ACT: A DFT Tool for Self-Timed Circuits

Ajay Khoche
Sunrise Test Business Unit
Viewlogic Systems Inc.
47211 Lakeview Blvd
Fremont, CA - 94538, USA

Erik Brunvand
Department of Computer Science
University of Utah
Salt Lake City, UT - 84112, USA

Abstract

This paper presents a Design for Testability (DFT) tool called ACT (Asynchronous Circuit Testing) which uses a partial scan technique to make macro-module based self-timed circuits testable. The ACT tool is the first of its kind for testing macro-module based self-timed circuits. ACT modifies designs automatically to incorporate partial scan and provides a complete path from schematic capture to physical layout. It also has a test generation system to generate vectors for the testable design and to compute fault coverage of the generated tests. The test generation system includes a module for doing critical hazard free test generation using a new 6-valued algebra. ACT has been built around commercial tools from Viewlogic and Cascade. A Viewlogic schematic is used as the design entry point and Cascade tools are used for technology mapping.

1 Introduction

Testing asynchronous circuits is a relatively new area. Despite the growing number of recent efforts in the specification and design of asynchronous circuits, testing of these circuits has not been explored to any great degree. Very few efforts have been made in this area compared to efforts in synchronous testing [11, 13, 14, 18, 19, 20, 21, 22, 23, 27].

Traditionally, testing asynchronous circuits has been considered a difficult problem, especially when compared to synchronous circuits, where significant advances have been made. Unfortunately, methods used to test synchronous circuits are not directly applicable to asynchronous circuits. This is due, in large part, to the absence of the global clock signal in the asynchronous circuits. New methods are required to adapt the rich knowledge about testing synchronous circuits to test asynchronous circuits. This paper describes a tool called ACT which adapts a partial scan methodology to test a subclass of asynchronous circuits called self-timed circuits and generates tests for these circuits under this partial scan environment. It has also been integrated with the place and route tools from Cascade Design Automation to generate the integrated circuit layout. The ACT tool, to the best of our knowledge is first of its kind for testing macro-module based self-timed circuits, which provides a path from design entry through DFT, to layouts and test vectors.

The paper is organized as follows. In the next section, self-timed circuits will be described briefly along with the library module and synthesis method used in the ACT tool. Section 3 describes the fault model used and the rationale behind using it. Section 4 gives an overview of the tool and describes each of its components. Section 5 gives the results of applying this tool to example circuits. Section 6 offers some conclusions.

2 Self-timed Macro-module Circuits

Self-timed circuits are a subset of a broad class of asynchronous circuits which do not use a global clock for synchronization. Specifically, self-timed circuits are asynchronous circuits that generate completion signals to indicate that they are finished with their processing [24]. A signaling protocol used with the completion signal allows self-timed systems to be composed of circuits which communicate using self-timed protocols. Self-timed protocols are often defined in terms of a pair of signals, one to request or initiate an action, and another to acknowledge that the requested action has been completed. One module, the sender, sends a request event (*Req*) to another module, the receiver. Once the receiver has completed the requested action, it sends an acknowledge event (*Ack*) back to the sender to complete the transaction.

Although self-timed circuits can be designed to implement their communication protocols in a variety of ways, the circuits used in our library use two-phase transition signaling for control and a bundled data protocol for data paths. Two-phase transition signaling [24] uses transitions on signal wires to communicate the *Req* and *Ack* events described previously. Only the transitions are meaningful; a transition from low to high is the same as a transition from

0-7803-4209-7/97 $10.00

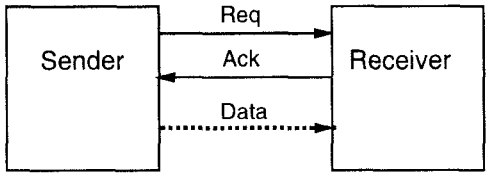

Figure 1: Two Modules Connected with a Bundled Data Path

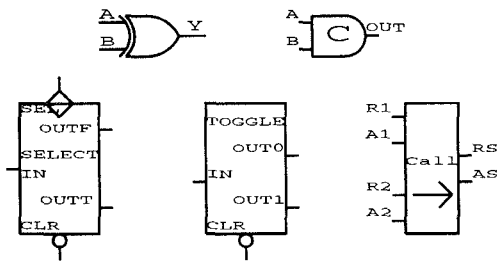

Figure 2: Control Modules for Self-Timed Designs

high to low and the particular state, high or low, of each wire is not important.

