Arithmetic Logic Circuits using Self-timed Bit Level Dataflow and Early Evaluation

Robert B. Reese, Cherrice Traver, and Mitch A. Thornton

Abstract-- **A logic style known as *Phased Logic(PL)* is applied to arithmetic circuits. Phased logic is a dual-rail LEDR logic style that allows automatic translation from a clocked netlist to a self-timed implementation. Bit level dataflow, early evaluation and automatic filtering of transient computations within PL circuits can lead to both increased performance and higher energy efficiency than the original clocked netlist. Simulation results for a 16x16 iterative multiplier based on a LUT4 design show a 23% speed improvement and 20% energy improvement over the clocked design. A Y= Y@1*a +b calculation using an array multiplier design shows a 15% performance decrease but is 2X more energy efficient than the clocked counterpart.**

Index Terms--**self-timed, dataflow, asynchronous, arithmetic**

I. Introduction

THROUGHOUT the ITRS-99 Roadmap on Design, continual references are made to clock and timing related challenges facing designers through and beyond 2005. Many of the problems stem from simply taking the current design methodology for computing systems (a global clock between components, with higher rate clocks within components) and projecting this usage within the System-on-a-Chip circa 2005 and beyond. In [1], [2] an innovative clockless design methodology called Phased Logic (PL) was introduced. Phased Logic is based upon LEDR signal encoding [6] and marked graph theory [7]. One of the features of Phased Logic is that it produces a clockless design via an automated translation from a netlist of DFFs + combinational logic. This paper demonstrates Phased Logic as applied to arithmetic circuits, principally addition and multiplication.

II. A Brief Overview of Phased Logic

A. Phased Logic Operation

A phased logic netlist can be thought of as a marked graph with data tokens flowing throughout the graph. Each data token has a *phase* that is either *even* or *odd*. A data token is represented by a dual-rail signal that uses LEDR encoding [4]. A phased logic gate has an internal state bit used to represent the gate phase and a phased logic gate *fires* whenever all of the phases of the inputs matches the internal gate phase. In [1] Linder showed that for correct operation of a phased logic system, its marked graph equivalent had to be both *live* and *safe*. A live marked graph has an active token on each directed circuit of the graph and every signal must be part of a directed circuit. This essentially means that each directed circuit in the phased logic netlist must have at least one PL gate ready to fire at any time. A graph with a liveness problem will result in no token circulation, and hence no activity in the PL system. A safe marked graph is one in which each directed circuit has only one active token on it at a time. This means that there can only be one PL gate ready to fire within a given directed circuit. A graph with a safety problem will result in incorrect operation because multiple tokens on a directed circuit can overwrite each other. During a computation cycle along a directed circuit in a phased logic netlist, all gates will change phase. Token circulation continues in a PL system even if the value of the tokens are unchanging; each computation cycle will simply change the phases of the tokens and gates.

The mapping of a clocked netlist (D-Flip-Flops + combinational logic) to a phased logic netlist consists of the following operations:

- All gates are replaced by PL gate equivalents. A PL gate equivalent is the original combinational logic function plus the control logic necessary for gate firing. A DFF in the original netlist can be absorbed into the combinational gate that provides the DFF's input signal.
- Feedback signals are synthesized for the PL netlist to ensure safety and liveness of the resulting netlist. Feedback signals carry no value information, only phase information and thus result in single wires. The details of feedback generation are described in [1], [2], [3]. Muller C-elements [9], [10] are used to concentrate feedbacks at a particular gate when needed.

III. Phased Logic System Design

A. A LUT4-based PL Gate

A four input PL gate (*pl4gate*) that uses a *LookUp Table* (LUT) as a compute element is shown in Figure 1. (this is actually a variation of a design presented in [8] that also used a LUT4 LEDR computation element). The Muller C-element (C-gate) contains the gate phase, and the toggling of this element activates the output latches to capture the new output value from the LUT. The *dly* block indicates a delay-matched path in which it is assumed that the output of the LUT4 is stable when the output latch gate control is asserted. From a power perspective, it is important to note that if the value bits remain stable, then only the control elements switch during a phase change. The gate shown in Figure 2 is the basic gate used in the simulations presented in this paper. The *fi* input is used as a dedicated feedback input. Unused inputs can also be used for feedback inputs. It is assumed that each PL4gate will also have a 4-input Muller C-element paired with it to be used as a resource for

This work was supported in part by an internal grant from the MSU/NSF Engineering Research Center..

