Automated Design for Current-Mode Pass-Transistor Logic Blocks

Matthew David Pierson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-70
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-70.html

May 19, 2007

Automated Design for Current-Mode Pass-Transistor Logic Blocks

Matthew David Pierson
University of California at Berkeley
Department of Electrical Engineering and Computer Sciences

May 18, 2007

Automated Design for Current-Mode Pass-Transistor Logic Blocks

by Matthew David Pierson

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Jan M. Rabaey
Research Advisor

Date

* * * * * *

Professor Kurt Keutzer
Second Reader

Date

Table of Contents

List of Figures

Acknowledgements

This work would not have been possible without the help and influence of many people. First, I would like to thank professors Jan Rabaey and Kurt Keutzer. Both have offered guidance to me during my tenure here at Berkeley as well as reading this paper and offering helpful suggestions. Second, to all of the people I have met and worked with at the BWRC. I have learned more here in the last two years than I could've ever imagined, and a lot of it is due to my colleagues at BWRC: Nathan Pletcher, Louis Alarcon, Simone Gambini, Cristian Marcu, Jesse Richmond, Michael Mark, Tsung-Te Liu, David Chen, and Luca DeNardis. Whether it's helping me with work or just offering a friendly distraction, I could not have done it without all of your help. Special thanks also go to all of the friends I made when I arrived who made it so much easier to adapt to new surroundings in a new city. I hope to be at all of your weddings, just like you all came to mine.

Next, to my friends and family, most of whom I left back in Texas. My parents have always offered their everlasting support, and I am extremeley lucky and grateful for all of it. To my friends back home, I can't wait to make up for the time I've missed with you all the last two years. Being able to talk to you all with 1800 miles

between us has made this entire experience easier.

Finally, I owe the world to my wife Jennifer. You have been there for me since we met, and I can't imagine how my life would be now without you by my side through all of it. To everyone here, once more, Thank You.

Chapter 1

Introduction

1.1 Motivation

The rapid advancement of technology constantly creates new avenues for electronics application. This innovation, at the same time, leads to greater complexity and increased design time. To keep up with this complexity, automated tools are needed in the design process. Digital circuits are robust enough to make automatic implementation possible, and tools are widely used in industry and academia. New design styles, however, do not usually fit directly into current tools, and until tools are available their applicability is severely limited and research is slowed. Application in industry is not possible until tools are available since Time To Market (TTM) is one of the most important, if not the most important constraint.

While transistors are shrinking, integration levels and the size of silicon die are growing. In traditional synchronous designs, clock distribution and synchroniza-

tion become difficult tasks and lead to very conservative design margins or excessive power/area budgets. For these reasons, self-timed logic is slowly emerging as a viable alternative. This solution allows each block in a design to move at its natural speed and does not require a large synchronized network to distribute a global clock. Unfortunately, tool support for self-timed designs is not as widespread as traditional synchronous designs.

1.2 Contribution

This research is meant to create an automated design flow to speed implementation and take advantage of key characteristics of a new self-timed logic style. Presented is a complete front to back design flow for flat[1] datapath blocks. The tools not only automate the implementation but also optimize in the energy, area, and delay domains.

1.3 Organization of Report

This report focuses on the self-timed logic style, its characteristics, and the design flow that allows designers to quickly create implementations that meet design requirements. Chapter 2 presents background information on the new logic style, asynchronous logic techniques and their application to the logic style, and current automated design. Chapter 3 discusses the synthesis portion of the design flow which

[1] Designs without multiple levels of hierarchy

creates a gate level implementation of RTL to meet design goals. Next, Chapter 4 focuses on filling the gap between synthesis and the physical implementation of the circuit. Then Chapter 5 covers the automation of the physical implementation of the circuit onto silicon for manufacturing. Finally, Chapter 6 summarizes the work as a whole and looks at future directions for the design flow.

Chapter 2

Background

2.1 Current-Mode Pass-Transistor Logic

Over the past 15 years power consumption has gone from a non-factor to a major design constraint, especially with the popularity of mobile devices that cannot afford a heat sink or active heat dissipation. Since power is energy per unit time, reducing power means decreasing the dissipated energy or spreading the same energy over a longer time. Since performance requirements often quickly follow power requirements making the computation time longer is not usually an option. Therefore, the energy per operation must be reduced in order to keep the same performance and still reduce the power consumption. This is the goal of the current-mode pass-transistor project.

This current-mode pass-transistor logic is made up of logic gates called CLBs. A CLB can implement any logical function up to seven inputs. Figure 2.1 shows a two-input CLB implementing an AND function. On the left, a current steering tree

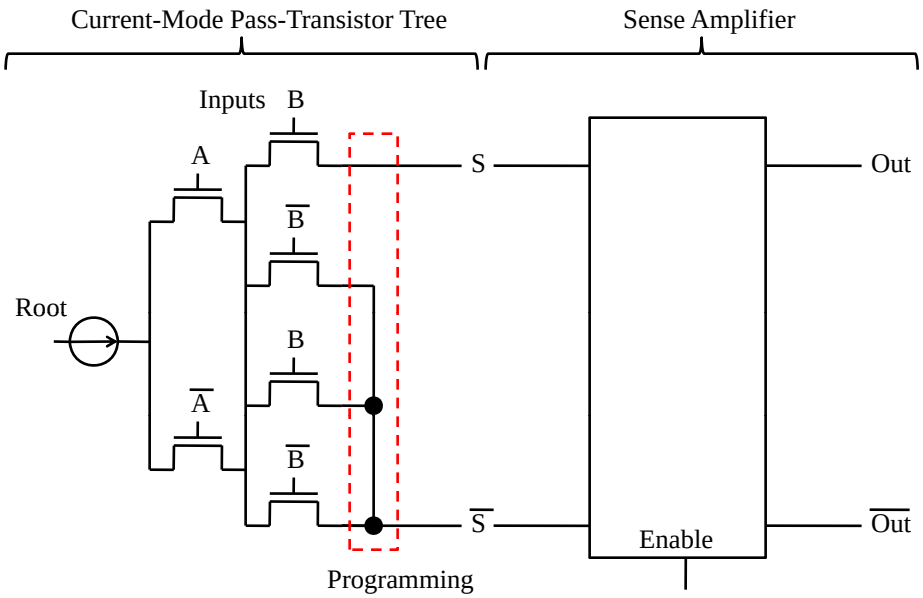

Figure 2.1: CLB Structure

computes the output once the inputs are valid and the root drive is activated. The logic function of the CLB depends on which terms of the tree are tied to S (on set) or $\overline{S}$ (off set). Then a sense amplifier, right, is activated by a clock to latch the output and bring it to full swing at Out and $\overline{Out}$.