A bundled data path uses a single set of control wires to indicate the validity of a *bundle* of data wires [25]. This requires that the data bundle and the control wires be constructed such that the value on the data bundle is stable at the receiver before a signal appears on the control wire and remains valid until *Ack* is received. This condition is similar to, but weaker than, the equipotential constraint [24]. Two modules connected with a bundled data path are shown in Figure 1. The modules used to build the circuits targeted by ACT are described below.

2.1 Control Modules

The modules used to build the control path, described in more detail elsewhere [2, 5, 25], are shown symbolically in Figure 2. The functionality of the main control modules is as follows:

XOR: An XOR behaves as an OR for transition signals. When a transition occurs on any of its inputs, the XOR generates a transition at its output.

C-Element: A C-element is used as an AND function for transitions. A transition occurs at the output only when there have been transitions at both of its inputs.

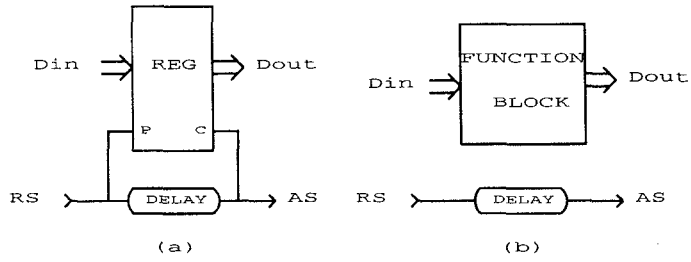

Figure 3: Generic latch and data Path Module

Note that the C-element must start in a state where both inputs are at the same value to behave in this way. A global clear signal to the control modules ensures this condition on system reset.

Select: A two-way transition Select module, in response to an input transition, causes a transition on one of two outputs depending on the value of its select (*SEL*) signal. The *SEL* signal should be valid before the input transition arrives and must remain valid until after an output transition is generated at one of the outputs. In other words, *SEL* is bundled with respect to the input transition.

Toggle: A Toggle module causes, in response to an input transition, an output transition on one of its two outputs. After initialization, the first input transition causes a transition on *out0* and subsequent input transitions cause transitions on alternate outputs.

Call: A Call module acts as a hardware subroutine call allowing multiple requesters to access a shared resource. The Call module routes the *Req* signal from a client (for example, either *R1* or *R2* in a two-way Call) to the subroutine circuit, and after the subroutine acknowledges on *AS*, routes the *Ack* back to the appropriate client. The requests must be mutually exclusive.

2.2 Target Data Path Modules

The module library contains many data path modules such as transition-controlled latches, adders, incrementers, decrementers, etc. All these modules are self-timed and follow a 2-phase *Req/Ack* protocol. Additional data path modules which follow this protocol can be added to the library as needed. A transition-controlled latch is shown in Figure 3(a).

Other data path modules also have a *Req/Ack* interface. Figure 3(b) shows a generic data path component. Here the logic represents the function of the data path module, and the delay represents the bundling delay of the module which models the delay of the logic block.

