A Standard Cell Implementation of a Phased Logic CPU

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

Mitch A. Thornton
Southern Methodist University
Computer Science and Engr.
mitch@engr.smu.edu

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

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 UMC 0.18μ and Artisan /IBM 0.13μ standard cell libraries, and a five-stage pipelined MIPs-compatible integer unit mapped to the Artisan/IBM 0.13μ library. The design styles 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.

Categories and Subject Descriptors
B.7.1 [Integrated Circuits] Types and Design Styles – *microprocessors and microcomputers*; B.6.1 [Logic Design] – Design Styles

General Terms
Design, Performance.

Keywords
Asynchronous, Phased Logic, micropipeline, self-timed, two-phase, microprocessors.

1. INTRODUCTION
Proponents of self-timed design have repeatedly touted advantages such as lower 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 double-precision floating-point clip operation. The CPU implementation uses an Artisan standard cell library and register file generator designed for an IBM 0.13μ technology. The clip operation is mapped to both Artisan 0.13μ and UMC 0.18μ libraries. No extra cells were added to either the Artisan or UMC libraries even though this could have increased the efficiency of the PL implementation. A PL implementation is a two-phase micropipelined system that uses bundled-data signaling. A PL netlist is produced by an automated translation from the clocked netlist, which allows the designer to use familiar languages and tools for producing the clock 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% 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 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 Clocked to PL Transformation

A marked graph is a directed graph consisting of edges, vertices, and tokens. A vertex will fire if all input edges have a token count that is positive; upon firing, the token count of all input edges is decremented by one, and the token count of all output edges is incremented by one. In a PL coarse-grain netlist, an edge is represented by a data bundle that consists of multiple data wires and one phase wire (similar to a "request" wire in other asynchronous methodologies). The phase value can either be EVEN (0) or ODD (1). Each PL block also contains an internal state element called the block phase, which is either EVEN or ODD. If the phase of an input edge matches the block phase, then that edge is said to contain a token. A PL block will fire if all input edges have tokens; firing toggles the internal gate phase, and toggles all output phases. Upon firing, all data wires in output bundles are updated with new values.

The starting point for a coarse-grain clocked-to-PL netlist transformation is a hierarchical clocked netlist, in which the 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 is it called a *through* block. In the marked graph equivalent, a barrier block will have an initial token on its output, and a through block will not. The assignment of initial tokens is simply a netlist wiring decision. Each PL block has a phase output that is the same value as the internal gate phase, and a phase output that is the logical negation of the gate phase. A connection between two blocks in the clocked netlist designates an edge in the marked graph equivalent. All signals between two blocks are placed in one data bundle and assigned one phase wire. At reset, all block phases are set to EVEN. Figure 1 shows the simplest possible PL system, which consists of one barrier block and one through block.

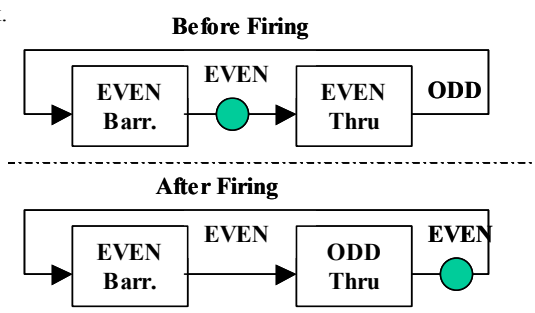

Figure 1: Two Block PL System

A marked graph is *live* if every edge belongs to a directed circuit C that contains at least one token (m(C) > 1). A marked graph is *safe* if every edge belongs to at least one directed circuit that contains only one token (m(C) = 1). The vertices in a live and safe marked graph will fire in a continuous cyclic manner. The PL system in Figure 1 is both life and safe. Figure 2 shows a PL system that has an edge that is not part of a directed circuit, which means that block C will never fire after the initial firing. The transformation process detects this liveness problem and adds an additional control signal called a *feedback* (aka acknowledgement) to make this edge part of a directed circuit with a token count of one. A feedback signal does not have a data bundle associated with it. The initial token marking rules do not allow barrier-to-barrier block connections; a through block is inserted automatically by the transformation process to break a barrier-to-barrier block connection.