The stack effect in the tree and the limited number of power/ground connections allow for very low threshold devices without major leakage penalties. Reducing the threshold of these devices will allow for high speed with large trees. Using pass transistors this way also eliminates the possibility of sneak paths (shorts from power to ground) common to traditional pass-transistor design. The ability of the sense amplifier to sense small voltage differences will allow for a minimum amount of energy required to do the actual computation in the tree. However, these advantages come

at the price of timing complexity. The tree needs to be reset before each computation to "erase" all of the information from the previous one, and the sense amp needs to know when to latch the output. Originally, a pipeline of these CLBs was timed using two out-of-phase clocks [1]. However, global clock distribution is a major issue in large designs, and building control logic with feedback loops out of clocked CLBs is not trivial. These reasons led to a new self-timed CLB implementation.

2.2 Self-Timed Logic

Self-timed logic presents many advantages over traditional synchronous logic. A global clock distribution network is not required, freeing up design time, power, and area. Local timing makes the circuit more robust to manufacturing variations and allows for more aggressive timing strategies. Traditional synchronous designs have a clock transitioning all of the time, even when the circuit is doing nothing useful. Self-timed designs, however, remain in a stable quiescent state with no signal transitions until a request is made; this can be a great help in passive energy dissipation. This is very advantageous for applications running on batteries that are heavily duty cycled, but these advantages come at a design difficulty cost, and industrial tool support for asynchronous designs in still in its infancy.

Self-timed logic requires control signals to travel with data instead of a global clock synchronizing everything. In each data transaction a *sender* transmits data to a *receiver* [2]. The data transfer takes place on three lines connecting sender and receiver: one data bus, a request (Req) control line, and an acknowledge (Ack)

control line. The sender places data on the line and switches the value of the Req line. The sender then waits for the receiver to acknowledge acceptance of the data by changing the polarity of the Ack line; during this time the flow of data from sender to receiver is stalled. This process repeats for each piece of data moving through the pipeline. This protocol is referred to as two-phase handshaking and is the simplest option. However, this scheme depends on circuitry sensing transitions in both directions. Typical CMOS circuits are usually level triggered or sensitive to one specific transition, not both transitions [3].

Another protocol for self timing is four-phase or return-to-zero signaling. In this protocol, all control signals return to their initial value between each data transfer. Just as before, the sender places data on the line and signals Req. The receiver signals on Ack when the data is accepted. Then the sender returns the Req line to its initial value followed by the receiver returning Ack to its initial value, signaling that it is ready for the next cycle [3].

The CLB logic block operates in a four-phase mode, starting in a reset phase, evaluating, passing on its data, and then returning to the reset phase before the next evaluation. For this reason a four-phase protocol is a natural choice for self-timed CLBs. Four phase signaling can be easily synthesized using Muller C-elements and is shown in Figure 2.2 [4]. Below in the same figure is the C-element implementation applied to a CLB pipeline.

There is one minor change needed to the four-phase implementation of Figure 2.2. The dotted connection will lead to incorrect operation because of the sense amplifiers' operation. When the enable input to the sense amp is a logic 0, the sense

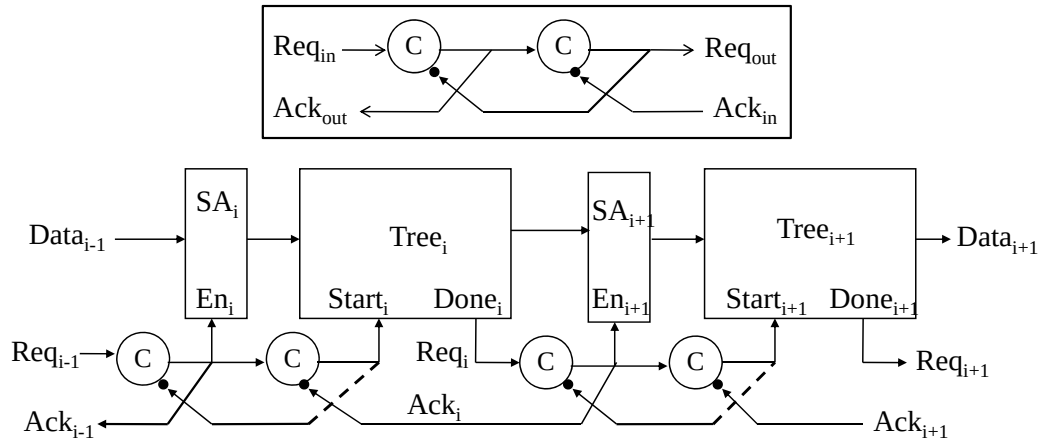

Figure 2.2: Four-Phase Handshaking with Muller C-elements

amplifier goes into a reset mode and brings both outputs high to precondition the next evaluation tree. In the above implementation this will happen before the next sense amp will latch the correct output, erasing the data value at the output of the sense amp. To fix this, the C-element controlling the first sense amplifier must take the acknowledge from the next C-element as shown in Figure 2.3.

