Synchro-Tokens: Eliminating Nondeterminism to Enable Chip-Level Test of Globally-Asynchronous Locally-Synchronous SoC's

Matthew W. Heath, Wayne P. Burleson
University of Massachusetts Amherst
{mheath, burleson}@ecs.umass.edu

Ian G. Harris
University of California Irvine
harris@ics.uci.edu

Abstract

Globally asynchronous locally synchronous (GALS) clocking applied to a system-on-a-chip (SoC) results in a design in which each core is a synchronous block (SB) of logic with a locally generated clock. Inter-core communication is asynchronous and controlled by wrapper logic around the cores. The nondeterministic synchronization used by most GALS architectures makes chip-level silicon debug and functional test difficult and costly. Deterministic GALS methodologies make dataflow assumptions which are only valid for a very limited set of applications. This paper describes a novel deterministic GALS methodology called "synchro-tokens" whose parameterized wrappers are flexible enough to be useful for a wide range of applications while supporting synchronous debug and test methodologies such as 1149.1 and P1500. The validation of determinism, estimation of area overhead, and analysis of performance impact are detailed.

1. Nondeterminism

A system specification defines a sequence of states and outputs which must be produced in response to a given input sequence. A correct implementation must conform to this specification. If the specification includes don't-care bits and partially ordered sequences, there may be many possible responses which a correct implementation may exhibit. A *deterministic* implementation always chooses the same correct response sequence. A *nondeterministic* one, on the other hand, randomly chooses a correct response sequence which may differ when the input sequence is applied to multiple copies of the chip or repeatedly applied to one copy of the chip.

The principal sources of nondeterminism are mutual exclusion elements and their close cousins arbiters and synchronizers. As shown by the waveforms for a synchronizer in Figure 1, the output sequence of these circuits depends on the relative order of input transitions, which is in turn sensitive to variables such as clock frequencies, clock skew and jitter, process variation, and noise, all of which are increasing concerns in deep submicron CMOS. These circuits make the fully asynchronous and GALS systems in which they are used nondeterministic. These circuits are also vulnerable to metastability, the condition where a bistable state is neither 0 nor 1 for a period of time. Metastability is a special case of nondeterminism which occurs when the temporal separation of the input signal transitions is very small, and the lack of it does not imply determinism.

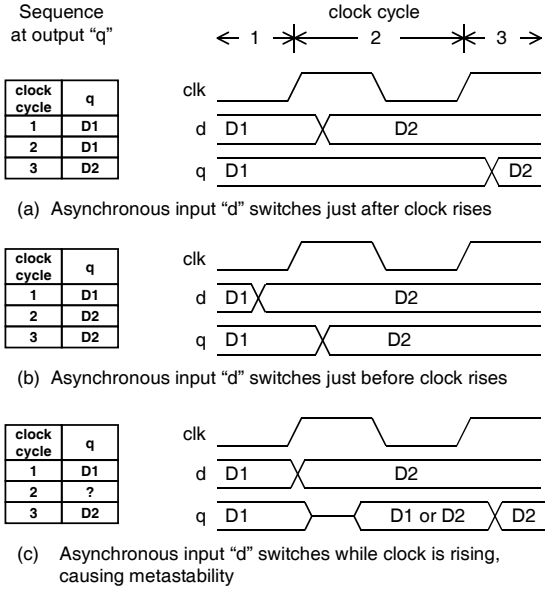

Figure 1. Nondeterminism in a synchronizer.

A synchronous design methodology which disallows the use of nondeterministic circuits produces deterministic systems. Any nondeterminism arising from setup or hold time violations is considered an error even if the resulting

This work was funded in part by NSF Grant No. 0204134, SRC Task 1075 and Intel Corporation.

sequence conforms to the higher-level specification. Although each SB in a GALS system is deterministic, the use of synchronizers and/or arbiters in a SB wrapper makes the interface between SBs nondeterministic. In any system, the precise transition times of each state bit may vary, so that the total state of the system at a particular time instant may be non-unique. However, it is the *unique sequence of states* which is the hallmark of deterministic behavior.

