

Power Considerations in Phased Logic Gate Design

MSU Tech Report MSSU-COE-ERC-01-03

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

Cherrice Traver
 Union College
traverc@union.edu

Abstract

Phased Logic (PL) is a synthesis and mapping methodology that translates a standard synchronous gate-level netlist to a non-clocked netlist of special gates called phased logic gates [1][2][3]. In this paper the implementation of PL gates and the switched capacitance of phased logic systems is discussed. An analysis of the dynamic power consumption of a phased logic system reveals a major deficiency in the original programmable LUT-based design [2], leading to an alternate phased logic gate design. Switched capacitance comparisons of designs using the new phased logic gate versus clocked FPGA implementations provide insights to PL gate design requirements and the type of applications best suited for phased logic implementation.

1. Introduction

Phased Logic (PL) systems use Level Encoded two-phase Dual-Rail (LEDR) encoding of signals [4] [5] and phased logic gates[1]. Circuitry using the LEDR signaling method has been developed by McAuley [5], Dean, Williams and Dill [6], How [7] and Linder [2]. Linder's work also includes a synthesis and mapping methodology that allows a designer to specify a clocked system that can be translated to a PL system. This approach takes advantage of the well-established synchronous design methodology and results in systems that are insensitive to delays between PL gates. The implementation of this methodology has recently been enhanced so that synchronous designs are synthesized by SynopsysTM to EDIF netlists and then translated to a VHDL netlist of PL gates. Designs ranging in size from a few hundred to few thousand gates have been synthesized, mapped and simulated [3]. Simulations have verified that the optimizations and architectural approaches used to improve performance in synchronous designs also improve the performance of the corresponding phased logic implementation.

To move toward the practical application of phased logic systems, an efficient and reliable implementation of phased logic gates must be developed. Linder demonstrated the feasibility of a programmable phased logic gate, implemented in a 2u CMOS technology (1994), that would lend itself to a FPGA architecture. The coarse-grain logic implementation of FPGA look-up tables reduces the impact of the logic overhead necessary to implement phased logic gates. The FPGA approach for phase logic system implementation offers the advantage of rapid-prototyping along with a delay-insensitive implementation. Reconfigurable systems would especially benefit from this environment due to the lack of concern about timing changes with different configurations.

This paper is organized as follows. Section 2 reviews phased logic theory, a previous implementation of a phased logic gate, and the current methodology for mapping a clocked netlist to a phased logic netlist. Section 3 discusses performance and power consumption issues of phased logic systems. Section 4 presents an alternate control scheme for the programmable phased logic gate. Section 5 compares estimated switched capacitance for clocked and phased logic implementations of example systems, while Section 6 does the same for PL versus Null Convention Logic. Section 7 summarizes the progress and findings reported in the paper.

2. An Overview of Phased Logic

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 LEDR encoding [4]. A phased logic gate also has an even or odd phase; a phased logic gate *fires* whenever all of the phases of the inputs matches the internal gate phase.

Figure 1 illustrates the encoding of the dual rail signals used between PL (Phased Logic) gates. A sample gate firing is also shown. The controlled firing of PL gates is what gives a phased logic system its delay insensitivity to wiring delays between PL gates. 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.

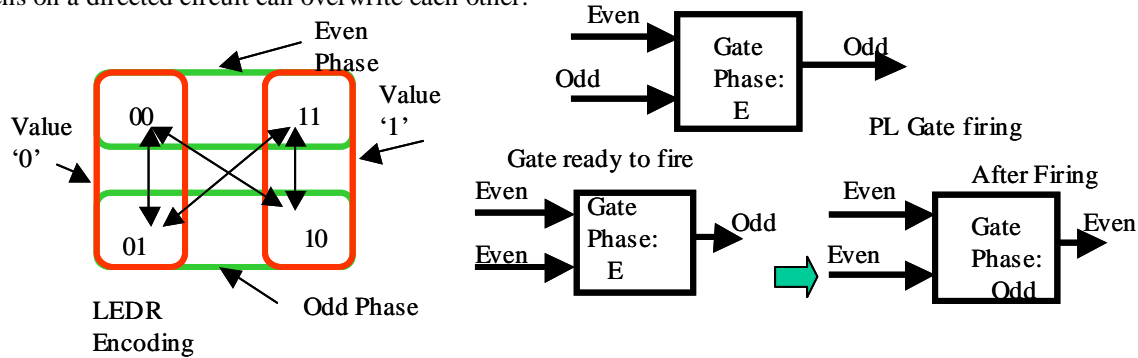


Figure 1: Signal Encoding, Gate Firing

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. As such, a phased logic system is not a true asynchronous system that only has activity when data values change. This has implications about power consumption that will be discussed later.