This modification actually leads to further optimization of the handshaking pro-

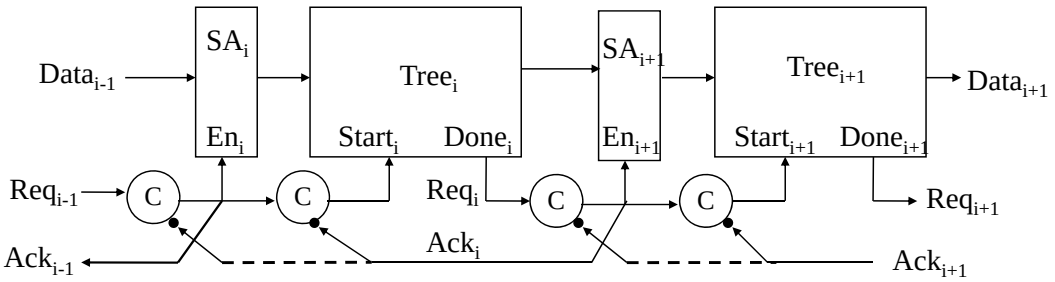

Figure 2.3: Four-Phase Handshaking for CLB Pipelines

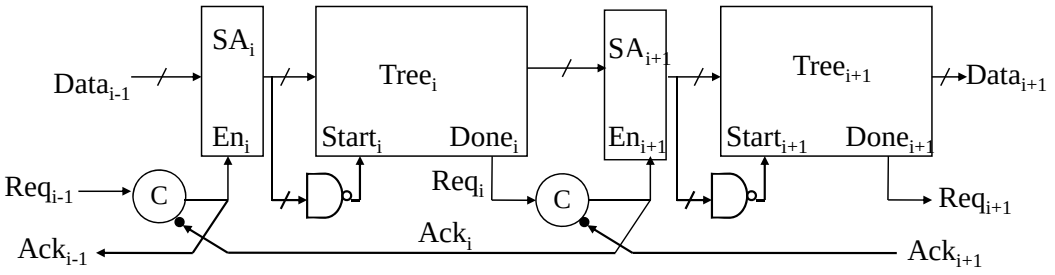

Figure 2.4: Final Self-Timed CLB Pipeline

tocol. The C-element between the output of the sense amp and input of the tree now acts as a delay and does no useful work. This C-element can be taken out entirely, but this could lead to a race condition resulting in tree evaluation before the inputs are completely valid. Instead of removing the C-element, a static NAND gate replaces it. This NAND gate acts as a completion detector, sensing when the inputs of the tree are valid. When the outputs of the sense amp are (1,1) the NAND gate keeps the tree in reset mode. Once the outputs are (1,0) or (0,1), the NAND output is asserted, signaling valid data. (0,0) is a disallowed combination and never occurs by design. Figure 2.4 shows the final optimized pipeline for the CLB logic blocks.

To make the CLBs fit into the design flow each block will be standardized with the necessary logic gates to make it completely self-timed. Then the blocks simply need to be hooked up to create more complex circuits, just like standard library-based design flows. To do this, fan-in and fan-out differences need to be considered. This self-contained block is seen in Figure 2.5 with the dark, thick arrows representing data and the thin, dotted lines control.

When all of the input valid signals are high, the AND gate turns on the root drive

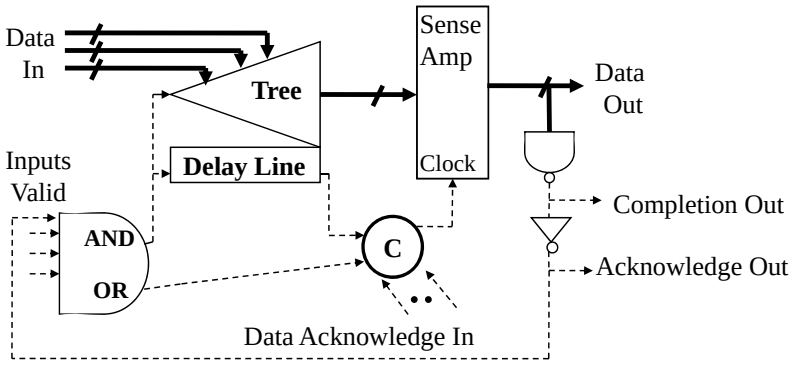

Figure 2.5: Self-Timed CLB Block Diagram

for the tree and begins computation. Next, the tree must signal when its output is evaluated enough that the sense amp can be triggered and latch the output. The reduced swing of the tree's output makes completion detection difficult, so a delay line mimics the delay of the stack and signals when the output is evaluated. The OR gate signals when all of the inputs have gone into reset and then allows the current block to go into reset to keep all data signals in the same timing plane. All of the gates in this block have known sizes depending on the number of inputs to the CLB except for the C element, which has a number of inputs based on the fan-out of the output signal. All that is left is to determine the functionality of each CLB and hook the blocks up to create more complex circuits, just like hooking up gates in static CMOS.

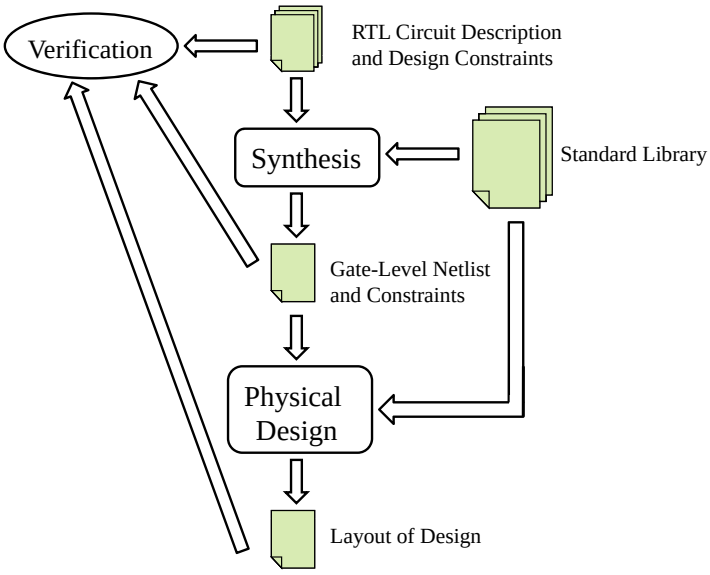

Figure 2.6: Typical Automated Design Flow

2.3 Automated Design

As mentioned in Chapter 1, the reduction of transistor dimensions has led to the growth in complexity and size of current designs. At current technology levels, tens of millions of transistors can exist on one single chip. Even small designs easily contain ten to one hundred thousand devices, still too many for hand design. Current tools start from a high level programming language (normally Register Transfer Language) and compile it down to physical layout in two major phases: synthesis and physical design. The typical RTL tool flow is best conveyed with the flow chart in Figure 2.6.

