# Test Generation for Gigahertz Processors Using an Automatic Functional Constraint Extractor*

Raghuram S. Tupuri
Texas Microprocessor Division
Advanced Micro Devices
Austin Texas 78741
raghuram.tupuri@amd.com

Arun Krishnamachary and Jacob A. Abraham
Computer Engineering Research Center
The University of Texas at Austin
Austin Texas 78712
arun, jaa@cerc.utexas.edu

## Abstract

*As the sizes of general and special purpose processors increase rapidly, generating high quality manufacturing tests which can be run at native speeds is becoming a serious problem. One solution is a novel method for functional test generation in which a transformed module is built manually, and which embodies functional constraints described using virtual logic. Test generation is then performed on the transformed module using commercial tools and the transformed module patterns are translated back to the processor level. However, the technique is useful only if the virtual logic can be generated automatically. This paper describes an automatic functional constraint extraction algorithm and a procedure to build the transformed module. We describe the tool, FALCON, used to extract the functional constraints of a given embedded module from a Verilog RTL model. The constraint extraction for embedded modules of benchmark processors using FALCON takes only a few seconds. We show that this method can generate functional patterns in a time several orders of magnitude less than one using a conventional, flat view of the circuit.*

## 1 Introduction

The trend in processor architectures is towards the use of super-pipelining and super-scalar operation to increase the clock frequency and to reduce the number of clocks per instruction. This translates to circuits with increased sequential depth and many internal interacting state machines, both of which tend to make testing more difficult. Automatic test generation tools are generally geared to models at the gate level. However, gate level test generation for these processors will be impossible without any significant design for testability incorporated into the designs. The effort among researchers has, therefore, been to develop techniques to add testability to the design, with scan being the most commonly used technique. The scan features also help with debugging the manufactured chips.

In deep sub-micron process technologies, faults other than stuck-at faults, such as bridging faults and delay faults, are also becoming dominant. Experimental data presented in [1] show that the defect level of at-speed functional vectors with 75% stuck-at fault coverage is the same as the defect level of scan vectors with a higher fault coverage. Therefore, even when chips are designed with scannable memory elements, at-speed functional tests are used to reduce the defect rate. Unfortunately, we will not be able to achieve desired fault coverage levels for at-speed vector generation on complete processors by using current sequential ATPG tools [2]. Manual generation of functional vectors is an alternative, but the process is tedious and time consuming if we want to achieve high fault coverage for complex processors.

There has been some prior work in generating functional tests targeting structural, stuck-at faults in processors. Lee and Patel [3] describe a hierarchical level test generation using a two step process. First, a modified gate level PODEM algorithm is used to generate module level test vectors and the module level vectors are justified and propagated using an architectural level test generator. During the architectural level vector generation conflicts are handled using heuristics, rather than generating a new vector at the module level. A synthesis based ATPG method was proposed in [4]. In this method, an RTL model of a complete design is synthesized, and during the synthesis process, functional constraints are extracted and used to guide a custom

*This research effort was supported by the Semiconductor Research Corporation under contract 98-DJ-483.


DAC 99, New Orleans, Louisiana

ATPG tool which is tightly coupled with the synthesis tool. Automatic functional constraint extraction from VHDL RTL using a tool called ATKET was proposed in [5]. ATKET extracts propagation and justification modes through a given embedded module, and provides them to a custom test generator. These approaches have several drawbacks. The test generation system must be custom designed, it needs to deal with the complete chip model, it is not possible to synthesize a modern design in a single piece, not all blocks in a design are synthesized, and the storage requirements for the constraints grow rapidly with the size of the circuit.

A novel functional test generation method which shows promise for large designs was published in [2]. This method targets one embedded module at a time and uses a commercial ATPG tool to derive tests for faults within the module. Functional constraints are extracted manually for inputs and outputs of the module. The extracted constraints are described in Verilog/VHDL and synthesized to the gate level. Then a transformed module is built using the synthesized virtual gates (these gates are not part of the design) and the embedded module, which is not changed. A commercial sequential ATPG tool is used to generate transformed module level vectors for faults within the module. Finally, these module level vectors are translated to processor level functional vectors. Although the approach shows promise, the manual generation of the functional constraints precludes its use in real applications. In addition, the reported times did not include the time needed to generate the constraints, so the comparisons with automatic test generation are not meaningful.

