International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 6, Issue 6, June 2017 **ISSN 2319 - 4847**

A Deterministic behavior of Globally Asynchronous Locally Synchronous Microprocessor Architecture

[1]Suma Gurulingaiah Charantimath, [2]Ravi S M

[1]Assistant professor, Electronics and Communication Department PDIT College, Hospet, Karnataka, India.

[2]Assistant professor, Mechanical Department SIT College, Mangalore, Karnataka, India

Abstract

The globally-asynchronous locally-synchronous (GALS) architecture called "synchro-tokens" which exhibits deterministic state and output sequences. This deterministic behavior facilitates industrial validation, debug, and test methodologies which rely on predictable and repeatable system behavior. The 'synchro-tokens' architecture uses token rings for handshaking and self-timed FIFOs for pipelined interconnect. Local counters keep track of how long a token is held and the elapsed time since it was last released to ignore early tokens and to stop the local clock to wait for late tokens. Because no synchronizers are used, there is zero probability of failure due to metastability. Architectural parameters, such as FIFO sizes, counter values, and clock frequencies, offer a great deal of flexibility for tuning the system performance.

1. Introduction

1.1. Motivation for GALS

Clock generation and distribution on microprocessors is becoming more challenging with advances in VLSI technology. Higher levels of integration and deeper pipelines on larger dies increase the total clock load. More clock buffers are Required, increasing the clock distribution latency. These buffers introduce more skew due to reduced manufacturing control of shrinking geometries. Higher frequencies give rise to more power supply fluctuations and more cross-coupling, and this noise increases clock jitter. All of these effects lead to more clock power. Furthermore, as skew and jitter become a higher percentage of shorter clock cycle times, the fraction of time available for logic evaluation is reduced.

Many recent microprocessor designs address the clock design challenge by relaxing the amount of skew which is tolerable in early stages of the clock distribution and compensating for the variation at each lower-level clock domain. The Alpha processor strives for zero skew with DLL-based phase locking, while the Pentium 4 processor uses a programmed nonzero inter-domain skew. In both cases, traditional timing analysis is used to verify that setup and hold time requirements on inter-domain signals are satisfied.

These methodologies are similar to the globally-asynchronous locally-synchronous (GALS) design style, in which the system is partitioned into synchronous blocks (SBs) of logic which communicate with each other asynchronously. The key difference is that true GALS architectures allow arbitrary skew between clock domains and use some form of synchronization for inter-block communication. Unfortunately, these synchronization strategies are often a source of nondeterminism, which greatly complicates validation, debug, and test.

1.2. Nondeterminism in GALS Systems

A system is nondeterministic if there are multiple possible sequences of states and outputs with which it may correctly respond to a given input sequence. Nondeterminism is not necessarily indicative of a faulty design, since an implementation is considered correct as long as it conforms to its higher-level specification.

Synchronous systems are typically designed to be deterministic. The next state and outputs are uniquely determined by the current state and inputs. All signals which are sampled by clocks are designed through worst-case timing analysis to be stable at their final logic value long enough before the clock edge that the sampling state element's setup time is satisfied.

Similarly, sampling the states and outputs of a SB in a GALS system with its local clock will produce deterministic state and output sequences in response to a given input sequence to the SB. In most GALS methodologies, however, asynchronous inputs to SBs are captured by synchronizers, so that the relative order of input transitions and clock transitions is unpredictable. This makes the input sequence, and therefore the state and output sequences, of the SB nondeterministic. For one SB input signal and one clock edge, there are two possible next states. In a GALS system with hundreds of asynchronous bits switching for thousands of clock cycles, the

number of possible state sequences combinatorial explodes. As a result, the system output sequence may differ when the input sequence is applied to multiple copies of the same design or when the input sequence is repeatedly applied to a single instance of the design.

Metastability, the condition where the output of a synchronizer is neither 0 nor 1 for a period of time, is a special case of nondeterminism which occurs when the time separation of the signal and clock transitions is very small . While metastability is also undesirable, the lack of it does not imply deterministic behavior.

