STATIC TIMING ANALYSIS OF GASP

by

Prasad Joshi

A Thesis Presented to the
FACULTY OF THE USC VITERBI SCHOOL OF ENGINEERING
UNIVERISTY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
MASTER OF SCIENCE
(ELECTRICAL ENGINEERING)

December 2008

Acknowledgements

I sincerely thank my advisor, Dr. Peter Beerel for inspiring me to take up one of the most significant academic challenges that I have taken in my life. He has been the best advisor a graduate student can wish for and without his support, guidance and patience; this work would not have been completed. I also thank his wife Janet and baby Kira for considering me as a part of their family and providing me with a warm and productive environment to work. I would like to thank my friends and colleagues from the USC Asynchronous CAD/VLSI group, Pankaj Golani, Mallika Prakash, Amit Bandlish, George Dimou, Arash Saifhashemi, Gokul Govindu and Roger Su for their support and invaluable suggestions. A special mention goes to Mallika, Pankaj and Amit for helping me with tool related problems and always entertaining my questions at the weirdest of hours.

I would take this opportunity to show my respect and gratitude towards Dr. Ivan Sutherland of Sun Microsystems. He is one of the greatest engineers of this era and it has been an honor to have had the opportunity to work under him. I also thank Marly Roncken of Intel for her brilliant technical inputs and making the necessary circuits easily available. Ivan and Marly have been the pillars of this project and their confidence is me gave me the freedom and motivation to do better. I thank the VLSI research team of Sun Labs for giving me the opportunity of developing the static timing analysis flow for GasP as a summer 2008 intern. A special mention goes to my manager Jonathan Gainsley, Russell Kao, Jo Ebergen and Mark Greenstreet for guiding me towards a great internship.

I would also like to thank my previous manager Drew Guckenberger for encouraging me to continue my research when I was working as a summer 2007 intern in Luxtera.

I am grateful to my thesis committee members Dr. Melvin Breuer, Dr. Massoud Pedram and Dr. Sandeep Gupta for taking out time from their busy schedules to entertain my questions and giving me valuable feedback. They have been excellent teachers and have helped me build a strong technical understanding. This work has been sponsored by SRC grant CADTS-1425 and I would like to acknowledge their generous support. I also thank the EE-systems staff, especially Annie Yu, Diane Demetras and Tim Boston for guiding me through the various requirements of the EE department at USC.

There are many friends who have helped and supported me in the last few years. First, I thank my great friend Rajesh Kotian for providing me with love, care and shelter during my initial days of struggle. Second, I would like to thank my roommates Murtaza Motiwala, Jameel Koita, Hussain Attarwala, Naazneen Gandhi and Srinivas Vadhuvath for bearing my random hours of work and always helping me great technical discussions. Third, I thank my friends Mohit Thatte, Anirudh Shetye and Prasanjeet Das for being a constant source of support. Fourth, I thank all my friends in India for always being there with me when I needed them the most.

Last but not the least, I wish to thank my parents Asha Joshi and Prakash Joshi for their unconditional love and support that cannot be matched. They have worked extremely hard to provide me with a great platform for succeeding in life. I am incomplete without them and I dedicate this dissertation to them.

Table of Contents

Acknowledgements ii

List of Figures vi

Abstract viii

Chapter 1: Introduction 1
1.1 Single track handshaking 1
1.2 GasP family of circuits 3
1.3 Fleet architecture 6
1.4 Contributions of this thesis 7

Chapter 2: Timing Constraints for GasP 9
2.1 Relative Timing (RT) constraints on control logic 9
2.1.1 Rail to rail constraints 10
2.1.1.1 Predecessor loop constraint on the successor state-wire 11
2.1.1.2 Successor loop constraint on the predecessor state-wire 12
2.1.2 Short circuit constraints 13
2.1.2.1 Short circuit constraint on the successor state-wire 13
2.1.2.1 Short circuit constraint on the predecessor state-wire 14
2.2 Relative Timing (RT) constraints on the data-path 15

Chapter 3: Library Characterization for GasP 18
3.1 Defining timing arcs 18
3.2 Characterization of the timing arcs 20
3.2.1 Setting up the simulation environments 20
3.2.2 Measuring arc delays 21
3.3 Measuring pin capacitances: 23
3.4 Generating the Liberty files 24

Chapter 4: Static Timing Analysis for GasP 26
4.1 Challenges in interpreting asynchronous netlists 26
4.1.1 Bi-directional pins 26
4.1.2 Handling loops 29
4.1.2.1 Explicitly breaking timing loops 30
4.1.2.2 Loops that cannot be broken 30
4.1.3 Lack of a global clock 32
4.2 Timing verification flow for GasP 34
4.2.1 Part 1 of the verification flow 34
4.2.1.1 Verifying RT constraints on the control cells 34
4.1.2 Measuring phase differences between fire signals 35
4.1.2.1 Measuring phase difference for setup checks 35

4.1.2.2 Measuring phase difference for hold checks 36
4.2.2 Part 2 of the verification flow 37

Chapter 5: Evaluating the effects of operating environments 39
5.1 Background 40
5.1.1 The Charlie Effect 40
5.1.2 Operating regions 42
5.2 Charlie Effects on the RT constraints for control logic 45
5.3 Charlie Effects on the RT constraints for the data-path 46
5.3.1 Finding the worst case for the setup constraint 46
5.3.2 Finding the worst case for the hold constraint 48

Chapter 6: Example 50
6.1 The Predicate circuit 50
6.1.1 Three input AndOrLatch 52
6.2 Creating timing libraries and the Verilog netlist 53
6.3 PrimeTime scripts 53
6.4 Interpreting Timing Reports 55

Chapter 7: Conclusions and Future Work 57

Bibliography 59

Appendices
Appendix A: Fast timing library 61
Appendix B: Verilog netlist for the predicate circuit 65
Appendix C: PrimeTime script for verifying the RT constraints on the control logic and obtaining phase differences for verifying the RT constraints on the data-path 68
Appendix D: PrimeTime script to verify the RT constraints on the data-path 70
Appendix E: PrimeTime report for the predecessor loop constraint on the successor state-wire 71
Appendix F: PrimeTime report for the short circuit constraint on the successor state-wire 72
Appendix G: PrimeTime report for the successor loop constraint on the predecessor state-wire 73
Appendix H: PrimeTime report for the short circuit constraint on the predecessor state-wire 74
Appendix I: PrimeTime report showing the phase relation between FIRE signals 75
Appendix J: PrimeTime report for setup checks on the data-path 76
Appendix K: PrimeTime report for hold checks on the data-path 77

List of Figures

Figure 1: GasP Plain 4

Figure 2: GasP connections 4

Figure 3: A linear pipeline in the FLEET Architecture 7

Figure 4: Predecessor loop constraint on the successor state-wire 12

Figure 5: Successor loop constraint on the predecessor state-wire 12

Figure 6: Short circuit constraint on the successor state-wire 14

Figure 7: Short circuit constraints on the predecessor state-wire 14

Figure 8: Setup and Hold Checks 15

Figure 9: RT constraints on the data-path 16

Figure 10 : Complete signal transition diagram for GasP 19

Figure 11: Reduced signal transition diagram for GasP 20

Figure 12: Pre-driver method 21

Figure 13: Measuring pin capacitances 24

Figure 14: Characterization flow 25

Figure 15: Bi-directional pin problem 27

Figure 16: Split pin architecture 28

Figure 17: Connectivity of the split pin architecture 29

Figure 18: Loops in GasP 29

Figure 19: Loops that cannot be broken 31

Figure 20 : Adding a pseudo pin 32

Figure 21: Using set_data_check command 33

Figure 22: Verifying the RT constraints 35

Figure 23: Measuring phase difference for setup checks 36

Figure 24: Measuring phase difference for hold checks 37

Figure 25: Verifying the RT constraints on the data-path 38

Figure 26: NOR gate 40

Figure 27: Charlie Diagram 41

Figure 28: Data limited region 43

Figure 29: Bubble limited region 44

Figure 30: Full throughput region 44

Figure 31: Worst case for setup checks 47

Figure 32: Worst case for hold checks 48

Figure 33: The Predicate circuit 51

Figure 34: 3ip-AndOrLatch 52

Figure 35 : On-chip variation in PrimeTime 53

Figure 36: Verification of RT constraints on the GasP control cells 54

Figure 37: Measuring phase difference between fire signals for setup checks 54

Figure 38: Verification of RT constraints on the data-path 55

Figure 39 : Timing Report 56

Abstract

The 6-4 GasP family of asynchronous circuits has been sought for its potential advantages of ultra-high performance and low power especially in the processor and the network on chip (NoC) domains. However, the use of these circuits is currently limited to custom design where extensive SPICE simulations are required to verify timing correctness and performance. In order to incorporate these circuits in the standard ASIC designs, it is essential to establish a more efficient CAD flow.

A fully automated characterization flow for developing timing libraries of single track circuits was shown in [13]. This thesis extends that flow to the GasP family of circuits and addresses the issue of validating the timing performance of these non-standard circuits using static timing analysis. We first discuss some of the relative timing constraints that were identified to ensure the desired working of the GasP control circuits. Then we discuss the characterization flow used for developing timing libraries for these circuits. Thereafter, we discuss how a static timing analysis tool, Synopsys PrimeTime, was used to verify these relative timing constraints as well as perform setup and hold checks on a substantial industry design. We conclude this thesis by identifying the worst cases of operation for the relative timing constraints which can be used for post analysis debugging.

