Generalized Early Evaluation in Self-timed Circuits*

M. A. Thornton, K. Fazel, R. B. Reese
Department of Electrical and Computer Engineering
Mississippi State University
Mississippi State, MS 39762
{mitch,kf1,reese}@ece.msstate.edu

C. Traver
Department of Electrical Engineering
Union College
Schenectady, New York 12308
traverc@doc.union.edu

Abstract

Phased logic has been proposed as a technique for realizing self-timed circuitry that is delay-insensitive and requires no global clock signals. Early evaluation techniques have been applied to asynchronous circuits in the past in order to achieve throughput increases. A general method for computing early evaluation functions is presented for this design style. Experimental results are given that show the increase in throughput of various benchmark circuits. The results show that as much as a 30% speedup can be achieved in some cases.

1 Introduction

Phased Logic (PL) was devised in [14] as a design style for digital logic that produces delay-insensitve circuits that do not require a global clock signal. Implementation of synthesis tools for PL [19, 20] have resulted in a package that is well-suited for implementation in a *Field Programmable Gate Array* (FPGA) type of device. A major advantage of this design style is that designers may use synthesis tools and design styles that are currently used for the design of synchronous digital circuitry. Another recently developed method for mapping synchronous circuit designs to asynchronous ones is described in [8] although the target circuit technology is quite different from that used here. As is described in [14, 15], direct mapping from synchronous digital circuitry to PL circuitry is possible and has recently been implemented as is described in [19, 20]. Phased Logic is based upon *Level-Encoded Dual-Rail* (LEDR) signal encoding [6] and marked graph theory [5].

PL circuits address the important issues of clock and timing challenges since all PL circuits are guaranteed to operate as fast as possible. Furthermore, since global clocks are not used, issues related to clock distribution networks and clock skew correction are irrelevant. Nevertheless, the topology of the PL circuit does affect the overall achievable throughput. Synthesis tools that are designed for producing optimized clocked circuits do not take advantage of characteristics inherent in the PL design style although it is possible to directly map the output of such tools to functionally correct PL circuitry. As is discussed in [4], asynchronous circuit technology mappers should be designed to optimize average-case delay versus worst-case delay. Typically, synchronous circuit mappers are designed to minimize worst-case delay, thus the direct mapping from synchronous designs to PL based ones does not account for this criterion. Here, we describe a PL circuit optimization known as *Early Evaluation* (EE). Several other researchers have investigated similar approaches and have referred to them as *Speculative Execution*, *Eager Evaluation* or similar names [7, 17, 18, 21]. In addition, it is noted that similar methods referred to as "telescopic units" have been employed to speed-up synchronous pipelines [2, 3]. The novelty in the approach described here is that it is general and is not specifically designed to speed-up certain types of sub-circuits.

In comparison to other approaches, a self-timed FPGA based upon LUT3s and using LEDR encoding was presented in [12]. The PL gate design as shown in Figure 1 is a variation of the cell design used in [12]. In [12], three feedback inputs are included in each cell, so the Muller C-element [16, 24] has 6 inputs (3 data, 3 acknowledge). Also in [12] the circuit is used in the context of micropipelines [22] and self-timed iterative rings [9]. Both methods require a feedback signal for each output destination. The PL methodology removes the need for a feedback for every output signal destination as multiple output signals can be covered by the same feedback signal, and some output signals need no feedback signal if they are already part of a loop.

An FPGA-based architecture for asynchronous logic is also proposed in [11]. This FPGA architecture was aimed at accommodating a range of asynchronous design styles, and allowed for mixed synchronous and asynchronous designs. All signals were single rail. By contrast, the PL gate is only intended for supporting the PL design style and thus implements PL designs more efficiently than [11].

The asynchronous design methodology known as Null Convention Logic (NCL) also offers automated synthesis of asynchronous designs using commercial synthesis tools [13]. A restriction with the NCL methodology is that while the design can be coded in VHDL RTL, the user must write the RTL such that combinational logic and regis-

*This project was supported by the NSF under grants CCR-0098272, CCR-0000891, CCR-0097246.

ters are separated. In comparing physical implementation characteristics, NCL has some delay sensitivity between NCL gates whereas PL has no delay sensitivity between PL gates. Both NCL and PL use dual rail signals, where NCL uses a NULL/DATA/NULL encoding instead of LEDR. NCL has the same advantage of eliminating transient computations as PL, and does not have the disadvantage of the PL control overhead. The computation blocks in PL are the same as their synchronous counterparts with only a different control scheme, while NCL computation blocks are quite different from their synchronous equivalents.

