A Coarse-Grain Phased Logic CPU

Robert B. Reese, *Member*, *IEEE*, Mitchell Aaron Thornton, *Senior Member*, *IEEE*, and Cherrice Traver, *Senior Member*, *IEEE*

Abstract—This paper describes an asynchronous design tool flow known as Phased Logic that converts a clocked design into an asynchronous design implemented as a micropipeline using two-phase control and bundled data signaling. Example designs include variations of a double-precision floating-point clipping operation mapped to two commercial standard cell libraries (0.18μ and 0.13μ) and a five-stage pipelined MIPs-compatible integer unit mapped to the 0.13μ library. The design style includes a feature known as early evaluation, which is a generalized form of bypass, that allows the self-timed design to recover some of the inherent latch delay penalty in micropipelines.

Index Terms—Automatic synthesis, self-timed, asynchronous, pipelined processor, micropipelines.

1 Introduction

Proponents of self-timed design have repeatedly touted advantages such as reduced EMI signatures, lower power, and control scalability over traditional clocked designs. However, there is no asynchronous methodology that can claim mainstream success; instead, asynchronous designs have been relegated to niche applications. The reason for this lack of wide acceptance has been that most asynchronous methodologies have one or more disadvantages that outweigh any advantages. Barriers to adopting an asynchronous methodology are:

- Area—delay insensitive approaches have a 2X to 3X area increase due to the required dual-rail routing.
- Performance—micropipeline approaches add extra latency in the critical path, resulting in a performance penalty. Other approaches use fine grain cells that are more complex and slower than typical standard cells, resulting in a performance slowdown.
- Tool Support—many asynchronous methodologies require new languages and/or new tool chains, requiring a substantial investment in design engineer retraining.
- Custom libraries—many asynchronous methodologies require custom cell libraries and cannot use the same commercial standard cell libraries used for clocked designs.

Phased Logic (PL) [1], [2] is a self-timed design methodology that avoids significant penalties in the above areas and offers the typical advantages of other asynchronous approaches. This paper describes the first PL netlists that have been mapped to a commercial standard cell netlist. The example designs are a five-stage pipelined MIPS-compatible integer-unit [3] and a double-precision floating-point clip operation. The CPU implementation uses a standard cell library and register file generator designed for a 0.13μ technology. The clip operation is mapped to both a 0.13μ library and a 0.18μ library (library vendors are not identified by prior agreement). No extra cells were added to either standard cell library even though this could have increased the efficiency of the PL implementation. A PL netlist is a two-phase micropipelined system that uses bundled-data signaling. A PL netlist is produced by an automated translation from a clocked netlist which allows the designer to use familiar languages and tools for producing the clocked design. The performance penalty in the micropipeline due to the additional latch latency on blocks can be reduced by a technique known as early evaluation. This technique allows blocks in the micropipeline to evaluate on arrival of a subset of the inputs, increasing the amount of parallel activity in the micropipeline, which improves performance. The use of bundled data signaling keeps the area penalty to approximately a 50 percent increase in active cell area, which does not include the area required by the global clock network.

2 Phased Logic

A PL netlist is a two-phase micropipeline system whose distributed control network is automatically generated from a clocked netlist. This transformation uses marked graph theory [4] to produce a PL netlist that is both live and safe. The control network only replaces the global clock network; the original logic of the clocked design is retained. Two distinct implementation technologies are supported, *fine-grain* and *coarse-grain*. The fine-grain approach [5] uses a one-to-one mapping of gates in the clocked system to PL gates that use a 4-input Lookup-Table (LUT4) 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

- *R.B. Reese is with the Electrical and Computer Engineering Department, Mississippi State University, Box 9571, Mississippi State, MS 39759. E-mail: reese@ece.msstate.edu.*
- *M.A. Thornton is with the Department of Computer Science and Engineering, School of Engineering, Southern Methodist University, PO Box 750122, Dallas, TX 75275-0122. E-mail: mitch@engr.smu.edu.*
- *C. Traver is with the Electrical and Computer Engineering Department, Union College, Schenectady, NY 12308. E-mail: traverc@union.edu.*

Manuscript received 9 June 2003; revised 16 June 2004; accepted 24 Jan. 2005; published online 16 May 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0059-0603.

mechanisms external to a PL gate that can cause a failure due to timing. The coarse-grain approach used in this paper maps groups of gates in the clocked netlist to the combinational compute function of 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 between local interconnected blocks; there are no global mechanisms that can cause failure due to timing.

2.1 The Marked Graph Model

The clocked-to-PL transformation is based upon marked graph theory and definitions are required before discussing the transformation process. The following definitions are taken from [13] as they are concise and complete. Expanded definitions can be found in [12].

A Petri net (PN) is a triplet, PN = {T, P, F} where:

- T is a nonempty finite set of transitions,
- P is a nonempty finite set of places, and
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation between transitions and places.

A PN can be represented as a directed bipartite graph, where the arcs represent elements in a flow relation. A PN marking is a function m: $P \rightarrow \{0, 1, 2 \ldots\}$, where $m(p)$ is called the number of tokens in p under marking m. A transition $t \in T$ is *enabled* at a marking m if all of its predecessor places are marked. An enabled transition t may fire, producing a new marking m' with one less token in each predecessor place and one more in each successor place (denoted by $m[t > m'$).

A sequence of transitions and intermediate markings $m[t_1 > m_1[t_2 > \ldots m'$ is called a firing sequence from m. The set of markings m' reachable from marking m through a firing sequence is denoted by $m[>$. The set $m_0[>$ is called the reachability set of a marked PN with initial marking m_0, and a marking $m \in [m_0 >$ is called a reachable marking.

A PN marking is live if, for each $m' \in [m$ and for each transition t, there exists a marking $m'' \in [m' >$ that enables t. Similarly, a transition t is live if, for each $m' \in [m$, there exists a marking $m'' \in [m' >$ that enables t. A marked PN is live if its initial marking is live. A marked PN is *k-bounded* if there exists an integer k such that, for each place p for each reachable marking m, we have $m(p) \leq k$. A marked PN is *safe* if it is 1-bounded (this ends the definitions used from [13]).

A PN is a marked graph (MG) if every place has exactly one predecessor and one successor. A shorthand graphical notation is usually adopted for MGs in which arcs are shown as between transitions, with the intervening places understood. We use this shorthand notation for MGs in our figures unless explicitly noted otherwise. We also use the terms node and transition interchangeably. In an MG (G, m_0), a *directed circuit* C is a directed path that begins at a transition t and ends at the same transition t. The sum of the tokens in the set of places contained in C is the token count of C, designated by $m_0(C)$. Two important theorems [4] about the liveness and safety of MGs are:

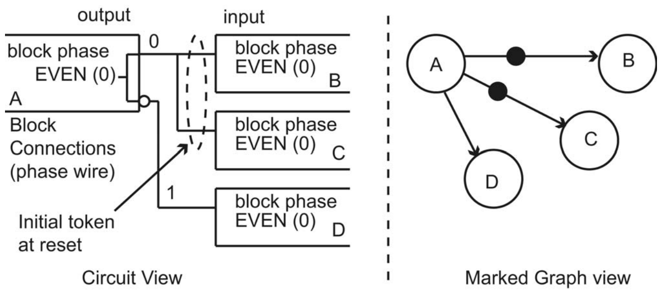


Fig. 1. Token abstraction.

Theorem 1. *A marked graph* (G, m_0) *is* ***live*** *if and only if, for all directed circuits C of G,* $m_0(C) > 0$, *i.e.,* m_0 *places at least one token on each directed circuit in G.*

Theorem 2. *A marked graph* (G, m_0) *is* ***safe*** *if and only if every arc of G belongs to some directed circuit C with* $m_0(C) \leq 1$. *As a corollary, a live marked graph is safe if and only if every arc belongs to a directed circuit C with* $m_0(C) = 1$.

If a PL netlist is not live, then signal transitions do not occur (tokens do not circulate) and, thus, there is no activity in the netlist. A PL netlist requires token circulation for computation, so a dead PL circuit is not useful. If a PL netlist is unsafe, then a PL block can fire and produce a second output value before a destination block has consumed the first output value, resulting in incorrect operation.