The RTL specification, commonly in VHDL or Verilog, is compiled into gates during the synthesis process. Usually, the gates are part of a standard library of commonly used logic functions with each cell having logic function, timing, area,

power, and layout information. There are a few cycles of optimization during synthesis to get as close to design constraints as possible. This phase produces a structural netlist file containing gates and interconnection information. The physical compiler receives this file along with design constraints on power, area, timing, and loading and creates the physical layout of the interconnected devices ready to be manufactured. There is often optimization during the physical design steps since each completed step brings more specific information such as wire length and resistance. Also, between each step verification is necessary to make sure the optimized design is the same as the original [5]. It is a main goal of the new design flow to remain as close to the current state of industry tools as possible.

Chapter 3

Synthesis

3.1 Introduction

During ASIC[1] synthesis the design is mapped to a standard library of gates. Using this limited library allows extensive testing of each gate to give the software enough information to choose the best combination of gates. This type of synthesis is normally done in two steps, technology-independent optimization followed by technology-dependent optimization. In the first step the Boolean function is optimized using mathematical techniques such as transformations, division and others. In the second step the library gates are optimally mapped onto the function using tree covering [6].

The synthesis process for field programmable gate array (FPGA) designs is slightly different. The first step of FPGA synthesis is still a technology-independent optimiza-

[1] A circuit created for a specific task, not a reprogrammable or general purpose circuit

tion of the Boolean function, but the second step is different because a lookup table (LUT) can implement any logic function of N inputs. The mapping step takes the logic function of each block into account directly instead of using a predetermined library [6].

Just like LUTs in FPGAs, CLBs can be programmed with any logic function of a set number of inputs. Using FPGA synthesis takes advantage of this and should be the better choice for the synthesis step. This results in a hybrid FPGA/ASIC design flow since the CLBs are not intended to be field programmable.

3.2 CLB Synthesis using FPGA Technology

The Logic Synthesis and Verification group at UC Berkeley provides state-of-the-art variable-size-LUT FPGA synthesis in its software named ABC [7]. The synthesis is based on DAOmap, a cut-based enumeration method that provides a depth-optimal solution using heuristics to reduce the area [8]. ABC builds on this system with optimizations to cut computation, delay optimum mapping, and heuristic area recovery [9]. In short, ABC traverses the design and chooses the logic function and size of all CLBs to provide a depth-optimal mapping. Unfortunately, the current version of ABC only provides optimization for LUTs of six or less inputs. CLBs come in sizes up to seven, and we use the RASP Technology Mapping tool from the UCLA VLSI CAD lab to create the data points for CLB seven [10]. RASP implements the same DAOmap algorithm integrated into the ABC environment, but without the optimizations performed by the Logic Synthesis and Verification group at Berkeley.

To incorporate this into the flow, the design must be translated from behavioral RTL into the Berkeley Logic Interchange Format (BLIF) [11]. This can be done using any software capable of outputting netlists in BLIF format, but in this work this is done using Altera Quartus II software. The Altera software will also perform technology independent optimization, but using Synopsys's DesignCompiler in this research led to better results.

3.3 Synthesis for Energy, Delay, and Area

Once the design has been translated into BLIF format, it needs to be fed into ABC along with the variable-LUT characteristic file. This is a text file which has the area and delay characteristics for each LUT size. By design, all CLBs with the same number of inputs have the same delay and area characteristics because the only difference between same-size CLBs is the programming. The timing differences based on fan out are not included in this research and exist as an avenue for further study. After the design and LUT characteristic file have been input, the *fpga* command performs the optimizations.

Figure 3.1 visually shows the optimization space with the available CLBs. All of the data is normalized in relation to the three-input CLB. As seen in the figure the area, delay, and active energy consumption grow quadratically as CLB inputs increase. The leakage, however, increases only slightly as CLB inputs increase because the sense amplifier dominates the leakage of the CLB. CLBs with three or four inputs use the same sense amplifier, and CLBs with five or six inputs share a sense amplifier as well.

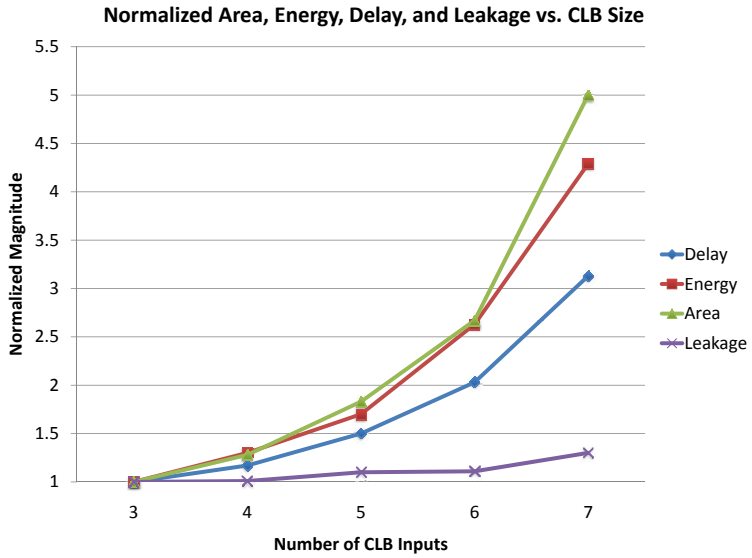

Figure 3.1: Relative Area, Delay, Energy, and Leakage Performance of CLBs

This causes the flat portions of the otherwise linear curve.

ABC only produces one implementation, but by limiting the CLB sizes available to ABC different implementations can be made. For example, the information about CLB six and seven can be eliminated from the text file, limiting ABC to using CLB sizes three to five. Synthesis was initially performed with a maximum CLB size of three. Then synthesis was repeated using three and four input CLBs, followed by three, four, and five, and so on to a maximum size of seven. Figure 3.2 shows the results of this process when applied to a reference design.