Robert B. Reese is with the Electrical and Computer Engineering Department, Mississippi State University, Starkville MS, USA, (e-mail:reese@ece.msstate.edu).

Cherrice Traver is with the Electrical Engineering and Computer Science Department, Union College, Schenectady N.Y., USA, (e-mail:traverc@doc.union.edu).

Mitch A. Thornton is with the Electrical and Computer Engineering Department, Mississippi State University, Starkville MS, USA, (e-mail:mitch@ece.msstate.edu).

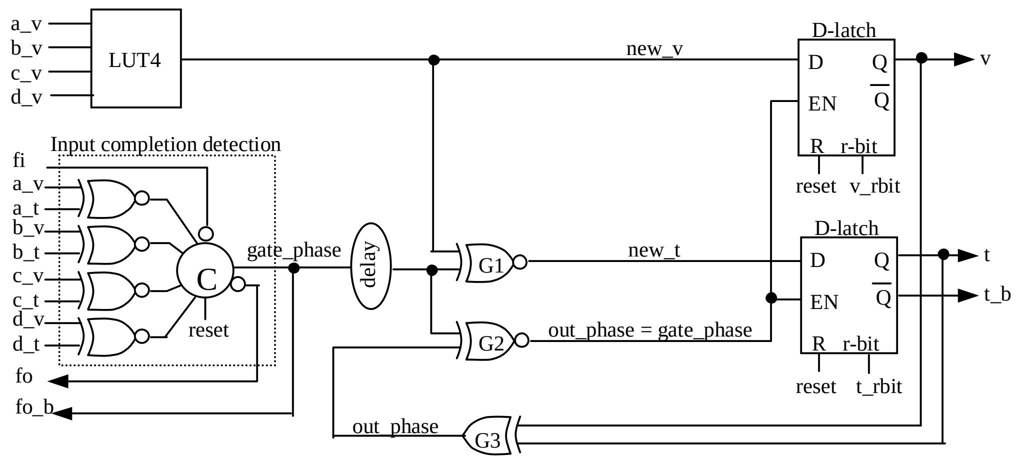

Figure 1: A LUT4-based PL Gate

feedback concentration for this gate or any other gate.

B. *Early Evaluation*

It can be seen from Figure 2 that the output latch delay is included in the critical delay path of the gate. The ratio of the output delay to compute block delay (in this case, the LUT4 delay) will determine a gate-level performance penalty when comparing PL systems to clocked systems. Fortunately, PL has some system level features that allow it to overcome this gate level penalty. Early evaluation is a speedup mechanism by which a PL gate is allowed to fire when only a subset of its inputs has arrived. This allows PL circuits to implement data dependent computation which is a well known benefit of self-timed systems.

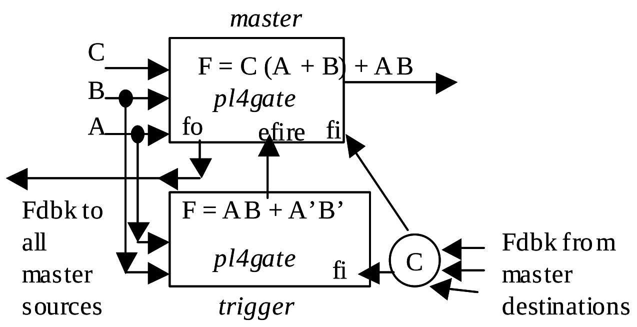

Figure 2: Early Evaluation Pair

Figure 2 illustrates how two PL4gates (termed a *master* and a *trigger*) can be combined to form one early evaluation PL4gate. The master contains the normal evaluation function while the trigger contains the early evaluation function. When the early evaluation function is true, then the master gate is fired and the current value of the master LUT4 is used. Obviously the trigger gate should be based upon early arriving signals, and the trigger function should depend upon a subset of the master function signals such that the late arriving signals are don't cares.

C. *Bit Level Dataflow*

Another system level speedup mechanism is bit-level dataflow. Because of the fine grained nature of the feedback signals, all available parallelism within the PL system is used automatically during its operation.

Figure 3 shows two systems whose only difference is the number of parallel stages surrounding a ripple structure. The input and output of both systems is word synchronized, but token flow occurs at the bit level within the system. Because the input and outputs are synchronized, the top circuit in Figure 4 can suffer from delays imposed by synchronization. The addition of parallel stages before and after the inputs serve to increase bit level dataflow and the stages can simply be buffers or elements performing useful computation. Simulation results indicate the average throughput of the top system is 4 gate delays, while the average throughput of the bottom system is 2.3 gate delays. Static figures cannot convey this flow mechanism adequately; Macromedia Flash animations of these two systems are available at [11].