A PL coarse-grain netlist is composed of multiple blocks, where each block has a mixture of DFFs and combinational logic (a *barrier* block) or combinational logic only (a *through* block). Each block output is a data bundle consisting of multiple data wires and one phase wire (similar to a "request" wire in other asynchronous methodologies). A block and corresponding phase wire are represented as nodes and arcs in the MG model. The phase wire toggles between "0" and "1," designated as the EVEN phase and ODD phase. Each PL block contains an internal state element, called the block phase, which is either EVEN or ODD. If the phase of an input phase wire matches the block phase, then the arc that models the data bundle is said to contain a token. A PL block fires if all input arcs have tokens; firing toggles the internal block phase and toggles all output phases. Upon firing, all data wires in output bundles are updated with new values.

Fig. 1 shows this token abstraction for a block A with fanout to three destination blocks (B, C, D). The block phase provides the output phase and both true and complement versions are available. Upon reset, all blocks have the EVEN phase (an arbitrary choice, the only requirement is that all blocks are reset to the same phase). Initial token marking in the MG is accomplished by a wiring choice; using the uncomplemented phase output assigns an initial token to that signal. In Fig. 1, the arcs from block A to blocks B and C will have an initial token after reset. In the MG model, all fanout is represented by separate arcs as each fanout must be considered individually in terms of safety and liveness.

2.2 The Clocked-to-PL Transformation

The starting point for a coarse-grain clocked-to-PL netlist transformation is a hierarchical clocked netlist in which the

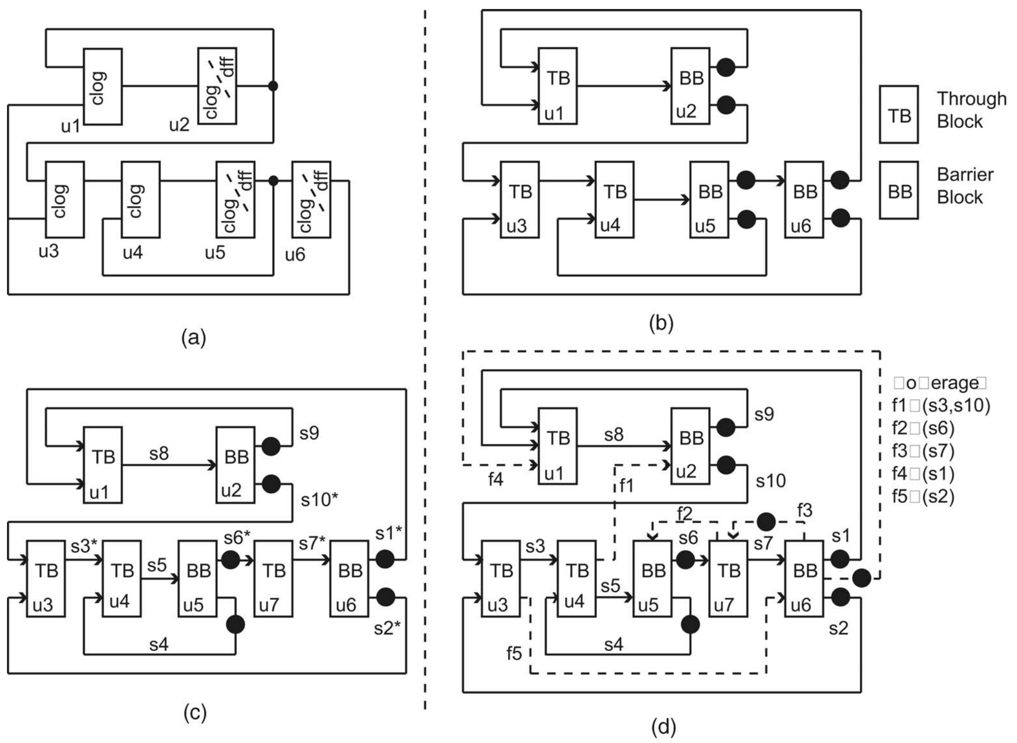


Fig. 2. Example translation. (a) Partitioned clocked netlist. (b) Initial Marked Graph (no splitter blocks). (c) After splitter block insertion, s*I** indicates an unsafe signal in initial marking. (d) After feedback insertion, graph is live and save.

components at the top level define the blocks that will have PL control logic placed around them. If a block contains D-Flip-Flops and combinational logic, then it is designated as a *barrier* block; if it contains only combinational logic, then it is called a *through* block. The terms *barrier* and *through* are used to distinguish these blocks for the purpose of initial token marking rules, which are specified later in this section. The translation procedure may need to insert additional phase wire signals, termed *feedback* signals, to make the resulting PL netlist live and safe (a more familiar term is *acknowledgment* signals, but we will use the terminology developed in [1]). In the MG equivalent, a feedback signal is the same as any other directed arc between nodes. However, in the PL netlist, a feedback signal does not have any data associated with it, so a feedback signal from a block is simply a phase wire without the corresponding data wires.

The translation algorithm that maps clocked netlists to PL netlists consists of the steps that are outlined below. These steps are illustrated in Fig. 2a, Fig. 2b, Fig. 2c, and Fig. 2d. See [1] for more details. These rules assume the PL netlist forms a closed system, i.e., that the global reset is the only external input signal. The method for addressing external I/O signals is discussed after the presentation of the translation rules.

1. To build the internal MG model, all mixed DFF/combinational logic blocks in the clocked netlist are mapped one-to-one to barrier blocks in the PL netlist. All combinational blocks are mapped one-to-one to through blocks in the PL netlist. All inputs to a block from a source block are collapsed to one arc in the MG model. The initial token marking rules assume that inputs from barrier blocks always have an initial token on them. Therefore, all barrier blocks have their uncomplemented phase outputs connected to fanout blocks to implement initial tokens at reset. The initial token marking rules require that any nonfeedback input signals from through blocks do not have an initial token, so these input signals are connected to the through block output whose phase is opposite the through block's internal phase. A global reset signal is used to reset all blocks to EVEN phase at startup. Fig. 2a and Fig. 2b illustrate this step.
2. Extra blocks, termed *splitter* blocks, are inserted to break any direct connection between barrier blocks. Splitter blocks are through blocks that implement buffer functions. Splitter blocks are required so that the initial token marking rules and feedback insertion rules can result in a live and safe MG. In Fig. 2c, through block $u7$ is a splitter block inserted between barrier blocks $u5$ and $u6$.
3. The network is traversed and any signals that are part of a directed circuit with token count = 1 are marked as *safe* signals. At this point, only signals from barrier blocks are marked with initial tokens, so a signal is safe iff it is in a directed circuit that begins at a barrier block and terminates at the same barrier block. It is important to understand that each fanout from an output counts as a separate signal for safety checking. If all signals are safe, then the transformation process is finished and the PL netlist is live and safe.

Extra arcs called *feedbacks* are now added to the MG model to make the remaining signals safe. A feedback

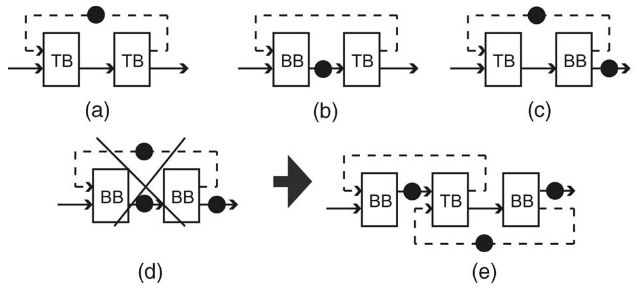


Fig. 3. Feedback insertion rules. (a) TB to TB feedback. (b) TB to BB feedback. (c) BB to TB feedback. (d) BB to BB feedback. (e) Splitter block inserted between barrier blocks.

signal is added from the output of a source block to the input of a destination block to form a directed circuit with token count = 1. Any initially unsafe signals contained in this directed circuit are now safe. Any signals made safe by the addition of the feedback are said to be covered by that feedback. The directed circuit cannot include a barrier block unless the barrier block is either the source or destination of the feedback. Fig. 3 summarizes the rules for feedback insertion and initial token marking for directed circuits formed by feedbacks. A feedback originating from a through block and terminating on a through block has an initial token (this marking supplies the token in the directed circuit since the nonfeedback output of a through block does not have an initial token, see Fig. 3a).

