LD/ST*), Memory Interface Unit (MIU), Immediate Unit (*Imm*) and Guard Unit. The IFE itself contains an embedded instruction cache (*ICACHE*), program counter (*IFEPC*), exception handler block (*IFEEXC*), and the state control unit (*SCL*). The complete unit is responsible for control of the program flow (sequential, jump and call operation) and control of the interrupts. The LD/ST unit takes care for the data transport between the MIU and the interconnection network using the hybrid-pipelining mechanism (see previous chapter). The MIU connects the IFE unit and the LD/ST unit to the external memory where an SRAM is assumed. The SRAM is located externally and it does not

belong to the structure of the TTA processor. If another type of memory is to be used, then an appropriate modification of the MIU is required. The *Immediate* unit connects the data to the buses i.e., its result socket is connected directly to the data bus. The role of the *Immediate* unit is to transfer the constant values to the functional units or register files. Finally, the Guard unit is just a register file of the Boolean type with the difference that some registers cannot be written since they have fixed values or values set from outside. The guard unit is responsible for conditional execution of instructions. The control part blocks are connected to the interconnection network via the input/output sockets in a similar fashion as the components from the datapath.

The MPG tool employs the so-called *CHD* language, a combination of *C* and *VHDL* to translate the architectural-level modules of the processor to behavioral-level *VHDL*. The MPG also allows the designer to write the behavioral-level VHDL code of the datapath components externally, while MPG places the sockets at their connectors, according to Figure 4.5. The creation of the control-block components is done automatically within MPG and the designer has no influence on the creation of their internal structure apart from the definition of certain parameters as will be explained later. The designer has full control over the interconnection network by specifying the connections of the functional units and control blocks to the busses.

Nevertheless, the resulting behavioral *VHDL* code of the application is a large and cumbersome file (named *Top_*<application>*.vhd*) with all modules flattened and described at the same level of hierarchy. Testing of such a design described in the previous text in a structural top-down fashion is not feasible. Moreover, the resulting file itself turned out not to be synthetisable in that form. Hence, additional steps to refine the files and make them acceptable for the *Synopsys* VHDL-analyzer had to be carried out prior to the synthesis.

A hierarchical form of the *Top_<application>.vhd* file according to the processor structure has been created to make such a complex design easier to manipulate. A script file written in Gawk has been used for that purpose. Figure 5.2 illustrates the resulting arrangement.

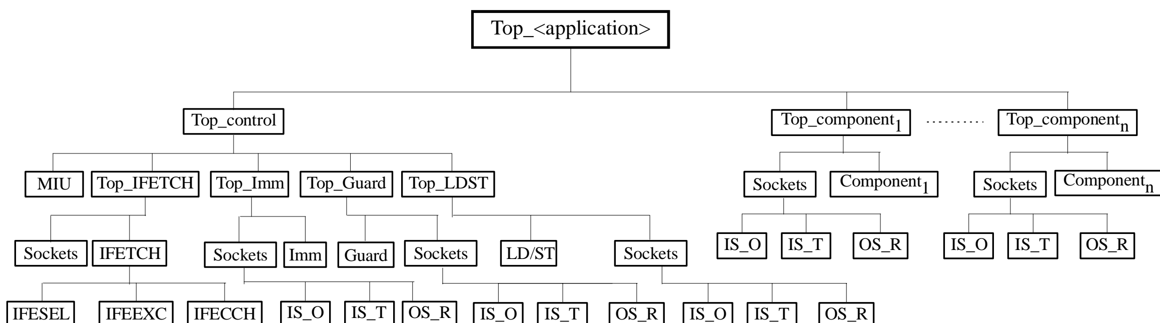


Figure 5.2. Design Hierarchy

The *Top_<application>* architecture contains the references to all subdesigns used in the application together with their mutual interconnections at the global level. All *Sockets* entities consists of *IS_O, IS_T* and *OS_R* files denoting the input-operand, input-trigger and output-result socket architectures, respectively (see Figure 4.5). The embedded instruction memory within the *IFE* unit has been defined as skipped block. That step was necessary in order to avoid loading of that part of the VHDL code by the *Synopsys* Design

Compiler (DC) and the creation of the embedded memory as a set of the flip-flops. Instead, the corresponding embedded memory described at gate-level has been inserted after the synthesis of IFE. We have chosen DRAM serving as cache for that purpose. The memory wrapper block is placed around the cache and contains the modules to allow the communication between the cache and the remaining part of the control block. Also, the DfT circuitry, to enable the test access to the memory, is inserted inside this block.

In addition, the following adjustments have been carried out in order to enable a straightforward synthesis:

- Elimination of HDL simulation models, i.e., explicit delay models.
- Elimination of VHDL configuration statements that the *Synopsys* DC does not support during the synthesis.
- Elimination of latches. The latches are absolutely unnecessary in a synchronous design such as the VLIW-TTA processor. Moreover, the latches decrease the overall fault coverage in a clocked synchronous design. Therefore, they have been removed accompanied with a proper modification in coding.

Currently, these modifications have been carried out manually. However, with the proper scripts, this task can be automated, too.

Before the synthesis, it is necessary to set the design constraints. It includes definition of the environmental attributes, maximum area and timing goals, such as maximum or minimum delay and clock specifications. Furthermore, the delays at input and output ports need to be constrained as well. This is the so-called *time budgeting,* which is done because the designer does not know a priori the delays at the inputs (coming from preceding cores) or the setup requirements (of proceeding cores) at the outputs. Table I gives an example of the constraints that have to be set prior to the synthesis. The synthesis itself has been performed using *Synopsys* dc-shell scripts to set the design constraints and perform its compilation, i.e., translation, optimisation and mapping into the gate-level components.

TABLE I EXAMPLE OF THE DESIGN CONSTRAINTS

Voltage and Temperature Variation	5.0±0.5V, 0 – 70°C
Process Variation	Slow (worst-case), Fast (best-case)
Wire Load Model	"Tc6a120m2"
Default Register Driving Input Port	Cell fde1a1, pin "Q"
Maximum Input Delay	2.5 ns
Load on all Output Ports	Cell buf1a4, pin "Z"
Maximum Output Delay	1.5 ns
Area Goal	Not specified
Clock Period	10 ns
Maximum Clock Skew	0.3 ns

Figure 5.3 depicts the overall design as a black-box showing the primary input/output ports that are initially available for the test-engineer. Looking back at Figure 5.1, it becomes obvious that the processor communicates with the external world via the Memory Interface Unit (MIU). These are *Address, Data, CS* (Chip Select)*, WE* (Write Enable) and *OE* (Output Enable) ports. *Clk* (Clock) and *Reset* are, of course, global

signals going to each module while *int* denotes an interrupt signal and is applied to the IFE unit. The port *restart_seq* starts the program's execution.

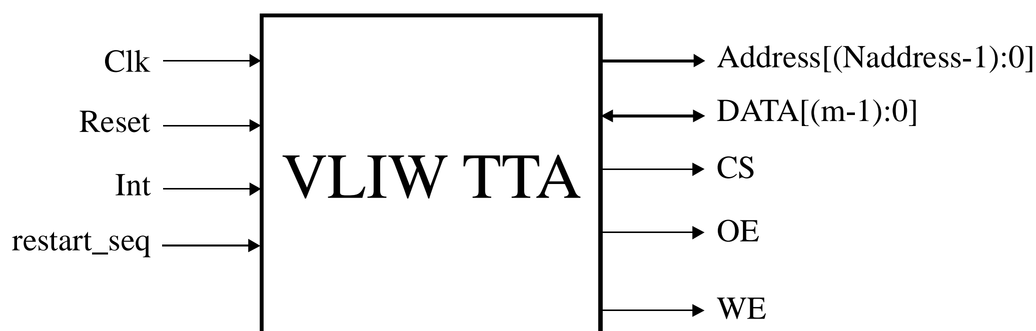


Figure 5.3. The ports of a VLIW TTA processor

Therefore, most of the primary inputs/outputs of the processor are the ports of the MIU while all other modules are embedded in the design, i.e., their ports are not directly accessible from the outside. However, applying the core-test strategy itself to test the embedded modules would be an overkill. From the practical point of view, the modules of the design are not sufficiently large to be considered as cores and the insertion of the test-shell as a wrapper around each module will unnecessarily increase the area and deteriorate the throughput. Rather, the complete VLIW TTA processor will be considered as a single core requiring its test-pattern generation strategy to be developed. Nevertheless, certain ideas from the core-based test strategy explained in chapter 3 may also take place during the generation of the test patterns, as will be explained in the sequel of the chapter.

5.3. Test-Pattern Generation

There are several conventional ways how to test this design such as full-scan, partial-scan, functional test, BIST. This chapter proposes a less conventional, original test-pattern generation strategy at the gate-level. The method is consistent with the test constraint introduced at the high-level of the design. It has to feature that a test flow is valid for each TTA instantiation of an arbitrary application realized with the TTA. The objective of the test strategy is to provide an efficient set of gate-level test patterns targeting the stuck-at faults occurred during the fabrication of the processor chip. Another objective of the test approach is that the modifications of the design and/or implementation of the DfT have to be minimized according to the requirement of the test synthesis in order that the area/throughput ratio remains intact.

The test approach of the TTA processor has been carried out in a structural top-down fashion using a divide-and-conquer strategy. Hence, the blocks of the processor are divided and placed in different groups determined upon their type and position in the processor structure and the test style they require (Figure 5.2). It has already been explained that the datapath has a regular structure while the datapath components are directly accessible from the interconnection network. Hence, the functional-structural test style whose fundamentals have been explained in the previous chapter will be employed to test the datapath components. However, the control-block is comprised of random

logic, sequential elements and also includes the embedded memory (DRAM). Hence, full-scan in combination with memory test seems the most logical choice for testing this part of the design.

As a result, the processor blocks are divided into three main groups according to the following test styles that are employed for their test:

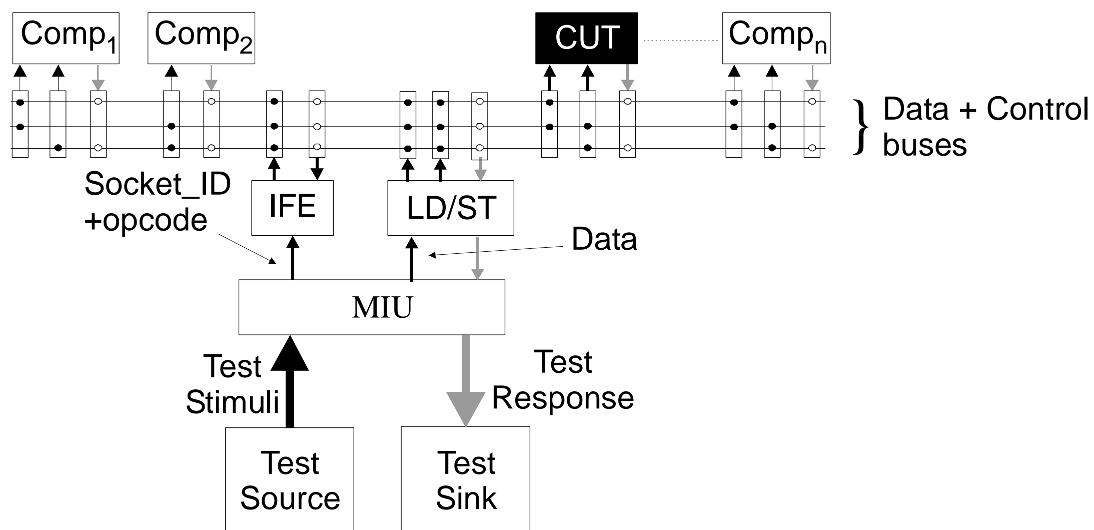
- Full-Scan
- Memory Test
- Functional-structural Test

The following subsections cope with more details of the application of these tests with a special attention paid to the application of the functional structural test.

5.3.1. Application of the functional-structural test

This style of test is the product of the test constraint introduced during the high-level phase of the design of the architecture. The pipelined execution of the operations within the datapath components may be used for test purposes as well, i.e. to schedule the subsequent test patterns of the functional unit/register file into the first possible consecutive timing slot [ZIV00]. The degree of pipelining depends on the bandwidth of the LD/ST and MIU unit, i.e. of the number of simultaneous transports that trigger the *load* or *store* operation, as already explained in the previous chapter.

The basic principle of functional-structural test in the MOVE architecture is illustrated in Figure 5.4. The test path through the sockets assumes the test-patterns consisting of two parts: a data part to the components under test (CUT) and applied via the data busses, and Socket ID's together with their opcodes being applied through the control busses. The patterns are provided externally and brought to the CUT via the MIU, IFE (socket ID's and possible opcode of an operation) and LD/ST (data) units. The test responses are read via the LD/ST and MIU, as Figure 5.4 illustrates.



5.4. Datapath component under test (CUT) using the functional-structural test style

Real-time stimulus application starts in the *test-source*, while real-time test-response evaluation is carried out by the *test-sink*. Both test-source and test-sink can either be implemented off-chip (Automatic Test Equipment – ATE) or on-chip (Built-In Self Test – BIST) or a combination of both. Figure 5.4 shows the so-called "sequential" application of a functional-structural test because it tests only one component at the time. Anyway, the configuration of the three data buses inside the interconnection network does not allow the testing of multiple CUTs in parallel. Of course, in the case of more busses, the test-scheduling algorithm described in the previous chapter may be applied resulting in the testing of more than one CUT at the same time. However, whether the parallel test will be possible also depends on the data bandwidth of the MIU and LD/ST units. For example, the LD/ST unit shown in Figure 5.4 can perform two load operations, and one store operation simultaneously. If more load/store operations are required, additional hardware is required to be included in the design of the LD/ST and IFE unit. This, in addition, will also be reflected in the data and control bit-width as well as on their overall design. However, that is a matter that has already been resolved during the design-space exploration when the particular instantiation with determined parameters has been selected for the implementation.

The test path through the components will now be clarified in more detail.

5.3.2. Test path through the control unit

Figure 5.5 depicts the test data flow (bold lines) from the Memory Interface Unit (MIU) towards the Instruction Fetch or LD/ST Unit during the functional-structural test.