Chapter 1

Introduction

The increasing power consumption and growing complexity of synchronous designs has led to a great deal of interest in asynchronous circuits. The presence of a single global clock in the synchronous designs has resulted into problems like clock tree synthesis, gated clocking design, hold time fixing, and clock skew management. As a result of this, globally asynchronous and locally synchronous (GALS) systems are gaining prominence. The GALS systems have shown several advantages including low power in the design of networks on chip (NoCs).

ARM and Sun Microsystems have already been exploring the designs of highly efficient processors using asynchronous templates. ARM worked with Handshake Solutions for the development of ARM996HS which is industry's first clockless processor [10]. The ARM996HS is targeted towards low power applications in the biomedical and the automotive fields. The VLSI Research team of Sun Microsystems is working on designing high performance processors using the GasP family of circuits [8][2]. This chapter provides a background on Sun's work and introduces the terminology used in the rest of the document.

1.1 Single track handshaking

In the absence of a global clock that controls the data flow; asynchronous designs rely on handshaking to transfer data between functional blocks. A particularly interesting form of handshaking is the single track protocol proposed by K. van Berkel [18]. In a

single track protocol a single wire carries both the forward and reverse handshake signals. A sender briefly drives the wire to one logic level, signaling the presence of data on adjacent data wires. The receiver, noticing that change in the wire's state, copies the corresponding data and then briefly drives the wire to the other logic state to indicate that it has absorbed the previous data value. The single track protocol has been widely used by Beerel [7] and Sutherland [17] in the development of different asynchronous architectures.

Single track signaling is attractive not only because a single wire occupies less space, but also because single track signaling consumes minimum energy per cycle. A transition must pass in each direction, and the single wire does exactly that with automatic return to the initial state after each handshake. These two advantages are offset by a timing issue inherent in the word "briefly." Because sender and receiver share the signaling wire and drive it in opposite directions, each must take care to cease its drive promptly so that the other has free use of the wire. Proper operation of single-track systems depends on the proper behavior of each participant to drive only briefly.

Another important problem with these single track circuits is their limited testing ability. In general, these circuits do not conform to the standard circuit templates supported by commercial timing libraries. Unavailability of standard timing libraries for these circuits thereby becomes the limiting factor for their verification by using commercial techniques like static timing analysis. Previous work [13] has shown that library characterization for single track circuits can be done. However, the presence of bidirectional pins and combinational loops in these circuits result in empty timing graphs

in static timing analysis tools like Synopsys PrimeTime. This thesis addresses the issue of timing verification of single track circuits by developing a flow for the GasP family of asynchronous control circuits which is such a single track system [17].

1.2 GasP family of circuits

The 6-4 GasP family evolved from an earlier circuit family called asynchronous symmetric pulse protocol (asP*) which was designed by Charles E. Molnar. In [12] Molnar articulated the basic control requirement for asynchronous pipelines where each stage fired to advance the data through the data latches. The last three letters in the name GasP acknowledge its asP* ancestry. The ‘6-4’ term represents the six logic gates in the forward direction between each stage and its successor, gates A B C D E F shown in Figure 1, and the four logic gates in the reverse direction, gates A B C X in Figure 1.

A linear pipeline of GasP control circuits can be viewed as a cascade as shown in Figure 2 where the predecessor signal (PRED) of a stage is connected to the successor signal (SUCC) of the previous stage. Similarly the SUCC of a stage is connected to the PRED of the next stage. Two stages are connected through the single track wire called the state-wire which is used for handshaking between stages.

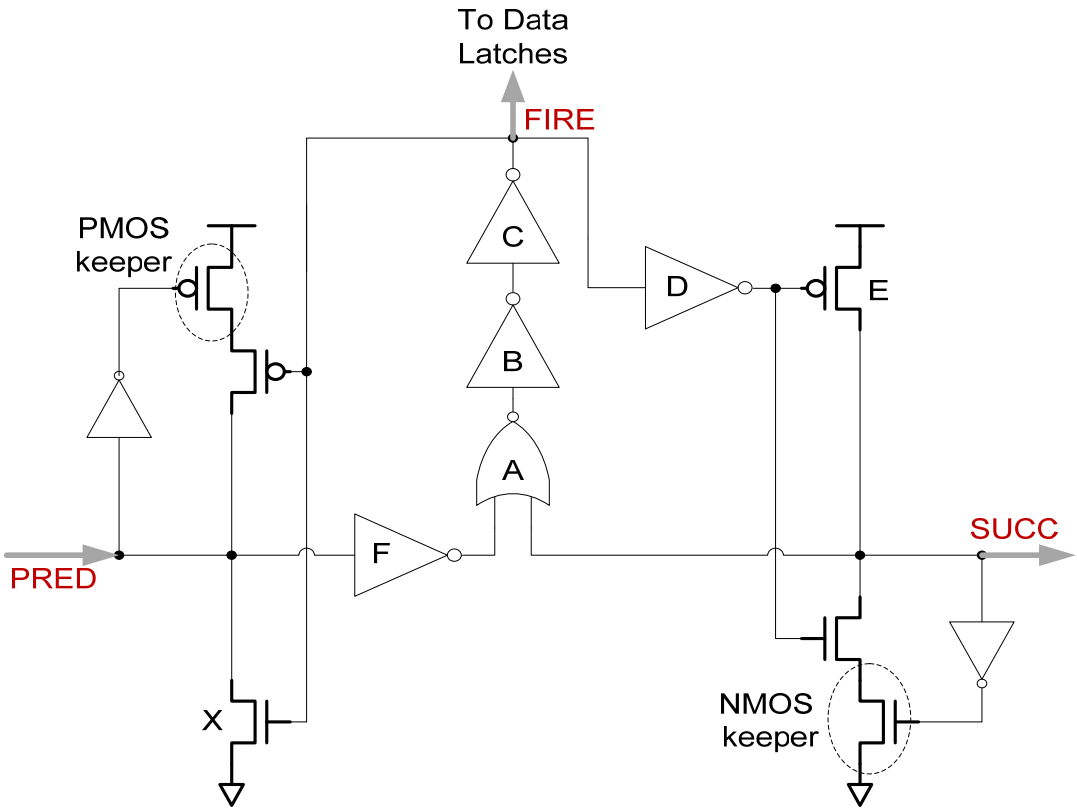

Figure 1: GasP Plain

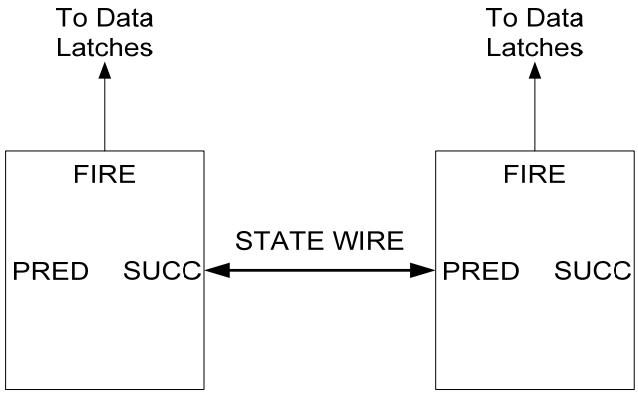

Figure 2: GasP connections

When there is no token on SUCC (i.e. the successor state-wire is empty), the NOR gate is enabled to process a new token on PRED. When a new token arrives on PRED, it indicates that the previous GasP stage has fired and the data is getting advanced to the latches controlled by the current GasP stage. As the PRED signal goes high, the current GasP stage raises its FIRE signal to make its latches transparent. Apart from making the latches transparent, this FIRE signal performs two other tasks. First, it makes the predecessor state-wire empty by pulling PRED low to inform the previous GasP stage that it has consumed the data. Second, it makes the successor state-wire full by pulling SUCC high to indicate the next GasP stage that the data is getting advanced to its data latches.

The presence of a token on SUCC (i.e. the successor state-wire is full) indicates that the next stage is yet to consume its data and any new token on PRED will thereby have to wait to be processed. In other words, a GasP stage cant fire for the second time until it receives an acknowledgement of data consumption from the next stage which is indicated by the successor state-wire being empty. This prevents the second data from corrupting the first data.

Notice that after the previous stage drives the PRED signal high, the current stage is responsible for actively maintaining the predecessor state-wire high through the PMOS keeper. Once the current stage fires, the PMOS keeper is turned off and the predecessor state-wire is pulled low. Similarly, after the next stage drives the SUCC signal low, the current stage is responsible for actively maintaining the successor state-wire low through the NMOS keeper. Once the current stage fires, the NMOS keeper is turned off and the

successor state-wire is pulled high. This handshake mechanism makes sure that both the state-wires are always statically driven thereby making the GasP family more robust to noise.

1.3 Fleet architecture

In a generic processor that operates on the notion of a common clock signal, the clock speed has to be slow enough to accommodate each computation. As a result, there exists a “worst path” that limits the clock frequency even though other parts of the chip might be able to complete their operation in much less time. In contrast, each part of an asynchronous system takes as much or as little time as it needs. Coordinating the asynchronous actions, however, also takes time and chip area. If the efforts required for local coordination are small, an asynchronous system may, on average, be faster than a clocked system. Another advantage of asynchrony is the negligible power consumption in the idle parts of the chip as an asynchronous chip by default is in power down mode. The VLSI research team of Sun Microsystems Laboratories plans to take advantage of these features in their novel computer design called FLEET [8][2].

A top level overview of a linear pipeline in the FLEET architecture is explained in Figure 3. Here the GasP cells act as local clock generators that drive the respective latches opaque and transparent. In general, the GasP cells guide the data across the data-path quite similar to the way the clock guides the data in synchronous systems.