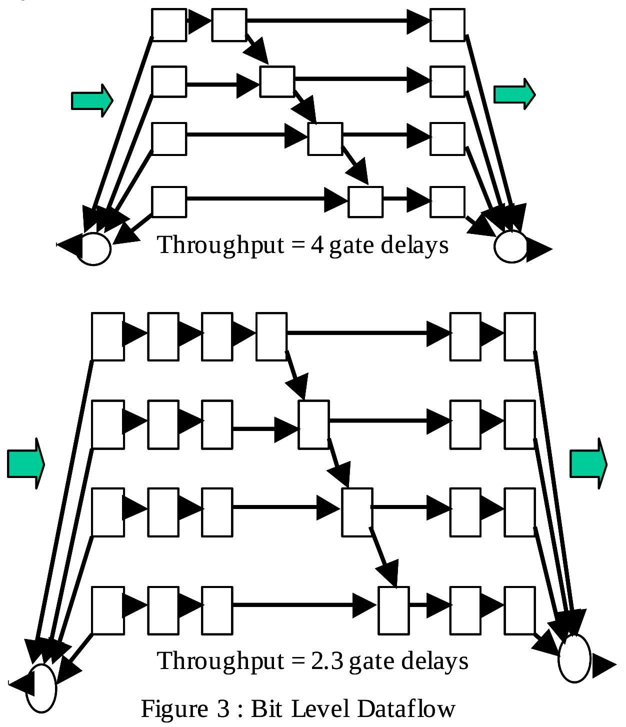

Figure 3 : Bit Level Dataflow

IV. Design Examples

This section discusses three example designs used to compare clocked and PL LUT4-based netlists for power and performance. All designs were specified in VHDL RTL and synthesized to a LUT4 netlist via Synopsys Design Compiler. The LUT4 VHDL netlist was used as the basis of the clocked simulations. The LUT4 VHDL netlist was then converted to EDIF for input to our mapping tool that outputs a VHDL netlist of *pl4gates*. Optimized LUT4 structures for addition and multiplication structures were installed in a Synopsys DesignWare library.

A. Comparing Clocked versus PL designs

One of our goals in implementing these examples was not only to verify functionality of our mapping program, but also to get some rough estimates of how a PL implementation compares to a clocked implementation in terms of performance and power. Our examples have all inputs and outputs registered, so the performance of the clocked system will be the longest register-to-register path as reported by Synopsys. A LUT4 was assigned a normalized delay of 1.0, with the sum of the Clock-To-Q (Tcq) DFF delay + DFF setup delay (Tsu) also estimated as a LUT4 delay (1.0). Clock skew and wiring delays were ignored for both systems. Ignoring wire delays penalizes the PL systems in performance comparisons because wire delays would only help reduce the PL4gate output latch delay penalty as percentage of the total delay. The PL4gate delay was estimated as 1.4 (ie., the output latch in the pl4gate adds a 40% delay penalty). These ratios were chosen by using specifications from the Altera and Xilinx data sheets for LUT4, TCQ and Tsu ratios. The value of 1.4 is the average these values. Four-input Muller C-elements were also used in the PL netlist for feedback concentration, and these were assigned a delay value of 0.6. The performance of the PL system is measured by simulating the netlist using 600 input vectors and calculating the resultant average computation delay. Variable computation delays are present in any PL netlist using early evaluation. The input vector values are digitized sine wave values that cover full range (in amplitude) and several cycles -- each successive cycle used a different offset that ensured that two cycles did not duplicate input vectors.