Nondeterminism negatively impacts debug and test by making the known good response of the system non-unique. In a GALS system with hundreds of asynchronous bits switching for thousands of clock cycles, the number of possible state sequences explodes combinatorially. Finding all possible traces consumes test creation time and is not feasible for sequential ATPG or manually-written tests. Storage of the possible responses costs die area (for BIST) and/or tester memory [1]. Comparing test results with all possible responses costs test time. If a fault effect maps to some other correct response, coverage is lowered. Divide-and-conquer test modes in which the synchronous and asynchronous components are decoupled and tested separately [2] do not enable functional testing and silicon debug at the chip and board levels [3]. Waiting for the test to reach a naturally deterministic state before checking the response provides insufficient observability. Event-based testers [4] which do not map signal transitions to specific clock cycles are still challenged by non-unique sequences. Protocol-checking testers might be able to address the non-unique sequence problem, but today's synchronous testers do not have such a capability, and the capital cost of replacing them is prohibitive.

2. Previous Work

Most existing GALS methodologies are nondeterministic because they use arbiters or synchronizers. Some sample all SB inputs with synchronizers with internal metastability detectors which stop the local clock until the metastability resolves itself [5] or which do not update their outputs if the metastability persists for too long [6]. Others use bundled data signaling and arbitrate between incoming requests and the local clock in a variety of ways: using the clock as a non-persistent arbiter input [7], generating a clock disable signal [8], or inserting an arbiter directly into the ring oscillator [9] [10].

Chapiro showed how to design a GALS *escapement organization* [11] without synchronizers and arbiters, instead using handshake signals to control a stoppable clock. Prior to the cycle during which a handshake is expected, the local clock remains enabled and insensitive to the state of the handshake signal. At the prescribed time, clock control is passed to the handshake signal. If the handshake has already occurred, clock control is immediately passed back to the synchronous logic, leaving the clock enabled. Otherwise, the clock synchronously stops until the handshake occurs. Because the clock enable interrupts the ring oscillator instead of simply gating its output, the handshake can restart the clock asynchronously with no runt pulses and return control to the synchronous logic. Because there is no decision to wait for an asynchronous signal or proceed with another local clock cycle, there is a deterministic relative order of asynchronous transitions and clock edges, and thus deterministic state sequences in the synchronous logic. An escapement design of a DSP chip [12] completes all local processing of a previous input data word, stops the clock, and waits for the next asynchronous data word to appear at its inputs. Determinism was neither a stated goal nor a recognized benefit of the design. This GALS methodology is useful only with low bandwidth communication between SBs.

Synchronizers can be avoided during steady-state operation by propagating data through a self-timed FIFO between the two SBs as in STARI [13]. To prevent the FIFO from asynchronously becoming empty or full, it is initialized to roughly half full and data is added to it and removed from it every cycle. The SB clocks are derived from a common source in order to be frequency-matched. Since the skew between the two SB clocks is absorbed inside the FIFO, each end always appears to be synchronized to the local clock. Another unintentionally deterministic methodology, it constrains the rates of output data production and input data consumption.

3. Making GALS Deterministic

In general, for a GALS system to be deterministic, each SB must receive each transition on each of its asynchronous inputs during a local clock cycle which is known in advance. A transition must not be recognized if it occurs earlier than expected, and the local clock must stop to wait for a transition if it occurs later than expected. Of course, such complete knowledge is never available in practice; indeed, its existence would imply that the inputs carry no information and thus aren't even needed! Fortunately, however, this knowledge can be inferred for all inputs if it is available for select inputs and if the timing relationship between those and all other inputs is known.

