Performance Enhancement in Phased Logic Circuits Using Automatic Slack-matching Buffer Insertion

Kenneth Fazel, Lun Li, Mitch Thornton
Southern Methodist University
Computer Science & Engr.
{kfazel,lli,mitch}@engr.smu.edu

Robert B. Reese
Mississippi State University
Electrical & Computer Engr.
reese@ece.msstate.edu

Cherrice Traver
Union College
Electrical & Computer Engr.
traverc@doc.union.edu

Abstract
A technique for automatic insertion of slack matching buffers for performance enhancement in the asynchronous design style known as *Phased Logic* (PL) is described. A description of how slack matching buffers can offer throughput increases in PL circuitry is presented and is supported through the use of a simulation tool developed for modeling the timing behavior of PL circuits. A description of the architecture of the simulator and its implementation is also discussed. Based on the analysis of results provided by the simulator and the topological characteristics of a PL circuit, an algorithm for automatic slack matching buffer placement is devised. Examples of the buffer insertion technique are given and the effectiveness of the technique is evaluated through a set of experimental results.

Categories and Subject Descriptors
B.6.1 [1] : Design Styles; B.2.2 [**Arithmetic and Logic Structures**]: Performance Analysis and Design Aids

General Terms
Performance, Design, Experimentation

Keywords
Phased Logic, Asynchronous, Slack Matching Buffer Insertion

1. Introduction
Asynchronous circuits have been proposed as an alternative to synchronous circuits based on the potential for decreased power dissipation, increased performance, reductions in *Electromagnetic Interference* (EMI), and other characteristics. Although asynchronous design styles have been the focus of research for several decades since the seminal work in [1,2] widespread usage has not been adopted by the design community. There are several reasons this has not occurred. In general, issues such as increased area and the lack of mature and standard design automation tools are major factors. Another reason that asynchronous circuits have not appeared in mainstream designs is that actual realization of increased performance has been elusive in many cases.

GLSVLSI '04, April 26-28, 2004, Boston, MA, USA.

Because the performance of asynchronous circuits is based on the fundamental fact that overall circuit throughput depends on average-case delay rather than worst-case delay (as is the case with synchronous circuits) many performance optimization techniques used for synchronous circuit design have little relevance to asynchronous design. This motivates the investigation of alternative performance optimization methods for asynchronous circuit design.

The work presented here describes a performance enhancement technique for a particular asynchronous design style referred to as *Phased Logic* (PL) [3]. Specifically, a method for the automatic insertion of *slack matching buffers* is described. Slack matching buffers perform no logical operation on the data signals. They serve as intermediate signal storage locations in a circuit and allow other portions of the circuit to continue to process available data signals, thus decreasing the number of localized "stall" situations. The idea of using slack matching buffers for asynchronous circuit performance enhancement is not new and has been described in past work[4,5]. In terms of PL, slack matching buffer insertion was identified as a viable performance enhancement method in [6]; however, an automated process for buffer insertion was not devised. In subsequent PL design efforts, slack matching buffer insertion was accomplished manually via ad hoc methods based on the designer's detailed knowledge of the circuit functionality [7,8].

The contribution of the work presented here is the development of a technique for automatic buffer insertions. The technique was devised through the use of a custom PL circuit timing simulator. This simulator models the performance characteristics of a PL circuit only and does not provide any functional information. Active portions of the circuit at any instant in time are represented as the flow of "tokens" representing the data, request, and acknowledge signals. This approach allows the simulator to have significantly faster runtimes as compared to a traditional functional simulator. The simulator is also efficient since it exploits the inherent structure present in PL circuits and is thus less general but more efficient than a generic Petri net modeling tool.

Based on the analysis of PL circuits using the simulator, an automated method for slack matching buffer insertion is formulated that allows buffers to be inserted based solely on the topology of the PL circuit. Because the automatic buffer insertion technique utilizes the token flow simulations, a crucial aspect of this work is the efficiency of the custom simulator tool.

The remainder of this paper consists of a brief overview of PL asynchronous circuits, a discussion of the PL token flow simulator, and a discussion of the automatic buffer placement technique. Examples of the performance enhancement of PL circuits based on buffer placement are given followed by a set of experimental results that illustrate the viability of the method.

2. Phased Logic

PL circuitry may be viewed as an implementation of micropipelines [9] with two-phase *Level Encoded Dual Rail* (LEDR) signaling [10]. A method for automatically mapping synchronous netlists to equivalent asynchronous circuits referred to as PL was devised in [3]. In the PL mapping process, techniques were identified to ensure preservation of functionality based on marked graph theory [11]. This mapping model was implemented using a fine-grain, self-timed approach [6] and was subsequently modified as a coarse-grain approach by replacing individual LEDR signals with bundled data LEDR-style signaling [7]. The coarse-grain approach allowed for significant area savings as compared to the fine-grain approach at the expense of requiring point-to-point delay matched internal busses.

