Testing Micropipelines*

Ajay Khoche Erik Brunvand
Department of Computer Science
University of Utah
Salt Lake City, UT, 84112

Abstract

Micropipelines, self-timed event-driven pipelines, are an attractive way of structuring asynchronous systems that exhibit many of the advantages of general asynchronous systems, but enough structure to make the design of significant systems practical. As with any design method, testing is critical. We present a technique for testing self-timed micropipelines for stuck-at faults and for delay faults in the bundled data paths by modifying the latch and control elements to include a built-in scan path for testing. This scan path allows the processing logic in the micropipeline, as well as the control of the micropipeline, to be fully tested with only a small overhead in the latch and control circuits. The test method is very similar to scan testing in synchronous systems, but the micropipeline retains its self-timed behavior during normal operation.

1 Introduction

Pipelines are pervasive in modern digital systems. As a means of increasing performance, they appear in almost all aspects of system architecture from the instruction and arithmetic units of general purpose processors, to special purpose hardware for graphics processing, signal processing, image compression and error detection. An interesting class of pipeline circuits was described by Ivan Sutherland in his Turing award lecture from 1989 [13] which he termed *micropipelines*. These pipelines are asynchronous event-driven pipelines that can be used to build very fine-grained pipeline structures. Micropipelines are flow-through elastic pipelines that are controlled by local handshaking rather than a global clock signal. The asynchronous nature of micropipelines allows them to be free of many problems related to the clock that plague synchronous pipelines. In particular, the processes that read from and write to the micropipeline need not be synchronized. The issue of whose clock should control the pipeline is avoided, and thus issues of clock synchronization are also avoided. A micropipeline is also free to take advantage of data-varying delay in the processing at each stage. Rather than set the clock period to the maximum delay incurred at any pipe stage, the individual pipe stages in a micropipeline finish as quickly, or as slowly, as the current data require. This allows the processing in the pipeline to occur at rates closer to the average processing time than the worst-case time which can mean significant performance improvements in some systems. Additional advantages of asynchronous circuits include freedom from a variety of clock related timing problems, simpler system composition, increased robustness in the face of process and environmental variation, and the potential of low power consumption.

Although micropipelines seem to have many advantages, the fact that they are asynchronous circuits may, in fact, cause concern. One specific area for which asynchronous circuits seem to lag behind synchronous circuits is in testing. Testing asynchronous circuits which are not subject to a global clock is traditionally considered difficult when compared to testing synchronous circuits. There are a number of techniques for integrating testability into system design, but many of these traditional testing methods used for synchronous circuits are not directly applicable to non-clocked asynchronous circuits. As a result, many asynchronous circuits do not employ design for testability techniques. In this context new methods are required to test asynchronous circuits. In this paper a design-for-test (DFT) approach is proposed to test micropipelines. Our approach involves modifying the asynchronous latches in the data path and the micropipeline control structures such that the micropipelines can be tested in a way similar to that in a synchronous pipeline. In particular, a scan path is

*This Research is supported in part by University of Utah Research Committee.

0-8186-6210-7/94 $4.00 © 1994 IEEE

established through the latch and C-element circuits that allows scan testing of the micropipeline. The micropipeline uses a global scan clock for testing only, and then reverts to fully asynchronous mode during normal operation, preserving the benefits of that type of operation.

In the next section the structure and operation of micropipelines is described in general. The proposed test method and the modifications made to the circuit elements are discussed in Section 3. Section 4 describes the test procedure and Section 5 contains a comparison with existing work. Section 6 describes an example testable micropipelined multiplier and Section 7 concludes the paper with a discussion of tradeoffs involved in this style of testing micropipelines.