The mapping of a clocked netlist (DFFs+ 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, just phase information and are single wires. The details of feedback generation are described in [1][2][3]. Feedback wiring typically adds about 15% to 25% additional wiring over the dual-rail routing already required. Muller C-elements [8][9] are used to concentrate feedbacks at a particular gate when needed.

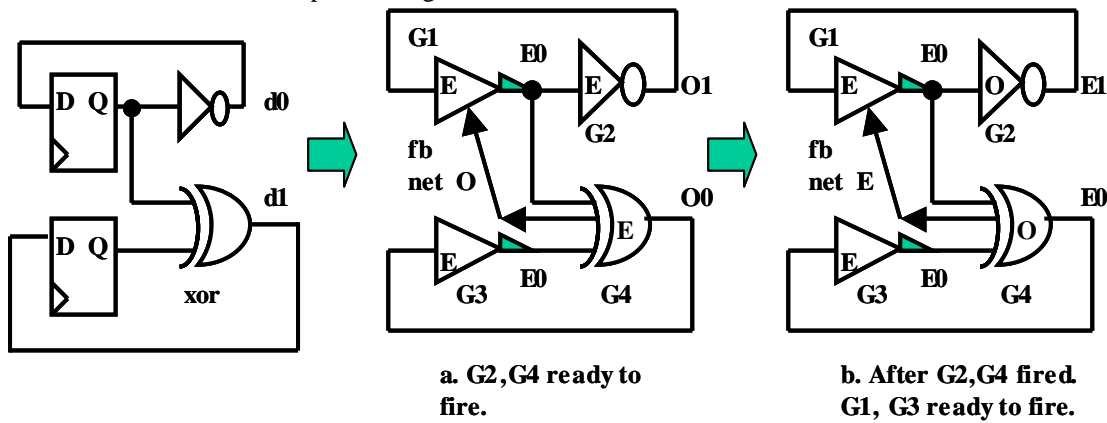


Figure 2: Clocked Netlist to Phased Logic Netlist

Figure 2 gives an example translation from a clocked netlist (a 2 bit counter) to a phased logic netlist. The DFFs are replaced by buffers in this case, even though most of the time they can be absorbed into neighboring gates. The 'wedge' placed on the output of gates G1, G2 in the PL netlist indicates that the net is connected to the inverted phase output of the gate in order to ensure that a gate is ready to fire in the loops formed by gates G1-G2, G1-G4, and G3-G4. The signal from the output of G1 to the input of gate G4 is *unsafe*, i.e., it is not part of a loop that includes gate G1. In this case, an extra feedback net has to be added from gate G4 to G1. After reset, all gate phase bits are reset to *even* and the signal phases are as shown. In diagram (a), Gates G2, G4 are ready to fire since all input phases match the gate phase. Diagram(b) shows the situation after gates G1, G3 have fired; gates G2 and G4 are now ready to fire. Firing will continue in a cyclic nature and a phased logic netlist will operate at its maximum speed. The firing of a phased logic system can be halted by preventing the phase firing of any input.

2.1 Original Programmable Phased Logic Gate – PL-LUT4

A 4-input programmable phased logic gate implemented in CMOS is presented in [1]. The gate is implemented using a look-up-table approach for the logic and is denoted PL-LUT4. A block diagram of Linder's PL-LUT4 is shown in Figure 3. Inputs a , b , c , and d are LEDR input signals and y is the LEDR output signal. The feedback output is fo , and it is a single wire. Inverted versions of both value and phase wires of y are available as y' . Programming inputs include 4 data inputs, d , and 8 write inputs, w . Note that some of these signal names are modified from [2].

The LEDR-encoded input signals are decoded via row and column decoders. The internal gate phase is stored on two non-overlapping precharge signals, pe and po , generated in the control logic. When the gate phase is odd, po is high (accepting odd inputs), and pe is low (precharging). Once all odd inputs arrive, the appropriate ro_i and co_i signals become high. This causes a read of the appropriate RAM cell, and either o or o_n becomes active, depending on the value read. The arrival of the o or o_n signal causes the gate to fire. The output value is latched, pe is set high by the control block, and this set of events is repeated for the even phase.