Figure 3.2 shows the delay reaching a lower bound as we increase the optimization space for the synthesis tool. The delay does not change when the seven-input CLB is added to the synthesis, meaning the tradeoff between functionality and delay for the

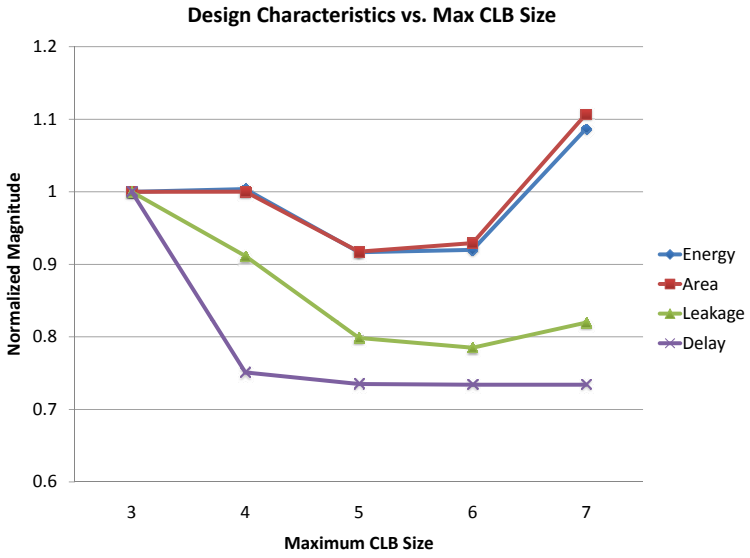

Figure 3.2: Optimization Chart for Example Circuit

seven-input CLB did not provide a benefit. Performing synthesis with a maximum CLB of six inputs resulted in a 25% delay reduction, a 22% leakage reduction, and an 8.5% area and energy reduction over the only three-input CLB implementation. This process allows the designer to constrain the tool to obtain the best implementation possible. Area and energy suffered in the last implementation. This interesting result could be an artifact of the heuristic optimizations present in ABC, but not in RASP. If this is the case, this shows the improvements in the heuristic area recovery. DAOmap already offers a depth-optimal solution before the ABC improvements so the delay shouldn't change between the tools.

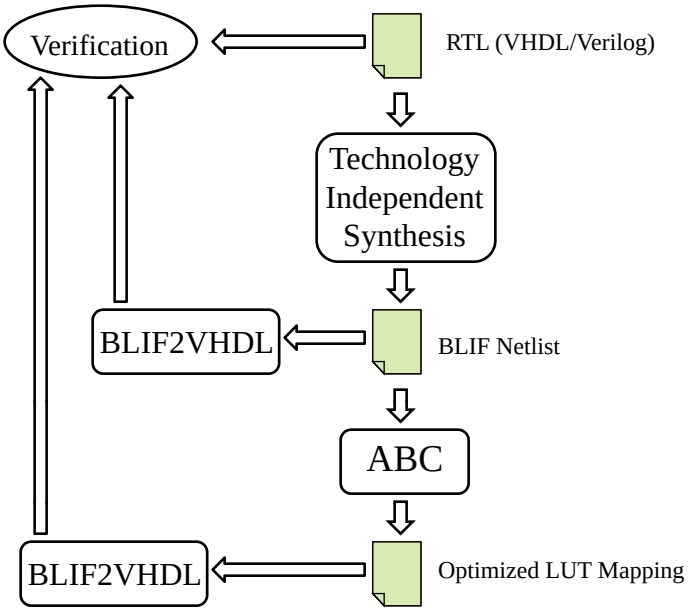

Figure 3.3: Synthesis Flow Diagram

3.4 Post Synthesis Verification

The next step in the process is to make sure that the optimized design is functionally equivalent to the original behavioral design. ABC has combinational and sequential equivalence checking with the *cec* and *sec* commands respectively for verification. Also, for verification with the original behavioral RTL, Synopsys's Formality can be used. For the second option the BLIF file must be converted to VHDL with blif2vhdl, a netlist translator [12]. This ends the synthesis process, and the BLIF netlist now contains a LUT implementation of the desired circuit. The next step is to transform the BLIF file into a structural verilog format the physical design tools can accept. Figure 3.3 shows the flow of the synthesis process including verification steps.

Chapter 4

Bridging Front End and Back End

Before handing off the netlist to the back end of the flow, a few issues need to be cleaned up, and the netlist needs to be transformed to structural verilog. The first step takes place in a Perl script named BlifFlow.pl which takes the optimized BLIF file and expands all of the reduced terms and changes the binary terms into decimal representation for the netlist translator. It also cleans up and conforms all of the net names for the translator. Finally, BlifFlow flow prints two files, one BLIF for input into the translator with conformed net names and decimal terms, and one BLIF for verification with ABC to ensure no functionality changes occurred. This file is then fed into a Python netlist translator which will create the verilog netlist of CLBs.

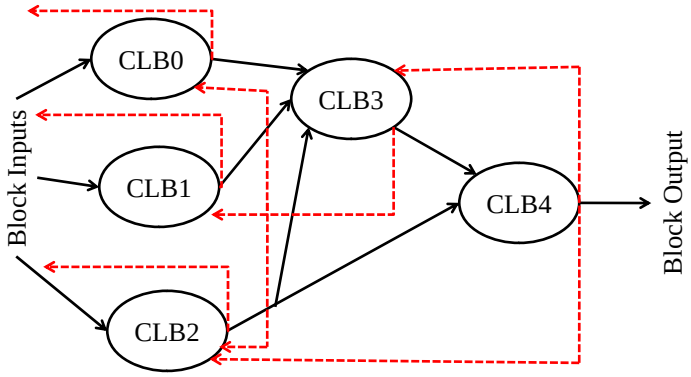

Figure 4.1: Graph of simple CLB design

4.1 Transforming LUTs to CLBs

The CLBDesign Python script reads in the modified BLIF netlist and stores the design as a directed acyclic graph (DAG), with CLBs representing the nodes of the graph and wires representing the edges. Each CLB knows where its output is routed as an input. This is important for routing the acknowledge signals which travel backwards instead of the normal forward propagating signals. An example graph of a simple circuit is shown in Figure 4.1. Solid lines represent forward traveling data and control signals, and dotted lines represent the backwards traveling acknowledge signals.