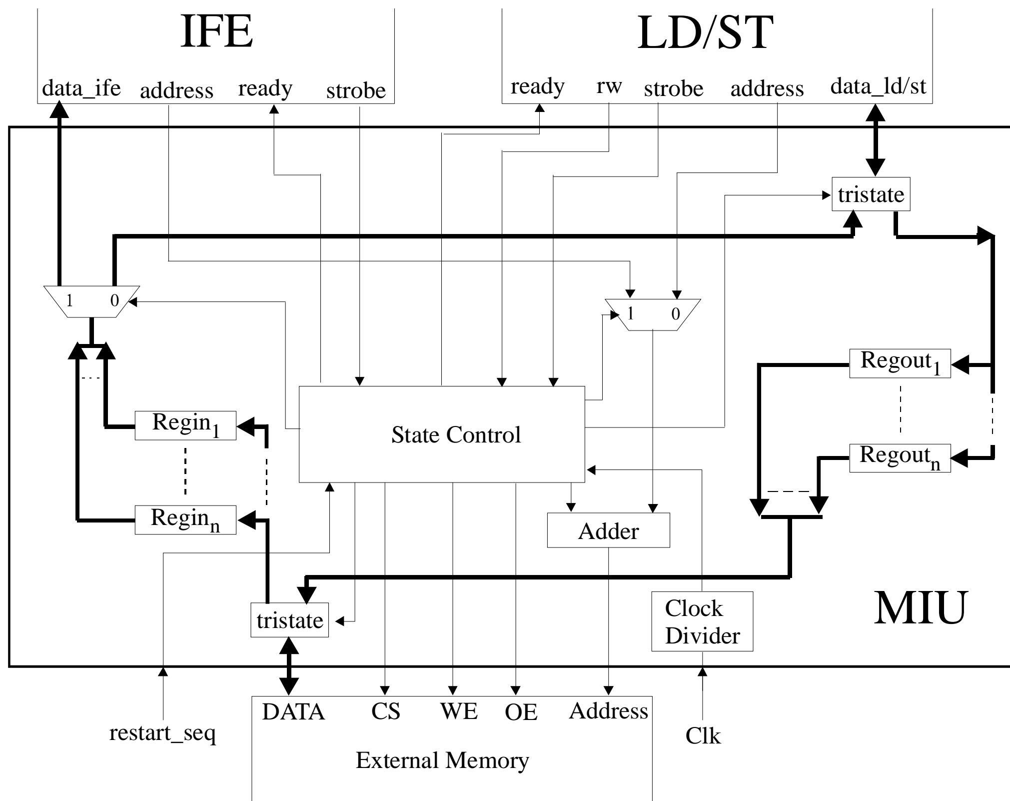


Figure 5.5. The test path in the memory interface unit

Basically, the data input (*DATA*) is used to transport the VLIW instructions and the data via two sets of registers (Regin and Regout). These registers are automatically generated upon the specification of the external memory and their task is to adjust the datawidth of the external memory to the data format used in VLIW processor. Of course, the fastest version is if there is only one register to store a complete VLIW word within one cycle. The state controller block, implemented as a finite-state machine controls whether the data goes to the IFE or LD/ST unit. The *ready*, *strobe* and *rw* signals coming from both units direct the program execution. The *strobe* signal from the LD/ST unit has highest priority. During the application of the sequential test-assembler program, the instruction and data load (or store) are continuously executed. The complete sequence needs to be started using the *restart_seq* port.

The test data flow through the IFE and LD/ST units is depicted in Figures 5.6 and 5.7 respectively [VAR00]. Of course the design of both units is more complex, consisting of both a data and control flow. The two figures only show the part relevant for traversing the test data. Figure 5.6 shows the test data transport from the input of instruction fetch unit to the control buses within the interconnection network. On the other hand, Figure 5.7 depicts the path of test stimuli from the inputs of LD/ST unit towards the data busses, but also the other way around, i.e. the transport of test responses from the data busses towards the outputs of LD/ST unit.

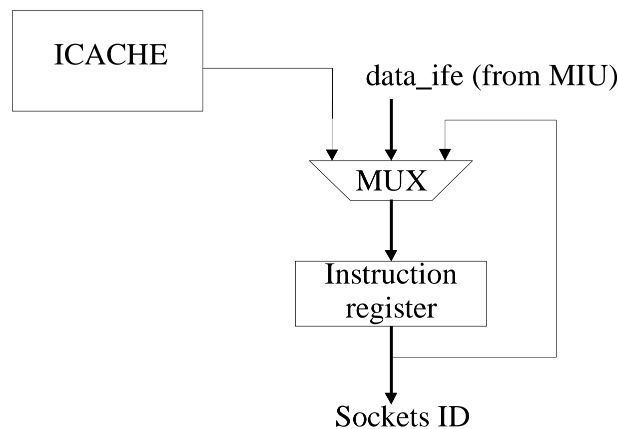


Figure 5.6. The test data flow in the IFE unit

The embedded instruction cache can also be used for testing purposes, as the access from the cache is faster than the access from the external memory. In the latter case, the instructions do not need to pass through the memory interface unit but will be directly taken from the embedded cache and brought to the appropriate bus. Of course, it would require that the instructions used during the functional-structural test are first stored in the cache. The MUX at the input of the IFE unit needs to be set in a state to pass the data coming from MIU. The role of the instruction register in Figure 5.6 is to hold the instruction during the decoding phase. Its outputs are connected to the control busses of the interconnection network housing the socket ID’s.

The memory run-out block of the LD/ST unit is organized as a FIFO buffer and its role is to freeze the internal data pipeline of the LD/ST unit in the case an interrupt occurs. During the test-sequence, of course, the interrupts do not appear and hence the test path within the LD/ST unit is straightforward. The trigger register in the LD/ST unit does not

participate during the application of the functional structural test because the test response will be finally captured in the DATA primary input/output port of the complete TTA processor (see Figure 5.3), not at the particular memory location.

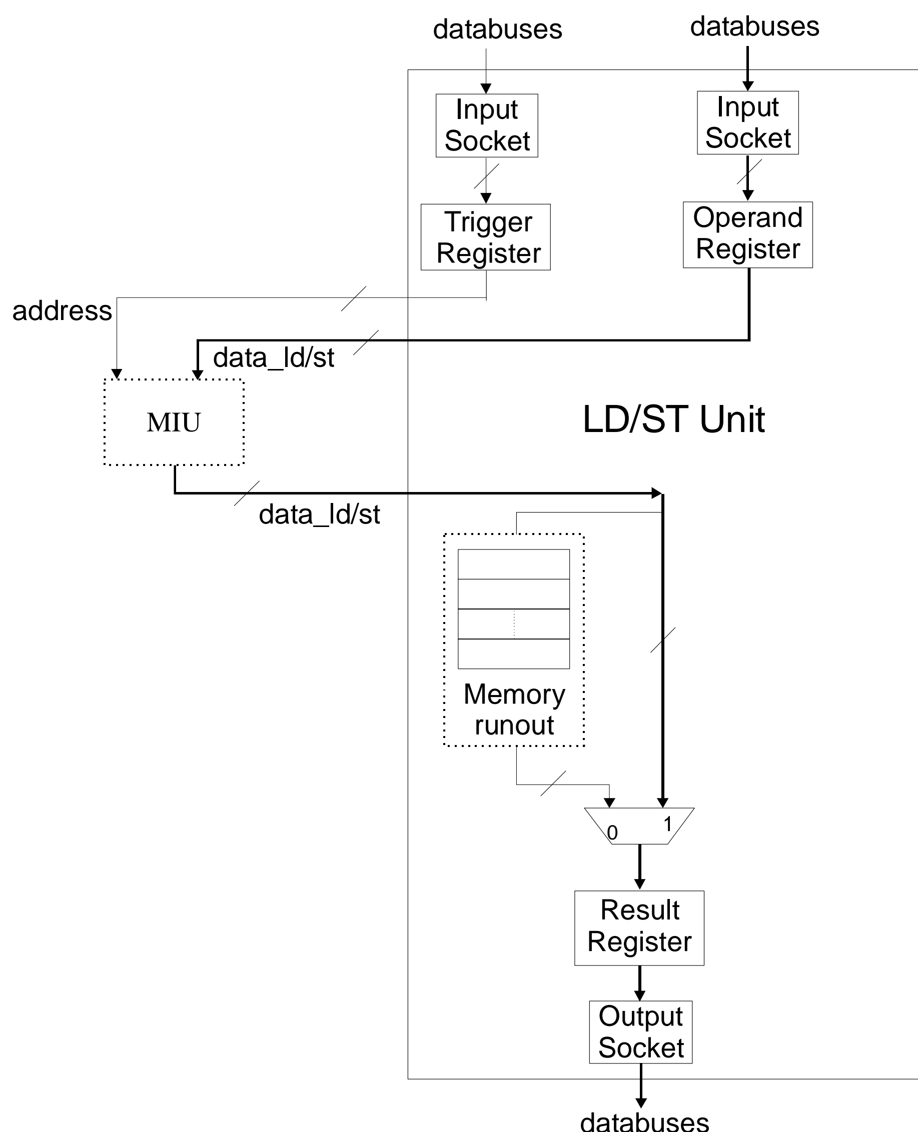


Figure 5.7. The test data flow in the LD/ST unit

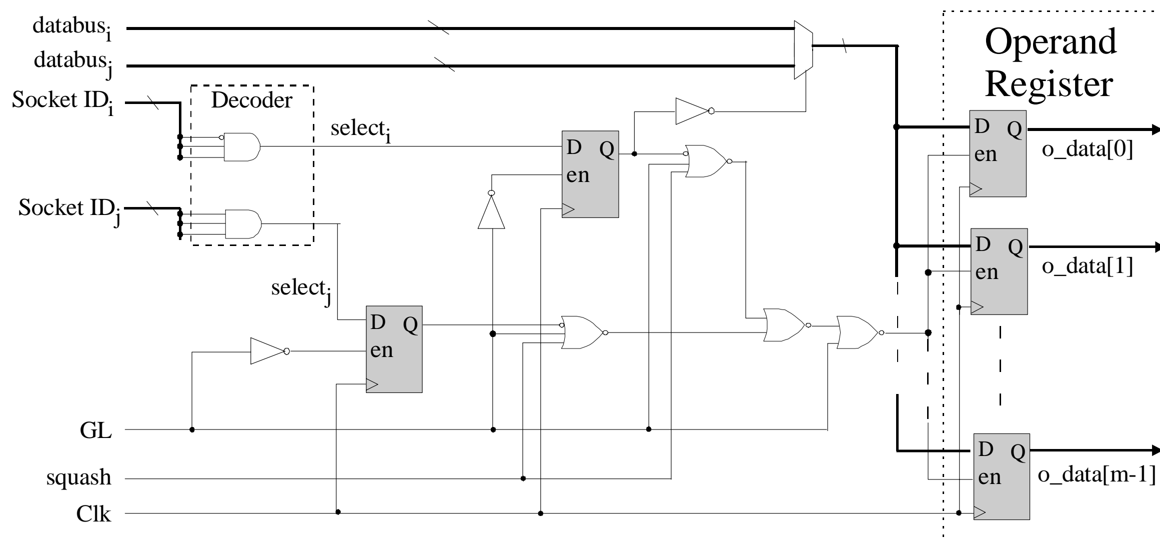
Before the application of the functional-structural test is illustrated in an example of a functional unit used in typical MOVE application, it is necessary to outline the implementation details of the input and output sockets.

5.3.3. Socket design

Both datapath and interconnection network are embedded in the design; hence their primary inputs/outputs are not accessible (Figure 5.1). The interconnection network, containing the sockets, has been tested in a functional way, by setting the control signals to pass (or not to pass) the structural test patterns for the components. This will decrease

the fault coverage of the overall processor, because the functional-structural test patterns target the faults inside the components, not the ones that may also pertain within the input/output socket circuitry, e.g. faults in decoder, control and sequential logic. However, the intention is not to introduce any DfT inside the datapath and sockets. Moreover, the detection of the faults within the sockets of one component is related to the application of functional-structural test of another component, as the fault simulation of the complete datapath, shown at the end of this chapter, will clarify.

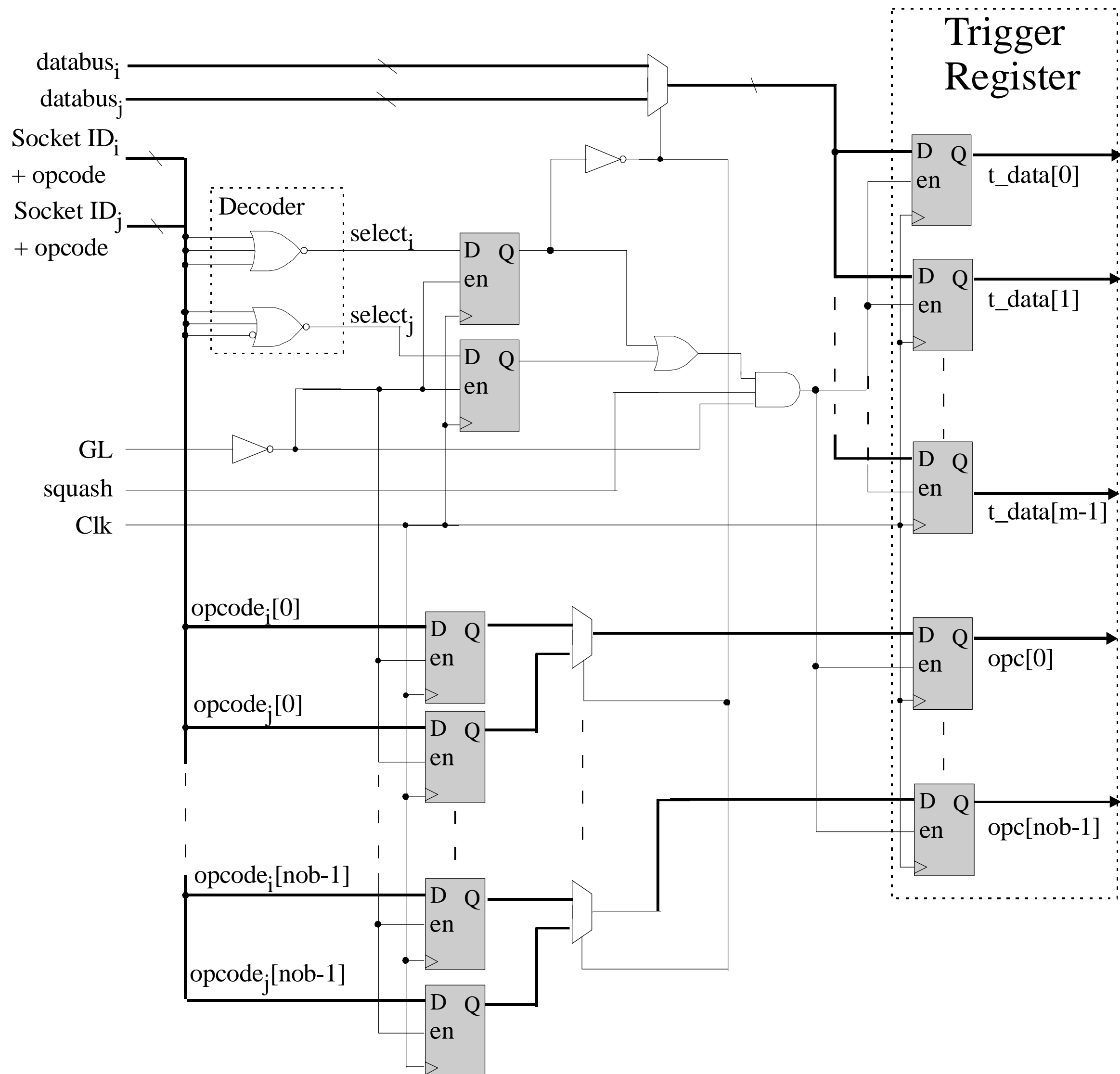
The test path through the input and output sockets has been drawn in the previous chapter in Figure 4.11. Figure 5.8 shows the test-path at the gate-level of the input operand socket of a functional unit.