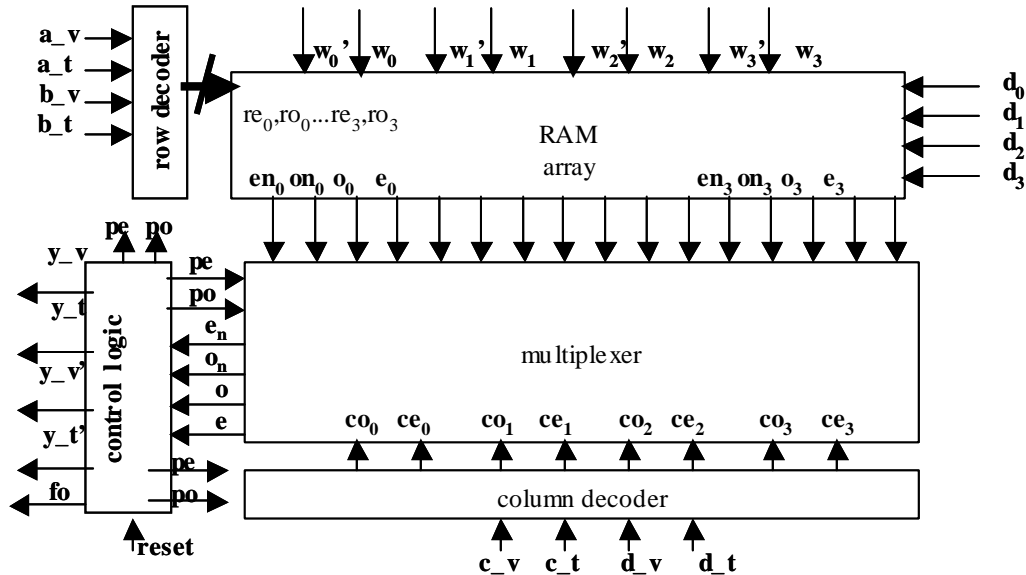


Figure 3. Internal PL-LUT4 Architecture

An important aspect of this gate is that the RAM array is read anytime the input phases match, even if the data values have not changed. We will discuss the impact of discussion on power consumption in a phased logic system in Section 3. This device was simulated in Hspice using Leda 0.25u technology and was found to have 1.8 pF of switched capacitance for any phase or value change on its inputs.

2.2 Mapping and Simulation of Phased Logic Netlists

In Linder's original work, no actual simulation of phased logic systems were done and the original mapping algorithm was implemented in C++ under WinNT. Recently [3], the mapping algorithm was re-implemented in C in

a UNIX environment (SUNOS) with some improvements that allow tradeoffs between CPU time and the number of feedbacks generated in the netlist. The mapping tool reads an EDIF netlist specifying a clocked system and outputs a VHDL netlist of phased logic gates. The resulting phased logic netlist is simulated using the ModelTech simulation environment. Figure 4 shows the methodology used to produce the netlists used in this paper.

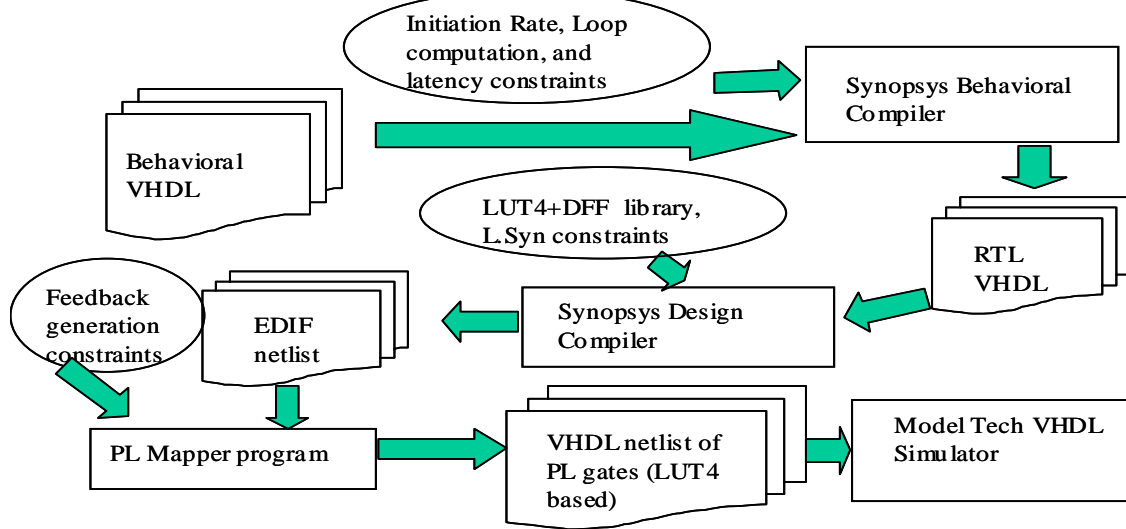


Figure 4: Design Methodology

3. Dynamic Power in a PL System

To evaluate the dynamic power consumption of a *phased logic system*, we must track the amount of *control* and *compute* capacitance switched during a particular computation [10]. In the original clocked system, any combinational logic ‘computations’ that occur within any DFF to DFF path during a clock period will also occur in the phase logic system. This means that the phased logic system will switch the same amount of compute capacitance as the synchronous system since the netlist mapping is a one-to-one mapping for combinational gates (this neglects the effect of transient firings in the synchronous systems; this will be discussed later). The period of this computation in the phase logic system is the delay of the compute sections along the path that corresponds to the longest DFF-to-DFF path in the clocked system. Each clock cycle in the clocked system is now a computation cycle in the phased logic system.

