ACHIEVING BACKEND ROBUSTNESS FOR TIMED ASYNCHRONOUS CIRCUITS

by

William Lee

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

The University of Utah

December 2016

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **William Lee**

has been approved by the following supervisory committee members:

Kenneth S. Stevens	, Chair	**3/23/2016** Date Approved
Alan Davis	, Member	**3/23/2016** Date Approved
Pryank Kalla	, Member	**3/23/2016** Date Approved
Erik L. Brunvand	, Member	**3/23/2016** Date Approved
Marly Roncken	, Member	**3/23/2016** Date Approved

and by **Gianluca Lazzi** , Chair/Dean of

the Department/College/School of **Electrical and Computer Engineering**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

The design of integrated circuit (IC) requires an exhaustive verification and a thorough test mechanism to ensure the functionality and robustness of the circuit.

This dissertation employs the theory of relative timing that has the advantage of enabling designers to create designs that have significant power and performance over traditional clocked designs. Research has been carried out to enable the relative timing approach to be supported by commercial electronic design automation (EDA) tools. This allows asynchronous and sequential designs to be designed using commercial cad tools. However, two very significant holes in the flow exist: the lack of support for timing verification and manufacturing test.

Relative timing (RT) utilizes circuit delay to enforce and measure event sequencing on circuit design. Asynchronous circuits can optimize power-performance product by adjusting the circuit timing. A thorough analysis on the timing characteristic of each and every timing path is required to ensure the robustness and correctness of RT designs. All timing paths have to conform to the circuit timing constraints.

This dissertation addresses back-end design robustness by validating full cyclical path timing verification with static timing analysis and implementing design for testability (DFT).

Circuit reliability and correctness are necessary aspects for the technology to become commercially ready. In this study, scan-chain, a commercial DFT implementation, is applied to burst-mode RT designs. In addition, a novel testing approach is developed along with scan-chain to over achieve 90% fault coverage on two fault models: stuck-at fault model and delay fault model. This work evaluates the cost of DFT and its coverage trade-off then determines the best implementation.

Designs such as a 64-point fast Fourier transform (FFT) design, an I^2C design, and a mixed-signal design are built to demonstrate power, area, performance advantages of the relative timing methodology and are used as a platform for developing the backend robustness. Results are verified by performing post-silicon timing validation and test. This work strengthens overall relative timed circuit flow, reliability, and testability.

This dissertation is dedicated to my wife and my parents.

"If I have seen further it is by standing on the shoulders of giants."

– Isaac Newton

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor Professor Stevens for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. He guided me during research and writing of this dissertation. I could not have imagined having a better advisor and mentor for my Ph.D study.

In addition to my advisor, I would like to thank the rest of my thesis committee: Professor Davis, Professor Brunvand, Professor Kalla, and Professor Roncken, for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives.

I thank my fellow lab mates, Vikas Vij, Jotham Manoranjan, Mac Wibbels, and Tannu Sharma for the stimulating discussions, the sleepless nights we were working together before deadlines, and for all the fun we have had in the last four years.

Last but not least, I would like to thank my family: my wife and my parents for supporting me spiritually throughout writing this dissertation and my life in general.

CHAPTER 1

INTRODUCTION

Scaling has enabled the transistor revolution, allowing the industry to embed as many as 4 billion transistors on a single chip. This increase in capabilities in semiconductor manufacturing has enabled the design of large, concurrent integrated circuits and systems. Computer-aided design (CAD) algorithms and tools enhance productivity and reliability of complex circuits. However, design methodology has not kept up with improvements in manufacturing. The clocked design flow is largely unchanged over the last 35 years and is targeted toward designs that operate at a single frequency.

Multiple frequency designs offer additional opportunities for optimization on power and performance avenues. Relative Timing (RT) is an asynchronous design methodology that empowers multiple frequency designs [1]. RT designs commonly demonstrate significant energy reduction and performance enhancement. Previous studies have show that several RT designs achieve 3× energy reduction [2]–[11]. Some representative designs are listed in Table 1.1. Clocked design forms the baseline. Numbers larger than 1.00 are improvements for all metrics. Pentium front end results are comparing the fabricated designs. The area reported for the 2-phase link design does not include transistor area. The mixed signal design will be described in Section 6.2.7.2. The 64-point FFT design will be discussed in Section 6.1.

RT design enforces circuit timing by constraining a circuit design with a set of timing constraints. The circuit is functional if, and only if, all the constraints are hold. This method for handling circuit timing requires more in-depth algorithmic support to validate the correctness of the circuit operation during the design phase. Postmanufacturing testing is one other essential step in the process needed to validate and observe the correctness of the designs. Reliability and correctness are two main qualities that must be assured if the technology is to be considered commercially ready. This dissertation develops the backend support for RT designs. Backend robustness is achieved by validating event sequencing of the design and implementing design for testability (DFT).

Table 1.1. ASIC Design Comparison Between Clocked and Relative Timed Designs

Design	Energy	Area	Freq.	Latency	Aggregate
Pentium F.E. [3]	2.05	*0.85*	2.92	2.38	12.11×
10-bit FIFO [4]	2.37	1.61	1.06	2.91	11.77×
2-phase Link [5]	1.11	0.92	*0.98*	1.77	1.77×
SAS [5],[6]	2.54	1.36	*1.00*	9.54	32.95×
Mixed Signal	3.25	1.02	*1.00*	*1.00*	3.32×
NoC - Aeth/Orion [7]	4.85	6.54	2.10	1.84	122.56×
NoC - COSI [8]	1.87	5.72	1.19	2.25	28.18
64-pt FFT [9]	2.43	2.42	1.97	3.17	36.72×
UART [10]	3.99	0.92	*1.00*	*1.00*	3.67×
OCP Socket [11]	4.65	1.36	2.75	–	17.39×

1.1 Background

1.1.1 Asynchronous Method

Asynchronous designs use handshake protocols to communicate between modules. There are two common types of handshake protocols: 2-phase and 4-phase. For this study, the 4-phase handshake protocol, which is shown in Figure 1.1, has been selected. Communication between modules starts with the sender raising the request signal. The receiver responds with an acknowledgment of the received signal. The sender then releases the request and the receiver returns to the initial state. Signal transitions are used for communication in the 2-phase handshake protocol. Circuit applies the 4-phase protocol generally have less overhead and are smaller than the one uses the 2-phase protocol [12], [13].

There are two common design styles for asynchronous circuits; bundled-data and delay-insensitive (DI) [14],[15]. The bundled-data style combines a standard combinational data path with a handshake mechanism to control data flow. The DI style encodes the handshake signals with the data to form 1-of-N encoding for communication [16]. In this study, bundled-data is the style chosen for asynchronous circuit design because of its superior power and performance over DI style, and due to the fact that timing verification is more complicated.

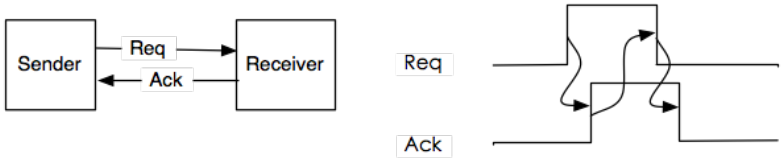

Figure 1.1. 4-Phase handshake signals

1.1.2 Relative Timing

Timing in circuit design is where the proverbial rubber hits the road. The effect of time on a system is to order and sequence events. Timing is where asynchronous designs differ from clocked designs.

This creates problems and challenges for circuit optimization and validation because commercial electronic design automation (EDA) tools only support clocked design. However, asynchronous design is still of high interest because it can provide significant power and performance benefits.

Circuit timing in asynchronous circuits can be modeled and observed by applying the relative timing methodology. Relative timing is a method of expressing the signal ordering property of time with the logical expression is shown in Eqn. 1.1. The relative timing constraints have a common timing reference point called the point of divergence (pod). The timing paths between the two points of convergence (poc_0 and poc_1) with respect to pod are constrained relative to each other to ensure signal ordering is guaranteed.

The maximum delay of the path from the pod to poc_0 must be less than the minimum delay of the path from the pod to poc_1 for signal ordering to hold. The margin m is added to provide for a more robust event separation.

$$\mathsf{pod} \mapsto \mathsf{poc}_0 + m \prec \mathsf{poc}_1 \tag{1.1}$$

Relative timing constraint paths can be directly translated into a set of synopsys delay constarint (SDC) `set_max_delay` and `set_min_delay` commands, as shown in Table 1.2. Note that the keyword `magin` below is a supportive command that refers to m in Eqn. 1.1.

1.1.3 Relative Timing Tool Flow [2]

Extending the application of relative timing to the entire end-to-end design and synthesis flow is addressed. There are four main phases of the basic design flow where additional steps are added to the traditional clocked flow.

1. The design and characterization of the relative timed design elements.

Table 1.2. SDC Constraint Translated from RT Constraint

set_max_delay $dpoc0 -from pod -to poc_0
set_min_delay $dpoc1 -from pod -to poc_1
#margin m -from pod -to poc_0 -from pod -to poc_1

2. Mapping of the RT constraints and timing values onto the physical architecture.

3. Performing timing closure on the timing targets supplied in the previous step.

4. Performing complete postlayout validation of the relative timing constraints.

Figure 1.2 shows the flow including necessary steps to seamlessly integrate with current commercial clocked CAD tools. The highlighted portions in Figure 1.2 are additions to the traditional clocked flow. This study directly addresses (iv) and addresses area (iv) by expanding it to include DFT with RT designs.

1.1.4 Timing Validation

Circuits will only perform correctly when all timing constraints are met. The purpose of timing validation for circuits is to guarantee the correctness of the circuit after the circuit. The typical approach for performing timing validation for physical design is implement-then-verify [17]. This approach requires multiple iterations of time-consuming tasks, including circuit synthesis followed by place and route. The validation flow consists of two inner iterations as shown in Figure 1.3. Automation of the timing validation is needed to optimize the circuit designs and reducing overall nonrecurring engineering cost on the iterations.

Commercial tools and flows use well-developed algorithms to automate timing validation for clocked systems. However, these algorithms only work with circuits that can be represented as a direct acyclic graph (DAG). Asynchronous circuits are naturally cyclic, thus, they are not supported by commercial tools. Although there is a handful of timing validation methods for asynchronous circuits that use timed protocol or time separation of events, such approaches require additional custom tools and modifications on the circuit representation [18],[19].

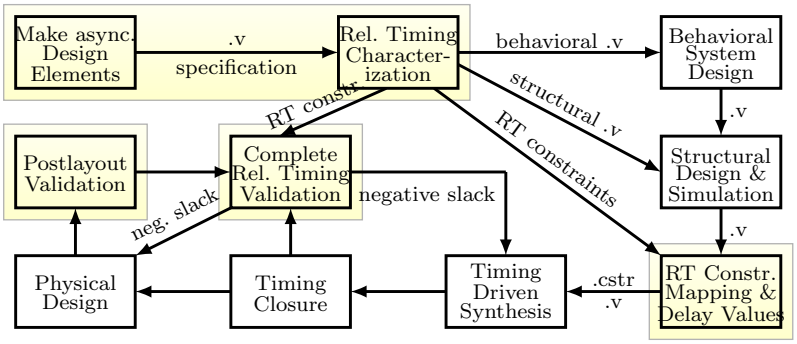

Figure 1.2. Simplified Relative Timing Design Flow

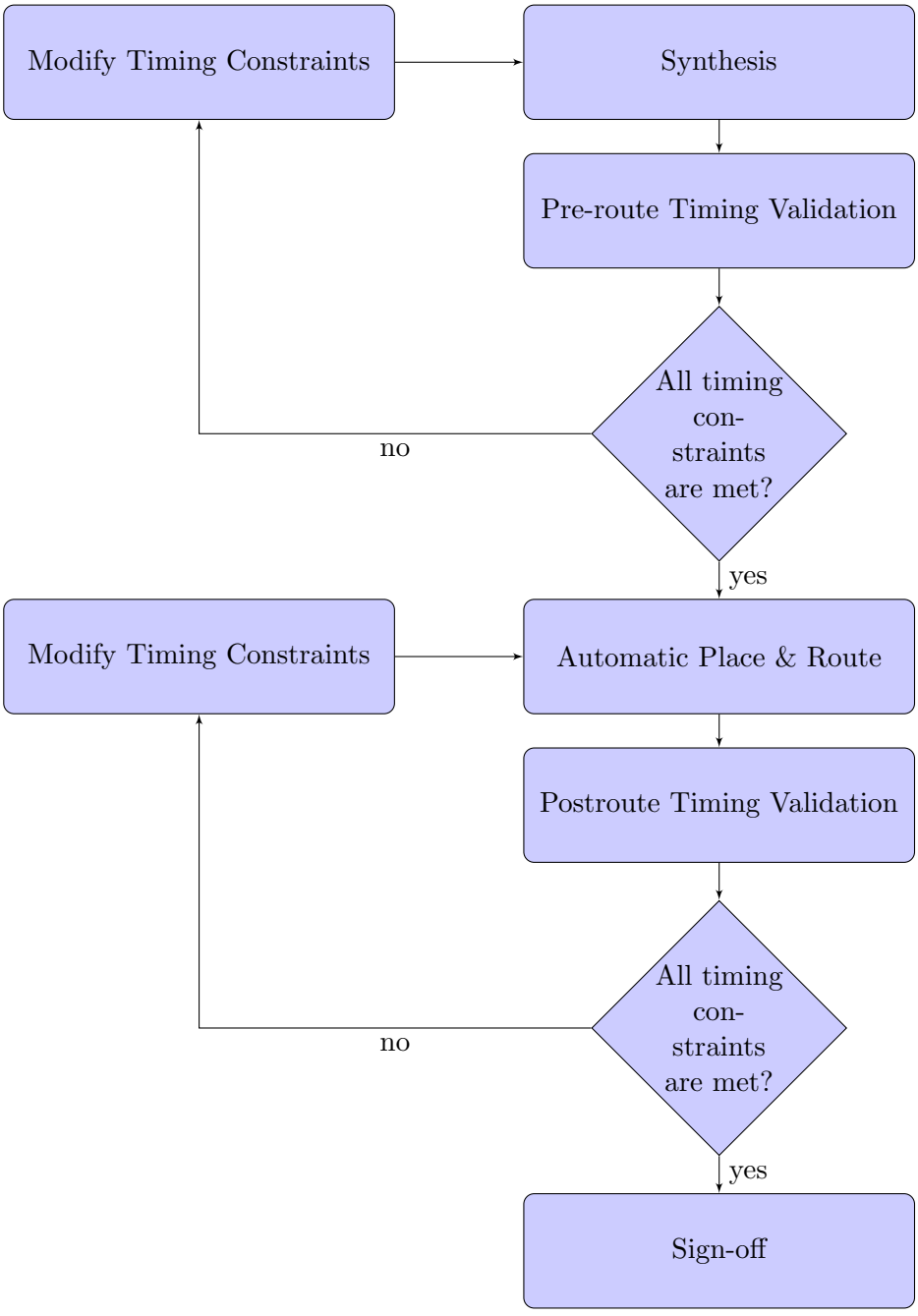

Figure 1.3. Timing Validation Flow

Timing validation with clocked designs uses static timing analysis (STA) to compute the critical path delays. The setup and hold time of register banks are verified using those critical path delays to compare against global clock signal [20]. Static timing analysis is capable of quickly reporting path delays in a system without feedback. Circuit behavior is not evaluated by using static timing analysis because it reports only the longest or shortest path delay between two timing endpoints. The false paths have to be excluded when applying static timing analysis [21].

An approach to validate timing of asynchronous blocks using STA has been reported for scalable delay-insensitive circuits [22]. A custom timing verification tool is built using STA algorithms. A statistical static timing analysis is reported to analyze asynchronous circuits models [23]. This approach requires custom timing models for each asynchronous component. A tool set to automates the generation of bundled data implementation has been developed [24]. The design methodology uses a desynchronization approach [25]. The tool set includes the use of primetime to analyze path delays. However, the tool only evaluates three specific timing constraints including setup time, hold time, and branch constraints. The tool is not capable of handling timing paths with cycles.

The timing validation for asynchronous circuits requires analyzing delays through cyclic paths. In addition, the tools and flows need to be general and support the richness of asynchronous designs. There are 137 protocols for the family of controllers that implement a 4-phase handshake with data valid on the rising edge of the request. There is also a myriad of different clocking schemes for each of the 137 protocols, including timing intense pulsed clock approaches as well as robust delay insensitive schemes [26].

Finally, there are also various data validity schemes that cooperate with 4-phase handshake protocols, including a scheme where the data is valid on the falling edge of the request signal. The protocol family where the data is valid on the rising edge of the request as well as on the falling edge of the request, along with a pulse clocked scheme for each is shown in Figure 1.4. A flexible and efficient RT validation mechanism, that utilized the commercial tools, is presented in this dissertation.

1.1.5 Design for Testability (DFT)

Manufacturing failures exists due to impurities or imperfect silicon crystalline structure, dust particles, nonplanarity of the wafer, etc [27]. DFT is necessary for developing and producing any IC in the market since it helps identify chips with manufacturing defects postfabrication and ensures the defective chips are discarded, or that workarounds are

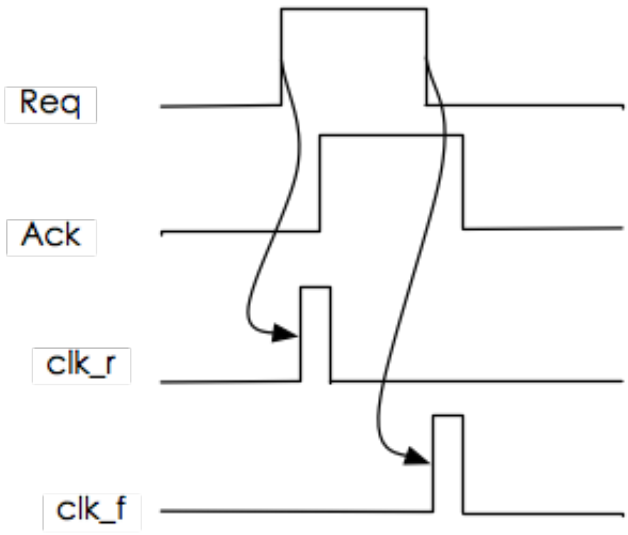

Figure 1.4. Pulse Clocking Schemes

devised to account for the defects. DFT enables circuit manufacturers to have the ability to test the IC by supplying a known state and to determine if the result is correct.

DFT also helps detect and diagnose the faulty ICs and save on overall chip cost and improve yield [28]. Finally, DFT is an essential component for better understanding the characteristic of manufactured ICs and providing a way to analyze and improve the product [29]. Including test structure in an IC allows designers and manufacturers to determine the reliability of the chip as well as debug the design after fabrication. Functional fault test and delay fault test are both required in this case since the relative timing methodology uses local clocks rather than a global clock signal. Typical clock systems can slow down the global clock to account for small delay variations caused by process variation. The RT circuit produces an invalid result if timing failures occur within any of the pipe stages. Namely, incorrect data will be stored if the data arrives later than the local clock signal.

There are several DFT approaches using scan design for various asynchronous circuits, including DI circuits, micropipelined, and self-timed circuits [30]–[34]. However, the approaches presented use in-house tools and require drastic modification in order to cooperate with other design approaches.

Similarly, automatic test pattern generation (ATPG) for asynchronous circuits has been implemented, but only applies to macro level asynchronous designs [35]. This ATPG approach did not scale to larger circuits or circuits with multiple asynchronous controllers.

MOUSETRAP, an asynchronous pipeline deisng, uses a D-latch as its state holding

element [36]. Thus, DFT can be performed on the control path by modifying the D-latch. However, the modification of D-Latch introduces 2.36× area and 3.45× power overhead. The addition of DFT to any designs will result in designs that consume more power and are larger and slower.

One approach is to attach additional fault detector circuitry onto the handshake control channels [37].

However, delay faults could not be detected with this approach. Another approach for detecting delay faults is developed [38]. A latch and muxes are inserted into the handshake control channels in order to control the handshake protocol.

This work targets DFT using scan design for the RT-based bundled-data circuits. These designs have a data path structure similar to those found in a clocked system so that commercial tools can be used to insert scan chains. The control network of the RT design remains unchanged and a functional test is applied. The control network achieves over 90% fault coverage with sequential test patterns.

1.1.6 Contribution of this Dissertation

This dissertation enables the ability to optimize and manufacture timed asynchronous circuit. Algorithms are developed and implemented to automate the analysis for circuit reliability and testability. Major contributions of this work are as follows:

- A methodology is presented that removes combinational cycles in the timing graph of an asynchronous design. This enables the ability to perform timing-driven synthesis and place and route using commercial EDA tools. The methodology includes a formal verification and a cycle cutting algorithm to correctly characterize asynchronous designs.

- Algorithms are developed to perform timing validation for asynchronous designs. The timing validation is capable of analyzing cyclic timing paths using static timing analysis from the commercial tools.

- A flow for design for testability (DFT) of asynchronous designs is presented. The overall test coverage of the asynchronous design is investigated using such flow.

- An algorithm specifies timing constraints for asynchronous designs to incorporate with commercial EDA tools is presented. Specifying design constraints on asynchronous designs is necessary for circuit correctness and performance and this is a manual step. The algorithm addresses these issues.

- Synchronous 16-point and 64-point fast Fourier transform (FFT) designs are implemented to compare with the asynchronous implementations. The asynchronous design is also used as the testbench for the timing validation and DFT flow.

- A clock gating technique for synchronous design is introduced that adopted relative timing methodology. A mixed-signal design and a digital design are implemented using this technique and show power saving benefit.

1.1.7 Thesis Organization

The thesis is organized as follows. Chapter 2 presents a method using a generic cycle cutting algorithm with true timing path to from a directed acyclic graph (DAG) that optimize timing driven synthesis and place and route. The DAGs generated are used for timing validation of true paths in Chapter 3. Chapter 4 presents the design for testing on the asynchronous circuits, including data path and control network. The automatic timing constraints mapping algorithm is introduced in Chapter 5. Chapter 6 shows several design applications that was used as test cases for Chapter 3, 4, and 5. In addition, a new design method for synchronous design is discussed in Chapter 6. Finally, a conclusion is drawn and future research works are discussed.

CHAPTER 2

TIMING PATH-DRIVEN CYCLE CUTTING FOR SEQUENTIAL CONTROLLERS[1]

Asynchronous handshake protocols and their associated controllers are used to implement the timing and sequencing of the design. The development of a complex asynchronous design can be simplified due to the modularity and composability of these controllers. All asynchronous circuits have cycles in their communication and timing graphs. Cycles in these graphs come from three primary sources. First, the handshake controllers themselves are often sequential controllers. The state memories in these sequential controllers are commonly implemented using combinational gates with feedback. Second, the basic nature of asynchronous handshake protocols produces cyclical feedback loops; the acknowledge signal creates a circuit cycle that responds to the request. This cyclical ring of request acknowledge logic gates produces an oscillator that dictates the operational frequency of each asynchronous pipeline stage. Thirdly, cycles are created in system level architectures where data is fed back to previous pipeline stages.

Many asynchronous design approaches leverage commercial electronic design automation (EDA) tools to optimize and validate power, performance, and timing correctness, and to perform timing driven optimizations for synthesis and place and route [40]–[45]. The timing driven algorithms in commercial EDA tools employ fast static timing analysis (STA) algorithms which require circuit models to be represented as directed acyclic graphs (DAGs). Unfortunately the sequential nature of asynchronous controllers and design approaches results in numerous topological feedback paths, presenting a fundamental challenge in employing commercial EDA tools.

---

[1]This section has been published in TODAES , 2016 [39]. http://doi.acm.org/10.1145/2893473. Reprinted, with permission, from William Lee, Vikas Vij, Kenneth S. Stevens, "Timing Path-Driving Cycle Cutting for Sequential Controllers," in ACM Transactions on Design Automation of Electronic Systems Volume 21, Issue 4, Article No. 64, June 2016

Sequential asynchronous circuits must be modeled with acyclic timing graphs to employ commercial EDA tools. This can be achieved with two fundamental elements which are directly supported by the commercial EDA flows in the Synopsys design constraints (sdc) format. First, circuit timing can be defined with a set of path-based timing constraints and their delay targets. Second, a set of timing cuts can be defined that model the native cyclic timing graph of the design as an acyclic graph without cutting the path-based timing constraints.

Following are three key observations for supporting cyclical circuits in the DAG-based commercial tools with the above two constraint sets.

1. If a cyclic circuit is given directly to a commercial EDA tool, a DAG representation will automatically be created without respect to timing paths. If a timing path is cut, it will not be employed in the timing driven algorithms of the commercial EDA tools. This is true for both optimization and validation. The cut timing paths are considered to be vacuously true and are not reported as failures since they can not be evaluated.

2. A DAG representation of a sequential circuit can not directly model all of the necessary timing of a sequential circuit. For instance, a handshake cycle in an asynchronous design can be implemented with a controllable ring oscillator that has a target frequency based on the function of the pipeline stage. A DAG-based model will cut the ring, and thus, one can not give a frequency based delay target to optimize the design or validate its performance. Thus, sequential timing constraints of an asynchronous design require two or more timing runs to validate full cyclical timing paths.

3. All design approaches that map sequential asynchronous designs to commercial EDA tools will require an additional methodology for cycle cutting, timing validation, and performance verification. The methodology and CAD tool reported here for creating DAGs are developed in such a way that it should be applicable to nearly all high level methodologies that address system level timing for asynchronous designs.

Providing a DAG that supports a timing model of a sequential circuit to commercial EDA tools results in several benefits. In some design approaches, such as bundled data design, a nonfunctional design results if timing constraints do not hold. This can occur if timing constraints are cut in creating the DAG. Likewise, commercial EDA can not be employed to evaluate circuit timing unless the graph is represented as a DAG and the timing paths used for evaluation are not cut. Providing a DAG with associated timing constraints enables the

commercial EDA tools to better optimize designs, resulting in significant improvement in the power, area, and performance of such designs.

The fundamental problem is addressed of creating a DAG timing model for sequential controllers and systems that does not cut the set of timing paths used for design synthesis, optimization, and validation. This is the first published work to do so. path-based timing constraints are identified by timing endpoints, just as is done using sdc constraints in the commercial EDA tools. This approach thus simplifies the translation of the timing directives to the commercial EDA tools. An algorithm is implemented that creates cycle cuts that preserve the path-based timing constraints. This work also introduces the concept of identifying paths that must be cut in a design in order to remove cycles formed by netlist connectivity external to the controller, along with its associated algorithm. This algorithm is also used to remove false paths in the design. The tool uses vectorless graph traversal algorithms similar to commercial static timing analysis approaches. It assumes that the design has been technology mapped to specific gate implementations, and inputs the Verilog used in the design. The CAD algorithm is intended to be applied to single asynchronous sequential controllers making them the focal point for timing paths and system level cycle cuts. The combination of vectorless algorithms and controller-based focus makes this CAD tool applicable to most if not all asynchronous design methodologies and styles. The tool writes out sdc constraints that are directly supported by the commercial EDA tools. The tool reports coverage and the quality of the results of the cycle cutting algorithm.

A timed burst-mode protocol along with its circuit realization is employed as an example for this paper. The CAD algorithms in this paper are applied to several examples and compared against the commercial EDA tool. Results for applying the algorithm to 131 separate asynchronous sequential control circuits and to eight benchmark designs are reported. The comparison of these designs is made with respect to forward latency, backward latency, cycle time, area, power and energy per token. The results are also analyzed for quality by ensuring that the cycle cuts produce a DAG, false paths have all been cut, and the number of gates that have no timing path passing through them. Having at least one timing path passing through each gate in a design results in the gates being power and performance optimized based on the timing path constraints.

2.1 Related Work

Combinational cycles are generally associated with sequential circuit designs like asynchronous circuits. Cycles can also be present in combinational logic, and some cyclic com-

binational circuits have been shown to substantially reduce area [46]. Since algorithms in EDA tools require acyclic timing graphs, the problem of finding cycles and analyzing the combinational nature of circuits with cycles has been investigated [47]. Algorithms that generate an equivalent acyclic combinational circuit which reproduces all the combinational behavior of the original cyclic circuit have been developed [48]–[50]. These approaches can not be applied to sequential circuits because they change the sequential behavior when state-holding feedback of a circuit are removed. In order to support general sequential circuits built as combinational logic with feedback, the cyclic circuit must be represented as a DAG without modifying its structure or behavior.

The work that is most closely related applies cycle cutting to the testing of digital circuits with feedback [51]. This is formulated as a covering problem where the sets of paths form the cycles, the solution is to find the minimal number of paths that cut all the cycles. The drawback of this approach is similar to the algorithms in current commercial CAD tools which also cut cycles. A set of cycle cuts, even if they are minimal, will create a DAG, but timing driven optimizations can not be performed because timing paths are cut.