A deterministic GALS system must handle the synchronization of inter-SB signals such that the input sequence presented to the SB is unique despite variation in clock skew, clock frequencies, and interconnect delays. However, since the skew between clocks in different SBs is uncontrolled, the total state of a deterministic GALS system at any instant in time is not unique, even though the sequences in each SB are unique.

1.3. Nondeterminism in a Processor

Nondeterminism can be observed in a GALS implementation of an out-of-order processor core which was implemented in Verilog, an environment which is able to simulate concurrency and nonzero delays. The processor core consists of a register file and four ALUs. Each ALU can perform the functions add, subtract, multiply, and move (copy). The register file has two read ports which are used simultaneously by a single ALU to read its operands. An arbiter assigns a static priority to each ALU and grants access to the register file's read ports to the ALU with the highest priority request. The register file also has one write port, managed by a separate arbiter, through which an ALU writes its result. Out-of-order execution is supported with a scoreboard, which controls four of the stages of an instruction's life cycle: issue, read operands, execute, and write result. The system consists of five synchronous blocks: one for each ALU and one for the register file and scoreboard. While this partitioning may not be practical in terms of area or performance, it allows nondeterminism to be seen easily at the behavioral level. All five clocks in the system run at the same frequency, although this is not generally true of GALS systems.
Consider the following in-order instruction sequence. Instructions are named I1 - I5. Registers are named R1 - R7. The destination register is always the first argument in the list, followed by one or two source registers.
I1: ADD R3, R1, R2
I2: MUL R5, R3, R4
I3: SUB R4, R2, R1
I4: MOV R6, R3
I5: ADD R4, R3, R2

The architectural spec for the processor defines a partial order of register read and write events to ensure the avoidance data hazards. These include RAW hazards between I1 and {I2, I4, I5}, a WAR hazard between I2 and I3; and a WAW hazard between I3 and I5. The architectural spec does not impose any constraints on the relative order of independent events, such as accesses to different registers.

Tables 1, 2, and 3 show three possible traces of the execution of the above instructions generated by varying clock phases and handshake wire delays. Each column corresponds to an instruction, and each row corresponds to a cycle of the register file / scoreboard SB clock. Table entries indicate the clock cycle on which each instruction stage completes. Table 1 is used as a baseline against which other traces may be compared.

Table 2 shows the effect of increased delay on the asynchronous handshake wire which is asserted by the ALU executing instruction I3 to indicate to the scoreboard that the result is ready. I3's execution and write stages are postponed by 1 clock cycle, and the WAW hazard between I3 and I5 postpones I5's entire execution sequence by 1 clock cycle. The cycles during which I2 and I4 write are unchanged. Likewise, the relative order in which all instructions write is unchanged.

Table 3 shows the effect of changing the clock phase of the ALU executing instruction I4. Because less time is spent on the synchronization of the handshakes, I4 finishes execution 1 scoreboard-clock cycle early. Since I4's ALU is no longer competing with I2's higher priority ALU for access to the write bus, the arbiter allows I4's write to occur before I2's, changing the sequence of writes.

The final state of the register file following the execution of all instructions is identical in all three scenarios. The out-of-order processor thus conforms to the architectural spec by correctly executing the instruction sequences, even though its intermediate sequences and cycle-by-cycle behavior vary due to clock skew and wire delays.