The chief advantage offered by PL designs is the capability to provide performance improvement when comparing average case throughput to synchronous circuit throughput. In addition to the performance enhancement provided by automatic slack matching buffer insertion that is the focus of the work provided here, two other performance enhancement techniques have been developed and are referred to as *Early Evaluation* and *Time Borrowing* [7]. Another key feature that motivated research for the PL style of asynchronous circuit implementation is the lack of dependence of specialized synthesis tools. This was a motivating factor in pursuing further research in the PL style designs that were initially devised in 1994 as this was a well-known pitfall of most asynchronous circuit design approaches. More recently, other asynchronous design styles have emerged that are capable of using standard commercial synthesis tools to various extents [12, 13, 14].

3. Slack Matching Buffering

As defined in [4], the *slack* of an asynchronous pipeline is the number of tokens that may be placed in the pipeline before it stalls. Additionally, the modification of slack properties in asynchronous pipelines may change the performance of the circuitry; this type of modification is a well-known optimization technique referred to as *slack matching* in [5]. This essentially means avoiding the situation where one pipeline is stalled while waiting for another pipeline to catch up. In terms of PL, we wish to be able to adjust the property of slack in order to improve performance by maximizing available parallelism between various paths. The goal of this work is to characterize where slack matching buffers can improve performance and then to automatically insert them into the netlist.

4. PL Token Flow Simulator

Marked Graphs introduced in [11] are a specific type of Petri net which model resource allocation problems. The theory behind such graphs is used as a basis for the automated synchronous to PL-style asynchronous mapping technique. A notion of *tokens*, and *token firing*, is used to represent data movement within a digital system.

In order to better understand the behavior of token movement with respect to the topology of a marked graph, a simulator was developed. The simulator allows for direct manipulation of token placement as well as nodes. This allows for the setup of experiments to characterize the behavior of the placement of token buffers with a given topology.

As there are numerous Petri net based simulators and asynchronous circuit design tools [15], the contribution of this simulator is the ability to measure latency and throughput of a PL graph. To do this, we have to add source and sinks to the marked graph that represent overall token producers and consumers of the entire circuit. Whenever a source fires, it provides a tag along with the token it places on the output. The tag contains information such as firing time and source id. This tag will propagate through the netlist and join with other tags from other sources, if encountered, to form larger tags. Once the tag reaches a sink, information such as latency may be computed using the tag.

The simulator uses an event-driven approach. A queue is maintained that keeps track of nodes that are ready to fire. A node is ready to fire when all its inputs have a token. Before the simulation starts, the queue is populated with nodes that are ready to fire based on the initial token mapping. The nodes of a marked graph may be either transitions or places as described in standard Petri net theory. These correspond to gates and signals of a physical circuit. Due to the nature of physical circuits, gates and signals may both have non-zero delay. We allow the possibility for varying delay of these components in the simulator.

The approach for inserting slack matching buffers is reliant on throughput and latency information provided by the simulator. To get this information, several simulation cycles must be done. However, the data quickly converges to stable values that may be used for slack matching buffer insertion.

To justify the use of a simulator in a synthesis tool, the simulator must be very fast. Let us assume a marked graph with t transitions and p places. An upper bound for nodes ready to fire is $t+p$. So the simulator must perform $t+p$ firing operations. A firing operation consists of checking whether to add the node driven needs to go into a queue. We know that a place may only input into one transition, so that is one check. If we suppose the transitions form a t-complete graph, we need t-1 checks for each transition. These means $t(t\text{-}1)= t^2\text{-}t$ checks. In total, there are $t^2\text{-}t+p$ checks, giving a $O(t^2)$ complexity for firing checks per iteration. If we assume more realistic graphs, such as transitions with a maximum of a fanout of 4 we would get $O(4t+p)$.

For realistic circuits, each simulation cycle has linear time complexity. However we only need to run the simulator to get latency and throughput times, which converge fairly quickly. Empirical results suggest as few as 10 iterations may provide good results. Additionally, 10 iterations frequently take less than a second, even for large graphs.

5. Results

A gates' feedback wait time at cycle *i*, ω_i, is the difference between the time that all data tokens are available at the inputs of a PL gate and the time when all feedbacks are available. If all feedbacks are available before the data, then we say $\omega_i=0$. During a simulation a gate may fire *n* times. We denote a gate's average feedback wait time as $\Sigma\omega_i/n = \delta$. The rule to add token buffers depends on a gate's average feedback wait time.