A core function of the algorithms in this work is to identify cycles and paths in a circuit graph. Reconvergent paths and circuit cycles are particularly problematic. Indeed, it has been shown that a circuit can have an exponential number of paths based on the number of gates. This path explosion has proven to be particularly challenging for the delay fault testing community. While path explosion is problematic in some domains, it has not been demonstrated to be a problem in this application. The sequential modules being evaluated have normally been designed to minimize hazards that can often be a byproduct of reconvergent paths. Our algorithms are applied to single sequential controllers which contain fewer than 100 gates. We have applied our tool to the largest published sequential controller designs which tax the limits of what can be synthesized. For all but one circuit, the run times are less than one second in the exhaustive search mode. Pipeline controllers used in nearly all commercial and academic design are more closely represented by the 131 controller set used in our example set.

2.2 Background

2.2.1 Circuit Representation

The circuit is represented as a directed graph $G = (V, E)$ where each gate, primary input, and primary output $v_i \in V$ is a vertex (node) of the graph, and edges $e_i = (v_x, v_y) \in E$ map primary inputs and gate outputs to primary outputs and gate inputs of the design.

We represent a path as a sequence of vertices (nodes) that are connected by edges in G. A cycle is a path that starts and ends with the same vertex.

The path $[lc1, lc2, lc1]$ is a cycle in the Figure 2.1 circuit. All noncyclic paths employed in this work are simple paths where repetitions of vertices and edges are not allowed. Each edge in the path can be directly derived; this cycle contains the edge set $\{(lc1, lc2), (lc2, lc1)\}$.

Four inputs are provided to this tool. A sequential circuit specified in structural Verilog that has been technology mapped to a set of library gates, a data structure that defines the input and output pins for each gate in the library, a set of *timing endpoints* $p_i = (v_x, v_y) \in \Theta$ which preserve the *greatest common path* (or GCP) uncut between v_x and v_y, and a set of timing endpoints $t_i = (v_x, v_y) \in \Phi$ that cuts all paths between v_x and v_y that are not GCP paths defined by Θ. In this paper we represent timing endpoints in Θ and Φ as a pair of nodes (A, B) or the relationship pair $(A \rightarrow B)$. The set Θ defines timing paths that must be preserved, and the set Φ defines the *cut paths*.

Each primitive gate in the cell library will have one or more input and one or more output. Each input for gate v_y will be associated with one or more edge $(v_x, v_y) \in E$; each output for gate v_x will be associated to one or more edge $(v_x, v_y) \in E$.

The algorithms that perform timing driven optimization in the EDA tools employ vectorless static timing analysis (STA). This is accomplished by employing the structural netlist, largely ignoring the functional behavior of the circuit. The simple directed graph G is sufficient to represent the timing graph of a circuit for such purposes. The algorithms developed in this thesis also use only the structural netlist, identifying paths using graph

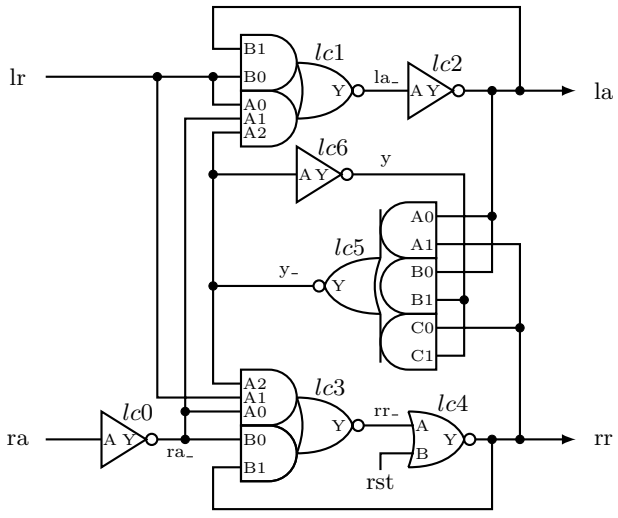

Figure 2.1. LC Circuit Implementation

traversal algorithms.

2.2.2 Greatest Common Path Between Timing Endpoints

A GCP is a minimal structural path between a pair of timing endpoints. If the node sequence in a shorter path is contained in a longer path between the same endpoints, then the longer path is not a GCP. For example, there are two simple paths that do not contain cycles which connect the timing endpoint lr and rr in Figure 2.1: $[lr, lc1, lc2, lc5, lc3, lc4, rr]$ and $[lr, lc3, lc4, rr]$. The longer path is not a GCP because it is not minimal; it contains the shorter path. Note that paths that contain cycles will not be GCPs.

2.2.3 Path Identification from Timing Endpoints

Our specification approach does not identify or enumerate individual paths through a circuit; rather we employ timing endpoints to correctly identify both the true and false paths in a circuit. *The specification employed in this tool identifies all shortest unique paths (GCPs) between timing endpoints in* Θ. These are identified as true paths in the design. Conversely, *all* simple structural paths between the timing endpoints are identified by the timing endpoints in Φ. This set is used to identify false paths.

2.2.4 Cutting the Timing Graph

The liberty timing file that is used by the commercial EDA tools defines the path delay from the inputs to outputs of the primitive gates. For example, the NOR gate in Figure 2.1 has a timing path from pin A to pin Y and from pin B to pin Y. These liberty gate level timing paths can be cut with the sdc command set_disable_timing. This is the mechanism we employ for cutting timing paths and cycles in a circuit, and the tool outputs the results in the sdc format.

Assume we want to cut the Figure 2.1 cycle $[lc3, lc4, lc3]$. This can be accomplished by cutting either of the two edges $(lc3, lc4)$ or $(lc4, lc3)$. Removing edge $(lc3, lc4)$ is implemented by disabling the liberty timing edge (A, Y) in gate $lc4$. We write this out as the sdc constraint `set_disable_timing -from A -to Y` for the $lc4$ gate in the design. Using this mechanism, our tool can create a timing graph DAG that is supported by commercial EDA.

Edge $(lc4, lc3)$ is the preferable arc to cut in the above example to remove the cycle from the timing graph. If edge $(lc3, lc4)$ is cut, the circuit has no timing path from the primary inputs lr and ra to the primary output rr.

2.2.5 True and False Path Specification

Many structural paths in a design will be false paths that are not behaviorally sensitizable. True paths are the result of the logical sequential behavior of the circuit and how it responds to changes in the primary inputs and internal state. Since both STA and the algorithms employed in this tool are structural, a mechanism must be employed to specify the true and false paths in a design. True paths must be preserved, and false paths in a circuit must be cut. Otherwise timing results will be incorrect and the quality of the timing optimizations performed by the commercial EDA tools will be significantly degraded.

True paths in a design are identified by the timing endpoints in Θ. False paths are identified by the timing endpoints in Φ.

Often the true timing paths can be directly identified as a GCP. Assume we need to specify that the timing path between lr and rr is to remain uncut because a timing path passes through those nodes. Two simple paths exist between these vertices in Figure 2.1: $[lr, lc1, lc2, lc5, lc3, lc4, rr]$ and $[lr, lc3, lc4, rr]$. In this case, the GCP identifies the shorter true path, thus placing the timing endpoints (lr, rr) into Θ correctly identifies the true paths of the circuit. (The longer path is false because it can not behaviorally occur in this design. Lowering rr is only sensitized by ra. For rr to rise, $ra\_$ and $y\_$ must be asserted and lr must rise. For the longer path to occur, $y\_$ would need to rise, which only occurs when la falls.)

A GCP can identify multiple paths. For instance, timing endpoints $(lr, lc5)$ identify two paths as part of the GCP: $[lc1, lc2, lc5]$ and $[lc3, lc4, lc5]$. In this case, both of these paths are true paths through the circuit.

In some designs, the true path is not identified as a GCP. Assume that the longer path from lr to rr through the circuit, $[lr, lc1, lc2, lc5, lc3, lc4, rr]$, is the true path which we want to identify. The two timing endpoints $(lr{\rightarrow}rr)$ will not correctly identify this path since its GCP will cut the longer path and preserve the shorter path $[lr, lc3, lc4, rr]$. Non-GCP paths can be identified as true paths by creating sets of timing endpoints whose transitive closure covers the full true path. By selecting $lc2$ as an intermediate timing endpoint, two sets of timing endpoints $\{(lr{\rightarrow}lc2), (lc2{\rightarrow}rr)\}$ are created to identify the longer true path and ensure it remains uncut. The GCP for $(lr, lc2)$ is $[lr, lc1, lc2]$ and the GCP for $(lc2, rr)$ is $[lc2, lc5, lc3, lc4, rr]$. Together these two paths ensure that the longer true path from $lr{\rightarrow}lc2$ is identified as true.

Simply identifying true paths through a design is not sufficient to ensure correct timing evaluation of the design. The false paths must also be identified and then cut. Assume that

both simple paths between timing endpoints lr and rr remain uncut. Also assume for a moment that the delay of each gate is roughly identical. When a minimum delay analysis is performed on these endpoints, the shorter path will be selected since the latency will be two gate delays. When a maximum delay analysis is performed, the longer path will be employed since it has five gate delays. If the longer (shorter) path is false, it must be cut to ensure correct timing evaluation of the design. Thus, every simple path between timing endpoints that is not a true path of a design must be cut.

When the true path is modeled as a GCP, it will be identified by placing the timing endpoints into set Θ. All false paths are identified by placing the same timing endpoints into set Φ iff the GCPs identified in Θ are not cut by paths identified in Φ. If the true timing path requires multiple GCPs to be identified, then each of these pairs will be placed in Θ. However, only the single pair that is the timing endpoints of the transitive closure of these paths is included in Φ to identify false paths. Thus, if the true path of $lr \rightarrow rr$ is the longer path, the set $\{(lr, lc2)(lc2, rr)\}$ will be added to Θ but only the single pair (lr, rr) will be added to Φ. The GCPs in Θ will ensure the longer path remains uncut, but the shorter path will be cut as it is a path identified in Φ.

The set of timing endpoints that correctly identify the true and false timing paths through the circuit is provided as an input to this algorithm because this tool is intended to support many circuit families and design methodologies. The generation of the timing paths is dependent on the circuit family and high level timing methodology employed.

2.2.6 Classification of Cycles

Handshake controllers and asynchronous sequential state machines are the primary source and convergence locations of combinational cycles in asynchronous architectures. Thus, high-level timing and optimization methodologies can eliminate most, if not all, cycles by pruning paths in the controllers. This work refers to two classes of combinational cycles based on their relationships to the controller under analysis, expressed below. This work supports cutting both classes of cycles.

1. *Local Cycles:* Cycles which can be identified by examining the structural netlist of the controller under analysis. Two of the eight local cycles of this circuit are $[lc1, lc2, lc1]$ and $[lc3, lc4, lc3]$, as shown in Figure 2.2.

2. *External Cycles:* Cycles which pass through the controller, but can not be identified from the structural netlist of the controller. The leftmost external cycle in Figure 2.3 is $[LC_0/rr, LC_1/lr, LC_1/la, LC_0/ra, LC_0/rr]$.

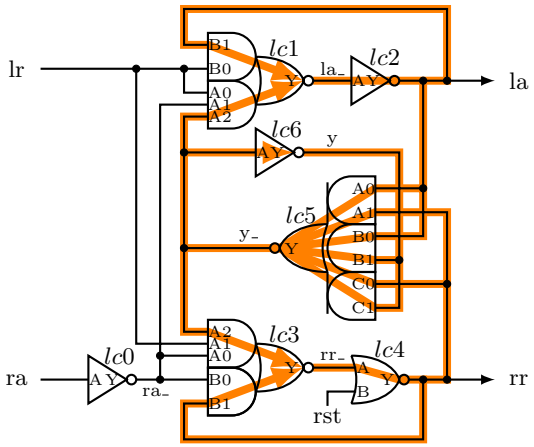

Figure 2.2. LC Circuit with the Eight Local Cycles Highlighted

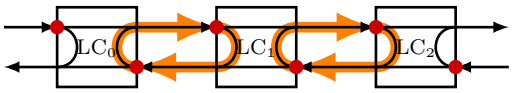

Figure 2.3. External Cycles Formed from Handshake Channels are Highlighted. Additional Unmarked Cycles Exist, such as $[LC_0/rr, LC_2/lr, LC_2/la, LC_0/ra, LC_0/rr]$.

Figure 2.2 highlights the eight local cycles to the burst-mode controller. As drawn, three cycles converge pass through $lc1$, three through $lc3$, and two cycles through $lc6$. Figure 2.3 highlights external cycles that are formed when a handshake controller is connected in a linear pipeline. A common set of paths, based on the design and timing methodology, can be identified which will remove the cycles created from the handshake channels. Cutting all paths between the $(ra \twoheadrightarrow rr)$ endpoints will cut the handshake channel cycle. Such an approach allows a single cut path constraint local to the pipeline controller to remove all handshake channel cycles in a design[2].

Therefore, this algorithm will accept timing endpoints to a controller that will cut external cycles. These cuts are identified by placing the timing endpoints into the set Φ.

Three observations can be made based on cycles external to the controllers being evaluated. First, placing a path that should be cut due to an external cycle into Φ does not *guarantee* the path will be cut. If there exists a path between the timing endpoints that is fully covered by GCPs in Θ, the path will not be fully cut. This will result in combinational cycles which remain in the timing graph of an architecture. Specifically, this condition will be reported as an error.

[2]The location and expression of these cycles are dependent on the high level design methodology employed.

Second, each handshake controller will have its cycle cut values generated independently. Therefore, if multiple different controllers are used in a single design, and they employ a different handshake cut methodology, then cycles can remain in an architecture. (For instance, assume one controller cuts the handshake cycle with the (lr, la) endpoints and another with the (ra, rr) timing endpoints. If both are used in a design, the system can have cycles left uncut.) Thus, the application of this work is dependent on correctly conforming to the system-level methodology employed, and ensuring that it is applied uniformly to the control modules.

Finally, while this work helps support different circuit and timing validation methodologies, it remains dependent on the high level models employed. External cycles created by data path feedback may not be directly supported by adding a cut path in Φ for all timed asynchronous circuit and timing methodologies.

2.3 Key Contributions

A tool which performs cycle cutting that preserves timing paths was introduced and implemented [2]. The implemented tool is capable of producing a DAG of a circuit once a set of GCPs is provided. The primary contribution of this section is forming an algorithm that integrates with the implemented cycle cutting tool. First, all true paths within the sequential module are identified. Second, the false paths are generated with respect to the true paths. Then, both the true paths and false paths are integrated with the developed cycle cutting tool. The provided true path information guides timing driven synthesis and place and route on circuit optimization. In addition, all false timing paths are removed from the timing graph to improve timing correctness and runtime of the commercial tools. A comparison of implemented tool with and without the true and false timing paths approach is studied.

2.4 Evaluation Approach

This flow is evaluated using a micropipeline approach [14]. A micropipeline consists of a traditional boolean logic data path and an asynchronous control path. The data path contains acyclic combinational logic (CL) and registers (L). The control path consists of linear control blocks (LC), which perform via handshaking the role of the clock. Handshake clocking generates the appropriate timing and sequencing for the design. It is elastic in nature and can stall if required. The minimum latency through the control logic plus a margin must be greater than the maximum delay of the combinational logic in order to

fulfill the setup and hold time constraints at the register bank. Thus delay elements may be required between LC blocks.

The C-element used in the Sutherland micropipeline is replaced with each controller in the full family of four-cycle handshake controllers with data valid on the rising request [52],[53]. To simplify the evaluation and to focus on the controllers, the data path of the micropipeline has been removed. Since there is no data path, this implements a token FIFO where the handshake controllers are allowed to operate at maximum frequency. The FIFOs are pipelines of a depth of four. The simulations employ controllable interfaces on the input and output of the pipeline that operate faster than the response time of the controller under test.

A set of relative timing constraints specific to each controller are generated that must hold for the circuit to perform hazard free. These are mapped onto each controller as a set of path-based constraints that are provided to the tool in the true path set Θ and the cut path set Φ. Three additional path constraints are added to reduce the cycle time of the controller: a constraint from lr of the current controller to lr of the downstream controller, lr of the downstream controller to ra of the current controller, and ra of the downstream controller to ra of the current controller. These endpoints are identified as red dots and black arrows in Figure 2.3. Those endpoints are mapped onto the local controller constraints $\{(lr, rr)(lr, la)(ra, la)\}$ that are passed to the tool in Θ and Φ (adjusting timing endpoints to identify true paths in each of the specific controllers). The handshake channel cycles are cut by adding timing endpoint (ra, rr) to the cut set Φ.

The sdc constraints generated by the tool are employed in synthesis and simulation, where delays are attached to each of the three high-level timing paths. The delays are customized to each specific controller design based on its response time such that negative slack does not occur.

The pipelines are synthesized with a commercial CAD tool employing optimization for power and performance using the Artisan 65nm library. This tool automatically cuts cycles in the design based on a minimum cut algorithm which does not respect timing paths provided to the design. The designs are then placed and routed with SoC Encounter to determine layout area and parasitics. The numbers for forward latency, backward latency, and cycle time are generated by simulating the post routed design using sdf (standard delay format) back annotation. The designs are tested for 50 handshake cycles to generate a vcd (value change dump) file that reports node activity. The vcd file is used with PrimeTime to generate power and simulation time numbers on the post APR (automatic place and

routed) design using back annotated parasitics extracted from the layout.

2.5 Preliminary Results

2.5.1 Benefits of Correct Cycle Cutting

Path-based timing driven optimizations are not able to be performed if timing paths through the circuit have been cut. The LC circuit in Figure 2.1 is used as a preliminary example. Optimization results are significantly degraded when commercial EDA tools cut cycles without respecting timing paths because the tools must perform untimed circuit optimization. We show the power, area, and performance benefits of applying timing driven optimizations on an asynchronous handshake controller when timing paths are ensured to remain uncut through the application of the algorithms in this paper. However, performance, power, and area degradation is a secondary problem. When true timing paths are cut, the circuit can not be optimized nor evaluated for that path. If this is a path that is required for correct operation, the circuit may fail.

The three timing constraints used for the micropipeline performance analysis described in Section 2.4 are employed. Verification identifies additional constraints that must hold in this design for correct operation. The relative timing constraint $lr\uparrow \mapsto y\uparrow \prec rr\downarrow$ states that y must rise before rr falls. A correctness timing endpoints (lr, y) is also included for the design. These four timing endpoints are added to both Θ and Φ. The external cycle cut constraint (ra, rr) is added to Φ. The tool is passed these constraints and the Verilog design. A set of sdc constraints are generated which create a DAG. All false paths and cycles are cut, and at least one timing path specified in Θ passes through every gate in the design. The design has 10 true paths identified as GCPs, cuts 8 cycles, and 14 false paths or external cycles. This is accomplished with 8 timing graph cuts.

The design is evaluated under the following four scenarios. This demonstrates the importance of supplying paths from Θ that can not be cut as well as paths from Φ that must be cut to create a DAG.

- **No Constraints:** Commercial CAD tool cuts all cycles in the design.
- **Local Cut Constraints:** The local cycles are cut using the timing constraint path driven algorithms in this paper using Θ, but the commercial CAD tool creates external cycle cuts.
- **External Cut Constraints:** Only the external cut path endpoint pair is provided. The commercial CAD tool creates the local cycle cuts.

- **Full Constraints:** The algorithms in this paper are given the full Θ and Φ endpoint sets and performs all cycle cutting. No paths are cut by the commercial EDA tool.

There is a substantial improvement in circuit quality obtained by employing cuts from the timing path constrained algorithms in this paper when compared to cuts automatically generated by the commercial CAD tool. Improvements for this circuit include 1.3× for cycle time, 2.5× for area, and 2.7× for energy per token which are shown in Table 2.1. Moreover, the cuts generated by the commercial tool remove two true timing paths. The table also points out the importance of including both the performance / correctness constraints and external cycle cut constraint paths. If only the external cycle cut path constraints are included, the results are generally worse than having the commercial CAD tool perform all cycle cutting. Simply employing the performance and correctness constraint paths helps, as this reduces energy by 1.5× over the commercial CAD tool. However, there still remains a penalty of 1.8× in energy over our algorithm if one allows the commercial CAD tool to perform external cycle cutting.

2.5.2 Generality of Approach

This tool can be used with any design methodology and applied to any controller design. This is illustrated by applying this to a well known quasi delay-insensitive (QDI) controller. A weak-condition half-buffer (WCHB) design is implemented [15]. This design, as shown in

Table 2.1. Comparison of Performance Metrics Using Timing Path Cycle Cutting (TPCC) Versus a Commercial EDA Tool Algorithm

	No TPCC Constraints	External Cut Constraints	Local Cut Constraints	All TPCC Constraints
Forward Latency (ps)	97.5	127.5	85.0	107.5
Backward Latency (ps)	327.5	347.5	305.0	232.5
Cycle Time (ps)	520.0	540.0	460.0	390.0
Area (um^2)	361.8	384.0	236.6	145.7
Power (mW)	2.29	2.30	1.72	1.01
Simulation Time (nS)	32.5	34.2	29.3	27.8
Energy/token (fJ)	374	395	253	141
No. of Cut Timing Paths	2	2	0	0

Figure 2.4, has been mapped to the same cell library and characterization flow described in the previous section.

Timing endpoints $\{(I_0, O_0),\ (I_1, O_1),\ (I_0, I_{ack}),\ (I_1, I_{ack})\}$ are included in the constraint sets Θ and Φ for this circuit. Two cut path endpoints $\{(O_{ack}, O_0), (O_{ack}, O_1)\}$ are used to remove cycles in the handshake channel are added to Φ. The cycle cuts are generated by specifying timing cut points. The cut points are highlighted in Figure 2.4.

Postlayout results for a four-deep pipeline are generated. Table 2.2 shows that using both local and external cycle cut constraints clearly results in the correct design without any true timing paths removed. This timing optimized implementation has nearly a 2× improvement in forward and backward latency and cycle time, a 1.4× area advantage, and a 3.2× energy per token advantage.

2.6 Rules for Timing Path Driven Cycle Cutting

The developed tool will create a DAG where true paths remain uncut and false paths and external cycles are cut when a correct set of timing paths are provided. If the set of paths are inconsistent and a DAG can not be generated, or external cycle paths can not be cut, an error is raised.

Paths are identified by a pair of timing endpoints. The timing paths consist of (a) a set Θ of timing endpoints that identify the true paths where the paths between the endpoints are GCPs, and (b) a set Φ of cut-paths which cut all paths between the timing endpoints that are not GCPs in Θ. Three rules are established to specify the overall cycle cutting algorithm which integrates the developed tool.

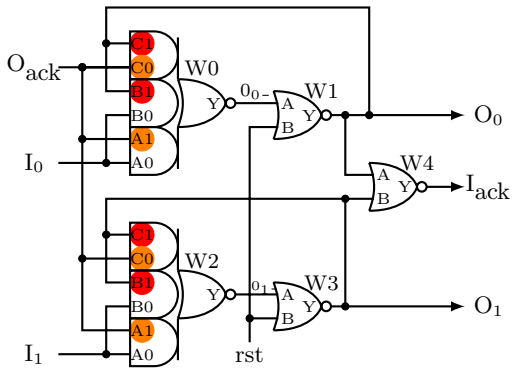

Figure 2.4. WCHB Circuit, W0 and W2 are C-Element Implementations. Red and Orange Circles Denote Locally and Externally Cut Timing Arcs.

Table 2.2. Comparison Using Timing Path Cycle Cutting (TPCC) Versus the Algorithm in a Commercial CAD Tool for WCHB

	No TPCC Constraints	External Cut Constraints	Local Cut Constraints	All TPCC Constraints
Forward Latency (ps)	162.5	160.0	152.5	82.5
Backward Latency (ps)	272.5	270.0	252.5	145.0
Cycle Time (ps)	510.0	520.0	460.0	270.0
Area (μm^2)	1269.5	1214.3	1232.7	890.6
Power (μW)	717	607	646	344
SimTime (nS)	53.8	54.3	52.5	34.5
Energy/token (fJ)	770	659	678	237
No. of Cut Timing Paths	2	2	2	0

2.6.1 Specifying True Paths and Ensuring Timing Arc Fidelity

The greatest common path, or **GCP**, is used to represent true paths in sequential circuits. A GCP is a minimal structural path between a pair of timing endpoints. If the node sequence in the shorter path is contained in a longer path between the same endpoints, then the longer path is not a GCP.

A GCP will not contain any cycles. Assume the following two paths between timing endpoints A and C: $[A, B, C]$ and $[A, B, E, B, C]$. Path $[A, B, E, B, C]$ can not be the GCP since shorter path $[A, B, C]$ exists.

- **Rule 1** All timing paths that are GCPs between timing endpoints in set Θ are considered true paths and must remain uncut.

If at least one structural path exists between pairs of timing endpoints $\forall(A, B) \in \Theta$, then there exists at least one GCP between each pair of timing endpoints. If all timing paths covered by a GCP can not be cut, then at least one timing path will be preserved for each pair of timing endpoints. The GCP(s) defines the true timing path(s) for a pair of timing endpoints. This rule takes precedence over the other rules.

2.6.2 False Path and External Cycle Removal

Only the true timing path(s) can be left uncut to correctly perform timing driven optimization. For example, a true path $[lr, lc3, lc4, rr]$ and a false path $[lr, lc1, lc2, lc5, lc3, lc4, rr]$ exist between timing endpoints $lr{\rightarrow}rr$ in Figure 2.1.

If both paths remain uncut, maximum delay calculations from ($lr{\rightarrow}rr$) will use the longer false path. The delay calculation through the false path will be substantially larger than the true path, resulting in timing optimization performing on the false path.

- **Rule 2** All paths between timing endpoints in the cut path set Φ which are not GCPs from set Θ must be cut.

This rule will require that all paths are identified between the timing endpoint pairs. All these paths will be cut, unless they are true paths which have been identified as GCP paths from the paths in set Θ.

All false paths can be removed by placing the first and last node for every true timing path into the cut path set Φ. This means that the true timing paths are also being identified as cut paths that will not be cut. Such paths are readily identified and are not counted in the quality metrics of cut paths that remain uncut. *Thus this algorithm requires that the initial and final endpoints of every timing path be included in* Φ.

For example, the true timing path from lr to rr in the controller of Figure 2.5 must be identified with the two timing endpoint pairs ($lr \rightarrow 3$) and ($3 \rightarrow rr$). Each of these endpoints identifies a single path, which is the GCP. No false paths are identified individually by these two endpoint pairs. However, there are multiple false paths between lr and rr which must be cut. By including ($lr \rightarrow rr$) in Φ the true path remains uncut, and all false paths are cut. The complete sets of constraints used to identify the true timing paths through the controller in Figure 2.5 are $\Theta = \{(lr, 0), (0, la), (lr, 3), (3, rr)(ra, la)\}$ and $\Phi = \{(lr, la), (lr, rr), (ra, la), (ra, rr)\}$. Under these constraints, all cycles are cut, there are no uncut false paths.

2.6.3 Creating an Acyclic Timing Graph

Timing driven optimization algorithms in commercial EDA operate on DAGs. The following rule will result in an acyclic graph.

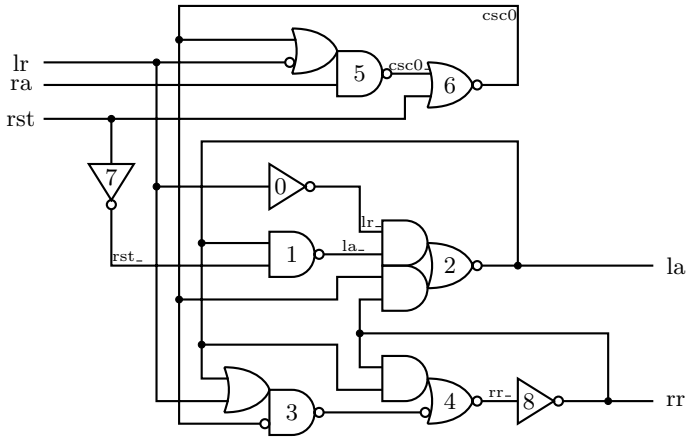

Figure 2.5. Handshake Controller L222 ◦ R2242 Synthesized with Petrify.

- **Rule 3** All cycles in the controller must be cut.

Algorithms in this tool automatically cut all cycles in the circuit while preserving the true timing paths specified in Θ.

2.6.4 Additional Information

2.6.4.1 Full Design Module Optimization

Best circuit optimization occurs with timing driven design, as shown in Section 2.5. Ideally every gate will have a timing path passing through it to define the intended delay bounds. An **orphan** is defined as a gate that does not have a timing path passing through it as defined by Rule 1. The optimization of handshake controllers will require multiple timing paths through the design to minimize or eliminates orphans and to create timing paths between primary inputs and outputs of a sequential controller.