First, the inputs of a SB can be divided into two sets: data signals which carry information in their logic levels and handshake signals which carry information in their transition times. The data signals can then be bundled to the handshake signals, with timing verification used during design to ensure that the logic level of a data signal

at the time of a transition of its associated handshake signal is deterministic.

Second, all handshake signals with a common source SB and a common destination SB can be bundled to a single master handshake signal. Again, careful design ensures that the values of all bundled handshake signals at the time of a transition of the master handshake signal are deterministic.

Finally, the master handshake signal can control the stoppable clock in an escapement organization.

Optionally, self-timed FIFOs can be used to pipeline the asynchronous communication channel and allow multiple data words to be transferred between SBs with each master handshake. However, care must be taken to prevent a FIFO which has been emptied from asynchronously becoming non-empty or a FIFO which has been filled from asynchronously becoming non-full. The most straightforward safeguard is to make access to the FIFO mutually exclusive for the two SBs, using the master handshake signal to control which is enabled.

4. Synchro-Tokens System Architecture

Synchro-tokens is a novel deterministic GALS methodology which adds parameterized wrapper logic to the SBs as shown in Figure 2A. Data is transferred between SBs through asynchronous communication channels which may be pipelined with self-timed FIFOs. Each channel has its own request and acknowledge handshake signals which accompany arbitrarily wide bundled data words. For each pair of SBs with at least one communication channel between them, a token ring with a node inside each SB's wrapper acts as the master handshake signal. Each node counts local clock cycles to determine when the token is expected to arrive and when it should depart. A SB may have any number of nodes, and each node may be associated with any number of communication channels propagating data in either direction. One SB in the system is designated as the Test SB and one or more SBs are designated as I/O. These SBs are synchronized to and communicate with the environment (a board or a tester) without any intervening wrapper logic. Standards 1149.1 and P1500 can be implemented with the Test SB and self-timed scan chains whose heads and tails are synchronized to the test clock. Custom debug and test hooks can take advantage of FIFOs and token rings for controllability, observability, and clock manipulation.

4.1. Wrapper Logic Design

The wrapper logic, which consists of a node for each token ring, asynchronous data ports, and one stoppable clock, is shown in Figure 2B.

The stoppable clock is a ring oscillator whose frequency can be digitally controlled with either variable delay inverters or a clock divider circuit on its output.