In terms of *Early Evaluation*, several approaches have been proposed for asynchronous circuits in the past. Most of these are applicable only to special subcircuits within a given design, most notably addition subcircuits. A speculative-completion carry-lookahead adder is presented in [17]. This technique is well suited for an ASIC implementation but its reliance on matched delays and a bundled datapath makes an FPGA LUT4 implementation questionable. Applying general speculative completion techniques within a PL system is the subject of this work. Other approaches are described in [7, 18].

The remainder of the paper is organized as follows. A brief review of PL is given and a description of the concept of EE is provided. Next, the algorithm for determining EE functions and a description of the cost function is given. Finally, experimental results and conclusions are provided.

2 Phased Logic

A PL netlist can be thought of as a marked graph with data tokens flowing throughout the graph. Each data token has a phase that is either even or odd. A data token is represented by a dual-rail signal that uses LEDR encoding [1]. A PL gate has an internal state bit used to represent the gate phase and a phased logic gate fires whenever all of the phases of the inputs matches the internal gate phase. In [15] it was shown that for correct operation of a PL system, the marked graph equivalent had to be both live and safe. A live marked graph has an active token on each directed circuit of the graph and every signal must be part of a directed circuit. This essentially means that each directed circuit in the phased logic netlist must have at least one PL gate ready to fire at any time. A graph with a liveness problem will result in no token circulation, and hence no activity in the PL system. A safe marked graph is one in which each directed circuit has only one active token on it at a time. This means that there can only be one PL gate ready to fire within a given directed circuit.

2.1 PL Cell Structure

Figure 1 shows the structure of a PL gate as used in our implementation. This figure originally appeared in [23] and is currently being implemented as a custom cell for implementation in a prototype integrated circuit. A PL gate consists of a control circuit shown as a set of equivalence gates and a Muller-C element, a function circuit shown as a 4-input *Look-Up Table* (LUT4), two latches for storing the output LEDR signal and a small amount of control circuitry. The operation of a PL gate is such that when all valid input tokens have arrived, the Muller-C element toggles and causes the output of the LUT4 and a new gate phase to be latched.

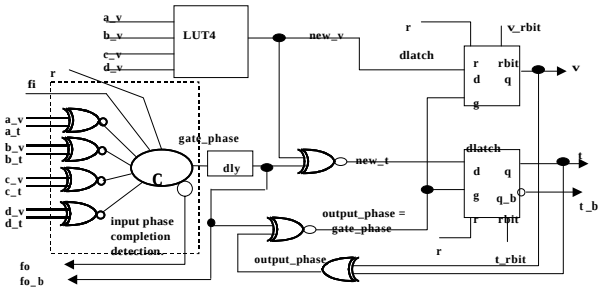

Figure 1. Phased Logic Gate Structure

The output of the PL gate is a dual-rail signal denoted as v and t signals in Figure 1. Both tokens and gates carry a phase that is referred to as even or odd. The phase of a signal is computed as $p = \overline{v \oplus t}$ with $p = 1$ denoting odd phase and $p = 0$ denoting an even phase. The phase of a gate is the output of the Muller-C element.

The v bit contains the logic value of the signal as normally used in single-rail systems. For this reason, the functionality of the LUT4 is the same as would be represented in a clocked sequential system mapped to FPGA cells containing LUT4 elements. The Muller-C element monitors the phases of all 4 input signals and toggles when the all have the same phase which in turn causes the output of the LUT4 to be latched and the feedbacks and gate phases to change. In this way, PL gates "fire" when all input signals have the opposite phase of the input signals.

Feedback signals are generated within each PL cell that can be used for producers of tokens and consumers of tokens. The feedback signals for token producers is used to acknowledge that new tokens may be produced for the PL gate and is computed as the inverse of the gate phase (inverse of the Muller-C circuit). This essentially creates a queue of unit depth at the input of the PL gate. The feedback signal for the consumer is used to signal that new tokens are available for processing and is the inverse of the phase of the current output token.