2.6.4.2 Consistency and Rule Correctness

If set Θ consists of a single pair of timing endpoints, then rules 1–3 are necessary and sufficient to create a DAG with only true timing paths passing through the design for optimization. Unfortunately, these rules can not all be guaranteed to hold when more than one pair of timing endpoints exists in Θ.

If true paths cover a cycle, rule 3 will fail, permitting the cycle to remain uncut. For example, the controller in Figure 2.6 contains 18 cycles, 26 cut paths, and three GCP paths. The true path for timing endpoints (lr, rr) is $[lr, 9, 10, 1, 2, 3, 7, 5, 6, 0, 1, 9, 10, 7, 4, rr]$ (which contains a cycle) while the true path for (lr, la) is $[lr, 9, 10, 1, 2, la]$. A set $\Theta = \{(lr, 10), (10, 1), (1, la), (3, 5), (5, 0), (0, 10), (10, rr)\}$ is provided to represent the true paths. The algorithm reports that three cycles remain uncut including $[0, 1, 2, 3, 7, 5, 6, 0]$, $[0, 1, 2, 3, 7, 5, 6, 0]$, and $[1, 9, 10, 1]$. This is due to GCPs that overlap to cover all 3 cycles in the true path.

Likewise, a false path in a design may be covered by a set of GCPs. Therefore, any uncut path between timing endpoints in Φ which are not true paths are reported as an error by the tool.

2.6.4.3 Providing Correct Endpoint Sets

An internal CCS formal verification engine, `Blacart`, is used to extract the true timing path from all the sequential controller. The true timing path is gathered using a path searching algorithm. The verification engine starts searching, creating a causal string of events fired from one timing end point. The search continues similarly from the event of the

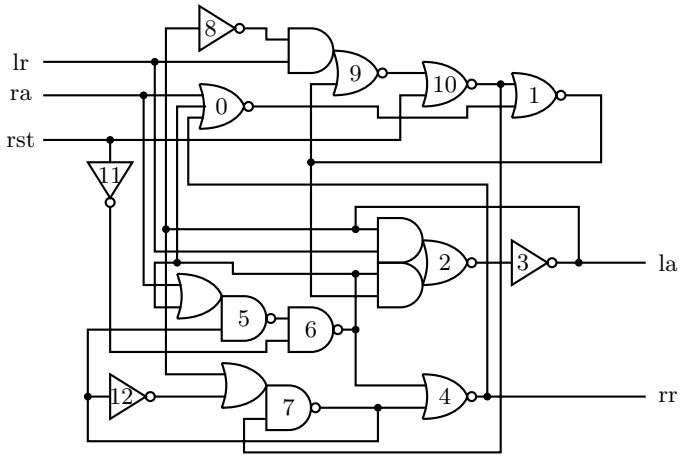

Figure 2.6. Handshake Controller L400 ◦ R0000 Synthesized with Petrify.

other timing endpoint until a common causal ancestor is found. This other timing endpoint becomes the point of divergence. A complete true timing path is reported for each of the two paths. The true path includes a sequence of internal nodes. Multiple true paths are reported if there exist multiple true paths between two timing end points. Table 2.3 presents all true timing paths associated with the sequential controller, which is shown in Figure 2.2. These true timing paths are required to be represented as GCPs for the cycle cutting algorithm. The conversion from the full path representation to the GCP representation is performed manually and listed in Table 2.3. Additionally, single GCP can express multiple true paths. For example, the GCP (lr $\mapsto$ y) expresses both true paths, $(lr, lc1, lc2, lc5, lc6, y)$ and $(lr, lc3, lc4, lc5, lc6, y)$.

2.7 Developed Cycle Cutting Algorithm

This section describes the developed algorithm which automates the process of generating cycle cuts for sequential circuit modules [2]. The structural Verilog circuit description is parsed and stored as an adjacency list $G = (V, E)$, where V is the list of vertices and E the edges of the circuit. Figure 2.7 demonstrates a graph representation of the Figure 2.1 circuit. All cycles present in the circuit module are identified. Based on the timing endpoints specified, a set of cycle cut constraints in the `sdc` format are output as well as a list of violations to the rules. These constraints can then be passed through the synthesis and place and route flows to allow the circuits to be automatically power and performance optimized by commercial EDA tools and then validated for timing correctness.

Table 2.3. Full Paths to GCPs Conversion

Full path	GCP
$(lr, lc1, lc2, la)$	lr $\mapsto$ la
$(lr, lc3, lc4, rr)$	lr $\mapsto$ rr
$(ra, lc0, lc1, lc2, la)$	ra $\mapsto$ la
$(lr, lc1, lc2, lc5, lc6, y)$	lr $\mapsto$ y
$(lr, lc3, lc4, lc5, lc6, y)$	

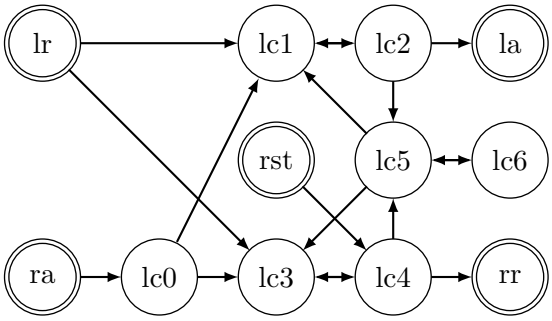

Figure 2.7. Graph Representation for LC circuit

2.7.1 Finding All the Cycles Present in the Circuit

A brute force algorithm is implemented to find all the local cycles in a circuit. These structural cycles are independent of the path constraints supplied to the tool.

A depth first search is performed for each vertex $v_i \in V$ in the adjacency list G to find paths that return to the vertex v_i. If such a path exists then the stack which stores the trace is recorded as a cycle. The LC controller in Figure 2.2 highlights the eight circuit cycles as shown in Table 2.4.

Note that Cycle2 and Cycle3 have the same gate sequence because the output of gate $lc2$ goes into two separate inputs of gate $lc5$. This can be verified from the adjacency list

Table 2.4. Internal Cycles

Cycles	Path
Cycle0	[$lc1$ $lc2$ $lc1$]
Cycle1	[$lc1$ $lc2$ $lc5$ $lc1$]
Cycle2	[$lc1$ $lc2$ $lc5$ $lc1$]
Cycle3	[$lc3$ $lc4$ $lc3$]
Cycle4	[$lc3$ $lc4$ $lc5$ $lc3$]
Cycle5	[$lc3$ $lc4$ $lc5$ $lc3$]
Cycle6	[$lc5$ $lc6$ $lc5$]
Cycle7	[$lc5$ $lc6$ $lc5$]

with gate $lc5$ appearing twice as the successor of gate $lc2$. Similarly, Cycle5 and Cycle6 and also Cycle7 and Cycle8 have the same gate sequence but are two separate cycles. This implies that two timing cuts may be necessary to cut a single cycle.

The fanout and fanin of paths through the pins of the gate (or through independent gates) can result in an exponential number of paths and cycles in a design. Hence, the complexity of finding paths and cycles is theoretically exponential based on the number of vertices (gates) in a design. Fortunately, sequential controllers generally have few gates (typically less than 20). Finding the paths and cycles for the circuits under investigation result in small run times, even with fanin and fanout as observed above.

2.7.2 Timing Paths with False Path Removal Using GCP

A depth first search is performed to generate all paths between the timing endpoints in set Θ specified by the user. All the timing paths from sets of timing endpoints in set Θ are identified and pruned to only GCPs for each pair of timing endpoints. The following paths are returned for a set of timing endpoints ($lr \rightarrow la$) for the LC controller are shown in Figure 2.1. These are pruned to the GCP by removing the covered paths $[lc1, lc2]$, $[lc3, lc4, lc5, lc1, lc2]$. Table. 2.5 shows the complete set of GCPs for Θ.

2.7.3 Generating Cycle Cuts

Two algorithms have been implemented to generate cycle cuts:

- **V1:** This is a polynomial time greedy approach in terms of the number of cycles. The solution is created by cutting maximum occurring edges in the covering table.
- **V2:** This is an exponential time approach in terms of the number of cycles which searches through the complete list of solutions possible to find the highest quality solution.

Quality metrics for the tool flow report the status whether all cycles are cut, if any cut paths remain uncut, and if there are any orphaned gates without a timing path passing

Table 2.5. The Set of GCPs for Θ

Endpoints	Paths
lr $\rightarrow$ la	$[lr, lc1, lc2] \times 2$
lr $\rightarrow$ rr	$[lr, lc3, lc4]$
ra $\rightarrow$ la	$[ra, lc0, lc1, lc2]$
lr $\rightarrow$ y	$[lr, lc1, lc2, lc5, lc6] \times 4$
lr $\rightarrow$ y	$[lr, lc3, lc4, lc5, lc6] \times 2$

through them. There is no need to create a minimal set of cuts; what matters is that the "right" set is created which does not violate any of the rules in Section 2.6.

The basis of both algorithms is the same. Up to this point, all cycles and true timing paths have been defined. The problem of generating the cycle cuts is converted into a covering problem for which a covering table is generated with the cycles as the rows and the edges as columns. Only the edges present in the cycles which can be cut are considered. All the edges which are present on a GCP are excluded since they can not be cut due to Rule 1 precedence.

Edges that have the same source and destination gates are combined into a single column for the covering table even though they might generate multiple set_disable_timing constraints. Shorthand is used in the following table. Columns la_0, y_{-0}, y_{-1}, y_{-2}, and rr_0 represent the edges $(lc2, lc1)$, $(lc5, lc1)$, $(lc5, lc3)$, $(lc5, lc6)$, and $(lc4, lc3)$ respectively. Figure 2.8 presents the covering table for the circuit of Figure 2.1.

After the generation of the covering table, the V1 algorithm selects the edge that cuts the maximum number of cycles (rows). Each selected edge is removed from future consideration by removing that column. One or more set_disable_timing constraints are written out, and all rows representing cycles which get cut by this edge are removed. The algorithm iterates through the table to find a solution by repeatedly selecting an edge that is present in the most rows and updates the table.

The generation of the local cycle cuts ends when there are either no more edges (some cycles were not cut), or there are no more rows (all cycles have been cut) in the table.

There are four edges which can result in cutting two cycles for the covering table of the circuit of Figure 2.1. Assume the rightmost edge y is selected. Cycle6 and Cycle7 are cut by removing this edge. This leads to the cycle count for the y_{-2} edge to become zero, and hence that column is also removed. Continuing this process leads to the cut set shown in

	la_0	rr_0	y_{-0}	y_{-1}	y_{-2}	y
[$lc1$ $lc2$ $lc1$]	√					
[$lc1$ $lc2$ $lc5$ $lc1$]			√			
[$lc1$ $lc2$ $lc5$ $lc1$]			√			
[$lc3$ $lc4$ $lc3$]		√				
[$lc3$ $lc4$ $lc5$ $lc3$]				√		
[$lc3$ $lc4$ $lc5$ $lc3$]				√		
[$lc5$ $lc6$ $lc5$]					√	√
[$lc5$ $lc6$ $lc5$]					√	√

Figure 2.8. Covering Table of the Local Cycles for the Linear Controller

Figure 2.9, which removes all the cycles with 6 cuts, graphically shown in Figure 2.10.

Algorithm V2 also creates the covering table but goes one step further. It generates the complete list of solutions. A solution cost heuristic is employed to select the best solution. After generating each new solution, its cost is calculated and compared against the previous best solution. The solution with the minimum cost is selected as the best solution. Following is the cost heuristic, which gives priority to remaining uncut cycles. The constant K gives different weights for *number of uncut cycles* and *number of orphaned gates*. A orphaned gate exists when no timing path passes through the gate. This paper assigns $K = 3$ to give a higher cost for uncut cycles.

$$\begin{aligned} cost &= K \times number\ of\ uncut\ cycles \\ &\quad + number\ of\ orphaned\ gates \end{aligned} \tag{2.1}$$

The search ends when the first solution with zero cost is found, or when all the partial solutions have been generated. In the latter case, the partial solution with the lowest cost is returned.

After generating cycle cut constraints for all the local cycles, the cut path constraints are applied to remove false paths and external cycles. These cuts are generated from set Φ. A depth-first search is performed to find all the cut paths in the circuit. This results in total of 24 paths listed in Table 2.6.

A covering table is constructed where the rows define paths and columns define the edges on those paths that can be cut. Similar to local cycle cutting, edges that are on true timing paths are not added to the table. Note that paths that are covered by transitive closures of GCPs will not be included in the table, just their individual segments. Edges which have already been removed from the graph to cut cycles are also excluded from the table, as well as paths that have been cut as part of cycle removal. The V1 algorithm, that eagerly selects cuts based on the number of paths cut, is employed. A set_disable_timing cycle cut constraint is written out, and all rows representing cut paths by this edge are removed. The algorithm

```
set_disable_timing -from A2 -to Y  [find -hier cell *lc1]
set_disable_timing -from A2 -to Y  [find -hier cell *lc3]
set_disable_timing -from B1 -to Y  [find -hier cell *lc5]
set_disable_timing -from C1 -to Y  [find -hier cell *lc5]
set_disable_timing -from B1 -to Y  [find -hier cell *lc1]
set_disable_timing -from B1 -to Y  [find -hier cell *lc3]
```

Figure 2.9. The sdc Constraints Generated to Remove All Cycles

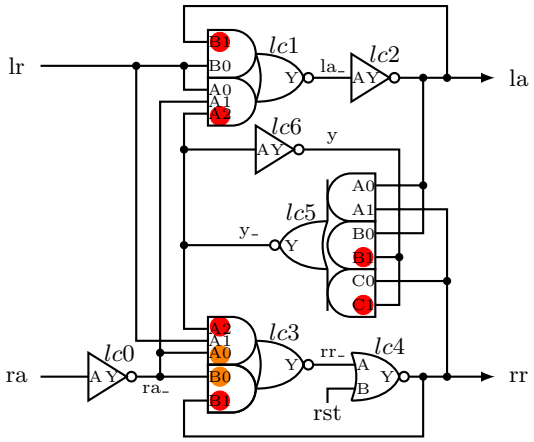

Figure 2.10. LC Circuit Implementation Showing Local (Red) and External (Orange) Timing Arc Cuts Through the Marked Gates

Table 2.6. The Timing Endpoints Specified in Set Φ

Endpoints	Paths
$ra \nrightarrow rr$	$[ra, lc0, lc1, lc2, lc5, lc3, lc4] \times 2$
$ra \nrightarrow rr$	$[ra, lc0, lc3, lc4] \times 2$
$lr \nrightarrow la$	$[lr, lc1, lc2] \times 2$
$lr \nrightarrow la$	$[lr, lc3, lc4, lc5, lc1, lc2] \times 2$
$lr \nrightarrow rr$	$[lr, lc1, lc2, lc5, lc3, lc4] \times 4$
$lr \nrightarrow rr$	$[lr, lc3, lc4]$
$ra \nrightarrow la$	$[ra, lc0, lc1, lc2]$
$ra \nrightarrow la$	$[ra, lc0, lc3, lc4, lc5, lc1, lc2] \times 4$
$lr \nrightarrow rr$	$[lr, lc1, lc2, lc5, lc6] \times 4$
$lr \nrightarrow rr$	$[lr, lc3, lc4, lc5, lc6] \times 2$

ends when there are either no more edges (some cut paths could not be cut), or there are no more rows (all cut paths have been cut) in the table.

Since edge $(lc5, lc3)$ and $(lc5, lc1)$ have been cut to remove cycles, those two columns and the 12 associated rows that are cut with these edges are not included in the table. This leaves just two rows to be cut with only a single edge as an option since the other 10 paths are GCPs. Figure 2.11 illustrated the covering table. Two associated sdc constraints are generated to remove these two rows in the covering table. which is shown in Figure 2.12.

The cut points for the circuit are illustrated in Figure 2.10. The algorithm has now generated a complete DAG for the system with all of the true paths intact, all of the cycles cut, all of the false paths cut, without leaving any orphaned gates such that true paths pass through each gate in the design. Timing targets can be placed across each of the timing

	$(lc0, lc3)$
$[ra, lc0, lc3, lc4]$	$\surd$
$[ra, lc0, lc3, lc4]$	$\surd$

Figure 2.11. Covering Table for Set ϕ

```
set_disable_timing -from A0 -to Y [find -hier cell *lc3]
set_disable_timing -from B0 -to Y [find -hier cell *lc3]
```

Figure 2.12. The sdc Constraints to Remove Timing Paths

endpoints in Θ to optimize the sizing and placement of the gates to ensure that hazards to not occur in the design resulting in failure, and the performance of the circuit can be optimized.

2.8 Results

The algorithm described in this paper is written in C++. The results are reported for runs on an Intel core i7 processor with 4GB memory. Most sequential control circuits are relatively small and so this problem is not constrained by runtime or memory.

2.8.1 Four-Cycle Handshake Controllers

This example set consists of the complete family of 131 untimed four-cycle handshake controllers with data valid at the rising edge of request (lr) [52],[53]. The specifications are generated from concurrency reduction rules, and they are synthesized with Petrify. This creates a rich set of controller modules with various properties, such as half and full data buffered pipelines. The concurrency reduction rules were applied to the most concurrent protocol to generate the complete set of untimed (speed-independent and delay-insensitive) protocols.

The evaluation approach described in Section 2.4 is employed. This includes the three performance paths ($lr \rightarrow la$), ($lr \rightarrow rr$), and ($ra \rightarrow la$). Additional timing paths specific to each controller that are required to remove hazards from the design are also included. The cut path ($ra \nrightarrow rr$) is included to remove cycles on the handshake channel.

Data are included in a tabular and graphical form. Table 2.7 shows results for each individual controller. Some of the handshake controllers failed synthesis from Petrify and are marked as '–' in the table. Those that deadlock due to too much concurrency reduction are marked with '.'. Graphical results collect data into sets for each of the left cut (Lxxx).

Thus, from 6 to 16 controllers are included for each Lxxx value in the graphs.

The table and graphs generally display data from higher to lower concurrency. (The improved left cut nomenclature of [53] is employed.) Larger Lxxx numbers represents increased concurrency reduction on the output rr/ra channel. Likewise, Rxxxx cuts for each Lxxx set create orthogonal concurrency reduction on the input lr/la channel. All concurrency reduction values for the input channel are included in the Lxxx value, so the numbers can have significant variance. The average value for the left cut set is identified by the line with the maximum and minimum values identified with the error bars.

Table 2.7 shows the total cycles, cycles and false paths left uncut, and orphans for the design. The total number of cycles in these controllers is identified in the table. The amount of concurrency in a design is directly proportional to the number of state variables required [26]. As expected, the most concurrent protocols contain the largest number of cycles due to more state holding feedback signals. Table 2.7 also shows the number of cycles and false paths left uncut, and the number of orphaned gates. The uncut false paths are reported next to the uncut cycles since false paths are removed when cycles are cut. Six designs have one false path that was left uncut, and two designs have two uncut false paths. All cycles were removed in 118 of the 131 test cases employing the given constraint paths.

Table 2.7. Total Cycles Found / Cycles Left Uncut$_{false\ path}$ / Orphans for the V1 Algorithm

LoR	L000	L200	L400	L220	L420	L222	L422	L440	L442	L444
R0000	–	–	$18/3_2/0$	35/0/2	–	–	11/0/2	$24/3_0/7$	$7/1_0/0$	$5/1_0/0$
R0020	38/0/1	–	$17/2_2/1$	14/0/0	11/0/0	13/0/0	$9/1_1/0$	13/0/0	9/0/0	8/0/0
R0040	20/0/0	23/0/1	15/0/0	14/0/0	44/0/6	19/0/8	8/0/0	5/0/0	8/0/0	10/0/0
R0022	25/0/0	59/0/9	$7/1_1/0$	10/0/4	19/0/9	8/0/0	7/0/0	3/0/0	4/0/0	$4/1_1/0$
R0042	39/0/15	13/0/1	14/0/0	–	16/0/0	35/0/7	7/0/0	$10/1_0/0$	6/0/0	$5/1_0/0$
R2022	22/0/11	30/0/6	44/0/12	7/0/0	12/0/5	12/0/3	8/0/0	6/0/0	4/0/0	.
R2042	50/0/6	20/0/1	10/0/0	5/0/3	7/0/0	8/0/0	6/0/0	4/0/0	4/0/0	.
R0044	10/0/0	7/0/0	10/0/1	4/0/0	5/0/5	10/0/6	4/0/0	6/0/0	3/0/3	$3/1_0/0$
R2044	7/0/1	9/0/0	7/0/0	4/0/6	6/0/7	6/0/1	4/0/0	2/0/0	3/0/5	.
R4044	18/0/0	7/0/1	.	3/0/0	.	$5/1_0/0$	.	.	.	.
R2222	19/0/8	7/0/3	5/0/0	3/0/0	5/0/0	5/0/3	4/0/0	4/0/0	2/0/1	.
R2242	17/0/0	9/0/0	8/0/0	7/0/3	6/0/5	5/0/2	4/0/0	3/0/0	3/0/0	.
R2262	7/0/2	7/0/0	10/0/0	4/0/0	$5/2_1/0$	.	.	3/0/0	.	.
R2244	3/0/1	4/0/0	4/0/1	1/0/0	1/0/0	1/0/0	1/0/0	1/0/0	1/0/0	.
R2264	5/0/0	6/0/0	5/0/0	1/0/0	2/0/0	.	.	1/0/0	.	.
R4244	5/0/4	6/0/3	.	1/0/0	.	2/0/0	.	.	.	.
R4264	4/0/1	4/0/0	.	1/0/3	.	.	.	.	.	.

A number of gates were orphaned. Gates are orphaned for two reasons: first, there is no timing path through the gate, and second, all input to output timing arcs get cut.

Further investigation reveals that all the orphan gates have no timing constraint path passing through them in both V1 and V2 algorithms. Theses gates are primarily associated with reset and the local state variable logic. Thus, these gates can be sized by applying constraint paths that are specific to the state logic of each design.

A summary reports the maximum, minimum, and average values of the reported parameters for the aggregate set of controllers. The parameters are listed including the energy-delay product, the forward latency, the backward latency, the cycle time, the postlayout area, the power consumption, the power consumption, the completion time, and the energy per token as shown in Table 2.8. Figures 2.13 to 2.20 presents the individual parameters of the set of controllers across the left cuts.

Figure 2.13 shows the energy delay product comparing cycle cutting being performed by the V1 algorithm and a commercial CAD tool. The benefit of the V1 algorithm ranges from an improvement of 13.7× to 1.2× over a commercial CAD tool. Figure 2.14 shows the forward latency, Figure 2.15 shows the backward latency, and Figure 2.16 shows the cycle time of each module with cycle cutting performed by a commercial CAD tool and our timing path driven cycle cutting. A comparison of these graphs show that the commercial CAD tool typically generates a slower circuit except for a few cases where it used big gates that improved performance by consuming substantially more energy.

Figure 2.17 shows the postlayout area. Gates are substantially over-sized when the commercial CAD tool performs cycle cutting. Four of the five cases with small area advantage (1.9× times or less) are for the circuits with uncut cycles. In these five cases, the commercial CAD tool creates additional cycle cuts after our timing driven algorithm is employed as it

Table 2.8. The Parameters for the Aggregate Set of Controllers

	This Work			Commercial CAD		
	min	max	mean	min	max	mean
$e\tau^2$	30.0	56.7	38.7	31.0	96.9	54.34
Forward Latency (ps)	260	2120	895	360	3400	1393
Backward Latency (ps)	90	2540	846	30	4000	1388
Cycle Time (ps)	260	2120	895	360	3400	1392
Area (μm^2)	56	442	145	145	882	450
Power (mW)	0.2	2.4	0.7	0.7	4.8	1.9
Completion Time (ns)	30.0	56.9	38.7	31.0	96.9	54.3
Energy per token (pj/token)	0.2	2.2	0.5	0.6	4.1	2.0

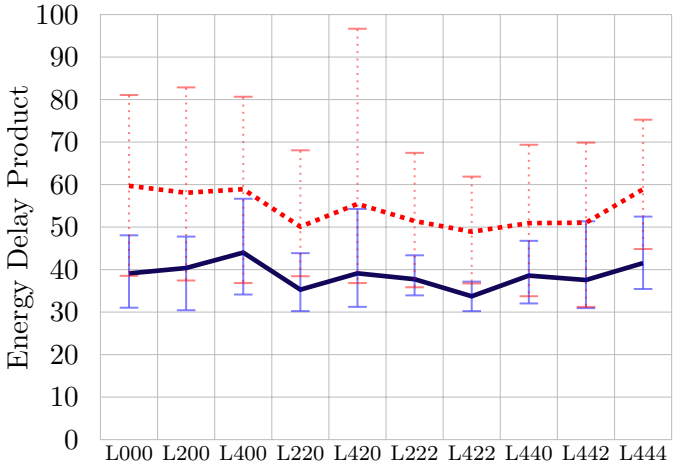

Figure 2.13. $e\tau$ Ratio Averaged Across Left Cuts. Interval Shows Largest and Smallest Values for the Cut, Line Passes Through the Mean. Dotted Line Depicts Using Commercial CAD Tools While Solid Line Depicts Using Our Algorithm

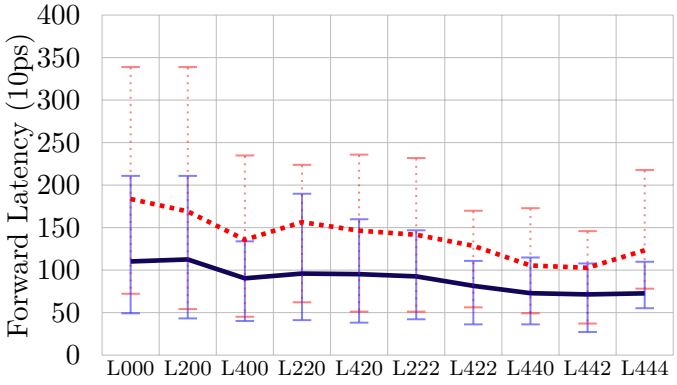

Figure 2.14. Forward Latency Averaged Across Left Cuts. Interval Shows Largest and Smallest Values for the Cut, Line Passes Through the Mean. Dotted Line Depicts Using Commercial CAD Tools While Solid Line Depicts Using Our Algorithm

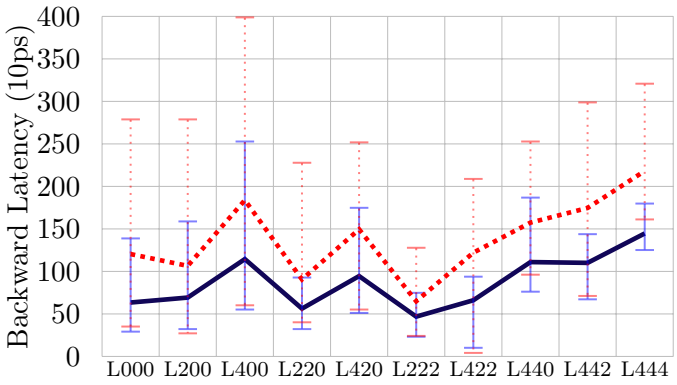

Figure 2.15. Backward Latency Averaged Across Left Cuts. Interval Shows Largest and Smallest Values for the Cut, Line Passes Through the Mean. Dotted Line Depicts Using Commercial CAD Tools While Solid Line Depicts Using Our Algorithm

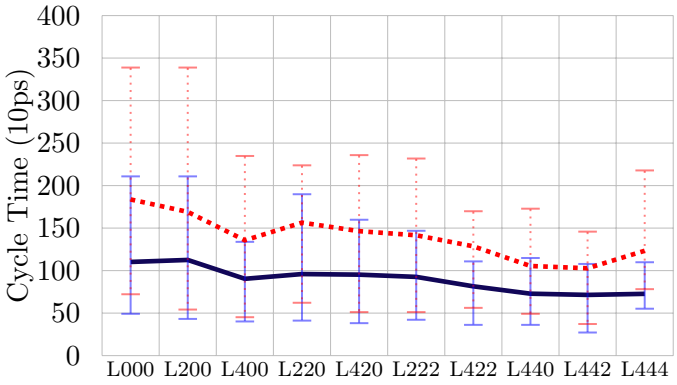

Figure 2.16. Cycle Time (Post APR) ($10ps$) Averaged Across Left Cuts. Interval Shows Largest and Smallest Values for the Cut, Line Passes Through the Mean. Dotted Line Depicts Using Commercial CAD Tools While Solid Line Depicts Using Our Algorithm