The node is a synchronous state machine clocked by

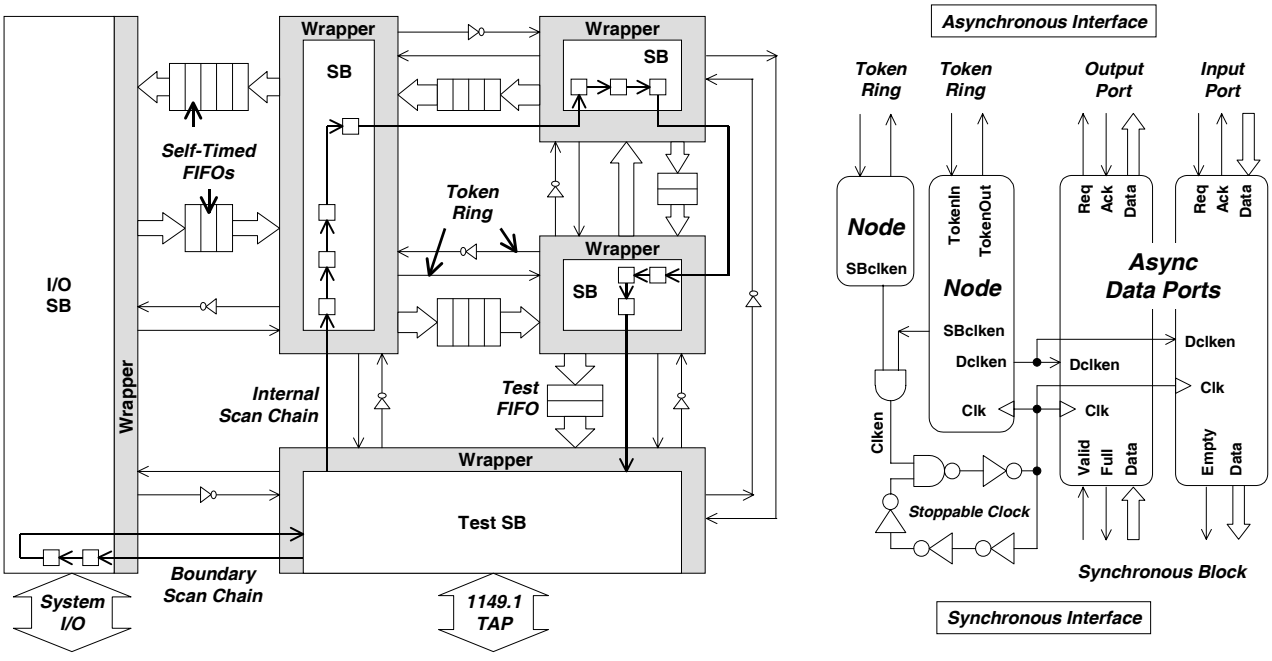

Figure 2: A. Synchro-tokens system architecture. **B. Wrapper logic.**

the SB's stoppable clock. The node connects to the token ring through the TokenIn and TokenOut signals. The node produces a data port clock enable, Dclken, and a stoppable clock enable, SBclken. The SBclken signals from all nodes in the SB are ANDed together so that the clock stops when any node de-asserts its SBclken.

Each node contains a pair of decrementing counters, called the *hold counter* and the *recycle counter*, which control the arrival and departure of the token. Each counter is parallel loadable from a dedicated register, which in turn may be downloadable from ROM bits, fuses, or directly from the tester. The hold counter determines how many local clock cycles the node holds the token before passing it to the other node on the token ring. While the token is being held, both SBclken and Dclken are asserted. The recycle counter determines how many local clock cycles after passing the token to the other node it expects to get the token back. While the token is recycling, SBclken is asserted but Dclken is not. This allows multiple SBs to operate concurrently and other data ports of the same SB to be enabled if their nodes are holding their tokens.

The operation of the node state machine is illustrated in Figure 3. When the incoming token has arrived (A) and the recycle counter reaches zero (B), the Dclken signal enables the node's associated data ports (C). The hold counter decrements by one for each local clock cycle (D). When the hold counter reaches zero, it immediately presets to its original value (E), the token is passed (F), and the data ports are disabled (G). The recycle counter then decrements by one for each local clock cycle (H). If the token has not yet returned by the time the recycle counter reaches zero, SBclken is de-asserted (I), synchronously stopping the local clock (J). When the token returns (K), the clock is asynchronously restarted (L). A late token stops the clock not only to the associated node but to the entire SB, even if there are other nodes which are holding (M) or recycling.

The data ports process the handshakes needed to ensure that each transmitted data word is received exactly once. While the associated node is holding that ring's token, data exchange between the SB and the asynchronous communication channel is permitted but not required. An input data port receives asynchronous requests and informs the SB when it is empty. An output data port produces asynchronous requests when valid synchronous data is available and informs the SB when the channel is full. Each stage of a FIFO must be able to complete a four-phase handshake within one local clock cycle of the transmitter or sender SB. Additionally, because data port handshakes are bundled to the token, data must propagate through a FIFO fast enough that data added to its tail just before the departing token disables the output port reaches the FIFO's head before the token enables the input port.