In this paper, we will describe an algorithm for automatic functional constraint extraction, called FALCON, and a method for automatically building the transformed module. An RTL description of the circuit is used to extract constraints for each module. During the analysis for constraints, testability bottlenecks can be identified and the design can be modified, if desired, early in the process. In particular, we are able to identify locations where accessibility of internal registers can be improved for testability (called PIERS in the figure). As the circuit is implemented, we automatically extract the constraints for each module and generate virtual logic for the constraints. As will be shown in this paper, the embedded module along with its constraints is of a size that can be handled by any commercial test generator tool.

The functional constraint extraction (FALCON) algorithm is explained in Section 2 and the implementation details are described in Section 3. Section 4 presents results on benchmark processors which include the times for extraction of constraints. Conclusions follow in Section 5.

# 2 Automatic Extraction of Module Constraints (FALCON)

A VLSI design (including micro-processors) can be represented as a set of interconnecting state-machines, organized as modules and controlling data flow between the datapath elements. Each module can have a combination of state-machine(s) and datapath element(s). When these modules are instantiated in the design, not all states of the module can be reachable. In this section, we will describe functional constraints and then describe a methodology for building an ATPG view of the module under test. An automatic functional constraint extraction algorithm is then presented, and we show that it will scale with design sizes, so that the process can be successfully applied to large designs.

## 2.1 Functional Constraints

Let a sequential circuit $C_i$ with corresponding state machine $S_i$ be instantiated in a design . The test generator has to use the state space, $S_i$, defined by the instantiating constraints and generate tests for all the detectable faults. Approaches which use patterns generated at the module level without these constraints will not be able to justify or propagate the vectors to the boundary of the full chip.

For example, a data cache block, designed for a high-end processor, $P_h$, is used in a low-end processor, $P_l$ to save the design time. This happens commonly to reduce both product development time and also to increase reliability since the block was already debugged and tested. To support multi-processing, the processor $P_h$ implements a MESI protocol [7] but the low end processor does not have the support for data coherency. A write-back protocol is used by the data cache. The state diagram of the MESI protocol is shown in Figure 1.

The data cache instantiated in the low-end processor, $P_l$, has some of its inputs tied and the state machine is modified as shown in the figure . The dotted lines indicate the unreachable functionality of the state machine. Functional test vectors that are generated for the high-end processor cannot be used to test the data cache instance in the low-end processor. For example, the transition from *exclusive* to *invalid* can be tested using a *Snoop Hit* in the high-end processor, but can only be tested using a *Flush* in the low-end processor. These are example functional constraints that are defined by the instantiation environment.

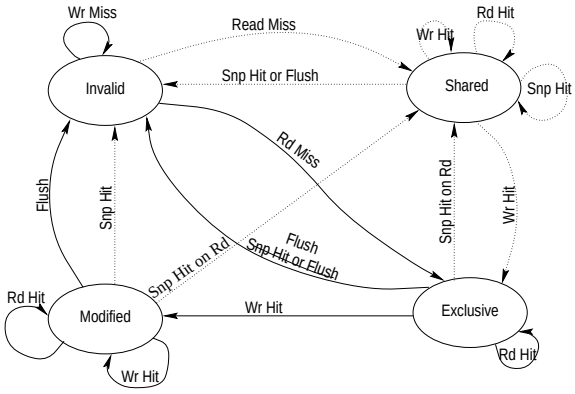


Figure 1: The MESI State Machine with No shared state and No Snoop

## 2.2 Primary Input/Output Accessible Registers (PIERs)

Processors and other complex VLSI designs generally have several internal registers accessible through primary inputs/outputs, i.e., using the processor instruction set (public or private instructions). The register file and configuration registers are examples of such registers. These registers can help in reducing the sequential depth of the transformed module and thus reduce ATPG difficulty. A **PIER** consists of one or more sequential elements, whose values can be initialized to a given value from the primary inputs and outputs.