2 Micropipelines

Micropipelines are asynchronous, event driven, elastic pipelines that may or may not contain processing between the pipe stages. If no processing is done between the pipe stages, the micropipeline reduces to a simple first-in first-out (FIFO) buffer. A block diagram of a generic micropipeline is shown in Figure 1. It consists of three parts: a control network consisting of one C-element per micropipeline stage, a latch in each stage, and possibly some processing logic between the stages. The C-elements control the action of the micropipeline by acting as protocol preserving AND gates for the transition control signals of the micropipeline. These gates, drawn as an AND gate with a C inside, will drive their output low when both inputs are low, and high when both inputs are high. When the inputs are at different states, the output is held at its previous level. Note that one input of each C-element used in Figure 1 is inverted. Thus, assuming that all the control signals start low, the leftmost C-element will produce a transition to the leftmost latch when the incoming request (RIN) line first makes a transition from low to high. The acknowledge from the latch will produce a similar request through the next C-element to the right. Meanwhile, the leftmost C-element will not produce another request to the leftmost latch until there are transitions both on RIN (signaling that there are more data to be accepted) and the Ack from the next latch to the right (signaling that the next stage has finished with the current data). Each pipe stage acts as a concurrent process that will accept new data when the previous stage has data to give, and the next stage is finished with the data currently held.

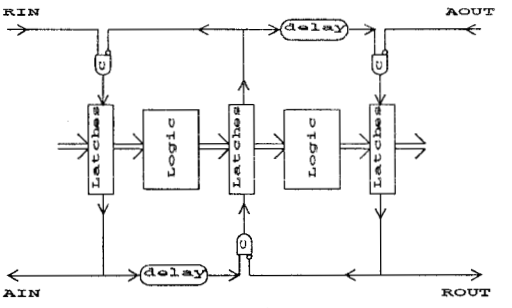

Figure 1: A Micropipeline

Note that the micropipeline structure is a flow-through design. The data entering at the left of Figure 1 will flow through each stage of the micropipeline on its way from input to output. When the data reach the rightmost latch, a transition is produced on the ROUT line to indicate that there are data available to be read. Once those data have been read, the reader produces a transition on the AOUT line to allow new data to enter that pipeline stage.

The pipelines considered in the paper use two-phase transition signaling and bundled data paths. Two phase transition signaling means that a transition on a control line from low to high or from high to low is considered as an event and no distinction is made between two type of transitions. Bundled data path means that data must be stable at a latch before the latching control signal arrives. This condition is similar to, but weaker than the equipotential constraint [10]. The transition control signal indicates that the bundled data path is valid, and that the latch may update its contents. The delay elements shown in Figure 1 model the delay required to satisfy this constraint as the data move through the logic at each stage. The logic could of course be self-timed and generate a completion signal, which would eliminate the need for the delay at the expense of more complicated logic for detecting completion of the processing. Latches used in such circuits are transition latches which latch new data upon receipt of a request event and produce an acknowledgment event when done. A description of the operation of these latch and C-element circuits, as well as schematics, are described in the following section.

3 Scan design

To modify a micropipeline for testability, we propose to modify the latch and C-element circuits to in-

clude a built-in scan chain. This scan-chain will allow logic in individual micropipeline latches to be tested independently using techniques well known from testing synchronous circuits.

3.1 Latch design

Latches used in transition-controlled micropipelines must latch normal data values in response to transition control signals. These latches [2, 13] are typically controlled by a pair of signals called *capture*(C) and *pass*(P). These control signals must alternate and control whether the latch is currently transparent, or holding a value. The initial condition, where the values of *capture* and *pass* are the same determines the type of the transition-controlled latch. If the latch is transparent in this state, the latch is known as a *transition latch normally transparent* (TLNT) and if the initial condition is to be holding the current state, the latch is known as a *transition latch normally opaque* (TLNO). These latches are identical in their operation except for their initial conditions. If the latch is currently opaque, the next transition should be on the *pass* wire, and if currently transparent, should be on the *capture* wire. In our circuits, the micropipeline latches can be of either type. The circuit for a TLNO latch is shown in Figure 2(a). It is sometimes convenient to build latches with a request-acknowledge interface instead of the capture-pass interface. This is readily accomplished using a transition latch and a bundling delay. The *pass* signal is delayed to become the capture signal with a bundling delay to make sure that the value to be captured has had time to traverse the feedback path of the latch.