5.8. A gate-level implementation of the input operand socket

The implementation is realized according to the scheme given in Figures 4.5 – 4.7. The design depicted in Figure 5.8 is not fixed for every input operand socket, since they all have different decoding functions and possibly different number of busses the component is connected to. However, those are the only differences among them, the remaining part of the control and sequential logic does not vary from one to another input operand socket. The implementation shown in Figure 5.8 has two data busses labelled *databus*$_i$ and *databus*$_j$ coming from the interconnection network with socket addresses *Socket_ID*$_i$ and *Socket_ID*$_j$ equal to 011 and 111, respectively. The test patterns are coming from one of the two data buses and they will be applied to the component if one of the *Socket ID'*s matches the instruction bits coming from one of the control buses. The *select*$_i$ (or *select*$_j$) signal is subsequently generated that enables the data to be stored in the operand register composed of D-flip-flops with enable signal. However, the data arrives one cycle later than the one from the data bus, and hence, the *select* signal has to be stored in a flip-flop before it is send to enable the operand register to accept the data. The same is true for the process of multiplexing that selects one of the data buses. Of course, *GL* (global lock) and *squash* signals, both of them generated in the case of an interrupt, also need to have appropriate values to enable the data storage into the operand register. The signal *o_data* goes directly to the input port of the component.

Figure 5.9 shows our implementation of the input trigger socket of the functional unit at the gate level.

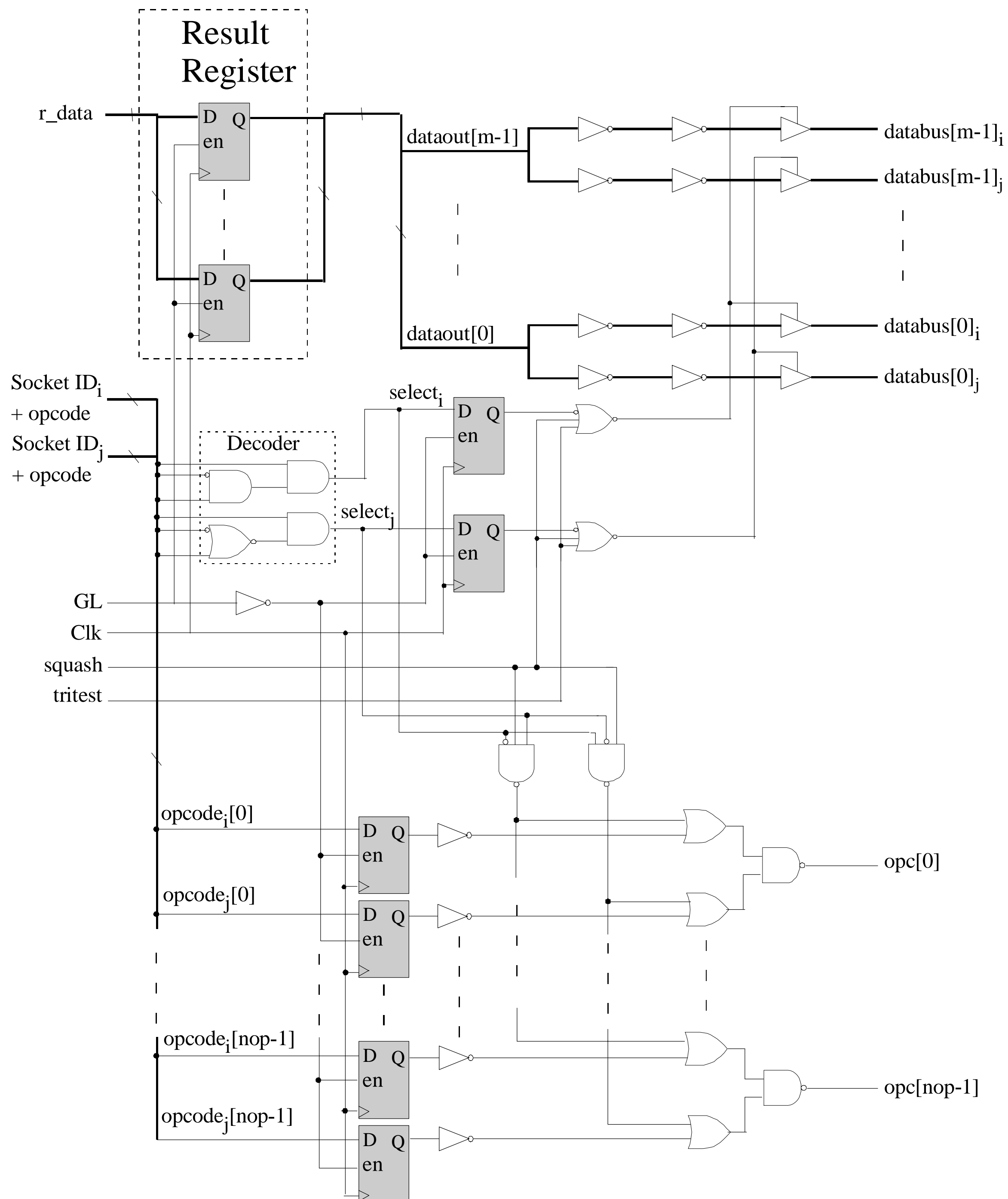


5.9. A gate-level implementation of the input trigger socket

Similar as its operand counterpart, the input trigger socket matches the “move” instruction bits using its decoder. However, the difference is that the trigger socket can eventually recognize the *opcode* of a certain instruction of the component (circuitry depicted in the lower half of the Figure 5.9), e.g., an ALU that supports multiple operations. The *nob* in the Figure 5.9 denotes the number of bits used for an opcode. Note that the opcode bits also require another pipeline stage before they are stored into the trigger register.

Finally, our output socket implementation at the gate level is illustrated in Figure 5.10. The circuit is similar to the input trigger socket design. When the output socket receives a control socket ID signal, it examines its value using the decoder. If that value matches its decoding function, one of the *select* signals will be generated and stored one cycle before

the data is sent to one of the data buses. The eventual *opcode* of an operation is handled in the same way as in the case of an input trigger socket. The resulting data at the output of the component (*r_data*), going to the result register, is now demultiplexed and put on a data bus after the component receives the opcode. Nevertheless in the recent version of MOVE architectures, the result register may be omitted thereby increasing in that way the overall throughput. Also, note the *tritest* signal added for the testing purposes, which clears off the buses during the scan-shift operation.



5.10. Output socket design

The described socket designs can be used to connect both functional units and register files to the interconnection network. In the latter case, the address of the location where the data has to be read or written is transported using the *opcode* ports of the sockets.

Example

The application of the functional-structural test at the gate-level with actual test patterns will be illustrated in a following example of an ALU taken from the library of the components. An ALU is a key core of the processing in MOVE architectures. It executes a set of arithmetic and logic micro operations on two input buses (*A* and *B*). It has *nob=6* encoded inputs (*S[4:0]&Cin*) for selecting which operation to perform. The select lines are decoded within the ALU to provide up to 2^{nob} possible different operations. The ALU in the given example, whose conceptual schematic is drawn in Figure 5.11, is capable of carrying out 14 different micro-operations, as listed in Table II.

TABLE II THE DIFFERENT ALU FUNCTIONS

Sel[4]	Sel[3]	Sel[2]	Sel[1]	Sel[0]	C_{in}	Operation	Function
0	0	0	0	0	0	$C := A$	Transfer A
0	0	0	0	0	1	$C := A + 1$	Increment A
0	0	0	0	1	0	$C := A + B$	Addition
0	0	0	0	1	1	$C := A + B + 1$	Add with carry
0	0	0	1	0	0	$C := A + \overline{B}$	A plus 1's complement of B
0	0	0	1	0	1	$C := A + \overline{B} + 1$	Subtraction
0	0	0	1	1	0	$C := A - 1$	Decrement A
0	0	0	1	1	1	$C := A$	Transfer A
0	0	1	0	0	0	$C := A \text{ and } B$	AND
0	0	1	0	1	0	$C := A \text{ or } B$	OR
0	0	1	1	0	0	$C := A \text{ xor } B$	XOR
0	0	1	1	1	0	$C := \overline{A}$	Complement A
0	0	0	0	0	0	$C := A$	Transfer A
0	1	0	0	0	0	$C := \text{shl } A$	Shift Left A
1	0	0	0	0	0	$C := \text{shr } A$	Shift Right A
1	1	0	0	0	0	$C := 0$	Transfer 0's

The ALU has been modelled with a separate arithmetic, logic and shifting unit as indicated by the modelled circuit structure. Separating the arithmetic and logic unit in this way, while multiplexing their outputs to the shifter, results in a better starting point for the synthesis [SMI98]. This structure uses unsigned data operands. The synthesis, using a Synopsys Design Compiler resulted in a circuit of 610 gates.

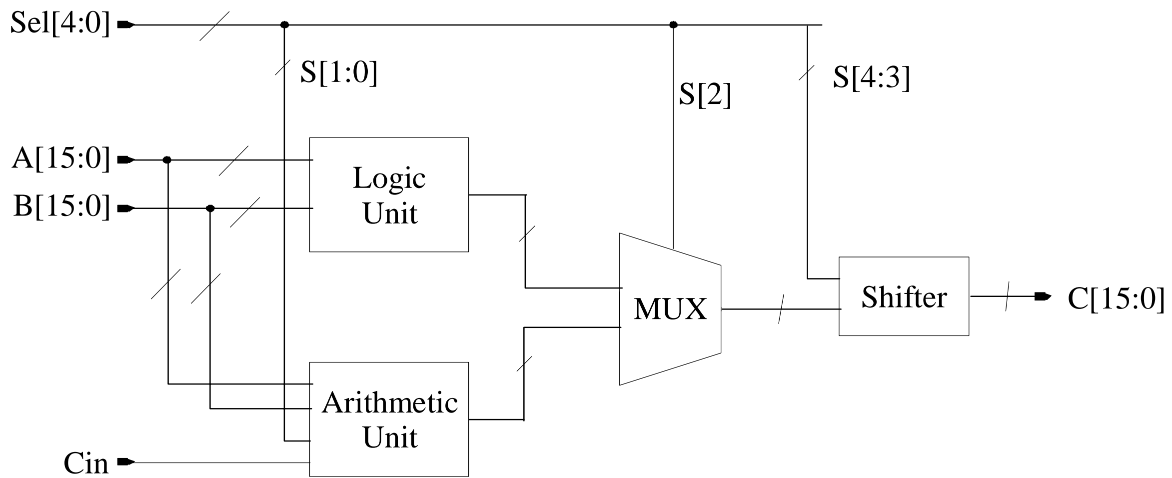


Figure 5.11 ALU conceptual structure

The ATPG of such a synthesized circuit was straightforward and a number of 69 test patterns targeting stuck-at faults have been obtained. For example one resulting test pattern obtained after ATPG has the following form:

TABLE III A TEST PATTERN OF ALU UNIT

Sel[4:0]	Cin	A[15 ....0]	B[15....0]
00011	1	1110011101100100	0101101101101001

However, the actual test pattern list needs to be modified in such a way that the bits going to *Sel[4:0]* and *Cin* input ports of the ALU, i.e., its opcode, are merged with the ALU input socket ID's and applied one cycle earlier than the remaining, data part of the test pattern. Figure 5.12 shows the ALU equipped with sockets, with signal names given according to the notation in Figures 5.8 – 5.11. According to that notation and the notations in Figure 5.4, the following correspondence may be derived (the symbol "&" denotes the concatenation of bit-vectors):

Socket ID =ALU_ID ={os_id, ts_id, rs_id}
opcode = Sel[4:0]&Cin, nob = 6
Data = {A[15:0]&B[15:0]}
Socket_ID (operand socket) = os_id
Socket_ID (trigger socket) = ts_id
o_data = A
t_data = B
opc [5:0] = Sel[4:0]&Cin
Socket_ID (output socket) = rs_id
r_data = C

The output result socket does not transport opcode in the case of an ALU functional unit, so its opcode is omitted. It was assumed that the data to input *A* comes from the operand socket while the trigger socket sends the data to the *B* input of an ALU.