Table 1: Baseline trace of instruction execution

Cycle	I1	I2	I3	I4	I5
1	Issue				
2	Read	Issue			
3			Issue		
4			Read	Issue	
5	Exec				
6	Write				
7		Read	Exec		
8			Write	Read	
9					Issue
10					Read
11		Exec		Exec	
12		Write			
13				Write	Exec
14					Write

Table 2: Slower handshakes with I3's ALU

Cycle	I1	I2	I3	I4	I5
1	Issue				
2	Read	Issue			
3			Issue		
4			Read	Issue	
5	Exec				
6	Write				
7		Read			
8			*Exec*	Read	
9			*Write*		
10					*Issue*
11		Exec		Exec	*Read*
12		Write			
13				Write	
14					*Exec*
15					*Write*

Table 3: Clock phase difference in I4's ALU

Cycle	I1	I2	I3	I4	I5
1	Issue				
2	Read	Issue			
3			Issue		
4			Read	Issue	
5	Exec				
6	Write				
7		Read	Exec		
8			Write	Read	
9					Issue
10				*Exec*	Read
11		Exec		*Write*	
12		Write			
13					Exec
14					Write

1.4. Impact on Validation, Debug and Test

Nondeterminism makes simulation-based validation more expensive. The simulator must choose among multiple possible next state and output values. Simulation must be repeated for many different choices to ensure that the design conforms to the spec regardless of the nondeterministic outcome. Trying to avoid this cost by validating only individual SBs risks missing bugs associated with complex system-level behaviors.

Nondeterminism thwarts the use of test techniques which perform cycle-by-cycle comparisons of observed and expected response sequences, such as the clock gating validation. Comparing Table 2 with Table 1, for example, results in a

mismatch in the state of I3's destination register in cycle 8. If the test response analyzer adapts to the difference by postponing the entire expectation by one clock cycle, the writes of I2 and I4, which occur on schedule, will cause mismatches. If no adjustment is made, the write of I5 will cause another mismatch. In either case, it may not be clear whether the mismatch is the result of excessive delay on an asynchronous signal (which is acceptable) or a critical path violation within a SB (which could cause an unacceptable deviation from the spec for some other instruction sequence). Observing the system only after the test reaches a deterministic point, e.g. after all active instructions have completed, may provide insufficient observability. Only architectural state, such as the contents of the register file, would be eligible for observation since other internal state is not included in the spec. Observation points may be few and temporally distant, making root-cause analysis very difficult.

Event-based testers can handle a limited amount of nondeterminism by processing signal transitions on pins which need not be mapped to specific clock cycles. However, this approach is not effective for scan tests which shift out internal state captured on a specific clock cycle. It is also inapplicable to tests in which the sequence of events is changed as in Table 3, where the state of the register file after cycle 11 is one which is never reached by the expectation in Table 1.

Nondeterminism precludes the use of many powerful silicon debug techniques. Waveform acquisition using optical probing relies on a deterministic response to lock onto the trigger transition each time through the loop. Shmoo plotting uses output sequence mismatches to identify the boundaries of acceptable operating regions. Much of the debug of the McKinley processor is performed using a tester which stays synchronized with the internal state of the chip rather than a system platform in which asynchronous system events such as memory refreshes and interrupts cause nondeterministic behavior.

There is a high simulation and test application time cost associated with generating all possible correct responses for each test pattern. Storage of those responses on-chip for BIST costs precious die area, while off-chip storage requires either a large, expensive, high-speed memory or a further increase in test time for repeated memory reloads. If real faults map to an alternative valid output sequence, there may be a decrease in fault coverage and/or an increase in escape rate.

1.5. A Deterministic GALS Architecture

The GALS architecture called "synchro-tokens" which eliminates nondeterminism by adding wrapper logic around the synchronous blocks to transform the asynchronous inter-block signals into deterministic input sequences. Tokens are passed between each pair of communicating synchronous blocks. Early-arriving tokens are ignored until a predetermined cycle of the local clock, and the clock stops to wait for late tokens. The architecture has no synchronizers and thus zero probability of metastability.

2. PREVIOUS WORK

A variety of GALS architectures have been proposed which use synchronizers, stoppable clocks, or both to sample asynchronous signals with a clock.