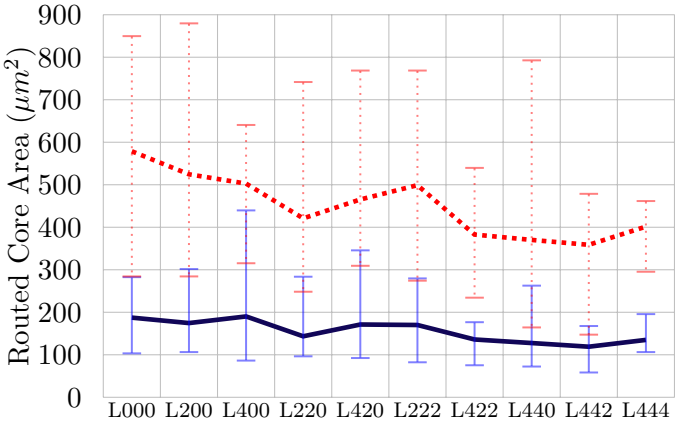

Figure 2.17. Core Area Averaged Across Left Cuts. Interval Shows Largest and Smallest Values for the Cut, Line Passes Through the Mean. Dotted Line Depicts Using Commercial CAD Tools While Solid Line Depicts Using Our Algorithm

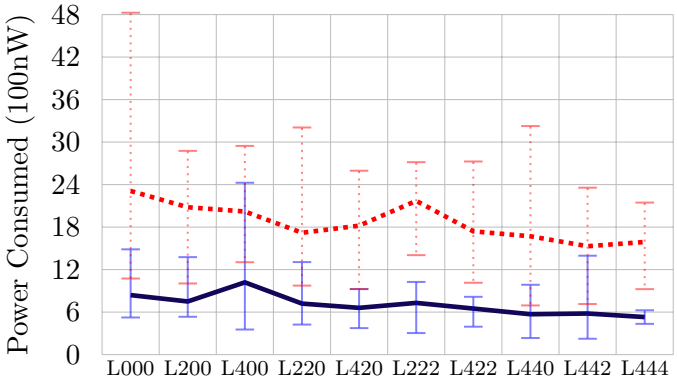

Figure 2.18. Power Comsumption Averaged Across Left Cuts. Interval Shows Largest and Smallest Values for the Cut, Line Passes Through the Mean. Dotted Line Depicts Using Commercial CAD Tools While Solid Line Depicts Using Our Algorithm

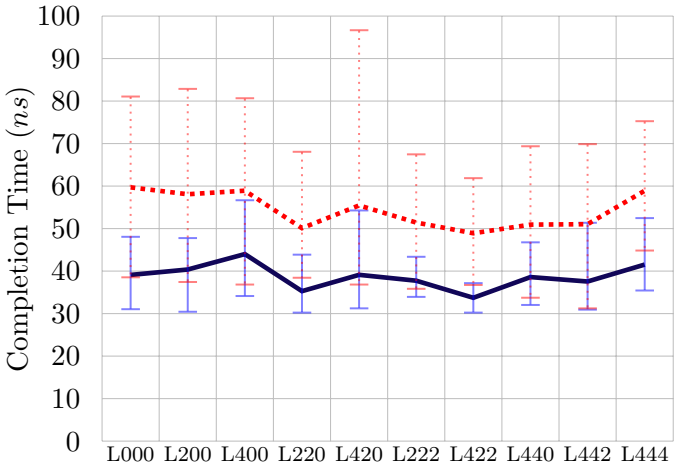

Figure 2.19. Computation Time Averaged Across Left Cuts. Interval Shows Largest and Smallest Values for the Cut, Line Passes Through the Mean. Dotted Line Depicts Using Commercial CAD Tools While Solid Line Depicts Using Our Algorithm

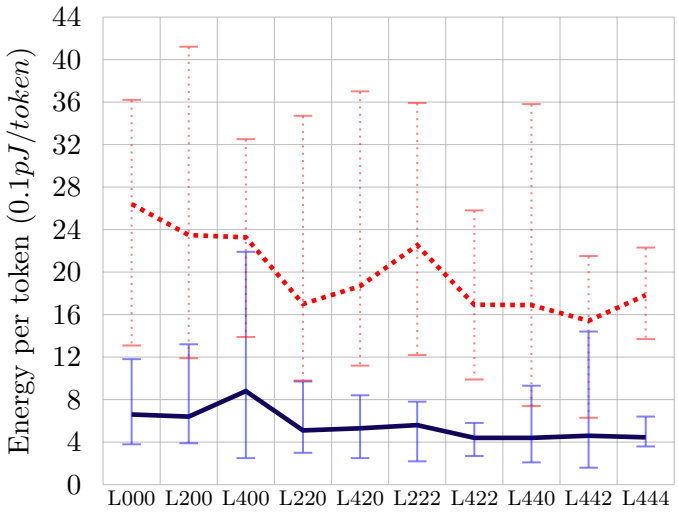

Figure 2.20. Averaged Energy Averaged Across Left Cuts. Interval Shows Largest and Smallest Values for the Cut, Line Passes Through the Mean. Dotted Line Depicts Using Commercial CAD Tools While Solid Line Depicts Using Our Algorithm

synthesizes and optimizes the design.

The same conclusion can be made by comparing the power consumption, completion time, and energy reported in Figure 2.18, Figure 2.19, and Figure 2.20. There are five designs that have higher performance when the commercial CAD tool performs cycle cutting, but the performance improvement again comes at the cost of expending lots more power. The energy numbers for the pipeline design average 3.84× larger for the case when the commercial CAD tool performs cycle cutting.

2.8.2 Benchmark Circuits

A number of benchmark circuits of varying complexity were also evaluated, including the largest published asynchronous finite state machines which sequential synthesis tools are able to create. These benchmarks include a GCD design and modules from the Post Office and pipelined small computer system interface (PSCSI) controllers [54],[55]. These designs were synthesized using Petrify to generate a gate level netlist to which reset was added by hand.

The test setup for these designs is similar to that for the examples in Section 2.5. The notable exception is that the circuits were not formally verified for correctness against the specification in order to generate the true timing paths in the design. Instead manual analysis of these designs was performed to identify the true paths taken for each input to output path in the design. Only one true path from each primary input to each primary output was identified, if one exists. These true paths are supplied to the algorithm. Also, the external connectivity of these designs is ignored, so no external cut paths were employed to cut handshake channel cycles.

Table 2.9 gives a comparison for the cycle cuts generated by the algorithms with and without true path information in terms of, area, energy per token, and run time performance. The complexity of each design is based on the number of paths, cycles, and gates. The generic algorithm uses primary inputs and outputs to form GCPs and generates cycle cuts [2]. These numbers give a comparison on the effectiveness of providing true paths of sequential designs. The average benefit for comparing the designs with and without true timing paths are 40% area reduction, 30% energy reduction with a 5% performance improvement. This gives an average 1.4× $e\tau$ benefit.

2.9 Summary

Timing paths must be cut to represent the timing graphs of sequential circuits as DAGs in the current state-of-the-art EDA tools. An algorithmic approach is presented

Table 2.9. Results Comparison for Benchmark Circuits

				GCP of PI to PO			True timing paths			Benefit		
	No. of Cycle	Gate Count	Paths	Area (um^2)	Energy/ token (pJ)	Comp time (ns)	Area (um^2)	Energy/ token (pJ)	Comp time (ns)	Area	Energy/ token	Comp time
rcv-setup	1	8	4	54.9	0.03	87.86	42.0	0.03	91.15	1.3	1	0.96
sbuf-send-ctl	12	28	120	122.6	0.34	329.45	103.2	0.29	316.09	1.18	1.17	1.04
pscsi-trcv-bm	6	26	32	152.6	0.25	139.69	100.8	0.16	149.78	1.51	1.56	0.93
pscsi-tsend-bm	10	33	377	197.2	0.32	234.11	115.2	0.22	199.47	1.71	1.45	1.17
pscsi-tsend	100	35	1819	193.7	0.3	213.77	122.4	0.21	188.61	1.58	1.42	1.13
pscsi-isend	325	43	6122	261.4	0.54	277.51	204.0	0.44	258.79	1.28	1.23	1.07
gcd	22	72	175	428.6	0.77	302.74	348.6	0.61	299.97	1.23	1.26	1.01
Average Benefit										1.40	1.30	1.05

for automating the timing path driven generation of these cycle cuts so that the EDA tools can properly perform gate sizing for performance, area, and energy optimizations on sequential circuits without modifying the underlying structural netlist. The algorithm leverages the CAD tool that performs cycle cutting and preserve timing paths [2]. True paths are generated using a verification engine and supplied to the developed tool. It is passed as timing endpoints of two forms: those that identify the true timing paths in the circuit and thus can not be cut to preserve necessary timing paths, and those that identify false paths and external cycles which must be cut. A method based on the greatest common path is provided for specifying the correct timing paths in the sequential circuits based on timing endpoints composition. The developed tool reports on the quality metrics of the results, consisting of the number of cycles left uncut, the number of false paths that were not cut, and the number of gates that do not have a timing path passing through them. Two versions of the algorithm are presented: a faster greedy search as well as an exhaustive algorithm that returns a result of the highest quality.

The cycle cutting constraints in the sdc format is generated by the tool. This timing path driven cycle cutting algorithm is a key component of any design and CAD flow that enables asynchronous designs to be synthesized, placed and routed, power and performance optimized, and validated for postlayout timing correctness using commercial EDA tools. The input is the structural Verilog design of sequential controllers that implement handshake protocols and sequencing in asynchronous designs.

The algorithms are general to any sequential circuit. The algorithm is demonstrated on a test bench of 131 four-cycle bundled data asynchronous controllers, one delay-insensitive design, and a set of large gate count benchmark circuits. Circuits in this example set have

as many as 325 cycles and over 6K paths in the implementation.

It is shown that allowing EDA tools to generate a DAG employing algorithms that do not respect timing paths passing through the sequential circuit leads to issues in timing optimization and validation, as well as producing inferior circuits. The generic cycle cutting algorithm generates on average of 40% smaller in size, 30% less in energy, and 5% faster designs while the true paths are provided. More importantly, true timing paths of the circuit can not be evaluated using STA unless a DAG is generated that respects the timing paths. If some timing paths do not meet specified delays, the circuit will fail to operate correctly.

CHAPTER 3

PATH BASED TIMING VALIDATION FOR TIMED ASYNCHRONOUS DESIGN[1]

The advent of novel circuit design methodologies, like asynchronous circuits, can enable a circuit to operate at multiple frequencies with power and performance benefits. Numerous advantages are accrued through employing commercial EDA tools for asynchronous design approaches. These tools have support for leading technology features such as double patterning and multiple timing corners, hence, design productivity is enhanced. Unfortunately, asynchronous circuit designs can not be directly supported with commercial EDA tools.

The lack of commercial EDA support is largely due to the disconnect in the timing models employed for clocked and asynchronous design. Numerous challenges must be overcome to support asynchronous timing models in commercial EDA tools. Timing paths in asynchronous design are not simple combinational paths, as is the case with clocked design. Many timing paths in asynchronous design must be calculated based on the specific logic being employed. Such timing constraints often consist of two or more related paths. Nearly all asynchronous designs contain combinational cycles, which also must be evaluated. Asynchronous sequential controllers often have hazards that can be avoided if specific delay relationships hold. Timing constraints that make hazards unreachable must hold for design correctness. Other timing constraints exist in timed asynchronous circuits to optimize and validate performance. Most of the timing constraints to ensure circuit performance are cyclical.

These challenges need to be resolved for asynchronous design to employ commercial EDA tools. This paper specifically addresses the challenge of creating accurate path based delays for asynchronous sequential circuits, including paths that contain cycles. A methodology

[1]This section has been published in VLSI design, 2016 [56]. © 2016 IEEE. Reprinted, with permission, from William Lee, Tannu Sharma, Kenneth S. Stevens, "Path Based Timing Validation for Timed Asynchronous Design," in proceedings of the 29th International Conference on VLSI Design and 15th International Conference on Embededd Systems (VLSI-Design), January, 2016.

is developed that utilizes the static timing analysis (STA) algorithms used by commercial EDA tools.

3.1 Background

Timing in a system is employed to enforce specific event sequencing in a design. This is normally enforced with a clock signal, where the cycle time must be greater than the combinational propagation delay between pipeline stages. Timing is where asynchronous design differs from the clocked designs. Bundled data asynchronous design employs similar timing requirements, but timing is localized, flexible, and irregular. This creates problems and challenges for circuit optimization and validation, but also can provide power and performance advantages.

3.1.1 Asynchronous Designs

Asynchronous designs communicate and synchronize based on local handshake signals which identify data validity and the ability to accept new data transactions. The communication link is called a *handshake channel* that consists of data wires, a request signal (req) identifying data validity, and an acknowledgment signal (ack) indicating the data has been accepted [57],[58].

Performance evaluation for asynchronous designs is inherently different than that for clocked design because each *handshake channel* implements a silicon oscillator designed to operate at a particular frequency that matches the delay of the associated data path. The handshake channel implements the oscillator as a timing cycle. The timing path is even more complicated in four-cycle protocols since each data transfer consists of both rising and falling transitions on req and ack with these signals propagating along many if not all the gates in the handshake cycle. Commercial CAD does not support combinational cycles, and so external means must be implemented to support such cyclic timing paths.

Many asynchronous design styles, such as bundled data, require timing constraints to hold for the circuit to operate correctly. The LC blocks in Figure 3.1 implement the silicon oscillator that control the frequency of operation for a pipeline stage, the synchronization between pipeline stages, and generate a clock signal to store data in pipeline latches. These are implemented as asynchronous finite state machines (AFSM). Here, we assume these controllers are implemented with combinational logic where state holding logic contains combinational cycles. These AFSMs often have hazards which must be made unreachable by controlling the delays in the circuit. The hazards are commonly associated with the combinational cycles that implement the state holding function.

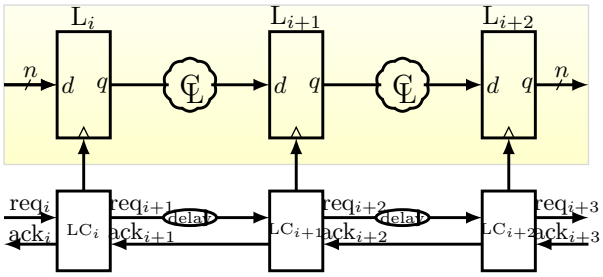

Figure 3.1. Timed (Bundled Data) Handshake Design. Each $req_i\uparrow$ Handshake on LC_i Indicates New Data are Presented to Pin d of L_i. Delay Sized by Relative Timing Constraint $req_i\uparrow \mapsto L_{i+1}/d + margin \prec L_{i+1}/clk\uparrow$.

This design approach poses multiple requirements which are not directly supported by the commercial EDA tool flow. First, timing paths must be explicitly identified that relate to the specific AFSM used in the design. Second, the timing graph of the circuit must be represented as a DAG. Third, cyclical timing paths must be evaluated based on a DAG timing graph representation.

In general, all the timing paths required evaluating a sequential circuit can not be known in a single iteration if the timing graph is represented as a DAG. Also, a DAG has some of the timing paths cut [59]. Thus, complete timing path analysis requires multiple timing path partitions, multiple acyclic timing graph representations, and multiple STA runs.

3.1.2 Controller Indexing for Mapping Timing Constraints

In our approach, asynchronous sequential controllers become the focal point for timing validation, much like the registers are in a clocked design. Every AFSM controller has been characterized with a set of RT constraints which must hold for its correct operation. The source of all timing paths to be validated are RT pods, which are all located inside the characterized controllers. The ending points of the paths are RT pocs which may be inside or outside of the controller.

Since timing evaluation is performed local to each controller, a representation is constructed to specify external module location relative to the current controller. The controller under evaluation is identified with an instance label "$i1." Upstream controllers (accessed through the "ack" port in Figure 3.1) are referenced as "$i0" design blocks; downstream pipeline elements (accessed through the "req" port) are referenced as "$i2" macros. Registers are connected to the controller through the "clk" output port and are identified by appending a capital R to the label (e.g., $i0R, $i1R, $i2R).

3.2 Relative Timing Verification

The relative timing verification flow used in this work is shown in Figure 3.2. This flow enables commercial EDA tools to be used with sequential controllers and asynchronous circuit design. Figure 3.3 is an example circuit used to explain the flow [44].

Relative timing constraints specify timing paths by listing their endpoints. The true timing path(s) between these endpoints must be identified because the structural STA algorithms may not select the true timing path through the circuit. Some of the paths may have cycles. The RT timing endpoints, along with the true timing paths, are provided as an

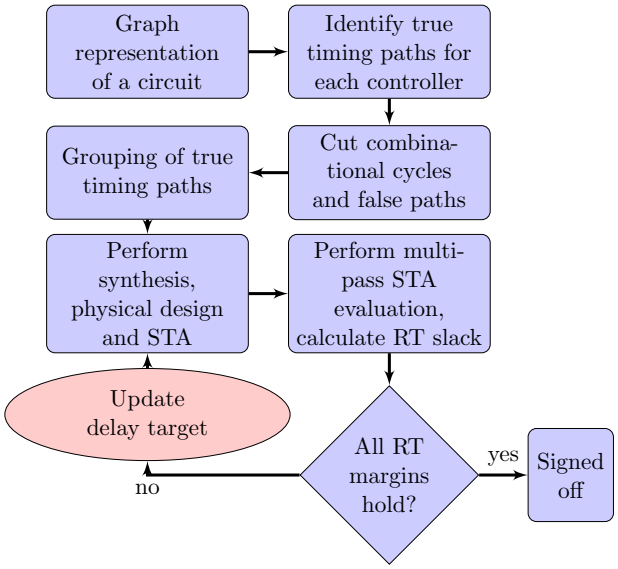

Figure 3.2. Relative Timing Verification Flowchart.

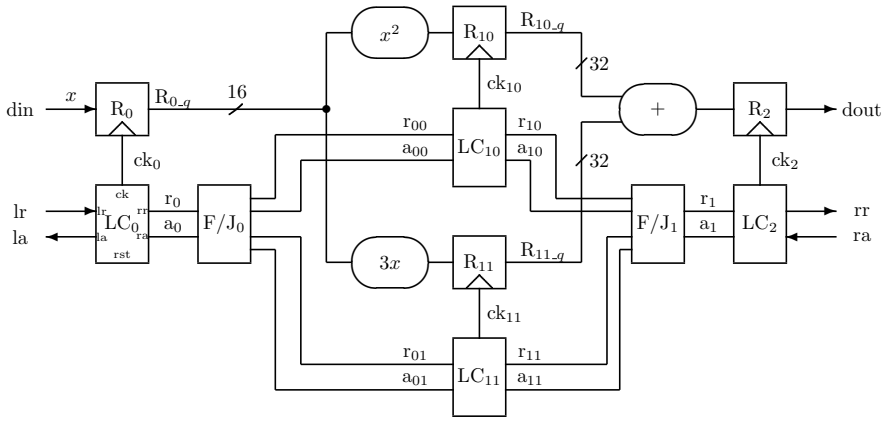

Figure 3.3. Example Design: a Simple ASIC Mathematical Pipeline Segment Computing dout $= x^2 + 3x$

input to this work. False paths are removed from the timing graphs by cutting timing arcs in the timing graph while preserving the true timing paths to ensure that the STA tools produce correct results. A separate CAD tool performs the timing path preserving cycle cutting operation [59].

Since timing validation can not be performed in a single pass, the RT constraints are grouped into compatibility sets. One set is used for timing driven synthesis and physical design. The full set of RT constraints are employed for timing validation. When performing timing validation, the full path delays are calculated along with RT slacks associated with each RT constraint.

A delay violation exists in an RT constraint when the margin is less than the min-delay minus the max-delay. The design will be signed-off if there are no violations. If timing violations are identified in the validation process, the delay targets associated with the RT constraints need to be updated and rerun synthesis and/or physical design or perform an ECO.

3.2.1 Graph Representation of a Circuit

Each asynchronous pipeline controller is represented as a cyclic graph $G = (V, E)$ where the input pins of each gate, the primary inputs, and the primary outputs are vertices (nodes) $v_i \in V$ of the graph, and edges $e_i = (v_x, v_y) \in E$ map connectivity between the vertices. The primary input and output vertices of G are identified with a double circle in figures. The sequential asynchronous handshake pipeline controller used in Figure 3.3, which is shown in Figure 3.4, can be represented to a directed graph form. Figure 3.5 is the directed graph representation for the controller.

A *path* is a sequence of vertices connected by edges in E in a directed graph. The starting and ending nodes of a path are called *timing endpoints*. There can be multiple paths between timing endpoints. One of the paths between the timing endpoints lr and rr is $[lr, 1/A0, 2/A, 5/A0, 3/A2, 4/A, rr]$. A *cycle* is a path that starts and ends with the same vertex. For instance, path $[1/B1, 2/A, 1/B1]$ is a cycle.

3.2.2 Identify Timing Paths for Each Controller

The control path is formed using pipeline controllers (LC). A set of RT constraints are associated with each controller and these constraints are mapped to the full design. Each relative timing constraint identifies two timing path sets: max-delay path(s) from pod to poc_0 and min-delay path(s) from pod to poc_1. The min-delay path must be at least m time units greater than the max-delay path.

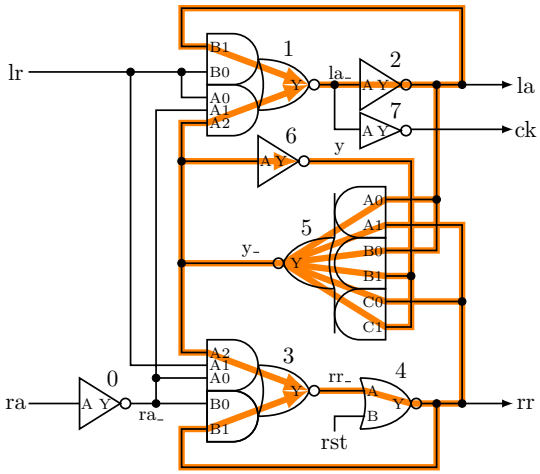

Figure 3.4. Circuit Implementation of a Burst-Mode Linear Controller. The Cycles in the Design Have Been Highlighted.

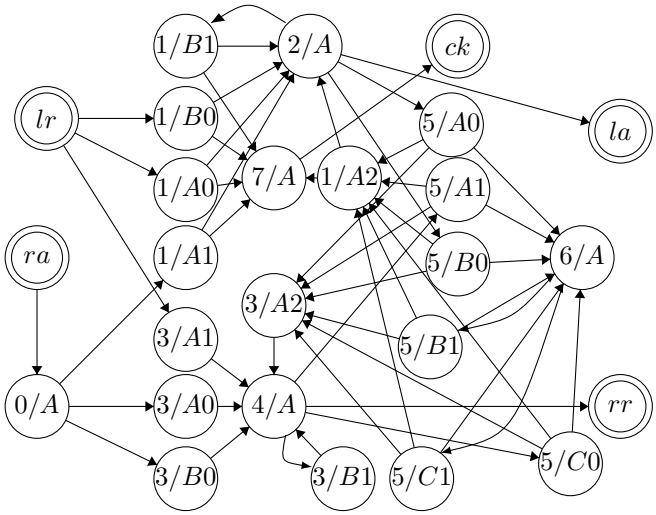

Figure 3.5. Graph Representation for the Linear Controller in Figure 3.4

The min-delay and max-delay paths for RT constraints (2) and (3) in Table 3.1 are both contained within the controller. A constraint may specify multiple true paths, as is the case for the min-delay path of (1) and the max-delay path of (8). The min-delay paths of constraints (0) and (1a) contain cycles because nodes $4/A$ and $2/A$ appear twice in the respective paths. These cycles are caused by the system-level interconnect of the handshake channel and the four-cycle RTZ handshake protocol. RT constraints (8) and (9) have timing endpoints at the datapath which are external to the controller. These two constraints represent the setup and hold time of the register or latch.

Table 3.1. Relative Timing Constraints and True Paths Representation of Figure 3.4

RT constraint	Full Timing Path	
rr+ $\mapsto$ y+ $\prec$ rr-	4/A-, 5/A1+, 6/A-, 5/B1+ 4/A-, rr+, $i2/lr+, $i2/la+, ra+, 0/A+, 3/B0-, 4/A+, 5/A1-	< (0)
lr+ $\mapsto$ y+ $\prec$ la-	lr+, 3/A1+, 4/A-, 5/A1+, 6/A-, 5/C1+ lr+, 1/A0+, 2/A-, la+, $i0/ra+, $i0/rr-, lr-, 1/B0-, 2/A+, 5/A0-	< (1a)
	lr+, 1/A0+, 2/A-, 5/A0+, 6/A-, 5/C1+ lr+, 1/A0+, 2/A-, la+, $i0/ra+, $i0/rr-, lr-, 1/B0-, 2/A+, 5/A0-	< (1b)
y_+ $\mapsto$ y- $\prec$ la+	5/C0-, 6/A+, 5/B1- 5/C0-, 1/A2+, 2/A-, 5/B0+	< (2)
y_+ $\mapsto$ y- $\prec$ rr+	5/C0-, 6/A+, 5/C1- 5/C0-, 3/A2+, 4/A-, 5/C0+	< (3)
lr+ $\mapsto$ ck+ $\prec$ la_+	lr+, 1/A0+, 7/A-, ck+ lr+, 1/A0+, 2/A-, la+, $i0/ra+, $i0/rr-, lr-, 1/B0-, 7/A+	< (4)
lr- $\mapsto$ ck- $\prec$ la_-	lr-, 1/A0-, 7/A+, ck- lr-, 1/A0-, 2/A+, la-, $i0/ra-, $i0/rr+, lr+, 1/B0+, 7/A-	< (5)
lr+ $\mapsto$ rr+ $\prec$ lr-	lr+, 3/A1+, 4/A-, 3/B1+ lr+, 1/A0+, 2/A-, la+, $i0/ra+, $i0/rr-, lr-, 3/A1-	< (6)
lr+ $\mapsto$ la+ $\prec$ ra+	lr+, 1/A0+, 2/A-, 1/B1+ lr+, 3/A1+, 4/A-, rr+, $i2/lr+, $i2/la+, ra+, 0/A+, 1/A1-	< (7)
lr+ $\mapsto$ $i2R/D $\prec$ $i2R/CLK-	lr+, 1/A0+, 7/A-, ck+, $i1R/CLK+, $i1R/Q, $i2R/D, lr+, 3/A1+, 4/A-, rr+, $i2/lr+, $i2/la+, ra+, 0/A+, 3/B0-, 4/A+, rr-, $i2/lr-, $i2/ck-, $i2R/CLK-	< (8a)
	lr+, 3/A1+, 4/A-, rr+, $i2/lr+, $i2/ck+, $i2R/CLK+, $i2R/Q lr+, 3/A1+, 4/A-, rr+, $i2/lr+, $i2/la+, ra+, 0/A+, 3/B0-, 4/A+, rr-, $i2/lr-, $i2/ck-, $i2R/CLK-	(8b)
lr- $\mapsto$ $i1R/CLK- $\prec$ $i1R/D	lr-, 1/A0-, 7/A+, ck-, $i1R/CLK- lr-, 1/A0-, 2/A+, la-, $i0/ra-, $i0/ck+, $i0R/CLK+, $i0R/Q, $i1R/D	< (9)

3.2.3 True Timing Path Driven Cycle Cutting

Asynchronous controllers contain cycles which must be cut to perform timing driven optimization and static timing analysis. These cycles must be cut in such a manner that the true timing paths in the circuit remain uncut. For example, as shown in Figure 3.5, the cycle $[1/B1, 2/A, 1/B1]$ can be cut at edge $(1/B1, 2/A)$ or $(2/A, 1/B1)$. If the latter cut is employed, all timing paths passing through gate 2 will also be cut. Thus, the preferred cut point for this cycle is the edge $(1/B1, 2/A)$.

In this paper, we assume that all the cycles through linear controllers are cut in the design. An external tool is employed to create a DAG when given a Verilog controller along with the associated set of true and false paths. A set of cut points are produced which

create acyclic timing graph that cuts all the cycles and false paths, while preserving the true paths. The tool also gives cut paths which will remove system level cycles in the design as described below.

In order to accurately gather the delay value of a cyclic RT constraint, at least two iterations of STA are required. We implement an algorithm to partition the design into independent timing runs that can be composed to create accurate full path timing, including paths with cycles. This is performed by partitioning the constraints into two partitions: forward cycle cut (FCC) to preserve downstream ($\$i1 \rightarrow \$i2$) paths, and backward cycle cut (BCC) to preserve upstream ($\$i1 \rightarrow \$i0$) paths. This works due to the locality of the timing constraints that are tied to each individual controller.