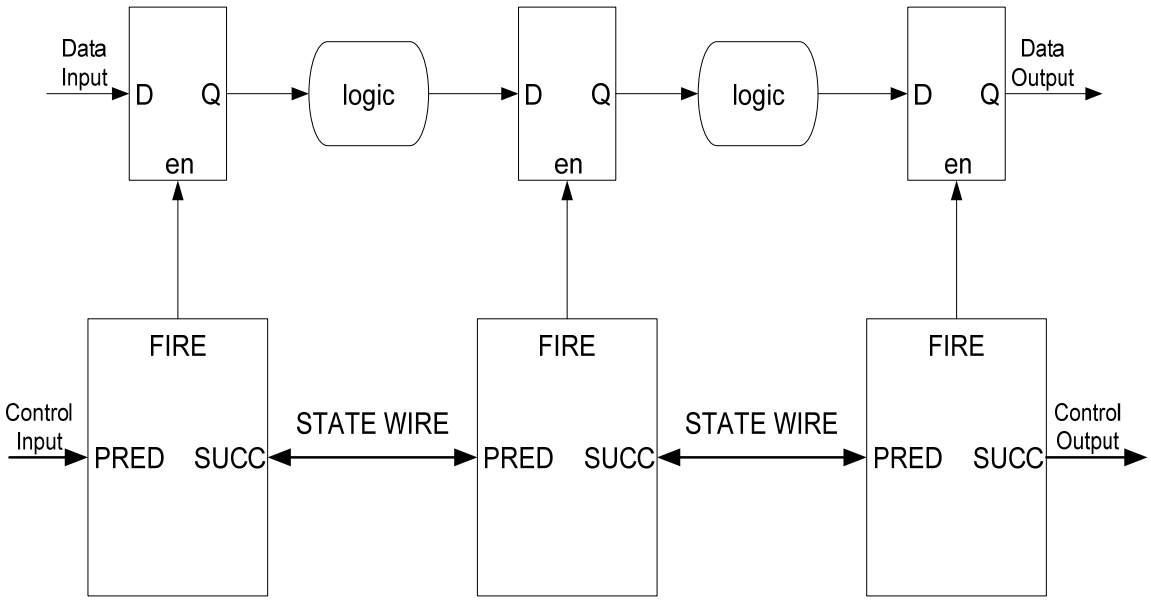

Figure 3: A linear pipeline in the FLEET Architecture

1.4 Contributions of this thesis

This thesis makes the following contributions:

- Timing constraints for the GasP family of circuits that address both correct operation and power considerations are identified.
- The presence of timing loops and bi-directional pins results in empty timing graphs in a commercial static timing analysis tool, Synopsys PrimeTime. As a result of this, verification of the timing constraints using a conventional static timing analysis flow is not possible. In order to overcome these challenges, pseudo pins and a novel split pin architecture were used. Thus a new static analysis flow was developed for the verification of the timing constraints imposed on the GasP family of circuits.
- After verifying the timing constraints, it is essential to determine the cause of timing violations, if any. However, PrimeTime provides little insight towards determining

the exact cause of the timing violations in these complicated single track circuits. The worst case of operation for the violated timing constraints were identified which will help the designer in post-analysis debugging.

Chapter 2

Timing Constraints for GasP

In order to ensure the timing correctness of the GasP circuits a set of timing constraints need to be met. These constraints can be classified into two parts. First, there are a set of internal timing constraints for the GasP control cells where a particular signal is required to arrive before another signal. These constraints depend on the relative ordering of signals and hence are termed as Relative Timing (RT) constraints on control logic. Second, the latches in the data-path which are clocked by the GasP control circuits need to meet the setup and hold constraints. Since the setup and hold constraints are also based on the relative ordering of signals we examine both sets of constraints using the Relative Timing (RT) framework defined in [16]. Similar RT constraints for other single track circuits like SSTFB were identified in [9].

2.1 Relative Timing (RT) constraints on control logic

The GasP family of circuits depends on the relative timing of logic gates to avoid drive conflict at the state-wire. Careful choice of the transistor sizes in GasP circuits enables the two participants to operate quickly while avoiding conflict. The transistors in the 6-4 GasP circuits are chosen to be strong enough so that each logic gate has approximately the same delay. This is possible because all but two of the logic gates drive fixed loads. The NOR gate, called A in Figure 1, drives only its own output capacitance, the capacitance of the relatively short wire to the inverter called B, and the input capacitance of inverter B. Likewise, B drives only its own output capacitance, a

short wire, and inverter C. In addition to driving both the inverter D and the NMOS transistor X and the wires to them, inverter C must drive the rather large load presented by the long control wire to the many latches that will capture the data. Thus inverter C tends to be rather large, but its load is the same in every module.

However, the two lone transistors E and X drive loads that vary from module to module. Their major load is the capacitance of the state-wire between modules. If the neighbor module happens to be nearby, the state-wire load will be small, but if it happens to be far away, it may be much larger. The module designer cannot know the exact length of the state-wire until the module has been placed in the system. This variable state-wire load imposes the RT constraints on all the GasP modules.

Using the RT approach, two important failure modes were identified for the intended behavior of the GasP circuits in [11]. Apart from these two constraints the GasP control circuits are required to satisfy two other RT constraints for desired operation. These four constraints are illustrated the following sections and the behavior of the GasP design is represented by up- and down-going transitions (+ or -) of key signals to the latches and neighboring GasP stages: FIRE, PRED, and SUCC. In all these constraints the delay of the dotted path is required to be less than that of the solid path. These four RT constraints are broken down into two groups for conciseness as follows:

2.1.1 Rail to rail constraints

The rail to rail constraints ensure that the state-wire is able to reach the power and ground rails. This ensures that every circuit in the GasP control pipeline successfully

completes the handshake with its nearest neighbors. Note that the 6-4 GasP circuit shown in Figure 1 has sets of five inverting gates that form closed loops. One such loop, called the successor loop, involves gates A B C D E. Whenever the inputs to the NOR gate A cause gate A to act, the successor loop will change the state of the successor state-wire in such a way as to discontinue that action. Similarly, the five gates A B C X F form a predecessor loop that serves a similar purpose. The five gates in each loop form, in effect, a pair of five-inverter ring-oscillators coupled to neighbors by the NOR logic function inside the GasP module and the state-wires through which the module communicates with its neighbors. It is also important to notice that these two loops in each GasP module meet at the NOR gate. Completion of either of the loops shuts off both of them. Failure can result if either of these two loops completes before the other is adequately underway. This results into two timing constraints that involve the difference in delay of the two loops.

2.1.1.1 Predecessor loop constraint on the successor state-wire

The first timing constraint, illustrated in Figure 4, states that PRED of the next GasP stage should go high before PMOS gate E of the current stage stops driving it; see Figure 1. This constraint has a margin of four gate delays. However, the presence of a large state-wire load from SUCC of stage a1 to PRED of stage a2 and a small state-wire load on PRED of stage a1 reduces this margin and may lead to a violation of this constraint. Moreover, if PMOS gate E in stage a1 fails to drive the state-wire all the way up before the predecessor loop shuts E off, data moving forward may be lost.

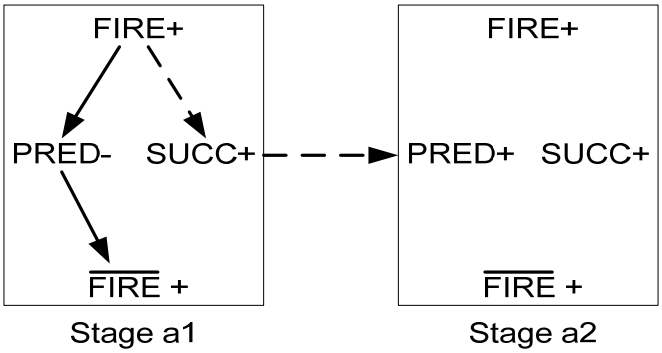

Figure 4: Predecessor loop constraint on the successor state-wire

2.1.1.2 Successor loop constraint on the predecessor state-wire

The second timing constraint, illustrated in Figure 5, states that SUCC of the previous GasP stage should go low before NMOS gate X of the current stage stops driving it; see Figure 1. This constraint has a margin of four gate delays. However, the presence of a large state-wire load from PRED of stage a2 to SUCC of stage a1 and a small state-wire load on SUCC of stage a2 reduces this margin and may lead to a violation of this constraint. Moreover, if NMOS gate X in stage a2 fails to drive the state-wire all the way down before the successor loop shuts X off, bubbles moving backward may be lost, resulting in duplication of data.

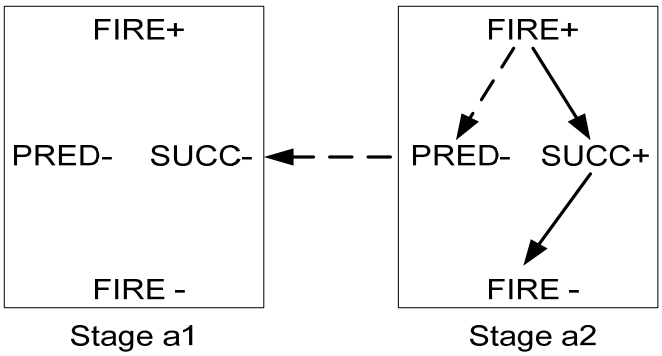

Figure 5: Successor loop constraint on the predecessor state-wire

Notice that each constraint involves the relative ordering of two actions on the state-wire that leads to an adjacent module. When the first of the two actions completes, it turns off the driving action of both loops. Thus a very fast predecessor loop may prematurely terminate the drive of a slower successor loop, and vice versa.