A branch and bound procedure used to determine slack matching buffer placement is the following:

1. Run the simulator on a topology to get δ for each gate.
2. Collect the gates with high δ, call it set *C*.
3. Determine a candidate for buffering from *C*.
4. Add slack matching buffer and ensure the resulting topology is a marked graph
5. Re-run the simulator
   a. If the buffer does not improve performance, remove the buffer.
   b. If the buffer improves performance, keep the buffer.
6. Repeat steps until some threshold is met.

In step 3, we need rules to determine a candidate from the set *C*. The priority for selecting gates is based on:

- Feedback latency
- Fan-out

Feedback latency, $fl_{i,j}$, is the time for the *j*th feedback to become available after all data inputs have arrived on a given gate at cycle *i*. If the *j*th feedback arrives before all data inputs then $fl_{i,j}=0$. This means that $\omega_i=\max(fl_{i,j})$ for a given gate. Fanout is a good indicator of slack matching buffer placement since a signal that fans out will result in more feedbacks needing to return to the gate. This increases the odds that the gate will have to wait on a feedback.

For the gates that have a high waiting time on feedbacks, a method for determining if a slack-matching buffer would be beneficial is the following:

a. Find the cycle that shares the feedback and a data output of the gate
b. If the gate that the data output is driving is waiting for other data, then place a slack-matching buffer on the data output. If the output fans out, place the buffer on the path between the output and the next gate on the cycle.
c. Re-run the simulator to make sure throughput and latency have improved.

We will use the following examples to denote how these rules are applied. Let us examine figure 1 with latency L=4 and throughput T=4. Notice that D2 has a wait time of $\delta=2$. In addition it is a fan-out point of two pipelines. However, D2 cannot evaluate until the acknowledge signal is received from G3 and is thus stalled. This implies that the slack-matching buffer should be placed between D2 and G3. If we add a slack-matching buffer at the output of D2, as shown in Figure 2, the delay characteristics improve. One may notice that B is now waiting; however, insertion of another buffer adds no performance improvement. Another example is shown in Figure 3.

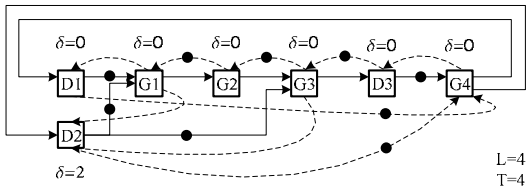

Figure 1: Example Topology with L=4 and T=4

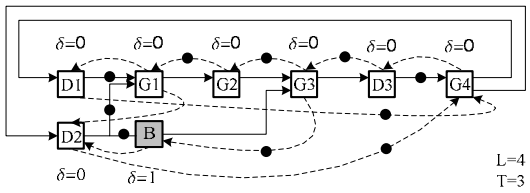

Figure 2: Example Circuit with Slack Matching Buffer

The latency is L=8 and average throughput is T=2.5 for the circuit represented in Figure 3. If we apply our rules, candidates for application of slack matching buffers at the outputs are G2, G5, and GD. It turns out that token buffers on G2 and G5 help performance, while a token buffer on GD does not. Figure 4 contains the topology of the circuit depicted in Figure 3 with the slack matching buffers included.

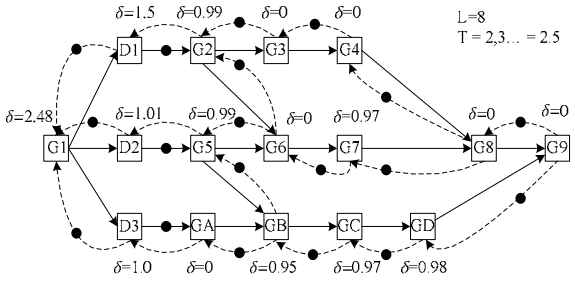

Figure 3: Second Example PL Circuit Topology

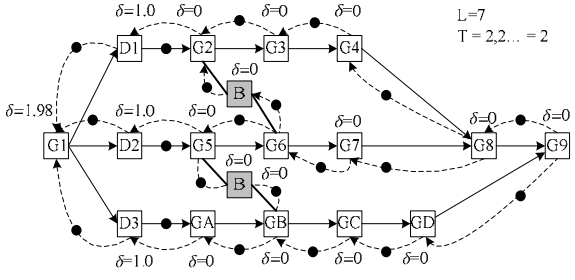

Figure 4: Second Example with Slack Matching Buffers

Topologies that do not benefit from slack matching buffer insertion are those that are purely serial in nature, or simple pipeline structures. An example of such a topology is shown in Figure 5. No matter where a buffer is placed, the throughput and latency do not improve. This is because there is no parallelism to take advantage of.