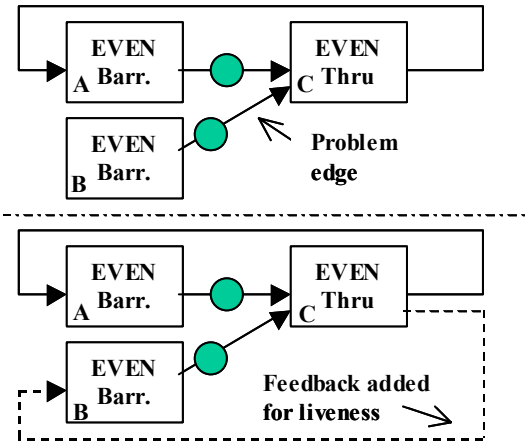

Figure 2: Adding Feedback

2.2 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 gate fires after all early inputs have arrived. An early evaluation block has separate signals for output phase (Op) and output feedback (Fo). In non-early evaluation blocks the output phase also serves as the output feedback. The feedback output is not updated until all inputs have arrived.

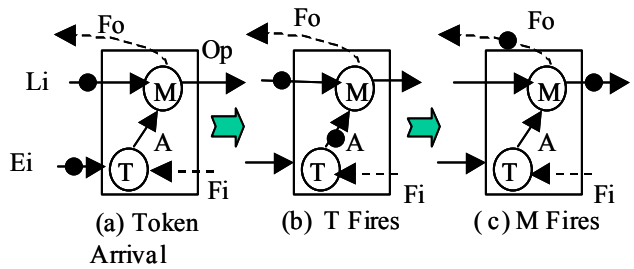

Figure 3. Normal Fire Case for EEnode

Figure 3 shows a two node marked graph model that used by the mapping algorithm to represent an early evaluation block. 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 Figure 3. 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 Figure 4. 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.

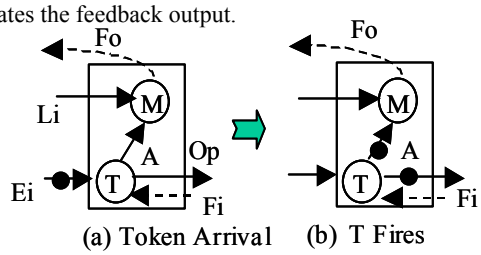

Figure 4. Early Fire Case for EEnode

Figure 5 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.