2.1.2 Short circuit constraints

Each participant in a single-track signaling protocol must cease driving the state-wire soon enough to make room for the action of the other participant. Were a participant to drive the wire for too long a time, both might drive it concurrently in opposite directions, consuming unnecessary energy and producing an indeterminate logic signal. Some single-track systems [7][18] make use of the analog properties of the state-wire. Each participant drives the wire “long enough” for it to pass some threshold voltage that will alert the other participant. Other single-track systems, including GasP, depend on the relative timing of logic gates to avoid drive conflict at the state-wire. This results in two timing constraints, one for each state-wire.

2.1.2.1 Short circuit constraint on the successor state-wire

The third timing constraint, illustrated in Figure 6, states that the PMOS gate E of the current stage must cease driving the state-wire before the NMOS gate X of the next stage is turned on. This constraint has a margin of zero gate delay and hence is really tight.

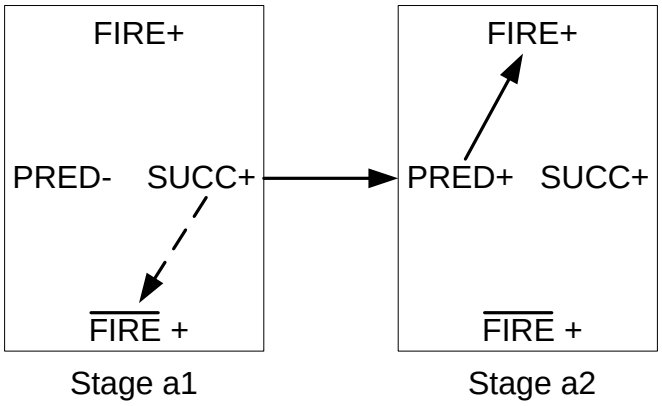

Figure 6: Short circuit constraint on the successor state-wire

2.1.2.1 Short circuit constraint on the predecessor state-wire

The fourth timing constraint, illustrated in Figure 7, states that the NMOS gate X of the current stage must cease driving the state-wire before the PMOS gate E of the previous stage is turned on. This constraint has a margin of zero gate delay and hence is really tight.

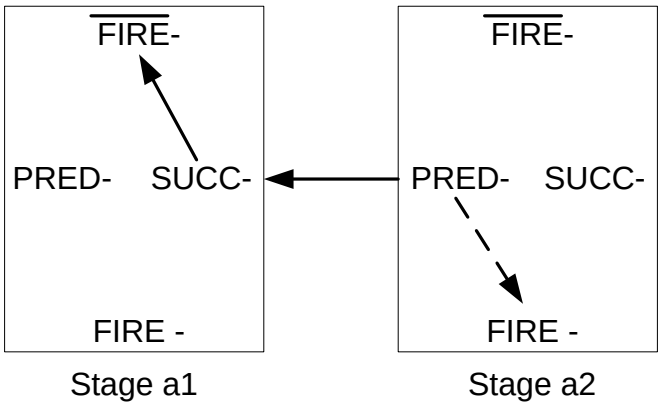

Figure 7: Short circuit constraints on the predecessor state-wire

Notice that both the short circuit constraints have a margin of zero gate delays and hence it is evident that these constraints will be violated much of the time. However these

constraints are specifically used for margin analysis. A margin is set up by the designer for the amount of short circuit allowed in the design. If there are violations beyond this margin, then there is increase in the total dynamic power consumption. However, it is important to note that the violation of this constraint does not result in functional failures.

2.2 Relative Timing (RT) constraints on the data-path

The GasP control circuits are used to clock the data-path consisting of latches. It is thereby essential for the data to meet setup and hold checks at every latch. An example for these checks is illustrated in Figure 8.

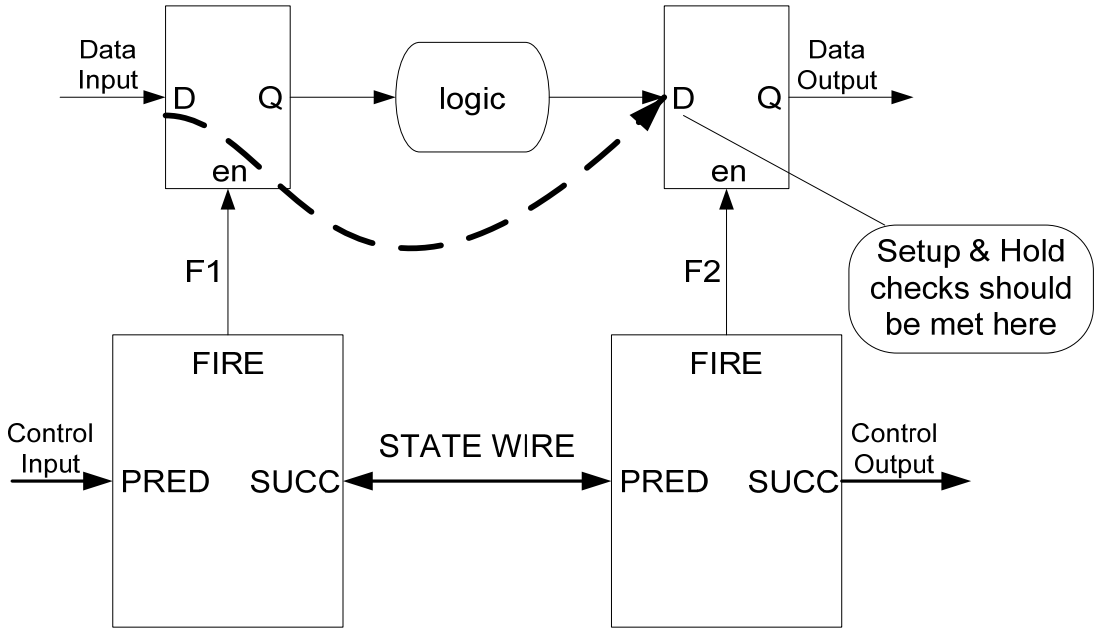

Figure 8: Setup and Hold Checks

The entire FLEET architecture has been based on the premise that time borrowing in latches is an added advantage and not a luxury. Hence for verification

purposes, we should assume that no time borrowing is allowed between latches. In this scenario, the data has to be available before the transparent phase of the clock starts.

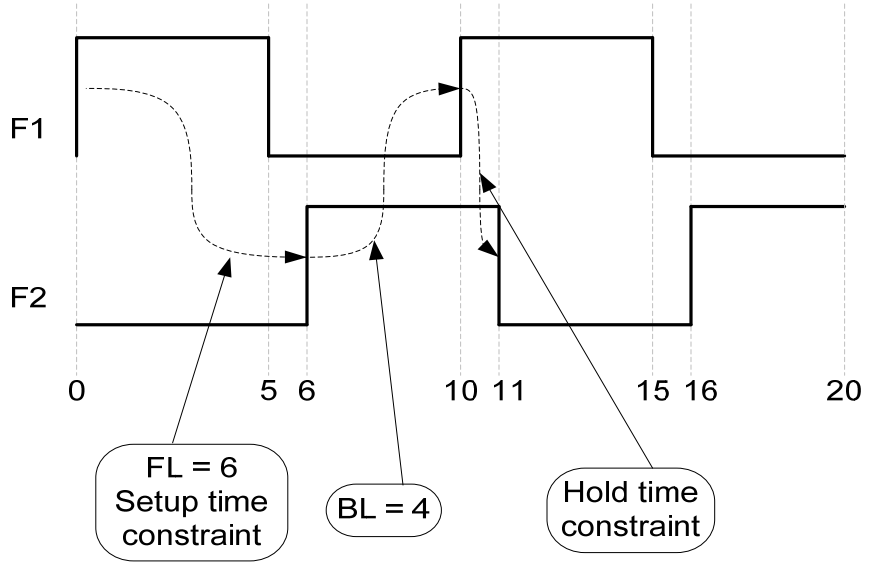

Figure 9: RT constraints on the data-path

It is important to recollect that the FIRE signal of the current stage (F2) goes high six gate delays after the FIRE signal of the previous stage (F1) had gone high. This is illustrated in Figure 9, which assumes a 10 gate delay cycle time, a forward latency(FL) of 6 gate delays and a backward latency(BL) of 4 gate delays. In order to meet the setup requirement, the data values coming from the F1-enabled latches should reach the F2-enabled latches before the start of the five gate-delay transparency window for F2. From Figure 9, we can see that this means the transition time from the data inputs of the F1-enabled latches to the data inputs of the F2-enabled latches can be a maximum of six gate delays. For the hold time violations the second data should not corrupt the first data. In order to meet the hold requirement, the next data values coming from the F1-enabled

latches should not reach the F2-enabled latches before the end of the current five gate-delay transparency window for F2. From Figure 9, we can see that this means the transition time from the data inputs of the F1-enabled latches to the data inputs of the F2-enabled latches has to be at least one gate delay.

Chapter 3

Library Characterization for GasP

The industry standard format for representing delay and power information of a library is the liberty format. The liberty description identifies the characteristics of a technology and the cells it contains. The procedure of creating a liberty file for single track circuits was shown in [13]. This thesis uses the same procedure to characterize the delay information of GasP cells. The delay model used is the non-linear delay model as it provides a reasonable tradeoff between accuracy and complexity. This delay model uses lookup tables indexed by input slews and load capacitance. There are three steps involved in such a delay characterization procedure. The first step is to identify the various timing arcs present in the different cells to be characterized. The second step is to carry out the spice simulations required to measure the propagation times and the slew rates for each of these arcs. The final step involves measuring the capacitance of the different pins present in the cell. The following sections go over these steps in detail.