The control capacitance in the clocked system will be the amount of capacitance switched in the DFFs plus the clock tree routing/buffering capacitance. In the phased logic system, *all gates* will change phase exactly once during a computation, even if the compute section does not switch because of quiescent value wires.

The comparison of capacitance switched during a clock cycle in a clocked system versus the equivalent phased logic system is then:

$$C_{\text{switched}}(\text{clock}) = \#\text{ComputeFirings} * C_{\text{compute}} + \#\text{DFFs} * C_{\text{dff}} + C_{\text{clock_tree}}$$

$$C_{\text{switched}}(\text{PL}) = \#\text{ComputeFirings} * C_{\text{compute}} + \#\text{PLgates} * C_{\text{PLcontrol}} + C_{\text{cgate}}$$

C_{compute} is the capacitance of a combinational gate (assumed all equal for now); $C_{\text{PLcontrol}}$ is the capacitance of a PL gate that is switched during a phase change, and C_{cgate} is the switched capacitance of the Muller-C elements used for feedback concentration. For a PL system to switch the same or less capacitance during a computation than the clocked system, the following should be true:

$$\#\text{PLgates} * C_{\text{PLcontrol}} + C_{\text{cgate}} \leq \#\text{DFFs} * C_{\text{dff}} + C_{\text{clock_tree}}$$

It is useful to repeat the key assumptions of the above equations at this point:

1. The $\#\text{ComputeFirings}$ in the phased logic system is the same as in the clocked system (to be examined in detail later)
2. The Compute switching of a phase logic gate is tied only to value bit changes, not to phase changes.

Assumption #2 is only important for reduced power consumption and not for functionality. Unfortunately, Linder's PL-LUT4 implementation accessed the RAM array on every phase change. This meant that the capacitance switched during a computation of a PL system using Linder's PL-LUT4 would be:

$$C_{\text{switched}}(\text{PL}) = \#\text{Plgates} * (C_{\text{compute}} + C_{\text{Plcontrol}})$$

This would obviously consume more power than the clocked system.

4. A PL Gate with Decoupled Control/Computation

The control for a PL gate that decouples compute switching activity from phase change activity for power-savings purposes is shown in Figure 5. This is actually a variation of a design presented in [7] (differences discussed in Section 6).

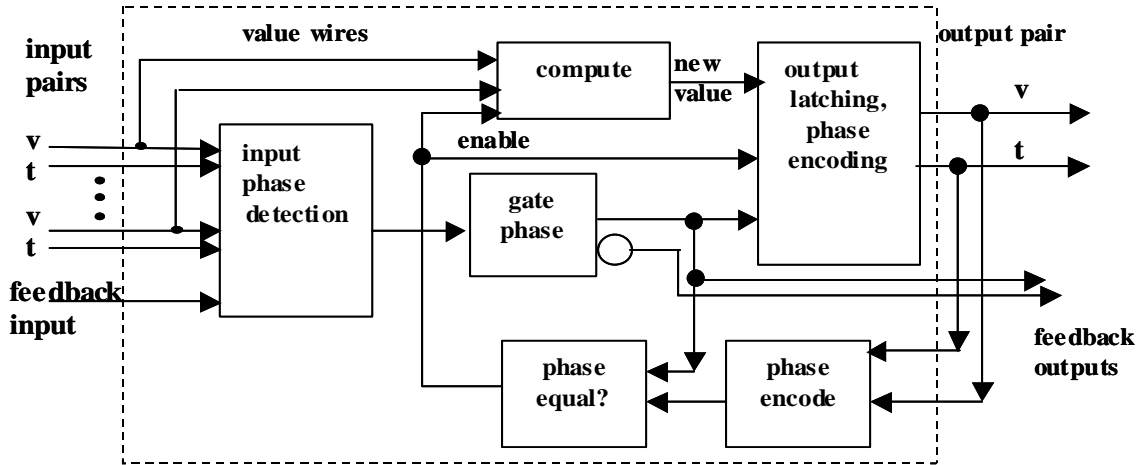


Figure 5. Block diagram Decoupled Compute/Control PL gate.

The *input phase detection* block fires whenever all input phases match and this causes a toggle of the *gate-phase* latch. The change in gate phase will assert *enable* because the output phase and internal phase will match. The asserted *enable* latches a new output value from the *compute* block, which generates a new *t, v* output value that will be opposite in phase to the internal gate phase, negating the *enable* signal. The *enable* can also be used to protect the compute block from changes on the value input signals until all input phases have matched. An implementation of the control is shown in figure 6.