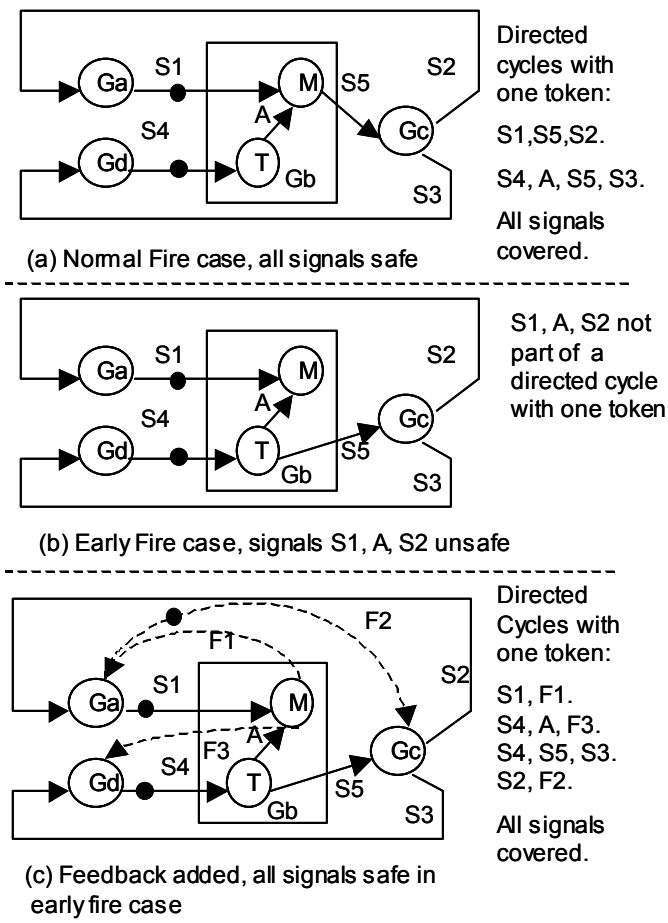

Figure 5. Safety in PL netlist with EEnodes

In Figure 5(a), the graph is live and safe if the normal fire configuration of the EEnode is used. However, using the early fire configuration of Gb in Figure 5(b) 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). Figure 5(c) adds feedback signals F1, F2, and F3 to make the graph live and safe. Two rules for feedback insertion in the presence of EEnodes are evident from Figure 5:

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.
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.

A third rule that is not as evident from Figure 5 is:

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 than 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 [12]. However, using one of these representations means that the simple properties of liveness and safety of marked graphs are lost. So, how do we keep a marked graph model, and yet still account for choice as represented by an EEnode?

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-EE node 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, than 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 non-feedback output edges 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 gate 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 1: 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(C) = 1 affected by the early-to-normal configuration change of u_i are the ones 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*(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 remain unchanged, at m(C) = 1.

Theorem 1 can be trivially extended to cover any number of nodes U*k* performing early-to-normal configuration changes as each directed cycle with m(C) = 1 can only have one node ready to fire, so the cycles with m(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', mi') reachable from (G, m0) by non-EE node firings, EE node early firings, or EEnode early-to-normal configuration changes is live and safe. The next theorem covers normal-to-early configuration changes:

Theorem 2: From any marking m_k' reachable from graph (G', mi') by non-EEnode firings, EEnode early firings or EE node 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(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(C) > 0. As these directed circuits have m(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(C) = 1. In the original graph (G, m_0), the A arc had to be covered by a directed circuit with m(C) = 1 that included a feedback output of the EEnode, and configuration changes of the EEnode does not affect this cycle.

Theorem 2 can be trivially extended to cover any number of nodes U*k* performing normal-to-early configuration change by the same reasoning used to extend Theorem 1. This means that given a starting marked graph (G, m0) that is live and safe, then any marked graph (G', mi') reachable by non-EEnode firings, EEnode early firings, EEnode late firings, EE node early-to-late or late-to-early configuration changes is also live and safe.

3. Related Work

The 'de-sync' self-timed design style [10] is the most similar to our coarse-grain design style in that it uses a coarse-grained micropipeline with bundled-data signaling, uses a commercial standard cell library, begins with a clocked netlist and replaces the global clock network. The micropipelined implementation uses four-phase control [9] built from standard cells. For the DLX processor implementation (without forwarding) described in [10], the asynchronous design has equivalent area, performance and power consumption when compared to the clocked design. The main difference between our approaches is that the de-sync approach has not been shown to support the concept of early evaluation, which has 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 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 from the Artisan standard cell library 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.

Null Convention Logic (NCL) [8] is a fine-grain, delay-insensitive, four-phase approach that allows the use of a commercial synthesis tool for mapping combinational logic to a netlist of NCL gates. Registers and acknowledgement logic are specified separately from the combinational logic register transfer level (RTL) VHDL specification. Delay insensitivly between NCL gates is achieved by a TRUE, FALSE, NULL dual rail signaling method. This approach has a minimum 2X area penalty, requires the use of a custom library that implements the NCL logic family, and has a fairly high performance penalty when compared to a clocked implementation. There is no equivalent to early evaluation in the NCL approach.