3.1 Defining timing arcs

The timing arcs that identify the behavior of the GasP cells can be represented using a signal transition diagram as shown in Figure 10. In the figure, the ‘+’ symbol indicates a rising transition, the ‘-’ symbol indicate the falling transition, the ‘0Z’ symbol indicates the low to tri-state transition and the ‘1Z’ symbol indicates the high to tri-state transition. Notice that all the timing paths used to define the timing constraints in Chapter 2 are covered in this signal transition diagram.

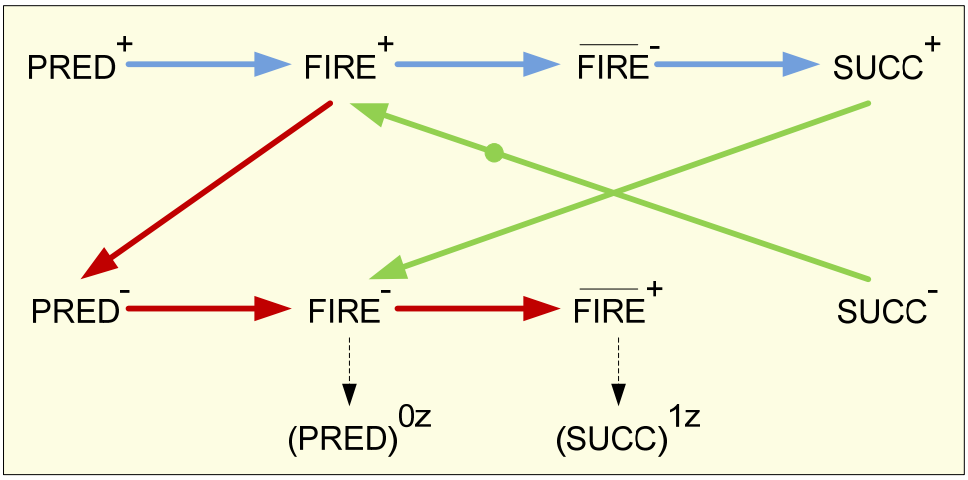

Figure 10 : Complete signal transition diagram for GasP

In order to reduce the complexity of the characterization process and to achieve a faster solution, we decided to compromise on accuracy by not including the $\overline{\text{FIRE}}$ pin in the flow. The reason behind excluding $\overline{\text{FIRE}}$ came from the fact that the timing arcs from FIRE to $\overline{\text{FIRE}}$ were internal to the cell. The two arcs FIRE- to $\overline{\text{FIRE}}$ + and FIRE+ to $\overline{\text{FIRE}}$ - represented the rising and falling delays of the inverter D in Figure 1. These delays will be essentially the same for all the GasP cells and hence can be easily estimated. These estimated values can then be used as timing margins while verifying the Relative Timing constraints on the control cells which will be discussed in Chapter 4.

The reduced signal transition diagram for GasP is shown in Figure 11. Notice that removing the $\overline{\text{FIRE}}$ pin does not affect the timing paths passing through it. However, the timing paths which would have terminated on $\overline{\text{FIRE}}$ will now have to terminate on FIRE.

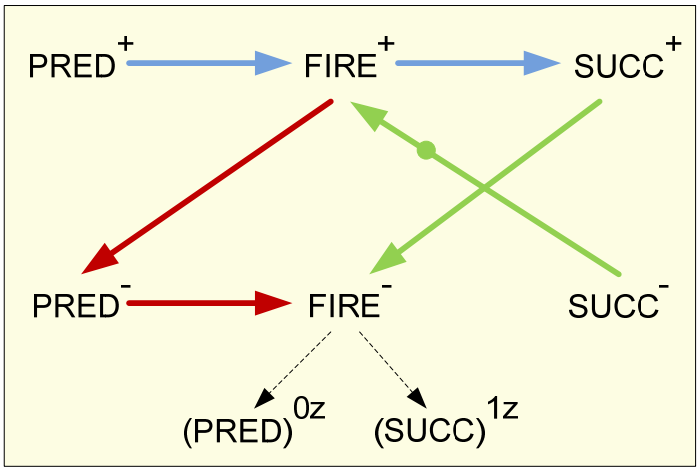

Figure 11: Reduced signal transition diagram for GasP

3.2 Characterization of the timing arcs

3.2.1 Setting up the simulation environments

After identifying the timing arcs, the next step is to create appropriate simulation environments to characterize them. As mentioned earlier, the liberty file consists of lookup tables indexed by input slews and load capacitance. In order to generate real world input waveforms, the pre-driver method mentioned in [13] was used. This method is shown in Figure 12 which uses a variable sized buffer B and a variable load capacitance C_L. In order to characterize the timing arc PRED+ to FIRE+, the size of the input buffer B is changed for generating different input slews. Also, the load C_L is varied based on the cell drive strength to generate a 2D table (6 x 6) for each timing arc. The table thus generated is expected to have enough points for interpolation and extrapolation.

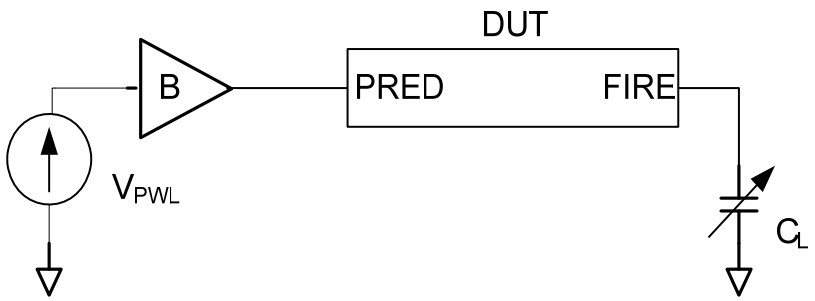

Figure 12: Pre-driver method

3.2.2 Measuring arc delays

The industry standard for performing the worst case analysis to encounter on-chip variation is to implement multiple timing libraries. The general paradigm of having fast and slow timing libraries allows a tool like PrimeTime to choose minimum and maximum delays for different timing paths. Recollect that all the RT constraints are of the form where one timing path A has to be shorter than the other timing path B. For worst case analysis, we need to verify that the maximum delay through timing path A has to be smaller than the minimum delay through timing path B. This is a conservative, yet safe approach. Thus, two timing libraries having the minimum and maximum delays for each arc are required for verification. In order to speed up the process of making the timing libraries, we chose to be less conservative by taking only the simultaneous switching effects of the NOR gate into account. Another more conservative approach of making the timing libraries would be to consider fast and slow circuit corners for each gate in the design.

These two timing libraries were generated by using different initial conditions while measuring the various timing arcs. These conditions are articulated as follows:

1. PRED+ to FIRE+

    a. Fast .lib: SUCC pin of the DUT is held at 0 with no initial conditions on the other pins.

    b. Slow .lib: SUCC pin of the DUT changes from 1 to 0 at the same time as the inverted PRED signal. This can be done by putting an inverter on the SUCC pin which is of the same strength as the inverter F shown in Figure 1.

2. FIRE+ to SUCC+

    a. Fast .lib: SUCC pin is set to 0 by using .ic command in spice.

    b. Slow .lib: Same as above.

3. FIRE+ to PRED-

    a. Fast .lib: PRED pin is set to 1 by using .ic command in spice.

    b. Slow .lib: Same as above.

4. PRED- to FIRE-

    a. Fast .lib: SUCC pin of the DUT changes from 0 to 1 at the same time as the inverted PRED signal. This can be done by putting an inverter on the SUCC pin which is of the same strength as the inverter F shown in Figure 1.

    b. Slow .lib: SUCC pin of the DUT is held at 0 with no initial conditions on the other pins.

5. SUCC+ to FIRE-

    a. Fast .lib: The inverted PRED signal of the DUT changes from 0 to 1 at the same time as the SUCC pin changes from 0 to 1. This can be done by putting an

inverter on the SUCC pin which is of the same strength as the inverter F shown in Figure 1.

b. Slow .lib: PRED pin of the DUT is held at 1 with no initial conditions on the other pins.

6. SUCC- to FIRE+

a. Fast .lib: PRED pin of the DUT is held at 1 with no initial conditions on the other pins.

b. Slow .lib: The inverted PRED of the DUT changes from 1 to 0 at the same time as the SUCC pin changes from 1 to 0. This can be done by putting an inverter on the SUCC pin which is of the same strength as the inverter F shown in Figure 1.

3.3 Measuring pin capacitances:

The liberty format requires pin capacitances for all pins. A standard delay matching technique used to measure these pin capacitances is shown in Figure 13. In order to measure the capacitance on PRED, the input to output delay ‘d1’ of buffer B is measured. The DUT is then replaced by a variable capacitor C_V and its value is swept till the input to output delay ‘d2’ of buffer B matched the delay ‘d1’. The capacitance at which the delays match is the capacitance of PRED.