Because the transition latch includes the equivalent of two simple latches in its implementation, it can be converted into a scan latch by providing a multiplexed serial input and making the connection shown in Figure 2(b). In this way the asynchronous transition latch becomes a muxed master/slave flip/flop. The normal and scan mode operation is controlled by *SCAN* input, which is global scan control like that in synchronous scan designs. The latches can be designed such that the latching takes place on either low or high value of the control signal during scan mode.

The basic and scan version of these latches have been designed in CMOS using Magic layout tools and simulated in SPICE for a 2 micron process. The area and performance overheads are shown in Figure 4. The transistors column gives the overhead in terms of transistors while the area column gives overhead in terms of square lambda for our implementation. The

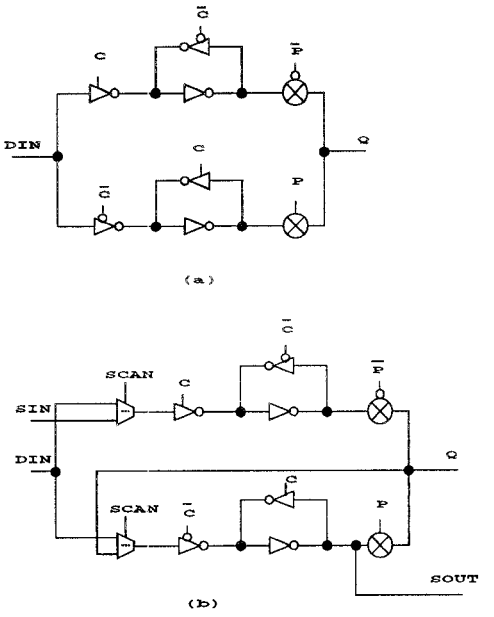

Figure 2: Basic and Scan Versions of a Transition Latch

delay column gives the average time between occurrence of a request transition and receipt of corresponding acknowledgment transition for rising and falling request transitions.

3.2 C-Element design

A transistor level schematic of a generic C-element is shown in Figure 3(a) [13, 14]. The first column of transistors at the left of this schematic implement a dynamic C-element. The transistors on the right implement a latch stage to make the C-element static.

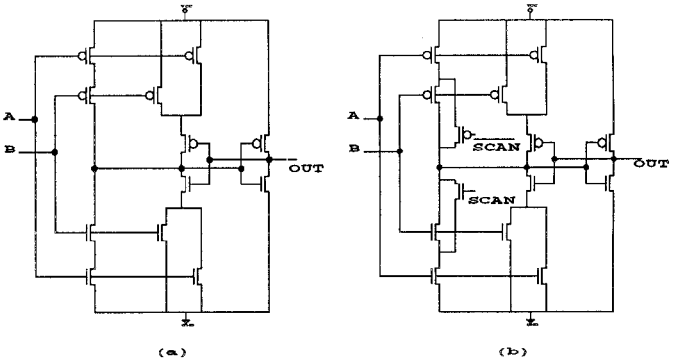

Figure 3: Basic and Scan Versions of a C-element

Attribute	Latch		C-element	
	Basic	Scan	Basic	Scan
Transistors	34	42	14	16
Area	34656	40432	5776	6080
Delay	3.58ns	3.63ns	2.07ns	2.25ns

Figure 4: Area and Performance Overheads

The feedback transistors in the latch stage driven by the C-element inputs are weak and are used to provide feedback when the dynamic stage is not driving the internal node. This C element has been modified by bypassing the B input transistors in the input stage in scan mode. Thus in scan mode input stage passes $\overline{a}$ directly to the output stage. The operation between normal and scan mode is selected by the $SCAN$ control signal as in the latch case. The modified schematic is shown in Figure 3(b) and requires only two extra transistors. The basic and scan versions of C element have also been designed using Magic layout tools and simulated using SPICE for a 2 micron process. The results are shown in Figure 4. Here the delay represents the average time after both inputs have changed to the time when the output changes. The area is again in terms of square lambda units.