4. The Coarse-Grain PL Methodology

4.1 Tool Flow

Figure 6 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. A slack-matching buffer can improve performance in some cases if a block output feeds several other blocks that complete at different times. Having this tool automatically insert the slack-matching buffer frees the user from having to 'pollute' their top level RTL with slack buffers. The output netlist can still be simulated as a clocked netlist. An automated partitioning capability is planned for future versions that would partition a flat netlist into blocks for PL wrapper encapsulation. 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 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.*

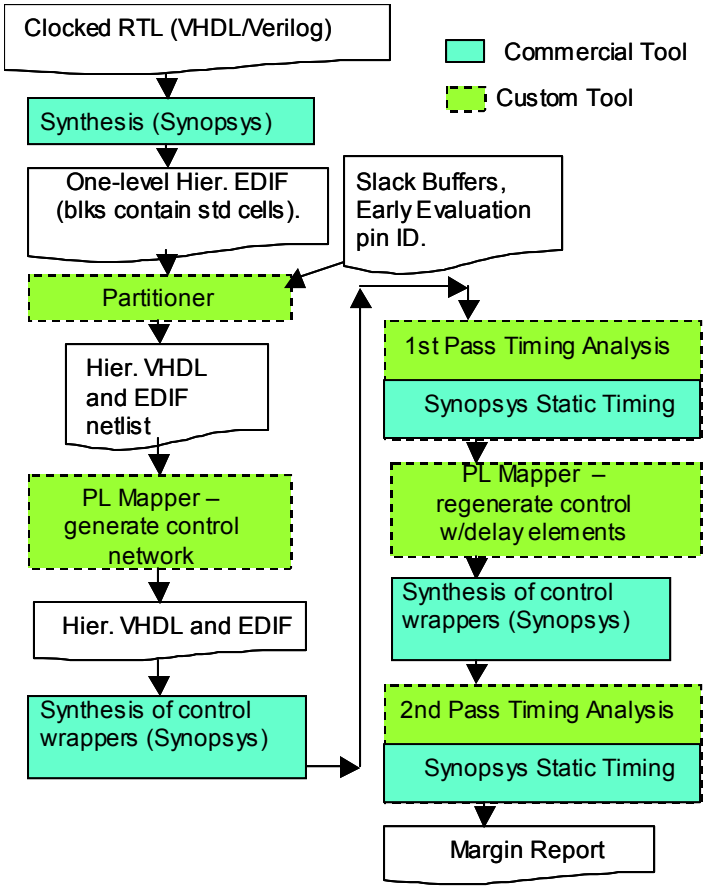

Figure 6: PL Coarse-Grain Methodology Flow

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 methodology.

(2) Early evaluation opportunities must be identified by the designer. In the PL MIPS, early evaluation was used in blocks with inputs from external memory and/or branch PC as these inputs were not often required, and had longer latency compared to other inputs. 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.

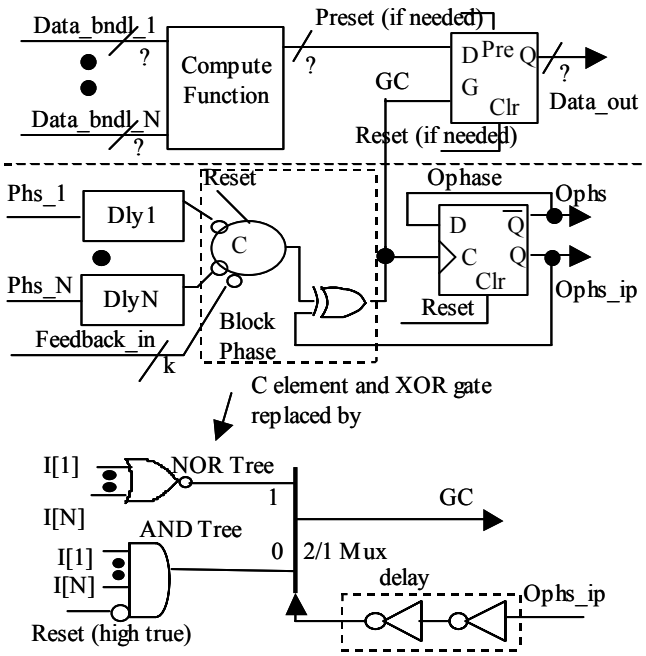

Figure 7: PL Control Wrapper (no early eval)