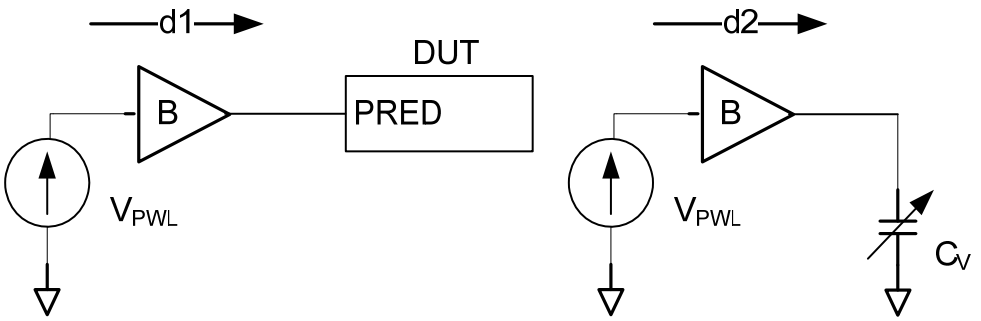

Figure 13: Measuring pin capacitances

3.4 Generating the Liberty files

Sun's internal CAD tool Electric [6] was used to generate the two liberty files. The flow used by Electric is shown in Figure 14, where it uses the layout and a constraint file as inputs for generating the final liberty file. Electric automatically generates the required Spice netlists and their stimuli by using the information provided in the constraint file. These files are then fed to Hspice for simulation and performing the necessary measurements. The data output from Hspice which is in the .mt# file format is used by Electric to generate the liberty file.

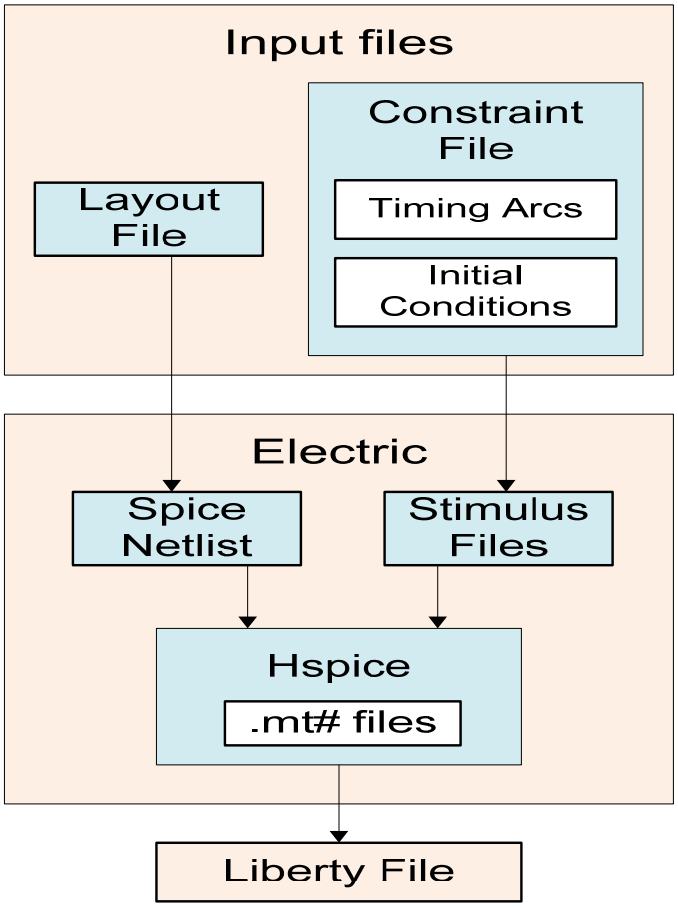

Figure 14: Characterization flow

Chapter 4

Static Timing Analysis for GasP

After the generation of the timing libraries, the last step is to verify the articulated timing constraints using static timing analysis. Static timing analysis validates the timing performance of a design by checking all possible paths for timing violations. Synopsys PrimeTime is a gold standard static timing analysis tool which measures minimum and maximum delays through timing paths and verifies them against the given timing constraints. However, PrimeTime is a tool designed to verify synchronous designs and hence it heavily relies on the notion of having a global clock against which it verifies the timing constraints. As a result of this, a number of challenges were encountered in making PrimeTime correctly interpret a netlist containing asynchronous cells like GasP. The first part of this chapter discusses these problems and their solutions. These solutions are then used to develop a complete flow for timing verification as shown in the second part.

4.1 Challenges in interpreting asynchronous netlists

4.1.1 Bi-directional pins

In the single track protocol, a single state-wire is driven by the two communicating modules that are connected to it. As a result of this the PRED and SUCC pins act as both input as well as output pins and hence are termed as bi-directional pins. When these pins are instantiated in a Verilog netlist that is fed to PrimeTime, each pin is broken by PrimeTime as a pin having one input port and one output port. We believe that it is a characteristic feature of PrimeTime to break the timing paths at every input port.

This was inferred from observing that when a netlist having bi-directional pins was given to PrimeTime, it generated empty timing graphs. As the timing paths are broken at bi-directional pins, the timing information is lost at these pins as shown in Figure 15.

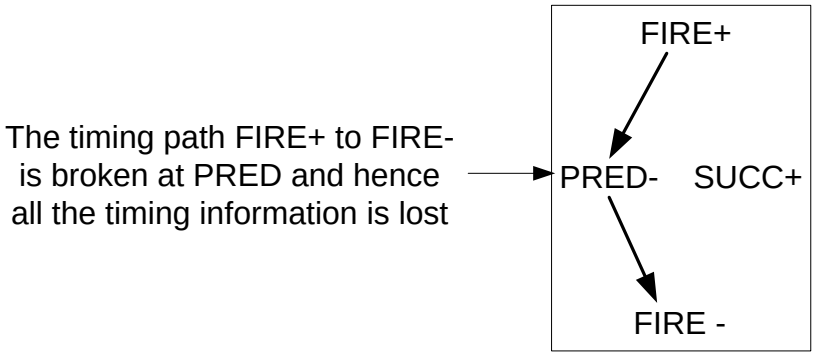

Figure 15: Bi-directional pin problem

In order to overcome this problem pertaining to the bi-directional pins, we propose a split pin architecture shown in Figure 16. The split pin architecture requires inherent changes to be made to the Verilog netlist as well as the timing libraries. In the Verilog netlist, all the bidirectional pins need to be split into different input and output pins. As shown in Figure 16, PRED is split into PRED_IN and PRED_OUT and SUCC is split into SUCC_IN and SUCC_OUT. The PRED_IN pin of a GasP stage is connected to the SUCC_OUT pin of the previous stage and the SUCC_IN pin of a GasP stage is connected to the PRED_OUT pin of the next stage as shown in Figure 17. Similar changes are required to be made to the timing libraries where timing information pertaining to the PRED and SUCC pins is distributed among these separate input and output pins. As a consequence of splitting the pins, the measured short circuit constraints would be less conservative. However, this effect can be compensated by taking the effect

of short circuit constraint into account during the characterization of the timing arcs FIRE+ to PRED- and FIRE+ to SUCC+.