4 Test methodology

4.1 Organization of scan path

A micropipeline with a scan path included is shown in Figure 5. Bold lines indicate the control path in the test mode which forms the clocking network for scan path. The latches in the micropipeline are connected in a scan path with SIN input coming to the first stage on the left and $SOUT$ coming out from the last stage on the left. The latches are enabled on alternate transitions of the control signal starting with the latch on the right which is enabled on a high to low transition.

The C-elements used are such that they pass their negated input onto the output in $SCAN$ mode. This way the control path forms a clocking network for the scan path with $AOUT$ being the clock input. Note that the clock runs in the opposite direction to that of data: the clock first reaches the stage closest to the output and then cycles through the stages to the AIN output. Also note that the clock value is inverted as it passes through each C-element, as it is incident on the negated input of the C element. This is why latches with different enabling properties are used in the alternate stages.

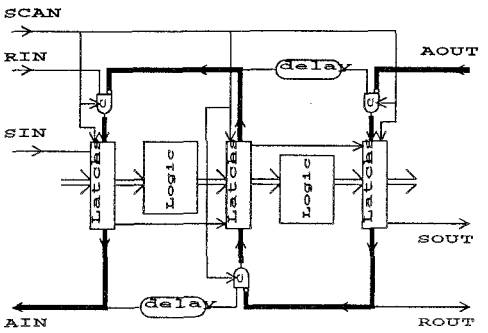

Figure 5: Micropipeline with Scan Path

The reason for routing the $SCAN$ clock in the reverse direction from the data is twofold: First, while testing the processing logic the preceding stage holds the test vector while the following stage serves the purpose of being the observation point. If the clock were run in the same direction as the data, the test vector would have changed its value during test before the response had been captured in the observation stage. This is because the clock would have reached the preceding stage before cycling to the next stage. The second reason is related to efficiency. By running the test clock backwards we have removed the effect of the bundling delays, so we can run our test clock at higher rate for shifting the patterns in and out.

4.2 Test procedure

Testing Micropipelines involves testing for stuck-at faults in the circuits and testing for bundling constraint violations which is similar to delay testing in synchronous circuits at each stage. The procedure for testing for these two types of faults is described below.

4.2.1 Testing for stuck-at faults

There are three parts of a micropipeline that must be tested: the processing logic between the stages, the control logic that controls the micropipeline in normal operation (the C-elements), and the latches that hold the data values in the micropipeline. The test procedure for each of the three components of the micropipelines is described below. Note that the procedure is very similar to that used in synchronous circuits. The asynchronous micropipeline uses a synchronous test mode, but reverts to fully asynchronous operation in normal mode preserving the benefits of asynchrony. The following three tests will fully test each net in the micropipeline for stuck-at faults.

Testing the processing logic

The processing logic is assumed to be combinational and is tested using the following steps. Intuitively, the micropipeline is filled with values that become the inputs to the processing logic at each stage, and the C-elements are left in a state where the pipeline state is full. The circuit is returned to normal mode, and one item is removed from the pipe which latches the values from each bundle of processing logic into the next micropipeline latch. The circuit is returned to scan mode and the results are shifted out and compared with the expected results. The test vectors for the combinational processing logic at each stage of the circuit may be generated with standard test-generation programs [1]. The detailed operation is as follows:

1. Put the micropipeline into scan mode by asserting the $SCAN$ signal ($SCAN = 1$).

2. Shift the test vector into the scan path (the vectors can be generated using any conventional test pattern generator) by clocking $AOUT$. Clocking consists of the pattern $AOUT = 0 \rightarrow 1 \rightarrow 0$.

3. Set $RIN = 0$ if the number of stages in the pipeline is odd otherwise set $RIN = 1$.

4. Return the circuit to normal operation mode ($SCAN = 0$). At this point the inputs to the latches come from the processing logic outputs and the C-elements return to their normal operation. The value of the C element outputs in each stage are those which occur when the pipeline is full under normal operation with $ROUT$ being high and unacknowledged. In this stage when the acknowledgment arrives on $AOUT$ it passes through all the stages up to AIN.