A PIER is *orthogonal* if its contents cannot be disturbed by performing any operation other than writing into the PIER(eg. any register in the register file), it is *non-orthogonal* if it can be initialized but can hold the initialized contents only for a limited number of clock cycles (eg. program counter).

In general, PIERs are accessed using special register mechanisms in microprocessors. All accessible registers have an assigned address and during the special register operation this address is given along with a read or write request. In general the special register operations do not disturb the state of the processor. If necessary, even the program counter can also be maintained. The PIERs are defined during the architectural level specification and their access mechanisms are part of the specification.

## 2.3 ATPG View Abstraction

Because of design sizes, it is not possible to use an ATPG tool on the full chip without any design for testability. Our goal is to reduce the size of the ATPG view without affecting its ability to detect the faults in a given module. Using PIERs as the boundaries, we extract the transformed module which is given to the ATPG tool. It is also assumed that the design has a known initialization sequence since the design becomes untestable if there is no initialization sequence [8].

## 2.4 Transformed Module

We define the transformed module to be the ATPG view of the embedded module under test which includes its instantiating constraints. In order to explain the nature of the transformed module, let us consider a processor represented as a Moore machine, where there is no path from an input to an output which does not pass through a sequential element(the arguments can be easily extended to Mealy machines). Then the output function, $f_o$, depends only on the sequential elements, $V_p$. The next state logic ($f_p$) is a function of both the sequential elements in the design and the primary inputs.

The Moore machine can be divided into three interconnected state machines (Figure 2) with logic functions denoted by $f_i$ and state elements by $V_j$. The embedded module targeted for the test generation is part of $f_2, f_3$ and $f_o$. If the sequential elements in $V_1$ can be initialized from the primary inputs by using special register load procedures then, the function $f_1$ is independent of the sequential elements $V_2$ and $V_3$. Then we can eliminate the next state logic, $f_1$ from the transformed module. Similarly, if we can unload the contents of the sequential elements, $V_3$, then the output logic, that does not depend on $V_2$ can be eliminated from the transformed module. In Figure 2, the logic that is part of the transformed module is bounded by the dotted line. All the signals that are crossing the boundary will become inputs and outputs of the transformed module for the test generation phase.

Test generation on the transformed module(with PIERs and PI/POs as ports) ensures that we take into account the instantiating constraints of the module. The ability to load/unload the PIERs from the PI/POs easily, enable us to transform the tests that we generate (using a commercial ATPG tool on the transformed module) to the full processor level. We ensure good fault coverage by simulating the tests on all possible faults in the Module Under Test.

## 2.5 FALCON

As described above, constraint extraction serves to decrease ATPG difficulty by reducing the sequential depth and the size of the ATPG view. FALCON extracts propagation paths from the module either to PIERs or to primary outputs. Similarly, it extracts justification paths to the module inputs either from the PIERs or from the primary inputs. Once the constraints are extracted, they are written out in Verilog format. The

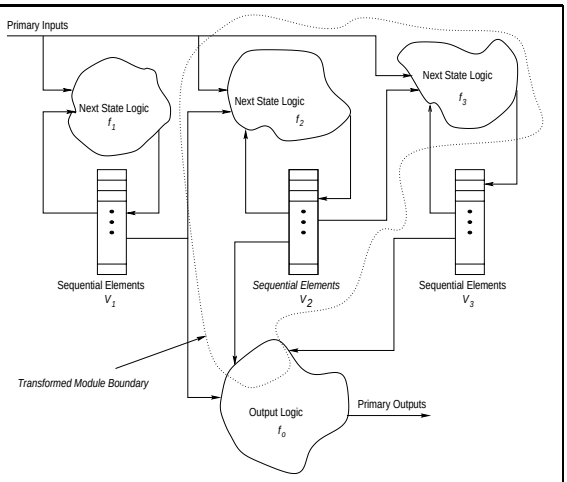


Figure 2: Sequential Circuit with Transformed Module Identified

constraint Verilog (output of FALCON) is then synthesized to generate *virtual logic*. A transformed module (which becomes the ATPG view of the module under test) is formed by combining the virtual logic and the module under test.