SBs whose inputs are sampled by flip-flops with internal metastability detectors and which stop the local clock until the metastability resolves itself have been proposed. Since the metastability can persist for an unbounded amount of time, some schemes don't update a state which remains metastable after a certain period of time. Some methodologies only synchronize data request lines and ensure through careful design that bundled data is valid before the request is asserted. They arbitrate between incoming requests and the local clock in a variety of ways: using the clock as a non-persistent arbiter input, generating a clock disable signal, or inserting an arbiter directly into the ring oscillator. However, all of these cases are nondeterministic because whether an input transition occurs before or after a particular local clock edge is unpredictable.

GALS architectures which achieve deterministic behavior do so by imposing constraints on their environments. Some require that incoming requests occur with such low frequency that all local processing completes and the local clock is stopped before the next request arrives. Others require that both SBs receive different fanout branches of the same global clock. Data is added to and removed from the FIFO at the same rate so that the FIFO never becomes empty or full. These methodologies are not applicable to blocks with high I/O bandwidths and sporadic workloads, such as a specialized execution unit in a microprocessor.

Chapiro described an escapement organization which uses handshaking signals following a known protocol to restart a stopped clock. Unlike data signals which may or may not transition before they are ready to be sampled again, all of the information in the handshaking signal is contained in its transition time rather than its logic level. Consequently, the asynchronous signal does not require synchronization and thus poses no risk of metastability.

A token ring communication protocol known as FDDI includes counters at each node which keep track of the length of time the token has been held and the length of time since the token was last released.

3. SYNCHRO-TOKENS SYSTEM ARCHITECTURE

As shown in figure 1, a synchro-tokens system consists of a collection of SBs surrounded by wrapper logic and connected with asynchronous communication channels and token rings. A functional unit containing a synchronous module and an asynchronous wrapper logic, shown in figure 2, consists of token ring nodes, asynchronous interfaces, and a stoppable clock. One or more SBs are designated as I/O SBs and are synchronized to and communicate with the environment (a board or a tester) without any intervening wrapper logic.

The asynchronous communication channels transport data between pairs of synchronous blocks. These channels are optionally pipelined with self-timed FIFOs to improve performance and/or to avoid the need for wave pipelining. At each end of a channel is an asynchronous interface; this piece of wrapper logic converts between synchronous valid bits and pulsed request and acknowledge handshakes. Data is transmitted using a bundled data signaling convention.

Each pair of communicating SBs has a token ring with a node in each SB's wrapper logic. A single token ring regulates the operation of all asynchronous communication channels in both directions between the two SBs. An asynchronous interface is enabled only while the associated token ring node in its SB is holding the token. The communication channel is designed such that the propagation delay of the token is no faster than that of the data. This scheme effectively synchronizes the data to the clock of SB whose interface is enabled. It prevents a transmitting SB from adding data to an empty channel and producing a request which reaches the receiver on a nondeterministic cycle of its local clock. It also prevents a receiving SB from removing data from a full channel and producing an acknowledge which reaches the transmitter on a nondeterministic cycle of its local clock. An n-stage self-timed FIFO in the channel allows up to n words of data to be exchanged per token cycle.