Assuming e.g. that the ALU_ID's of *os_id = 010* and *ts_id = 011* for operand and trigger registers respectively, the actual test stimuli obtains the following form:

TABLE IV THE TEST STIMULI APPLICATION

Cycle	os_id&ts_id	opcode	Sel[4:0]&Cin	A[15:0]	B[15:0]
T	010011	000111	XX…XX	XX…XX	XX…XX
T+1			000111	111…100	010…001

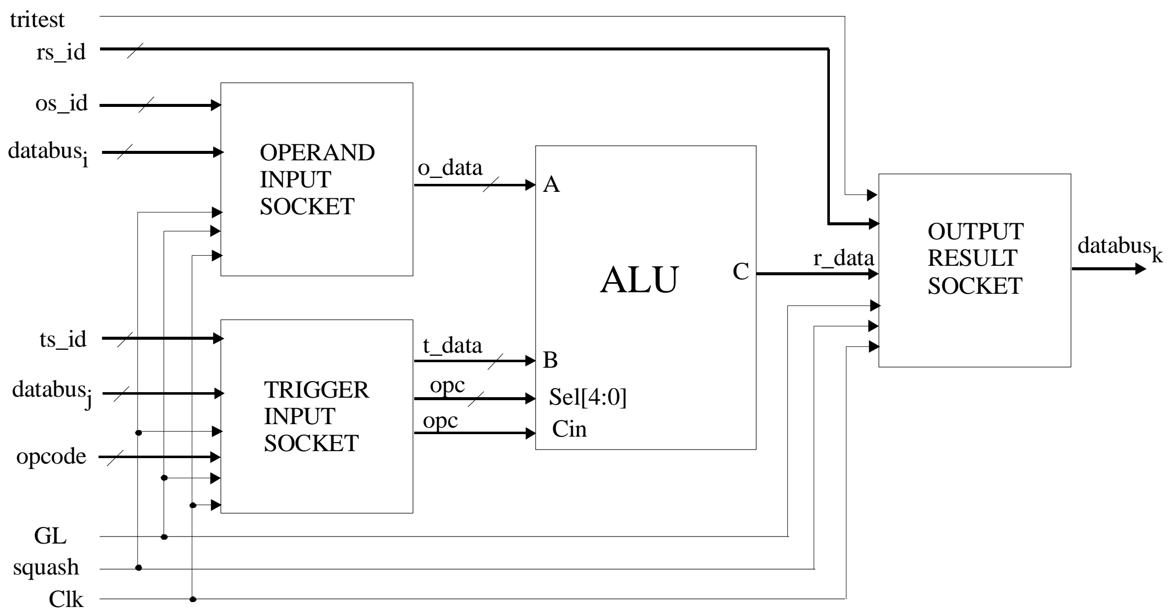


Figure 5.12. ALU and its sockets in the datapath of TTA

Pipelining allows that the opcode part of the next pattern i.e., *Sel[4:0] & Cin* in this case, can be scheduled in the same cycle together with the data parts of the preceeding test pattern. This is illustrated in the table bellow, where the test pattern belonging to the certain timing slot is enclosed within the brackets, with bracket index denoting the timing slot, itself:

TABLE V THE TEST STIMULI PIPELINE

Cycle	os_id&ts_id	Opcode	Sel[4:0]&Cin	A[15:0]	B[15:0]
T	(010011	$000111)_1$	XX…XX	XX…XX	XX…XX
T+1	(010011	$000010)_2$	$(000111)_1$	(111…010	$010…001)_1$

The test response is available in the result register of the component a number of cycles after the test-stimulus is applied, depending on the component's latency. In the case of this ALU design, the latency is one, and hence, the result appears after one cycle at the output C[15:0]:

TABLE VI THE TEST STIMULI PIPELINE AND RESPONSE

Cycle	os_id&ts_id	opcode	Sel[4:0]&Cin	A[15:0]	B[15:0]	C[15:0]
T	(010011	000111)$_1$	XX…XX	XX…XX	XX…XX	XX…XX
T+1	(010011	000010)$_2$	(000111)$_1$	(111…010	010…001)$_1$	XX…XX
T+2	(010011	000001)$_3$	(000010)$_2$	(101…001	011…011)$_2$	(111…100)$_1$

Finally, after the result is available in the result register, a socket ID for the output port of the ALU (*rs_id*) needs to be applied in order to read the result. Assuming its code is 100, the sequence of the test stimuli application and test response capturing test pattern is as following:

TABLE VII THE COMPLETE SEQUENCE OF THE TEST STIMULI APPLICATION AND TEST RESPONSE CAPTURING OF ONE TEST PATTERN OF THE ALU UNIT

Cycle	os_id&ts_id opcode	Sel[4:0]&Cin	A[15:0] B[15:0]	rs_id	C[15:0]
T	(010011 000111)$_1$	XX…XX	XX…XX	XX..X	XX…XX
T+1	(010011 000010)$_2$	(000111)$_1$	(111…010 010…001)$_1$	XX..X	XX…XX
T+2	(010011 000001)$_3$	(000010)$_2$	(101…001 011…011)$_2$	(100)$_1$	(111…100)$_1$

Of course, the test pattern response still needs to be transported through the network to the operand register of the LD/ST unit, assuming that the trigger register of the LD/ST unit is enabled during the test-response capturing, see Figure 5.7.

The presented test-sequence assumes the situation when the test response capturing does not interfere with the test-pattern stimuli application. However, this depends to a large extent of how the MIU and LD/ST units are designed. For example, Figure 5.5 shows there exists only one data-port at the interface between MIU and LD/ST and MIU and External Memory transferring the data in both directions. This means that after the first test stimulus to the CUT is applied, the succeeding one may not be applied before the data is captured and brought to the primary outputs. The current version of MPG only supports this type of MIU and LD/ST design, which not only slows down the test but also the application itself. It is expected that a more advanced version becomes available in the future. Alternatively, the current version may be extended in such a way that more than one Load/Store unit is incorporated in the design. Of course, the area will be significantly increased in this case. However, that is the trade-off that needs to be tackled earlier in the design, during the design space exploration phase.

The symbolic waveforms in Figures 5.13 and 5.14 depict the simulation results when the functional-structural test is applied to the ALU. They illustrate two situations. The first, shown in Figure 5.13, is the fastest one in the case the bandwidth of the MIU and LD/ST is sufficient to apply the stimuli and read the responses in parallel. Of course, the input and output ports of the ALU need to be connected to the busses in such way to allow for simultaneous test-pattern application, as explained in previous chapter.

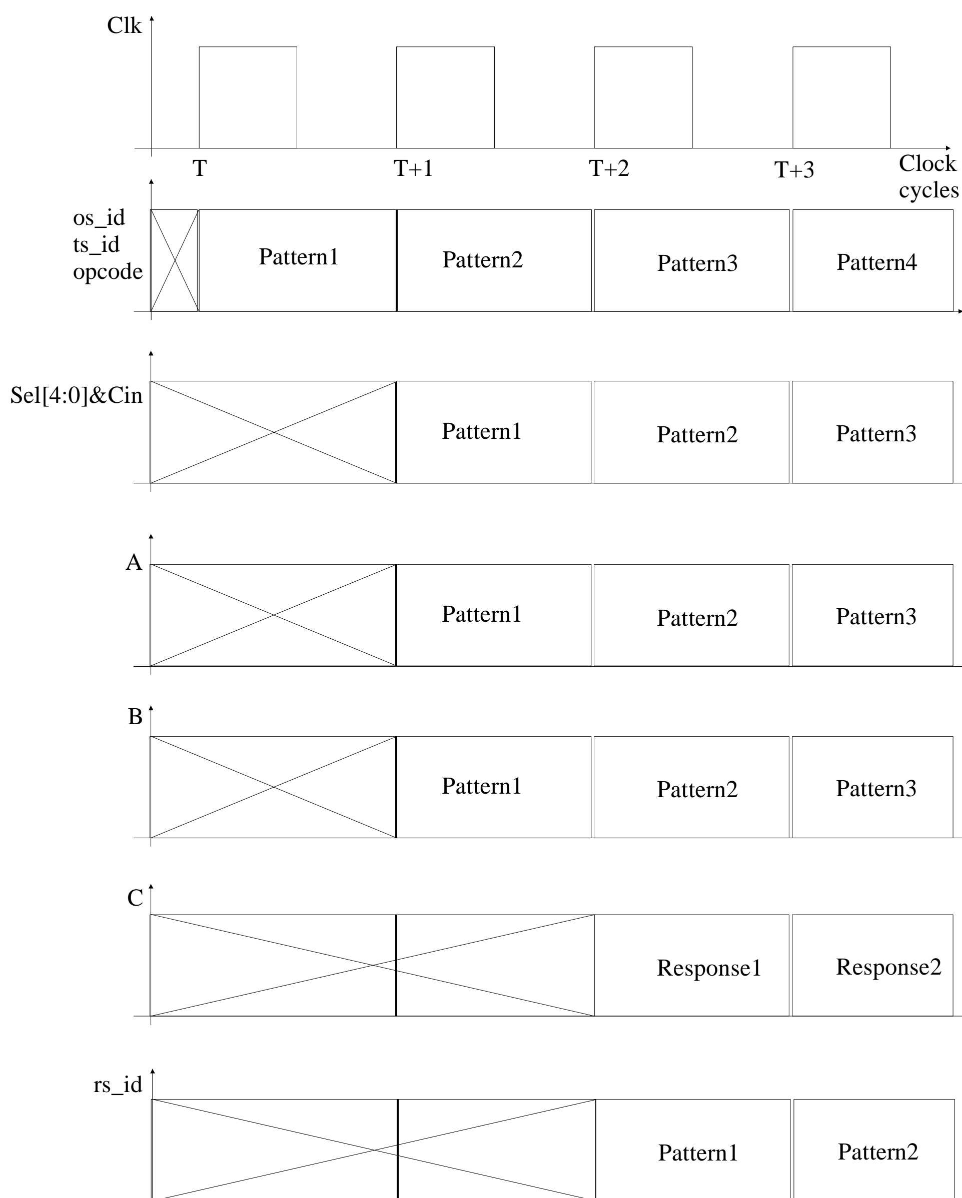


Figure 5.13. Response evaluation of an ALU equipped with sockets

The waveforms in Figure 5.14 reflect the case when the current LD/ST and MIU units with limited capabilities are employed in the design. Each transport (MOVE) to one of the registers in this situation requires its own timing slot both for instruction fetching and actual data transport. Hence, pipelining cannot be exploited. Consequently, the application of the functional-structural test requires more test-cycles in this case. However, that number is still much lower as compared to the full-scan approach as will be shown in the subsequent sections. The term “Hold” in the Figure 5.14 has the meaning that the pattern values are kept steady in the registers during multiple timing slots.

The *Data_on_bus* signal corresponds to the $databus_k$ from Figure 5.12. This signal is also given to show that transportation of the test response to the *DATA* primary input/output port (Figure 5.5) also requires its own timing slot in which the instruction fetch is not possible. Actually, the waveforms shown in Figure 5.14 can also be regarded

from the *DATA* port, which is kept busy up to the maximum extent during the test of the datapath.

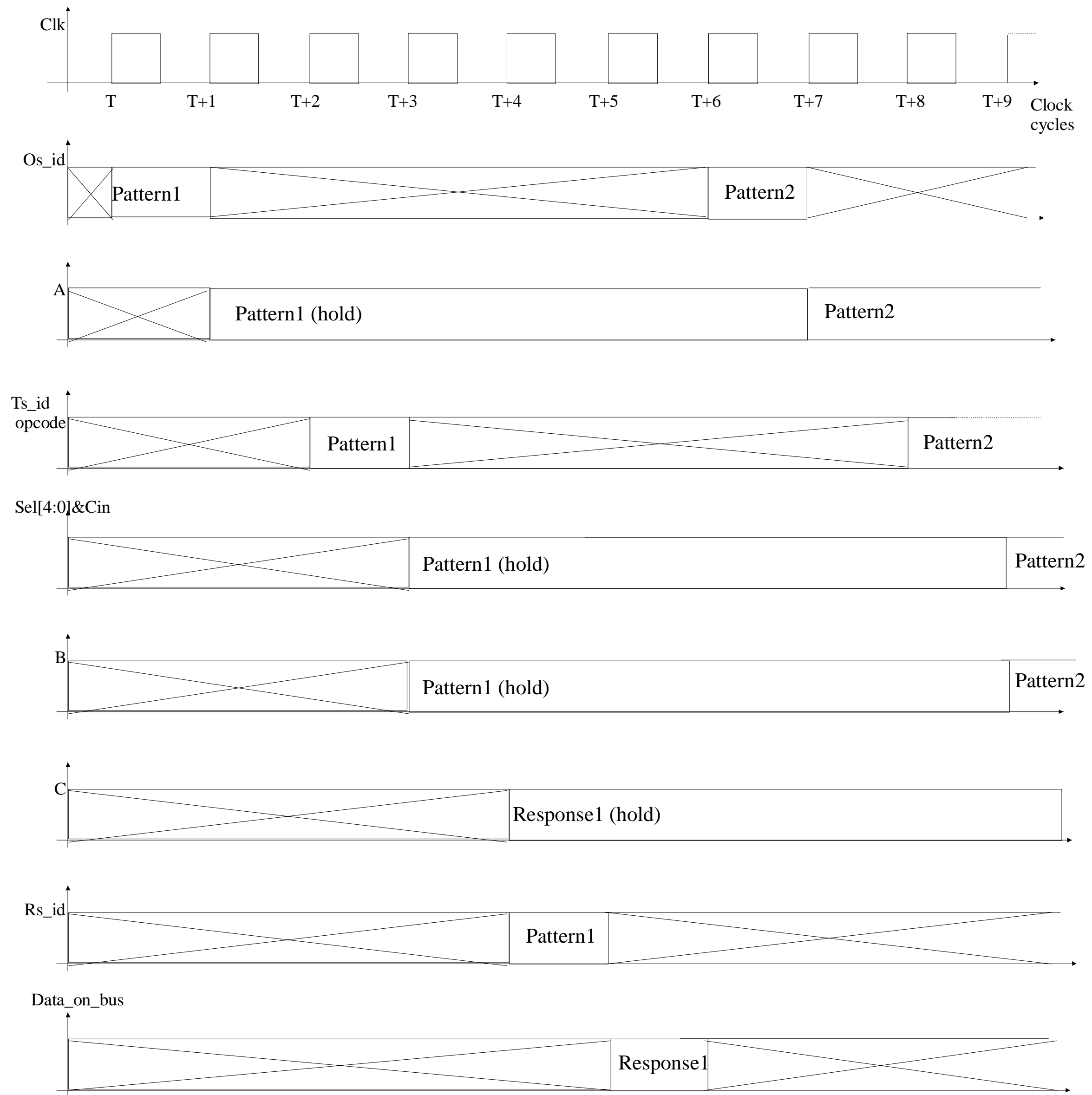


Figure 5.14. Response evaluation of the ALU equipped with sockets with the current implementation of MIU and LD/ST units

The simulator used to obtain the waveforms in Figures 5.13 and 5.14 was Verilog XL 2.8, part of the *Cadence* design kit running on HP-UX 10.20 platform.