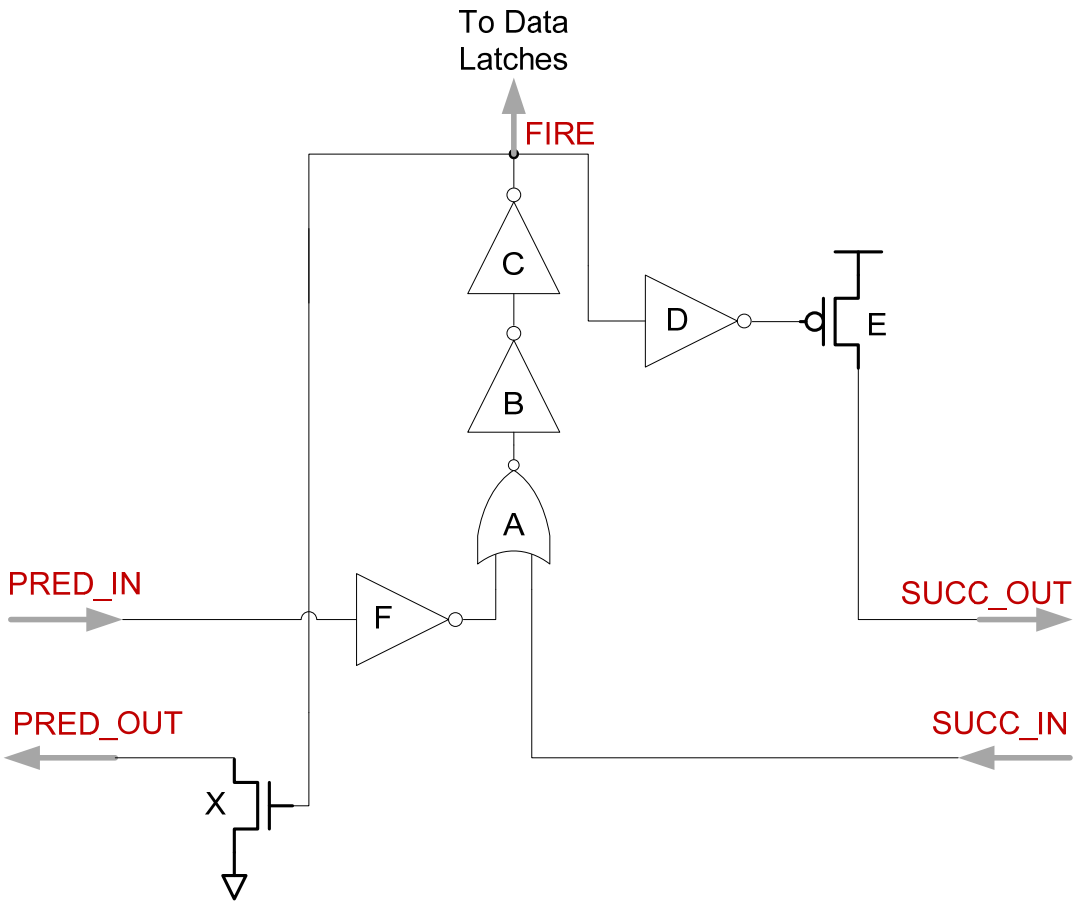

Figure 16: Split pin architecture

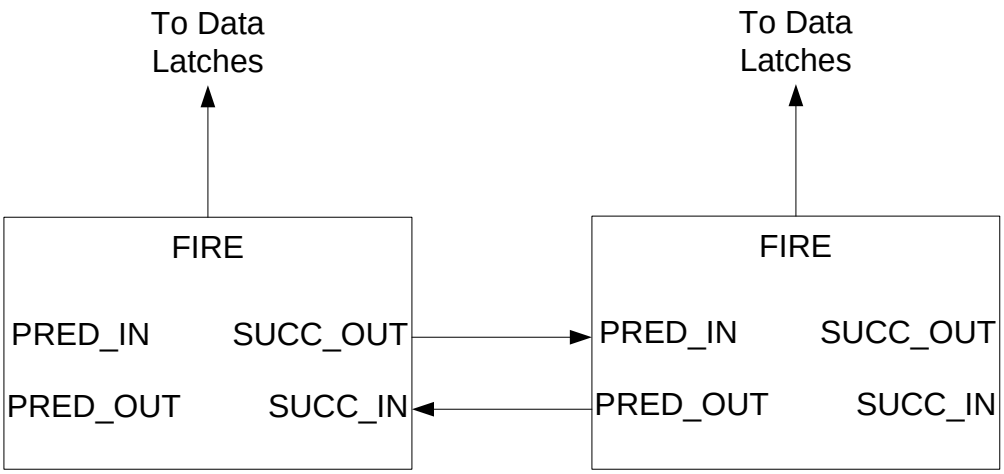

Figure 17: Connectivity of the split pin architecture

4.1.2 Handling loops

As all asynchronous architectures are designed using some handshaking protocol, there are implicit combinational loops present in such designs. The GasP architecture also has inherent timing loops, one of which is shown in Figure 18.

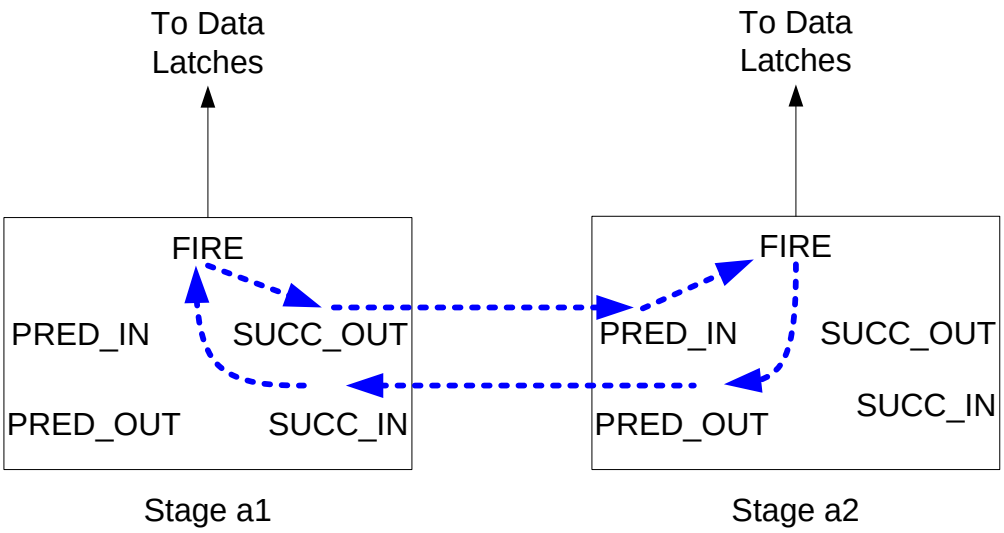

Figure 18: Loops in GasP

PrimeTime has two loop breaking techniques: static loop breaking and dynamic loop breaking. The static loop breaking technique can disable timing paths of interest and thus result in incorrect timing reports. Although, the dynamic loop breaking does not disable any timing path, it is impractical due to the large run-times and heavy memory usage associated with it. Hence, both these techniques do not work for asynchronous circuits [13].

4.1.2.1 Explicitly breaking timing loops

As shown in [13], the timing loops are required to be broken explicitly by using the command *set_disable_timing*. The timing arcs to be disabled have been carefully chosen so that the critical timing information is never lost. The timing arc from SUCC_IN to FIRE is a non critical arc for the predecessor loop constraint on the successor state-wire as well as the short circuit constraint on the successor state-wire. Hence this arc can be disabled while verifying these two constraints. Similarly, the timing arc from PRED_IN to FIRE can be disabled while verifying the successor loop constraint on the predecessor state-wire and the short circuit constraint on the predecessor state-wire.

4.1.2.2 Loops that cannot be broken

When a timing loop consists of only two timing arcs, the delay and slew information of one timing arc is dependent on the delay and slew information of the other timing arc. Hence both timing arcs are critical and disabling any one arc will result in incorrect delay calculations. The GasP cells have two such timing arcs out of which one is shown in Figure 19.

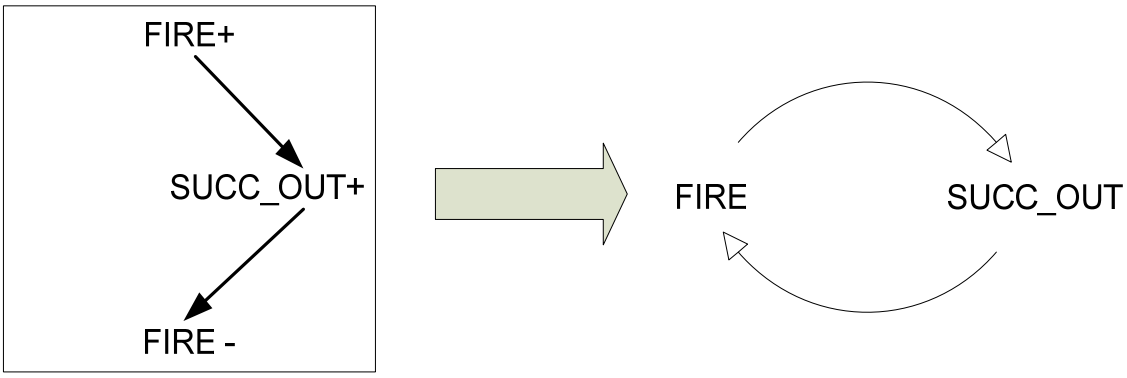

Figure 19: Loops that cannot be broken

Since both the timing arcs are critical from a timing point of view, we decided not to disable either of them. Instead we chose to add a pseudo pin named FIRE_PS as shown in Figure 20. Adding the pseudo pin resulted in breaking the loop by not allowing it to terminate on FIRE and hence the loss of timing information is prevented. Note that, the FIRE_PS pin is only associated to the falling transition on FIRE. As a result of this, the timing arcs SUCC_OUT+ to FIRE- and PRED_OUT- to FIRE- will now terminate on FIRE_PS. The addition of FIRE_PS requires slight modifications to the Verilog netlist and the timing libraries. The Verilog netlist requires an additional pin in the definition of every GasP module. It is essential to note that this pin should be connected to a pseudo load which is similar to the load on FIRE for obtaining correct timing reports. This can be easily done by either editing the DSPF file or by instantiating a pseudo load in the Verilog netlist so that the load information on FIRE_PS is same as that on FIRE. As far as the changes in the timing libraries are concerned, an additional pin needs to be added to the cell definition of the GasP cell. The timing information pertaining to the falling

edge of FIRE then needs to be removed from the pin definition of FIRE and added to the pin definition of FIRE_PS.

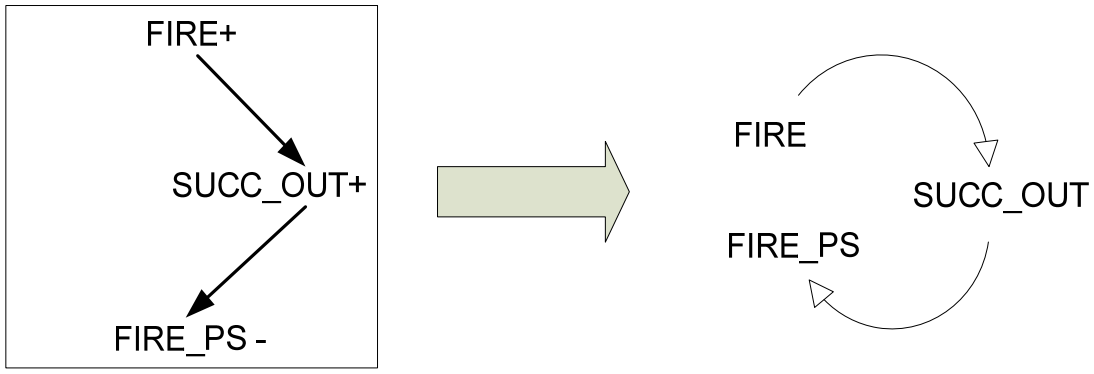

Figure 20 : Adding a pseudo pin

4.1.3 Lack of a global clock

Asynchronous architectures lack the presence of a global clock and rely on local handshakes for data transfer. However, PrimeTime relies on the presence of a global clock to break a design into various timing paths [15]. In the synchronous world, these paths start at a clock edge where data is launched and end at the clock edge where data is captured. In contrast to this, the timing paths for GasP discussed in the previous chapters do not have a guiding clock edge. The method of non-sequential data to data checks was introduced in [13] to overcome this problem. PrimeTime can perform setup and hold checking between two data signals, neither of which is defined to be a clock, at any two pins in the design using the *set_data_check* command [14].