Figure 7 shows the PL control wrapper used for non-early evaluation blocks and it's interface to the datapath block. This is simply a variation on the traditional control used for micropipelines [11]. Each delay block for a phase input is chosen to match the datapath delay of the corresponding data bundle. The delay block is a chain of delay cells where the delay cell is provided as a standard cell by the Artisan library. 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 the Artisan library. This 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. 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.

Figure 5 shows the PL control wrapper with early evaluation capability. A normal fire occurs when EE_sel = '0' after all inputs have arrived; both output phase and feedback phase signals are

updated. An early fire occurs when EE_sel = '1' and all inputs have arrived to the trigger phase C-element; this updates the output phase but not the feedback phase.

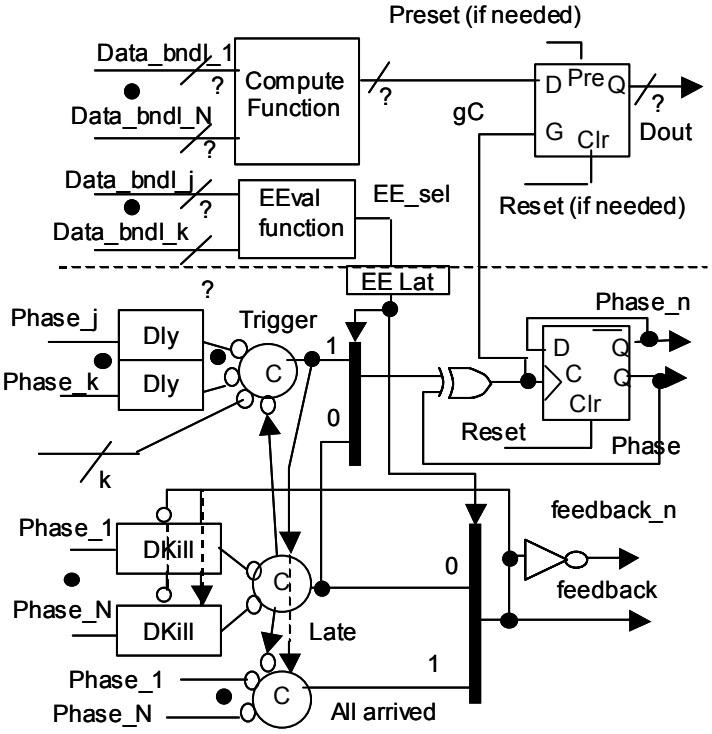

Figure 8: PL Control Wrapper with Early Evaluation

After an early fire, the feedback output phase is updated once all of the late inputs (Phs_1 – Phs_N) arrive. The *Fb_Phase* 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 Phs_1–Phs_N inputs could arrive while old inputs are still traversing delay blocks, causing input hazards to the *Late_Phase* C-element.