FALCON uses Verilog constructs to determine constraints. The algorithm is independent of processor architecture and can be applied to any design.

### 2.6 Reduction of ATPG Complexity

ATPG complexity is a function of the sequential depth, S, and the number of gates, G [6]. The sequential depth of a design is defined as the maximum number of sequential elements between its inputs and outputs. Let $S_p$ and $S_t$ represent the sequential depths of the complete design and the transformed module, respectively. Similarly, $G_p$ is the number of gates in the design and $G_t$ is the number of gates in the transformed module. If the design has no PIERs then $S_t = S_p$. Most designs have internal registers that are PIERs, and then the sequential depth of the transformed module is reduced by the PIER sequential depth, $k$, i.e., $S_t \leq S_p$. In addition to the sequential depth, the state search space is also reduced for the transformed module because of the reduction in the number of gates. In the worst case, the transformed module will be no worse than the original design i.e., this method does not make the problem worse.

## 3 Implementation of FALCON

FALCON is implemented in C++. The internal data structure of the connectivity tree is described in Figure 3. This data structure supports both gate level and RTL level Verilog and provides a convenient way to traverse the tree. The leaf nodes of the connectivity tree are either Verilog statements or library primitives.

During the extraction of constraints, a new connectivity tree is constructed which represents the constraints. The constraints are merged as we extract them from the processor connectivity tree into the constraint connectivity tree. Once the constraint connectivity tree is constructed, we can perform an optional constraint minimization to reduce the size of the ATPG view.

Table 1: Reduction of Sequential Depth

| Module Name | Trans. Module | Original Circuit | Red. % |
|---|---|---|---|
| Viper-ALU | 1 | 8 | 87.5 |
| DLX-Ex Stage | 1 | 8 | 87.5 |
| DLX-IF Control | 2 | 8 | 75 |
| DLX-WB Control | 2 | 8 | 75 |
| ARM-DP | 3 | 10 | 70 |

Table 2: Processor Level Test Generation

| Module Name | Fault Cov. | ATPG Eff. | Time (secs) | Coverage per sec |
|---|---|---|---|---|
| Viper-ALU | 77.17 | 80.84 | 74782 | .00103 |
| DLX-Ex Stage | 82.06 | 85.94 | 78852 | .00104 |
| DLX-IF Control | 38.99 | 50.58 | 53707 | .00072 |
| DLX-WB Control | 49.78 | 49.78 | 26877 | .00185 |
| ARM-DP | 17.66 | 17.66 | 316199 | .000055 |

## 4 Results

We have performed constraint extraction and building of the transformed module for several embedded blocks (Viper-ALU, DLX-IF Control, DLX-EX Control, DLX-WB Control) from the three benchmark processors, Viper, DLX and ARM. Viper is a simple processor, DLX is a RISC processor with a 5-stage pipeline, and ARM is a model of the commercial ARM-6 processor (but without the multiply unit).

FALCON was provided with the Verilog RTL models for functional constraint extraction. A transformed module was constructed from the extracted constraints by synthesizing the constraints to gates. The transformed module was used for test generation and the generated test patterns were translated to full chip level. Table 3 below describes the complexity of the modules that are used as test cases. Prior work [2] wrote the constraints manually and the indicated test generation times do not reflect the constraint extraction and synthesis times to create the transformed module.