A feedback originating from a through block and terminating on a barrier block does not have a token (the initial token on the directed circuit is provided by the barrier block, see Fig. 3b). Any feedbacks originating from barrier blocks have initial tokens (i.e., all signals originating from barrier blocks have initial tokens, see Fig. 3c). A feedback cannot both originate from a barrier block and also terminate on a barrier block (the directed circuit formed would be unsafe as it would have an initial token count > 1, see Fig. 3d). Fig. 3e shows how this problem is solved by splitter block insertion to break barrier-to-barrier block paths.

There are usually multiple ways in which feedback can be added to cover unsafe signals [1]. A feedback's destination can be multiple block levels back, tracing from the source, which then covers multiple signals (i.e., feedback f1 in Fig. 2). However, this can create a long loop delay from feedback destination block firing to feedback source firing, adversely affecting system performance. In the coarse grain methodology, feedback destinations are limited to a source block's immediate predecessor. This maximizes the number of required feedbacks, but prevents feedbacks from limiting system performance. Optimal insertion of feedback signals in a coarse-grain netlist is an area of future study.

In [1], the marked graph equivalent as produced by the clocked-to-PL translation algorithm has been proven to result in marked graphs that are live and safe and whose firing sequence displays the same cyclic, synchronous, deterministic behavior as the clocked netlist. The safety and liveness of external inputs and outputs are handled at the VHDL testbench level during simulation; the PL netlist provides a feedback output to the testbench for each input and accepts a feedback input from the testbench for each output. The same token marking rules are applied to these feedbacks as are applied in the clocked-to-PL transformation process.

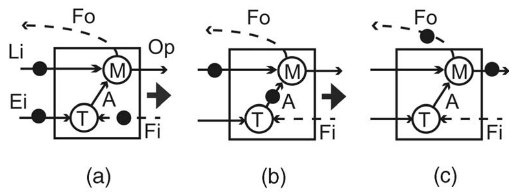


Fig. 4. Normal fire case for EEnode. (a) Token arrival. (b) T fires. (c) M fires.

2.3 Early Evaluation

Early evaluation [7] allows a block to fire upon arrival of only a subset of inputs. This increases the amount of parallel activity in the graph, which increases performance. Inputs to an early evaluation block are separated into early arriving inputs (Ei) and late arriving inputs (Li). A logic function, called the *trigger* function, which is based on the data bundles of the early inputs, determines if the block fires after all early inputs have arrived. An early evaluation block has separate signals for output phase (Op) and feedback output (Fo). In an early evaluation block, the feedback output is not updated until all inputs have arrived. In a nonearly evaluation block, the output phase also serves as the feedback output.

Fig. 4 shows a two node marked graph model that is used by the mapping algorithm to represent an early evaluation block (this two node marked graph is henceforth collectively referred to as an EEnode). All late inputs terminate on the M node and all early inputs on the T node. Feedback output originates from the M node; feedback input terminates on the T node. We view an early evaluation block as dynamically switching between two configurations: normal fire and early fire.

A *normal fire* occurs when the trigger function evaluates to false; the output phase is updated after all inputs have arrived and is viewed as originating from the M node, as shown in Fig. 4. An *early fire* occurs when the trigger function evaluates to true after all early inputs have arrived; the output phase is updated and is viewed as originating from the T node, as shown in Fig. 5. In the early fire case, the M-node fires after the T-node fires and after all late inputs have arrived. The firing of the M-node updates the feedback output.

Fig. 6 shows an example of a marked graph model for a simple PL netlist that includes an EEnode (Gb). A key question for PL netlists with EEnodes is how to maintain liveness and safety. In Fig. 6a, the graph is live and safe if the

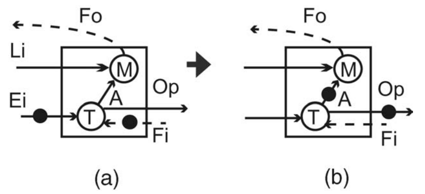
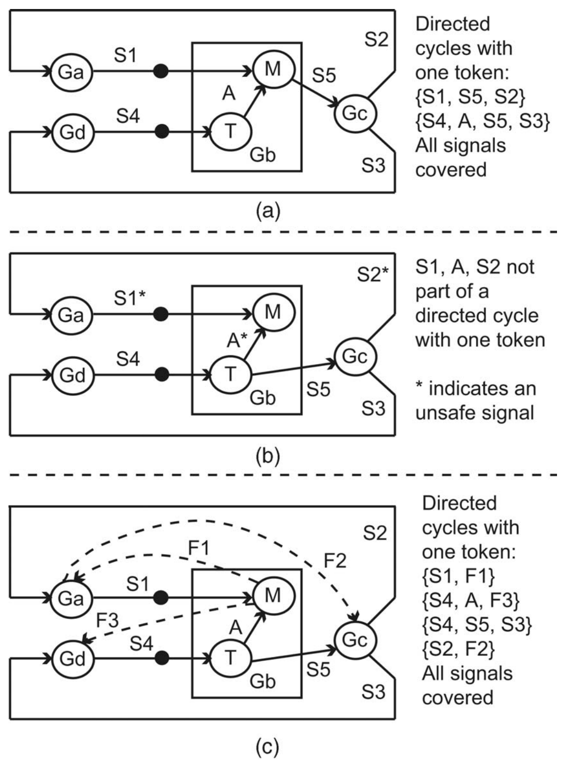


Fig. 5. Early fire case for EEnode. (a) Token arrival. (b) T fires.



Directed cycles with one token: {S1, S5, S2} {S4, A, S5, S3} All signals covered
Ga
Gd
M
T
Gb
Gc
A
S1
S2
S3
S4
S5
(a)
S1, A, S2 not part of a directed cycle with one token
* indicates an unsafe signal
S1*
S2*
A*
(b)
Directed cycles with one token: {S1, F1} {S4, A, F3} {S4, S5, S3} {S2, F2} All signals covered
F1
F2
F3
(c)

Fig. 6. Safety in PL netlist with EEnodes. (a) Normal fire case, all signals safe. (b) Early fire case, signals S2, A, S2, unsafe. (c) Feedback added, all signals safe in early fire case.

normal fire configuration of the EEnode is used. However, using the early fire configuration of Gb in Fig. 6b makes the graph unsafe as signals S1, A, and S2 are not part of a directed circuit C with token count 1 (m(C) = 1).

Fig. 6c adds feedback signals F1, F2, and F3 to make the graph live and safe. Three rules (presented as lemmas) for feedback insertion in the presence of EEnodes are evident from Fig. 6.

Lemma 1. *All late arriving signals into an EEnode must be part of a directed circuit that includes the feedback output from the EEnode as the feedback output is the only signal originating from the M-node in the early fire configuration.*

Lemma 2. *At least one early input must be in a directed circuit that includes the feedback output from the EEnode as this is the only way to include the internal signal A of the EEnode in a directed circuit in the early fire configuration.*

Lemma 3. *The output signal of an EEnode must be in a directed circuit that either includes an early input or a feedback input that terminates on the T node. This ensures that the output signal will be in directed circuit in either the normal or early fire configurations.*

It is clear then that an EEnode represents a form of choice and a marked graph is defined as being choice-free. Different formal methods for representing choice are Free-Choice Petri nets, Change Diagrams, and Causal Logic nets [13]. However, using one of these representations means that the simple properties of liveness and safety of marked graphs are lost. In the remainder of this section, we define a reconfigurable marked graph model and prove that any marked graph containing EEnodes that has a live and safe initial marking created using Lemmas 1-3 will be safe and live for any early or normal firings of EEnodes.

Let a marked graph consisting of a directed graph G and marking m_i be designated by (G, m_i). The firing of a non-EEnode simply changes the marking and the graph transitions from (G, m_i) to some (G, m_k), where m_k is the new marking of the marked graph. It is well-known that, if a marked graph G with initial marking m_0 is live and safe, then any marked graph (G, m_i) reachable by a set of node firings from (G, m_0) is also live and safe.