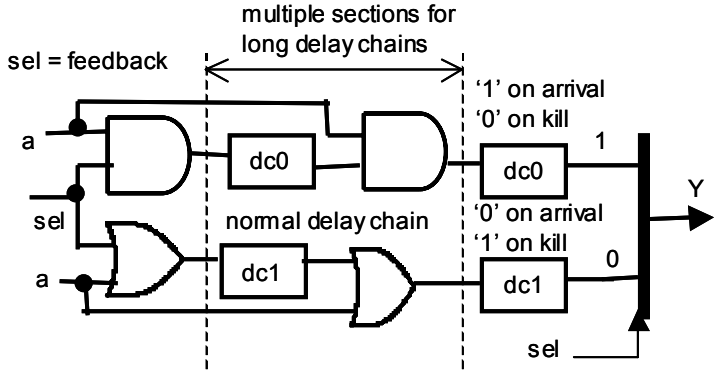

Figure 9: DlyKill Block Internals

To avoid this problem, the *DlyKill* block uses two delay chains internally as shown in Figure 6. The toggling of the *Fb_C* 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 a+ → N1+ (sel = 1) or a- → N0- (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 *late* C-element has to be an input to the *trigger* and *all-arrived* C-elements to prevent firing of these C-elements before the late C-element has caught up to these states. The *EEsel* latching logic (detail not shown) holds the *EEsel* signal stable as long as the late C-element and trigger 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 pre-layout 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 an Artisan standard cell library targeted for an IBM 0.13μ technology. The register file is implemented using the Artisan two-port register file generator.

Figure 10 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 Artisan 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 Figure 7 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 use 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 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 de-synchronize 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.

5.2 CPU Performance Results, Area Values

Table 1 gives the performance results of the PL CPU compared to the clocked CPU for five benchmark programs. 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 pre-layout timing delays. Synopsys static timing reported the register-to-register critical path of the clocked design as 2170 ps; to this value was added a 3% clock skew budget of 65 ps for a total clock period of 2235 ps. For the PL netlist, a target timing margin of 20% was specified for delay chain generation, with a 10% minimum timing margin allowed for failure flagging for margin checking in the 2nd timing pass. The minimum margin reported for any path after 2nd pass timing was 12%. All benchmarks were written in C and compiled using gcc.

The non-reordered 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 show below, 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.

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

Sequence A

Sequence B

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

In [2], an average speedup of 41% 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:

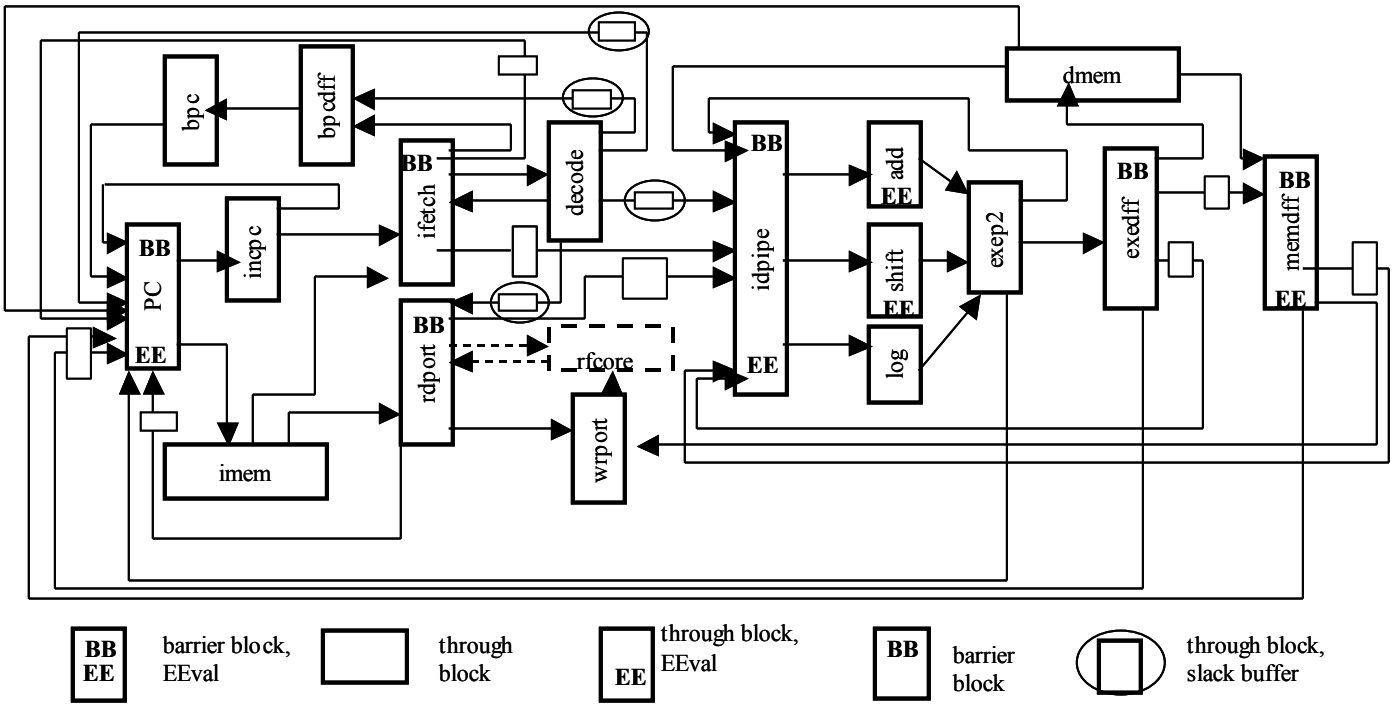

Figure 10: CPU Architecture Diagram

- 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.

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

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 shows that the PL design has an increase of 47% 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.

Table 2: Area Results

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

5.3 Double-Precision Floating Point Clip

A double precision floating point clip operation defined as:

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

was implemented in several ways and mapped to both UMC 0.18µ and Artisan /IBM 0.13µ standard cell libraries. Figure 11 shows the variations

a. Multi-cycle 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 are not.

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 (b).

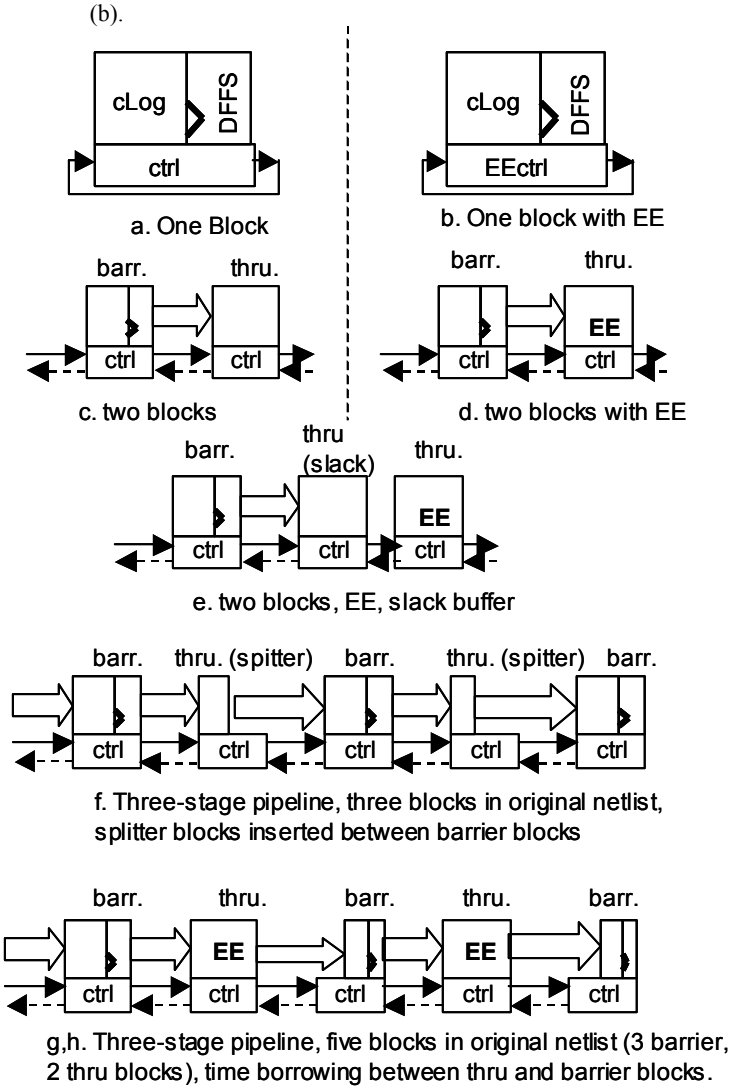

Figure 11: DP FP Clip variations

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 5 blocks, with the stage2 and stage3 blocks of (f) 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 thru and barrier blocks when creating delay chains for the thru 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 was done in the first stage and then passed to the successive stages