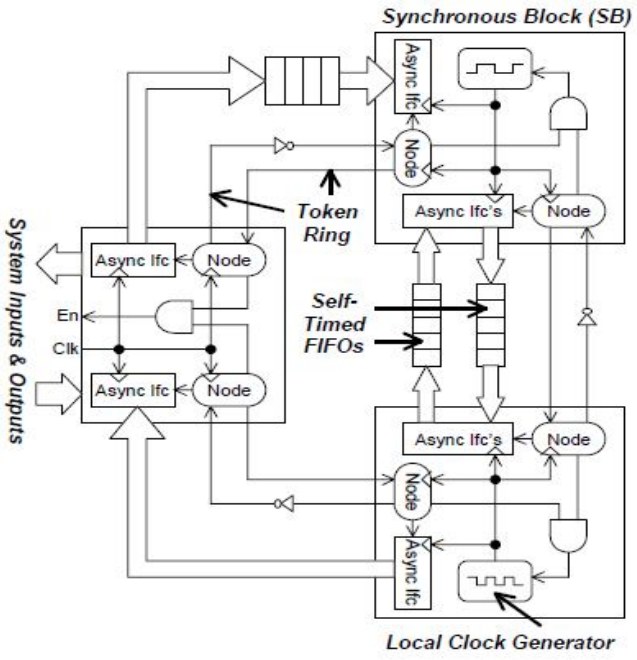

Figure 1: Synchro-tokens system architecture

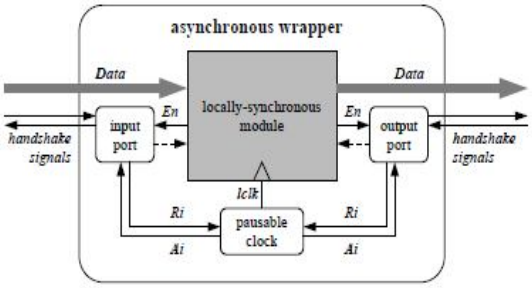

Figure 2: A functional unit containing a synchronous module and an asynchronous wrapper

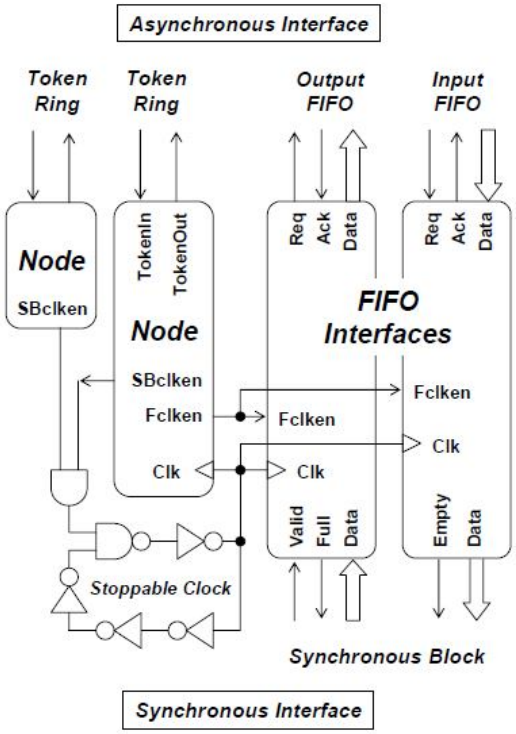

Figure 3: Wrapper logic: Nodes, FIFO interfaces, and a stoppable clock

One of the two connections which form a token ring must be inverting so that there are an odd number of inversions around the ring; this characteristic of all handshaking loops is needed for sustained oscillation. In the synchro-tokens methodology, the token rings use transition signaling in a two-phase handshaking protocol.

The node is a synchronous state machine clocked by the SB's stoppable clock, the frequency of which can be digitally controlled with either variable delay inverters or a clock divider circuit. The node connects to the token ring through the TokenIn and TokenOut signals. The node produces a FIFO clock enable, Fclken, for its associated channels which gates a branch of the stoppable clock before it reaches the asynchronous interface. The node also generates an enable for the stoppable clock itself, SBclken; the enables 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, each of which is parallel loadable from a dedicated register, which may in turn be downloadable from an on-die fuse array. The "hold counter" and register control how long the node holds the token before passing it to the other node on the token ring. The "recycle counter" and register control how long after passing the token to the other node it expects to get the token back.

When the incoming token has arrived and the recycle counter reaches zero, the interfaces of the node's associated asynchronous interfaces are enabled. The hold counter decrements by one for each local clock cycle. When the hold counter reaches zero, the token is passed and the channel interfaces are disabled. The recycle counter then decrements by one for each local clock cycle. During this recycle time, local processing continues and access is granted to any other communication channel whose associated node is holding its ring's token. If the token has not returned by the time the

recycle counter reaches zero, the clock to the entire SB is synchronously stopped. When the late token eventually returns, the clock is asynchronously restarted. This scheme ensures that SB input data is received on a deterministic local clock cycle.