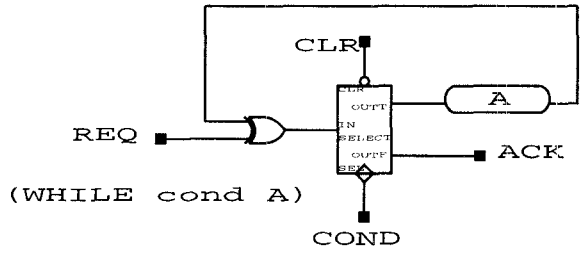

Figure 4: Translation of WHILE construct into Circuit

Module	Fault Coverage
C element	60.0%
Call2	60.2%
Select	71.5%
Toggle	90.0%

Table 1: Fault Coverage of Modules using Self-testing Functional Test

2.3 Synthesis Method

One way this module library is used is with an OCCAM based automatic circuit compilation system [3, 7]. The software constructs of OCCAM have been implemented using these library components and allow programs written in OCCAM to be translated automatically into self-timed circuits. An example of translation of a WHILE construct is shown in Figure 4. The OCCAM compiler has been used to build a number of systems ranging from a memory controller for standard DRAMs used in a self-timed environment [3], to a simple wormhole router designed for a mesh-connected multiprocessor array [6]. This library has also been used to build large circuits by hand, including a self-timed microprocessor [4], using commercial schematic capture software from Viewlogic .

Irrespective of the method used for synthesizing these circuits they can be used as input to ACT if they are in Viewlogic schematic form.

3 Fault Model

In the ACT tool a stuck-at-input model [12] with single stuck-at-fault assumption has been used. Unlike some previous work [18] faults inside the modules are also considered. The reason for also considering the faults inside the modules is that a significant percentage of faults inside the modules are not covered by the tests generated using models which only consider fault at the input/output of the modules. The fault coverages of the control modules under the module input/output stuck-at model with self-testing functional tests are shown in Table 1. Self-testing functional tests are based on the assumption that a self-timed circuits halts if there is fault in the circuits, so activating all the paths in the circuit will test for all the faults in the circuits. However this assumption holds only for the faults on the input/output of modules, which results in the low fault coverage for the modules when faults inside them are also considered as shown in Table 1. Details of the analysis of the fault coverage for these modules are described in [17].

4 ACT Tool

ACT is the first tool of its kind for macro-module based self-timed circuits which provides a complete design path from schematic capture to physical layout. Figure 5 shows the complete flow of the ACT tool. It takes a Viewlogic schematic of macro-module based self-timed circuits as input, performs testability analysis for selecting the latches for partial scan, and then automatically generates the testable design. This testable design is then mapped to a CMOS standard cell library. Placement and routing is done using tools from Cascade design automation. The testable design is then simulated in Viewlogic's Viewsim simulator with back annotated delays from Cascade's place and route tools. The testable design is also used to generate test vectors and to compute fault coverage of the tests generated. A custom test generation system has been developed to generate test vectors which includes a module for doing critical hazard free test generation [16]. Various submodules of the ACT tool are described in the following subsections.

4.1 Netlist Generator

This module takes a Viewlogic schematic of the design built from macro-modules. This submodule uses the Viewbase library [26] from Viewlogic to access Viewlogic's netlist data structures.

4.2 Testability Analysis

This submodule analyzes the circuit to select latches for partial scan. The analysis is done separately for control and data sections of the circuits. The rules for selection for each of these sections are described below.

4.2.1 Selection of Latches for Control Section

Latch selection for the control path uses the analysis of the modules described above. It also uses structural analysis to break any cycles. Overall the latch selection follows these three steps.

1. Analysis of Select and Toggle elements reveals that faults inside the Select and Toggle modules are difficult to test using functional test methods [15], so the