4.2. Debug and Test Features

The core of the Test SB is a Test Access Port (TAP) and associated controller which is 1149.1 compliant. The clock of the Test SB is provided by the tester through the TCK pin. The test clock has two modes of operation, similar to [14]. In Interlocked Mode, tokens passing through the Test SB may stop the clock; data exchange

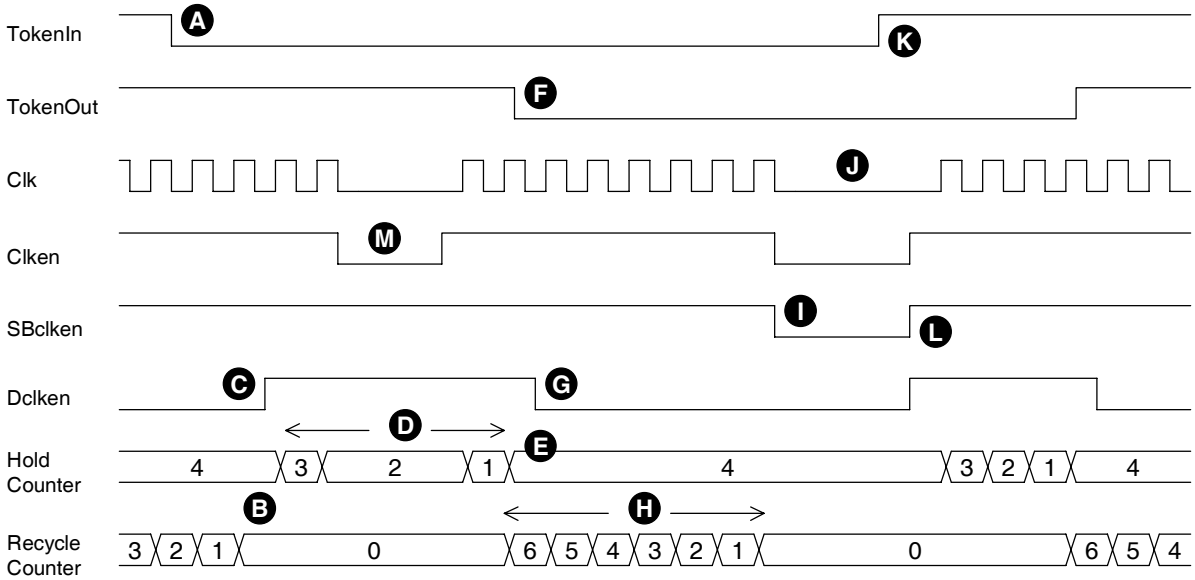

Figure 3. Waveforms illustrating operation of the node state machine.

between the tester and the mission mode logic is deterministic. Interlocked Mode is best suited for on-tester debug and production test. In Independent Mode, the operation of TCK and the flow of tokens through the Test SB have no effect on each other, but communication with mission mode logic is nondeterministic. Independent Mode is appropriate for off-tester usage of TAP public instructions and for mission mode.

Self-timed shift registers can be used for the boundary scan chain, P1500 registers in the core wrappers, internal scan chains for ATPG or BIST, or controls for other custom features. Adding several empty stages to the tail of the chain allows both ends of the chain to be synchronized to TCK. Making the hold, recycle, and clock frequency registers in each system SB accessible through a scan chain facilitates system performance tuning and clock frequency shmooing to find critical paths within SBs.

System clocks can be stopped while TCK is in Interlocked Mode by holding tokens indefinitely in the Test SB and waiting for all of the recycle counters in the system to reach zero and deterministically stop the local clocks. The granularity of these natural breakpoints can be increased - all the way to single stepping if desired - by adding token rings between more of the system SBs and the Test SB, and by changing the values of the hold and recycle registers. After the system clocks have stopped, the asynchronous scan chains can be used to deterministically read and write system state.

5. Results

The deterministic behavior of synchro-tokens was demonstrated using Verilog, a language which is able to specify nonzero delays and concurrent events. The test case was a system composed of three SBs and six FIFOs. Nominal values for FIFO delays, token ring delays, and local clock frequencies were chosen such that the token returned exactly when expected – never early and never late. Scenarios in which one or more of the delays could change to 50%, 75%, 150%, or 200% of their nominal values were simulated. The data sequences on each SB's I/Os were monitored for the first 100 local clock cycles and compared with the data sequences associated with the nominal delay settings. In all 16,000+ simulations, all data sequences were found to match exactly. However, when the synchro-tokens control logic was bypassed by keeping the data ports and local clocks always enabled, the data sequences nondeterministically mismatched on all local clock cycles.