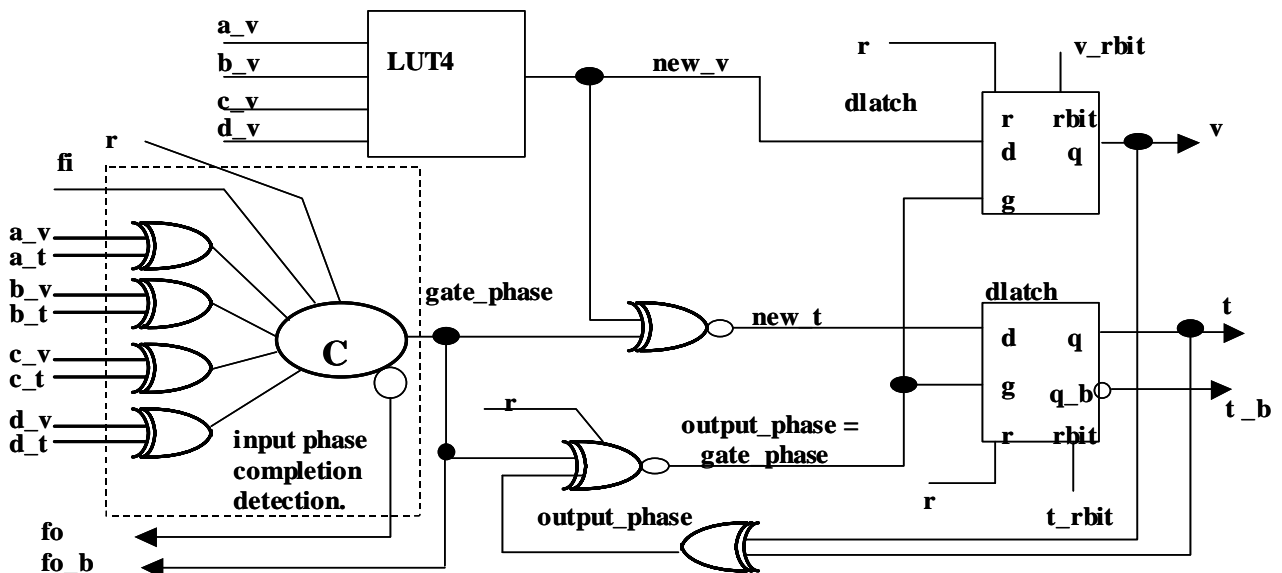


Figure 6. Logic diagram for improved PL gate (PL4GATE)

The v_rbit , t_rbit signals are used in conjunction with reset r to provide initial output values. During reset, gate phase becomes *even*, represented by a logic value of 0. The output phase must therefore be *odd*. The t_rbit value will be '1' for odd phase encoding, and v_rbit with either be '0' or '1' depending upon if the corresponding DFF in the clocked netlist was cleared or set during reset. The combination of t , v provides an output of opposite phase from gate phase; t_b , v provides an output of the same phase as the gate phase (required for token generation on circuit loops). The gate signal for the output latches do not affect the compute element (a standard LUT4 element). This means that the LUT4 will respond to any changes on the value inputs, but the new_v output signal will not be latched until all input phases match. To reduce transient firings of the LUT4, the $Dlatch$ gate signal could also be used as an enable signal for the LUT4. However, this would impact the speed of the design as it would place the *input phase completion detection* delay in series with the LUT4 delay. Depending on the relative magnitudes of these two delays, it might prove beneficial to provide a configuration bit that would choose between the two configurations with cells off the critical path configured for an enabled-LUT4 operation in order to reduce transient LUT4 firings. Obviously the internal operation of this gate is not delay insensitive, and internal delay paths will have to be adjusted to ensure a single transition on either the v or t outputs.

A transistor level implementation of the above control logic (minus the LUT4) was implemented and simulated in HSPICE using a Leda 0.25u technology file at 2.5V with a 4x inverter load. The control logic was found to have 0.200 pF of switched capacitance for a phase change. The 89% decrease in switched capacitance (0.200 pF vs 1.8 pF) over Linder's PL-LUT4 implementation is primarily due to the decoupling of compute from control functions within the new PL gate. A 4-input Muller C-element used for feedback concentration was found to have 0.075 pF switched capacitance. These C-elements are sprinkled throughout a PL netlist based upon the number of feedback signals required to maintain liveness and safety.

5. Switched Capacitance Estimation: PL vs Clocked for FPGA Applications

In this section we will estimate the capacitance switched per computation for both PL and clocked systems using an FPGA implementation technology. FPGA vendors have begun providing power estimation spreadsheets [11][12] for their FPGA families. The Xilinx Virtex and Altera Apex FPGAs use LUT4s as their basic compute element, and the power estimation spreadsheets can be used to calculate capacitance estimates for LUT4s, DFFs, and clock tree routing. Table 3 shows the capacitance estimations of LUT4s/DFFs in both Altera and Xilinx FPGA implementations.

	LUT4	DFF	Clock Tree (DIE)	Clock Tree/LUT4
Xilinx Vertex	1.2 pF	0.21 pF	Not available	
Altera Apex	0.900 pF	0.14 pF	825 pF (EP20K400)	34 pF (EP20K400)

Table 3. Switching capacitance for Xilinx Vertex, Altera Apex FPGAs, 0.25 μ

The Xilinx Virtex spreadsheet did not provide a clock tree power figure for a particular die; instead it amortized the clock tree cost over the DFFs.