5. Set $AOUT = 1$. As described in the previous step, this *ack* transition trickles through all the stages of the pipeline making latches to capture the outputs of their processing logic into their master stage.

6. Return the circuit to scan mode ($SCAN = 1$).

7. Set $AOUT = 0$. This causes the latches to copy the value from their master stages to their slave stages.

8. Clock $AOUT$ to shift out the outputs of logic stages.

Testing the control logic

The part of the control logic consisting of C-elements and the nets that connect them is tested concurrently with above steps. The C-elements in the even stages i.e. 0,2,4 ... starting from the right side are tested for a $1 \rightarrow 0$ transition, while the C-elements in the odd stages 1,3,5... are tested for a $0 \rightarrow 1$ transition. Any stuck-at-0 fault on the complemented input or stuck-at-1 fault at the output of the C-elements in the even stages will result in no transition being produced on the AIN line when the the $AOUT$ is toggled in the normal mode. The same is true for stuck-at-1 faults on the complemented input and stuck-at-0 the output of the C-elements in the odd stages. In order to test the control network for the remaining part, the pipeline is put into another state where the micropipeline is full with $AOUT = 1$. In this state the values on the C-element outputs are exactly opposite to that in the previous case. The pipeline can be brought to this state by putting it into scan mode and setting $AOUT = 1$. To test for these remaining faults the pipeline is returned to normal mode and $AOUT$ is set to 0. This causes a $0 \rightarrow 1$ transition on the C-elements in the even stages and a $1 \rightarrow 0$ transition in the odd stages, which tests the nets in the control part for the remaining faults.

Testing the latches

The latches are tested using the fault model proposed in [8]. This work models two types of faults for latches of this type: stuck-at-capture and stack-at-pass. In the stuck-at-capture model the storage elements assume a particular value forever. In the stuck-at-pass model the latch fails to latch any data and always passes the input to output.

The latches can be tested for the above faults using a shift operation. Stuck-at-pass faults can be detected by shifting an alternating 0-1 pattern through scan path. Stuck-at-capture faults are equivalent to stuck-at-i fault, where i is the value being permanently captured in the latch. Stuck-at-capture faults can be detected by shifting 0s for stuck-at-1 faults and 1s for stuck-at-0 faults.

In these steps, however, the net segment feeding the processing logic is not tested, but a fault on that segment is equivalent to the fault on the input of the processing logic, and hence gets tested along with the processing logic.

4.2.2 Testing for bundling constraint violations

As mentioned earlier, testing for bundling constraint violations is similar to path delay testing in synchronous circuits. In synchronous circuits a path delay fault is said to have occurred if delay through any path in the combinational logic between two latches exceeds the clock period [12]. The corresponding behavior in micropipelines is that a bundling constraint is violated if any path in the combinational logic between two stages has greater delay than the bundling delay. However in micropipelines the bundling delay in each stage is not the same, as it is in synchronous case where the delay is measured against the global clock everywhere in the circuit. (Of course, if the processing logic between the micropipeline stages is itself self-timed (generating a completion signal when the logic has computed a result) then there is no bundling constraint to test.)

Various methods have been proposed in the literature for robust path delay testing in synchronous circuits [6, 9, 11]. All these methods require a two pattern test to test a path. The first pattern is used to setup the values in the circuits such that when the next pattern is applied a transition is produced on the path under consideration and propagated to the outputs. These patterns are required to have a property that propagation of the transition through the path is not invalidated by the delays in the other part of the circuits.

The test procedure used in these tests is first a pattern is applied to the combinational block and the circuit is allowed to settle. The second pattern is then applied at time t1 and the output latch is clocked at time t2 such that (t2-t1) models the clock period in the circuit.