However, the firing of an EEnode can change the graph, as the nonfeedback output arcs of an EEnode can change their origination points from the M node or the T node. In order to keep the marked graph model in PL netlists with EEnodes, we view a PL netlist as transitioning from a marked graph (G, m_i) to a new marked graph (G', m_j) any time an EEnode changes configuration (from normal to early configuration or vice versa). A configuration change occurs when the block fires and if the type of firing (early or normal) is different from the previous firing. An *early-to-normal* configuration change means the current fire is a normal fire and the previous fire was an early fire. A *normal-to-early* configuration change means the current fire is an early fire and the previous fire was a normal fire. Our approach for making a PL netlist with EEnodes live and safe is to make the marked graph equivalent (G, m_0) live and safe by adding appropriate feedback signals and an initial marking, where each EEnode in (G, m_0) is represented by its *early fire* configuration. We then claim that any combination of firings of non-EEnodes or EEnodes results in a live and safe marked graph (G', m_i). We prove this through two theorems.

Theorem 3. *From any marking m_i reachable from (G, m_0) by non-EEnode firings or EEnode early firings, allow a single EEnode u_i to perform an early-to-normal configuration change. The resulting graph (G', m_i') is live and safe.*

Proof. The only directed circuits C with $m(\mathrm{C}) = 1$ affected by the early-to-normal configuration change of u_i are those containing an output arc *Op* of EEnode u_i as the predecessor node to *Op* is now the M node instead of the T node. All of these directed circuits now contain internal arc A as a result of the configuration change. The configuration change resulted from the T node firing, which places a token on arc A, so all of these circuits are live, $m(\mathrm{C}) > 0$. When the T node fired, the only arcs in these directed circuits that could have contained a token are the arcs incident upon the T node. The firing of the T node consumed these tokens, so the token count of these directed circuits remains unchanged, at m(C) = 1.□

Theorem 3 can be trivially extended to cover any number of nodes $\mathrm{U}k$ performing early-to-normal configuration changes as each directed cycle with $m(\mathrm{C}) = 1$ can only have one node ready to fire, so the cycles with $m(\mathrm{C}) = 1$ affected by an early-to-normal configuration change are independent of any other nodes that undergo the early-to-normal configuration change.

Thus, any graph (G', m_i') reachable from (G, m_0) by non-EEnode firings, EEnode early firings, or EEnode early-to-normal configuration changes is live and safe. The next theorem covers normal-to-early configuration changes.

Theorem 4. *From any marking m_k' reachable from graph (G', m_i') by non-EEnode firings, EEnode early firings or EEnode early-to-normal configuration changes allow a single EEnode u_i to perform a normal-to-early configuration change. The resulting graph (G'', m_k'') is live and safe.*

Proof. The only directed circuits C with $m(\mathrm{C}) = 1$ affected by the normal-to-early configuration change are the ones containing an output arc Op of EEnode u_i as the predecessor node to Op is now the T node instead of the M node. All these directed circuits now no longer contain arc A as a result of the configuration change, but they do still contain the output arc Op. The firing of the T node that caused the normal-to-early configuration change places a token on Op, so these directed circuits are live, $m(\mathrm{C}) > 0$. As these directed circuits have $m(\mathrm{C}) = 1$ at the time of T node firing, the only arcs in these directed circuits that could have contained a token are the arcs incident upon the T node. The firing of the T-node consumed these tokens, so the token count of these directed circuits remain unchanged, at $m(\mathrm{C}) = 1$. In the original graph (G, m_0), the A arc had to be covered by a directed circuit with $m(\mathrm{C}) = 1$ that included a feedback output of the EEnode and configuration changes of the EEnode does not affect this cycle. □

Theorem 4 can be trivially extended to cover any number of nodes $\mathrm{U}k$ performing normal-to-early configuration changes by the same reasoning used to extend Theorem 3. Theorems 3 and 4 taken together means that, given a starting marked graph (G, m_0) containing EEnodes and non-EEnodes that is live and safe using the EEnode early fire configuration, then any marked graph (G', m_i') reachable by non-EEnode firings, EEnode early firings, EEnode late firings, EEnode early-to-late or late-to-early configuration changes is also live and safe.

3 Related Work

The "de-sync" self-timed design style [9] is the most similar of recent published methods to our coarse-grain design style in that it uses a coarse-grained micropipeline with bundled-data signaling, a marked graph model, and a commercial standard cell library. It also begins with a clocked netlist and replaces the global clock network. The micropipelined de-sync implementation uses four-phase control [8] built from standard cells. For the DLX processor implementation (without forwarding) described in [9], the asynchronous design has equivalent area, performance, and power consumption when compared to the clocked design. The main difference between our approach and that in [10] is that the de-sync approach has not been shown to support the concept of early evaluation, which provides the potential for increased performance. Another difference is that the de-sync approach splits all DFFs in the original netlist into master/slave components with separate control wrappers for each. This can potentially result in an overhead-free (performance) asynchronous implementation of the clocked system if the master/slave delay is the same as the original DFF delay + setup time. We initially tried this approach in [2], but abandoned it in the standard cell designs as we found that using available latches within the libraries resulted in a higher performance penalty than just using a single DFF and satisfying the setup time penalty. The choice of a master/slave latch approach versus a DFF approach is highly dependent on available latch and DFF designs in a given commercial standard cell library.

Other self-timed CPUs that have been designed in the past include the MIPs integer subset [14], the ARM processor [15], and the 8051 [16]. The distinguishing features of our design are the automated mapping from the clocked netlist to a self-timed netlist and the use of early evaluation. Previous self-timed CPUs, such as the Amulet3, have used bypass paths to speed execution. The Amulet3 execution unit had an iterative multiplier and barrel shifter in series with the ALU; these two components were bypassed when instructions did not require them. Bypass operations are essentially a degenerative case of early evaluation in which all phase inputs are part of the trigger phase and the early evaluation function has a smaller delay than the normal compute function. The bypass operation is used when all signals arrive at the same time, but different delays are desired depending upon the block operation for that particular compute cycle (i.e., within an ALU, shift versus addition). As such, the early evaluation control wrappers discussed in Section 4.3 support bypass operations and our design makes use of bypass in much the same way as the Amulet3. However, the CPU design presented in Section 5 also uses the general case of early evaluation in which blocks fire upon arrival of a subset of inputs.

4 The Coarse-Grain PL Methodology

The tool flow and circuit implementations that transform a clocked circuit into a PL implementation are described below.

4.1 Tool Flow

Fig. 7 illustrates the PL coarse-grain methodology flow. Synopsys is used for RTL synthesis and static timing. The custom tools in the flow are:

pl_partitioner: The principle function of this tool is to insert slack matching buffers into the netlist as specified by an external configuration file. In the clocked system, a slack-matching buffer is simply a combinational buffer on all signals between two blocks. In the PL implementation, a PL control wrapper is placed around this block and, thus, it functions as an intermediate storage location for data tokens. It was shown in [14] that slack-matching buffers added to asynchronous pipelines can improve performance in some cases; one case where slack-matching can be beneficial is if a block output feeds several other blocks that complete at different times. Having this tool read slack buffer locations from an external file frees the user from having to "pollute" their top-level RTL with slack buffers. This tool is implemented in C and has been tested under Linux Redhat 7.3 and Sun Solaris.

pl_mapper: This tool automatically generates the PL control wrappers and the control signal network based upon the