==

All functional units used in the design can be tested in the presented fashion, no matter their internal design. The same is true for register files. The register file test patterns are applied and read using the proposed functional-structural test without any difference in the way how the patterns are applied referring to the functional units in the design. The only

difference is that the operand input socket in the case of register files does not exist i.e., an input socket is always a trigger one. The register file from the data path has been tested with a marching-test like algorithm, which is slightly more complex than for a RAM due to the simultaneous read while writing to the register file [SAC97]. Consider the register file with one input and two outputs, as given in Figure 5.15. The register file is embedded in the datapath and equipped with sockets in the same way as a functional unit. Also, the socket ports have the same functions and notations as in the case of functional units. The write port of the register file is denoted as *W_Data*, while the two read ports are denoted as *Read1* and *Read2*, respectively. The port that is carrying the address bits in case of writing to register file is denoted as *W_Addr*, while two read address ports are labelled as *R1_Addr* and *R2_Addr*, respectively.

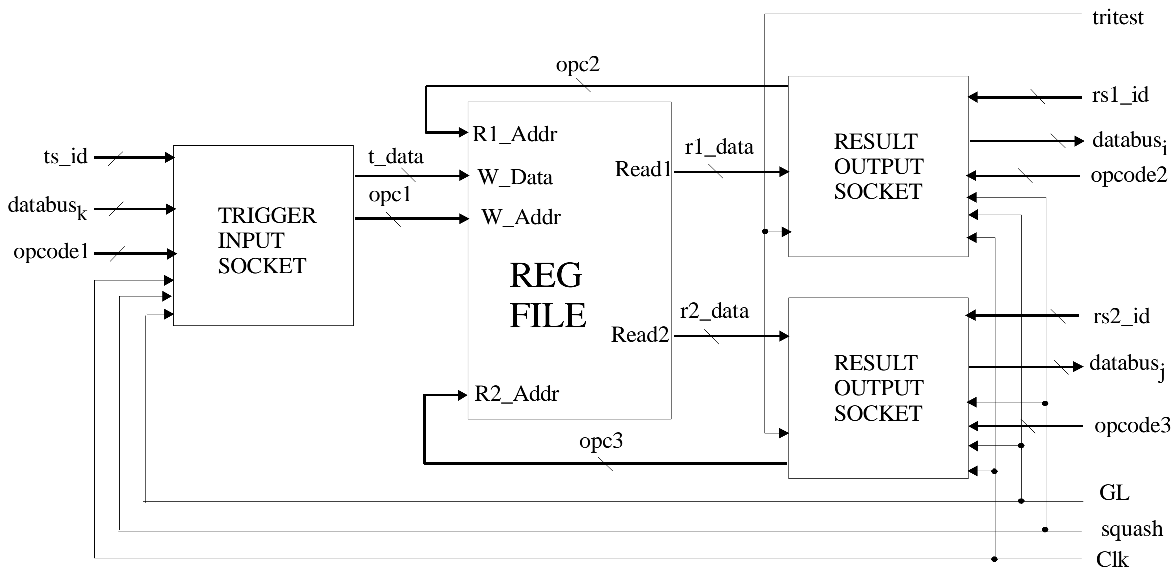


Figure 5.15 Register file with one input and two outputs equipped with sockets in the datapath of TTA

Let us assume that the data bit-width has value of N_{bit} while there are M_{addr} bits used to transport the address of the register file locations. Also, let *Laddr* denote the address of the last location. Note that $Laddr \leq 2^{Maddr}$, as it is not mandatory that all addressable locations be in use within the design (see, e.g., previous chapter and the example of Crypt application where the register file RF1 has 12 locations). The test procedure for the register file consists of the following steps:

- Step1: A write from address *0* until the *Laddr-1* is done, with the data written in register file equal to the address value.
- Step2: A simultaneous read at the two data output of the register files is performed. This step can be actually combined with previous steps in order to enhance the speed of the algorithm. For example, while writing to address *i*, a read at *Read1* output port can be performed by reading address *i-1* while read at *Read2* port is done by reading address *i-2*.
- Step3: After the write operation is completed, a new simultaneous read at the two data outputs are carried out. However, in this case, the read only part starts by

reading at *Read1* output port address *0* going towards *Laddr –1* , while the read at output port starts at the address *Laddr –1* ending at address *0*.

- Step4: The first 3 steps are repeated with checkerboard data (0101…01) and (1010…10) for the two adjacent locations [GOO91].
- Step5: Similarly to Step4, the first three steps are performed with inverted checkerboard data.

The described procedure detects Stuck-at faults and shorts between adjacent cells. Not all memory cell faults according to the fault models given in [GOO91] can be detected with this algorithm, though. However, the applied approach is justified, as the size of the register files used in typical MOVE application is not large. Moreover, in order to enhance the architecture efficiency, large register files are usually split into smaller ones [JAN95]. The small sized register files do not have high fault complexity that requires thorough fault modelling and applications of various memory test algorithms such as described in e.g. [HAM99] targeting the n-port memories. However, if the testing requirements include the analysis of larger set of fault models, the presented test sequence needs to be extended with additional n-port memory algorithms.

5.3.4. Control-block test

The control block is not customisable during the design-space exploration phase, i.e. the control block consists always of the same elements listed in the previous section. Certain parameters inside the control block are, of course, customisable, such as e.g. instruction cache size, memory interface, exception handler, etc. In any case, the control block is comprised of random logic and sequential elements. It has been decided that the full-scan technique supplemented with memory test will be used for testing of the complete control block, including its sockets. Of course, one may object that it requires certain DfT modifications that will result in an area increase and throughput deterioration. However, as already explained in the previous chapters, the non-exhaustive functional test can not achieve an acceptable level of fault coverage. In addition, it has been already explained that the datapath is the most critical part of the processor in terms of performance and the major subject of the design-space exploration, Hence the control part is less sensitive to the DfT insertion than the datapath and interconnection network.

In order to prepare the control unit for full-scan test, certain modifications of the design had to be done. These modifications, carried out at the Register-Transfer Level (RTL) by the designer and driven by the full-scan DfT constraints are as follows:

- Multiple-clock domain handling. The memory interface unit has internal generation of its clock. Hence, an additional *process* statement is inserted in the RTL-code. The result of this statement is that a multiplexer is introduced to bypass the internally generated clock, according to Figure 5.16. The test control signal *Te* is used to pass either the MIU internally generated clock during the functional mode of the circuit or the system clock (*Clk*) during the full-scan test. The shaded colour of the flip-flops is used to distinguish scanable flip-flops from the originals.

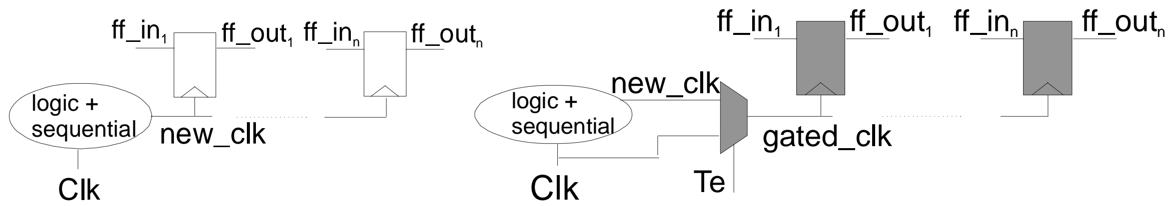


Figure 5.16. Multiple-clock handling for a full-scan test

- Three-state busses. As already mentioned earlier in this thesis, all components in the design are connected to each other via the interconnection network. Obviously, the data buses used in the design are bi-directional, implying the use of three-state logic in the sockets. During the shift mode of the full-scan, it is necessary to provide that no more than one input driver is active. The same requirement holds for an Iddq test pattern generation. Hence, an additional test signal (*tritest*) is brought to the sockets as the Figure 5.17 outlines.

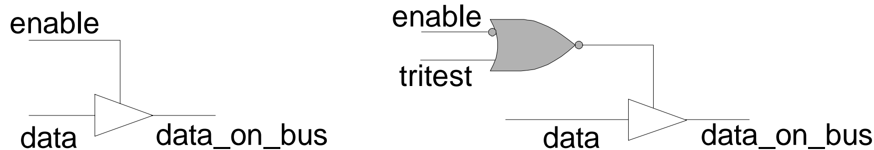


Figure 5.17. The additional test control signal “tritest” prevents bus conflict during testing

- Elimination of latches. As already mentioned, the latches decrease the fault coverage of the clocked design when full-scan is applied.

After completion of the above modifications, the insertion of the scan-chains at the gate-level and subsequent ATPG was relatively straightforward by means of an appropriate in-house industrial tool. Of course, not all ports of the control block are available for direct test control or observation. The ports that are connecting the control block with the interconnection network and datapath need to be defined as “skipped” prior to the ATPG of the control block. This was necessary, as these ports are embedded and not accessible for ATPG to generate signals or observe the outputs (see Chapter 3). Figure 5.18 depicts the control block and denotes the signals that may be used as primary inputs, outputs or (bi-directional) i/o’s and the signals that need to be skipped. Also, the additional test pins are indicated in the figure.

Socket_ID and *databus* are control and data busses ports with *k* and *m* bit-widths, respectively, while *nb* denotes the number of busses in the design. Hence, there are *2nb* control buses, for destination socket ID’s (the input socket ID’s) and the source ID’s (referring to the output sockets). The ports *si*, *so*, and *se* denote the scan-inputs, scan-outputs and scan-enable signals, while *te* and *tritest* are the test-control signals to handle the presence of multiple-clock domains or three-state busses.

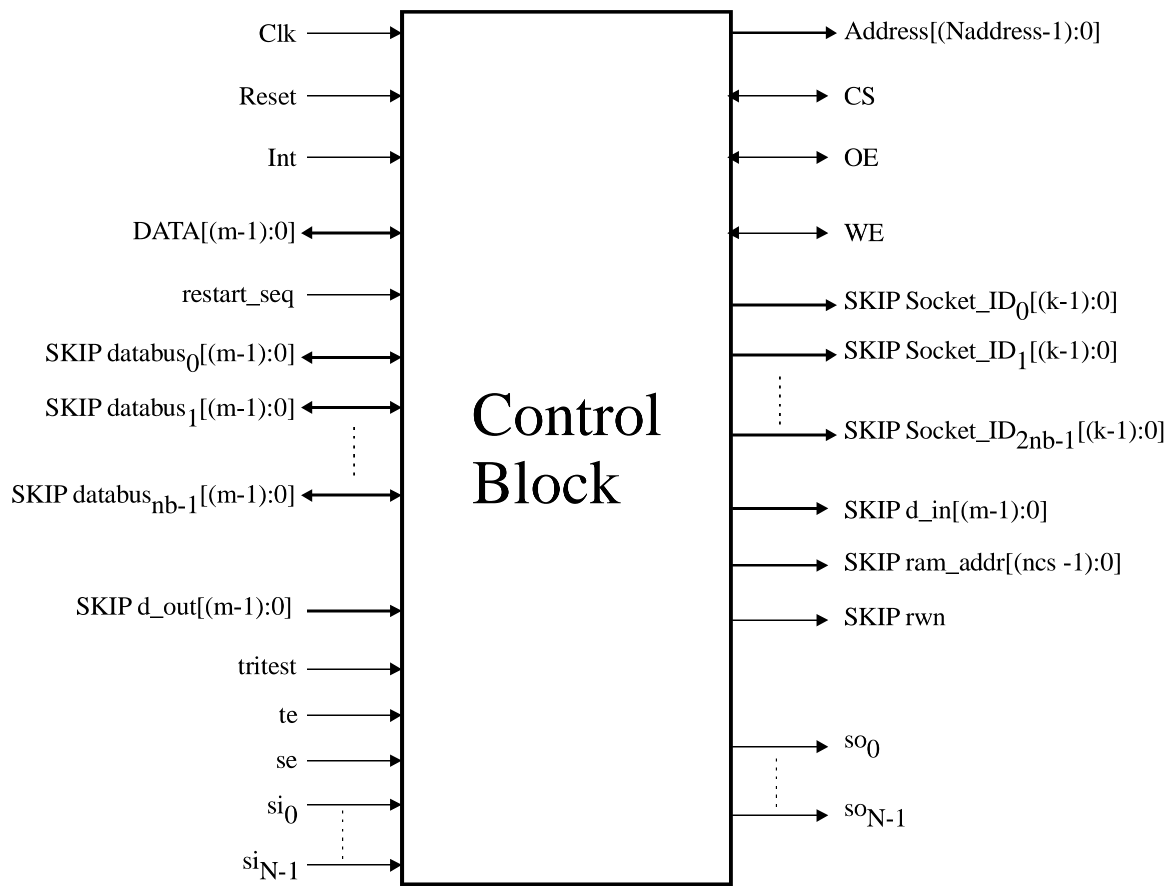


Figure 5.18 Control Block pins

The signals *d_in*, *d_out*, *ram_addr* and *rwn* with their corresponding bit-widths are the ports of the generic instruction cache (ICACHE) that has been extracted from the VHDL code of the instruction fetch unit. *m* denotes the bit-width of data while *ncs* indicates the number of bits used to address the locations within the cache. Of course, in the latter stage of the design, the generic cache, size of ($m \times 2^{ncs}$) needs to be inserted and subsequently tested. In order to insert an embedded memory, the industrial in-house library of memory blocks of various sizes has been explored and an appropriate memory inserted. No BIST solution is implemented, as the memory size (typically, the cache would not exceed 4 Kb) would not justify the area overhead of that kind of solution. The accessibility to the memory inputs has been achieved in a similar fashion as described in [BEE98] with an exception that the memory outputs have already been registered with flip-flops, as Figure 5.19 depicts. The isolation cells from Figure 5.19 have been implemented in the same fashion as the test shells described in Chapter 3. Therefore, they are transparent in the functional mode. The isolation of the memory inputs from the interface logic during memory test is provided with *hold* signal. During scan shift mode, memory operations are blocked with the scan enable signal *se* (Figure 5.19). During normal mode, the functional memory port *rwn* allows access to or from the memory. The embedded instruction cache (block ICACHE from Figure 5.1) can be tested with the MATS+ marching memory test algorithm described in [GOO91], which is also convenient in the case when used memory technology is not known. In order to achieve the optimal transportation of test stimuli and

responses via scan chains the flip-flops from the isolation cells that control memory inputs have to be located at the 'head' of the scan chains, while the flip-flops at the output need to be placed at the 'tails' of scan chains. This arrangement will guarantee optimal solution during the test pattern expansion of memory test patterns at the top level of the processor.