To compare the switched capacitance of a PL system versus a clocked system in an FPGA implementation, two designs specified in behavioral VHDL were mapped to a LUT4+DFF implementation using Synopsys Behavioral Compiler + Synopsys Design Compiler. Two variations (low performance, high performance) of an FIR filter and two variations (low performance, high performance) of a $[4 \times 4] * [4 \times 1] = [4 \times 1]$ Matrix multiplication engine were done. Table 4 summarizes the characteristics of the designs. Both the FIR and MATRIX designs have a startup period for coefficient loading (FIR loads the tap coefficients, MATRIX loads the $[4 \times 4]$ matrix), but then enter a continuous compute loop.

	EXE Units	States	LUT4	DFF	LS Const.	Compute Rate
FIR (lp)	1 mult (4x4=8), 1adder (8 bit)	32	573	178	Min area	12 clocks/result
FIR (hp)	8 mults (4x4=8), 7adders (8 bit)	23	1294	248	Min delay	1 clk/result, 5 clk latency
Matrix (lp)	1 mult (6x6=12), 1adder (12 bit)	56	1115	335	Min area	21 clks/[4x1] result
Matrix (hp)	4 mult (6x6=12), 3 adders (12 bit)	46	1255	325	Min delay	4 clks per [4x1] result

Table 4. Design Summary

Using our PL netlist conversion tool [3], equivalent PL designs were created. The PL netlist replaces all LUT4/DFFs in the original clocked netlist with the PL gate shown in Figure 5, adds feedback nets to maintain safety/liveness, and adds Muller C-elements for feedback concentration when necessary. The PL netlist has the same number of LUT4s with the same Boolean functions as in the clocked netlist. Both sets of netlists, clocked and PL, were simulated in the ModelTech VHDL simulator and verified to produce the same results.

To estimate switched capacitance, the VHDL models of the LUT4, DFF used in the clocked netlist simulation and the PL4GATE used in the PL netlists were instrumented to track the number of LUT4 firings and control firings. In the clocked netlist, a control firing was defined as a rising clock edge arriving at a DFF while a phase change within a PL4GATE was a control firing in the PL netlist. These firings were totaled over the period of time it took to output 100 values on the result bus for each design, and several sample periods were averaged together to produce average compute firings and average control firings for each design. Table 5 gives the switching statistics produced by the VHDL netlist simulations (CL denotes the clocked netlist, PL denotes the phased logic netlist), . Two sets of numbers are given for PL netlists; the 'A' design has the LUT affected by the value inputs only after all input phases match while the 'B' has the LUT affected anytime a 'v' input changes.

The most interesting data in Table 5 is the reduction in LUT4 firings by the PL netlists in 7 out of the 8 designs. The reduced firings are due to filtering of transient firings, where the transient firings occur because of mismatched path lengths. All LUT4s in simulations were assigned the same delay, so the firings indicate arrival of new 'value' signals at a spacing of at least 1 LUT delay. The LUTs of the 'A' PL designs do not fire until all input phases have matched; this removes ALL transient LUT4 firings. The LUTs of the 'B' PL designs fire anytime a value wire changes. This shows that our assumption in Section 3 that compute firings in clocked and phased logic netlists are equal was flawed. The correct assumption is that if the PL compute element does not fire until all input phases match, then the number of compute firings in a PL system will be less than or equal to the compute firings in the clocked system.

Design	LUT Firings	%diff	Ctrl Fire	#of Cgate4s	Cgate Firings
CL fir lp	333170		213600		
PL fir lp(B)	211233	-37%	687600	111	117600
PL fir lp(A)	163070	-51%	687600	111	117600
CL fir hp	48907		24800		
CL fir hp(B)	50127	2%	129400	268	25100
CL fir hp(A)	41223	-16%	129400	268	25100
CL Mat lp	368498		175875		
PL Mat lp(B)	224249	-39%	585375	207	75600
PL Mat lp (A)	171905	-53%	585375	207	75600
CL Mat hp	187972		32500		
PL Mat hp(B)	117153	-38%	125500	619	35200
PL Mat hp(A)	83276	-56%	125500	619	35200

Table 5. Switching Statistics

Table 6 gives the switched capacitance numbers using 1.05 pF for a LUT4 (average of Altera and Xilinx), 0.14 pF for a DFF (Altera), 0.035pF for a clock tree cost per LUT (Altera), 0.20 pF for PL4gate phase change, and 0.075 pF for a Cgate firing. All capacitance numbers are based upon a 0.25u technology. The table shows reduced total switched capacitance for the PL designs over the clocked designs in 7 out of 8 cases.