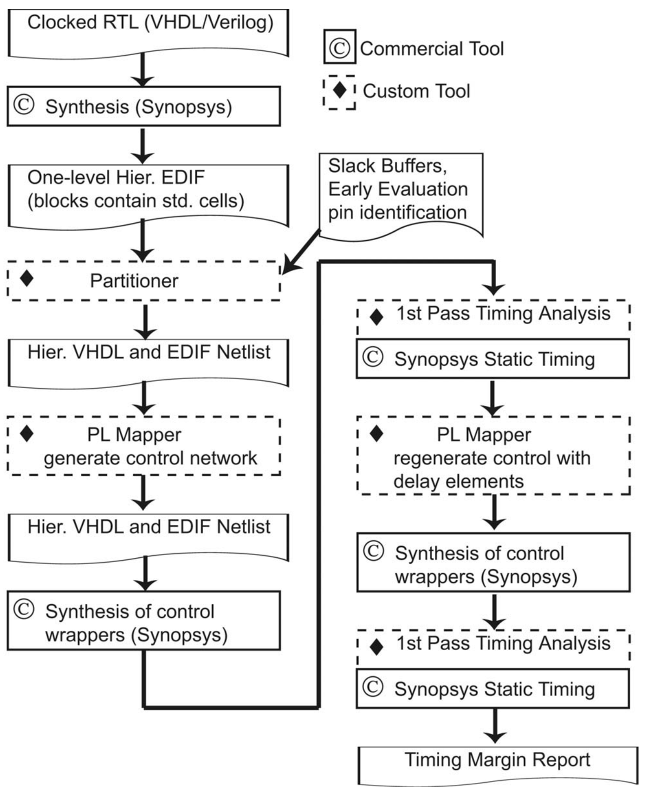

Clocked RTL (VHDL/Verilog)
© Commercial Tool
◆ Custom Tool
© Synthesis (Synopsys)
One-level Hier. EDIF (blocks contain std. cells)
Slack Buffers, Early Evaluation pin identification
◆ Partitioner
Hier. VHDL and EDIF Netlist
◆ PL Mapper generate control network
Hier. VHDL and EDIF Netlist
© Synthesis of control wrappers (Synopsys)
◆ 1st Pass Timing Analysis
© Synopsys Static Timing
◆ PL Mapper regenerate control with delay elements
© Synthesis of control wrappers (Synopsys)
◆ 1st Pass Timing Analysis
© Synopsys Static Timing
Timing Margin Report

Fig. 7. PL coarse-grain methodology flow.

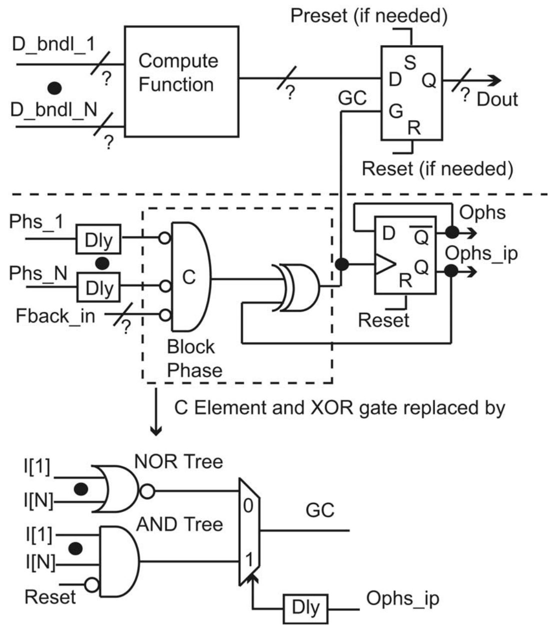


Fig. 8. PL control wrapper (no early evaluation).

clocked netlist. This is the heart of the PL methodology and this tool shares common code with the fine-grain methodology. This tool is implemented in C and has been tested under Linux Redhat 7.3 and Sun Solaris.

A perl script called *plcg_timing.pl* is used to perform the timing analysis for each block using Synopsys static timing. The overall flow is controlled by a top-level perl script called *plcg_make.pl*.

4.2 Designer Responsibilities

Aside from providing clocked RTL, designer responsibilities for producing a working PL system are:

1. The top-level RTL must contain the components that define the blocks to be encapsulated by PL wrappers. Currently, there is no automated partitioning capability in the PL coarse-grain methodology.
2. Early evaluation opportunities must be identified by the designer. The designer must identify a single output port from the block that is a "1" when the block early evaluates. RTL code must be added to the block to support this function (in many cases, the logic function is already available). More details on early evaluation are provided in [2].
3. The designer must identify locations of slack matching buffers via a configuration file provided to the *pl_partition tool*.

4.3 PL Control Wrappers

Two standardized control wrappers are used for PL blocks —one that supports early evaluation and one that does not. Fig. 8 shows the PL control wrapper used for nonearly evaluation blocks [17] and its interface to the datapath block. This is a variation on the traditional control used for micropipelines [10]. Each delay block for a phase input is chosen to match the datapath delay of the corresponding data bundle through the compute function. The delay block is a chain of delay cells where the delay cell was provided as a standard cell within both libraries. Delay blocks are generated automatically as part of the tool flow, with first-pass static timing used to calculate the required delay block values and second-pass static timing used to verify that delay-matched paths met the user-specified minimum timing margin. Initially, a Muller C-element and XOR gate was used to produce the gating signal for the output latches. The C-element is mapped to a standard cell implementation, as described in [6]; no monolithic C-element is available in either library. The C-element output could be used to provide the phase output directly, but the DFF is used as it provides clear starting and stopping points for static timing analysis and makes control/datapath delay matching easier. This C-element and XOR gate combination was found to be slow (especially the XOR gate) and was replaced in the final design by the logic in the lower half of the figure for through blocks that did not perform time-borrowing with a successor block (time borrowing usage is discussed in Section 5.1). Fast control is helpful when the compute function delay for a particular phase input is lower than the control path delay, which occurred in some instances. In barrier blocks, the latches shown in the datapath block section are replaced by the DFFs in the original clocked netlist. If a barrier block has outputs that loop back to the compute function (which is usually the case), then that barrier block has a feedback to itself with a delay equal to the longest delay of the loopback path through the compute function.

Fig. 9 shows the PL control wrapper with early evaluation capability [18]. A normal fire occurs when EE_q = ''0'' after all inputs have arrived; both output phase and feedback phase signals are updated. An early fire occurs when EE_q = ''1'' and all inputs have arrived to the

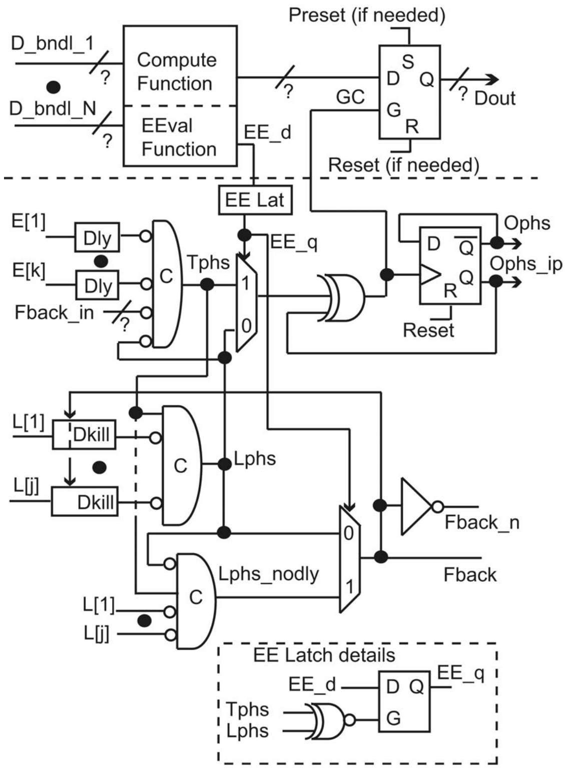


Fig. 9. PL control wrapper with early evaluation.

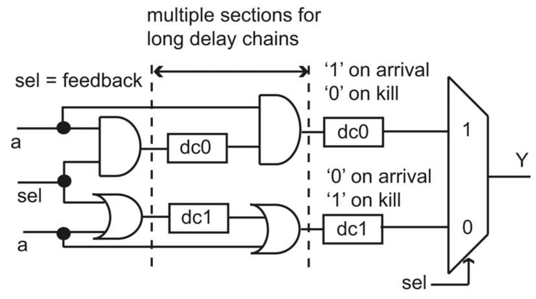


Fig. 10. Delay kill block internals.

Tphs phase C-element; this updates the output phase, but not the feedback phase.