Three mechanisms have been proposed in the literature for applying the two pattern test described above in a standard scan environment. The first approach requires use of an enhanced scan flip/flop which can store two values at one time corresponding to two patterns [4, 5]. This approach is very expensive in terms of area if the number of latches in the design is large. In the second method, a second pattern is derived by shifting in the scan path [15]. This method is limited in the sense that it is not always possible to derive the second pattern just by shifting for any arbitrary combinational circuit. The third method described in [3] exploits the pipeline property of the circuits. In this method a pipeline is fully delay fault testable if individual combinational blocks are delay testable individually and if the blocks can produce all possible output combinations. In this case the first pattern is applied by shifting through the scan path while the second pattern is derived from the combinational block in the previous stage. This can be applied to the block under test by returning the circuit to normal mode and then clocking the circuit once, so that the output of the previous stage gets latched into the input latch of the block under consideration. The output of the block under test is latched after a time equal to the clock period and later scanned out. The condition that combinational blocks can produce all possible combinations can be handled during the synthesis process [3]. If an output combination is not possible, then it implies that this output is a don't care for the circuit. The circuit can then be optimized with this additional don't care condition.

The third method discussed above is used to test micropipelines. In this method, the patterns can be shifted in the scan path such that pattern *(v1)* is applied to the target block and another pattern *(v3)* is applied to the block B previous to the target block in the pipeline, such that B(v3)=v2 which is the second pattern required to test a path in the target block. The circuit is then returned to normal mode and a *req* event is applied to the micropipeline. This causes each pipeline latch to capture new inputs after a delay equal to bundling constraint. If any path in target block has a delay fault i.e. has delay more than the target fault, then the values captured by the latch at the output of that block will not be correct which can be scanned out to verify.

5 Comparison with existing techniques

The only published work in testing micropipelines is in [8] where only stuck-at faults were considered and no timing faults resulting in bundling constraint violation were tested. In their approach the TLNT (Transition Latch, Normally Transparent) type of latches are used in the micropipeline and the processing logic between all the stages in the micropipeline is assumed to be lumped together for testing. Because the latches are transparent in the initial case, the pipeline is transparent from input to output after initialization and the entire logic path may be tested by applying vectors at the input to the micropipeline and observing results at the output of the micropipeline. This approach of lumping all the logic in the micropipeline together for testing it has two major drawbacks: the time required to generate the test vectors is much larger (in fact, the test pattern generator may fail to generate tests

for some otherwise detectable faults because of circuit complexity), and testing the processing logic in a lumped fashion will require a lower test clock speed on the automatic tester to account for the delay through the lumped block and latches.

In contrast, using our approach the processing logic between the stages is tested separately, so the test generation requires less time and the tester clock rate is determined only by the delay through the latches. Of course, as in all scan based approaches, the test application time will increase due to the time taken by the shifting.

In the approach of [8], the latches in the micropipeline are tested by a two pattern test for stuck-at-pass faults, where, to test for faults in a latch in stage i, enough elements are pushed in the pipeline such that the pipeline is completely full from stage i+1 to the last stage. At that point another vector is applied to the micropipeline which causes stage i to latch a certain value. Then the next vector is applied to the pipeline so that if there is a fault at the target latch its output will be different than it would have been in fault free case. The effect of the faults are then propagated through the processing logic of the following stages. Implicit in this procedure is the assumption that the combinational logic does not have any redundant faults. This approach is very complex, as each latch may have to be tested individually, and every time the test setup will be required to follow above steps. Also fault propagation through the combinational block has the implication that certain faults in latches may not be detectable due to redundant faults in the processing logic. In contrast our approach is very simple involving only shifting.

Our approach to test the control path (C-element network) is similar to their approach.