Design	Lut Cap	%cap	Control Cap	%cap	Cgate Cap	%cap	ClkTreeCap	%cap	Total Cap	%diff
CL fir lp	349829	87%	29904	7%	0		24066	6%	403799	
PL fir lp(B)	221795	60%	137520	38%	8820	2%			368135	-9%
PL fir lp(A)	171224	54%	137520	43%	8820	3%			317564	-21%
CL fir hp	51352	87%	3472	5%	0		4529	8%	59353	
PL fir hp(B)	52633	65%	25880	33%	1883	2%			80396	35%
PL fir hp(A)	43284	61%	25880	36%	1883	3%			71047	-12%
CL Mat lp	386923	90%	24623	5%	0		20488	5%	432034	
PL Mat lp(B)	235461	66%	117075	34%	5670	2%			358206	-17%
PL Mat lp (A)	180500	60%	117075	38%	5670	2%			303245	-30%
CL Mat hp	197371	96%	4550	2%	0		4393	2%	206313	
PL Mat hp(B)	123011	82%	25100	18%	2640	2%			150751	-27%
PL Mat hp(A)	87440	76%	25100	22%	2640	2%			115180	-44%

Table 6. Switched Capacitance Comparison PL vs Clocked

Some observations based upon the data in Table 6 are:

- The reduced firing transitions by the PL netlists are key in reducing power over the clocked netlists. In the case where the PL transitions were actually a bit higher (high performance FIR) than the clocked system (2% diff), the higher cost of PL control logic results in the 35% increase in switched capacitance for the PL system. The high performance FIR netlist represented a netlist with very streamlined control and one that was dominated by datapath components.
- It is vital that the relative capacitance of the compute block (LUT4) to the control capacitance in the PL4gate be as large as possible. Because control logic is included in every PL4gate, this must be amortized over as large a compute function as possible. For example, if the compute block of the PL4gate had approximately the same capacitance as the control element, this would change the best case capacitance %diff from -44% to -5%, and the worst case capacitance %diff from +35% to +110%, and only 2 out of 8 PL netlists would have less switched capacitance than the clocked netlists.
- The Cgate capacitance in the PL netlists was not a significant percentage of the total system switched capacitance.
- PL netlists are not as efficient as clocked netlists in terms of switched capacitance for control purposes. In every case, the PL netlists switched anywhere from 4x to 8x more control capacitance than the clocked netlists. However, this was offset in 7 out of 8 cases by the reduction in switched capacitance for computation.
- The pro-rating of the clock tree power among LUT4s tends to diminish the true impact of the clock tree distribution network on power consumption in a clocked system. Altera's power calculator for a 600E device (24,320 LUTs) shows that the clock tree would account for 10% of the power dynamic power consumption assuming 95% LUT utilization (12% toggle rate) and 10% DFF utilization. Another example is the Alpha 21264 of which 36% of its 72W power consumption at 600 Mhz were due to the global and local clock network [10]. This means that the data in Table 6 could be skewed even more in favor of PL netlists if the clock tree contribution was increased.

6. Comparisons to Other Work

A self-timed FPGA based upon LUT3s and using LEDR encoding was presented in [7]. The cell design presented in Figures 5, 6 is a variation of the cell design used in [7]. In [7], 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 [13] and self-timed iterative rings [14]. 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. The 4-input C-elements assumed available as part of the clocked to PL netlist translation can be made available as separately routable, independent elements. A 4-input C-element can be included as part of

each PL4GATE (Figure 6) which should provide enough 4-input C-elements for any clocked to PL netlist translation. If a feedback input (\bar{f}_i) on a PL4GATE is not needed, it can be tied to the negation of the *gate_phase* signal. In [7], an area comparison is made between clocked and self-timed networks using the LUT3-based cell. An interesting observation is made that routing area is only increased by 50% due to the fact that routing pitch is limited by the SRAM routing configuration cells, thus leaving routing area for the threefold increase in wiring (dual-rail + acknowledge). No analysis is made of the power performance of designs using the LUT3 cell.

An FPGA architecture for asynchronous logic was proposed in [15]. Two types of function blocks were provided; a LUT3+Dlatch block and a block that could implement an arbiter, enabled arbiter, or synchronizer. 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 aimed at only supporting the PL design style, and would implement PL designs more efficiently than [15].

The asynchronous design methodology known as Null Convention Logic (NCL) also offers automated synthesis of asynchronous designs using commercial synthesis tools [16]. 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. Furthermore, the NCL methodology has not been demonstrated to work with an existing high-level synthesis tool such as Synopsys Behavioral Compiler. 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 [17]. NCL designs have been shown to be well suited for standard cell implementation technologies [18]. 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 would be needed to see if the extra compute switched capacitance required by an NCL LUT4 implementation would offset the control switched capacitance in a PL design.