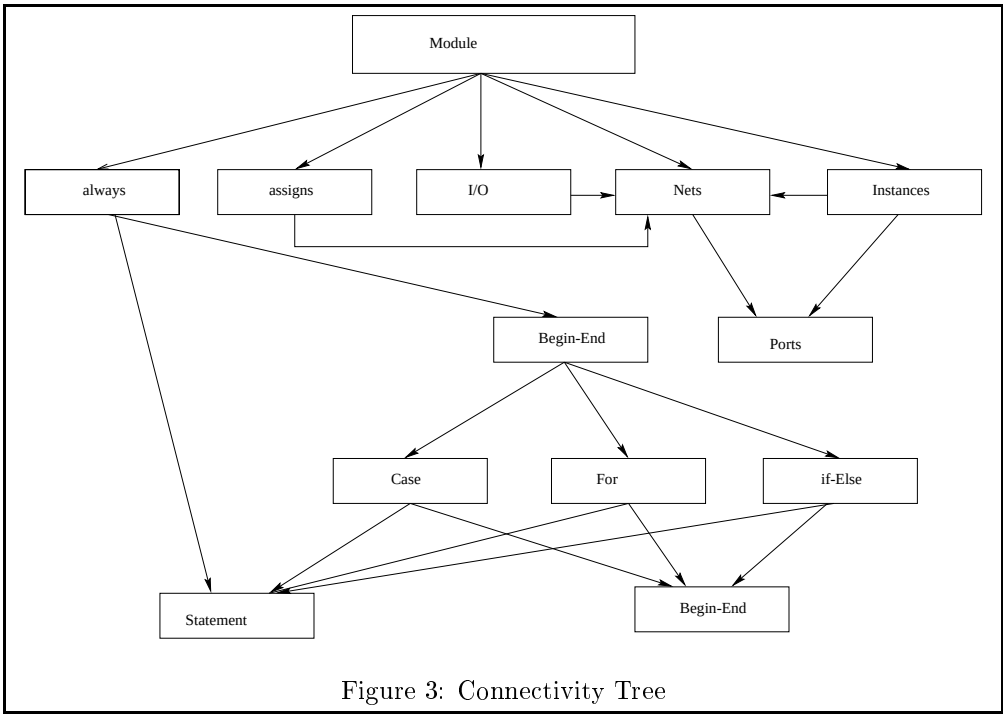


Figure 3: Connectivity Tree

Table 3: Processor/Module Details

| Processor/ Module | Combinational Gates | Sequential Elements | Primary Inputs | Primary Outputs | Stuck-At Faults |
|---|---|---|---|---|---|
| Viper | 5863 | 438 | 34 | 53 | 15358 |
| Viper-alu | 1784 | 104 | 72 | 34 | 6696 |
| DLX | 15177 | 1610 | 69 | 100 | 97556 |
| DLX-Ex Stage | 4776 | 64 | 77 | 32 | 28148 |
| DLX-IF Control | 1802 | 96 | 67 | 32 | 9114 |
| DLX-WB Control | 42 | 32 | 32 | 2 | 230 |
| ARM | 16029 | 1270 | 63 | 67 | 99198 |
| ARM-DP | 8893 | 295 | 199 | 161 | 51824 |

## 4.1 Extracting Functional Constraints

Table 4 presents the constraint extraction times (using FALCON) and synthesis times (using a commercial synthesis tool). Note that the time for this step is dominated by the synthesis time rather than the time for extraction. Also presented in this Table is the reduction in the number of gates seen by the ATPG. The smaller number of gates reduces the ATPG difficulty as discussed earlier. This reduction in ATPG complexity is reflected in the reduced ATPG run times.