For power comparisons, the VHDL simulations are instrumented to track compute and control signal transitions. A compute transition is counted as any change of value on a LUT4 input (simultaneous or near simultaneous arrivals were only counted once). For the clocked netlist, an active clock edge arrival at a D-Flip-Flop is a control transition. For the PL netlist, the firing of a PL gate is a control transition. From Altera Apex [12] and Xilinx Virtex [13] FPGA power estimation spreadsheets for 0.18µ technologies, a LUT4 is estimated to switch approximately 1.05 pF, and a D-Flip-Flop 0.14 pF. HSPICE simulations based on a 0.25µ technology of the *pl4gate* control indicate that it switches approximately 0.2 pF per firing (this value would be expected to shrink somewhat for a 0.18µ process but we will use this somewhat inflated value so as to help factor in the output token phase or 't' bit wiring capacitance). Four-input Muller C-elements were found to switch 0.75 pF per output change. The 'v' bit (value bit) wiring capacitances were ignored since the value wire switching capacitance of the PL systems will be equal or less than that of the clocked system (less if the clocked system has transient computations). We can estimate the amount of capacitance switched per computation using these control and compute capacitances in conjunction with the transition counts. An energy figure of merit (work per sample) can be obtained by multiplying the delay per sample by the capacitance per sample. 16 bits of Y@1 is used). The 16 x 16 multiplier design used an un-optimized Carry-Select Addition (CSA) array [14], with the final carry/sum merge accomplished using a CLA in the clocked design and a ripple adder with early evaluation in the PL design.

B. 32-bit Accumulator

The first design example is a 32-bit accumulator with a synchronous clear. A carry look-ahead adder is used for the clocked netlist and a ripple carry adder with early evaluation is used for the PL netlist. The CLA adder was used in all clocked designs because of its obvious performance advantage over a ripple adder.

C. 32-bit Iterative Multiplier

Figure 4 shows the second design, a shift/add 16x16 iterative multiplier using a single adder.

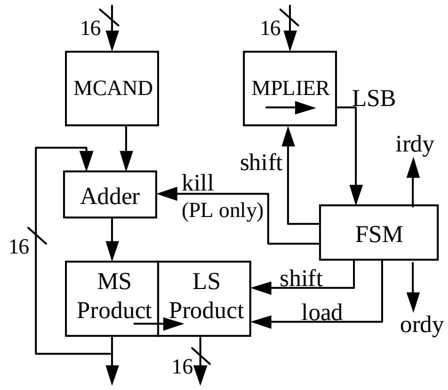

Figure 4: 16x16 Iterative Multiplier

Again, the clocked design used a CLA while the PL design used a ripple adder with early evaluation. The PL design had an extra speedup path in that a kill line was used to early fire every carry bit in case the multiplier bit was '0'.

D. Filter Calculation Y = Y@1 * A + B

Figure 5 shows the datapath used for the third design example. The 16 x 16 multiplier design used an un-optimized Carry-Select Addition (CSA) array [14], with the final carry/sum merge accomplished using a CLA in the clocked design and a ripple adder with early evaluation in the PL design.

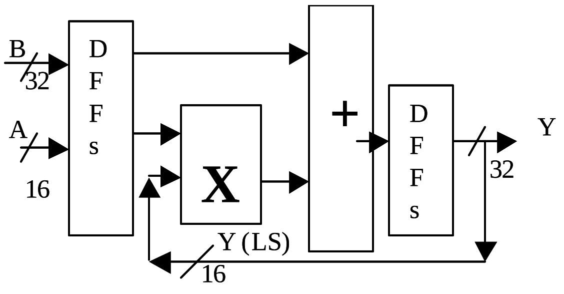

Figure 5: Y = Y@1*A + B

V. Design Comparisons: Clocked vs. PL

A. Performance, Energy Comparisons

Table 1 gives the delay, capacitance and energy per sample measured from the VHDL gate level simulations. The delay values are normalized to LUT4 delays. The energy value is simply a figure of merit for work obtained by multiplying the capacitance column times the delay column, and dividing by a constant scale factor. The accumulator design is designated as 'acc', the iterative multiplier as 'mult', and filter calculation as 'filt'. In all cases, the PL designs are more energy efficient than the clocked designs. The PL iterative multiplier actually switched slightly more capacitance than the clocked design, but was more energy efficient due to higher performance. The PL filter application was slower than the clocked netlist, but was more energy efficient due to less capacitance switched per sample.

Design	dly(LUT4s)	cap(fF)	Energy	%diff
Clk (acc)	12	205	24.7	
PL (acc)	8.2	193.1	15.8	-36.0%
Clk (mult)	187	3415	6385	
PL (mult)	151	3557	5357	-16.1%
Clk (filt)	29	2286	663	
PL (filt)	33	976	322	-51.4%

Table 1: Performance values for Design Examples

In looking at the performance figures of Table 1, the ripple early evaluation capability allows the PL designs to overcome the 40% gate delay penalty and to outperform the clocked LUT4 netlists. We are aware that both Xilinx and Altera include fast carry generation logic within their cells, while our clocked netlists had the carry logic as a dedicated LUT4. Obviously, the carry logic along with the early evaluation for the carry could be integrated into the *pl4gate* design. The exact effect upon the performance and capacitance values in Table 1 is an area of future study.