7. Summary

The principle contribution of this paper is the comparison of switched capacitance in equivalent clocked and PL netlists. Based upon this comparison, we state the following design goals for a PL gate:

- Separate compute switching activity from phase switching activity to reduce power consumption.
- Keep the ratio of switched compute capacitance to switched control capacitance as high as possible.
- Protect the compute element from changes on value wires until all input phases have matched in order to remove all transient firings of the compute element.

It seems that the 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. Furthermore, one reason that designers use an FPGA technology over a higher performance ASIC technology is ease of design and quick turnaround. However, large FPGAs are following down the same evolution path for clock networks as high performance ASIC and microprocessor designs. Successive generations of the Alpha microprocessor [19] went from a single clock tree with one driver, to a buffered clock tree, to a PLL-generated buffered global clock tree enhanced by conditioned local clocks skew-matched to arriving data [20]. The complex timing requirements due to global and local clocks lead to increase design time and effort, negating one of the previous advantages of FPGA technologies. A PL-based FPGA would offer the following advantages:

- eliminate design complexity due to clock timing
- offer a scalable technology in terms of LUT count because of the elimination of global wiring (would not have to redesign the clock distribution network for each successive generation and change timing validation methodologies)
- allow designers to use standard high-level/RTL synthesis design methodologies and commercial synthesis tools since a PL netlist is converted from a clocked netlist.
- be competitive with clocked netlists in terms of switched capacitance
- free designers from timing conflicts within the design since a PL netlist is delay insensitive to wiring delay

Handcrafted asynchronous designs [21] and synthesis methodologies [22][23] optimized for asynchronous techniques such as burst mode circuits will almost certainly surpass PL designs in performance, area, and power. However, this is no different from current ASIC implementations being more efficient than FPGA-based implementations in performance, area, and power for clocked designs. The advantage of quick turnaround, flexibility, and ease of design for FPGA implementations result in many design wins over ASIC technologies. PL system design using familiar RTL synthesis methodologies combined with an FPGA implementation technology can fill an important niche within the full spectrum of asynchronous design and implementation methodologies.

Future work in this area involves:

- Investigating alternate control schemes
- Defining the internal PL4gate timing constraints between the compute and control elements,
- Investigating system level issues in PL such as early evaluation.
- Compare performance of PL systems to clocked systems

10. Acknowledgements

The authors would like to thank Dan Linder, Jim Harden, and Mitch Thornton for valuable comments during paper preparation. This work was supported in part by an internal grant through the MSU/NSF Engineering Research Center.

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. Also available for download at http://www.erc.msstate.edu/labs/mpl/projects/phased_logic/yr2000results/newmapping.pdf.
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. 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.
8. 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.
9. Tzyh-Yung Wu 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.
10. A. Chandrakasan and R. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits", *Proceedings of the IEEE*, Vol 83, No. 4, April 1995.
11. Xilinx Virtex Power Estimator, <http://www.xilinx.com/support/techsup/powerest/index.htm>
12. Altera Apex Power Estimator, http://www.altera.com/html/products/power_calc.html
13. I. Sutherland, "Micropipelines", *Communications of the ACM*, Vol 32, No. 6, June 1989, pp. 720-738.
14. 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.
15. Scot Hauck, Steven Burns, Gaetano Borriello, Carl Ebeling, "An FPGA for Implementing Asynchronous Circuits", *IEEE Design and Test of Computers*, Fall 1994, pp. 60-69.
16. 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.
17. Gerald E. Sobelman, Karl Fant, "CMOS Circuit Design of Threshold Gates With Hysteresis", In *1998 IEEE International Symposium on Circuits and Systems*, Monterey, CA, May 1998.
18. Ross Smith, Karl Fant, Dave Parker, Rick Stephani, Ching-Yi Wang, "An Asynchronous 2-D Discrete Cosine Transform Chip", In *Fourth Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async '98)*, San Diego, California, March, 1998.

19. P.E. Gronowski, et al, "High-Performance Microprocessor Design", *IEEE Journal of Solid-State Circuits*, Vol.33, No 5, May 1998.
20. D.W. Bailey and B. J Benschneider, "Clocking Design and Analysis for a 600-Mhz Alpha Microprocessor", *IEEE Journal of Solid-State Circuits*, Vol 33, No. 11, November 1998.
21. K. Yun, P. Beerel, V. Vakilotojar, A. Dooply, J. Accco. "The Design and Verification of a High-Performance Low-Control-Overhead Asynchronous Differential Solver, In *Third Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async'97)*, Eindhoven, The Netherlands, April, 1997.
22. S.M. Nowick and D. L. Dill "Synthesis of Asynchronous State Machines Using a Local Clock", In *Proceedings of the 1991 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, IEEE Computer Society Press, October 1991.
23. K. Y. Yun and D. L. Dill. Automatic Synthesis of 3D Asynchronous Finite-State Machines. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer Aided Design*, IEEE Computer Society Press, November 1992.