Early Evaluation for Phased Logic Circuits Using BDDs and MVL*

Kenneth Fazel, Mitchell A. Thornton
Southern Methodist University
{kfazel,mitch}@engr.smu.edu

Robert B. Reese
Mississippi State University
reese@ece.msstate.edu

Abstract

Phased Logic (PL) is a design style for binary-valued asynchronous logic circuits. A performance enhancement known as Early Evaluation (EE) allows for increased throughput in PL circuits. PL circuits are produced using clocked circuit descriptions as input and then automatically mapping them into PL equivalents while adding optimization features. The PL mapping process requires initially partitioning the input circuit and then inserting additional control circuitry around each partition while removing the synchronous clock control. In the process of adding the EE performance enhancement, a special function known as a "trigger function" is extracted from the partitions. Here, we describe a method for finding candidate trigger functions using BDDs and a technique for combining multiple trigger functions to support a single circuit partition using Multiple-valued Logic (MVL). Experimental results show that these methods yield better coverage as compared to using a single trigger function.

1.0 Introduction

Asynchronous design styles can offer advantages over their clocked circuit counterparts. Some advantages include decreased power consumption, increased performance, more desirable EMI characteristics, and avoidance of clock distribution network design. The *Phased Logic* (PL) design style offers advantages in increased performance. Furthermore, PL has the advantage that the design methodology is the same as that used for clocked circuits resulting in no new techniques or specialized libraries required. This latter advantage leads to an efficient and fast method for producing PL circuits that is not available for other asynchronous design methodologies.

The basis of PL is an implementation of micropipelines [1] that utilizes a dual-rail signal encoding, *Level-Encoded Dual-Rail* (LEDR) [2]. In [3], this approach was generalized through the use of marked graph theory [4] and the development of circuit constraints that preserve liveness and safeness. Based on the results in [3], a PL mapping tool was implemented [5] that allows for a netlist of a clocked sequential circuit to be automatically transformed into a PL netlist utilizing cell designs based on those described in [6]. A performance enhancement technique known as *Early Evaluation* (EE) was described in [7] that allowed the mapped PL designs to have significant performance advantages as compared to their clocked counterparts. Several moderately large circuits have been successfully mapped into PL forms including the MIPs processor [8,9].

In mapping a clocked netlist to a PL netlist, the circuit is partitioned into a collection of subcircuits functionally described by master functions. The incorporation of EE requires the identification of a subcircuit that depends on a proper subset of inputs for the master function. This subcircuit is called the *trigger function*. The purpose of the trigger function is to produce a signal that indicates that the master function can evaluate only when a proper subset of input signals have arrived.

The work described here describes two enhancements to the EE optimization technique; 1) automatic determination of trigger functions for large master functions, and 2) the use of multiple trigger functions in support of a single master function. In the first enhancement, BDDs are used to find trigger functions for large master functions. In the second optimization, a method based on a *Multiple-valued Logic* (MVL) formulation is used to allow multiple trigger functions to support a single master function.

The remainder of the paper is organized as follows. Section 2 will provide an overview of the PL circuit structure and mapping technique with an emphasis on EE performance optimization. Section 3 will describe the problem of trigger function extraction and explain how the new technique based on the use of BDDs is implemented. In Section 4, we describe the idea of using multiple trigger functions to support a single master function and describe the underlying circuitry and how MVL is used to formulate the appropriate set of trigger functions. Section 5 contains the experimental results and their interpretation. Conclusions are provided in Section 6.

**These research results were obtained with support by the National Science Foundation under grants CCR-0098272 and CCR-0243365 and the Texas Advanced Technology Program under grant 003613-0029-2003.*

2.0 PL Background

PL is a self-timed design methodology that provides an automated translation of a clocked system in the form of D-flip-flops (DFFs) and combinational gates into a self-timed netlist of PL gates. The only global net in the resulting self-timed netlist is a reset signal. The PL netlist is a micropipelined system with two-phase control. Two distinct implementation technologies are supported, *fine-grain* and *coarse-grain*. The fine-grain approach uses a one-to-one mapping of gates in the clocked system to PL gates that use a 4-input Lookup-Table as the logic element with delay-insensitive dual-rail routing between gates. This technology forms the basis for the implementation of a self-timed FGPA. Because all routing between gates is delay-insensitive, there are no timing mechanisms external to a PL gate that can cause a failure due to timing. The coarse-grain approach maps groups of gates in the clocked netlist into a combinational compute function embedded in a PL block with bundled data signaling used between blocks.