3 Early Evaluation

Each PL gate in a circuit can potentially be coupled with another PL gate in order to achieve *Early Evaluation* (EE). We refer to such PL gate pairs as "master" and "trigger" PL gates. Figure 2 illustrates an EE-PL gate pair. As is shown in Figure 2, a small amount of additional control circuitry is needed for an EE pair that consists of a pair of Muller C-elements and some control signals generated in

the trigger PL gate and used to cause the master PL gate to "fire" or "evaluate" before all valid inputs to the master PL gate have arrived. Details of the this implementation are given in [23] including a discussion of the preservation of a safe and live network when these elements are used.

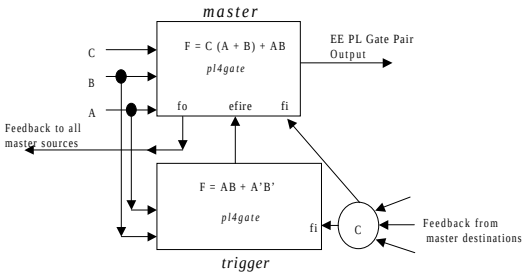

Figure 2. Early Evaluation PL Gate Pair

The idea behind the trigger PL gate is to extract a subfunction from the master PL gate that depends on a subset of faster arriving input signals only. When such a subfunction can be found, an output from the trigger gate can be used to cause the master gate to produce an output value before all inputs are present. Since all PL gates in the current implementation depend only on 4 input signals, all possible subfunctions with a support set of three or fewer inputs are evaluated and a cost function is applied to choose the trigger function with the best characteristics.

The cost function is based upon the degree of coverage that the trigger function provides versus the master Boolean function realized in the LUT4 and is weighted by the relative arrival times of the maximum-delay trigger versus master gate input signal. The arrival times are assumed to be equivalent to the maximum path length in terms of PL gates from the primary circuit inputs to the inputs of the PL gates. The cost function is given in Equation 1. The term *Coverage* is the percentage of minterms that are in common with the trigger and master function (both 0 and 1-valued). The higher this percentage is, the more often early evaluation can occur. M_{max} and T_{max} are the maximum delay of the input signals to the master and trigger PL gates respectively.

$$Cost = (\%Coverage) \times \frac{M_{max}}{T_{max}} \quad (1)$$

As an example, consider the truth table for a carry-out function of a full adder cell, $c(a+b)+ab$, as shown in the fourth column of Table 1. Since this function depends on three input signals, a search for the trigger function would consist of generating all candidate functions with support sets of $\{a\}$, $\{b\}$, $\{c\}$, $\{a,b\}$,$\{a,c\}$ and $\{b,c\}$. In column 5 of Table 1 a trigger function is shown, $ab + \overline{a}\overline{b}$, that is based on the support set $\{a,b\}$. Since the trigger function contains 4 minterms in common with $f^{ON} \cup f^{OFF}$ of the master function, an overall coverage of 4/8=50% is computed. Each time the trigger function evaluates to '1', the master gate can go ahead and evaluate even if the input signal c has not arrived since it's value is a don't care in these cases.

Table 1. Truth Tables for Master and Trigger Functions

a	b	c	Master	Trigger
0	0	0	0	1
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

Early evaluation for addition circuits is well known and has been exploited in several asynchronous design styles [7, 17, 18]. If the signal c represents the carry-in of a full-adder the trigger function in this example evaluates the presence of generate or kill signals in the carry-out circuitry and does not allow for EE if the carry propagate situation arises. For addition circuits this case is particularly advantageous since carry-in signals are the latest in arriving among the three inputs.

In the results presented here, we have generalized the notion of EE to work for any arbitrary master function. We search over all 14 possible support sets of 3 or fewer variables, compute the cost of each candidate trigger function and implement the best choice. Although the search is exhaustive, our restriction to a LUT4 in the PL gate allows for the approach to be practical. Candidate trigger functions are computed by processing the cube list representation of the f^{ON} and f^{OFF} functions for the master function. Table 2 illustrates this process. Since 2 cubes in Table 2 depend only upon master inputs a and b and each of those two cubes covers 4 of the 8 possible outputs of the master function, a coverage of 50% is computed for the trigger function $f_{trig} = ab + \overline{a}\overline{b}$. The trigger function is easily found as the function corresponding to the cube list given by $f^{ON}_{trig} = \{00-, 11-\}$.

