

Synthesis and Simulation of Phased Logic Systems

Technical Report MSSU-COE-ERC-00-09

Robert B. Reese(reese@erc.msstate.edu), Mississippi State University
Cherrice Traver (traverc@doc.union.edu), Union College

ABSTRACT: Phased logic is a synthesis/mapping methodology that allows a standard clocked netlist (combinational gates + DFFs) to be automatically mapped to a non-clocked netlist that uses special gates called phased logic gates. The new netlist has no clock networks and the only required global signal is a power-on reset. We demonstrate the viability of the phased logic approach via the synthesis and simulation of several designs ranging from a few hundred to a few thousand gates. Since the input to the phased logic synthesis/mapper tool is a standard clocked netlist, any standard synthesis tool can be used to produce the clocked netlist. Our methodology starts with behavioral VHDL and produces a simulateable phased logic netlist. It is shown that optimizations and architectural approaches that produce a fast clocked design results in a fast phased logic design.

See http://www.erc.msstate.edu/labs/mpl/projects/phased_logic/index.html for more information.

Introduction

Recently, IEEE Spectrum [1] posed the following question to Aart de Geus, CEO and co-founder of Synopsys Inc:

Spectrum: What is the biggest challenge that design automation faces going into the next millennium?

*Geus: The biggest hurdle of designers using EDA is **timing closure**. You design a circuit at a high level using a HDL. You look at its timing and everything is great. Then you move the circuit into its physical design phase and you find out that what you assumed were the delays are much longer. When you try to fix this, you find yourself in the "**hell of iteration**".*

The above statement provides the motivation behind phased logic [2]. Phased Logic is a synthesis/mapping methodology that allows a standard clocked netlist (combinational gates + DFFs) to be automatically mapped to a non-clocked netlist that uses special gates called phased logic gates. Some important features of phased logic are:

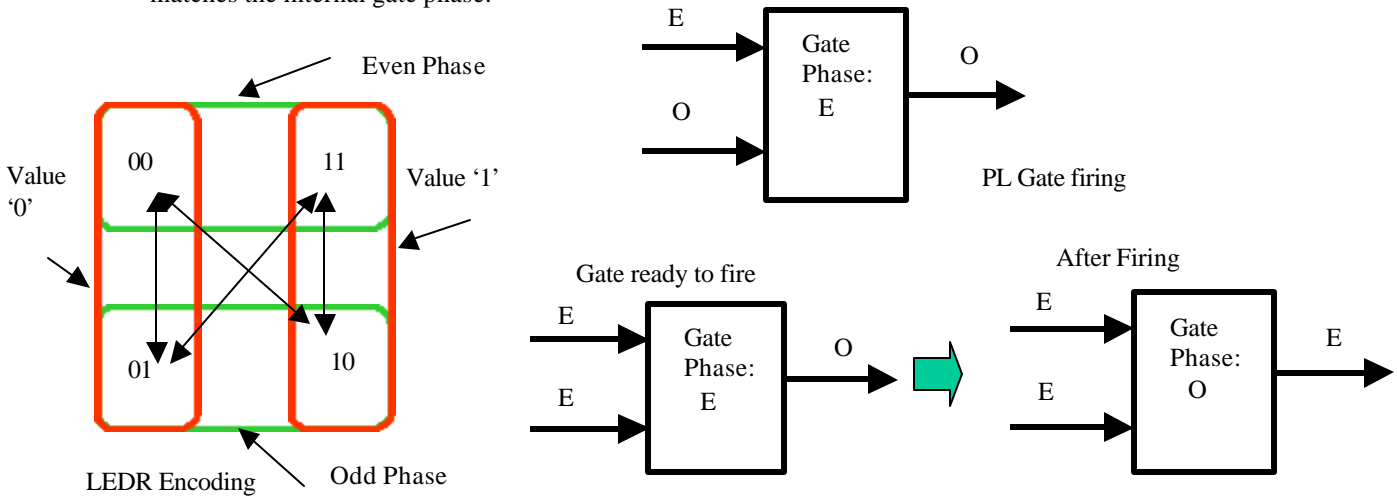
- No global clock network. The only global net is a power on reset.
- A phased logic netlist is delay insensitive to wiring delays between phased logic gates.
- The starting point for the phased logic synthesis/mapping tool is a clocked netlist (combinational logic + DFFs). This means that all standard logic synthesis tools and methodologies can be used to produce the clocked netlist. No massive re-inventing of synthesis tools or design methodologies is needed.
- A phased logic system always computes at its maximum speed.