In Table 1, the PL circuits switched less capacitance in 2 out of 3 designs due to a reduction in compute transitions. This reduction comes from the different adder structures (CLA versus ripple) and also from of filtering of transient computations. Table 2 shows the total transition counts for the three designs.

Design	Compute	%diff	Control	Cgate
Clk (acc)	110133		38115	0
PL (acc)	81031	-26.4%	112135	80454
Clk (mult)	1806621		791863	0
PL (mult)	1155446	-36.0%	3735097	1750466
Clk (filt)	1273941		46960	0
PL (filt)	436545	-65.7%	439122	365142

Table 2: Transition Counts for Design Examples

The large compute transition savings in the filter example is mainly due to the CSA multiplier array, which tends to propagate transient computations.

While PL designs can reduce compute transitions, PL dramatically increases control transitions since there is now control in every cell, and every cell changes phase during a compute cycle. Because of this control overhead, it is critical that the ratio of compute capacitance to control capacitance be large. In our LUT4 based *pl4gate* design, this ratio is 5 to 1 and we believe that this might be near the lowest ratio in order for a PL system to be power competitive with clocked control.

All of the PL blocks used early evaluation and thus their computations are data dependent. Table 3 shows the delay statistics for the phased logic designs, where delay values are in LUT4 delays.

Design	AvgDly	MinDly	MaxDly	Std Dev
PL (acc)	8.2	4.2	20.2	2.4
PL (mult)	151	123	177	8.3
PL (filt)	33	27	42	2.4

Table 3: Delay Statistics on Phased Logic Simulations

Obviously, if a PL system encounters worst-case data patterns for a significant amount of the time, then a ripple adder with early evaluation will not be the best choice. We have evaluated a few different adders structures (carry-skip with early eval, and CLA with early eval) that reduce worst-case delay. Unfortunately, these structures tend to also negatively impact the average delay. This is an area that also needs further study.

B. Delay Analysis

Table 4 shows the values in Table 3 in terms of *pl4gate* delays instead of LUT4 delays (1 pl4gate delay = 1.4 LUT4 delays).

Design	AvgDly	AvgDly (Exp)	MinDly	MaxDly
PL (acc)	5.7	8.0	3.0	13.4
PL (mult)	107.9 (6.3)	170(10)	87.9 (5.2)	126.4 (7.4)
PL (filt)	23.6	24.0	19.3	30.0

Table 4: PL Example Design Delays in *pl4gate* delays

For the multiplier, the value in '()' is the single compute cycle delay, which is the delay divided by 17 since it takes 17 clocks to complete a multiplication. The *MinDly* column gives the straight-line delay path through the PL design and represents the best case where all carry gates do an early evaluation. The value of 3.0 for the Accumulator represents the InputDFF + Sum + Mux LUT4s. The MUX LUT4 is used for the synchronous clear, the output DFFs were absorbed into this mux. The value of 5.2 for the multiplier represents the InputDFF + 3 Mux LUT4s + SUM + a small amount of additional delay from feedback signals. The 3 MUX LUT4s in the multiplier data path were used for clearing, shifting functions. The value of 19.3 for the multiplier represents Input DFF + 16 LUT4 delays for the CSA Array + Sum (carry/sum merge) + Sum of final adder + small amount of feedback delay.

The expected average delay of designs is also shown in Table 4. This value is computed by using the expected average delay of a ripple adder with early eval as (log(N) + 1)[19] added to the other path delays in the design). The accumulator is approximately 29% less than the expected value.

To see if the better than expected performance was possibly be due to favorable data patterns, the accumulator example was re-simulated with 10K vectors, both sampled sinewaves (10 cycles) and random. The average delay results were the same to one decimal place (5.7) as the previous simulation. Figure 6 shows the delay distributions of this simulation. We believe that the better than expected average performance is due to the bit-level pipelining feature of PL.