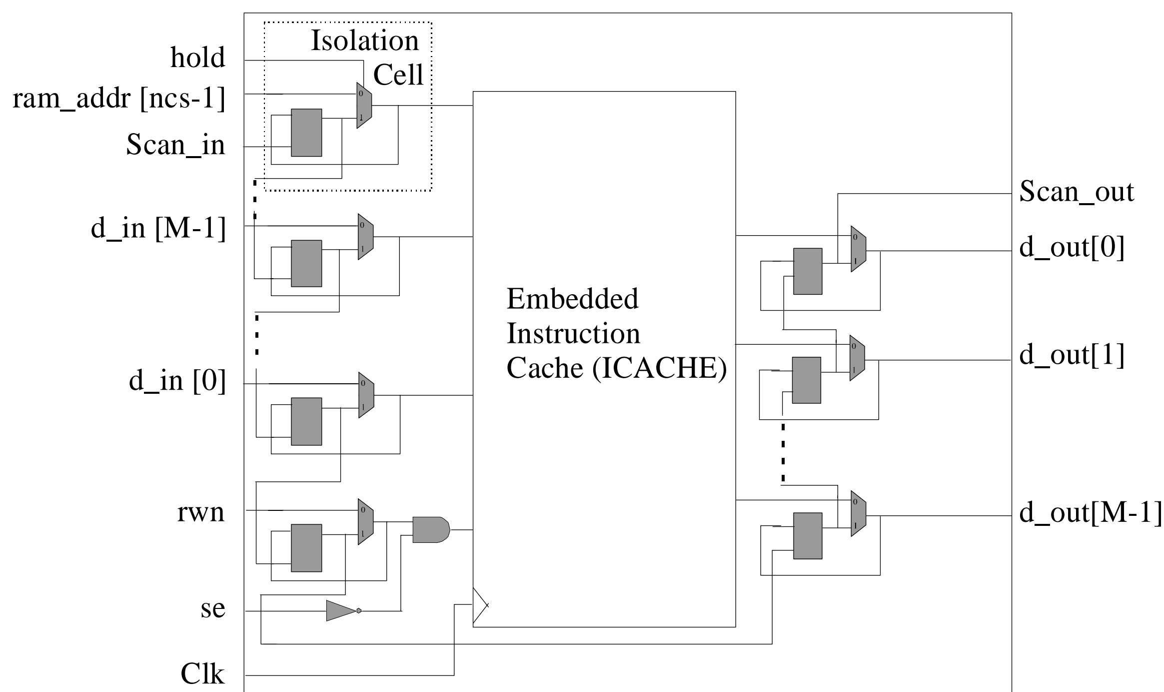


Figure 5.19. The embedded cache test

Due to the readability reasons, the multiplexer control signal (hold) distribution over all isolation cells, as well as the distribution of clock and scan enable signals over the flip-flop circuitries are not shown in this figure.

The marching test patterns of the embedded cache are expanded with the full-scan patterns using the test-pattern expansion tool to create the complete control-block test patterns. That has also been done using an industrial in-house test-expansion tool.

The order of test is important with such partitioning of the test resources. It is obvious that before the functional-structural test of the datapath is undertaken, it is necessary to complete first the full-scan and memory test of the control part. This, because the test patterns and responses for datapath components will traverse the control block first.

The proposed method tests the datapath in an at-speed fashion, since it applies the real-time stimulus and evaluates the real-time response. Of course, in case that an external ATE is used, the term "at-speed" will be limited to the speed of testing device. The functional-structural test does not require any DfT inside the datapath and interconnection network of the processor, being the most critical part with respect to the performance. However, the price is paid by a somewhat lower fault coverage, as the structural test-patterns (*DATA* part from figure 5.5) do not target the faults that may exist within the sockets. Though, the fault coverage of the complete processor may still be sufficiently high, as will be proven in the sequel of the chapter.

5.4. Test-time analysis

Test-time analysis [ZIV01a], [ZIV01b] is required to prove the advantage of our approach over the full-scan technique and gives the boundary condition for which this advantage holds. Prior to the analysis, the following parameters given in the table VIII, relevant for the proposed test method, have to be introduced:

TABLE VIII THE DESIGN & TEST PARAMETERS

ff	# of flip-flops in the circuit
N	# of scan chains
f_{tc}	test cost of the component *c*
p	# of test patterns
L_{op}	latency of the operation *op*
T_{clk}	clock period

The full-scan technique consists of three phases: 1) scanning in of the test stimuli to fill all flip-flops, 2) the application of the scanned pattern, 3) scanning out the response. However, the first and third phase may be scheduled together, so that when the captured response of the previous pattern is scanned out, the next pattern is already scanned in. The flip-flops are, with an appropriate scan insertion tool, well balanced so the time that is needed to scan in (scan out) one pattern (t_1) is:

$$t_1 = \left\lceil \frac{ff}{N} \right\rceil T_{clk} \qquad (5.1)$$

With this definition, the time t_{fs} to test the overall design in a full-scan fashion expressed in the number of clock cycles T_{fs}, amounts to:

$$T_{fs} = \frac{t_{fs}}{T_{clk}} = \left\lceil \frac{ff}{N} \right\rceil (p_{fs} + 1) + p_{fs} \qquad (5.2)$$

where p_{fs} denotes the total number of full-scan patterns. The term “+ 1” means that scan in-out scheduling cannot be used in the case of scanning out the last pattern.

The time required to test the design using our approach (t_{new}) is the sum of the full-scan time to test the control block including the embedded memory and the time required for testing the datapath together with the interconnection network. With assumption that the number of available scan chains (N) remains the same and the clock with the same period is used for both purposes, the normalized total time equals to:

$$T_{new} = \frac{t_{new}}{T_{clk}} = T_{control} + T_{dpic} \qquad (5.3)$$

where $T_{control}$ and T_{dpic} are also expressed in the number of clock cycles, i.e., they are all divided by the clock period T_{clk}. In this case, the full-scan test of the control block requires the following number of test cycles:

$$T_{control} = \left\lceil \frac{ff - ff_{dpic}}{N} \right\rceil (p_{con} + 1) + p_{con} \qquad (5.4)$$

where f_{dpic} represents the number of flip-flops within the interconnection network and the datapath. These flip-flops are not made scannable. The number of control block test patterns (including the memory patterns) is labelled p_{con}.

The number of clock cycles required for functional-structural test of the complete datapath is expressed as:

$$T_{dpic} = \sum_{i=1}^{n_c} \delta_i L_i \qquad (5.5)$$

where n_c denotes the total number of the components (functional units and register files) inside the datapath. The number of the structural test patterns necessary to test the *i-th* component is denoted by δ_i, while L_i represents the latency of all operations required to apply and read the test patterns during the test of the *i-th* component. L_i consists of the three components: 1) latency of the *Load* operation (L_{ld}) that reads the patterns from the external memory and brings them to the input registers of the components, 2) the test cost of the component under test as introduced and described in the previous chapter (f_t), and 3) the *Store* operation latency (L_{st}) that transport the test response from the result register of the component towards the external memory. Ideally, the Load and Store operation should be pipelined, so that one does not have to add both values to the overall latency. However, the MIU design shown in Figure 5.5 has only one data bus primary port (*DATA*) of bi-directional nature so that Load and Store operation in parallel are not supported. Hence, the total latency of component *i* (Lt_i) can be expressed as:

$$Lt_i = L_{ld} + L_{st} + \frac{f_{t_i}}{\delta_i} \qquad (5.6)$$

The term f_{ti}/δ_i stands for the test-cost function per single pattern, as the operation latency of the component is defined per single pattern. The latency of the Load operation depends on the bandwidth of the *DATA* primary port in Figure 5.5 and the width of the *data_ld/st* and *data_ife* ports of the LD/ST and IFE unit, respectively. In the best case, when these widths are equal, the latency of Load and Store operation are 2 because there exists only one bus to transport both instructions merged with opcode and data during the Load operation, i.e., when the latch registers at the output stage within MIU match the bit-width of the *DATA* port during Store operation. Inserting the formula (4.14) into formula (5.6), the formula (5.5) gets the following form:

$$T_{dpic} = \sum_{i=1}^{n_c} [\delta_i \max\left(\left\lceil \frac{n_{port}}{n_b} \right\rceil, \left\lceil \frac{n_{in}}{n_{out}(LD/ST)} \right\rceil, \left\lceil \frac{n_{out}}{n_{in}(LD/ST)} \right\rceil \right)_i + CD_i(t_{Din}, t_{Dout}) - 1$$
$$+ \delta_i (L_{ld} + L_{st})] \qquad (5.7)$$

For the sake of readability, the formula (5.7) can be rewritten as:

$$T_{dpic} = \sum_{i=1}^{n_c} (a_i \delta_i + b_i) \qquad (5.8)$$

with coefficients

$$a_i = \max\left(\left\lceil \frac{n_{port}}{n_b} \right\rceil, \left\lceil \frac{n_{in}}{n_{out}(LD/ST)} \right\rceil, \left\lceil \frac{n_{out}}{n_{in}(LD/ST)} \right\rceil \right)_i + L_{ld} + L_{st} \qquad (5.9)$$

and

$$b_i = CD_i(t_{Din}, t_{Dout}) - 1 \qquad (5.10)$$

Combining (5.7) with *(5.4)* and *(5.3)*, one obtains:

$$T_{new} = \left\lceil \frac{ff - ff_{dpic}}{N} \right\rceil (p_{con} + 1) + p_{con} + \sum_{i=1}^{n_c} (a_i \delta_i + b_i) \qquad (5.11)$$

Our approach will be faster than full-scan if $T_{fs} > T_{new}$. Thus:

$$\left\lceil \frac{ff}{N} \right\rceil (p_{fs} + 1) + (p_{fs} - p_{con}) - \left\lceil \frac{ff - ff_{dpic}}{N} \right\rceil (p_{con} + 1) > \sum_{i=1}^{n_c} (a_i \delta_i + b_i) \qquad (5.12)$$

Both coefficients on the right side of inequality *(5.12)* are known from the library of the components used during high-level design. Hence, one may easily calculate the test time that our approach will consume during the execution of the gate-level test patterns for the datapath before the actual synthesis. Of course, other parameters from the left side of equation will become known after the actual gate-level implementation and after the user inserts the scan-chains.

==

Example

Consider the instantiation of the "Compress" application given in Figure 4.21. Before, the gate-level implementation, the following parameters are available due to the test constraint insertion during the high-level of the codesign, as explained in the previous chapter:

$a_{ALU} = a_{COM} = a_{LOG} = a_{RF1} = a_{RF3} = 5$, $b_{ALU} = b_{COM} = b_{LOG} = 2$, $b_{RF1} = b_{RF3} = 1$, $n_c = 5$, $\delta_{ALU} = 69$, $\delta_{COM} = 70$, $\delta_{LOG} = 10$, $\delta_{RF1} = 72$, $\delta_{RF2} = 96$.

Therefore, the right-hand side of inequality (5.12) is calculated as:

$$\sum_{i=1}^{n_c}(a_i\delta_i + b_i) = 1593. \qquad (5.13)$$

After the synthesis, the ff and ff_{dpic} parameters become known and they are 1418 and 598, respectively. Let us assume that the designer has opted to insert 5 scan-chains within the complete processor. First, the ATPG is run on the complete processor with 5 scan chains has been done and it resulted in $p_{fs} = 360$ full-scan test patterns. Next, certain ports of the control unit have been defined as "skipped" and ATPG has been applied at the control block only. 202 test patterns for the top level of the control unit have been obtained in that way. Therefore the left-hand side of inequality (5.12) equals :

$$\left\lceil\frac{ff}{N}\right\rceil(p_{fs}+1)+(p_{fs}-p_{con})-\left\lceil\frac{ff-ff_{dpic}}{N}\right\rceil(p_{con}+1)=$$
$$\left\lceil\frac{1418}{5}\right\rceil(360+1)+(360-202)-\left\lceil\frac{1418-598}{5}\right\rceil(202+1)=102253-33494=68759 \qquad (5.14)$$

The inequality *(5.12)* enables the user to compare the two test styles in terms of the speed. It is obvious that our approach in the previous example significantly reduces the number of test cycles compared to conventional full-scan. It is also obvious that the proposed functional-structural test method requires less DfT and has lesser impact on the overall performance. However, full-scan still beats our approach in terms of the fault coverage, as the next section will reveal.

5.5. Results and Discussion

The test implementation at the gate-level using the proposed functional-structural test style will be illustrated by the examples already used in the previous chapter. The architectural-level descriptions have already been given in the Figures 4.20–4.22. Therefore, the VHDL behavioural-level source codes of the "Crypt", "Compress" and "Eight-Queens" applications have been generated according to the high-level synthesis within the MOVE codesign kit. Of course, the test constraint in terms of the test cost has been taken into account during the selection of the resulting instantiation, referring to the procedure described in the previous chapter. Table IX summarizes the architecture of the three designs.

Conceptually, the control unit of all three designs is the same, having the form as given at the beginning of this chapter. The control-block parameters of the first two applications are chosen to be the same; hence, the embedded instruction cache has size of 48 × 16 bits (i.e., the length of VLIW word is 48, containing 16 locations), exception handler, TLB, guard parameters, number of bits at the control bus that have been used and other parameters are as defined in [VAR00]. The datapath elements are however different. The datapath of the "Crypt" application consists of ALU, Adder and Compare functional units followed with one register file with one input and two outputs containing 12 registers inside. The data bus width is 16 bits. The "Compress" application has the datapath

consisting of an ALU, Compare and Logic unit. Two register files have been used in the design, both with one input and two outputs (containing 8 and 15 registers) while the data bus is also 16-bits wide. The control block of the "Eight-Queen example" is also similar to the previous two, with the distinction that the cache is twice the original size. The datapath is consisting of two ALU's: ALU1 is implemented according to Figure 5.11 while ALU2 has capabilities of sign extension of the input operands besides arithmetic and logical operations. In addition, the design has Adder/subtractor, Shifter, Logic unit, Compare unit, Multiplier, and the two register files (both with one input and two outputs with 12 and 32 registers, respectively).