6 Example

A micropipelined version of a 4X4 multiplier shown in was designed using ViewLogic tools to test the ideas described in this paper. The same circuit was also coded in VHDL at a structural level for test generation purposes. The Attest Software Inc. software was used for test pattern generation. Tests were generated first for all the stages being lumped together and then for individual stages. A summary of test generation results is shown in Figure 7. In this table module *Mult* represents the lumped version and the individual stages represent our approach of testing each stage independently. In this table *stage2-3* represent stage 2

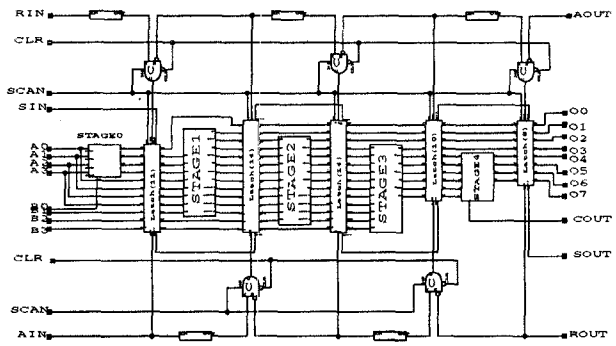

Figure 6: A 4X4 Micropipelined Multiplier

Module	Test generation Time	No. of Vectors
Mult	8.2sec	30
stage0	.1sec	7
stage1	.1sec	13
stage2-3	.4sec	19
stage4	.5sec	17

Figure 7: Test generation statistics for Micropipelined Multiplier

or 3. Since these stages are identical, test generation has to be done for only one of them. The test generation time includes time taken for test generation and time taken for fault simulation. The vector column indicates the total number of vectors required for the faults that are detectable.

As shown in the table the total test generation time for the stages is much less than that required when the tests are generated with all the stages lumped together. The test vectors for various stages are concatenated to form a single scan vector. The number of times vectors have to be scanned in, which is equal to maximum over number of vectors required for any single stage, is also less than number of times tests to be applied in lumped mode.

7 Conclusions

We have presented a technique for testing self-timed micropipeline circuits using a built-in scan path. The event-driven latches used to build micropipelines are readily converted into scan latches with only a small overhead. This allows the processing logic in the micropipeline stages to be tested in a way that is very similar to familiar synchronous scan testing. The C-element control of the micropipeline is also modified to provide a scan clock during scan mode, and is also testable under a stuck-at fault model. However, al-

though the scan testing is performed under control of a scan clock, the micropipeline retains its fully self-timed nature in normal operation. This technique allows the micropipeline to be completely tested for stuck at faults and also for delay faults in the bundled data paths. We are able to test for delay faults because each micropipeline stage is tested independently. This also allows the test generation and application time to be reduced significantly compared to techniques that test the entire circuit at once.

References

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

[2] Erik Brunvand. Translating Concurrent Communicating Programs into Asynchronous Circuits. PhD Thesis, Carnegie Mellon University, 1991.

[3] Kwang-Ting Cheng, Srinivas Devadas and Kurt Kuetzer. Robust Delay Fault Test Generation and Synthesis for Testability Under A Standard Scan Design Methodology. *28th ACM/IEEE Design Automation Conference*, 1991.

[4] S. Devadas and K. Kuetzer. Synthesis and Optimzation Procedures for Robustly Delay-Fault Testable Logic Circuits. *27th ACM/IEEE Design Automation Conference*, 1990.

[5] S. Devadas and K. Keutzer. Design of Integrated Circuits Fully Testable for Delay Faults and Multifaults. *International Test Conference*, 1990.

[6] K. Fuchs, F. Fink and M. H. Schulz DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults. *IEEE Transaction on Computer Aided Design*, Vol 10, No 10, Oct 1991.

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

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

[9] S. M. Reddy, C. J. Lin and S. Patil. An Automatic Test Pattern Generator for the Detection of Path Delay Faults. *IEEE International Conference on Computer Aided Design*, Nov 1987.

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

[11] J. Savir and W. H. Mcanny. Random Pattern Testability of Delay Faults. *International Test Conference*, 1989.

[12] G. L. Smith. Model for Delay Faults Based Upon Paths. *International Test Conference*, 1985.

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

[14] Ivan E. Sutherland, Robert F. Sproull, and Ian Jones. Standard Asynchronous Modules. Technical Memo 4662, Sutherland, Sproull and Associates, 1986.

[15] J. A. Waicukauski, E. Lindbloom, B. Rosen and V. Iyengar. Transition Fault Simulation. *IEEE Design & Test of Computers*, April 1987.