Each linear controller in a design will have an upstream and downstream channel connecting it to the adjacent pipeline stages, as shown in Figure 3.1. Consider controller LC_1, and the cycles in the channel connecting it to LC_0 and LC_2. After applying FCC (where timing path $ra \rightarrow rr$ is cut) and BCC (where timing path $lr \rightarrow la$ is cut), we obtain the DAGs shown in Figure 3.6 and Figure 3.7, respectively.

A search algorithm is implemented to generate overlapping cut points to allow more accurate delay calculations of paths, that must be cut to create a DAG. For instance, path $[rr+, \$i2/lr+, \$i2/la+, ra+, 0/A+, 3/B0-, 4/A+, rr-]$ is a cycle. The total delay of the path can be computed by cutting it at ra and calculating the timing from $rr+ \rightarrow ra+$, then cutting the path at $\$i2/la+ \rightarrow rr-$, adding the two delays, then subtracting the $\$i2/la+ \rightarrow ra+$ delay from the total.

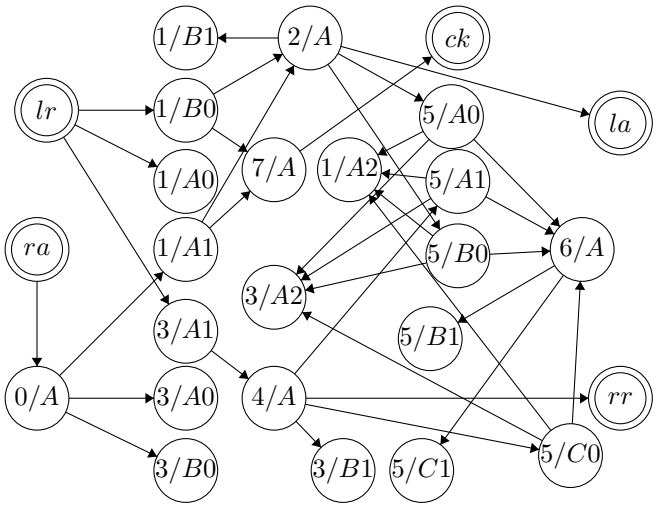

Figure 3.6. Local Cycles Have Been Cut and Handshake Channel Cycles are Removed with Forward Cycle Cutting (FCC) by Cutting All Timing Paths Between ra and rr.

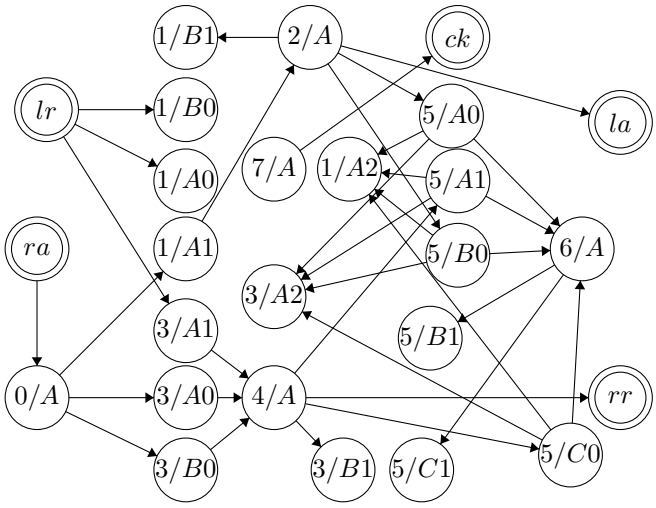

Figure 3.7. Local Cycles Have Been Cut and Handshake Channel Cycles are Removed with Backward Cycle Cutting (BCC) by Cutting All Timing Paths Between lr and la.

3.2.4 Graph Coloring Algorithm

An algorithm is written to break all paths with cycles into overlapping acyclic path segments. This generates a set of path segments and timing cut points which are composed to accurately time the full path.

A greedy graph coloring algorithm is applied to group nonintersecting true paths into sets for each controller instances in the design [60]. Two timing paths are intersecting if an endpoint of a timing path is also an internal node of another timing path. The algorithm ensures that a path in the set will not introduce a cut point in another path in the same set. Partitioning timing paths reduces the number of STA iterations required to generate full-path timing.

Table 3.2 reports the results of cutting cycles and partitioning path segments for the RT constraints for the controller shown in Figure 3.4. Each set contains a number of non-intersecting paths.

Asynchronous designs can be constructed using more than one type of linear controller. Each controller type will have its associated partition table. The RT constraints are applied to each controller instance in a design. The design shown in Figure 3.3 is built using a single type of controller so it requires only one partition table. The RT constraints are applied on each controller instance with independent delay targets.

All the path delays in each set can be analyzed in one iteration. However, when analyzing a pipelined design, the odd controllers (LC_0,LC_2) and even controllers (LC_{10},LC_{11}) of Figure 3.3 must be evaluated in separate STA runs. Because the same constraint sets

Table 3.2. Relative Timing Graph Nodes as Timing Paths

Set	No.	Constraint
A	1	(5/C0-, 6/A+, 5/B1-)
	2	(5/C0-, 6/A+, 5/C1-)
	3	(lr+, 1/A0+, 7/A-, ck+, $i1R/CLK+, $i1R/Q, $i2R/D)
	4	(lr+, 3/A1+, 4/A-, rr+, $i2/lr+, $i2/la+, ra+, 0/A+, 3/B0-)
	5	(lr+, 3/A1+, 4/A-, 3/B1+)
B	1	($i0/ra+, $i0/rr-, lr-, 3/A1-)
	2	(3/A2+, 4/A-, 5/C0+)
	3	(5/C0-, 1/A2+, 2/A-, 5/B0+)
C	1	(lr-, 1/A0-, 7/A+, ck-)
	2	(lr+, 1/A0+, 7/A-, ck+)
	3	(4/A-, 5/A1+, 6/A-, 5/B1+)
	4	(4/A-, rr+, $i2/lr+, $i2/la+, ra+, 0/A+, 3/B0-)
	5	(lr+, 3/A1+, 4/A-, 5/A1+, 6/A-, 5/C1+)
	6	(lr-, 1/A0-, 2/A+, la-, $i0/ra-, $i0/ck+, $i0R/CLK+, $i0R/Q, $i0R/D)
	7	(lr+, 1/A0+, 2/A-, la+, $i0/ra+, $i0/rr-)
D	1	(0/A+, 3/B0-, 4/A+, 5/A1-)
	2	(la-, $i0/ra-, $i0/rr+, lr+, 1/B0+, 7/A-)
	3	(la+, $i0/ra+, $i0/rr-, lr-, 1/B0-, 7/A+)
	4	(la+, $i0/ra+, $i0/rr-, lr-, 1/B0-, 2/A+, 5/A0-)
	5	(0/A+, 3/B0-, 4/A+, rr-, $i2/lr-, $i2/ck-, $i2R/CLK-)
E	1	(lr+, 3/A1+, 4/A-, rr+, $i2/lr+, $i2/la+, ra+, 0/A+, 1/A1-)
	2	(lr+, 1/A0+, 2/A-, la+, $i0/ra+)

are employed for all the controllers, some external constraints would overlap its adjacent controllers' constraint sets. The number of STA iterations required to perform timing validation would be twice the largest number of constraints with a controller.

The results report the implemented algorithm that cut paths into overlapping segments (including with and without cycles when necessary), partition path segments into compatible sets, and partition controllers into odd and even sets.

3.2.5 Perform Static Timing Analysis

Static timing analysis is performed after synthesis or layout. A commercial tool such as Synopsys PrimeTime or Cadence Tempus is invoked to perform STA. This paper uses PrimeTime. Multimode, multicorner analysis is performed to incorporate process, voltage, and temperate variation on timing path delays. Timing path delays from the complete set of paths (e.g., Table 3.2) are stored into a database.

3.2.6 Evaluate RT Slack for Every Timing Path

The RT slack for the RT constraint is the amount of time difference between the max-delay and min-delay timing paths. These slacks are calculated, stored, and compared against the minimum value required for the design. A value less than the required value indicates a

failed timing constraint (negative slack). Negative slack must be fixed by changing circuit timing by modifying timing targets, rerunning synthesis or place and route, and then this timing validation tool.

The slack of each RT constraint is plotted as a histogram in Figure 3.8. Constraint 8 is the data setup, 9 is the hold time, and the rest make hazards unreachable. This data allows designers to trade off performance and yield of a design.

Table 3.3 shows the detailed RT slack values obtained for the multiplier design in Figure 3.3. The pod column is the timing start point, the poc_0 column contains maximum delay endpoint, and the poc_1 column contains endpoint for the minimum delay path for the specified RT constraint.

3.3 Results

The methodology is now demonstrated on a 16-point Fast Fourier Transform (FFT-16) design will be shown in Section 6.1. The histogram in Figure 3.9 shows the cycle time distribution for various categories of pipeline stages in this multifrequency design. The three different color groups in the graph correspond to three different operational frequencies in the FFT-16 design. The slack distribution for RT constraints which are required for circuit correctness is shown in Figure 3.10. Figure 3.11 shows the setup time (blue) and hold time (red) slack distribution for the design. Due to the multifrequency nature of the design some paths are expected to have large setup and hold times.

3.4 Summary

Timing is the primary difference between asynchronous and clocked designs. Traditional static timing analysis algorithms can not be directly applied to asynchronous designs due to cycles and conflicting timing paths. A methodology and algorithm are presented which,

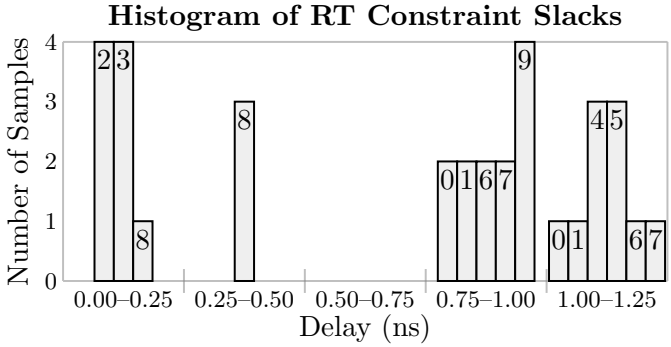

Figure 3.8. Slack Distribution for Table 3.1 Constraints Applied to Figure 3.3 Design.

Table 3.3. RT Slack Evaluation for the Multiplier Design

RT Constraint	**pod**	**poc$_0$**	**poc$_1$**	**RT slack (ns)**
rr+ $\mapsto$ y+ $\prec$ rr-	LC_0/rr	LC_0/y	LC_0/rr	0.97
	$LC_1$0/rr	LC_{10}/y	LC_{10}/rr	1.05
	$LC_1$1/rr	LC_{11}/y	LC_{11}/rr	0.84
lr+ $\mapsto$ y+ $\prec$ la-	LC_{10}/lr	LC_{10}/y	LC_{10}/la	0.97
	LC_{11}/lr	LC_{11}/y	LC_{11}/la	0.9
	LC_2/lr	LC_2/y	LC_2/la	1.07
y_+ $\mapsto$ y- $\prec$ la+	LC_0/y_	LC_0/y	LC_0/la	0.12
	LC_{10}/y_	LC_{10}/y	LC_{10}/la	0.09
	LC_{11}/y_	LC_{11}/y	LC_{11}/la	0.09
	LC_2/y_	LC_2/y	LC_2/la	0.11
y_+ $\mapsto$ y- $\prec$ rr+	LC_0/y_	LC_0/y	LC_0/rr	0.12
	LC_{10}/y_	LC_{10}/y	LC_{10}/rr	0.09
	LC_{11}/y_	LC_{11}/y	LC_{11}/rr	0.09
	LC_2/y_	LC_2/y	LC_2/rr	0.11
lr+ $\mapsto$ ck+ $\prec$ la_+	LC_{10}/lr	LC_{10}/ck	LC_{10}/la_	1.12
	LC_{11}/lr	LC_{11}/ck	LC_{11}/la_	1.07
	LC_2/lr	LC_2/ck	LC_2/la_	1.18
lr- $\mapsto$ ck- $\prec$ la_-	LC_{10}/lr	LC_{10}/ck	LC_{10}/la_	1.12
	LC_{11}/lr	LC_{11}/ck	LC_{11}/la_	1.07
	LC_2/lr	LC_2/ck	LC_2/la_	1.18
lr+ $\mapsto$ rr+ $\prec$ lr-	LC_{10}/lr	LC_{10}/rr	LC_{10}/lr	1.02
	LC_{11}/lr	LC_{11}/rr	LC_{11}/lr	1
	LC_2/lr	LC_2/rr	LC_2/lr	0.9
lr+ $\mapsto$ la+ $\prec$ ra+	LC_0/lr	LC_0/la	LC_0/ra	1
	LC_{10}/lr	LC_{10}/la	LC_{10}/ra	0.91
	LC_{11}/lr	LC_{11}/la	LC_{11}/ra	1.09
lr+ $\mapsto$ $i2R/D $\prec$ $i2R/CLK-	LC_0/lr	R10/D	R10/CLK	0.31
	LC_0/lr	R11/D	R11/CLK	0.31
	LC_{10}/lr	R2/D	R2/CLK	0.45
	LC_{11}/lr	R2/D	R2/CLK	0.25
lr+ $\mapsto$ $i2R/CLK- $\prec$ $i2R/D	LC_0/lr	R10/CLK	R10/D	0.76
	LC_0/lr	R11/CLK	R11/D	0.77
	LC_{10}/lr	R2/CLK	R2/D	0.78
	LC_{11}/lr	R2/CLK	R2/D	0.77

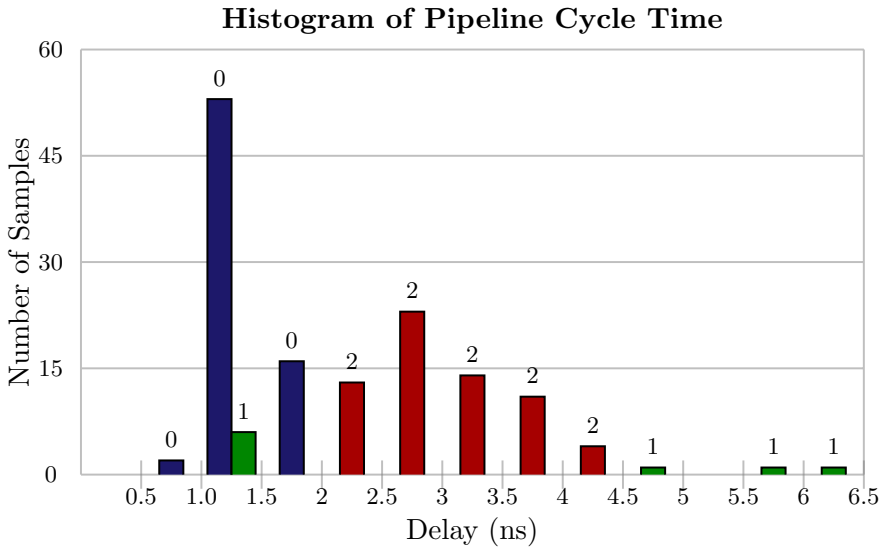

Figure 3.9. Cycle Time Distribution

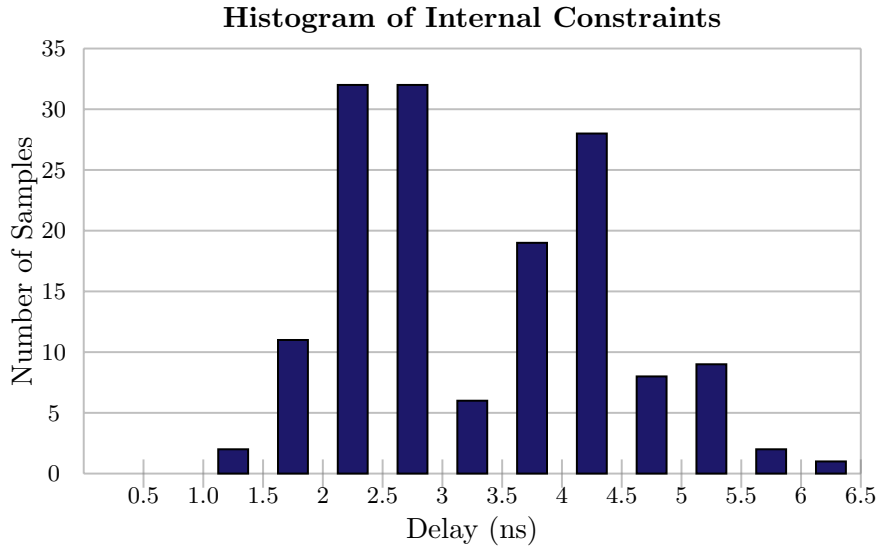

Figure 3.10. Distribution of Internal Slack

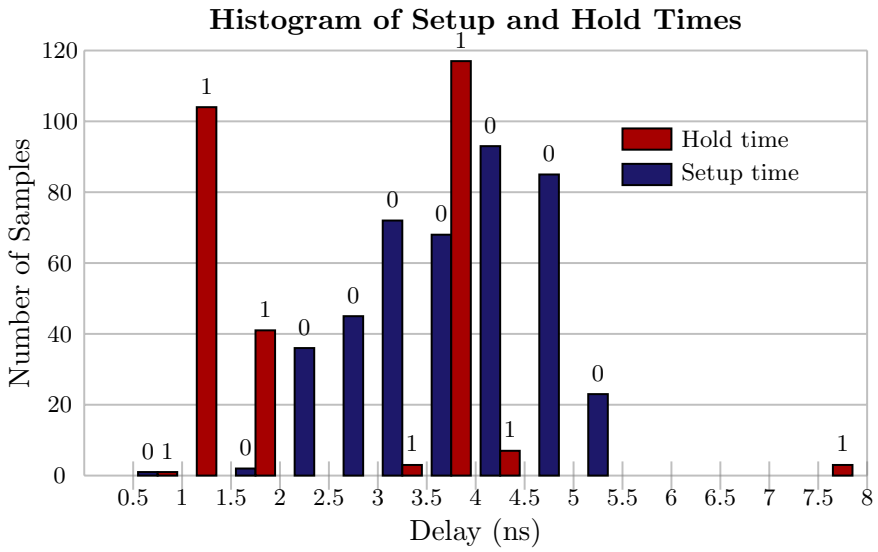

Figure 3.11. Slack for External RT Constraints

when provided timing path information, allows arbitrary timing paths, including those with cycles, to be evaluated using commercial static timing analysis tools such as PrimeTime. Total run times are larger than for a comparably clocked design as multiple iterations through the STA tool are required, but the runs can all be performed concurrently. The paper demonstrates algorithms performing full cyclic path timing validation of a 57K-gate 16 point FFT design in under 10 min.

CHAPTER 4

FAULT COVERAGE FOR RELATIVE TIMED ASYNCHRONOUS DESIGN

The testability of a synchronous design is measured automatically with commercial CAD tools. The commercial tools perform automatic test pattern generation (ATPG) and report the fault coverage with the generated test pattern. However, commercial tools such as TetraMax can not perform ATPG and evaluate the fault coverage on the asynchronous design because the control channels contain combinational feedbacks. Also, the commercial tools have no support toward generating test patterns for asynchronous circuits. Therefore, a flow that utilizes commercial CAD tools for testing asynchronous circuits is implemented.

The testing flow is discussed in the following two sections, testing the control channels and testing the data path. A 4-point FFT and a 64-point FFT, which will be shown in Section 6.1, are built to demonstrate the complexity and effectiveness of the testing flow.

4.1 Fault Coverage on the Control Channels

The fault detection on the control channels requires a tool that supports fault simulation on sequential circuits or circuits with combinational feedback loops. COSMOS, a switch level fault simulator, is adopted to perform the fault simulation of the control channels [61]. COSMOS takes a transistor level implementation of the design and translates it into a compiled simulation program [62]. The tool requires the design to be expressed in a flattened transistor level structure (".sim"). The conversion is performed using a structural Verilog to .sim script called `IRSIM` [63]. COSMOS performs fault simulation under a unit delay model.

In this study, the functional test patterns are generated manually and the fault coverages are reported on a various asynchronous designs including FIFOs, a 4-point FFT, and a 64-point FFT designs.

4.1.1 Performing Stuck-At Fault Simulation

COSMOS uses a single stuck-at fault model. The single stuck-at fault model covers most other faults such as bridging or multiple stuck-at faults. Both stuck-at 1 (SA1) and stuck-at 0 (SA0) faults are injected in the control channels. All the data paths and control channels outputs are observable for detecting faults. Additionally, no faults are injected into the data paths. Unlike the test patterns for the synchronous designs, a set of sequential test patterns is needed to test the asynchronous control channels. Specifically, the control channels operate with respect to its current state and the incoming input vectors. Certain input sequences would change the control channels to a certain state. A sequential test pattern is required, which needs to traverse all the internal states of an asynchronous design. Specific stuck-at faults of internal nodes can be detected at the specific internal state.

In this work, the sequential test pattern is generated with the understanding of the normal and illegal operations of the asynchronous design. The sequential test patterns are manually derived.

4.1.1.1 Single Controller Sequential Test Pattern

Table 4.1 shows seven sets of sequential test patterns corresponding to seven modes of operation of the single asynchronous linear controller. The D signal indicates the data change on the registers connecting to the linear controller. Reset mode toggles the control signals while holding the circuit in the reset state to detect reset failure. Left and right handshake mode only toggles either upstream or downstream handshake signals. Filled-then-drain mode starts with left handshakes that insert tokens into the controller until the circuit stall, then apply the right handshake to remove tokens from the pipe-stages. Concurrent mode puts the asynchronous controller under typical operation, which both sides of the handshake channels are switching concurrently once the response is seen. Next, a rising transition of the request is applied followed by activating the reset signal to detect stuck-at faults with respect to the reset state. Lastly, a false operation sequence that assert ra before lr to test the circuit response under incorrect input sequences. Moreover, the + sign appended to the signal name presents a rising transition while - sign indicates a falling transition.

4.1.1.2 Sequential Test Patterns

The set of sequential test patterns of a single controller is duplicated as many times as the number of pipe stages within an asynchronous design. The reset signal is a special case since it will only trigger once throughout the test pattern except for the Filled-then-reset

Table 4.1. The Sequential Test Pattern for a Linear Controller

Operation	Test Pattern Sequence
Reset	reset+, lr+, ra+, lr-, ra-
Left-handshake	reset-, D, lr+, lr-, D, lr+, lr-
Right-handshake	reset-, ra+, ra-, ra+, ra-
Filled-then-drain	reset-, D, lr+, lr-, D, lr+, lr-, ra+, ra-, ra+, ra-
Concurrent	reset-, D, lr+, lr-, ra+, ra-, D, lr+, lr-, ra+, ra-
Filled-then-reset	reset-, D, lr+, reset+
False operation	reset-, ra+, lr+

case. For example, a three-stage linear pipeline can hold up to three data tokens. Thus, lr has to toggle at least six times to fill up the pipeline. Once the pipeline is full, ra also has to toggle six times to empty the pipeline.

4.1.1.3 Fault Analysis of Linear Controllers

Four FIFO structures are constructed with liner controllers and tested using COSMOS. Again, stuck-at faults are only injected into the control channels. LC1, LC2, and LC4 are linear FIFO with the depth of 1, 2, and 4, respectively. LC4P is a parallel FIFO structure with a depth of 3. The test provides 89+% fault coverage among all the designs with the sequential test patterns shown in Table 4.1. The test pattern is duplicated as many times as the depth of the FIFO.

COSMOS reports two types of detected fault including hard faults (H) and soft faults (S). When a stuck-at-0 or a stuck-at-1 at a signal is causing a faulty output, the hard fault is reported. The soft faults are reported when the faulty output is observed while either 0→1→0 or 1→0→1 transitions occurs at the fault location. Hard and soft faults are considered as detected faults since both of them generates a faulty result. Undetected faults (UD) is reported as a number of total faults subtracts by detected faults.

The falling-edge controller has the property that data is valid only at the falling edge of lr. This controller is used as the asynchronous controller to construct the FIFOs and FFTs. The controller can be forced into its reset state by asserting the reset signal. Table 4.2 shows four FIFO designs with detected hard faults, detected soft faults, and undetected faults (UD). There are 2 UDs reported while testing a single controller (LC1). One of the UD is a SA0 fault since the state is unreachable. The other fault is because of the sequential redundancy of the internal state variable.

More faults are undetected once the design has a deeper pipeline. LC2 and LC4 are linear FIFO with the depth of two and four. LC4T is a parallel FIFO that contains C-elements,

Table 4.2. Fault Coverage of FIFOs

Design	H	S	UD	Total Faults	Fault Coverage
LC1	77	4	2	84	96.4%
LC2	148	9	7	164	95.7%
LC4	254	73	13	340	96.1%
LC4P	287	68	41	396	89.6%

as shown in Figure 4.1. LC4T has only 89% coverage because of the low coverage in the C-elements (Figure 4.2). The highlighted stuck-at faults are not detectable due to the inputs A and B switch at same step under unit-delay simulation. Input sequence of (A,B) as (0,1) and (1,0) are not reachable in this architecture. The C-element can be replaced with a testable C-element to detect those undetected faults [64].

4.1.1.4 Fault Coverage of FFT Designs

The structural 4-point and 64-points FFT designs are compiled with COSMOS. Faults are only injected into the control channels. The control channels include controllers, fork and join, and control steering elements such as the expander and the decimator. One assumption is made that all the register within the design can be scanned out. Thus, all the registers are considered as observable output. Table. 4.3 shows detected hard faults (H), detected soft faults (S), total faults injected, and fault coverage of the control channels. The control channels of 4-point FFT achieves 91.0% fault coverage while the 64-point FFT has 92.4% fault coverage.

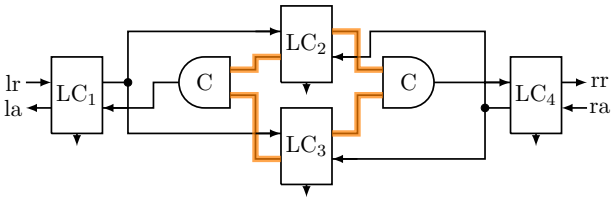

Figure 4.1. A 3-Deep Pipeline Contains a Broadcast Fork and a Join Element

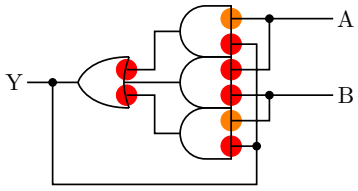

Figure 4.2. Circuit Implementation of a C Element. The Undetected Faults Have Been Highlighted in Red (SA1) and Orange (SA0).

Table 4.3. Fault Coverage of Control Channels

Design	Control channels				Run Time
	H	S	Total	Coverage	
FFT_4	517	131	712	91%	0.76 seconds
FFT_64	46159	3861	55560	92.4%	60 hours

The undetected faults in the control channels are mainly the C-elements, the clock pulse logic, and the internal gates of the controller. The increasing number of undetected faults in the control channels is because the unreachable states of the controllers increase. The fault coverage can be improved by applying Mr.Go to force the controllers to any certain states [65]. The runtime for the fault simulator is reported in Table 4.3. Because the simulation evaluates transitions on each transistor and the 64-point FFT consists of complex multipliers, the runtime for testing the 64-point FFT is 60 hours. The run time of the simulator is exponentially proportional to the number of transistors in the design. Parallel fault simulation is performed to improve the runtime of the tool. Multiple fault simulations are executed simultaneously. Other techniques such as deductive or concurrent fault simulation can help the run time [66],[67].

4.2 Fault Coverage on the Data Path

An approach was explored that isolates the data path and use commercial CAD tools to insert scan chains within the data path of the RT design [2]. The testing flow adopts this approach for testing the data path of the RT design.

There are ATPG untestable faults, which are associated with the reset signal since synchronous reset registers are used throughout the design. The stuck-at 1 with those reset signals can not be detected because they are active low resets. The fault coverages are 93.24% and 93.92% for 4-point and 64-point FFT, respectively. The registers can be substituted into asynchronous reset registers to further improve the fault coverage in the data path. The testing flow combines the detected and total faults from the control channels and from the data path. Table 4.4 shows the overall fault coverage of the FFT designs. Although both 4-point and 64-point FFT have low 90% fault coverage on the control channels, the number of faults injected into the data path are 10× than the ones injected into the control channels. In these examples, the fault coverage of the data path dominates the coverage from the control channels.