4.1.1 Changing sizes of CLBs

First, the script transforms CLBs with less than three inputs into three-input CLBs. Anything smaller than a three-input CLB will have virtually the same performance

as a three-input CLB because the self-timed gates and sense amp overhead dominate the area and power budget at that size. For this reason no CLBs smaller than three inputs were made. However, the synthesis tools will instantiate CLBs with less than three inputs so these must be transformed into three-input CLBs. This is done by inverse Boolean elimination. The tree before and after transformation is shown in Figure 4.2. The first input is set to V_{dd}, and then each minterm and maxterm has 2^{inputs} added to it and added to the original list of terms. This way the Boolean equation will still reduce back down to its original form. This process is repeated as needed to make all CLBs have at least three inputs.

4.1.2 Making Heterogeneous Mappings Homogenous

This next transformation is an option to the designer instead of a required step. Since this design will be auto-placed and auto-routed, layout regularity can be a very important property.

The layouts for CLBs range in size greatly from three inputs to seven inputs, and having a completely heterogeneous mapping can make it difficult for the auto place and route tools with blocks of different sizes. If a more regular layout is desired then the entire design can be transformed to only contain a single size of CLBs. This is referred to as a homogenous mapping. In this case, every CLB is checked and if it does not match, then the size transformation is done as many times as necessary. This transformed BLIF can now be verified against the original in either ABC or with Formality after blif2vhdl. This can be extremely costly in the design optimization

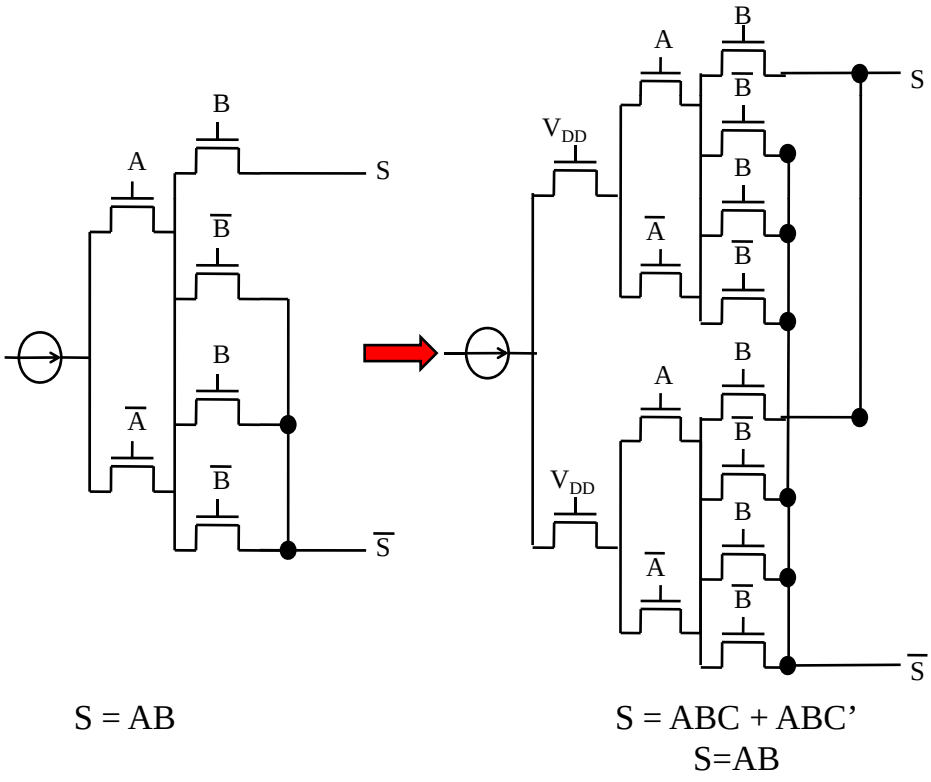

Figure 4.2: Left, a two-input AND tree. Right, a logically equivalent three-input tree.

space. In the future, changing the layouts of the CLBs to make them easier to auto place and route might be a good idea and would make this option obsolete.

4.2 Simulation and Physical Design

Once all of the necessary transformations have been completed the design can now be printed in a verilog netlist. This design includes structural verilog for the higher level design and the individual CLBs, along with Skill[1] code to generate the programmed layout for the CLBs. The structural verilog can be imported and simulated in spectre or similar circuit simulator. This is where functionality, transient, and energy performance can be checked and the circuit modified if necessary.

[1]Programming Language used in Cadence CAD tools

Chapter 5

Physical Design

The final step in the design process is to create the mask layout for design manufacturing. This can be the most tedious step in the process to do manually and with current tools can be automated. Most of the physical design process is unchanged from typical design flows with one step added to generate the layout of each CLB.

5.1 Creating the CLBs

5.1.1 Programming the CLBs

The Python script in the last step generated a Skill script to create the programmed layouts of the CLBs. Since every un-programmed tree looks the same, this is a simple process to automate. Shown below is an example layout of the tree along with the locations of all terms. The S and $\bar{S}$ lines are also highlighted. A via simply needs to be applied connecting all terms to either S or $\bar{S}$ and saved as the new programmed

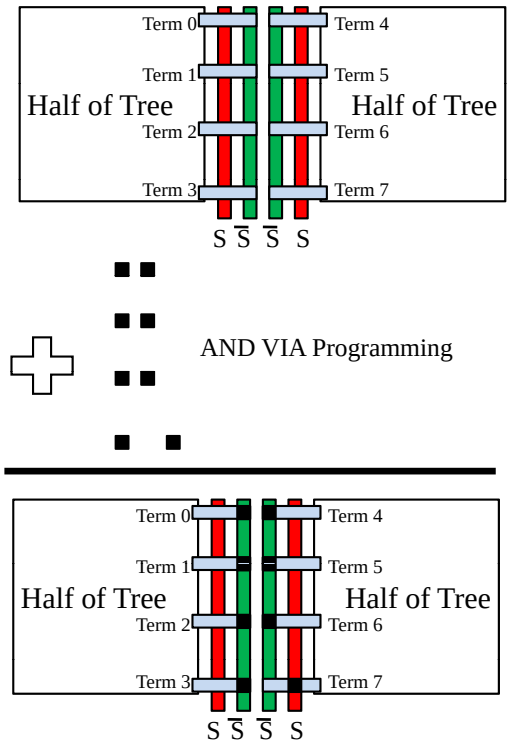

Figure 5.1: Programming the Tree

CLB layout. Figure 5.1 shows how this process works.