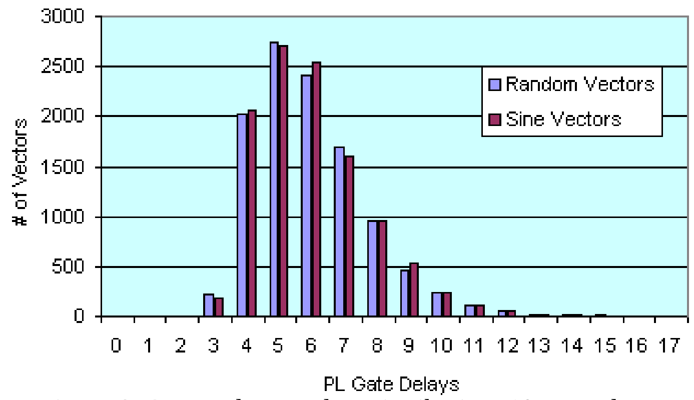

Figure 6: Accumulator Delay Distribution, 10K Random and Sine Vectors

The expected average delay calculation of the iterative multiplier did not take into account the speedup obtained via forcing the adder to early evaluate for each multiplier '0' bit. This accounts for the measured performance being better than the expected performance.

The expected average delay of the filter was: 1 (InputDFF) + 16

(CSA Array) + 5 (16-bit add for carry/sum merge) + 1 (Carry) + 1 (Sum) of final adder = 24 delays. Due to bit-level dataflow, two adders in series do not add as 2*(log(n) + 1). This is because the carry chain in the 2nd adder can start firing as soon any bit from the 1st adder arrives, so the carry chains essentially compute in parallel with only a constant delay for each additional adder in series.

It can be seen from the maximum delay column that no vectors exercised the worst case delay of N delays in the adder plus straight line delays in the remainder of the datapath.

C. Cell Count, Wire Count Comparisons

No physical cell layouts of the *pl4gate* were done, so area comparisons of clocked designs versus PL designs can only be done on a cell count and wire count basis. It should be noted that assuming the LUT4 dominates the active cell area, the complexity of the *pl4gate* cell is about the same as an Altera or Xilinx LUT4 logic cell.

Design	LUT4s	DFFs	Sgates	Cgates	TotCells
Clk (acc)	179	65	0	0	179
PL (acc)	126	33	1	42	160
Clk (mult)	317	71	0		317
PL (mult)	150	50	103	121	303
Clk (filt)	697	80			697
PL (filt)	621	48	32	575	701

Table 5: Cell Counts for Example Designs

Table 5 gives the cell count comparisons for the three example designs. The clocked designs only have LUT4s and DFFs. The *TotCells* column for the clocked designs is only the LUT4s since it is assumed that the DFFs are integrated into the logic cells as in Xilinx and Altera designs. The PL designs have LUT4s, DFFs, Sgates, and Cgates. The PL DFFs are actually any cells in the PL netlist that act as a buffer function. It is assumed that these cells will bypass the LUT4 in the cell and that only the control section of the *pl4gate* is active.

The mapping program attempts to absorb any DFFs present in the clocked netlist into its corresponding driving cell in the PL netlist. Input DFFs are not absorbed. The DFF absorption is why the PL designs have less 'DFFs' than the clocked designs. However, these DFFs count against the PL cell count since the control circuitry functionality is needed. The *Sgates* column is for *splitter* gates that are added by the mapping program. For feedback generation to work correctly, the mapping program inserts a buffer on any path in the original clocked netlist that has a DFF output driving a DFF input directly. These splitter gates are simply buffer functions, but also count against the cell count since the control circuitry is needed. The Cgates column is the number of 4-input Muller C-elements added to the netlist by the mapping program for feedback concentration. We assume that each *pl4gate* has an unassigned 4-input Cgate to be used for feedback concentration so these gates do not count towards the PL cell count. Thus, the PL cell count is the sum of LUT4s, DFFs, and Sgates while the clocked cell count is the LUT4 count only. It should be noted that early evaluation gates such as used in the PL ripple adders require two LUT4s and count as two cells.

The iterative multiplier has a high number of splitter gates due to the shift register functions in the design. All designs are able to absorb some of their DFFs into neighboring cells. The combination of DFFs counting against PL cell counts and splitter gates can lead to PL requiring more cells as in the filter design. PL will only use less gates in the case where it can use a structure like an early eval ripple adder that saves cells over a carry lookahead adder.

Table 6 shows the wire count comparisons for the three designs. The signal net column is the number of signal nets in the original netlist, and for clocked designs is also the number of total wires (ignoring the clock net and reset net). The FBnets column is the number of feedback nets added to the PL design by the mapping program. The total nets column for the PL design is the sum of the signal nets and feedback nets. The number of total wires in the PL design is the signal nets multiplied by 2 because of the dual rail signaling, plus the feedback nets (feedback signals are single rail).