The combinational compute function of a coarse-grain PL block can be implemented using a traditional standard cell library. The coarse-grain technology is an ASIC approach to the implementation of PL systems. All timing concerns in a coarse-grain implementation are block-to-block; there are no global mechanisms that can cause failure due to timing. If desired, delay insensitive signaling can be used between coarse grain blocks to remove timing uncertainty due to wire delays. This will add extra latency in the control path but this latency can be hidden if the coarse grain block delay is long enough.

2.1 Mapping Technique

The algorithm for mapping a clocked netlist to a PL netlist was developed in [3] and is summarized below:

a. All DFFs are mapped one-to-one to *barrier* gates in the PL netlist. The output phase of a barrier gate always matches the gate phase resulting in an active token at the output of a barrier gate upon reset.

b. All combinational compute blocks are mapped one-to-one to *through* gates in the PL netlist. The output phase of a through gate is always opposite the gate phase.

c. Single rail signals called feedbacks are added where necessary to ensure *liveness* and *safety* of the resulting marked graph. *Liveness* means that every signal is part of a loop that has at least one gate ready to fire. *Safety* means that a gate cannot fire again until all destination gates have consumed the output data. To ensure safety, all signals must be part of a loop containing at most one active token. Feedbacks cannot be added between two barrier gates because this would result in a loop with two active tokens violating the safety constraint.

d. If necessary, buffer-function through gates called *splitter* gates are inserted between barrier gates to provide a source and termination for feedback. This feedback signal is equivalent to an acknowledge signal in a micropipeline [1].

e. Feedbacks that originate from a barrier gate have an initial token on them (after reset) since all outputs from barrier gates have tokens. This implies that feedbacks from barrier gates must terminate on a through gate, hence the need for splitter gates in some circumstances.

f. A feedback that originates from a through gate and terminates on a through gate must have a token present after reset since the output of the destination through gate will not have an initial token.

g. A feedback that originates from a through gate and terminates on a barrier gate must not have an initial token since the output of the destination barrier gate will have an initial token.

More details and examples of the PL mapping process are available in [5, 8, 9].

2.2 Early Evaluation

Both fine-grain and coarse-grain PL implementations support a speedup mechanism known as *early evaluation* (EE) that can enable a PL system to outperform a clocked system. All micropipeline approaches suffer a performance degradation compared to clocked systems because the output latch latency of a micropipeline block is in the critical path. EE allows PL systems to overcome this performance penalty. Simulations of fine-grain and coarse-grain netlists of a MIPs-compatible 5-stage pipelined CPU mapped to four-input lookup tables (LUT4s) indicate a speedup of over 35% compared to equivalent clocked versions. Results in mapping other moderately large open cores have shown similar performance enhancements.

3.0 Trigger Function Definition

When master functions depend on a relatively small number of input signals, such as is the case for the fine-grain approach, it is feasible to search over all possible subsets of inputs for a suitable trigger function, however; as the support set of the master function grows, such an exhaustive search quickly becomes impractical. For this reason, the focus of the work described here is applicable only to the coarse-grain version of PL.