Table 4.5 shows the power, area and performance overhead for adopting a scan-chain for both 4-point and 64-point FFT designs. The 64-point design is implemented in the Artisan

Table 4.4. Fault Coverage of Complete Asynchronous Designs

Design	Faults	Control Channels	Data Path	Combined	Coverage
FFT_4	Detected	648	9798	10446	93.1%
	Total	712	10508	11220	
FFT_16	Detected	50020	632123	682143	93.6%
	Total	55560	673027	728587	

Table 4.5. Results Comparison

Design	**Cycle Time (ns)**	**Forward Latency (ns)**	**Power (mW)**	**Area (mm^2)**	**Energy per Point (pJ)**	**Relative Area**	**Relative Energy**
FFT-64	1.184	177.8	50.0	716,189	68.3	1.00	1.00
FFT-64-Test	1.162	178.7	79.2	1,235,667	106.6	1.73	1.56
FFT-4	2.267	20.4	28.0	95,628	64.2	1.00	1.00
FFT-4-Test	2.690	22.4	30.5	135,521	82.9	1.42	1.29

65nm library. The 4-point design is implemented using the IBM7RF 180nm library. Each design operated across 1024 samples. The designs without test used latches for sequential storage elements; the designs with the test used scan flops. This resulted in a 73% and 42% increase in area for the 64 and 4-point designs respectively. Likewise, the average energy per point increased 56% and 29% for the designs. There is about 10% performance degradation while adopting the test to the designs. All numbers are for postlayout design and simulation with power from extracted parasitics and the vcd activity file.

4.3 Summary

Bundled-data asynchronous designs are tested in two sections, the data path, and the control channels. The data path is inserted with a scan chain. ATPG was performed by commercial tools on the data path and achieves 93% coverage with area, power, and performance penalty. The control channels remain unchanged. A functional test was performed on the control channels. A set of sequential test patterns is generated. Multiple control structures, including FIFOs and FFTs, are tested with a fault simulator. The 4-point FFT has an overall 93.1% fault coverage while 64-point FFT achieves 93.6% coverage. Future work includes improve the fault coverage on the control network, improve the run time of the fault simulation, and performing delay fault on asynchronous designs.

CHAPTER 5

MACRO-BASED TIMING CONSTRAINT MAPPING TO TIMED ASYNCHRONOUS SYSTEMS

A macro based method for automatically mapping timing constraints onto system level asynchronous designs using commercial EDA tools is presented. This method is a path-based approach which uses netlist information and symbolic asynchronous timing constraints to create a Synopsys Design Constraints (sdc) file to drive system level design and optimization. The automatically generated constraint set enables timing driven optimization by commercial EDA tools for synthesis, place and route, and performance verification.

This work starts with a set of simple circuit macros and a design which deploys the macros. The macros are typically asynchronous handshake controllers or other sequential asynchronous design blocks, such as the LC handshake controller blocks in Figure 5.1. The macros have been designed and mapped to a standard cell library using AND/OR/AOI and other gates found in the library. The macros are then characterized using relative timing (RT) [1]. This provides three types of constraints for the macro. (1) Timing paths between two endpoints are expressed as either maximum or minimum delay paths. A pair of maximum and minimum paths are linked to form a relative timing constraint; delay targets and margins are annotated to the paths. (2) Constraints that cut the timing graph of a cyclic sequential circuit into a directed acyclic graph the do not cut constrained timing paths. (3) Constraints which do not allow synthesis and place and route tools to re-synthesize or modify the library gate footprints of the macro.

This section of the thesis presents algorithms which take circuit macros with their associated constraints and a design which employ the macros to produce an sdc file to use commercial EDA tool such as Synopsys Design Compiler or IC Compiler for system level synthesis and place and route.

This work is divided into the front-end and backend. The frontend identify all control

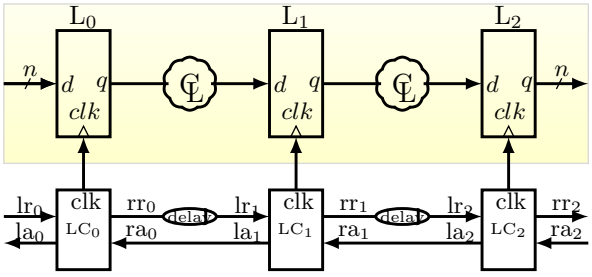

Figure 5.1. Three Stage Pipeline Design

and data connections. The backend uses the identified connection and the macro characterization information to construct a search map and apply symbolic timing constraints onto the system level design. This algorithm enables designing large asynchronous system by automating the application of timing constraints. A multirate Fast Fourier Transform circuit is used to demonstrate the benefit of this algorithm.

5.1 Related Work

There are several approaches to asynchronous circuit design. Each approach bears different levels of complexity in terms of logic composition, timing assertion, and circuit validation. However, a predominant issue of asynchronous design techniques is CAD flow support. Many approaches rely on custom languages, custom libraries, tools, and flows to generate asynchronous designs [68],[69]. Synthesis, place and route, and timing optimization are under the control of the custom tool suite. Several other approaches attempt to leverage existing EDA tools and flows to build asynchronous circuits [24],[25],[40],[70],[71]. However, these tools either relied upon manual timing driven optimization or do not specified a method of applying timing constraints.

The presented work builds upon RT characterized and verified asynchronous macros [44]. Timing constraint templates, supplied from the macros, are applied to asynchronous systems in order to enable automatic timing driven synthesis and place and route using commercial EDA tools.

5.2 RT Constraint Template

5.2.1 End Point Specification Format

Eqn. 1.1 shows the generic form of the timing relation between the path $\mathsf{pod} \mapsto \mathsf{poc}_1$ and $\mathsf{pod} \mapsto \mathsf{poc}_0$. Such RT constraints must be mapped onto specific timing paths that must hold when a design macro is used in a system level architecture. Observe the timing

constraints shown in Figure 5.1. The same timing equations in the caption apply to all instances of the design. These handshake macros communicate with other upstream blocks (with the ack signal), downstream blocks (with the request signal), and the local latch or flip-flop.

A representation is constructed for specifying the relative location in a pipeline of the current macro under evaluation. Rather than enumerate pipeline stages, as is implied in Figure 5.1, a *relative* instance identification mechanism was developed. The current macro is identified with an instance label "$i1". Macros and memory elements upstream in a pipeline (accessed through "ack" handshakes) are referenced as "$i0" design blocks, and downstream pipeline elements (accessed through "req" handshakes) are referenced as "$i2" macros. Registers are typically connected through an additional "clk" output port from the macros. They are identified separately from asynchronous macros by appending a capital R to the back of the label.

Using this mechanism, general RT constraints from Eqn. 1.1 can be written that identify timing endpoints relative to the current macro ($i1). Eqn. 5.1 expresses the constraint in the caption of Figure 5.1 in our new format that identifies timing endpoints. This says that the maximum delay plus margin m from the local LC macro's *req* input to the downstream register's data input ($\$i2R/d$) must be less than the minimum delay from the local LC macro's *req* input to the downstream register's *clk* input. (Note that the macro pin label for *req* has been renamed *lr*.)

$$\$i1/lr \mapsto \$i2R/d{+}m \prec \$i2R/clk \tag{5.1}$$

5.2.2 Timing Path Constraint

In order to represent Eqn. 5.1 in a symbolic sdc constraint format that can be mapped to a design, Eqn. 5.1 is translated into the set of sdc constraints shown in Table 5.1. Note that due to cycle cutting to create a DAG, the paths are broken into two segments. Constraints (1) and (4) are derived from path $\$i1/lr \mapsto \$i2R/d$. In this example, the path from $\$i1/lr \mapsto \$i2R/d$ is broken into two sdc constraints in order to individually specify the target clock tree skew and data path delay. The combination of constraint (3) and (5) forms the RT constraint for path $\$i1/lr \mapsto \$i2R/clk$. Constraint (3) from Table 5.1 controls the optimization of the gates driving $i1 *req* signal and the size of delay element inserted between the $i1 and $i2 controllers, while constraint (5) specifies the maximum clock tree delay of the downstream controller.

Table 5.1. A Subset of the RT Constraints Template for the LC Circuit (Figure 5.2)

set_max_delay $dpdelay -from $i1R/$CLK -to $i2R/$D	(1)
set_max_delay $cdelay_max -from $i1/lc3/A1 -to $i2/$lr	(2)
set_min_delay $cdelay_min -from $i1/lc3/A1 -to $i2/$lr	(3)
set_max_delay $clk_delay -from $i1/lc1/A0 -to $i1R/CLK	(4)
set_max_delay $clk_delay_i2 -from $i2/lc1/A0 -to $i2R/CLK	(5)
#margin $dpmarg -from $i1R/$CLK -to $i2R/$D \ -from $i1/lc3/A0 -to $i2R/$CLK	(6)
set_max_delay $idelay_max -from $i1/lc5/A1 -to $i1/lc1/A2	(7)
set_min_delay $idelay_min -from $i1/lc5/A1 -to $i1/lc1/A2	(8)

The margin m is specified as the `#margin` command in constraint (6). The `#margin` command is not interpreted by the EDA tools but serves two purposes. It identifies two related paths in a relative timing constraint and specifies the size of the timing margin. An additional maximum delay is specified by constraint (2) to maintain the performance of the handshake network by ensuring that the minimum delay value does not grow to be too large. There are variables included in the Table 5.1, such as $dpdelay, $dpmarg, $clk_delay, $cdelay_min, $cdelay_max, $idelay_min, and $idelay_max. The variables serve as the delay target to determine circuit performance. Selecting the values of these variables depends on the process node and the entire system. The system timing or circuit performance are beyond the scope of this work.

There are external and internal timing path constraints to consider used in timing specifications. The external timing path constraints start in one macro instance (e.g., $i1) and ends in another macro (e.g., $i2 or $i0). The internal timing path constraints, on the other hand, would have $i1 as its start and end points. The external timing path constraints are timing relations between the control path and data path in a system. The internal timing path constraints are required for state-holding and to prevent the asynchronous controller from entering an unwanted state. Constraints (7) and (8) are internal timing path constraints that guarantee internal feedback is settled before changes in external signals.

5.2.3 Timing Graph Specification

Part of the relative timing characterization of an asynchronous macro is to represent the circuit timing graph as a directed acyclic graph. The algorithm that performing cycle cutting has been discussed in Chapter 2. Often, creating a circuit timing graph that is a DAG requires that some relative timing constraints be composed of multiple segments. The timing endpoints of the critical timing paths are cut from the timing graph. Additionally, the timing graph is automatically cut within registers since the “clk” pin of the register is always

the timing end point in synchronous systems. However, combining segmented timing paths introduces timing inaccuracies while performing timing-driven synthesis. Since the path is separated, there is no timing relation such as rise delay or fall delay at the break point. Segmented paths also reduce the optimization capability of timing driven synthesis because delay cannot be re-distributed across path segments.

5.2.4 Function Constraint

The precharacterized asynchronous cell macros are verified for hazard freedom and timing relationships based on the RT constraints and delays specified in the timing path constraints. Commercial EDA tools can modify the internal gates and functions of the asynchronous cell macros in their optimization functions. The `set_size_only` constraint is required to prevent the commercial EDA tools from modifying the gate footprints of the macros.

5.2.5 Symbolic Pins and Keywords

Keywords and symbolic pins are introduced to identify ports and paths of a macro in a general and portable manner.

5.2.5.1 Symbolic Pins

Symbolic pins are used to specify endpoints of timing paths in the controller network and the data path. The $lr, $la, $rr, and $ra pins represent the left (upstream) request, left acknowledge, right (downstream) request, and right acknowledge handshake signals. The $clk keyword represents the clock key of the register banks, and $Q and $D symbols represent data inputs and outputs of register banks. The application uses symbolic pins to direct mapping to external instances.

5.3 Constraint Mapper

The constraint mapping application is written in C++ and uses STDIO commands to interact with Synopsys Design Compiler and Cadence RTL Compiler. The mapped results are printed out to a sdc file, which can be interpreted by synthesis, place and route, and timing validation tools.

5.3.1 Path Reporting and Parsing

The application starts by identifying all the asynchronous macros used in the design. This macro set, along with the design and the macro characterization information, is used

to identify all relative timing path instances in the design. The procedure reports path instances in the system-level design using commercial synthesis tools.

The path finding function first locates the hierarchical instances of the asynchronous marcos in the system. The module-instance relation for the macros is reported.

Paths between asynchronous macros and either macros or registers are found by iterating with Design Compiler or RTL Compiler over all macro symbolic input pins. The pin instances are found with the `all_fanin` function and matched to the symbolic pins. Macro to macro paths are formed if the start pin is from a macro while register to macro paths are formed when the start pin is from a register pin. The function calls `all_register -data_pins` to obtain all the register's data input pins and then calls `all_fanin` to identify register to register paths. Finally, the function calls `all_register -clock` to obtain all the register's clock pins, and then uses `all_fanin` to create register to macro connections.

The reported paths are filtered into four output categories: macro to macro paths (*c_path*), macro to register paths (*clk*), register to macro paths (*dc_path*), and register to register paths (*d_path*). Table. 5.2 shows the result of running the algorithm for reporting and filtering the resultant paths on this example. The *dc_path* is empty since there exists no direct path starts from a register and ends at a controller.

The reported paths are stored as directed edges that are indexed by both the beginning and ending vertexes.

5.3.2 Mapping

The mapping process is performed by iterating through all the controller macro instances within the design. For each controller type, the necessary cycle cutting and optimization

Table 5.2. Connectivity Between LC_0 - LC_2

c_path	$LC_0/rr_0 \mapsto LC_1/lr_1$ $LC_1/la_1 \mapsto LC_0/ra_0$ $LC_1/rr_1 \mapsto LC_2/lr_2$ $LC_2/la_2 \mapsto LC_1/ra_1$
clk	$LC_0/clk \mapsto L_0/clk$ $LC_1/clk \mapsto L_1/clk$ $LC_2/clk \mapsto L_2/clk$
d_path	$L_0/q \mapsto L_1/d$ $L_1/q \mapsto L_2/d$
dc_path	N/A

disabling commands are printed out once. Internal timing constraints have both endpoints inside the same local $i1 macro. These paths are mapped using the current controller instance and symbolic information associated with the controller. Using this information, controller paths are located by performing an exhaustive depth-first search to all controllers.

5.4 Experiments

The experiments are developed using IBM's 180nm 7RF process. The circuits are designed in behavioral Verilog, synthesized using Synopsys Design Compiler. Place and route is performed with Synopsys IC Compiler and simulated with ModelSim with post-layout parasitic back-annotation. Every timing path constraints contain positive slack, as well as the asynchronous systems, are functioning. The asynchronous controller used in these experiments is in Figure 5.2 and consists of 20 symbolic timing constraints, 8 optimization constraints, and 6 cycle cutting constraints.

5.4.1 64-Point Multirate FFT

The asynchronous FFT design is hierarchical and fully pipelined. There is a total of 864 asynchronous controllers and associated latch banks that construct a 28 deep pipeline design. There is the total of 7000+ constraints generated with 5s of run-time. The design has three levels of hierarchy and is constructed using 4-point FFTs and 16-point FFTs. Relative timing constraints are applied within FFT modules and in between 4-point and 16-point FFT blocks as well as at the top level I/O ports. Table 5.3 shows RT constraints in this design that specify controllers across different hierarchy levels. Constraints (1) and (2)

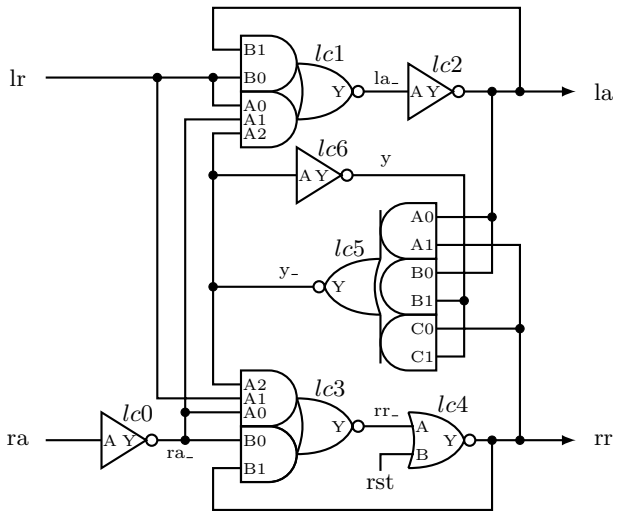

Figure 5.2. LC Circuit Implementation

are paths that go from the top hierarchy level to the inside of a 16-point FFT. Constraints (3) and (4) are for inside a 16-point FFT to the 4-point FFT inside the 16-point FFTs.

5.5 Summary

The approach uses asynchronous controller macros which have been precharacterized. The characterization provides information to create a timing graph represented as a directed acyclic graph, prevent macros from being resynthesized, and provide correct timing and signal sequencing information. The application uses asynchronous design macros as well as system design connectivity information to generate an sdc constraint set. The constraint set enables clocked EDA tools to synthesize and optimize an asynchronous design. In the past, timing constraints have been applied manually, which is time-consuming, error-prone, and has limited the feasibility for large scale asynchronous designs. The constraint mapping has been applied to a 64-point FFT and enables designers to quickly and easily use asynchronous macros in large scale digital systems.

Table 5.3. A Subset of RT Constraints for a 16-Point FFT Design

set_min_delay $cdelay_min -from tk00/lc3/Y -to F16_0/tk0/lc1/A1	(1)
set_max_delay $dpdelay -from P00/clk -to F16_0/P0/d	(2)
set_min_delay $cdelay_min -from F16_0/tk0/lc3/Y \ -to F16_0/F4_0/tk0/lc1/A1	(3)
set_max_delay $dpdelay -from F16_0/P0/clk -to F16_0/F4_0/P0/d	(4)

CHAPTER 6

CASE STUDIES

Various sizes of designs are developed to demonstrate the effectiveness of the backend robustness tools and flows. This chapter describes multiple frequency designs which are constructed and used as the benchmark circuits.

6.1 Synchronous and Asynchronous 64-Point FFT Design[2]

A case study exploring multi-frequency design is presented for a low energy and high performance FFT circuit implementation. An FFT architecture with concurrent data stream computation is selected. An asynchronous and synchronous implementations for a 4-point, a 16-point and a 64-point FFT circuit were designed and compared for energy, performance, and area. Both versions are structurally similar and are generated using similar application specific integrated circuits (ASIC) CAD tools and flows. The asynchronous design shows a benefit of 2.4×, 2.4× and 3.2× in terms of area, energy and performance respectively over its synchronous counterpart. The circuit is further compared with a low power design that is not streaming and shows a 0.4×, 4.8× and 32.4× benefit with respect to area, energy, and performance.

6.1.1 Key Contribution

- A synchronous FFT that operates with multiple frequency domain is developed.
- A golden model is constructed using Matlab.
- Analysis is performed on various published FFT works to present the area, power, and performance benefits.

[2]This section has been published in DATE, 2013 [9]. © 2013 IEEE. Reprinted, with permission, from William Lee, Vikas S. Vij, Anthony R. Thatcher, Kenneth S. Stevens, "Design of Low Energy, High Performance Synchronous and Asynchronous 64-Point FFT," in Proceedings of the Conference on Design, Automation and Test in Europe, March, 2013.

6.1.2 FFT Architecture

The FFT is an algorithm that requires global dependencies, but it can be derived in a multirate form that allows a hierarchical representation as shown in Eqn. 6.1 [72]. This multirate architecture exploits performance from concurrency by allowing parallel computations to occur at reduced frequencies. The equation represents N_2 FFTs using N_1 values as the inner summation, which are scaled and then used to produce N_1 FFTs of N_2 values. This representation has the advantage that it takes a high frequency stream and decimates it so that each of the internal FFTs operate at a lower decimated data stream frequency. This allows the architecture to simultaneously have lower energy and higher performance.

$$X_{m_1}(m_2) = \sum_{n_2=0}^{N_2-1} \left[W_N^{m_1 n_2} \sum_{n_1=0}^{N_1-1} x_{n_2}(n_1) W_{N_1}^{m_1 n_1} \right] W_{N_2}^{m_2 n_2} \tag{6.1}$$

The general architecture derived from Eqn. 6.1 is shown in Figure 6.1. There are three architectural control structures: a decimator, expander, and crossbar block. Each of the N_i blocks can be another hierarchical instance of the design where i is the size of the FFT performed in that block. The values of $N_1 \times N_2$ equals N_1 or N_2 at the higher level in the hierarchy.

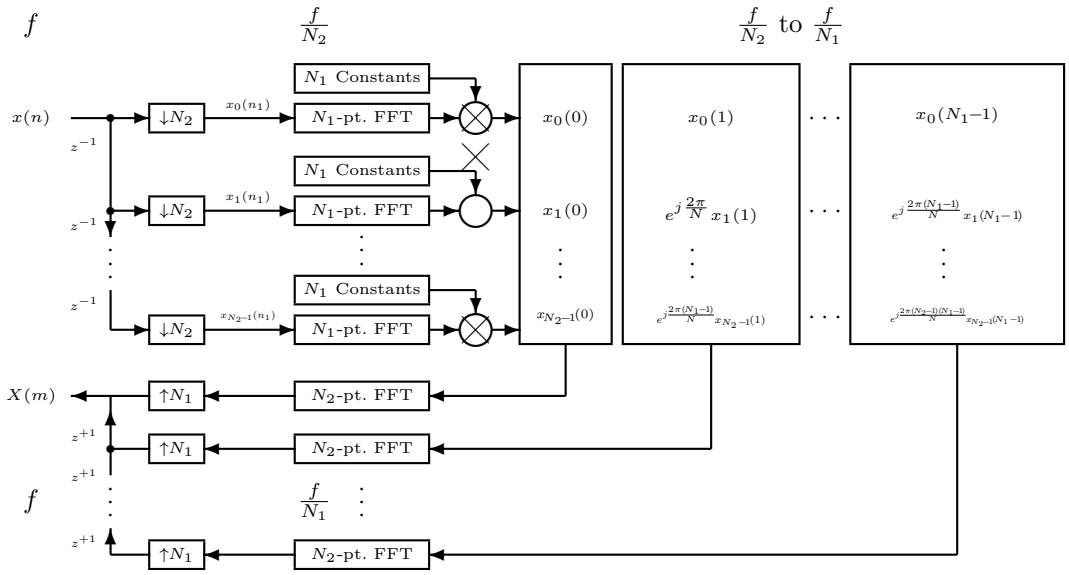

Figure 6.1. Multirate FFT Architecture [72]

The decimator block down-samples the input stream [73]. For a sampled signal $x(n)$, the output of the M-fold decimator is given by $y(Mn)$. The sampling of the N_2 decimator is arranged in a regular repeating fashion where the first sample is steered to the first output stream, the second to the second stream and so on. The M^{th} item is steered back to the first stream. This effectively produces M parallel streams operating at $1/M$ the frequency of the input.

The expander block is the dual of the decimator block. They take M low-frequency streams and up-sample by combining them into a stream that has an M-fold higher frequency. In the FFT architecture, the expander operates on a stream of data $x_0(m_2), \ldots, x_{N-1}(m_2)$ reproducing a stream at the original frequency and in the correct functional order for the algorithm.

Product blocks multiply a stream of results coming from the N_1 point FFT units by a set of constant values. Both constants and results are complex numbers, requiring four multiplications and two additions per sample. The constants are calculated by $W_N^{m_1 n_2}$, where $m_1 = 0, \ldots, N_1 - 1$ and $n_2 = 0, \ldots, N_2 - 1$.

The crossbar switch maps results from the product block to the N_2 FFT units. The N_2 FFT units take a transform of time displaced Fourier transform samples. Each N_1-point FFT provides one data sample to each of the N_2-point FFT units. The first row of the N_2 FFT units takes the first sample from each of the N_1 rows, the second row the second sample, and so on. This is implemented by performing an N_2 up-sampling followed by a N_1 down-sampling. Another solution is to steer the data to N_2 N_1-way decimators, followed by N_1 N_2-way expanders. Decimator sequencing here is different than that of the top level block because it steers the first N_2 samples to each row before moving onto the next row.

6.1.3 FFT Design

Multifrequency asynchronous and clocked 64-point FFT designs are implemented from the architecture block diagram shown in Figure 6.1. Both designs are hierarchically decomposed at the top level such that $N_1 = 16$ and $N_2 = 4$. The 16-point FFT implementations are also hierarchically decomposed with $N_1 = N_2 = 4$. The terminal hierarchical nodes in the designs are the 4-point FFT blocks since it can be implemented with simple add and subtract operations due to the value of the constant data values. There are four frequency domains in this design. The frequency of the incoming data is f, which gets decimated to derive $f/4$, $f/16$, and $f/64$ frequencies.

The data path for all the designs is specified behaviorally with the control being the only

differentiating point. The asynchronous design is implemented as a bundled data pipeline (Figure 3.1). The LC block that controls timing and sequencing is a 4-phase handshake protocol similar to that in Figure 2.1. This cell generates a local clock signal to control the pipeline stage based on the handshake with the adjacent handshake controllers.

These designs operate on fixed-point data. The input and output are 32 bits wide, with the upper 16 bits representing the real value and the lower 16 representing the imaginary value. The fields use two's complement representation of signed numbers that are decimal values less than or equal to plus or minus one. The first four bits are used for the whole part of the number and the rest 12 bits for the fractional part.

6.1.4 Synchronous Design

The synchronous FFT is designed using the architecture in Figure 6.1. The 4-point synchronous FFT design is a six deep pipeline. The 16-point and 64-point FFTs are 19 deep pipelines and 32 deep pipelines, respectively.

A decimator and a expander are built (Figures 6.2 and 6.3). These two modules act as a 2-flop synchronizer while crossing between different frequency domains.

Figure 6.2 shows the clocked 4-way decimator. The design consists of a high frequency

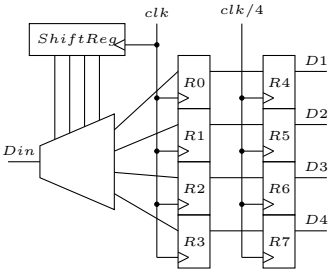

Figure 6.2. Synchronous Decimator

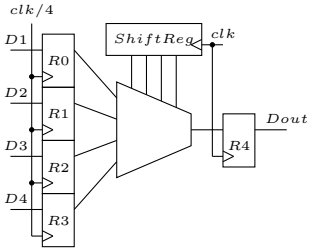

Figure 6.3. Synchronous Expander

register bank and a low frequency register bank, a clock divider, and a shift register to track the relationship between the two clocks. The shift register must be properly initialized in relation to the global state of the circuit based on data arrival to ensure proper data steering. The data is incrementally stored into the high frequency register bank. At the low frequency clock the data is then shifted into the low frequency register bank, where it is sampled at a $1/N_2$ frequency. The expander in Figure 6.3 is the dual of the decimator. The parallel data stored into a low frequency register is streamed and stored in the output register based on the higher frequency clock. The channel selection is dependent on the shift register and requires properly initialization similar to the decimator.

Additional timing constraints are needed for the CAD tools to properly synthesis the multi-frequency design. The fastest clock (f) is ditributed into f/4, f/16, and f/64 slower clocks. There are multiple clock dividers at every level of hierarchy. Extra timing margin is required for the clock network to accommodate the clock skew and jitter. The `sdc` constraint `set_generated_clock` creates correct timing relation across frequency domains.

6.1.5 Asynchronous Design

The first step in an RT asynchronous design is to create and characterize the handshake elements. This design uses four circuit elements: a linear pipeline controller (LC) (Figure 2.1), a 2-input Fork/Join element, the decimator, and expander. The LC circuit interfaces two pipeline stages by controlling the protocol between the stages and storing one data word (Figure 3.1). The fork (Figure 6.4) broadcasts a request from a sender to two receivers. The ack from the two receivers is synchronized with a C-element before being passed on to the sender [74]. The join element contains the same logic and is dual of the fork. Requests from two senders are first synchronized before being sent to a receiver while the ack signal from the receiver is broadcasted to both the senders.

A 4-way asynchronous decimator is designed and implemented. It consists of a ring connected shift register with one bit asserted to steer the requests to four different pipelines based on the value in the shift register. The req and the ack signals are active high. Since only one acknowledgment is active at a time, the four ack signals are passed through an

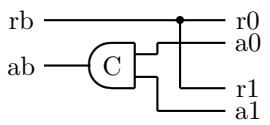

Figure 6.4. Fork/Join Template

OR gate. Values in the shift register update when the input request goes low. The circuit is characterized for its timing constraints. As long as the shift register can change in one half cycle time (before the next req occurs), this logic will operate correctly. Note that this block adds a 2-input AND gate delay on the request path and a 4-input OR gate delay on the acknowledge path. This is the only overhead of the decimator, and adds approximately 8 gate delays to the cycle time of the architecture, allowing it to operate at approximately a 16 gate delay cycle time. This resulted in a frequency that was close to 1.3 GHz, which we deemed as a sufficiently fast performance target. The design is shown in Figure 6.5.

The design of the asynchronous expander in Figure 6.6 is similar to the decimator. It includes an N_i-bit ring connected shift register and some combinational gates to select the data and control signals to be driven to the output channel.