The cost function is also weighted by the relative arrival times of the input signals to the master and candidate trigger PL gates. This is necessary since, unlike the case of the adder circuit, a large coverage of a potential trigger function may depend on slowly arriving signals and thus not be as effective in terms of a speed-up mechanism as using a trigger function with less coverage but based on faster arriving inputs.

4 Experimental Results

Several benchmark circuits were synthesized both with and without the use of the EE algorithm. The benchmarks

Table 2. Determination of Candidate Trigger Functions

Master Cube	Master Outputs	$\{a, b\}$ Coverage	Trigger Function
00-	0	2	1
010	0	0	0
100	0	0	0
11-	1	2	1
1-1	1	0	0
-11	1	0	1

used were the *International Test Conference* 1999 (ITC99) suite [10]. These are available in RTL level VHDL format and were synthesized using the Synopsys Design Compiler tool. The resulting EDIF netlist was then mapped to PL technolgy using the tool described in [20] and then post-processed for the inclusion of EE circuitry. For the 15 benchmarks presented in Table 3, an average speedup of over 13% was achieved with an average increase in the amount of circuitry for the EE gates resulting in 33%. In these results, EE circuitry was added to all PL gates where a speedup was possible. It is also possible to reduce the increase in area by requiring a candidate trigger function to have a cost value that exceeds some threshold. Thresholding the cost function allows for a tradeoff in area versus delay of a PL circuit.

Table 3 contains columns representing the description of the benchmark circuit, the number of PL gates required without EE, the number of EE gates when the circuit is synthesized using EE, the average delay with no EE, the average delay with EE, the difference in the delay values, the percent of area increase in terms of additional gates when the EE algorithm is applied and the percent decrease of delay when EE is used in the synthesis of the benchmark circuits. These results are based upon the average statistics of 100 simulations where the input vectors were randomly generated.

For each PL circuit, we determined the average delay time between the presence of a stable input vector and a stable output word. In a PL circuit, new values cannot be presented to the inputs until a stable output is generated for the current input values. Furthermore as is discussed in [19], delays are statistically distributed based on the value of an input vector and are not constant in PL circuits. To determine the delay values between the PL circuits with and without EE, we computed the average of the difference between the cycles for both circuits. Mentor Graphic's `qhsim` was used to simulate the PL VHDL we generated for each test bench.

Because a master/trigger pair of PL gates requires the use of an additional Muller-C element, some benchmarks suffered a slight degradation in overall delay values when the EE algorithm was applied. Overall, the EE algorithm resulted in a speedup for most of the benchmarks. Not surprisingly, those benchmarks with significant amounts of arithmetic circuitry tended to take more advantage of the EE algorithm since arithmetic circuits tend to be composed of addition circuits where EE techniques are known to perform well.

5 Conclusion

A generalized technique for increasing the throughput of PL circuits was presented. The technique works for arbitrary functions and is not dependent upon certain classes of subcircuits such as adders. The technique was implemented and run for several large benchmark circuits with an average speedup in excess of 13% with an average increase in circuitry of approximately 33%. Threshold values may be used such that no EE circuits are produced unless a candidate trigger function has some minimal cost. In this case the increase in area may be controlled and reduced.

This synthesis optimization results in local transformations of the logic network for overall circuit delay reduction. Future work will consist of identifying other transformations both local and global that also enhance performance.

References

[1] D. B. Armstrong, A. D. Friedman, and P. R. Menon. Design of asynchronous circuits assuming unbounded gate delays. *IEEE Trans. on Comp.*, 18, 1969.

[2] L. Benini, E. Macii, and M. Poncino. Telescopic units: Increasing the average throughput of pipelined designs by adaptive latency control. In *Design Automation Conf.*, pages 22–27, June 1997.

[3] L. Benini, G. De Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino. Automatic synthesis of large telescopic units based on near-minimum timed supersetting. *IEEE Trans. on Comp.*, 48(8):769–779, 1999.

[4] W.-C. Chou, P. A. Beerel, and K. Y. Yun. Average-case technology mapping of asynchronous burst-mode circuits. *IEEE Trans. on CAD*, 18(10):1418–1434, 1999.

[5] F. Commoner, A. W. Hol, S. Even, and A. Pneuel. Marked directed graphs. *Journal of Computer and System Sciences*, 5:511–523, 1971.

[6] M. E. Dean, T. E. Williams, and D. L. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (ledr). In *Advanced Research in VLSI*, 1991.