Design	Signal Nets	FBnets	Tot Nets	Tot Wires
Clk (acc)	194	0	194	194
PL (acc)	160	171	331	491
Clk (mult)	252	0	252	252
PL (mult)	303	366	669	972
Clk (filt)	704	0	704	704
PL (filt)	701	1230	1931	2632

Table 6: Wire Count Comparisons for Example Designs

Feedback wiring is probably the largest challenge for efficient implementation of PL-based programmable logic. Feedback nets represent a new routing resource that must be dealt with. The number of feedback nets are high in all three of the example designs for several reasons: a) a small amount of logic between DFFs requires a lot of feedback nets since a feedback net cannot cover a long path (as in the accumulator example); b) large fanout will require large feedback concentration at a node which will require building trees of C-elements that will be expensive in terms of feedback wiring required. The last case of feedback wiring due to trees of Muller C-elements can be somewhat reduced by including a larger C-element for feedback concentration in each cell (e.g., an 8-input element).

VI. Comparisons to Other work

A self-timed FPGA based upon LUT3s and using LEDR encoding was presented in [8]. The cell design presented in Figure 2 is a variation of the cell design used in [8]. In [8], three feedback inputs are included in each cell, so the Celement has 6 inputs (3 data, 3 acknowledge). The author uses the cell in the context of Sutherland's micropipelines [15] and self-timed iterative rings [16]. Both methods require a feedback signal for each output destination. The PL methodology removes the need for a feedback for every output signal destination as multiple output signals can be covered by the same feedback signal, and some output signals need no feedback signal if they are already part of a loop. No performance or power analyses are made of designs using the LUT3 cell.

An FPGA-based architecture for asynchronous logic is also proposed in [17]. This FPGA architecture was aimed at accommodating a range of asynchronous design styles, and allowed for mixed synchronous and asynchronous designs. All signals were single rail. By contrast, our proposed function block is only intended for supporting the PL design style, and thus implements PL designs more efficiently than [17].

The asynchronous design methodology known as Null Convention Logic (NCL) also offers automated synthesis of asynchronous designs using commercial synthesis tools [18]. A restriction with the NCL methodology is that while the design can be coded in VHDL RTL, the user must write the RTL such that combinational logic and registers are separated. In comparing physical implementation characteristics, NCL has some delay sensitivity between NCL gates whereas PL has no delay sensitivity between PL gates. Both NCL and PL use dual rail signals, where NCL uses a NULL/DATA/NULL encoding instead of LEDR. NCL has the same advantage of eliminating transient computations as PL, and does not have the disadvantage of the PL control overhead. The computation blocks in PL are the same as their synchronous counterparts with only a different control scheme, while NCL computation blocks are quite different from their synchronous equivalents. NCL designs have been shown to be well suited for standard cell implementation technologies. NCL mapping to a LUT4-based technology would require a conservative estimate of 2X the number of LUT4 gates as the clocked netlist due to the dual rail encoding of the data values and the NCL implementation using *m*-of-*n* threshold gates. A thorough investigation is needed to determine if the extra compute switched capacitance required by an NCL LUT4 implementation will offset the control switched capacitance in a PL design.

There are many examples of self-timed adder structures in the literature. A good summary and analysis of these approaches can be found in [19]. Two interesting conclusions are reached in [19], a) asynchronous adders can give performance improvement over Conditional Sum Adders in only limited conditions, and b) large variations in processing time of an addition limits the speed of an asynchronous pipeline. It is somewhat difficult to compare our results against these conclusions (especially 'a') since we used LUT4s for implementation, while [19] used primitive two input gates, and we used a more conventional CLA structure for our synchronous adders instead of a CSA structure. However, some general comments can be made. The conclusions in [19] are all based on asynchronous adders for ASIC technologies in which a completion signal is generated for the entire adder result. Our feedback signals for the LUT4 *pl4gate* work at the bit level, so this allows more parallelism via bit level dataflow. Our iterative multiplier design can be regarded as a 'deep' asynchronous pipeline since it takes 16 clocks to produce one result, where each addition is dependent on the previous result forming a long dependency chain. If some set of bits within an addition is slow in producing a result because of a non-favorable early evaluation pattern, this does not inhibit the other bits from proceeding in the calculation due to the bit-level dataflow nature of Phased Logic. If the coarseness of the compute block needs to be increased due to compute/control capacitance concerns, the coarseness can still be limited to something less than a word level (eg., groupings of 4 bits) in order to promote more parallel dataflow.