TABLE IX THE ARCHITECTURAL-LEVEL PARAMETERS OF THE THREE TTA APPLICATIONS

Application	Datapath Components	Data bus width	Control bus width	# busses	Cache size
Crypt	ALU1, ADD, COMP, REG1[11:0]	16	12	3	48×16
Compress	ALU1, LOG, COMP REG2[7:0], REG3[15:0]	16	16	3	48×16
8-Queens	ALU1, ALU2, LOG, COMP, ADD/SUB, SHF, MUL, REG1[11:0], REG4[31:0]	32	24	10	48×32

The implementation is 32–bit like, i.e., the data bus is 32-bits wide. Due to the already described implementation limitations of the current Load/Store and MIU units, the test-scheduling algorithm to save the test time could not be employed. For the same reason, Load and Store operations could not be scheduled within the same timing slot.

The parameters concerning the test of the datapath components in a functional-structural fashion, L_{op} and δ, are given in Table X. These parameters are assigned to the library of the components used during the high-level synthesis.

TABLE X THE PARAMETERS CONCERNING THE FUNCTIONAL-STRUCTURAL TEST OF THE DATAPATH COMPONENTS

Component	δ	L_{op}
Adder	7	5
Adder/Subtractor	8	5
ALU (type 1)	69	5
ALU (type 2)	98	5
Logic Unit	10	5
Shifter	6	5
Compare	70	5
Multiplier	56	6
Reg file1 [11:0]	72	5
Reg file2 [7:0]	48	5
Reg file3 [15:0]	96	5
Reg file4 [31:0]	192	5

All components but the multiplier have the same latency equal to 1 clock cycle. The multiplier that has been built as a cellular iterative array and has a latency of 2 clock cycles since it was difficult to achieve the necessary throughput without adding an internal pipeline stage inside the multiplier. The Load and Store operations have a latency of two clock cycles.

Next, the already explained DfT modifications had to be introduced in the RTL code and the complete VLIW-TTA processor is then synthesized using the *Synopsys Design Compiler.* The processors have been designed to work with a power supply of 2.7-3.3 V (depending on the operating conditions) at an operating frequency of 100 MHz with an accepted clock skew of 0.3 ns. All implementation details of the three TTA design are summarized in the Table XI. The complexity of the circuit is expressed as the number of *NOR* equivalent gates, i.e., the circuit area divided by the area for a 2-input NOR gate.

TABLE XI SYNTHESIS RESULTS OF THE THREE TTA APPLICATIONS

Application	Complexity [# of 2-input NOR gates]	Area [μm^2] (0.35µ technology)	Maximum throughput [ns]	Power [mW]
Crypt	15115	851266	9.14	132.5
Compress	21828	110457	9.78	169.7
8-Queens	32508	181589	9.89	176.7

After the synthesis, the already mentioned industrial in-house CAT tools have been used for scan chain insertion and ATPG. The following design parameters concerning the test are then obtained and given in Table XII.

TABLE XII THE RELEVANT PARAMETERS OF THE FUNCTIONAL-STRUCTURAL TEST

Application	**# ff**	**# ff_{dpic}**	**# p_{fs}**	**# p_{con}**	**N**
Crypt	1059	437	284	182	6
Compress	1418	598	360	205	8
8-Queens	2167	948	350	245	11

The optimal number and assignments of scan chains that will be used, depends on the various tradeoffs in the design process. There are several factors that play a role in this trade-off such as e.g. the scan-chain length (determining the number of scan cycles), and the number of additional pins for scan-in/scan-out/scan-enable purposes. The problem itself is *NP-complete* and often application driven [VAR98]. In the references [AER98] and [MAR00] research is carried out with regard to the optimal scan-chain insertion in order to achieve the minimal test time. Table XII shows the number of resulting scan-chains within the three applications taking into account the suggestion in [VAR98] and partially, the algorithms proposed in [AER98] and [MAR00]. For example, the scan-chain insertion within the control block for the "Crypt" application has ended up with 6 scan-chains with 104 and 103 flip-flops in length, after scan-chain balancing.

Table XIII shows the overall results of our test approach for the three applications while the table XIV refers to the results obtained with full-scan. The assumption is that the available number of scan-chains remains the same in both situations.

TABLE XIII THE OVERALL TEST RESULTS AT THE GATE-LEVEL USING OUR APPROACH

Application	# test cycles	Fault coverage (%)	Additional DfT area (%)	Throughput penalty (%)
Crypt	20412	98.02	8.64	2.98
Compress	21534	98.15	8.57	2.93
Eight Queens	32486	98.7	7.98	2.84

The tables reveal that our approach achieves significant savings in the number of the test cycles compared to the full-scan method due to the application of the functional-structural test instead of the full-scan in the datapath part of the processor. The full-scan has higher fault coverage than our approach due to the already explained reasons (related to socket-test). However, the difference is not significant. The fault simulation of the complete VLIW-TTA processor has indeed shown that a number of faults within the sockets will be detected if, besides the component under test (CUT), another functional unit becomes active due to the presence of faults within its sockets. These faults may activate the two functional units/register files forcing different values on the bus (if connected to the same bus) and hence, these are very likely to be detected.

TABLE XIV THE OVERALL TEST RESULTS USING THE FULL-SCAN

Application	# test cycles	Fault coverage (%)	Additional DfT area (%)	Throughput penalty (%)
Crypt	50268	99.37	15.00	3.1
Compress	57200	99.44	15.52	3.16
Eight Queens	88562	99.78	16.76	2.89

The fault coverage in the Tables XIII and XIV refers to the number of detected stuck-at faults. The bridging faults will be detected using IDDQ test patterns during the test of the manufactured chip. For that purpose, the outputs of the design registers need to be set in high impedance state using the *tritest* signal (see Figure 5.10), to keep busses cleared off. The numbers of IDDQ test patterns for “Crypt”, “Compress” and “Eight Queens” have been 37, 54 and 96, respectively, and they all exceeded 99 % of fault coverage.

The additional area and throughput characteristics are both in favour of our approach. They both result from the scan-chain insertion in the control block only and that change has then been reflected at the top level of the processor. If high-fault coverage is the primary goal, but also the testing time, a trade-off is still possible as reported in [ZIV00]. In that case, scan chain(s) may be inserted into the socket registers, independently of scan chains within the control-block. The scan chains within the sockets will apply the test patterns that target the faults within the socket circuits only. After that, the functional-structural test will take place to test the datapath components. In that way, the same fault coverage as reported in the case of the full-scan test of the whole processor can be obtained, while still keeping the number of test-cycles lower than compared to full-scan. Of course, the area and throughput penalty are the same as in the case of a full-scan approach.

Looking at the tables XIII and XIV, one may conclude that as the VLIW-TTA application design gets bigger and more complex, the difference between full-scan and our proposed method becomes more expressed in the favour of our approach:

- The decrease in the ratio with respect to the number of test cycles becomes larger
- The fault coverage difference becomes less expressed
- The additional DfT hardware and throughput penalty decreases
- The throughput penalty is always lower.

These advantages are due to the application of the functional-structural test. Namely, a more complex application has also a larger datapath which is tested with pre-determined test patterns with minimum DfT, while the control block does not change that much, as compared to the datapath and interconnection network.

The presented functional-structural approach may also be regarded as a kind of deterministic soft logic BIST approach, as mentioned in the second chapter. However, there exists a substantial difference: it is difficult to find a soft-BIST that will remain general for an arbitrary circuit. For example, the so-called ABIST as introduced in [RAJ98] is largely dependent on the circuit structure and hence, it is difficult to predict the overall test characteristics at the gate-level before the actual gate-level implementation. Our approach remains generally applicable for each VLIW-TTA implementation offering the designer the possibility to have an insight into the test implications at a high-level of the design.

For obvious reasons, BIST implementations employing test-points insertion have not been taken into account, as well as implementations with pseudo-random pattern-generators due to the low fault coverage. Other advanced BIST schemes, such as [KIE98] or [KIE00] described in the second chapter, may be integrated into our approach as the patterns can be applied and read internally on the chip or used as canned IP test patterns in the core-based design. Note that the order of tests is important for VLIW-TTA, i.e. it is necessary to perform the interconnect test of the sockets and busses before carrying out the functional test of the components. In that sense, our approach has also some similarities with the core-based test strategy, where the test consists of two steps: interconnect and IP test, as presented in the preceding chapters. The functional-structural test of the TTA may be regarded as an IP test in the Core-based test when predefined patterns are applied to test the embedded cores with a properly configured surrounding infrastructure.

The complete CAD/CAT flow for design and test of VLIW-TTA processors up to the gate-level implementation has the form as drawn in Figure 5.20. It is the continuation of the flow presented and described in the previous chapter (Figure 4.15) that resulted in the architectural description of a particular TTA instantiation. The tool MPG has been used to translate the architectural level into an RTL behavioral VHDL description. Necessary DfT (and not only DfT) modifications had to be carried out in order to make VHDL code first synthetisable and subsequently, gate-level DfT insertion and test pattern generation feasible. According to the proposed "Divide and Conquer" test strategy, the scan-chain insertion and ATPG for the control block has been done first. The embedded-memory test has also been generated and combined with the control-block full-scan test patterns. The datapath of the processor has been tested with the functional-structural test. The ATPG took into account the test constraint introduced at the high-level phase of the design.

Hence, the pre-generated gate-level test patterns of the datapath components had to be applied according to the protocol driven by the processor assembler. The test patterns are applied by setting the corresponding sockets into an appropriate state.

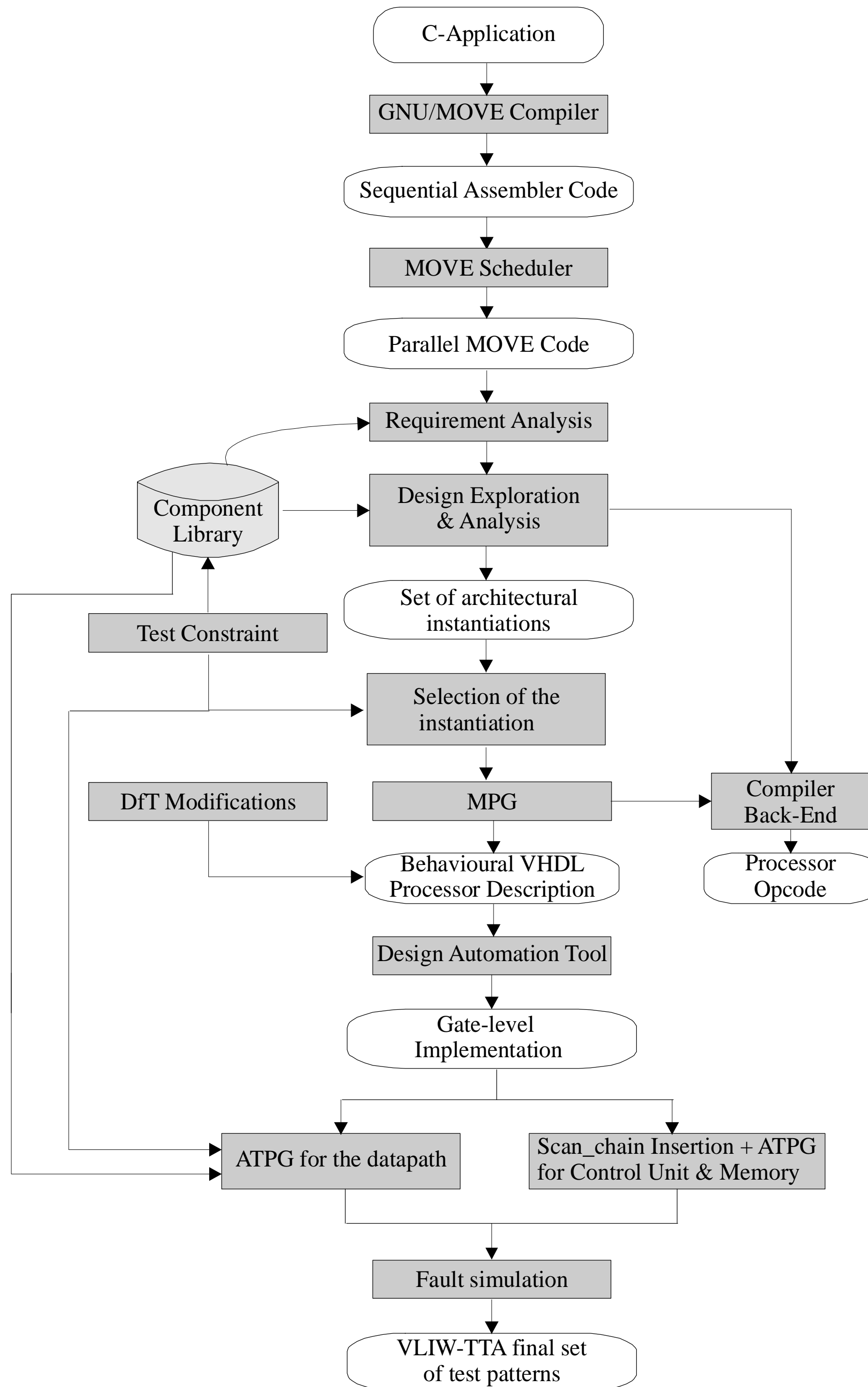


Figure 5.20. The complete MOVE codesign & test flow

Both sets of test patterns, i.e., the full-scan and memory test for the control block and functional-structural test patterns of the datapath have been fed into the fault simulator in

order to obtain the real fault coverage. The fault simulator used in the examples was Cadence Verifault XL 1.7.1. The fault simulation of the three circuits occupied a significant amount of CPU time – it took 12.5, 18 and 36 hours for the Crypt, Compress and Eight-Queens application, respectively. The examples were run on UNIX operating system HP-UX 10.3, a Hewlett Packard HP 9000-785/J 5000 workstation (processor type 2 × PA-8500, 440 MHz + 512 Mb RAM).