## 4.2 Test Generation

To compare our method with conventional test generation, sequential ATPG is performed on the complete processor using the same commercial tool. Only the faults in the embedded module are targeted in each case. Table 1 shows the reduction in the sequential depth as seen by the test generator. In Table 5, test generation times using the proposed approach (including constraint extraction and synthesis times) are shown when using the same commercial test generator on the transformed modules. The proposed approach lowered the test time by several orders of magnitude, even when including the times for constraint extraction, synthesis and mapping back the tests for the transformed module to the original design. The reduction in the time for ATPG on the transformed module (Table 5, column 5) over the time for conventional ATPG (Table 2), and the improvement in the fault coverage indicate that the transformed module greatly reduces ATPG difficulty. Comparing the Tables 2 and 5 we see that we obtain several orders of magnitude improvement in the coverage per unit time

Table 4: Time for Building Transformed Module

| Module Name | Extraction Time (secs) | Synthesis Time (secs) | Original Gates | Gates in Virtual Logic | Reduction % |
|---|---|---|---|---|---|
| Viper-ALU | 1.27 | 143 | 4413 | 357 | 91.9 |
| DLX-Ex Stage | 1.33 | 488 | 11947 | 5160 | 56.8 |
| DLX-IF Control | 0.98 | 439 | 14889 | 1622 | 89.1 |
| DLX-WB Control | 0.84 | 128 | 16713 | 691 | 95.9 |
| ARM-DP | 13.84 | 642 | 8111 | 719 | 91.1 |

Table 5: Test Generation using Proposed Method

| Module Name | Fault Coverage | ATPG Efficiency | Test Gen. Time (secs) | Total Time (secs) | Coverage per sec. | Improvement factor |
|---|---|---|---|---|---|---|
| Viper-ALU | 92.81 | 99.07 | 81 | 225.27 | .41199 | 399.25 |
| DLX-Ex Stage | 96.25 | 99.68 | 85 | 574.33 | .16759 | 161.04 |
| DLX-IF Control | 88.41 | 97.38 | 1017 | 1456.98 | .06068 | 86.58 |
| DLX-WB Control | 97.17 | 100 | 2 | 130.84 | .74266 | 400.97 |
| ARM-DP | 81.14 | 99.42 | 3956 | 4612 | .01759 | 315.00 |

by using the proposed approach.

# 5 Conclusions

We have presented an automated technique for functional test generation using constraint extraction. This method reduces overall test development time dramatically and, at the same time, increases the test quality with no DFT overhead. The results presented in Table 4 indicate that the functional constraint extraction does not add a significant time to the test generation. These constraints reduce both the sequential depth and ATPG gates by a large amount. This, in turn, reduces test generation time by several orders of magnitude without affecting the test quality. The proposed approach decouples constraint extraction and ATPG. This allows us to use the best available ATPG tool without requiring any modification in the ATPG tool.

# 6 Acknowledgments

We would like to thank Dinos Moundanos of the University of Texas at Austin, for providing us a Verilog Parser, and Professor Trevor Mudge, University of Michigan, for the DLX and ARM models.

# References

[1] P. C.Maxwell et al., "The effect of different test sets on quality level prediction: When is 80% better than 90%?," *Proceedings of the International Test Conference*, October 1991, pp. 358–364.

[2] R. S. Tupuri and J. A. Abraham, "A Novel Functional Test Generation Method for Processors," *Proceedings of the International Test Conference*, November 1997, pp. 743–752.

[3] J.Lee and J.H.Patel, "ARTEST: An Architectural Level Test Generator for Data Path Faults and Control Faults," *Proceedings of the International Test Conference*, October 1991, pp. 729–739.

[4] R.S.Ramachandani and D.E.Thomas, "Behavioral Test Generation using Mixed Integer Non-linear Programming," *Proceedings of the International Test Conference*, October, 1994, pp. 221–229.

[5] P. Vishakantaiah, J. A. Abraham and M. Abadir, "Automatic Test Knowledge Extraction From VHDL (ATKET)," *29th ACM/IEEE Design Automation Conference*, April 1992, pp. 273–278.

[6] T. E. Marchok, A. El-Makeh, W. Maly and J. Rajski, "Complexity of sequential ATPG," *Proceedings of the European Design and Test Conference*, March 1995, pp. 252–261.

[7] D. Anderson and T. Shanley, "Pentium Processor System Architecture," *Addison-Wesley Publishing Company*, 1995.

[8] A. Miczo, "The sequential ATPG: A theoretical limit," *Proceedings of International Test Conference*, October 1983, pp. 143–147.