After an early fire, the feedback output phase is updated once all of the late inputs (L[1]-L[j]) arrive. The *Lphs_nodly* C-element provides this value; note that this C-element has no delay blocks as these delays do not have to be satisfied once the output has been updated. This provides a fast path for feedback once all inputs have arrived. However, new L[1]-L[j] inputs could arrive while old inputs are still traversing delay blocks, causing input hazards to the *Lphs* C-element.

To avoid this problem, the *Dkill* block uses two delay chains internally, as shown in Fig. 10. The toggling of the *sel* (feedback) signal routes the a input between the two delay blocks so that one delay block is "recovering," while the other delay block is "active." Normal operation is either $\mathrm{a}+ \leftarrow \mathrm{N1} + (\mathrm{sel} = 1)$ or $\mathrm{a}- \leftarrow \mathrm{N0} - (\mathrm{sel} = 0)$, where the full delay chain penalty is used. An early fire can cause *sel* to change while the a transition is still within *dly1* or *dly0*. A change in *sel* chooses the opposite delay path, whose value is the normal arrival value for the previous delay path.

Providing early feedback before the late C-element fires means that the *Lphs* C-element has to be an input to the *Tphs* and *Lphs_nodly* C-elements to prevent firing of these C-elements before the late C-element has caught up to these states. The *EEsel* latching logic holds the EEsel signal stable as long as the *Lphs* C-element and *Tphs* C-element states are not equal to each other.

5 Design Examples

Mapping and simulation results for a MIPs-subset CPU and a double-precision floating-point clipping operator are presented here. Simulation results are from a gate-level Verilog simulation using prelayout back-annotated SDF generated by Synopsys.

5.1 A MIPs-Subset CPU

The PL coarse-grain methodology was first discussed in [2] and applied to the same MIPs-compatible CPU. In that design, the logic was synthesized to four-input lookup tables (LUT4s), the control wrappers used behavioral models, the register file used a behavioral model, timings did not account for output loading, and no timing margins were assumed for either the PL or clocked netlists. In this CPU implementation, all logic has been mapped to a standard cell library targeted for a 0.13μ technology. The register file is implemented using a two-port register file generator for the same technology.

Fig. 11 shows the CPU architecture. Behavioral models were used for instruction and data memories. Labeled through blocks contain only combinational logic. The unlabeled through blocks are buffers added automatically by the mapping tool to break connections between barrier blocks. A Verilog model with full timing as produced by the 0.13μ two-port register file generator was used for the *rfcore* block. A custom PL wrapper was designed for the *rdport* and *writeport* blocks to interface to the *rfcore*, space constraints prevent its inclusion in the figures. The connections in Fig. 11 indicate the phase signal connections between blocks; feedback connections are not shown as most blocks provide feedback to their immediate predecessor. However, feedback is not required in all cases. For example, the *incpc* block does not provide feedback to the *PC* block as these phase signals are part of a naturally occurring loop that contains one barrier block. Also, fanout from one block to multiple blocks uses the same phase/data bundle even though separate connections are show in the figure. The *idpipe* block provides the same phase/data bundle to the *add/shift/log* blocks; this phase/data bundle is not duplicated. However, separate feedbacks are required for these three fanouts.

Early evaluation is used as follows:

- The PC block fires early if the branch PC computation (*bpc* block) is not required.
- The *idpipe* block (decode stage) fires early if the new operands for the execute stage do not require either a data memory value or a forwarded value from the ALU output (exep2 block).
- The *add* block fires early if the operation is not an add or subtract. The *shift* block fires early if the

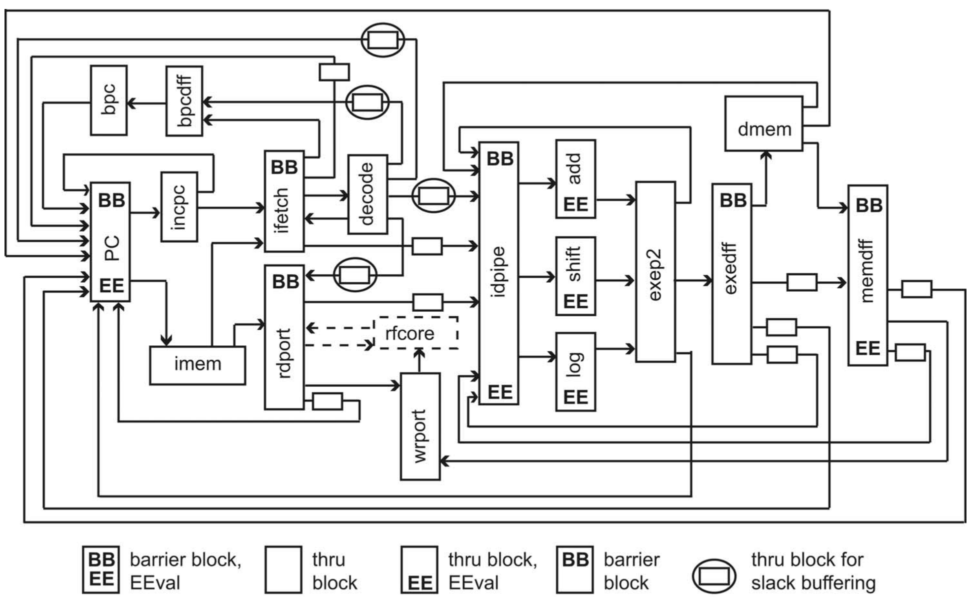


Fig. 11. CPU architecture diagram.

operation is not a shift operation. The *add* and *shift* blocks only have one input, so this use of early evaluation is simply a bypass operation.

- The *memdff* block fires early if a data memory operation is not required.

Slack matching buffers were used on the output of the decode block as it fans out to multiple destinations which finish at different times. These buffers further desynchronize the firing of the blocks, improving throughput.

Because of the use of latches in through blocks, *time borrowing* can be used between two through blocks or between a through block and a barrier block if the data delay in one block is less than the control path delay. Time borrowing was used across block pairs *add/exep2, shift/exep2, log/exep2, incpc/pc, decode/ifetch*, and in the through block between the *rdport* and *idpipe* blocks. Time borrowing is performed automatically by the mapper between a source block and a destination block if the destination block is not an early evaluation block, has only one control input, and if the data delay is less than the control delay in the destination block. More aggressive time borrowing is possible, but is not implemented in the current methodology.

5.2 CPU Performance Results and Area Values

Table 1 gives the performance results of the PL CPU compared to the clocked CPU for five benchmark programs. All benchmarks were written in C and compiled using *gcc*. A number > 1.0 for the PL/CLK ratio means that the PL design had slowdown compared to the clocked design; all benchmark programs had slowdown for the PL versus the clocked design. The simulations used a gate-level Verilog netlist with Synopsys-generated prelayout timing delays. Synopsys static timing reported the register-to-register critical path of the clocked design as 2,170 ps; to this value was added a 3 percent clock skew budget of 65 ps for a total clock period of 2,235 ps. For the PL netlist, a target timing margin of 20 percent was specified for delay chain generation, with a 10 percent minimum timing margin allowed for failure flagging for margin checking in the second timing pass. The minimum margin reported for any path after second pass timing was 12 percent. Suggested timing margins [19] for delay matching in micropipelines range from 10 percent for regular/tiled layout blocks to 30 percent for synthesized standard cell blocks. However, delay path matching in micropipelines is equivalent to gated clock/datapath delay matching in high performance microprocessors [20], [21]. These designs regularly use margins of less than 10 percent of the clock period. The target margin selected for a commercial design will depend heavily on the quality of timing analysis tools available, the skill of the engineering team, and the amount of time available for fine-tuning.

The nonreordered code used the assembly code unchanged as produced by the *gcc* compiler. The reordered code used manual reordering of code sequences in critical loops, increasing early evaluation opportunities as reported in [2]. An example code reordering is shown in Fig. 12, where both sequences give equivalent results, but the *bne* instruction in Sequence B does not require operand forwarding from the *exep2* block, resulting in faster execution.