The overall flow requires various tools. Wherever necessary, a script written in *Gawk* has been employed in order to convert the output files from one tool to another format acceptable as an input for the succeeding tool. Hence, the flow is now almost completely automated.

5.6. Conclusion

An approach of implementing the test at the gate level of a VLIW-TTA processor has been proposed. A test method has been derived referring to the results of the test synthesis of the TTA processor at a high-level. The overall codesign flow has been modified to enable straightforward DfT at gate level. The test strategy fully exploits the regularity and modularity of the VLIW-TTA structure and remains general for an arbitrary application and instantiation of the TTA processor. It is a functional test based on the structural test patterns of the components from the datapath obtained by a conventional ATPG tool. It uses minimal additional hardware inside the datapath and interconnection network. Consequently, the area/throughput ratio from the high-level design space exploration phase is hardly influenced, since the datapath and interconnection network are the parameterisable parts of the processor defined by the application and instantiation. The control unit of the processor has been tested with the full-scan approach combined with a standard memory test applied for the embedded instruction cache. Experiments have been performed starting from the behavioral VHDL description of the “Crypt”, “Compress” and “Eight-Queens” examples. These applications, described initially in the *C*-programming language were obtained using the MOVE codesign tool with the high-level test approach as described in Chapter 4. The various TTA processors have been synthesized using the Synopsys Design Compiler while an industrial CAT tool has been used as an ATPG tool and to insert scan-chains. The results using the proposed approach have indicated the superiority over a full-scan approach with respect to the number of the test cycles, DfT overhead and throughput penalty. The fault coverage is somewhat lower than in the case of full-scan test, due to undetected faults in the interconnection network. However, the difference was almost neglectable. The future research has to focus on additional, probably functional test vectors to be applied for to test the sockets in the interconnection network, without influencing the area and throughput. Also, the integration of our approach into advanced BIST architectures, such as [KIE00] will be a challenge.

References:

[AER98] J. Aerts, E. J. Marinissen, "Scan Chain Design for Test Time Reduction in Core-Based ICs," *Proc. of the International Test Conference,* October 1998, Washington D.C., pp. 448-457.

[BEE98] J. van Beers, H. van Herten, "Test Features of a Core-Based Co-Processor Array for Video Applications," *Proc. of the International Test Conference,* October 1998, Washington D.C., pp. 638-647.

[COR98] H. Corporaal, "*Microprocessor Architectures from VLIW to TTA,*" ISBN 0-471-97157-X, John Wiley, 1998.

[GOO91] A. J. van de Goor, "Testing of the Semiconductor Memories", Kluwer Publishers, 1991.

[HAM99] S. Hamdioui, A. J. van de Goor, "Port Interference Faults in Two-Port Memories," *Proc. of the IEEE International Test Conference,* September 1999, Atlantic City, NJ, pp. 1001-1010.

[JAN95] J. Janssen, H. Corporaal, "Partitioned Register Files for TTAs," *Proc. of the 28th Annual International Workshop on Microprogramming* (Ann Arbor, Michigan, November 1995).

[KIE99] G. Kiefer, H.-J. Wunderlich, "Deterministic BIST with Partial Scan," Proc. of the European Test Workshop, IEEE, 1999, pp. 110-116.

[KIE00] G. Kiefer, H. Vranken, E. J. Marinissen, H.-J. Wunderlich, "Application of Deterministic Logic BIST on Industrial Circuits," *Proc. of the International Test Conference,* October 2000, Atlantic City, NJ, pp. 105-114.

[MAR00] E.J. Marinissen, S. K. Goel, M. Lousberg, "Wrapper Design for Embedded Core Test," *Proc. of the International Test Conference,* October 2000, Atlantic City, NJ, pp. 911-920.

[SAC97] M. Sachdev, "Open Defects in CMOS RAM Address Decoders," *IEEE Design & Test of Computers*, June 1997.

[SMI98] D. J. Smith, *HDL Chip Design*, Doone publications, 1998.

[RAJ98] J. Rajski, J. Tyszer, *Arithmetic Built-In Self-Test*, Prentice-Hall PTR, 1998.

[VAR98] Various authors, "*Core-based Test: Cookbook*", Philips Internal publication, 1998.

[VAR00] Various authors, "The MPG manual", Documentation of the Laboratory of Computer Architecture and Digital Techniques (CARDIT), Faculty of Electrical Engineering, Technical University of Delft, 2000.

[ZIV00] V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff, "Computer-Aided Test Flow in Core-Based Design," *Elsevier Journal of Microelectronics*, vol. 31, issue 11/12 November-December 2000, ISSN 0026-2692/00, pp. 999-1008.

[ZIV01a] V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff, "An Implementation for Test-Time Reduction in VLIW Transport-Triggered Architectures," *Proc. of the IEEE European Test Workshop (ETW 01)*, May-June 2001, Stockholm, Sweden, pp. 255-262.

[ZIV01b] V. A. Zivkovic, R. J. W. T. Tangelder, H. G. Kerkhoff, "An Implementation for Test-Time Reduction in VLIW Transport-Triggered Architectures," preselected for publishing in Journal of Electronic Testing – Theory and Applications (JETTA), 2001.

CHAPTER 6

Conclusions

6.1. Motivation

The research work presented in this thesis deals with the test constraint in hardware/software codesign of Very Long Instruction Word (VLIW) Application Specific Instruction Processors (ASIP). The test issue, becoming a dominant cost factor, is very important during the high-volume production of modern ICs organized as complex Systems-on-Chip (SoC). The initial point of research has considered testability of hardware/software codesign systems as an item that need to be tackled in the early phase of codesign. The research target and ultimate goal has been the generation of the efficient set of test vectors that can be used during the production test of manufactured VLIW ASIP devices. While dealing with test constraints at high-level, another condition had to be fulfilled: minimal impact of the implemented test strategy on the overall performance of the circuit. In addition the test method had to be consistent with current state-of-the-art in digital IC design and test.

6.2. Summary

Chapter 1 contains the introduction to the hardware/software codesign of ASIP-VLIW processors, motivation and the objectives of this thesis. The ultimate goal is to develop a

generic method towards the implementation of test that targets the faults that may occur during the chip manufacturing of the VLIW ASIP HW/SW codesign product.

In *Chapter 2*, emerging trends, problems and ways of development of modern digital test techniques such as e.g. Core-based test, being applicable in complex SoC design, have been introduced. The concept of so-called test cost has been tackled. The test cost represents a measure of efficiency of the test vectors that are used to test the SoC. The factors that determine the overall cost of test such as fault coverage, test time, DfT area overhead, throughput penalty, test-development time, number of pins used for test and Automatic Test Equipment (ATE) usage have also been addressed. These factors have been taken into account while developing the test strategy for a certain family of VLIW-Application Specific Instruction Processors (ASIP).

Chapter 3 addresses the Core-Based test development in Peripheral Interconnect (PI) - bus based design. This chapter is the product of the traineeship carried out within the Philips Semiconductors ASG ESTC group in Eindhoven. The core test methodology used for the test pattern generation and realistic fault coverage calculation in embedded core-based design has been explained. The so-called test shell is introduced providing the infrastructure for the test and isolation of embedded cores. The test of the embedded core consists of the test of the inner part of the core (IP test) and the test of its interconnections together with the test shell. The problem of getting the accurate fault coverage due to the introduction of the test shell is explained. The analysis of the DfT structures used within the test shell in core-based design has been carried out. An automatic test-pattern generation flow according to the core test methodology that results in accurate fault coverage of embedded cores has been proposed and implemented. The proposed flow has been used in the Philips pilot IC project.

Chapter 4 deals with the original test constraint insertion during the high-level phase of the design of a particular type of VLIW ASIP called MOVE and CASTLE. The MOVE codesign package generates the so-called Transport-Triggered Architectures starting from the *C*-application. Our approach that assesses the instantiation of the architecture with respect to the area, throughput and test has been explained. It is based on a functional test with structurally generated test patterns. The test cost has been introduced, for the sake of analytical calculation of the testability at high level of the design. A heuristic test-scheduling algorithm to further optimize the test time has also been explained. Our approach has been captured in procedures written in the *C* programming language and integrated within the MOVE codesign package using *Gawk* script files. The method has been validated with respect to the example of "Crypt", “Compress” and “Eight-Queens” applications. A similar test-constraint has been added into the CASTLE codesign tool during the design space exploration, enabling the designer to estimate different instantiations of the application. The test method is arbitrary and is validated using the slightly modified functional-structural test in a video-processor example.

In *Chapter 5*, an original method of integrating design and test flow has been described. Also, an efficient test at the gate level of a VLIW-TTA processor has been implemented. The control unit of the processor has been tested with the full-scan approach combined with a standard memory test applied for the embedded instruction cache while the functional-structural test has been used to test datapath components. Implementation details of control blocks and interconnection network relevant for traversing the test

stimuli and responses have been clarified. The application of a functional-structural test has been illustrated on the example of an ALU unit within the datapath of the TTA. Experiments have been performed starting from the behavioral VHDL description of the "Crypt", "Compress" and "Eight-Queens" examples. These applications, described initially in the *C*-programming language, were obtained using the MOVE codesign tool with the high-level test approach as described in Chapter 4. The various TTA processors have been synthesized using the *Synopsys* Design Compiler, while an industrial CAT tool has been used as an ATPG tool and to insert scan-chains. The test-time analysis, in order to compare our approach with the conventional full-scan method, has also been presented in this chapter.

6.3. Conclusion

A new automatic test-pattern generation flow according to the Core-based test strategy has been proposed and implemented and has been used in the Philips pilot IC project. It results in an efficient test pattern set and accurate fault coverage of the core that can be used in an arbitrary environment. The achieved results with respect to the number of test patterns and the fault coverage were better compared to other designs with similar complexity and functionality. Also, the total DfT area overhead was quite acceptable. Another important benefit of the proposed core-test flow is the clear separation of responsibility between core provider and core user with respect to test. The experience gained while developing this flow has been used to derive the original test strategy for the VLIW ASIP called MOVE and CASTLE. The test constraint is added at the high-level phase of the design and is based on a functional test with structurally generated test patterns. The method does not degrade the already achieved area/execution-time ratio obtained after the high-level design space exploration. The fact that allows one to minimize the extra circuitry is the regularity of the MOVE-TTA structure, i.e. the direct accessibility of the components in the datapath via busses. The analytical test cost functions for the functional test of the components are derived according to the transport-timing relations and they are dependent on the architectural parameters only. The examples, performed on UNIX Crypt, Compress and Eight-Queens applications, have also pointed out that the test costs may vary quite significantly, even for architectures that have similar characteristics with respect to the area and throughput. That is a strong argument in favour of applying the high-level test synthesis in this environment, before the actual hardware implementation. Similar as it has been done in MOVE, the test constraint is added during the high-level design phase inside the CASTLE codesign tool. In both cases, the proposed method does not introduce any DfT circuitry within the datapath. However, the method can achieve a high-fault coverage of the components used in the datapath. Furthermore, the method has proven to be highly efficient with respect to the number of test vectors that have to be applied in order to test the components as confirmed after the logic synthesis.

The test implementation at gate-level of the MOVE processor has been derived referring to the results of the test synthesis of the TTA processor at a high-level. The overall codesign flow has been modified to enable the straightforward DfT at gate level. The test strategy fully exploits the regularity and modularity of the VLIW-TTA structure and remains general for an arbitrary application and instantiation of the TTA processor. It is derived referring to the high-level test constraint introduced in Chapter 4. It uses minimal additional hardware inside the datapath and interconnection network.

Consequently, the area/execution-time ratio from the high-level design space exploration phase is hardly influenced. This, because the datapath and interconnection network are parameterisable parts of the processor defined by the application and instantiation. The results using the proposed approach have indicated the superiority over a full-scan approach with respect to the number of test cycles, DfT overhead and throughput penalty. The fault coverage is somewhat lower than in the case of full-scan test, due to undetected faults in the interconnection network. However, the difference was almost neglectable, thus justifying the application of our approach to test the VLIW-TTA processor.

6.4. Original contributions and recommendations for future research

The thesis contains several original contributions that may be summarized as follows:

- A new automatic test-pattern generation flow according to the core test strategy has been proposed and implemented. It results in an efficient test pattern set and accurate fault coverage of the core that can be used in an arbitrary embedded environment.
- The test constraint insertion during the high-level phase of the design within the MOVE and CASTLE hardware/software codesign systems based on the analytical test cost function has been carried out. The test cost function only depends on the architectural parameters.
- The functional-structural test style, derived according to the test cost function, to tackle the VLIW-ASIP test in a most efficient way. The test is structurally based, but performed in a functional way. Applying our method, one saves the time required to apply the test patterns, keep the DfT area and throughput penalty low, while still maintaining an acceptably high fault coverage. In addition, our method is independent with regard to the test stimuli application. It can be applied using external ATE equipment or BIST or a combination of both. Hence, the generated set of the test patterns can be used in core-based testing.
- An integrated design and test flow within the MOVE codesign package resulting in an efficient set of test patterns to test a VLIW-TTA processor has been developed. The overall flow requires the use of various tools. The flow is almost completely automated.

The future research in the field of hardware/software codesign test of VLIW-ASIP has to focus on several subjects listed below:

- It has already been explained that the full-scan method still beats our approach in terms of the fault coverage as the structural test-patterns do not target the faults that may exist within the sockets. If the acceptable fault coverage of non-redundant faults is 100 %, additional test vectors have to be applied. Hence, research is required to generate these additional test vectors for sockets in the interconnection network, without influencing the area and throughput.
- To address this problem, it would be interesting to provide testability-enhancement for the data-path, and integrate the data-path DfT methods into the design space exploration process. This reduction of fault coverage may also be due to the fact

that the three parts of a design (the datapath, controller and memory) are extensively tested, while their interactions (communication and synchronization) are not. Hence, it is useful to incorporate a test strategy for these interactions in order to improve the fault coverage.

- The integration of our approach into advanced BIST architectures, such as Logic BIST will be a challenge.
- The study of delay faults that may appear in a VLIW-ASIP and generation of the appropriate test set for their detection. The study performed in this thesis may be a good basis in that direction. Next a research is required with respect to the influence of other types of fault models in addition to stuck-at in a VLIW-ASIP environment.
- Extension of our method to test other existing VLIW processors of a similar bus-oriented structure.