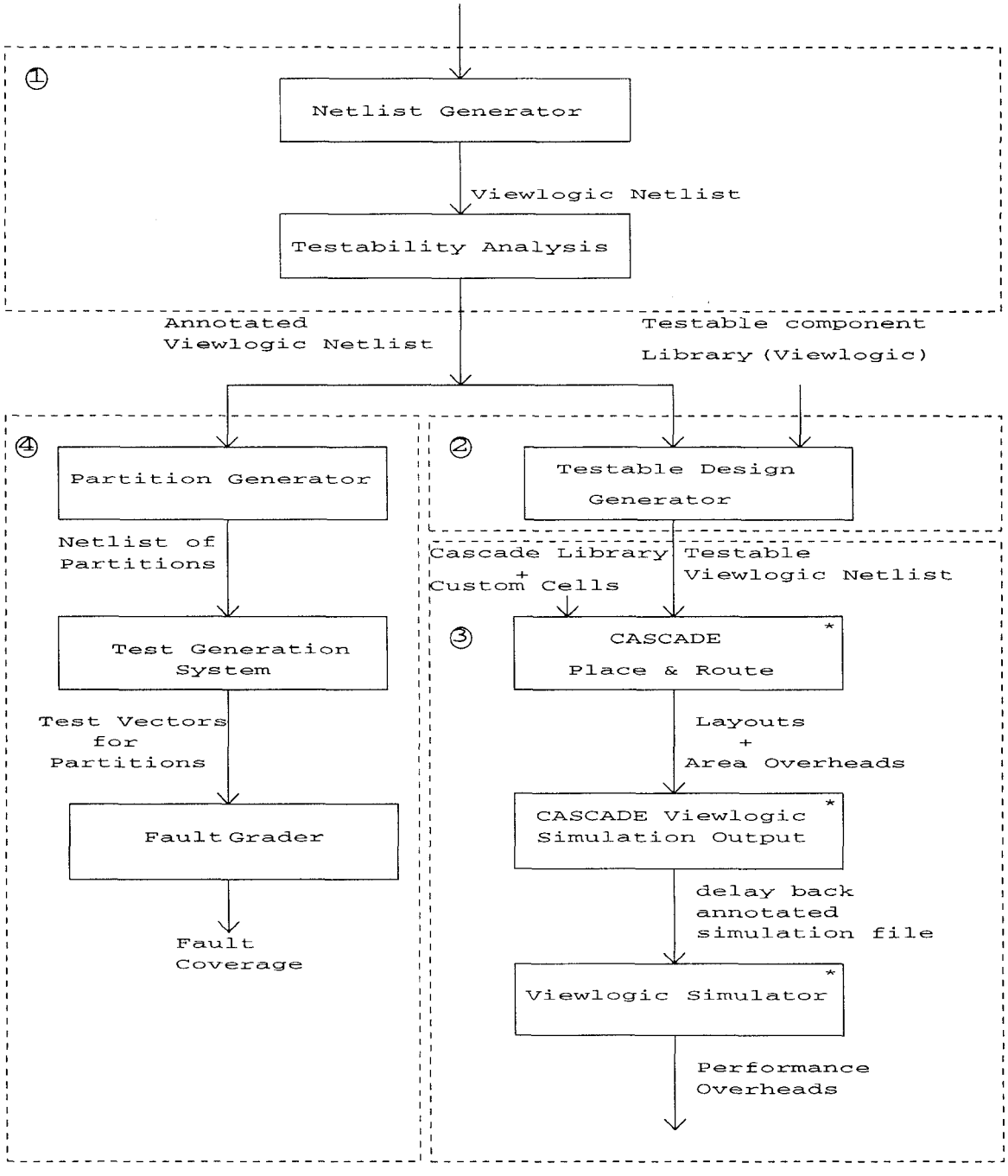

Figure 5: ACT Flow

latches in Select and Toggle elements are added to the scan path. This partitions the remaining circuit into networks of XORs and C-elements (Call elements are made of XORs and C-elements).

2. In the second stage, the Call elements are analyzed to see if it is possible to justify values of the *AS* line independent from the values of *R1* and *R2*. This is required to test for some faults which otherwise would be untestable using a functional test on the Call [17]. If this is not possible then one of the modules on the path from *RS* to *AS* is made scannable.

3. The last stage involves analysis of the circuit for loops not having any scannable latch in them. In such cases a component in the loop is made scannable. This step is added to ensure that the partitioned logic is cycle free except for the cycles inside the C-elements. The element selection is done in such a way as to break as many cycles as possible, thus minimizing the number of elements which need to be scanned over the entire circuit. The selection criteria follow a priority order (Delay>C-element>Call) in case of a tie. The reason for this priority order is that the extra delay for scan latch insertion can be hidden if a delay element is chosen.