[7] A. DeGloria and M. Olivera. Completion-detecting carry select addition. *IEE Proceedings-Computer and Digital Techniques*, 147(2):93–100, 2000.

Table 3. Experimental Results Comparing the Use of EE in PL Synthesis

Description	PL Gates (no EE)	EE Gates	Avg. Delay (no EE) ns	Avg. Delay (w. EE) ns	Delay Diff. ns	% Area Increase	% Delay Decrease
FSM that compares serial flows	25	9	49	43	6	36%	12%
FSM that recognizes BCD numbers	4	0	18	18	0	0%	0%
Resource arbiter	78	25	49	50	-1	32%	-2%
Computer min and max	274	102	84	85	-1	37%	-1%
Elaborate contents of memory	322	136	98	88	10	42%	10%
Interrupt Handler	10	1	26	27	-1	10%	-3%
Count points on a straight line	240	95	87	67	20	40%	23%
Find inclusions in sequences	82	24	66	52	14	29%	21%
Serial to serial converter	74	23	46	45	1	31%	2%
Voting system	126	49	63	59	4	39%	6%
Scramble string with a cipher	275	112	132	93	39	41%	30%
1-player games(guess a sequence)	635	263	80	73	7	41%	9%
Interface to meteo sensors	141	44	56	51	5	31%	9%
Viper processor (subset)	3360	1565	332	207	125	47%	38%
80386 processor (subset)	5648	2611	336	184	152	46%	45%

[8] J. Ebergen. Squaring the FIFO in GasP. In *Advanced Research in Asynchronous Circuits and Systems*, pages 194–205, 2001.

[9] M. R. Greenstreet, T. E. Williams, and J. Staunstrup. Self-timed iteration. In C. H. Sequin, editor, *VLSI'87*, pages 309–322. Elsevier Science Publishers, 1988.

[10] Electronic CAD and Reliability Group. ITC99 Benchmark Set-Politecnio di Torino. http://www.cad.polito.it/tools/itc99.html, 1999.

[11] S. Hauck, S. Burns, G. Borriello, and C. Ebeling. An FPGA for implementing asynchronous circuits. *IEEE Design & Test of Comp.*, pages 60–69, 1994.

[12] D. L. How. A self clocked FPGA for general purpose logic emulation. In *Custom Integrated Circuits Conference*, pages 148–151, 1996.

[13] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Advanced Research in VLSI*, 2000.

[14] D. H. Linder. *Phased Logic: A Design Methodology for Delay Insensitive Synchrnous Circuitry*. Ph.D. Dissertation, Mississippi State University, 1994.

[15] D. H. Linder and J. C. Harden. Phased logic: Supporting the synchronous design paradigm with delay-insensitive circuitry. *IEEE Trans. on Comp.*, 45(9), 1996.

[16] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Int'l Symp. on Theory Switching Funct.*, pages 204–243, 1959.

[17] S. M. Nowick. Design of a low-latency asynchronous adder using speculative execution. *II Proc. Comput. Digit. The.*, 143(5):301–307, 1996.

[18] S. M. Nowick, K. Y. Yun, P. A. Beerel, and A. E. Dooply. Speculative completion for the design of high-performance asynchronous dynamic adders. In *Advanced Research in Asynchronous Circuits and Systems*, 1997.

[19] R. B. Reese, M. A. Thornton, and C. Traver. Arithmetic logic circuits using self-timed bit-level dataflow and early evaluation. In *Int'l Conf. on Comp. Design*, pages 18–23, 2001.

[20] R. B. Reese and C. Traver. Synthesis and simulation of phased logic systems. In *International Workshop on Logic Synthesis*, pages 255–259, Dana Point, California, 2000.

[21] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn, and D. Ferguson. Speedup of delay-insensitive digital systems using NULL cycle reduction. In *International Workshop on Logic Synthesis*, pages 185–189, Lake Tahoe, California, 2001.

[22] I. Sutherland. Micropipelines. *Comm. of the ACM*, 32(6):720–738, 1989.

[23] C. Traver, R. B. Reese, and M. A. Thornton. Cell designs for self-timed FPGAs. In *ASIC/SOC Conference*, pages 175–179, 2001.

[24] T.-Y. Wuu and S. B. K.Vrudhula. A design of a fast and area efficient multi-input Muller C-element. *IEEE Trans. on VLSI Systems*, 1(2), 1993.