The area overhead of synchro-tokens has been approximated using a gate-level model of the wrapper logic and layouts from a 0.25-micron cell library [15]. Summing the areas of the library cells used in the wrapper logic and using the average area of a 2-input gate as the unit of measurement, the models shown in Table 1 have been developed. A comparison with another GALS implementation should not include the data ports (which are always needed to ensure error-free transmission of asynchronous data) or the FIFO stages (which are always optional). Since there is just one pair of nodes for each pair of communicating SBs, the system-wide area overhead is reasonably low.

Table 1. Synchro-tokens area models.

Component	Area (2-input gates)
Data port	13 + 4.5*(number of data bits)
FIFO stage	4 + 4.5*(number of data bits)
Node	136

The impact of synchro-tokens on throughput and latency is due to each channel being accessible by only one SB at a time. For a worst-case analysis, synchro-tokens can be compared with STARI [13]. In the synchro-tokens system, let the hold register value of both nodes be H, and let the recycle register value of both nodes be R (the smallest value which prevents the local clock from stopping due to a late token). In both systems, let the period of both clocks be T and the propagation delay of one FIFO stage be F. Also in both systems, let the FIFO depth be H, so that the FIFO depth equals the hold register value of the synchro-tokens system and the FIFO depths of both systems are equal.

The throughput of the STARI FIFO is 1 data word per cycle, while that of the synchro-tokens FIFO is at most $H/(H+R)$. The synchro-tokens system can match the throughput of STARI by increasing the channel width by a factor of at least $(H+R)/H$ and providing hardware within the SB to synchronously queue data while the data port is disabled. This is an area/performance tradeoff.

The latency L_{STARI} of the STARI FIFO, which is kept half full, is the FIFO delay for the empty half of its stages plus one data word per clock for the full half of its stages:

$$L_{STARI} = F*H/2 + T*H/2 \qquad (1)$$

In this analysis, the synchro-tokens FIFO is repeatedly filled by the transmitter and emptied by the receiver. Its latency $L_{SYNCHRO}$ is the sum of the time from when data is ready at the output port until the token is passed, plus the token delay (approximately equal to the FIFO delay), plus the time until data reaches the input port:

$$L_{SYNCHRO} = T*(R+H+1)/2 + F*H + T*(H+1)/2 \qquad (2)$$

Subtracting (1) from (2), and making T as small as possible (that is, $T=F$, so that a four-phase handshake completes within one clock cycle), the latency penalty of synchro-tokens compared with STARI is:

$$\Delta L = T*(R+2H+2)/2 \quad (3)$$

This difference can be minimized by decreasing T and H (recall that R is already a minimum value), causing the token to be passed more frequently but decreasing the throughput. Another option, available for synchro-tokens but not for STARI, is the removal of the FIFO stages. The latency difference is then:

$$\Delta L' = T*(R+2)/2 \quad (4)$$

Its performance impact notwithstanding, the benefit of synchro-tokens is its ability to be optimized for different dataflow profiles and its deterministic behavior even if the actual data fails to follow the expected profile.

A synchro-tokens system may deadlock if there is a cyclic dependency among a set of SBs in which each SB has stopped its clock to wait for a late token [17]. Whether or not deadlock occurs is deterministic; thus, no detection or recovery methodology is needed. A set of deadlock-preventing design rules has been formally derived. While the details are beyond the scope of this paper, a sufficient but not necessary condition for preventing deadlock is, for each SB, the sum of the hold and recycle register values of all of its nodes being equal. This ensures that the local clock never stops while any node in the SB is holding its ring's token.