The role of a trigger function is to determine when the output of the master function can be deduced by the

currently arrived inputs, regardless of the other inputs that have not yet arrived. To illustrate this idea, consider the function $f = ab + ac + bc$. A truth table for this function is shown in Figure 1 with four entries emphasized in bold and italic font. The emphasized entries yield values of *f* that are only dependent upon variables *a* and *b*. A trigger function can be formulated that has an output of 1 when *f* may be evaluated based only upon the values of variables *a* and *b*. The truth table for this trigger function (f_{trig}) is given in Figure 2.

a	*b*	*c*	*f*
0	***0***	***0***	***0***
0	***0***	***1***	***0***
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	***1***	***0***	***1***
1	***1***	***1***	***1***

Figure 1: Truth Table of *f*

a	*b*	f_{trig}
0	0	1
0	1	0
1	0	0
1	1	1

Figure 2: Truth Table of f_{trig}

If *f* represents a master function then the trigger function f_{trig} can be extracted from *f* and the output of f_{trig} can be used as an additional input to *f* allowing it to go ahead and evaluate when only *a* and *b* are present at the inputs. This would allow for *f* to evaluate based on only two data inputs being present yielding a *coverage* value of 50% of the truth table entries of *f*. It is noted that other trigger functions could be formulated such as a trigger function only dependent upon variables *b* and *c*. This alternative trigger function would also yield a coverage value of 50%.

In order for trigger function usage to be effective in increasing PL circuit throughput, the dependent variable set for the trigger function should be comprised of variables representing signals that tend to arrive early. For this reason, a merit function is evaluated to choose appropriate variables for the support set of the trigger function [7]. This merit function allows the trigger function to be chosen based upon the coverage and the relative arrival times.
The merit function is:

$$Merit = (\%\ coverage) \times \frac{M_{max}}{T_{max}}$$

Where M_{max} is the maximum delay value among all signals in the support of the master function and T_{max} is likewise the maximum delay value among all signals in the support of the candidate trigger function.

3.1 Trigger Function Extraction

In the fine-grain PL mapping method, trigger function extraction is completely automated based on the technique described in [7]. In the work reported in [8], automated trigger function extraction was accomplished by evaluating the cost function for all possible variable subsets of master functions implemented using 4-input look-up tables. Exhaustive evaluation of all possible trigger functions was not a costly endeavor since only 14 different candidate trigger functions were evaluated for each master function that was dependent upon only 4 variables.

For the coarse-grain approach, master functions are utilized that can have large numbers of dependent variables, hence, exhaustive searches for appropriate trigger functions is impractical. As master functions can be quite large in terms of primary inputs, a convenient way to represent them is through the use of *Binary Decision Diagrams* (BDDs). This approach motivated us to develop a trigger function extraction method based on the BDD representation of the master function.

Because the trigger function must depend upon a proper subset of dependent variables of the master function, all cubes in a candidate trigger function correspond directly to the 1-paths and 0-paths in the master function BDD. Such paths are easily extracted from the master function BDD through a single traversal. In order to incorporate the timing constraint, we reorder the variables in the BDD such that those variables corresponding to minimum arrival time signals are first in the ordering. This approach effectively translates the process of extracting trigger functions to variable reordering of a BDD although we are not using the typical constraint of BDD minimization to perform reordering. In the event that the BDD representing the master function exceeds some preset size limitation, the master function can be partitioned into two new master functions and the process repeated. As an example, consider the BDD depicted in Figure 3 that corresponds to the example master function truth table in Figure 1. The circled paths correspond to the cubes in the trigger function that depend only upon variables *a* and *b* since those paths terminate without inclusion of variable *c*.

This idea is implemented as an algorithm that accepts a BDD representing a master function and with variables ordered according to the earliest arriving signals closer to the initial node and building another BDD that represents the trigger function.

4.0 Multiple Trigger Function Support

Circuits produced using the PL method reported in the literature have used one trigger function per master function. However, a master function may yield other trigger functions with equal cost that are dependent on different subsets of the input signals. These other trigger functions may cover EE opportunities that are not recognized by the originally chosen trigger function. Therefore, the use of more trigger functions will cover more early evaluation opportunities for a given master function.

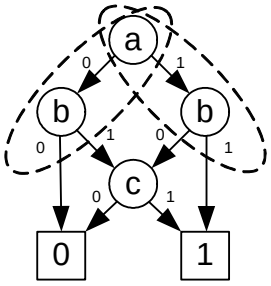

Figure 3: BDD Representation of Example Master Function

In the initial mapping of a PL netlist, the clocked netlist is mapped to the set of master functions corresponding to the partitioned subcircuits. EE is implemented by evaluating each master function for the possibility of improvement using EE. Whenever the cost value indicates the improvement can be achieved, the master function is replaced with a PL EE gate. A PL EE gate consists of the original master function PL cell augmented with another PL cell [10] (along with a small amount of control circuitry between the two). Figure 4 contains a diagram of the organization of a PL EE gate with a single trigger function and $n>m$. The box labeled "*Input Detect*" outputs a signal indicating that all dependent variables of f_{trig} have arrived.