Once these blocks were designed the top level asynchronous architecture was built by simply composing the pipeline control and datapaths together. We employed a hierarchical structural design style which was almost identical to drawing and connecting block level schematics for the design. In this method, a functionally correct design was hierarchically

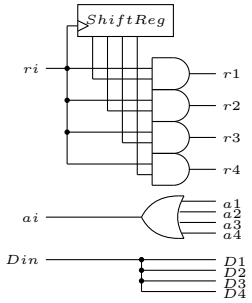

Figure 6.5. Asynchronous Decimator

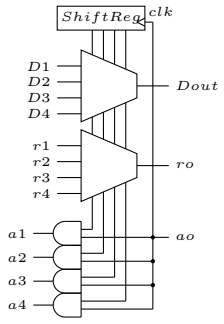

Figure 6.6. Asynchronous Expander

designed and validated for performance and correctness. First a simple 4-point FFT was built, which was used to build a 16-point FFT, and then these components were integrated into the 64-point FFT.

The design of the pipeline was almost as simple as drawing the figure on the paper. The dataflow graph of a 4-point FFT is shown in Figure 6.7. The pipelined asynchronous control logic for that design is shown in Figure 6.8. The butterfly network and first set of adders are between stages LC1 and LC2, the second butterfly network and adders are between stages LC2 and LC3, and the last network convolution is between stages LC3 and LC4. Following is a code snippet from the design to give you a flavor of the RTL (Figure. 6.9). Some liberty is taken in the syntax to compress the example. This shows a pipeline stage at the input of the design that feeds into the next stage of 16-point FFTs. Each pipeline stage and the structural block are similarly designed.

Performance and functionality optimizations in an asynchronous design are somewhat independent operations. A design that is functionally correct can be created relatively quickly. However, particularly for multi-frequency designs, some effort is needed to balance

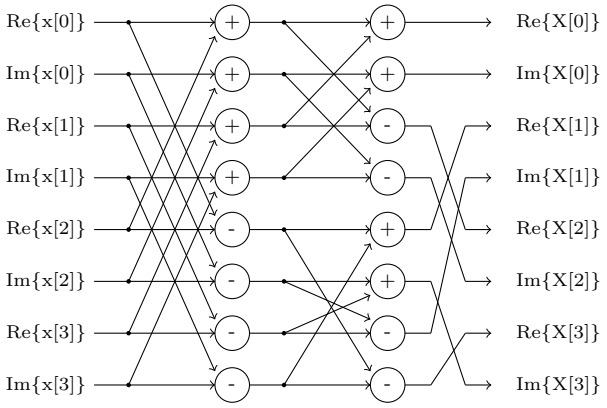

Figure 6.7. Data Flow Graph of 4-Point FFT Calculation

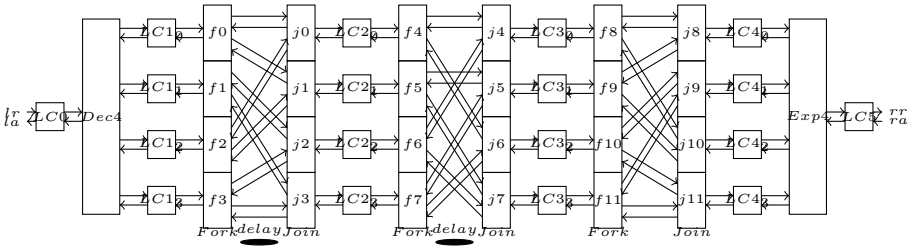

Figure 6.8. 4-Point FFT Design

```
module FFT_64 (ri, ai, DI, ro, ao, DO, rst);
   input  [`WORD_SIZE-1:0] DI; ...
   // input pipeline
   linear_control  LC0 (.lr(ri), .la(ai), .rr(p0r),
           .ra(p0a), .ck(ck0), .rst(rst));
   latch   P0   (.d(DI), .clk(ck0), .q(P0D0));

   decimator_4 D4_0 (.DI(P0D0), .D1(P0DT1), .D2(P0DT2),
      .D3(P0DT3), .D4(P0DT4),
      .ri(p0r), .ai(p0a), .rst(rst),
      .r1(p0rt1), .r2(p0rt2), .r3(p0rt3), .r4(p0rt4),
      .a1(p0at1), .a2(p0at2), .a3(p0at3), .a4(p0at4));

   // The FFT_16 modules.
   FFT_16 F16_0 (.ri(p0rt1), .ai(p0at1), .ro(p1rt1),
     .ao(p1at1), .DI(P0DT1), .DO(P1DT1), .rst(rst));
```

Figure 6.9. RTL of FFT Design

the cycle times and pipelining to optimize performance. This is very different from clocked design where performance and pipelining are essential for correct functionality, and part of the initial specification.

A primary aspect of optimizing performance of an asynchronous architecture is to calculate the critical paths and focus on those. Experimenting with the power-performance tradeoffs allowed us to quickly identify the critical paths in the asynchronous design. Due to the multirate architecture, it was not the complex multipliers or adders that operate at 1/4, 1/16, or 1/64 the input frequency. Rather, the top level decimators and expanders limit the operating frequency of the design. We, therefore, focused on designing high throughput decimators and expanders.

The performance optimizations will be illustrated with the 4-point FFT pipeline shown in Figures 6.7 and 6.8. From a correctness perspective, the data through the expander could pass straight through the expander through the butterfly network to the adders. However, this would create too long a cycle time at the decimators. Increased performance is obtained by adding pipeline stages before and after the decimators and expanders since they are the critical paths in the design.

The next power-performance optimization of the asynchronous 4-point FFT design was to determine the frequency target for the smallest area and lowest power adder in the given technology. A 16-bit ripple carry adder needed about 860ps in this technology, so that became the performance target of the 4-point FFT design. This was less than the time available for the computation (3ns in the top level 64-point block and 12ns in the 16-point blocks at 1.3 GHz operating frequency). However slowing the operation down beyond 860ps

simply adds more area, energy, and latency to the control path.

An additional performance critical aspect of a design is due to pipeline synchronizations. Adding or removing pipeline stages in an asynchronous design can be employed to remove forward and backward stalls in an architecture. This has been referred to as "slack matching" in the asynchronous literature [15]. Therefore, a version of the performance critical 4-way decimator was designed as a 2×2 pipelined decimator to increase throughput and reduce sensitivity to backward stalls. Likewise the crossbar and 16-way expander in the 64-point design of Figure 6.10 have been pipelined.

The asynchronous design was built using "natural" pipelining for each block with pipeline performance targets based on the top level architecture. For example, pipeline stages exist between the adders of the design whereas they can be removed from a performance perspective. A few modifications to the original pipeline structure have been made to improve area in the "async-opt" design.

6.1.6 Results

These circuits use the Artisan academic library in IBM's 65nm 10sf process. The circuits were designed in behavioral Verilog, synthesized using Design Compiler, and place and routed using SoC Encounter. Circuits were simulated for timing and functional correctness

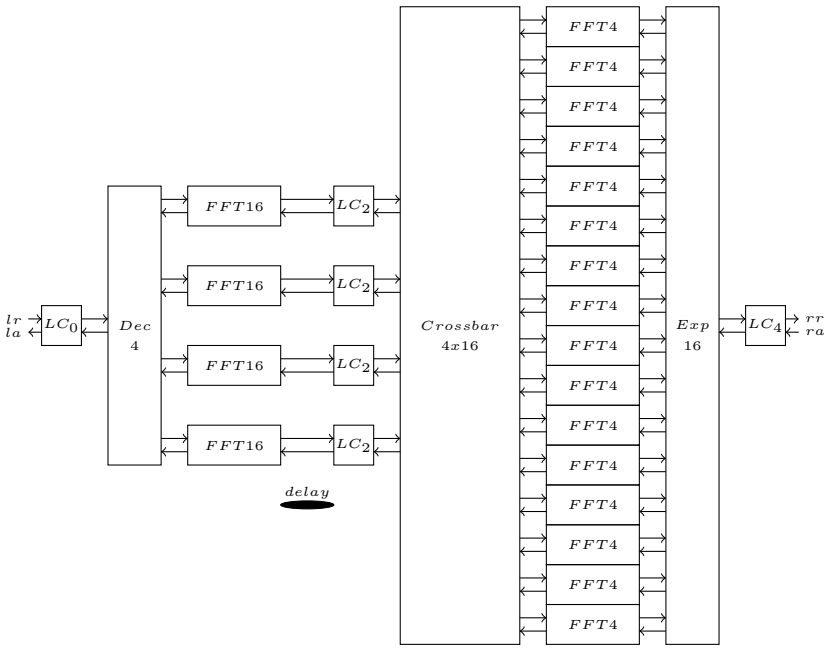

Figure 6.10. 64-Point FFT Design

using Modelsim with post layout parasitics back-annotated. Testing was performed using pre-defined input vectors which included 1024 random numbers. Both 64-point FFT circuits have less than ±0.3% variation as compared to MATLAB FFT computation. Various performance parameters including forward latency, cycle time, and throughput were also generated from the simulation along with vcd (value change dump) file. The simulation vcd file along with the parasitics of the place and routed design was used to calculate the power numbers for each design by PrimeTime.

Tables 6.1 and 6.2 summarize these multirate designs against several other designs. These 16-point implementations are compared against a design that is similar in architecture [75]. The 64-point benchmark is a low power Texas Instruments design [76]. Performance is measured as the time to completely process 1024 samples.

The simplicity of making architectural and performance modifications to the asynchronous design allowed us to quickly explore a simple area improvement to our asynchronous architecture. The 64-point architecture contains four 16-point FFT's. Each of these contain three complex multipliers operating at 1/16 the top level frequency. At this frequency, the multipliers could be shared, removing 8 complex multipliers from the design (Async-opt) resulting in an overall 18% area reduction. This modification had a minor positive affect on performance and negative affect on latency and energy per point. Other modifications that reduce area at little or no energy and performance cost can also be explored, as well as other optimizations based on target versus required frequencies. Asynchronous designs are particularly amenable to such architectural explorations.

For comparison, results in these tables are optimistically scaled to an equivalent for 65nm technology node by using theoretical constant-field scaling assuming the scaling factor $\kappa = 1.43$ per node (let $s = 1/\kappa = 0.7$) [77]. This results in delays in the tables multiplied by s, s^2, and s^6 for the 90nm, 130nm, and 600nm nodes. Energy values are scaled by s^3, s^6, and s^{18}. Area reduces by s^2 per generation.

The biggest advantage of this multifrequency architecture against the others comes in the form of throughput. These designs can sustain a rate of one data point per clock cycle, at a relatively constant frequency regardless of the point size. The asynchronous design also provides a substantial reduction in latency. From an idle start, the asynchronous 16 and 64-point designs can complete processing 1024 samples over 8 and 32 times faster respectively than the benchmark designs. Multifrequency design also shines in energy per sample. The asynchronous designs consume approximately one-fourth the energy per sample of the competitors. This 16-point pipelined design is less than half the size of this comparable

Table 6.1. The 16-Point FFT Comparison Result (* Constant Field Scaled to 65 nm Technology)

Design	Tech. nm	Points – Samples	Word $bits$	Clock MHz	1K-point Exec. Time μs	Power mW	Energy/point pJ	Area Kgates	Exec.Time Benefit	Energy Benefit	Area Benefit
This Design (Async)	65	16-1024	16	1,274	0.83	30.9	25.05	54	8.32	3.93	2.73
This Design (clock)	65	16-1024	16	588	1.73	24.7	41.83	71	3.98	2.35	2.07
Guan [75]	130	16-1024	16	653*	6.91*	14.6*	98.33*	147	1.00	1.00	1.00

Table 6.2. The 64-Point FFT Comparison Result (* Constant Field Scaled to 65 nm Technology, + Nominal Process Voltage)

Design	Tech. nm	Points – Samples	Word $bits$	Clock MHz	1K-point Exec. Time μs	Power mW	Energy/point pJ	Area μm^2	Exec.Time Benefit	Energy Benefit	Area Benefit
This Design (Async-opt)	65	64-1024	16	1,357	0.87	69.4	59.23	395	32.24	4.51	0.47
This Design (Async)	65	64-1024	16	1,316	0.87	65.5	55.65	479	32.39	4.80	0.39
This Design (Clock)	65	64-1024	16	667	2.76	50.2	135.30	1,160	10.21	1.97	0.16
Baireddy [76]	90	64-4096	–	514*	28.18*	9.7*	266.95*	186*	1.00	1.00	1.00
Chong (1.1V) (Async) [78]	350	128-128	16	–	1,633.64*	–	4.45*	45*	0.02	59.98	4.12
Chong (3.5V) (Async)+[78]	350	128-128	16	–	513.43*	–	45.06*	45*	0.05	5.92	4.12
Baas (3.3V) [79]	600	1024-1024	20	1,470*	3.53*	11.7*	40.31*	679*	7.98	6.62	0.27
Baas (5V)+[79]	600	1024-1024	20	2,228*	2.33*	40.7*	92.55*	679*	12.10	2.88	0.27

clocked hierarchical pipelined design. When comparing this design against the low power 64-point design from Texas Instruments, the clocked design and area optimized asynchronous designs consume six and two times the area. This points out the very different design targets and architecture styles. Their architecture shares the computation units for area efficiency at a cost of higher energy and much lower performance.

The Async-opt design is significantly better than the clocked design of the same architecture. The 64-point design shows an improvement of 2.28× the energy per data point and 3.16× the performance while costing only one-third the area.

Accurately comparing FFT designs with different point sizes, technology nodes, and architectures is challenging. Table 6.3 therefore provides a design comparison based on three metrics - *Benefit Product* [79] and $e\tau^2$ using Baireddy as the reference, and *Normalized FFTs per Energy* [78]. Benefits Product is the product of the area, energy, and execution time. Baas and Chong employ voltage scaling to quadratically reduce energy. Energy times square of the execution time ($e\tau^2$) provides a reference that is independent of voltage scaling. *The Normalized FFTs per Energy* metric largely disregards performance. Those results are normalized to the 350nm node to produce the same values as reported in [78].

6.1.7 Summary

Multirate asynchronous and a synchronous 16 and 64-point FFT circuits were implemented and compared against published FFT designs. A novel relative timing design flow which enables the use of precharacterized sequential templates with synchronous CAD tools and flows to develop asynchronous circuits is used. This work demonstrated that the flow can be efficiently applied to a large asynchronous design.

Table 6.3. Design Comparisons (+ Nominal Process Voltage)

Design	$e\tau^2$ Advantage	Normalized FFTs per Energy [78]	Benefit Prod. [79]
This Design (Async-opt)	4,683.38	17.35	68.54
This Design (Async)	5,031.00	18.47	60.37
This Design (Clock)	205.60	7.60	3.23
Baireddy [76]	1.00	–	1.00
Chong (Async-1.1V)	0.02	8.33	4.26
Chong (Async-3.5V) $^{+}$	0.02	17.01	1.34
Baas (3.3V) [79]	421.98	8.44	14.48
Baas (5V)$^{+}$[79]	421.98	3.31	9.56

The relative cost of development of asynchronous circuits with the new flow is similar to its synchronous counterpart for the development of these multirate designs. The FFT circuit operates at 1.4GHz and consumes 59.2pJ of energy per data point.

A 2.4×, 2.4× and 3.2× benefit in terms of area, energy and throughput respectively over its synchronous counterpart is achieved. Also a 0.48×, 4.5× and 32.20× benefit over a low power 64-point FFT design by Texas Instruments [76] as well as a 2.77×, 8.01× and 8.32× benefit over a similar 16-point FFT architecture [75] are reported respectively for area, energy and throughput.

6.2 Relative Timed Clocking

The synchronous methodology relies on a global clock signal to store data into registers. The clock signal toggles periodically and continuously. The clocked design does not contain explicit signals or information for forking or joining information; cycle counts are used to ensure correct data flow.

Synchronous designs rely on a global clock signal to store data into registers. The clock signal ensures the data sequencing and allows pipelining to parallelize computation. The system clock signal dissipates 30%-70% of the total dynamic power [80]. The clock network is the best candidate for reducing the overall power of a design. Traditional synchronous methodology implies at every clock edge the registers are storing a new data. The assumption is that any data presented at the input port is crucial and valid. This results in a waste of energy in many systems which new data will be presented only when the certain control signal is asserted.

Clock gating is one of the power reduction techniques largely used in the clocked design. The system clock signal is gated with a clock gating cell, which consists of a latch and the logic AND gate. A data dependent clock gating can achieve power reduction of 15%-20% [81]. This approach uses an XOR gate to compare the current computation with the one stored to determine if clocking is necessary for the current operation. However, the evaluation requires area and power penalty and is unnecessary when the data is changing.

Relative timed clocking is a clock gating method utilizing the relative timing methodology. The local clock signal is either directly derived from enable signal transitions or use the enable signal along with the global clock. Specifically, the data validity token is directly associate with the enable signal.

6.2.1 RT Clocking Method

Asynchronous circuit design carries data validity tokens with each data item. Multiple convergent data paths will not interact until valid data is present on all paths. If data on one path is early or late, the stage where data interacts will stall until all data arrives. Clocked design, on the other hand, has optimized the logic to remove data validity information from the system. Therefore, clocked design instead relies on cycle counts to ensure that multiple convergent data paths interact correctly. This is achieved by ensuring that data from all paths will always arrive at the stage where they interact in the exact same cycle. While this leads to some efficiencies, it is inefficient in others. All the explicit data validity information and logic is removed, at the expense of wasting energy by clocking registers when no new data is present or will be used.

6.2.2 RT Clocking Types

There are two types of RT clocking, direct and indirect (Figures 6.11 and Figure 6.12). Direct RT clocking is using the control signal from data as the clock connecting to the register blocks. The control signal has to be hazard free to prevent any clock glitching. Also, the control signal has to be a pulsing signal. Namely the control signal has to toggle at every clock edge or only on rising or falling transitions indicates a new data is presenting.

Figure 6.11 shows a direct RT clocking design. The setup time and hold time of the flop can be constrained by delaying the *data_clk*. In this example, when a clocked register bank conditionally updates its values, rather than use the global clock, the output of the clocked

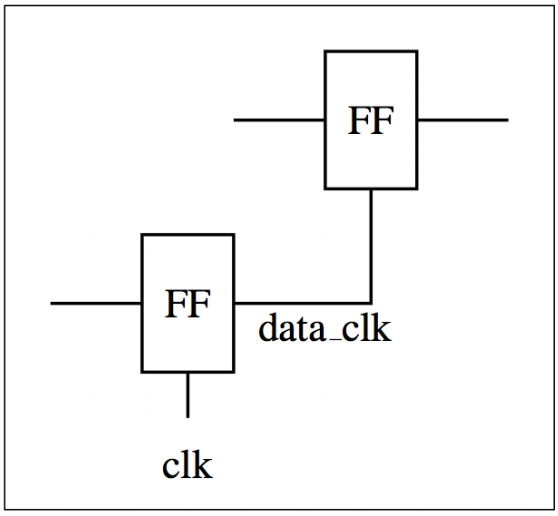

Figure 6.11. Direct RT Clocking

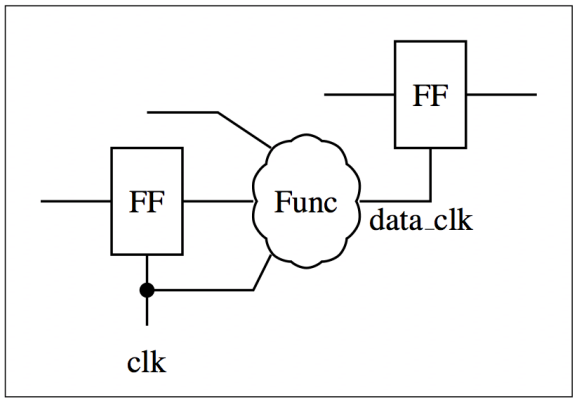

Figure 6.12. Indirect RT Clocking

flop is directly employed as a local clock to other register(s). Thus, no clock gating energy is used in the system.

Indirect RT clocking (Figure 6.12) uses the system clock and the control signals to clock the next stage registers. Indirect RT clocking is required when the control signal of logic level 1 indicates a continuous operation through multiple global clock cycles. Inputs to the combinational or sequential block *Func* can come from two different sources: it can be data coming from a register, as in the previous example, multiple registers, and even employ the clock signal itself as an input to the *Func* block. The delay through this block will result in a delayed clocking of the second Flip-Flop in relation to the first Flip-Flop. One property that must hold for this approach to work is that the *data_clk* signal must be monotonic. Any glitch on the *data_clk* results in the incorrect circuit. The methods that generating a glitch free circuits are known [82],[83].

6.2.3 Verilog Conversion

An example of such a modification from a traditional clocked design to the design of this invention can be expressed in Verilog code as follows. In the traditional expression of clocked design, the register is clocked by the clk signal (it is in the always @ block). The verilog coding is shown in Figure 6.13. The data signal trigger is sampled every clock cycle. When trigger is true on the rising edge of the clock, the value of function will be stored in register result.

Rather than sample the signal trigger every clock cycle, the following Verilog code will store the value of the function in the register result every time trigger rises. This is a much more energy efficient implementation than the above code. In order to implement this change, the way the Verilog code is written must be modified, and the timing of the design

```
always @ (posedge clk) begin
  if (trigger) begin
    result <= function;
  end
end
```

Figure 6.13. Code Snap of Typical Register

changes, as shown in Figure 6.14.

The power advantages can be proportional to the number of cycles that trigger is true compared to the total number of clock cycles. In many designs, this savings is significant. A mechanical translation of a design into a design using this invention can result in a more energy efficient circuit. Ideally the modification would not modify the behavior or performance of the design. This general translation changes the timing of the design and can change the behavior of a design as well.

6.2.4 Direct RT Clocking

The timing for direct RT clocking is explained using a 32-bit counter design as an example. A synchronous 32-bit counter contains 32 registers and a 32-bit increment (+1) function to compute the result.

When direct RT clocking is employed, the following structure is designed (Figure 6.15). This design contains four 2-bit shift registers, which are initialized with the higher bit to be logic zero and the lower bit to be logic one. The global clock only connects to the first 2-bit shift register, while the rest are clocked from the previous stage. A 28-bit counter is built and clocked by the output of the final 2-bit shift register. The 32-bit output is from by concatenating the output of the four 2-bit shifter registers and the 28-bit counter. Four additional registers are required for RT clocking design.

However, the dynamic clocking energy of this implementation is equivalent to clocking 5.5 registers, as shown in the following equation.

$$\textit{Direct RT clocking} : 2 * 1 + 2 * \frac{1}{2} + 2 * \frac{1}{4} + 2 * \frac{1}{8} + 28 * \frac{1}{16} = 5.5 \qquad (6.2)$$

The RT clocking also reduces the design complexity of the adder in this design. A 28-bit increment function is used rather than a 32-bit one. The lower 4 bits are computed using the shifting of the shifter registers. The 28-bit counter only being clocked once every sixteen cycles.

```
always @ (posedge trigger) begin
  result <= function;
end
```

Figure 6.14. Code Snap of RT Clocking Register

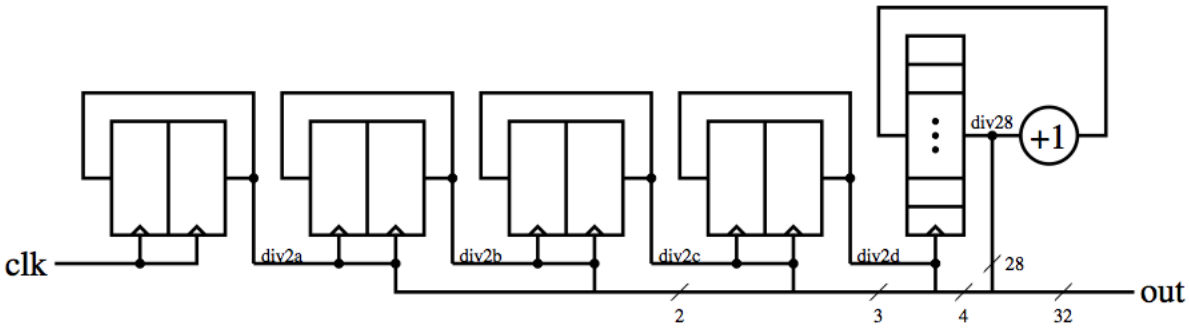

Figure 6.15. Direct RT Clocking 32-Bit Counter

Timing that results from RT clocking is different than typical clocked design. The timing validation would rely on method presented in Chapter 4 rather than traditional method that employ to clocked designs. Additional delay is introduced in RT clocking. For this example, the local clock of the 28-bit counter is delayed for 4 clock-to-q delay as illustrated in Figure 6.16. The counter output is not stable until div28 is stable. The more pipe stages are cascaded using direct RT clocking, the larger the clock skew would be for the output. When interfacing the RT clocking design to the traditional clocked design, sufficient setup and hold time is required.

6.2.4.1 Additional Setup and Hold Time

Considering the design in Figure 6.17, the *data_clk* signal as well as the input of FF_1 are sourced from the same *trigger* signal. The setup time and hold time constraints of the FF_1 needs to hold for correct operation. Those constraints can be satisfied by adding path delay in one of the two location. If data are to arrive at FF1 before the clock (data_clk), then minimum delay buffering is added to the data clk signal path. This delay must be sufficient in the worst case corners to allow the data input to the flip-flop to arrive a setup time before the clock. If data are to arrive at FF_1 after the clock, then minimum delay buffering is added to the FF_0 to FF_1 data path through *Func*. This delay must ensure in the worst case corner that sufficient hold time occurs on the data input to FF_1 before the

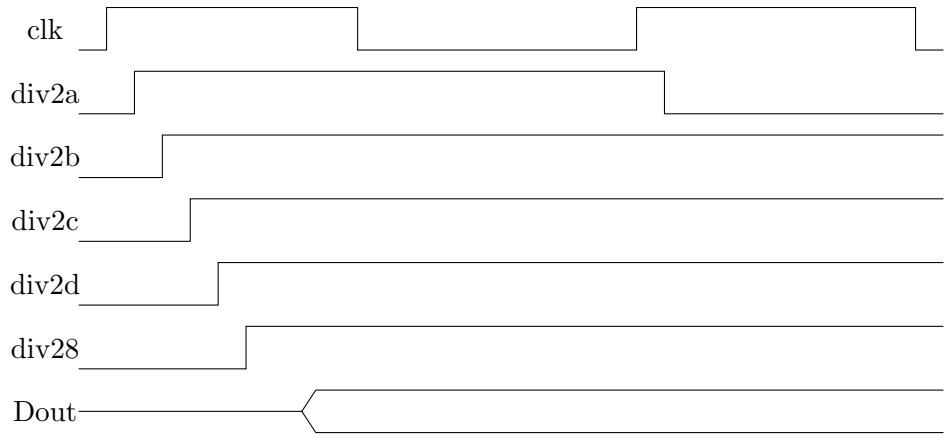

Figure 6.16. Waveform of Direct RT Clocking

clock signal can change. The size of the delays that are necessary can be calculated with traditional EDA tools that evaluate circuit timing.

6.2.5 Indirect RT Clocking

The local clock is generated by ANDing the trigger with the global clock. The local clock toggles if the trigger signal remain at the logic level 1. However, there is a timing problem with directly connect the clock signal and the trigger to the AND gate which invalidates the monotonicity requirement. Since *trigger* signal (*data_clk* in Figure 6.12) is generated from the clock, it will normally become asserted after the clock. This results in a shorter width clock pulse and a glitch when the trigger signal switch to logic level 0. The shorter clock pulse can violate the minimum clock width constraint for registers. The small pulse at the end of the trigger can result in a runt pulse or false data latched into the register, as shown in Figure 6.18.

These two problems can be solved by ANDing the trig signal with a delayed clock signal (signal *clkd*). The delay of the clock has to be greater than the combinational delay of the *trigger* signal. Figure 6.19 demonstrates a correct glitch-free operation.