6. Conclusion and Future Work

Synchro-tokens, a novel GALS design methodology, has been presented. Its deterministic behavior facilitates debug and test, while its parameterization makes it useful for a wide variety of dataflow profiles. Compatibility with IEEE Standards 1149.1 and P1500 and other common test and debug methodologies has been shown. The deterministic behavior and the test and debug features have been validated in Verilog. Its area overhead has been shown to be modest. Its performance impact has been analyzed in terms of its parameters.

Future research includes SPICE simulations of the synchro-tokens control logic and further performance studies. Formal methods need to be applied to prove deterministic behavior. Performance-improving methodology enhancements also need investigation.

7. References

[1] Y. Zorian, E. Marinissen, and S. Dey. "Testing Embedded-Core Based System Chips". *Proceedings of the 1998 International Test Conference*, pp 130-143.

[2] F. Gurkaynak, T. Villiger, S. Oetiker, N. Felber, H. Kaeslin, and W. Fichtner. "A Functional Test Methodology for Globally-Asynchronous Locally-Synchronous Systems". *ASYNC 2002*.

[3] D. Josephson, S. Poehlman, and V. Govan. "Debug Methodology for the McKinley Processor". *Proceedings of the 2001 International Test Conference*, pp 451-460.

[4] J. Katz and R. Rajsuman. "A New Paradigm in Test for the Next Millennium". *Proceedings of the 2000 International Test Conference*, pp 468-476.

[5] F. Rosenberger, C. Molnar, T. Chaney, and T.-P. Fang. "Q-Modules: Internally-Clocked Delay Insensitive Modules". *IEEE Transactions on Computers*, Vol. 37, No. 9, September 1988, pp. 1005-1018.

[6] W. S. VanScheik and R. F. Tinder. "High Speed Externally Asynchronous / Internally Clocked Systems". *IEEE Trans. on Computers*, July 1997, pp. 824-829.

[7] S. Kim and R. Sridhar. "Hierarchical Synchronization Scheme Using Self-Timed Mesochronous Interconnections". *1997 IEEE ISCAS*, pp. 1824-1827.

[8] W. Lim. "Design Methodology for Stoppable Clock Systems". *IEE Proceedings*, Vol. 133, Part E, No. 1, January 1986, pp 65-69.

[9] K. Yun and A. Dooply. "Pausible Clocking-Based Heterogeneous Systems". *IEEE Transactions on VLSI Systems*, Vol. 7, No. 4, December 1999, pp. 482-488.

[10] J. Muttersbach, T. Villiger, and W. Fichtner. "Practical Design of Globally-Asynchronous Locally-Synchronous Systems". *ASYNC 2000*, pp. 52-59.

[11] D. Chapiro. "Globally-Asynchronous Locally-Synchronous Systems". PhD Thesis, Stanford University, Report No. STAN-CS-84-1026, Oct. 1984.

[12] P. Nilsson and M. Torkelson. "A Monolithic Digital Clock-Generator for On-Chip Clocking of Custom DSP's". *IEEE JSSC*, May 1996, pp. 700-706.

[13] M. Greenstreet. "Implementing a STARI Chip". *Proceedings of the 1995 IEEE ICCD*, pp. 38-43.

[14] D. Bhavsar, D. Akeson, M. Gowan, and D. Jackson. "Testability Access of the High Speed Test Features in the Alpha 21264 Microprocessor". *Proceedings of the 1998 International Test Conference*, pp. 487-495.

[15] J. Sulistyo and D. Ha. "Developing Standard Cells for TSMC 0.25um Technology under MOSIS DEEP Rules". Department of ECE, Virginia Tech, Technical Report VISC-2002-02, April 2002.

[16] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-a-Chip". *JETTA*, Vol. 18, April 2002, pp. 213-230.

[17] R. Javagal, A. Datta, and S. Ghosh. "Detecting Deadlocks in Distributed Systems". *IEEE ISCAS, Vol. 2*, 1991, pp. 1013-1016.