4.2.2 Selection of Latches for Data Path

A structural analysis method is followed to select the latches in the data path which are to be made scannable. It was shown in [8] that the two factors that affect the complexity of test generation most are cycles and sequential depth. Cycles are especially important in asynchronous circuits [1], where the control is autonomous, which results in uncontrollable number of frames in an iterative model for test generation. In our approach the latches are selected for scanning to break the cycles in the circuit. The latches are selected in such a way as to minimize the number of latches that need to be scanned to break all the cycles in the data path.

4.2.3 Modifications to C-elements for Control Path Testing

After making the latches selected in the above steps scannable in the control path the entire control path is partitioned into networks of XOR and C-elements. C-elements, being sequential, will require a sequential test pattern generation for testing these partitions. C-elements have been modified as shown in Figure 6 to help generate tests for these networks. This modification is based on the observation that a C-element acts as an AND gate if its internal state

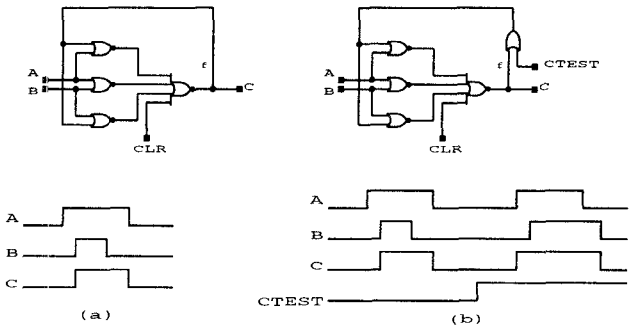

Figure 6: C-element Designs

is 0 and as an OR gate if its internal state is 1 as shown in Figure 6. The state of a C-element is the state of its feedback wire, so if we can control the state of the feedback wire we can make it act like an AND gate or an OR gate.

When the feedback wire for C-elements is set to 1 by asserting the *CTEST* signal the network of XOR and C-elements is combinational and any conventional test pattern generator can be used to generate tests for it. However testing in this mode does not cover about 40% of the faults inside the C-elements. For these faults to be tested, the C-elements have to configured in AND mode. A C-element can be put in this mode by asserting the *CLR* signal which sets the value of the feedback wire to 0. However as soon as the *CLR* signal is de-asserted the C-element reverts to its normal operation which means it will react to all the transitions (intended or unintended) on its inputs. This puts a requirement on the generated tests that no hazard can be generated in the circuit during tests which may put a C-element into an incorrect state leading to a steady state error and the invalidation of the test even when no fault really exists. In order to meet this requirement, a custom test pattern generator(described later in this paper) has been developed to generate critical hazard free tests for these networks. These tests are guaranteed to be free from any hazards which might lead to steady state errors during testing.

4.3 Testable Design Generator

This submodule uses the information collected in the analysis phase to automatically generate a testable design. All the latches which were selected for scan in the previous steps are converted to their scan equivalent. It also stitches the scan chain and adds the test controller block at the top level schematic to drive test mode signals. The modifications made in the design are highlighted by using a color coding scheme in the schematic.

The overall test environment after the partial scan insertion is shown in figure 7. *Test1* is a global control signal

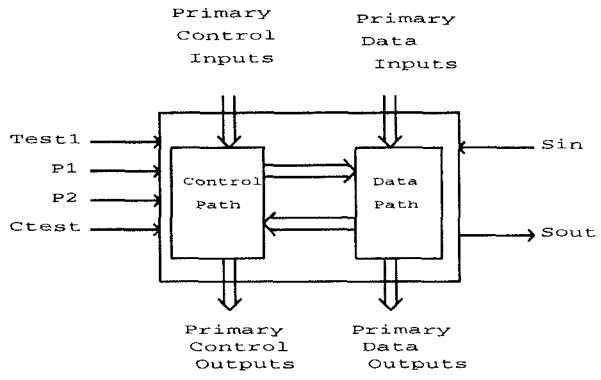

Figure 7: Generic testable circuit

which controls operation of circuit to be in normal mode or test mode. *P1* and *P2* provide the non overlapping clock for the scan path. Signal *CTEST* is used to help test the sequential portion of the circuit after partial scan insertion. Testing of circuit in this environment consists of testing the sequential partitions induced by scan path. The details of the implementation of the scan path are described in [15].