A speculative-completion carry-lookahead adder is presented in [20] . This technique is well suited for an ASIC implementation but its reliance on matched delays and a bundled datapath makes an FPGA LUT4 implementation questionable. Applying general speculative completion techniques within a PL system is an area that needs further study.

VII. SUMMARY

In this paper, we have shown the application of a self-timed logic style called *Phased Logic* to three example arithmetic circuits. The circuits were simulated using a LUT4-based gate (*pl4gate*) that implemented the Phased Logic protocol. All three circuits were more energy efficient than their clocked counterparts due to transient computation filtering and increased performance. Two of the three designs had higher performance than the clocked designs despite a 40% gate level penalty performance of the LUT4-based *pl4gate* cell. The increased performance of the PL netlists was due principally to the use of early evaluation in the adder circuits. An early evaluation PL gate is formed by using a pair of normal PL gates, where one gate provides the early evaluation function and one gate provides the normal evaluation function.

We believe a natural application of PL gates is a LUT-based FPGA technology. The high ratio of compute to control capacitance allows PL netlists to be competitive with clocked netlists in terms of total switched capacitance. Areas of future work involve looking at compute blocks suitable for programmable logic other than a LUT4 and examining the same performance/energy comparisons on larger design examples that have multiple functional units and complex control.

REFERENCES

[1] Daniel H. Linder and James C. Harden, “Phased Logic: Supporting the Synchronous Design Paradigm with Delay-insensitive Circuitry.” *IEEE Transactions on Computers*, Vol 45, No 9, September 1996.

[2] Daniel H. Linder, *Phased Logic: A Design Methodology for Delay-Insensitive Synchronous Circuitry*, PhD thesis, Mississippi State Univ., 1994.

[3] R. Reese, and C. Traver, "Synthesis and Simulation of Phased Logic Systems", Technical Report MSSU-COE-ERC-00-09, MSU/NSF Engineering Research Center, June 2000. Presented at International Workshop on Logic Synthesis (IWLS 2000), Dana Point, CA, June 2, 2000.

[4] D. B. Armstrong, A.D.Friedman, and P.R.Menon, “Design of Asynchronous Circuits Assuming Unbounded Gate Delays,” *IEEE Transactions of Computers*, vol. 18, December 1969.

[5] A.J. McAuley, “Four State Asynchronous Architectures,” *IEEE Transactions on Computers*, vol. 41, February 1992.

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

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

[8] Dana L. How, “A Self Clocked FPGA for General Purpose Logic Emulation”, in proceedings of *IEEE 1996 Custom Integrated Circuits Conference*, 1996, pp. 148-151.

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

[10] Tzyh-Yung Wuu and Sarma B. K. Vrudhula, "A Design of a Fast and Area Efficient Mult-Input Muller C-element", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol 1, No. 2, June 1993.

[11] http://www.ece.msstate.edu/~reese/phased_logic FLASH Animations.

[12] Altera Apex Power Estimator, http://www.altera.com/html/products/power_calc.html

[13] Xilinx Virtex Power Estimator, http://www.xilinx.com/support/techsup/powerest/index.htm

[14] Rabaey, Jan M., *Digital Integrated Circuits: A Design Perspective*, Prentice Hall, pp 408-412.

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

[16] M.R. Greenstreet, T.E. Williams, and J . Staunstrup, "Self-Timed Iteration", *VLSI '87*, C. H. Sequin (Ed.), Elsevier Science Publishers, 1988, pp. 309-322.

[17] Scott Hauck, Steven Burns, Gaetano Borriello, Carl Ebeling, “An FPGA for Implementing Asynchronous Circuits”, *IEEE Design and Test of Computers*, Fall 1994, pp. 60-69.

[18] Michiel Ligthart, Karl Fant, Ross Smith, Alexander Taubin, Alex Kondratyev, "Asynchronous Design Using Commercial HDL Synthesis Tools", In *Sixth Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async 2000)*, Eilat, Israel, April 2000.

[19] D. J. Kinniment, "An Evaluation of Asynchronous Addition", IEEE Trans. On Very Large Scale Integration (VLSI) Systems, Vol. 4, No 1., March 1996.

[20] S. M. Nowick, "Design of a low-latency asynchronous adder using speculative completion", II Proc. Comput. Digit. The., Vol 143, No 5, Septembert 1996, pp. 301-307.