5.1.2 Import CLBs

Next, the CLBs need to be imported into the place and route tool as standard cells. This simple process can be scripted to tailor to the specific design needs. The layouts are created to allow for abutment of similar size CLBs, as well as flip and rotate options.

5.1.3 Place and Route

The final step in the physical design process is the actual place and route of the design. This is also a scriptable step and can be tailored to different designs. In the current version, the tools do not have enough information to re-optimize the gate-level netlist, but most other tool customizations are available to the designer.

5.2 Final Verification

After the design has been placed and routed, layout versus schematic (LVS) and design rule checks (DRC) can be run. This serves as the final verification step and ensures the implemented circuit matches the original design. Figures 5.2 and 5.3 show an overview of the physical design portion of the flow and a placed and routed circuit produced from the flow.

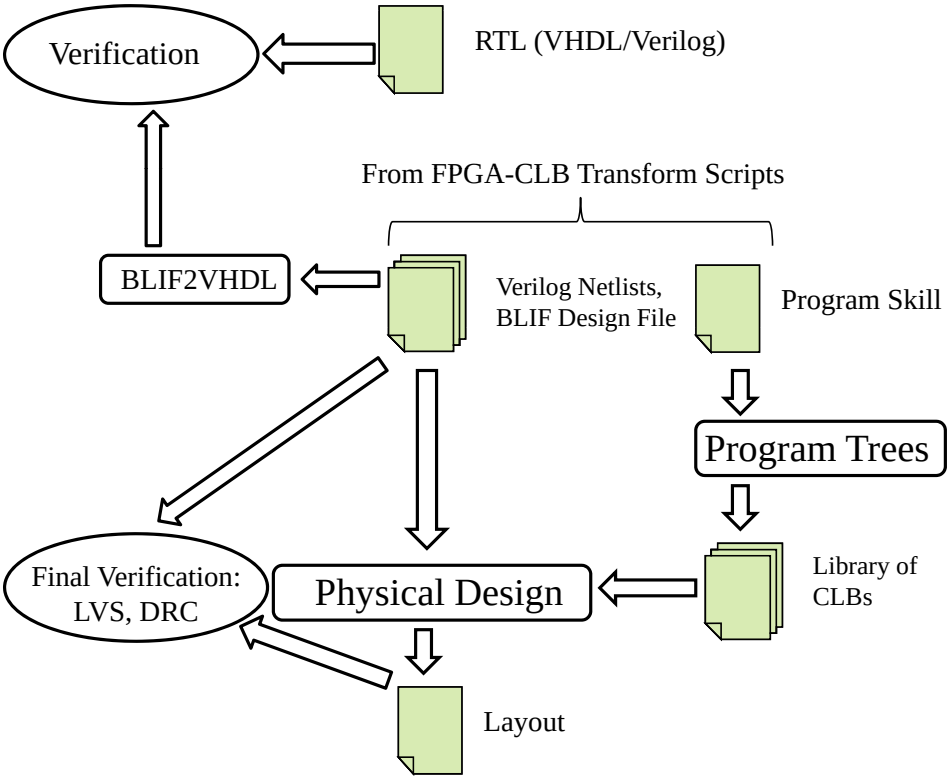

Figure 5.2: Physical Design Flow Diagram

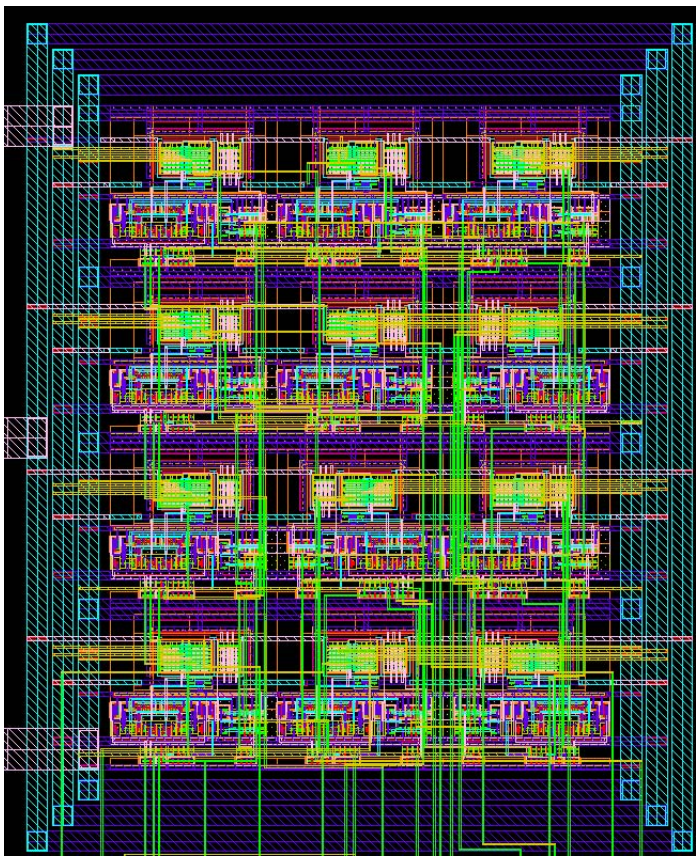

Figure 5.3: Final Placed and Routed Circuit

Chapter 6

Conclusion

6.1 Summary of Work Completed

This research provides a full automated design flow for the new self-timed current-mode pass-transistor logic design style. It includes a full synthesis package for optimizing area, delay, and energy consumption. Scripts produced for this research provide the FPGA to CLB transformations and produce the necessary files to feed the design into industrial physical design software. Finally, a full back end is mainly made up of current state of the art industry tools with slight modification to the automated process. This flow greatly speeds up design efforts in both research and application fields. This is a must for any new logic style to have a chance at acceptance and application in user environments. Figure 6.1 shows all steps of the flow including verification points. Appendix A the end of this document has a table including all of the software used in this new design flow.