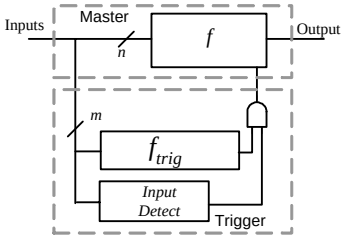

Figure 4: PL EE Gate with Single Trigger

In order to effectively utilize multiple trigger functions, the control mechanism between the master function and the multiple trigger functions must be augmented resulting in a new organization of a PL EE gate. A naive implementation of such a gate would be to replicate trigger function cells for each candidate trigger function and then use a large fan-in OR gate to concentrate their outputs into a single signal causing the master function to early evaluate as shown in Figure 5. While this resulting functionally is correct, this approach is too costly in terms of area since a considerable amount of circuitry in the trigger functions could potentially be shared. The unlabeled uppermost boxes represent the candidate trigger functions, t_i and the lowermost represent the respective "*Input Detect*" functions for each t_i.

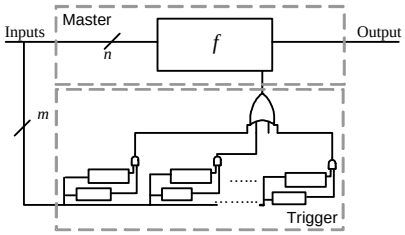

Figure 5: Naïve Implementation of Multi-trigger EE PL Gate

4.1 Formulation of Multi-trigger EE PL Gates

It is desired to use a single *super trigger function* by forming a function that is the union of the on-sets of each individual trigger function allowing for term sharing to be exploited. If the outputs of the corresponding C-Muller elements are also ORed together, as shown in Figure 6, "false" EE can occur. In Figure 6, the uppermost OR gate concentrates outputs of all possible trigger functions while the lowermost OR gate concentrates outputs of the respective "*Input Detect*" functions for each.

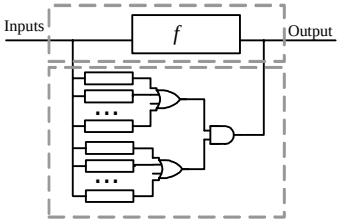

Figure 6: <u>Incorrect</u> Multi-Trigger EE PL Gate

A "false" trigger can occur since the trigger function should evaluate only when all dependent variables are present. However, an individual C-Muller element may evaluate when only a subset of variables of the super trigger function are present.

To illustrate this problem, consider a master function $f(a,b,c,d)$ with individual trigger functions $t_1(a,b,c)=ab+bc$ and $t_2(a,b,d)=abd$. If the organization in

Figure 6 were used, the super trigger function is formed as $t_{sup}=t_1+t_2=ab+bc+abd$ and the two C-Muller elements (supporting t_1 and t_2) evaluate when variables $\{a,b,c\}$ are all present (corresponding to t_1) while the second C-Muller element will evaluate when variables $\{a,b,d\}$ are present.

Suppose that the signals corresponding to variables $\{a,b,d\}$ have arrived with values $\{1,1,0\}$ respectively and that variable c has not yet arrived. In this case, the C-Muller element corresponding to trigger t_2 will evaluate although the trigger function t_2 has value 0. However, t_{sup} has a value of 1 causing the master function f to incorrectly evaluate early although the signal corresponding to variable c has not arrived. In such a case, a false trigger occurs causing the master function to evaluate before it should have.

To solve this problem, a notion of "which" variables in the support of t_{sup} have arrived is needed. With a single trigger function, this notion is not needed since the single C-Muller element will evaluate only when all inputs corresponding to the trigger function are valid. Our approach is to formulate the super trigger function such that the individual C-Muller elements are combined into a single *binary valued-function* that *depends on ternary-valued variables* resulting in t_{mvl}. The organization of this form of the EE PL gate is then depicted as shown in Figure 7.