4.4 Partition Generator

This submodule generates partitions of the circuit which result from scan insertion for each primary and pseudo-primary(scan path input) outputs. These partitions are written to a file in ASCII format. The test pattern generator then generates tests for each of these partitions. A centralized data base is kept during test generation for marking detected faults so that efforts to generate tests for faults on nets which may be part of more than one partitions are not duplicated.

4.5 Technology Mapping

The testable design obtained above is mapped to CMOS standard cell library using Cascade Design Automation's place and route tools. Cascade's 2.0 CMOS micron library together with custom designed cells for the control modules are used in this step. The same place and route tool is also used to generate layouts for the original design and the two layouts are compared to get the area overhead.

The delays from the place and route tool are back annotated on the design using Cascade's Viewlogic interface and the testable design is resimulated with actual delays to get the performance impact figures.

4.6 Test Generation System

The test generation system of ACT consists of four components: a non-fault oriented test pattern generator, a D-algorithm based combinational test pattern generator, a critical path tracing based fault simulator and a new custom critical hazard free test generator.

The test generation for the partitions is done in two stages. In the first stage the C-elements are configured in OR mode by asserting the *CTEST* signal. In this mode the partition is totaly combinational. The following two steps are performed in this stage. First some random vectors are generated using the non-fault oriented test pattern generator module. These vectors are then fault simulated and the faults that are detected are marked. In the second step the deterministic test pattern generator is used to generate tests for the remaining undetected faults. Each of the tests generated in this step are fault simulated and the detected faults are marked in the data base.

The C-elements in the second stage are put in AND mode by asserting the CLR signal and de-asserting the *CTEST* signal. A custom critical hazard free generator is then used to generate tests for the remaining faults inside the C-elements. A new 6-valued algebra has been developed to generate these tests. The details of this algebra are described in [16]. In this step the C-element is kept in AND mode as described above.

5 Experimental Results

The ACT tool was applied to the example self-timed macro-modular circuits shown in Tables 2, 3, 4. In these tables Division is a simple serial division circuit. The GCD and the Barcode are 1992 high-level synthesis benchmarks. GCD is a self-timed implementation of Euclid's GCD algorithm, and Barcode is a self-timed implementation of a barcode reader. The Router circuit is a torus-connected wormhole routing chip for message routing in a multiprocessor [3, 9]. The circuit called IF-unit and ID-unit are the instruction fetch and instruction decode unit of the NSR, a self-timed pipelined RISC processor [4]. ErrDet is the error detector section of DCC error corrector from Philips Inc. All these circuits were designed in Viewlogic schematic form using the self-timed library described earlier. ACT was then applied to these circuits to get the testable designs and to generate test vectors. The original and testable designs are compared for area, performance, and fault coverage in the following part of this section.

- **Area:** In this table the latches-scanned column indicates the percentage of latches which have been converted into scan latches for partial scan. The area column indicates the actual silicon area for original and

Table 2: Latches Scanned and Area

Design	Latches			Area		
	Total	Scanned	%Scanned	Basic (sq mm)	Testable (sq mm)	% Overhead
Division	59	21	35.5%	2.51	2.80	11.5%
GCD	36	14	38.8%	2.08	2.38	15.0%
Barcode	89	27	30.3%	2.89	3.25	12.4%
Router	52	25	48.0%	1.44	1.92	33.3%
IF-unit	78	29	37.1%	3.79	4.37	15.3%
ID-unit	57	10	17.5%	2.51	2.80	11.5%
DCC-Errdet	95	56	58.9%	5.55	6.46	16.3%

testable designs with the percentage area overhead. As shown in the last column the area overhead is about 17%

- **Performance:** The measurement of performance is difficult for self-timed circuits because no such simple parameter as the clock for synchronous circuits exists. The effect of the performance of a component in the circuit is local as compared to synchronous circuits where the slowest component in the circuit determines the global speed of the clock. The performance of a self-timed circuit is typically measured as the delay on the longest paths or as cycle time for some iterative computation in the circuit.