TABLE 1
Benchmark Performance Results

Benchmark	PL/CLK (noEE)	PL/CLK (EE,non-reordered)	PL/CLK (EE, reordered)
fibonnaci	1.29	1.12	1.12
bubblesort	1.29	1.16	1.15
crc	1.29	1.12	1.07
sieve	1.29	1.18	1.15
matrix transpose	1.29	1.19	1.16
average	1.29	1.15	1.13

```
addi  r4,r4,1        Sequence A
slti  r2,r2,8
bne   r2, r0, L10
---------------------
slti  r2,r2,8        Sequence B
addi  r4,r4,1
bne   r2, r0, L10
```

Fig. 12. Example code reordering.

The fastest instructions were logical operations that did not require operand forwarding from the *exep2* block. The CRC reordered benchmark has the fastest execution because it contains the largest number of logical operations and fewest operand-forwarding requirements.

In [2], an average speedup of 41 percent for PL versus Clocked was obtained for the reordered benchmarks using the generic LUT4 technology. Reasons for the slowdown in the standard cell MIPs are:

- No timing margin was used for the results presented in [2].
- In [2], the simulation did not take into account the effect of loading on nets; all cell delays were fixed, regardless of output loading.
- Output latch delay and control path delay were significantly underestimated for the behavioral models used for the wrapper logic in [2].

The results presented in [2] can be viewed as an estimation of the maximal speedup that could be obtained if extremely low latency latches/DFFs and highly optimized control was used in the design in the wrapper logic.

The performance numbers are disappointing in that the PL CPU has slowdown compared to the clocked design. However, the slowdown is not prohibitive from considering this approach as a viable alternative to a global clock distribution network.

TABLE 2
Area Results

Design	Cell Count	Cell area	% area increase
Clocked	9183	169854	
PL	14983	249570	47%

Table 2 shows that the PL design has an increase of 47 percent in active cell area as reported by Synopsys. The clocked design area figure does not include any area required by the global clock network, and neither figure includes interconnect area.

5.3 Double-Precision Floating-Point Clip

A double precision floating-point clip operation defined as shown was implemented in several ways and mapped to both 0.18 and 0.13 standard cell libraries.

```
if ( A < low_bound) then
     Y := low_bound
elsif (A < high_bound) then
     Y := A;
else Y := high_bound;
```

Fig. 13 shows the design variations, which were chosen to experiment with partitioning, time borrowing, and early evaluation choices.

a. Multicycle FSM, implemented as one barrier block. The FSM had four states: two states for loading values for high and low bounds and two for computing the clip value. The result was available in two or three clocks, depending on if it was out of bounds or not.

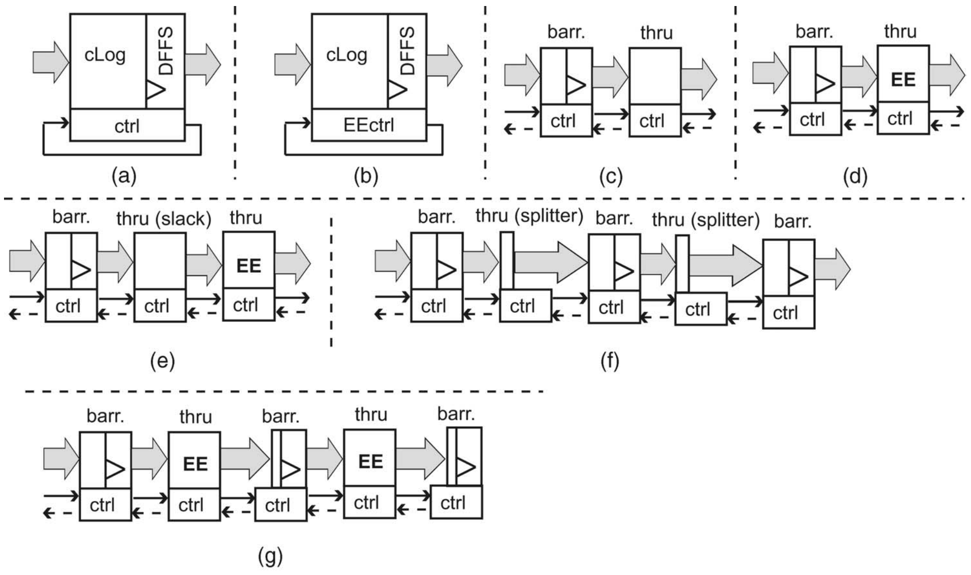


Fig. 13. Floating-point clip variations. (a) One block. (b) One block with EE. (c) Two blocks. (d) Two blocks with EE. (e) Three blocks, barrier, slack, EE. (f) Three-stage pipeline, three blocks in original netlist, splitter blocks inserted automatically between barrier blocks. (g), (h) Three-stage pipeline, five blocks in original netlist (three barrier, two thru blocks), second and third barrier blocks have thin logic, so time borrowing is done between thru and barrier blocks.

TABLE 3
Mapping and Simulation Results for Clip Variations

0.13μ		cells	area	PL/clk (area)	clk cyc (ps)	finish (ns)	PL/Clk (time)	Min. Margin	Borrow Margin
a. Mcycle	clk	2332	38075		2359	6725			
(1 blk)	PL	2456	39912	1.05	n/a	8322	1.24	11.4%	n/a
b. Mcycle	clk	2441	39254		2225	6343			
(1blk, EE)	PL	2564	42621	1.09	n/a	6540	1.03	13.3%	n/a
c. Mcycle	clk	2266	36446		1483	4230			
(2 blk)	PL	2678	42306	1.16	n/a	5151	1.22	14.4%	n/a
d. Mcycle	clk	2266	36446		1483	4230			
2 blk + EE)	PL	2753	42801	1.17	n/a	5031	1.19	17.4%	n/a
e. Mcycle	clk	2266	36446		1483	4230			
(2 blk + EE, buff)	PL	3613	55568	1.52	n/a	4865	1.15	17.4%	n/a
f. 3-stg pipe	clk	2822	50152		1411	1421			
(3 blks+2 splitter)	PL	4396	72878	1.45	n/a	2219	1.56	13.2%	n/a
g. 3-stg pipe	clk	2902	50102		1411	1422			
(5 blks, time brw)	PL	3898	62518	1.25	n/a	2038	1.43	16.1%	10.9%
h. 3-stg pipe	clk	2902	50102		1411	1424			
(5 blks, EE, time brw)	PL	4020	63190	1.26	n/a	1884	1.32	16.1%	15.5%
0.18μ									
a. Mcycle	clk	1920	63200		4542	12954			
(1 blk)	PL	2031	73441	1.16	n/a	15922	1.23	19.7%	n/a
b. Mcycle	clk	1872	62683		3801	10838			
(1blk, EE)	PL	2016	74229	1.18	n/a	9114	0.84	16.4%	n/a
c. Mcycle	clk	2091	65025		4274	12198			
(2 blk)	PL	2414	84162	1.29	n/a	15424	1.26	22.3%	n/a
d. Mcycle	clk	2091	65025		4274	12198			
(2 blk + EE)	PL	2493	89276	1.37	n/a	10375	0.85	18.3%	n/a
e. Mcycle	clk	2091	65025		4274	12198			
(2 blk + EE, buff)	PL	3149	113890	1.75	n/a	9914	0.81	18.3%	n/a
f. 3-stg pipe	clk	2295	82481		4110	4143			
5 blks)	PL	3488	135113	1.64	n/a	5810	1.4	16.9%	n/a
g. 3-stg pipe	clk	2548	84862		4110	4145			
(5 blks, time brw)	PL	3405	126904	1.50	n/a	5730	1.38	29.3%	20.2%
h. 3-stg pipe	clk	2548	84862		4110	4149			
(5 blks, EE, time brw)	PL	3535	135364	1.60	n/a	3434	0.83	20.4%	15.3%

b. Same as a, except early evaluation was used based on the sign bits of the high/low bound values and the input value.
c. Same as a, except logic was split into a barrier block and a through block.
d. Same as c, except EE was used in the same manner as 2.
e. Same as d, except a slack buffer was added.
f. Three stage pipeline, with a new input value every clock. The first stage had a simple two-state FSM that input the low/high bound values and accepted a new data input value every clock during computation. The second stage did low bound comparison, the third stage high bound comparison. The top-level netlist contained three mixed DFF/combinational logic blocks, which were mapped to three barrier blocks, with splitter blocks inserted automatically by mapping tool to break barrier-to-barrier gate paths.
g. Same as f, but the top-level netlist was partitioned into five blocks, with the stage 2 and stage 3 blocks of 6 split into combinational and sequential components. This resulted in no barrier-to-barrier gate paths, so splitter blocks were not inserted. Also, the mapper took advantage of available time borrowing between the through and barrier-blocks when creating delay chains for the through blocks.
h. Same as g, but early evaluation used for low/high bound comparisons based on both sign bits and exponent fields. The early evaluation computation conditions were done in the first stage and then passed to the successive stages.