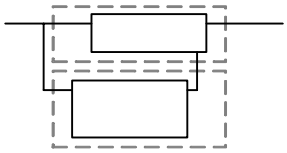

Figure 7: PL EE Gate with MVL-based Multiple Trigger Function Support

The notion of signal availability is incorporated into the PL EE Gate depicted in Figure 7 by using the dependent ternary-valued variable encoding shown in Table 1.

Table 1: Ternary Encoded PL Signals

Logic Value	Interpretation
0	Signal arrived with binary value 0
1	Signal arrived with binary value 1
2	Signal has not yet arrived

The MVL super trigger function t_{mvl} is formed as the disjunction of terms that represent each individual trigger function and the availability of their respective inputs. Each such term is the conjunction of the individual trigger functions and a corresponding expression indicating input signal availability. The expressions representing input availability effectively take the place of the C-Muller elements and are represented as MVL functions C_i.

Using the example shown above that precludes the organization as shown in Figure 6, the individual trigger functions and their corresponding C_i functions become:

$$t_1(a,b,c) = a^{\{1\}}b^{\{1\}} + b^{\{1\}}c^{\{1\}}$$

$$C_1(a,b,c) = \overline{a^{\{2\}}}\ \overline{b^{\{2\}}}\ \overline{c^{\{2\}}} = a^{\{0,1\}}b^{\{0,1\}}c^{\{0,1\}}$$

$$t_2(a,b,d) = a^{\{1\}}b^{\{1\}}d^{\{1\}}$$

$$C_2(a,b,d) = \overline{a^{\{2\}}}\ \overline{b^{\{2\}}}\ \overline{d^{\{2\}}} = a^{\{0,1\}}b^{\{0,1\}}d^{\{0,1\}}$$

Using these formulations, the super trigger function t_{mvl} is expressed as:

$$t_{mvl} = t_1C_1 + t_2C_2$$

$$t_{mvl} = a^{\{1\}}b^{\{1\}}c^{\{0,1\}} + a^{\{0,1\}}b^{\{1\}}c^{\{1\}} + a^{\{1\}}b^{\{1\}}d^{\{1\}}$$

It is now easy to see that the cube $a^{\{1\}}b^{\{1\}}c^{\{2\}}d^{\{0\}}$ corresponding to the scenario described previously will result in t_{mvl} evaluating to 0.

The actual implementation could be accomplished using MVL circuitry, or, by mapping back to a binary circuit where dual-rail lines are used for each variable. Although the binary-mapped version may appear to double the amount of required wiring, we note that signals are already present in dual-rail form in order to support the LEDR encoding used in PL circuits. This increase in wiring for the inputs of the super trigger portion of the EE PL gate is less overhead overall than would be present in the naive approach of ORing all individual single trigger function blocks as shown in Figure 5.

5.0 Experimental Results

Two sets of experimental results are reported here. In the first set, the effectiveness of extracting individual trigger functions from large master functions using BDDs is demonstrated. In the second set of results, enhancements obtained using a multiple trigger function for an EE PL gate are presented.

The BDD-based trigger function extraction experiment was carried out by constructing a BDD using the CUDD software and by using the mcnc benchmark circuits in `.pla` format. No variable reordering is applied to the BDDs as it was assumed that those variables higher in the assumed order correspond to earlier arriving inputs. Each output was considered to represent a single-output master function and trigger functions were extracted that depend upon the first $\lfloor j/2 \rfloor$ variables in the BDD ordering where j represents the total number of variables in the BDD. The choice of $j/2$ variables in the support of the trigger functions is arbitrary and the actual number chosen will depend upon the value of the cost function as selected by the designer.