  Since the partial scan affects only parts of the circuits, the performance overheads shown in Table 3 represent the change in delay of the path or a cycle time which is affected worst. The performance figures shown in Table 3 indicate that the performance overhead is in the range 1%-13%. The reason for the variations in the percentage overhead is twofold. First, the partial scan affects only a part of the circuits. Second, the percentage overhead depends on the length of the path. So if the path is long and has some modifications on it then it will have a lesser percentage of overhead compared to a smaller path with the same modifications. So despite the variation in the overhead percentage, the largest overhead is still small. The reason for such low overhead is that in most cases the delay of scan elements could be used as the bundling delay already required in the circuit. In the original circuits the bundling delay was explicitly inserted as delay elements which were replaced by scan elements in the testable version.

- **Fault Coverage:** Table 4 shows the fault coverages for the control path and for the whole circuit. The reason for showning the fault coverage for control path separately is that earlier methods which were based on self-testing property were only for the control path. Therefore the first column compares the fault coverages under the self-testing method and under the proposed partial scan method. The last column also gives the fault coverage for the whole circuit (control path and data path) under the the partial scan method described in this paper.

6 Conclusions

A new DFT tool called ACT for testing macro-module based self-timed circuits is described. This tool is the first of its kind. It provides a complete path from circuit schematic to physical layout and test vectors for self-timed circuits. It automatically incorporates a partial scan in user designs and uses commercial tools to generate layouts. Experimental results show that this technique results in about 60% savings in terms of the number of latches that need to be scanned compared to a full scan approach. The average area overhead of this technique for the example circuits is about 17%, which is not very high. The performance overhead in the example circuits is in the range of 1%-13%. The fault coverage achieved in all the example cases is over 97% which is above the required industry level of >95%. The results of this experiment indicate that the proposed partial scan technique for testing macro-module based self-timed circuits is viable and high fault coverage is possible using this technique without having very high area and performance overheads.

ACT tool is just a first step towards a more sophisticated tools for design and test of asynchronous circuits where the lack of commercial tool like the ones available to synchronous circuit designers remains one of the obstacles in their acceptance as viable alternative.

References

[1] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

Table 3: Performance Overheads

Design	Performance(delay)		
	Basic	Testable	% overhead
Division	270.28ns	283.4ns	4.8%
GCD	158ns	165ns	4.4%
Barcode	73.8ns	76.18ns	3.2%
Router	539.29ns	546.02ns	1.2%
IF-unit	13.2ns	14.9ns	12.9%
ID-unit	22.66	25.62	13.06%
DCC-Errdet	48.6214μs	48.8443μs	<1%

Table 4: Fault Coverage

Design	Control Path Coverage			Overall Fault
	Self-Testing	Partial Scan	Improvement	Coverage
Division	67.5%	95.63%	28.13	98.50%
GCD	74.6%	96.86%	22.26	99.30%
Barcode	66.2%	98.30%	32.10	98.75%
Router	65.7%	99.67%	33.97	99.72%
IF-unit	66.1%	98.48%	32.38	98.43%
ID-unit	67.1%	100.00%	32.90	99.77%
DCC-Errdet	58.2%	96.79%	38.59	97.20%

[2] Erik Brunvand. Using FPGAs to Implement Self-Timed Systems. *Journal of VLSI Signal Processing*, 6:173–190, 1993.

[3] Erik Brunvand. *Translating Concurrent Communicating Programs into Asynchronous Circuits*. PhD thesis, Carnegie Mellon University, 1991. Available as Technical Report CMU-CS-91-198.

[4] Erik Brunvand. The NSR Processor. In *Proceedings of the 26th International Conference on System Sciences*, Maui, Hawaii, January 1993.