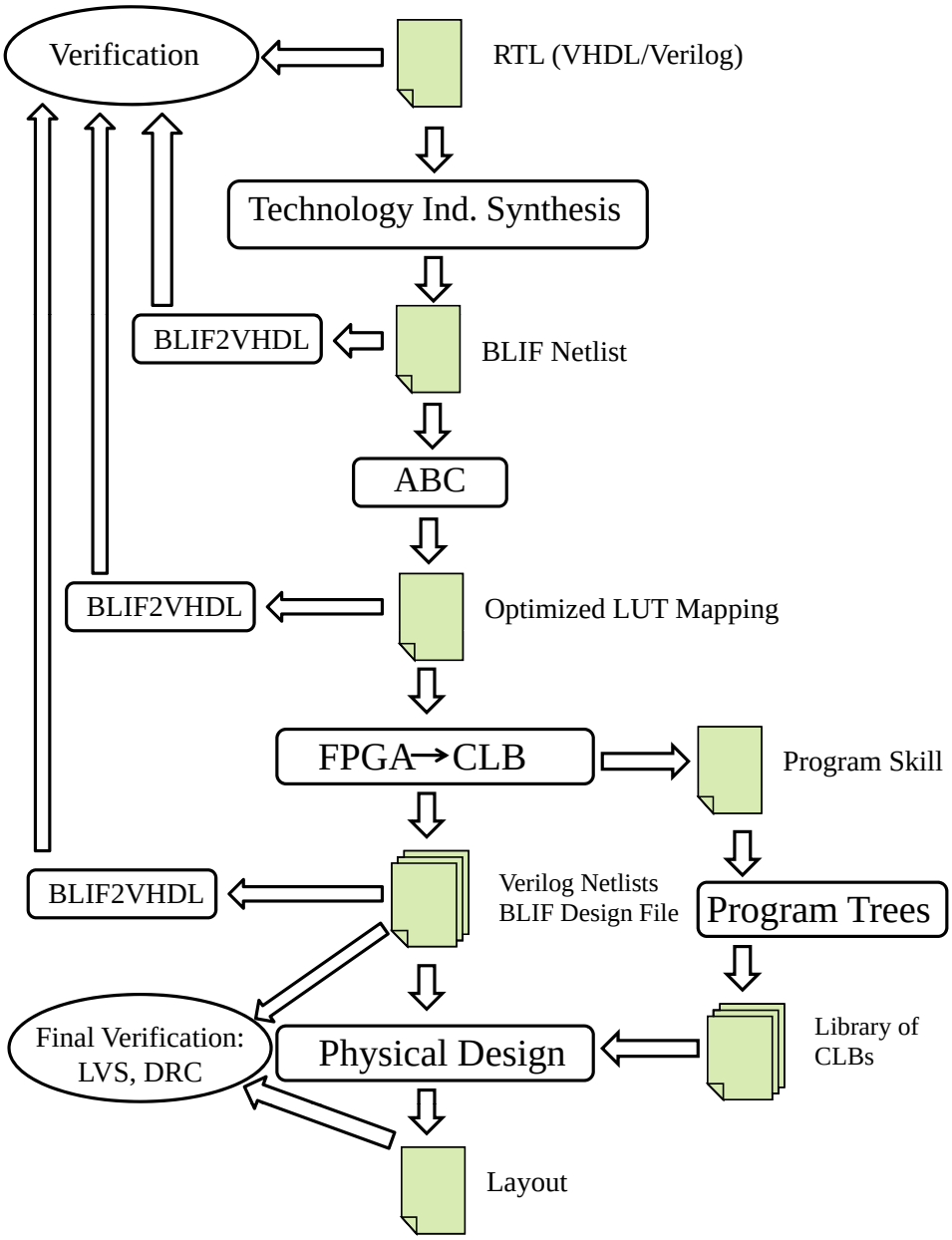

Figure 6.1: Flowchart of Full Design Flow

6.2 Future Work

As with any research, there are an infinite number of possible extensions to make the automated flow even better.

6.2.1 Automated Control Logic

The current applications of the logic style have been limited to data path logic with no feedback loops. For this reason, sequential optimization has been left out of the automated flow. Once more research is completed in applying the logic style to control logic, the flow can be extended to include sequential designs. All of the individual tools used in the flow have some sequential optimization capability.

6.2.2 New Architectures

The design flow currently only works with the self-timed version of the logic style. Other architectures for self-timing are being researched and can also be easily integrated into this design flow. These architectures could lead to different timing and energy behavior based on tree programming. A library-based approach might work better in this case.

Bibliography

[1] L. Alarcon, "Ultra-low energy logic using pass transistor stacks," to be published.

[2] I. Sutherland, "Micropipelines," *Communications of the ACM*, pp. 720–738, June 1989.

[3] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Prentice Hall, 2002.

[4] T. H. Meng, *Synchronization Design for Digital Systems*. Kluwer Academic Publishers, 1991.

[5] K. Keutzer, "EE244 Course Notes," University of California, Berkeley, 2005.

[6] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis*. McGraw-Hill, 1994.

[7] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification, Release 61225," http://www.eecs.berkeley.edu/∼alanmi/abc/.

[8] D. Chen and J. Cong, "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," *Proc. ICCAD '04*, pp. 752–757, 2004.

[9] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to Technology Mapping for LUT-Based FPGAs," *FPGA '06*, 2006.

[10] RASP: FPGA/CPLD Technology Mapping and Synthesis Package. [Online]. Available: http://cadlab.cs.ucla.edu/software release/rasp/htdocs/

[11] *Berkeley Logic Interchange Format (BLIF)*, University of California at Berkeley, July 1992.

[12] blif2vhdl. [Online]. Available: http://tams-www.informatik.uni-hamburg.de/vhdl/index.php?content=07-tools

Appendix A

List of Software used in Design Flow

Synthesis	
Technology Independent Synthesis	Altera II Quartus Web Edition Synopsys DesignCompiler
RTL to BLIF Translation	Altera II Quartus Web Edition
BLIF to VHDL Translation	blif2vhdl
FPGA Synthesis	ABC RASP
Post-Synthesis	
FPGA to CLB Mapping Scripts	BlifFlow.pl CLBDesign.py
Physical Design	
CLB Programming	Cadence ICFB
Physical Design Software	Synopsys Astro
Verification	Synopsys Formality ABC
Signoff Verification	Calibre LVS,DRC