Table 2 contains the experimental results. The computer runtime required to compute the trigger functions in Table 2 all required less than 1 ms. This is not surprising since the algorithm is merely a modified depth-first traversal, which is known to be $O(N)$, with N

being the number of vertices in the BDD. Column one contains the name of the benchmark circuit. Column two shows the total number of inputs/outputs. Columns three and four pertain to the first output bit. The third column contains the number of dependent variables in support of the trigger function (k) (maximum possible value of k is $\lfloor j/2 \rfloor$ and the minimum is 0). The fourth column contains the percentage of minterms in the on-set of the trigger function divided by the number of all those possible (ie. on-set minterms divided by 2^j). This latter value is a measure of the coverage value that the trigger function provides.

Table 2: Results for Single Triggers

name	in/out	k	Coverage
5xp1	7/10	0	0
addm4	9/8	5	0.781
majority	5/1	3	0.125
cordic	23/2	10	0.758
sao2	10/4	6	0.828
squar5	5/8	3	0.875
dist	8/5	4	0.625

In the second set of experimental results, super trigger functions are found and compared to single trigger functions using the method corresponding to that reported in [7]. The experiments were carried out by using the mcnc benchmarks to represent master functions with the first output being used to represent the master function output. Multiple trigger functions were generated using random subsets of variables. These trigger functions were then used to form an MVL trigger function. The results of these MVL trigger functions were then compared to the best single trigger function within the set of possible trigger functions. Table 3 contains these experimental results. The first column contains the name of the benchmark (master) function, the second contains the inputs/cubes of the master function, the third contains the inputs/cubes of the single trigger function found using the method in [7] the fourth contains the coverage achieved by the single trigger function, the fifth contains the inputs/cubes of the multi-trigger function, and the final (sixth) column contains the coverage provided by the multi-trigger function.

These results are given after *espresso* was used to minimize all functions after MVSIS version 1.1 [11] was used to map the t_{mvl} function back to binary form.

6.0 Conclusions

An improvement in the formation of EE PL gates is described where a BDD-based method allows for the extraction of trigger functions from large master functions is demonstrated as effective as compared to the exhaustive method described in [7]. The coverage provided by a trigger function was also enhanced by utilizing multiple trigger functions for the same master function and then using MVL methods to combine them into a super trigger function.

Table 3: Results for Multi-Triggers

name	master	single EE		MVL EE	
5xp1	7/7	3/3	0.75	6/3	0.75
Addm4	9/9	2/2	0.75	14/6	0.83
majority	5/5	3/2	0.63	6/2	0.63
cordic	23/143	8/10	0.55	18/33	0.59
sao2	10/10	6/7	0.81	14/14	0.84
squar5	5/2	3/3	0.88	8/4	0.94
dist	8/12	5/3	0.41	12/5	0.58

7.0 References

[1] I. Sutherland, "Micropipelines", *Comm. of the ACM*, Vol 32, No. 6, 1989, pp. 720-738.

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

[3] D.H. Linder and J.C. Harden, "Phased Logic: Supporting the Synchronous Design Paradigm with Delay-insensitive Circuitry." *IEEE Tran. on Comp.*, Vol. 45, No 9, 1996, pp. 1031-1044.

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

[5] R.B. Reese, M.A. Thornton, and C. Traver, "Arithmetic Logic Circuits using Self-timed Bit-Level Dataflow and Early Evaluation", *Proc. ICCD 2001*, Sept. 2001, pp. 18-23.

[6] C. Traver, R.B. Reese, M.A. Thornton, "Cell Designs for Self-timed FPGAs", *Proc. ASIC Conf.*, Sept. 2001, pp. 175-179

[7] M.A. Thornton, K. Fazel, R.B. Reese, and C. Traver, "Generalized Early Evaluation in Self-Timed Circuits", *Proc. DATE*, March 2002, pp. 255-259.

[8] R.B. Reese, M.A. Thornton, and C. Traver, "A Fine-grain Phased Logic CPU", *Proc. ISVLSI*, Feb. 2003, pp. 70-79.

[9] R.B. Reese, M.A. Thornton, and C. Traver, "A Coarse-grain Phased Logic CPU", *Proc. ASYNC*, May 2003, pp. 2-13.

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

[11] MVSIS version 1.1, http://www-cad.eecs.berkeley.edu/Respep/Research/mv