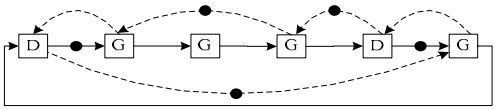

Figure 5: Single Pipeline Topology

Table 1 provides results of the slack buffer insertion algorithm. Six benchmark circuits topologies were chosen for these experiments. The circuits were initially mapped into PL circuits without any performance enhancements and were later modified to contain slack matching buffers using the technique described above. The table contains the name of the benchmark circuit, and, in columns two and three, the initial throughput T and latency L. After application of the buffer insertion technique, columns three, four, and five contain the number of buffers inserted and the new throughput and latency values. The overall percentage of throughput improvement computed as 100[(T-T')/T] is given in the last column of Table 1. Over the set of benchmarks, an average improvement in throughput of 21% was achieved.

Table 1. Experimental Results

Circuits	Original Latency L	Original Throughput T	Buffers added	New Latency L'	New Throughput T'	% of Throughput Improvement
4pipe	4	3.5	6	4	2	43%
Tb3	10	2.33	2	10	2	14%
T336	6	2.67	2	7	2	25%
Tb	4	4	1	4	3	25%
C17	4.5	2.5	1	4	2	20%
S27	11	6	1	13	5	17%
Average						21%

6. Conclusions

We have presented a technique for automatically determining the placement of slack matching buffers within a PL design. Details of the structure and implementation of a custom simulator for PL circuit throughput prediction are given and some simple examples illustrating the rules for automatic buffer insertion are provided. Experimental results are given that validate the approach and illustrate the performance enhancement that can be achieved using this technique. Through the examples and experimental results, it is shown that the automatic slack matching buffer insertion technique can enhance the throughput of a PL system.

7. References

[1] D. A. Huffman, "Design of Hazard -free Switching Circuits", Journal of the ACM, vol. 4, pp. 47-62, January 1957.

[2] D. E. Muller and W. S. Bartky, "A Theory of Asynchronous Circuits," *Proc. Int. Symp. on Theory of Switching*, vol. 29, pp. 204-243, 1959.

[3] D. H. Linder and J. C. Harden, "Phased Logic: Supporting the Synchronous Design Paradigm with Delay-Insensitive Circuitry," *IEEE Transactions on Computers*, vol. 45, pp. 1031-1044, 1996.

[4] A. M. Lines, "Pipelined Asynchronous Circuits," M.S. Thesis, Caltech, 1995.

[5] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee, "The Design of an Asynchronous MIPS R3000 Microprocessor," presented at Proc. 17th Conference on Advanced Research in VLSI, 1997.

[6] R. B. Reese, M. A. Thornton, and C. Traver, "Arithmetic Logic Circuits using Self-Timed Bit-Level Dataflow and Early Evaluation," *Proceedings of the 2001 International Conference on Computer Design*, pp. 18-23, 2001.

[7] R. B. Reese, M. A. Thornton, and C. Traver, "A Coarse-Grain Phased Logic CPU," *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems*, pp. 2-13, 2003.

[8] R. B.Reese, M. A. Thornton, and C. Traver, Technical Report, MSU.

[9] I. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, pp. 720-738, 1989.

[10] M. E. Dean, T. E. Williams, and D. L. Dill, "Efficient Self-Timing with Level-Encoded 2-Phase Dual-Rail (LEDR)," *Advanced Research in VLSI*, pp. 55-70, 1991

[11] F. Commoner, A. W. Hol, and A. Pneuli, "Marked Directed Graphs," *J. Computer and System Sciences*, vol. 5, pp. 511-523, 1971.

[12] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev, "Asynchronous Design Using Commercial HDL Synthesis Tools," *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems*, 2000.

[13] C.P. Sotiriou and L. Lavagno, "Desynchronization: Asynchronous Circuits from Synchronous Specifications", *Proceedings of the IEEE International SOC Conference*, 2003.

[14] A. Branover, R. Kol, and R. Ginosar, "Asynchronous Design By Conversion: Converting Synchronous Circuits into Asynchronous Ones," *Proceedings of the 2004 Design, Automation and Test Conference in Europe*, 2004.

[15] "Petri Nets Tools Database Quick Overview," vol. 2003: Petri Net World. http://www.daimi.au.dk/PetriNets/tools/quick.html.