4.CONCLUSION

A novel methodology for the design of globally-asynchronous, locally-synchronous systems called "synchro-tokens" has been presented. Such systems are deterministic, a property which facilitates validation, debug, and test. This deterministic behavior has been demonstrated with an out-of-order processor core implemented in Verilog.

Much work remains before the feasibility of the synchro-tokens architecture can be demonstrated. A larger system with bigger and more SBs will enable studies of the area and performance impact of the synchro-tokens architecture. A schematic implementation will enable a study of the critical paths and a more detailed clock design. Development of validation, debug, and test methodologies which are compatible with existing synchronous tools and testers is also needed.

REFERENCES

[1] 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.
[2] W. S. VanScheik and R. F. Tinder. "High Speed Externally Asynchronous / Internally Clocked Systems". IEEE Transactions on Computers, Vol. 46, No. 7, July 1997, pp. 824-829.
[3] S. Kim and R. Sridhar. "Hierarchical Synchro-nization Scheme Using Self-Timed Mesochronous Interconnections". 1997 IEEE International Symposium on Circuits and Systems, pp. 1824-1827.
[4] W. Lim. "Design Methodology for Stoppable Clock Systems". IEE Proceedings, Vol. 133, Part E, No. 1, January 1986, pp 65-69.
[5] K. Yun and A. Dooply. "Pausible Clocking-Based Heterogeneous Systems". IEEE Transactions on VLSI Systems, Vol. 7, No. 4, December 1999, pp. 482-488.
[6] J. Muttersbach, T. Villiger, and W. Fichtner. "Practical Design of Globally-Asynchronous Locally-Synchronous Systems". 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000), pp. 52-59.
[7] P. Nilsson and M. Torkelson. "A Monolithic Digital Clock-Generator for On-Chip Clocking of Custom DSP's". IEEE Journal of Solid-State Circuits, Vol. 31, No. 5, May 1996, pp. 700-706.
[8] M. Greenstreet. "Implementing a STARI Chip". Proceedings of the 1995 IEEE International Conference on Computer Design, pp. 38-43.
[9] D. Chapiro. "Globally-Asynchronous Locally-Synchronous Systems". PhD Thesis, Stanford University, Report No. STAN-CS-84-1026, Oct. 1984.
[10] 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.
[11] W. S. VanScheik and R. F. Tinder. "High Speed Externally Asynchronous / Internally Clocked Systems". IEEE Transactions on Computers, Vol. 46, No. 7, July 1997, pp. 824-829.
[12] S. Kim and R. Sridhar. "Hierarchical Synchro-nization Scheme Using Self-Timed Mesochronous Interconnections". 1997 IEEE International Symposium on Circuits and Systems, pp. 1824-1827.
[13] W. Lim. "Design Methodology for Stoppable Clock Systems". IEE Proceedings, Vol. 133, Part E, No. 1, January 1986, pp 65-69.
[14] K. Yun and A. Dooply. "Pausible Clocking-Based Heterogeneous Systems". IEEE Transactions on VLSI Systems, Vol. 7, No. 4, December 1999, pp. 482-488.
[15] J. Muttersbach, T. Villiger, and W. Fichtner. "Practical Design of Globally-Asynchronous Locally-Synchronous Systems". 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000), pp. 52-59.
[16] P. Nilsson and M. Torkelson. "A Monolithic Digital Clock-Generator for On-Chip Clocking of Custom DSP's". IEEE Journal of Solid-State Circuits, Vol. 31, No. 5, May 1996, pp. 700-706.
[17] M. Greenstreet. "Implementing a STARI Chip". Proceedings of the 1995 IEEE International Conference on Computer Design, pp. 38-43.
[18] D. Chapiro. "Globally-Asynchronous Locally-Synchronous Systems". PhD Thesis, Stanford University, Report No. STAN-CS-84-1026, Oct. 1984.