Table 3 shows the mapping and simulation results for the 0.13μ and the 0.18μ libraries. The designs were tested with 1,000 vectors of randomly generated numbers between the values of +/-15.0, with low and high bounds of +/-5.0. Approximately 2/3 of the input numbers were clipped. The "Min Margin" column is the minimum time margin for all blocks. The "Borrow Margin" column is the minimum reported margin across any two blocks used time borrowing.

The most interesting aspect of the numbers in Table 3 is the disparity in the PL/CLK area/speed comparisons between the 0.13μ and 0.18μ libraries. For the 0.18μ library, the EE versions of the PL designs achieved speedup; for the 0.13μ library, no speedup was obtained. The reason for this disparity is that the 0.13μ designs had much shorter combinational paths in terms of total gates as a result of more efficient technology gate mapping. The longer combinational paths in the 0.18μ designs reduced the effective DFF and latch delay overhead in those netlists. This reinforces the point that low latency DFFs and latches in micropipeline designs can significantly increase performance. Neither library had particularly fast DFFs; in both libraries, the ratio of DFF clock-to-Q delay to a typical gate delay was a 4X to 5X factor.

The effect of partitioning at the top level is seen in the performance difference between designs f and g for the 0.13μ library. In design f, the three blocks at the top level were all barrier blocks, which forced the insertion of splitter blocks by the mapping tool to break barrier-to-barrier block paths. In design g, the top level included combinational blocks between stages. This removed the need for splitter block insertion and also allowed the mapper to take advantage of time borrowing between the through and barrier blocks during delay block creation.

6 Conclusions

This paper describes the first coarse-grain PL designs mapped to commercial standard cell libraries. The examples show that this approach can produce self-timed designs that are comparable with clocked designs in performance. The benefits of early evaluation were clearly shown in both the CPU and floating clip operation examples.

Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) under Grant CCR-0098272.

References

[1] D.H. Linder and J.C. Harden, "Phased Logic: Supporting the Synchronous Design Paradigm with Delay-insensitive Circuitry," *IEEE Trans. Computers,* vol. 45, no. 9, pp. 1031-1044, Sept. 1996.

[2] R.B. Reese, M.A. Thornton, and C. Traver, "A Coarse-Grained Phased Logic CPU," *Proc. Ninth Int'l Symp. Advanced Research in Asynchronous Circuits and Systems (ASYNC 2003),* pp. 2-13, May 2003.

[3] A. Wallander, "A VHDL Implementation of a MIPS," project report, Dept. of Computer Science and Electrical Eng., Luleå Univ. of Technology, http://www.ludd.luth.se/~walle/projects/myrisc, 2000.

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

[5] R.B. Reese, M.A. Thornton, and C. Traver, "A Fine-Grain Phased Logic CPU," *Proc. IEEE CS Ann. Symp. VLSI,* pp. 70-79, Feb. 2003.

[6] T.-Y. Wuu and S.B Vrudhula, "A Design of a Fast and Area Efficient Multi-Input Muller C-Element," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 1, no. 2, pp. 215-219, June 1993.

[7] R.B. Reese, M.A. Thornton, and C. Traver, "Arithmetic Logic Circuits Using Self-Timed Bit-Level Dataflow and Early Evaluation," *Proc. 2001 Conf. Computer Design,* pp. 18-23, Sept. 2001.

[8] S. Furber and P. Day, "Four-Phase Micropipeline Latch Control Circuits," *IEEE Trans. VLSI Systems,* vol. 4, no. 2, pp. 247-253, June 1996.

[9] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, L. Lwin, and C. Sotiriou, "Handshake Protocols for De-Synchronization," *Proc. 10th Int'l Symp. Asynchronous Circuits and Systems,* pp. 149-158, Apr. 2004.

[10] I. Sutherland, "Micropipelines," *Comm. ACM,* vol. 32, no. 6, pp. 720-738, June 1989.

[11] J. Campos, G. Chiola, J. Colom, and M. Silva, "Properties and Performance Bounds for Timed Marked Graphs," *IEEE Trans. Circuits and Systems,* vol. 39, pp. 386-401, May 1992.

[12] T. Murata, "Petri Nets: Properties, Analysis and Applications," *IEEE Proc.,* vol. 77, pp. 541-580, Apr. 1989.

[13] A. Yakovlev, M. Kishinevskh, A. Kondratyev, and L. Lavagno, "On the Models for Asynchronous Circuit Behavior with OR Causality," Technical Report Series No. 463, Computing Science, Univ. of Newcastle upon Tyne, Nov. 1993.

[14] 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," *Proc. 17th Conf. Advanced Research in VLSI,* pp. 164-181, 1997.

[15] J.D. Garside, S.B. Furber, and S.B. Chung, "AMULET3 Revealed," *Proc. Async. '99,* pp. 51-59, Apr. 1999.

[16] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann, "An Asynchronous Low-Power 80C51 Microcontroller," *Proc. Async '98,* pp. 96-107, Mar. 1998.

[17] R. Reese, M. Thornton, and C. Traver, "Fast Two-Phase Micropipeline Control Wrapper for Standard Cell Implementation," *Electronics Letters,* vol. 40, no. 4, pp. 227-229, Feb. 2004.

[18] R. Reese, M. Thornton, and C. Traver, "Two-Phase Micropipeline Control Wrapper with Early Evaluation," *Electronics Letters,* vol. 40, no. 6, pp. 365-366, Mar. 2004.

[19] J. Garside private communication, Feb. 2003.

[20] F.E. Anderson, J.S Wells, and E.Z. Berta, "The Core Clock system on the Next Generation Itanium1 Microprocessor," *Digest of Technical Papers, Int'l Solid State Circuits Conf. (ISSCC 2002),* vol. 1, pp. 146-148, 2002.

[21] D. Harris and H. Naffziger, "Statistical Clock Skew Modeling with Data Delay Variations," *IEEE Trans. VLSI,* vol. 9, no. 6, pp. 888-898, Dec. 2001.

Robert B. Reese (S'77, M'80) received the BS degree from Louisiana Tech University, Ruston, in 1979 and the MS and PhD degrees from Texas A&M University, College Station, in 1982 and 1985, respectively, all in electrical engineering. He served as a member of the technical staff of the Microelectronics and Computer Technology Corporation (MCC), Austin, Texas, from 1985 to 1988. Since 1988, he has been with the Department of Electrical and Computer Engineering at Mississippi State University, where he is an associate professor. Courses that he teaches include VLSI systems and digital system design. His research interests include self-timed digital systems and computer architecture. He is a member of the IEEE.

Mitchell Aaron Thornton received the BS degree in electrical engineering in 1985 from Oklahoma State University, the MS degree in electrical engineering in 1990 from the University of Texas at Arlington, the MS degree in computer science in 1993 from Southern Methodist University, and the PhD degree in computer engineering from Southern Methodist University in 1995. He has worked in industry for E-Systems, Inc. and Cyrix Corporation and he is currently an associate professor in the Departments of Computer Science and Engineering and Electrical Engineering at Southern Methodist University. His research interests include self-timed digital circuit synthesis and verification of digital systems. He is a senior member of the IEEE.

Cherrice Traver received the Bachelor of Science degree in physics (summa cum laude) from the State University of New York at Albany in 1982. Her PhD degree in electrical engineering is from the University of Virginia (1988). She joined the faculty at Union College in Schenectady, New York, in 1986, where she is currently a professor of electrical and computer engineering. Her research interests are in the area of timing methodologies for digital circuits. She is a senior member of the IEEE.