This paper will give a brief overview of phased logic, then describe results from synthesizing and simulating several phased logic designs ranging from a few hundred to a few thousand gates. The main purpose of this paper is to convince the reader of the viability of the phased logic paradigm via some non-trivial designs, and to illustrate a complete design methodology.

A Brief Overview of Phased Logic

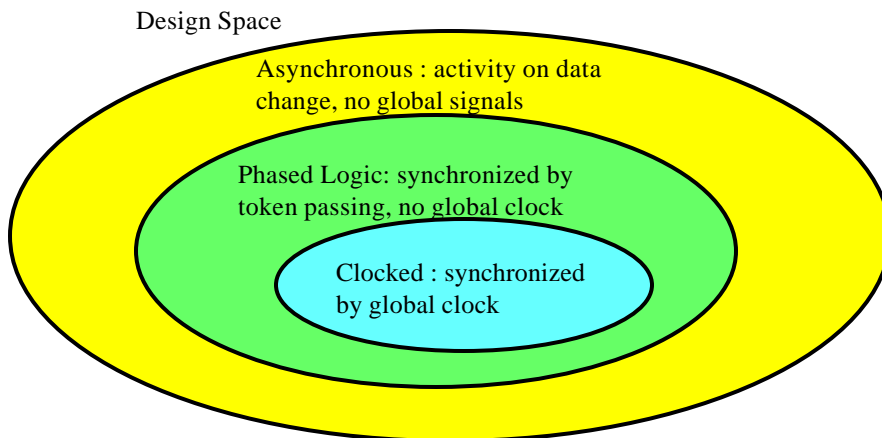
The phased logic methodology was invented by Linder [3] in his dissertation work in 1994. The goal of this work was to produce a digital design methodology that eliminated global clocks. A phased logic netlist can be thought of as a marked graph with data tokens flowing throughout the graph. Each data 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.



The above figure 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 [2], 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. As such, a phased logic system is not a true asynchronous system which only has activity when data values change.



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.
- 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.

Linder's work developed the phased logic theory and implemented a feedback synthesis algorithm that ensured safety and liveness. Linder showed that the additional wiring needed by feedback signals is about 15% to 20% above the overhead of what the dual rail signaling already incurs. No actual simulation of phased logic systems were done. Since the original work, a few small phased logic systems (< 50 gates) have been simulated, but no automated methodology has been available nor has non-trivial PL systems been demonstrated.

Our Contributions

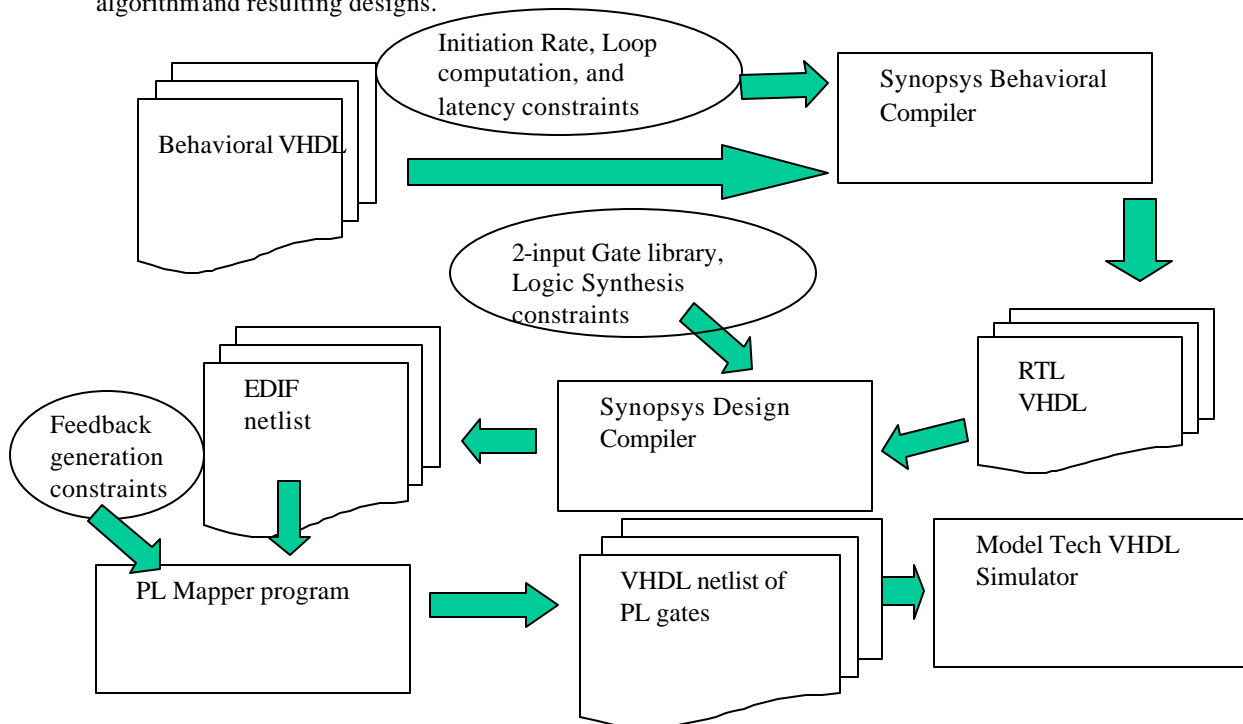
The contributions of this work are:

- Phased logic netlist mapping tool that reads an EDIF netlist and outputs a VHDL phased logic netlist as well as a template for a VHDL testbench. The netlist mapping tool is written in C and operates in a UNIX environment (Sun OS).
- User specified constraints on feedback generation that allow the user to balance CPU time against the number of feedbacks generated, as well as limit the maximum number of feedbacks at any gate.
- A VHDL library that supports simulation of the resulting phased logic netlist within the Model Tech simulation environment.
- An Altera component library that supports the implementation of PL designs in Flex 10K FPGAs.
- Insights into the dynamic behavior of phased logic designs based upon several non-trivial examples.

We have used several test cases to test out the phased logic design methodology. These test cases are:

- Four-bit synchronous counter
- Computation of integer 32-bit square root. The clocked version of this design takes 16 clocks for the computation. The implementation has a 4 state finite state machine.
- Three variations on an 8-tap FIR filter. The FIR has 8-bit input, 8-bit internal datapaths, and 8-bit output. Synopsys Behavioral Compiler was used to generate three variations: a) fir10p0 which has 12 clocks/sample, 1 multiplier, 1 adder; b) fir16p2 - 2 clocks/sample, 6 clock initial latency, 8 multipliers, 5 adders; c) fir15p1 1 clock/sample, 5 clock initial latency, 8 multipliers, 7 adders.

The four bit counter and integer square root designs were only used to verify algorithm functionality whenever changes were made. The FIR designs were used to measure performance aspects of the mapping algorithm and resulting designs.



The previous figure shows the design methodology used to produce the PL test cases. Note that both high level and logic level synthesis tools were used to produce the clocked netlist which was then given to the PL mapping/synthesis tool. All designs were simulated in the Model Tech VHDL simulator.

The table below shows the gate counts for the designs. The FIR designs included a wrapper testbench that provided FIR coefficients and data values via a LFSR, and accumulated an XOR checksum for 64 computations. These gate counts are for minimum area constraints. The designs were synthesized to a two input-gate library that was then mapped to the PL gate library.

Design	Gate Count (DFFs)
Cnt 4	38 (4)
Sqroot32	709 (88)
Firl0p0	1603 (179)
Firl6p2	3106 (214)
Firl5p1	3356 (249)

Feedback Generation

The most CPU-intensive task of the PL netlist mapping is the feedback signal generation. We implemented Linder's original algorithm (ALG-DL) which aimed at introducing the minimum number of feedbacks. Feedbacks can be attached to unused gate inputs or concentrated via additional Mueller C-elements whose output is tied to a gate input. The primary CPU bottleneck in this algorithm is determining which gates are sources of other gates, and computing a score for each possible feedback net. We implemented our own variation (ALG-RT) which allowed user-specified limits to the depth of the search for source gates, and also on the maximum number of feedbacks that can be generated at any node. We wanted to see the effect on CPU performance and number of required feedback nets. The results of the two feedback generation algorithms are shown below:

Design	Fanout Nets	ALG-DL			ALG-RT		
		FBs	CGs	CPU(s)	FBs	CGs	CPU(s)
sqroot32	1351	421	60	54	462	65	34
firl0p0	2864	714	121	266	843	130	50
firl6p2	5612	1504	222	178	1727	226	116
firl5p1	6041	1603	215	199	1942	243	113

FBs are the number of feedback nets added, CGs are the number of additional Muller c-elements needed (4-input equivalent), and CPU is the execution time in seconds (UltraSparc 250Mhz). The ALG-RT numbers were generated using a look-back depth of 8 in the netlist. The results show that only a modest number of additional feedbacks and C-gates were generated by ALG-RT at the benefit of reduced CPU runtime.

Phased Logic System Performance

An interesting question is whether architectural and logic synthesis techniques in the clocked domain aimed at improving performance also improves performance in the phased logic domain. More simply, is a faster clocked design also a faster phased logic design? To test this, we measured the performance of the three FIR filter designs in both the clock and PL domains. In the clocked domain, we used Synopsys to give the longest register to register delay for each design, clocked each system at this rate, and measured the total time for computation of 64 samples. The 2-input gate library used by Synopsys had unit area, unit delay

numbers to emulate an FPGA technology in which all elements have the same area, delay. In the PL domain, all gates had unit delay and the time required to compute the 64 samples was simply recorded from the simulator (a PL system always computes at its maximum possible speed). The results are shown below:

Design	min area		min delay	
	clocked	PL	clocked	PL
firl0p0	1.0	1.0	1.0	1.0
firl6p2	8.6	9.2	7.0	7.2
firl5p1	17.4	16.7	14.4	14.0

The minimum area numbers were for designs synthesized to minimum area constraints; minimum delay numbers were for designs synthesized to minimum delay constraints. The numbers in each column were normalized to the worst case so that we could see relative improvements for different architectural variations. In each case, a fast clocked design meant a fast PL design, and the relative improvement between the designs were approximately the same.

Areas of Future Effort

Challenges for the future in phased logic are:

- **Power:** Power consumption is a problem with phased logic - tokens continue to circulate even if no data values are changing. Gated clock design in clocked netlists are used to save power, and the same techniques can be applied to PL designs to stop token circulation. We have demonstrated a test design where token circulation is only done while there is computation being performed.
- **Area:** Because of the complex nature of a phased logic gate, phased logic will only be able to compete in the reconfigurable logic arena. Current FPGA architectures are very inefficient at implementing PL gates. We are looking at alternate FPGA architectures for implementations of phased logic gates.
- **Speed:** The phased logic gate discussed in this paper is not capable of early evaluation (i.e. it will only evaluate when all input tokens are present). This means that the speed of a phased logic system is based upon the longest delay path, the same as in a clocked system. However, we have demonstrated internally a variation of a phased logic gate that does allow early evaluation based upon a single input value (a '0' logic value on a AND gate input should allow firing regardless of other inputs). This would allow PL system performance to be based upon the average delay path, not the longest delay path, and could give it a speed advantage over reconfigurable clocked logic for some applications.

Summary

We have successfully demonstrated the synthesis and simulation of non-trivial phased logic systems using familiar design tools and methodologies. We have implemented a new feedback generation algorithm that reduces CPU time for feedback generation at a small cost of additional feedback nets and C-elements. Finally, we have created an design environment that will allow us to more fully explore the possibilities of phased logic design.

References

1. Geppert, Linda (Senior Editor), "Electronic Design Automation", *IEEE Spectrum*, January 2000, pp 70-74.
2. Linder, Daniel H. and James C. Harden, "Phased Logic: Supporting the Synchronous Design Paradigm with Delay-Insensitive Circuitry", *IEEE Transactions on Computers*, vol 45, pp. 1031-1044, Sept. 1996.

Presented at IWLS 2000, June 2000.

3. Linder, Daniel H., *Phased Logic: A Design Methodology for Delay-Insensitive, Synchronous Circuitry*, Ph.D dissertation, Mississippi State University, 1994.
4. 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.