[5] E. Brunvand and R. Sproull. Translating Concurrent Programs into Delay-insensitive Circuits. In *Proceedings of International Conference on Computer Aided Design (ICCAD)* 1989.

[6] Erik Brunvand, Nick Michell, and Kent Smith. A Comparison of Self-timed Design Using FPGA, CMOS, and GaAs technologies. In *International Conference on Computer Design (ICCD)*, Cambridge, Mass., October 1992.

[7] Erik Brunvand and Mike Starkey. An Integrated Environment for the Design and Simulation of Self-timed systems. In *VLSI-91*. IFIP, August 1991.

[8] K. T. Cheng and V. D. Agrawal. A Partial Scan Method for Sequential Circuits with Feedback. *IEEE Transactions on Computers*, pages 544–548, 1992.

[9] William J. Dally and Charles L. Seitz. The Torus Routing Chip. *Distributed Computing*, 1:187–196, 1986.

[10] R. Gupta, R. Gupta and M. A. Breuer. BALLAST:A Methodology for Partial Scan Design. In *Fault Tolerant Computing Symposium*, pages 118–125, 1989.

[11] Pieter Hazewindus. *Testing Delay-Insensitive Circuits*. PhD thesis, California Institute of Technology, 1991.

[12] Henrik Hulgaard, Steven M. Burns, and Gaetano Borriello. Testing Asynchronous Circuits: A Survey. Technical Report UW-CSE-94-03-06, Department of Computer Science and Engineering, Univ. of Washington, Seattle, 1994.

[13] Ajay Khoche and Erik Brunvand. Testing Self-Timed Circuits Using Scan Paths. *5th NASA Symposium on VLSI Design*, Nov 1993.

[14] Ajay Khoche and Erik Brunvand. Testing micropipelines. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1994.

[15] Ajay Khoche and Erik Brunvand. A partial scan methodology for testing self-timed circuits. In *Proceedings of VLSI Test Symposium*, May 1995.

[16] Ajay Khoche and Erik Brunvand. Critical hazard-free test generation for asynchronous circuits. In *Proceedings of VLSI Test Symposium*, Apr. 1997.

[17] Ajay Khoche. *Testing Macro-module based self timed circuits*. Phd Thesis, University of Utah, 1996.

[18] P. Kudva and V. Akella. Testing Two Phase Transition Based Self-timed Circuits in a High Level Synthesis Environment. *High Level Synthesis workshop* May 1994.

[19] S. Pagey, G. Venkatesh, and S. Sherlekar. Issues in Fault Modeling and Testing of Micropipelines. *First Asian Test Symposium*, Hiroshima, Japan, Nov 1992.

[20] Marly Roncken and Ronald Saeijs. Linear Test Times for Delay-insensitive Circuits: A Compilation Strategy. In S. Furber and M. Edwards, editors, *Proceedings of IFIP Working Conference on Asynchronous Design Methodologies*, pages 13–27, Manchester, UK, 31 March – 2 April 1993, 1993. Elsevier Science Publishers.

[21] Marly Roncken and Eric Burls. Test quality of asynchronous circuits: a defect oriented evaluation. In *ITC*, Oct. 1996.

[22] Marly Roncken, Emile Aarts and Wim Verhaegh. Optimal Scan for Pipelined testing: An asynchronous foundation. In *ITC*, Oct. 1996.

[23] Marly Roncken. Partial scan test for asynchronous circuits illustrated on a dcc error corrector. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1994.

[24] C. L. Seitz. System Timing. In *Mead and Conway, Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[25] Ivan Sutherland. Micropipelines. *CACM*, 32(6), 1989.

[26] *Viewbase Manual*, Viewlogic Systems Inc.

[27] Chin-Long Wey, Ming-Der Shieh, and P. David Fisher. ASLCScan: A Scan Design Technique for Asynchronous Sequential Logic Circuits. In *International Conference on Computer Design(ICCD)*, Cambridge, Mass., October 1993.