Table 3 (after references) shows the mapping and simulation results for the Artisan 0.13μ and UMC 0.18μ libraries. The designs were tested with 1000 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 most interesting aspect of the numbers in Table 3 is the disparity in the PL/CLK area/speed comparisons between the Artisan and UMC libraries. For the UMC 0.18μ libraries, the EE versions of the PL designs achieved speedup; for Artisan libraries, no speedup was obtained. The reason for this disparity is that the Artisan 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 UMC 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 Artisan or UMC 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 Artisan 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 thru and barrier blocks during delay block creation.

6. CONCLUSIONS

This paper describes the first coarse-grain PL designs mapped to commercial standard cell libraries. These results show that more work needs to be done in terms of improving wrapper designs, adding automatic partitioning of top level netlists to remove the partitioning burden from the designer, and experimentation with larger designs.

7. REFERENCES

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

[2] Reese. R.B., Thornton, M.A., and Traver, C. A Coarse-grained Phased Logic CPU, Ninth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2003), Vancouver, BC, Canada, May 2003, pp 2-13.

[3] Wallander, W. *A VHDL Implementation of a MIPS*, Project Report, Dept. of Computer Science and Electrical Engineering, Luleå University of Technology, http://www.ludd.luth.se/~walle/projects/myrisc.

[4] Commoner, F., Holt, A.W, Even, S., and Pneuli, A. 1971. Marked Directed Graphs, *Journal Computer and System Sciences*, Vol. 5, No 5, 511-523.

[5] Reese. R.B., Thornton, M.A., and Traver, C. A Fine-grain Phased Logic CPU. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, Tampa, FL, February 2003, 70-79.

[6] Wuu, Toy-Yung Wuu and Vrudhula, S. B. A Design of a Fast and Area Efficient Multi-Input Muller C-element, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol 1, No. 2, June 1993.

[7] Reese, R.B., Thornton, M.A., and Traver, C. Arithmetic Logic Circuits using Self-timed Bit-Level Dataflow and Early Evaluation, In *Proceedings of the International Conference on Computer Design*, Austin, Texas, September 2001, 18-23.

[8] Ligthart, M., Fant, K., Smith, R., Taubin, A., and Kondratyev, A. Asynchronous Design Using Commercial HDL Synthesis Tools, In *Proceedings of the Seventh International Symposium on Asynchronous Circuits and Systems*, Eilat, Israel, April 2000, 114-125.

[9] Furber, S. and Day P. Four-phase micropipeline latch control circuits, In IEEE Transactions on VLSI Systems, Vol. 4, No. 2, June 1996, 247-253.

[10] Blunno I., Cortadella J., Kondratyev A., Lavagno L., Lwin L., and Sotiriou, C. Handshake Protocols for de-synchronization, In *Proceedings of the Tenth International Symposium on Asynchronous Circuits and Systems,* Crete, Greec, April 2004, 149-158.

[11] I. Sutherland, Micropipelines, *Communications of the ACM*, Vol 32, No. 6, June 1989, pp. 720-738.

[12] A.Yakovlev, M. Kishinevskh, A. Kondratyev, L. Lavagno, "On the models for asynchronous circuit behavior with OR causality", Technical Report Series No. 463, Computing Science, University of Newcastle upon Tyne, November 1993.

Table 3: DP FP Clip Operator Mapping/Simulation Results

IBM 0.13		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, time brw,EE)	PL	4020	63190	1.26	n/a	1884	1.32	16.1%	15.5%
UMC 0.18									
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, time brw,EE)	PL	3535	135364	1.60	n/a	3434	0.83	20.4%	15.3%