6.2.6 Cycle Accuracy

A second potential form for design failure is related to the cycle in which events occur. This can be illustrated from the example in Figure 6.17. In a clocked system, *trigger* and *data_clk* are both tied to the global clock. In this case, data through *Func* will be stored in FF_1 on the next clock cycle after it is stored in FF_0. However, both direct and indirect RT clocking, the designer can store data into FF_1 within the same global clock period. Once the *data_clk* is delayed to assert after the data from FF_0 through *Func* is valid, the computed

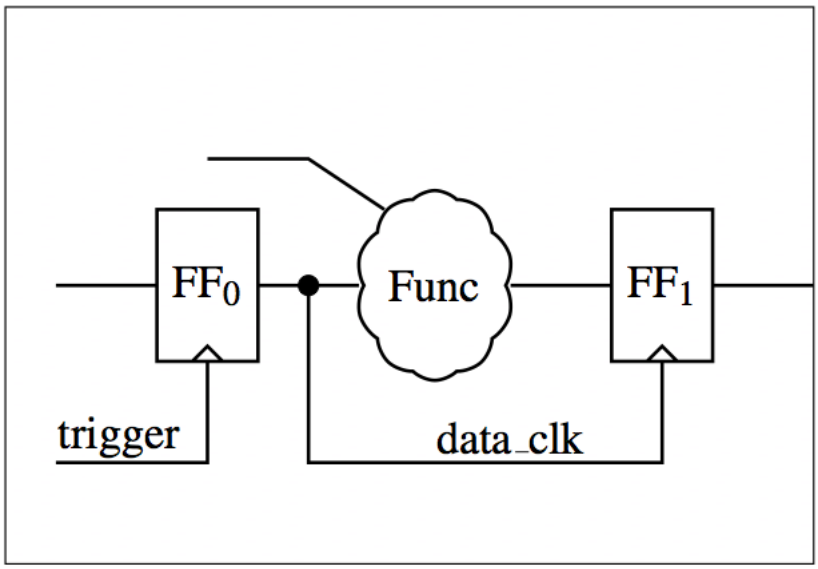

Figure 6.17. Direct RT Clocking with Data

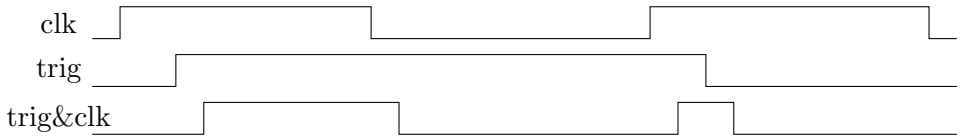

Figure 6.18. Waveform of Indirect RT Clocking that Glitches

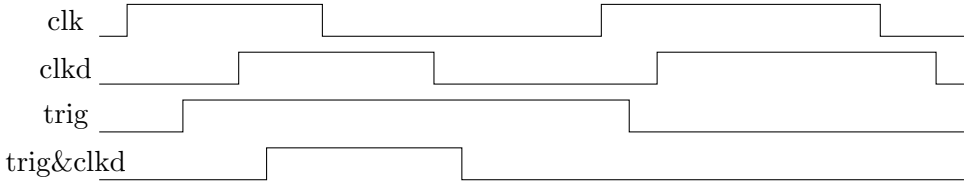

Figure 6.19. Glitch-Free Waveform of Indirect RT Clocking

data is stored in FF_1. If data from FF_1 is used in a convergent data path with other clocked data words, failure will occur as the data from this path will be off by one cycle.

6.2.7 Design Examples

6.2.7.1 I^2C with RT Clock Gating

The I^2C design consists of two finite state machine (FSM) such as byte-control and bit-control FSMs [84]. The byte-control monitors number of bits transferred and received, sends commands to the bit-control, and interfaces with the digital system. Once the commands

are sent, the byte-control is waiting for the bit-control to send a completion signal when the operation is completed. The bit-control decodes the two signal channel SCL and SDA of the I^2C bus. SCL is the clock line driven by the master and SDA is the data line shared between the I^2C master and slave. The indirect RT clocking is applied to both of the FSMs. The *trigger* for the byte-control is the completion signal and the *trigger* for the bit-control is the commands.

Table 6.4 pinpoints the benefit of applying RT clocking by comparing area and power with the clocked version. The design is synthesized using Design Compiler with 180nm IBM7RF library. PrimeTime is used to extract power number. Switching power, internal power, and leakage power are reported. The switching power and internal power drop by around 40% since the FSM operates only when command or completion signal is asserted. The RT clocking removes the redundant clock switching while in the idle state or waiting for data on I^2C bus. The I^2C bus operates at 400KHz while the byte-control operates at 20MHz. The power saving from RT clocking is because of the 50× difference in operational frequency.

6.2.7.2 Mixed-Signal Design

The indirect RT clocking is implemented on a sub-module of the digital part in the magnetic stripe reader [85]. The original reader design is sponsored by the industry partner ON Semiconductor. This design consists of an FSM and an up counter for measuring the period of the input analog signal.

Indirect RT clocking is applied to the FSM. There are two operations that the FSM performs. First, the FSM samples the analog signal and starts the computation. Second, once the computation starts, the FSM has to traverse through multiple states to finish the computation with respect to the analog inputs, counter values, and digital control signals. Finally, the FSM returns to the initial state and waits for the change of analog inputs.

The analog signal triggers once per milisecond while the FSM is operating with a 20MHz clock. The switching rate difference creates unnecessary switching for the state holding registers while in the polling stage. In addition, after the analog signal is sampled, the FSM

Table 6.4. I^2C Design Comparison

Design	Area(μm^2)	Switching(μW)	Internal(μW)	Leakage(μW)	Total(μW)
RT clocking	13585	376	223	0.12	599
Clock	12938	569	314	0.12	883
Benefit	95%	1.51×	1.4×	1×	1.47×

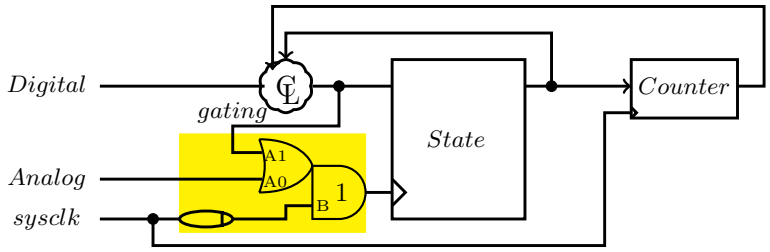

Figure 6.20. Digital Compute Unit

has to check for the value in the counter to update its state. Again, no actual computation is required while waiting for the counter to reach a threshold. An indirect RT clocking element, which considered the scenarios discussed above, is designed. Figure 6.20 shows the block diagram of the design while the indirect RT clocking design is highlighted in yellow. The RT clocking is a Boolean function of (*Analog* | *gating*) & *delayed_sclk*. The FSM also takes input digital signals for the computation. The gating signal is generated based on the current state, counter threshold, and the input digital signals.

Two timing constraints are required for this design. First, *gating* signal has to arrive before delayed *sysclk* arrives at 1/B. Second, the data feeding back to the register has to be stable before the gated clock arrives for the setup time of the state register. This design is implemented in half a micron and 350 nm technology nodes. Power is the main metric for optimization for this design since the frequency of change of the analog input is low. The computation of the digital part can easily be accomplished within the required sampling time window of the analog input. Hence, cycle accuracy is not required for this design. Table 6.5 shows the benefit of applying indirect RT clocking comparing the original clocked design. The power saving is 3× with 1% area penalty. The power saving comes from the low frequency of change of the analog signal because RT clocking removes unnecessary clock switching when the design is waiting for the analog inputs.

Table 6.5. Mixed Signal Design Comparison

Design	Gate Counts	Power (μW)
RT clocking	2709	1064
Clock	2672	3534
Benefit	99%	3.32×

6.2.8 Summary

Two RT clocking methods are introduced, direct and indirect RT clocking. Direct RT clocking can directly employ the data signal, or send it through a function. If the data clock can be asserted multiple cycles, indirect RT clocking is considered in order to create multiple edges on the trigger signal that will store new data in a register.

This data clock gating can result in significant power reductions. Two designs are shown with 50% power saving with less than 5% area overhead on a purely digital design. When employed on a mixed-signal chip, the design resulted in a 3.5× reduction in energy for the digital portion of the design. The ability to use data signals directly to gate a system allows it to becomes reactive, and can respond the same cycle data is produced. This is a property of asynchronous designs, that can be inherited in clocked designs. RT clocking can improve performance. It can also introduce cycle inaccuracies as events occur a cycle early. To preserve cycle accuracy, outputs would need to be delayed until the next clock cycle. When direct RT clocking is employed, the data clock signal must be monotonic. The same principle applies to the data clock generated by a function. Timing is quite different from traditional clocked design, as data is delayed and skewed each time this approach is cascaded and not synchronized back to the clock. This can result in extra hold time requirements and performance considerations.

The data clock trigger signals will all be initially referenced from the global clock. The global clock derived signals are used to store data in subsequent registers, and these signals in turn can be used to store data in registers, and so on. Thus later derived trigger signals can have a significant delay in relation to the global clock.

Additional timing constraints are needed to ensure the setup time and hold time of the registers with RT clocking. RT clocking creates different skew in relation to the global clock.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

The backend tools and flows to facilitate and validate the functionality of asynchronous designs are presented in this dissertation. This flow utilized the synchronous computer aided design (CAD) tools with additional asynchronous supports.

Timing driven synthesis and place and route algorithm are well developed for synchronous design. These algorithms only works with timing graph that is represented as a directed acyclic graph (DAG). Asynchronous circuits are naturally cyclic and sequential thus a generic cycle cutting algorithm is applied to represent its timing graph as a DAG. Moreover, the true timing paths of the asynchronous design are observed and verified.

The true paths are supplied to the generic cycle cutting to perform timing path driven cycle cutting. When true timing paths are provided to a set of benchmark circuits, they exhibited 40% smaller size, 30% energy reduction and 5% performance improvement when synthesized using Design Compiler.

Most of the timing paths of asynchronous circuits are cyclic. The cyclic timing paths are not supported by the commercial CAD tools. A path based timing validation for timed asynchronous circuits is introduced that uses static timing analysis (STA). The validation performs iterative STAs with multiple timing graphs to compute the full path delay of a cyclic timing path. Histograms of slacks are reported to reveal the potential timing optimization. A 57K-gate design is fully validated under 10 minutes total run time.

Asynchronous circuits are tested using both scan-chain and functional test. The scan chain is inserted in the data path and ATPG are performed using commercial CAD tools. Functional tests are used to determine the fault coverage of the control channels of an asynchronous circuit. A list of sequential test patterns is generated and is supplied to a switch level fault simulator. Stuck-at-faults are injected on every node. The outputs of the control network are observable. The combination of the two shows 93% fault coverage on a 225K-gate 64-point FFT circuit.

An automatic timing constraint mapping algorithm is developed to aid the overall design flow. Asynchronous macros are characterized with timing constraints. These constraints are translated to a template based representation. A path search algorithm is used to traverse the system and identify connections between macros. An sdc constraint set is formed for the system and thus timing driven synthesis and place and route can be performed by commercial CAD tools.

Asynchronous designs are built as demonstration applications to demonstrate the algorithms developed above. Synchronous and Asynchronous 64-point FFTs are constructed. It has shown that the asynchronous FFT achieves 2.4×, 2.4× and 3.2× benefit in terms of area, energy and throughput respectively over the synchronous design. An asynchronous inspired clocking mechanism is introduced.

This approach uses data signals for the clock to reduce the redundant clock switching. A digital design is built and achieves 50% power saving with less than 5% area overhead. A 3.5× energy reduction is achieved on the digital portion of another mixed signal chip.

This research work can be summed up as follows: backend support significantly improves all aspects of relative timed asynchronous design. This supports provide guidance that enables the future optimization towards faster, smaller, less power and, more robust circuit designs. Also, integration with commercial CAD tools for automatic validation of asynchronous designs is achieved.

7.2 Future Work

This dissertation addresses the validation portion of the backend tools and flows. The adoption of this research work can be enhanced by automation of design optimization. Future research works include the following:

- Slacks reported by timing validation can be used as an input for the timing optimization. The optimization can be accomplished by changing the timing constraints, either increase the delay target for the max path or reduce the delay target for the min path. This optimization aids the synthesis tool on better sizing the gates as well as aids the place and route tool for better placement and routing algorithms. Automation on bringing the slack between the min and max path toward zero helps the development time and can achieve a better design.

- Sequential test patterns are generated manually in this research work. Since the number of sequential test patterns is bounded, an automatic sequential test pattern

generation can be developed to remove the redundant test patterns thus improve the testing time. The sequential test patterns can be supplied to an asynchronous test processor [86] to test the manufactured chip.

- Asynchronous circuits are event-driven and data tokens are propagated freely once there is an empty place. In order to detect delay faults, single stepping from controllers to controllers is necessary. Mr.Go provides the ability to start and stop single controller thus provide single stepping support [65]. Modification of relative timed controller as well as the power, performance, and robustness trade-off have to be studied to optimize design.

- The RT clocking method provides a power advantage with minor design overhead. An automatic RT clocking adoption with multiple frequency designs or mixed signal designs can achieve power reduction easily. Evaluation on the power benefit and area overhead on more designs is needed to find the optimal RT clocking design.

REFERENCES

[1] K. S. Stevens, R. Ginosar, and S. Rotem, "Relative timing," *IEEE Trans. VLSI Syst.*, vol. 1, no. 11, pp. 129–140, Feb. 2003.

[2] V. S. Vij, "Algorithms and methodology to design asynchronous circuits using synchronous CAD tools and flows," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Utah, Salt Lake City, UT, May 2013.

[3] K. S. Stevens, S. Rotem, R. Ginosar, P. Beerel, C. J. Myers, K. Y. Yun, R. Kol, C. Dike, and M. Roncken, "An asynchronous instruction length decoder," *IEEE J. Solid-State Circuits*, vol. 36, no. 2, pp. 217–228, Feb. 2001.

[4] H. Han and K. S. Stevens, "Clocked and asynchronous FIFO characterization and comparision," in *17th Int. Conf. Very Large Scale Integration.* IFIP/IEEE, Oct. 2009, pp. 101–108.

[5] K. S. Stevens, P. Golani, and P. A. Beerel, "Energy and performance models for synchronous and asynchronous communication," *IEEE Trans. VLSI Syst.*, vol. 19, no. 3, pp. 369–392, March 2011.

[6] S. Das, V. Vij, and K. S. Stevens, "SAS: source asynchronous signaling protocol for asynchronous handshake communication free from wire delay overhead," in *Int. Symp. Asynchronous Circuits and Systems*, May 2013, pp. 107–114.

[7] D. Gebhardt, "Energy-efficient design of an asynchronous network-on-chip," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Utah, Salt Lake City, UT, 2011.

[8] D. Gebhardt, J. You, and K. S. Stevens, "Comparing energy and latency of asynchronous and synchronous nocs for embedded socs," in *4th Int. Symp. Network-on-Chip*, May 2010, pp. 115–122.

[9] W. Lee, V. S. Vij, A. R. Thatcher, and K. S. Stevens, "Design of low energy, high performance synchronous and asynchronous 64-point FFT," in *Design, Automation Test in Europe Conf. Exhibition (DATE), 2013*, March 2013, pp. 242–247.

[10] D. Bhadra, V. S. Vij, and K. S. Stevens, "A low power UART design based on asynchronous techniques," in *56th Int. Midwest Symp. Circuits and Systems (MWSCAS)*, Aug 2013, pp. 21–24.

[11] V. S. Vij, R. P. Gudla, and K. S. Stevens, "Interfacing synchronous and asynchronous domains for open core protocol," in *IEEE Int. Conf. on VLSI Design*, Jan 2014, pp. 282–287.

[12] L. T. Duarte, "Performance-oriented syntax-directed synthesis of asynchronous circuits," Ph.D. dissertation, Dept. Compute. Sci., Univ. Manchester, Manchester, United Kingdom, June 2010.

[13] G. Birtwistle and A. Davis, *Asynchronous Digital Circuit Design.* Springer-Verlag, 1995.

[14] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989, turing Award Paper.

[15] A. M. Lines, "Pipelined asynchronous circuits," Master's thesis, Dept. Comput. Sci., California Inst. Technol., Pasadena, CA, 1998.

[16] S. Tugsinavisut, Y. Hong, D. Kim, K. Kim, and P. A. Beerel, "Efficient asynchronous bundled-data pipelines for DCT matrix-vector multiplication," *IEEE Trans. VLSI Syst.*, vol. 13, no. 4, pp. 448–461, 2005.

[17] Synopsys. (2015, Feb.) IC Validator. [Online]. Available: http://www.synopsys.com/Tools/Implementation/PhysicalVerification/Pages/ICValidator-ds.aspx

[18] S. Chakraborty, K. Yun, and D. Dill, "Timing analysis of asynchronous systems using time separation of events," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 8, pp. 1061–1076, Aug 1999.

[19] M. Bozga, H. Jianmin, O. Maler, and S. Yovine, "Verification of asynchronous circuits using timed automata," *Electron. Notes Theoretical Comput. Sci.*, vol. 65, no. 6, pp. 47–59, 2002.

[20] R. B. Hitchcock Sr, "Timing verification and the timing analysis program," in *Proc. 19th Design Automation Conf.*, 1982, pp. 594–604.

[21] H. Chen, B. Lu, and D.-Z. Du, "Static timing analysis with false paths," in *Proc. 2000 Int. Conf. Comput. Design*, 2000, pp. 541–544.

[22] M. Ozcan, M. Imai, and T. Nanya, "Generation and verification of timing constraints for fine-grain pipelined asynchronous data-path circuits," in *Proc. 8th Int. Symp. Asynchronous Circuits and Systems.* IEEE, 2002, pp. 109–114.

[23] E. Yahya, L. Fesquet, Y. Ismail, and M. Renaudin, "Statistical static timing analysis of conditional asynchronous circuits using model-based simulation," in *19th Int. Symp. Asynchronous Circuits and Systems (ASYNC)*, May 2013, pp. 67–74.

[24] M. Iizuka, N. Hamada, H. Saito, R. Yamaguchi, and M. Yoshinaga, "A tool set for the design of asynchronous circuits with bundled-data implementation," in *29th Int. Conf. Comput. Design (ICCD)*, Oct 2011, pp. 78–83.

[25] N. Andrikos, L. Lavagno, D. Pandini, and C. P. Sotiriou, "A fully-automated desynchronization flow for synchronous circuits," in *44th ACM/IEEE Design Automation Conf.*, June 2007, pp. 982–985.

[26] G. Birtwistle and K. S. Stevens, "The family of 4-phase latch protocols," in *14th Int. Symp. Asynchronous Circuits and Systems*, April 2008, pp. 71–82.

[27] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective* . Menlo Park, CA: Addison-Wesley, 2010.

[28] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability.* Academic Press, 2006.

[29] *The International Technology Roadmap for Semiconductors*, 2013th ed., Semiconductor Industry Association, 2013.

[30] M. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-signal VLSI Circuits*. Springer Science & Business Media, 2000, vol. 17.

[31] A. Khoche and E. Brunvand, "A partial scan methodology for testing self-timed circuits," in *Proc. 13th VLSI Test Symp.*, 1995, pp. 283–289.

[32] V. Schober and T. Kiel, "An asynchronous scan path concept for micropipelines using the bundled data convention," in *Proc. Int. Test Conf.* IEEE, 1996, pp. 225–231.

[33] C.-L. Wey, M.-D. Shieh, and P. D. Fisher, "ASLCScan: A scan design technique for asynchronous sequential logic circuits," in *1993 IEEE Int. Conf. Compute. Design: VLSI in Compute. and Processors*. IEEE, 1993, pp. 159–162.

[34] O. Petlin and S. Furber, "Scan testing of asynchronous sequential circuits," in *5th Great Lakes Symp. VLSI*, Mar 1995, pp. 224–229.

[35] D. P. Vasudevan, "Automatic test pattern generation for asynchronous circuits," Ph.D. dissertation, Inst. Computing Syst. Architecture, Univ. Edinburgh, Edinburgh, Scotland, May 2011.

[36] K. Terayama, A. Kurokawa, and M. Imai, "Scan test of latch-based asynchronous pipeline circuits under 2-phase handshaking protocol," in *19th Workshop Synthesis and Syst. Integration of Mixed Inform. technol.*, 2015.

[37] C. Wolf, S. Zeidler, M. Krstic, and R. Kraemer, "Overview on ATE test and debugging methods for asynchronous circuits," in *12th Int. Workshop Microprocessor Test and Verification (MTV)*, Dec 2011, pp. 16–21.

[38] F. Shi and Y. Makris, "Testing delay faults in asynchronous handshake circuits," in *2006 IEEE/ACM Int. Conf. on Comput. Aided Design*, Nov 2006, pp. 193–197.

[39] W. Lee, V. S. Vij, and K. S. Stevens, "Timing path-driven cycle cutting for sequential controllers," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 4, pp. 64:1–64:25, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/2893473

[40] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev, "Asynchronous design using commercial HDL synthesis tools," in *Proc. 6th Int. Symp. Advanced Res. in Asynchronous Circuits and Syst.*, April 2000, pp. 114–125.

[41] A. Kondratyev and K. Lwin, "Design of asynchronous circuits using synchronous CAD tools," *IEEE Design & Test of Compute.*, vol. 19, no. 4, pp. 107–117, July-Aug. 2002.

[42] A. Taubin, J. Cortadella, L. Lavagno, A. Kondratyev, and A. Peeters, "Design automation of real-life asynchronous devices and systems," *Foundations and Trends® in Electronic Design Automation*, vol. 2, no. 1, pp. 1–133, 2007.

[43] A. Smirnov, "Asynchronous micropipeline synthesis system," Ph.D. dissertation, Dept. Elect. Compute. Eng., Boston University, Boston, MA, 2009.

[44] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of asynchronous templates for integration into clocked CAD flows," in *15th Int. Symp. Asynchronous Circuits and Syst.*, 2009, pp. 151–161.

[45] P. Beerel, G. Dimou, and A. Lines, "Proteus: An ASIC flow for GHz asynchronous designs," *IEEE Design Test of Compute.*, vol. 28, no. 5, pp. 36–51, Sept 2011.

[46] M. D. Riedel and J. Bruck, "The synthesis of cyclic combinational circuits," in *Proc. 2003 Design Automation Conf.*, 2003, pp. 163–168.

[47] S. Malik, "Analysis of cyclic combinational circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 7, pp. 950–956, 1994.

[48] S. Edwards, "Making cyclic circuits acyclic," in *Proc. 40th Conf. Design Automation*, 2003, pp. 159–162.

[49] O. Neiroukh, S. A. Edwards, and X. Song, "Transforming cyclic circuits into acyclic equivalents," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1775–1787, 2008.

[50] T. Shiple, V. Singhal, R. Brayton, and A. Sangiovnni-Vincentelli, "Analysis of combinational cycles in sequential circuits," in *1996 IEEE Int. Symp. on Circuits and Syst.*, vol. 4, 1996.

[51] V. V. Filippovich, "Transforming a cyclic directed graph into an acyclic graph," *Cybern. and Syst. Anal.*, vol. 9, no. 2, pp. 348–351, March 1973.

[52] S. Nagasai, K. S. Stevens, and G. Birtwistle, "Concurrency reduction of untimed latch protocols – theory and practice," in *IEEE Int. Symp. Asynchronous Circuits and Syst.*, May 2010, pp. 26–37.

[53] G. M. Birtwistle and K. S. Stevens, "Modelling mixed 4phase pipelines: Structures and patterns," in *20th IEEE Int. Symp. on Asynchronous Circuits Syst,*, May 2014, pp. 27–36.

[54] K. S. Stevens, S. V. Robison, and A. Davis, "The post office – communication support for distributed ensemble architectures," in *Proc. 6th Int. Conf. Distributed Computing Syst.*, May 1986, pp. 160 – 166, best paper award.

[55] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: Part II (automatic synthesis)," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 2, pp. 118–132, Feb 1999.

[56] W. Lee, T. Sharma, and K. S. Stevens, "Path based timing validation for timed asynchronous design," in *29th Int. Conf. VLSI Design and 15th Int. Conf. Embedded Syst. (VLSID)*, Jan 2016.

[57] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design – A Systems Perspective*. Kluwer Academic Publishers, 2001.

[58] C. J. Myers, *Asynchronous Circuit Design*. J. Wiley, 1999.

[59] K. S. Stevens and V. Vij, "Cycle cutting with timing path analysis," U.S. Patent No. 8,239,796, Assignee: University of Utah Research Foundation, 29 Jan 2013.

[60] J. C. Culberson and U. of Alberta. Dept. of Computing Science, *Iterated Greedy Graph Coloring and the Difficulty Landscape*. Citeseer, 1992.

[61] R. E. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler, *COSMOS: a compiled simulator for MOS circuits.* ACM, 1988.

[62] R. Bryant, "A switch-level model and simulator for MOS digital systems," *IEEE Trans. Comput.*, vol. C-33, no. 2, pp. 160–177, Feb 1984.

[63] T. Edwards. (2015) IRSIM 9.7. [Online]. Available: http://opencircuitdesign.com/irsim/download.html

[64] A. Khoche and E. Brunvand, "A partial scan methodology for testing self-timed circuits," in *Proc. 13th VLSI Test Symp.*, Apr 1995, pp. 283–289.

[65] M. Roncken, S. Gilla, H. Park, N. Jamadagni, C. Cowan, and I. Sutherland, "Naturalized communication and testing," in *21st IEEE Int. Symp. Asynchronous Circuits and Syst. (ASYNC)*, May 2015, pp. 77–84.

[66] D. B. Armstrong, "A deductive method for simulating faults in logic circuits," *IEEE Trans. Comput.*, no. 5, pp. 464–471, 1972.

[67] E. G. Ulrich, V. D. Agrawal, and J. H. Arabian, "Concurrent fault simulation," in *Concurrent and Comparative Discrete Event Simulation.* Springer, 1994, pp. 57–62.

[68] Tiempo. (2014). [Online]. Available: http://www.tiempo-ic.com/

[69] A. Bardsley and D. Edwards, *Balsa: An Asynchronous Circuit Synthesis System.* University of Manchester, 1998.

[70] M. N. Horak, S. M. Nowick, M. Carlberg, and U. Vishkin, "A low-overhead asynchronous interconnection network for gals chip multiprocessors," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 494–507, April 2011.

[71] Y. Thonnart, E. Beigne, and P. Vivet, "A pseudo-synchronous implementation flow for WCHB QDI asynchronous circuits," in *18th IEEE Int. Symp. Asynchronous Circuits and Syst.* IEEE, 2012, pp. 73–80.

[72] B. W. Suter and K. S. Stevens, "Low power, high performance FFT design," in *Proc. IMACS World Congr. on Scientific Computation, Modeling, and Appl. Mathematics*, vol. 1, June 1997, pp. 99–104.

[73] B. W. Suter, *Multirate and Wavelet Signal Processing.* Academic Press, 1997.

[74] R. E. Miller, *Switching Theory.* New York, New York: Wiley, 1965, vol. 2.

[75] X. Guan, Y. Fei, and H. Lin, "Hierarchical design of an application-specific instruction set processor for high-throughput and scalable FFT processing," *IEEE Trans. VLSI Syst.*, vol. 20, no. 3, pp. 551–563, March 2012.

[76] V. Baireddy, H. Khasnis, and R. Mundhada, "A 64-4096 point FFT/IFFT/Windowing processor for multistandard ADSL/VDSL applications," in *Int. Symp. Signals, Syst. Electron.*, 2007, pp. 403–405.

[77] A. Chandrakasan, W. Bowhill, and F. Fox, *Design of High-Performance Microprocessor Circuits.* Wiley-IEEE Press, 2000.

[78] K. Chong, B. Gwee, and J. Chang, "Energy-efficient synchronous-logic and asynchronous-logic FFT/IFFT processors," *IEEE J. Solid-State Circuits*, vol. 42, no. 9, pp. 2034–2045, 2007.

[79] B. M. Baas, "A low-power, high-performance, 1024-point FFT processor," *IEEE J. Solid-State Circuits*, vol. 34, no. 3, pp. 380–387, 1999.

[80] V. G. Oklobdzija, V. M. Stojanovic, D. M. Markovic, and N. M. Nedovic, *Digital System Clocking: High-performance and Low-power Aspects*. John Wiley & Sons, 2005.

[81] S. Wimer and I. Koren, "Design flow for flip-flop grouping in data-driven clock gating," *IEEE Trans. VLSI Syst.*, vol. 22, no. 4, pp. 771–778, April 2014.

[82] J. Sudhakar, A. Prasad, and A. Panda, "GFCG: Glitch free combinational clock gating approach in nanometer VLSI circuits," in *2nd Int. Conf. Electron. Commun. Syst.(ICECS)*, Feb 2015, pp. 146–150.

[83] H. Karthik and B. Kumar Naik, "Glitch elimination and optimization of dynamic power dissipation in combinational circuits," in *2014 Int. Conf. Advances Electron., Comput., Commun. (ICAECC)*, Oct 2014, pp. 1–6.

[84] OpenCores, providing open source hardware IP cores., http://opencores.org.

[85] J. Liu, K. Chang, and T. Chou, "Magnetic stripe reader," Sep. 30 2014, US Patent 8,844,818. [Online]. Available: https://www.google.com/patents/US8844818

[86] S. Zeidler and M. Krstic, "A survey about testing asynchronous circuits," in *2015 European Conf. Circuit Theory and Design (ECCTD)*, Aug 2015, pp. 1–4.