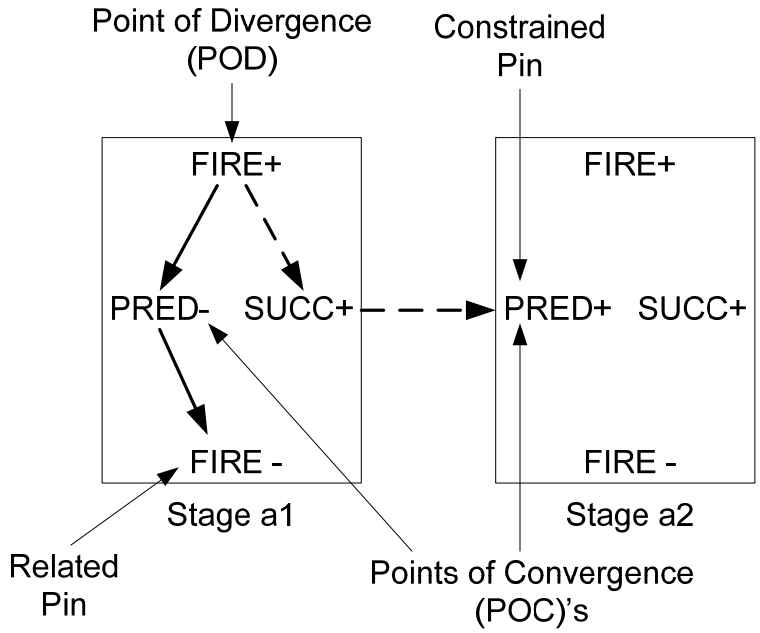

Figure 21: Using set_data_check command

The relative timing constraints stemming from a point of divergence can be modeled using the *set_data_check* command and the two points of convergence are modeled as the constrained and related pins. An example pertaining to the rail-to-rail constraints discussed in section 2.1.1.1 is shown in Figure 21. Here the delay of the dotted path has to be less than the solid path. In other words we want to constraint the rising edge on PRED pin relative to the falling edge on FIRE pin and thus the names constrained pin and related pin. As a consequence, the above data check will be met only if PRED+ happens $delay amount of time before FIRE- happens. Notice that we would have liked to constraint the rising edge on $\overline{\text{FIRE}}$ pin instead of the falling edge on FIRE pin. However we chose to ignore the inverter delay while modeling the timing libraries.

This inverter delay is taken into account while deciding the value of $delay and thus the effect of not having the $\overline{\text{FIRE}}$ pin is made negligible.

4.2 Timing verification flow for GasP

The timing verification flow involves verification of the RT constraints imposed on the GasP control cells as well as on the data-path. In order to verify the RT constraints on the data-path, it is essential to obtain the timing information pertaining to the FIRE signals which act as locally generated clock signals for the data-path. We thereby divide the complete flow into two parts. In the first part we verify the RT constraints on the GasP control cells and obtain the phase differences between the FIRE signals. Using the phase differences obtained from the first part, we can then verify the RT constraints on the data-path in the second part.

4.2.1 Part 1 of the verification flow

4.2.1.1 Verifying RT constraints on the control cells

The four RT constraints defined for the GasP control cells are verified in two runs as they require different initial conditions. As mentioned previously the timing arc from SUCC_IN to FIRE is a non critical arc for the predecessor loop constraint on the successor state-wire as well as the short circuit constraint on the successor state-wire. In the first run, this timing arc is disabled and the two constraints are verified for each GasP cell. An example is shown in Figure 22 where the predecessor loop constraint on the successor state-wire is being verified for a linear GasP pipeline. In the second run the timing arc from PRED_IN to FIRE is disabled for the verification of the successor loop

constraint on the predecessor state-wire as well as the short circuit constraint on the predecessor state-wire.

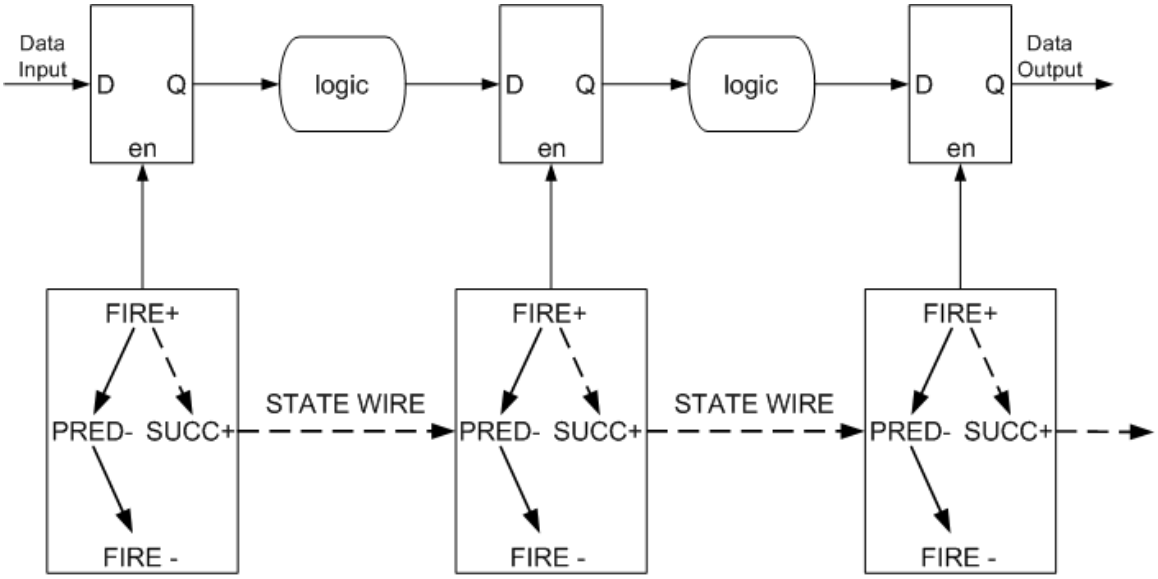

Figure 22: Verifying the RT constraints

4.1.2 Measuring phase differences between fire signals

4.1.2.1 Measuring phase difference for setup checks

In the previous chapters, we explained that the worst case for setup time violation on a latch will be caused due to an early arriving signal clock signal (F2) with respect to the arrival of clock signal (F1) on the previous latch. Thus we need to measure the shortest time from F1 to F2 to take the worst case into account. Recall that all the shortest times are characterized in the fast.lib. Thus by using the delay values from the fast.lib we can measure the relative phase differences between the fire signals as shown in Figure 23.

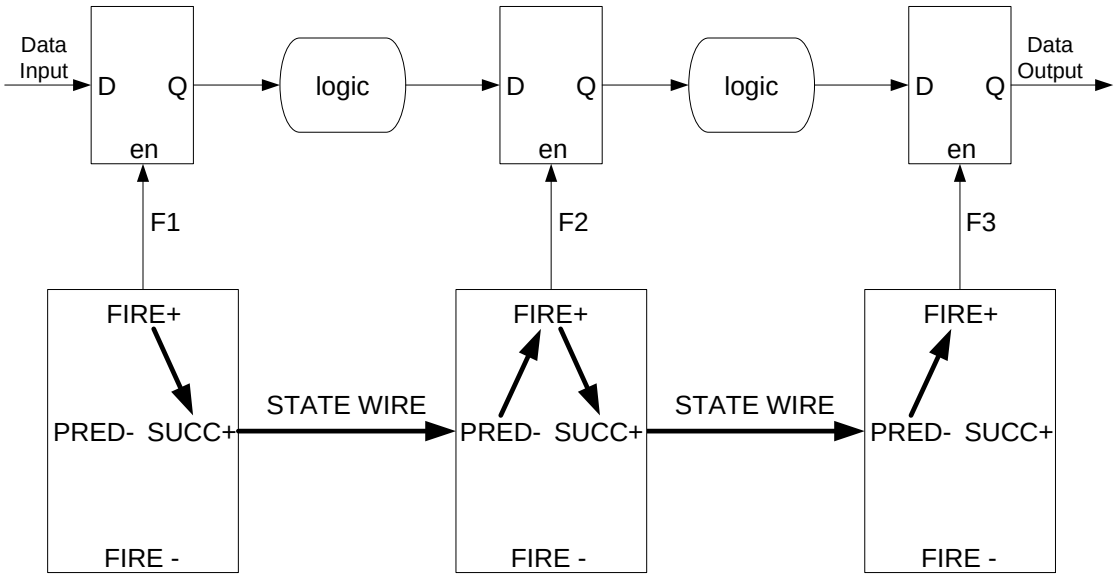

Figure 23: Measuring phase difference for setup checks

4.1.2.2 Measuring phase difference for hold checks

In the previous chapters, we also explained that the worst case for hold time violation F2-enabled latch happens when it is transparent and the clock signal (F1) on the F1-enabled latch arrives early. In this case, both the latches are simultaneously transparent for more time and hence are more likely to cause a hold time violation. Thus we need to find the shortest time from F2 to F1 to take the worst case into account. As all the shortest times are characterized in the fast.lib, we can use those delay values to find the relative phase differences between the fire signals as shown in Figure 24.

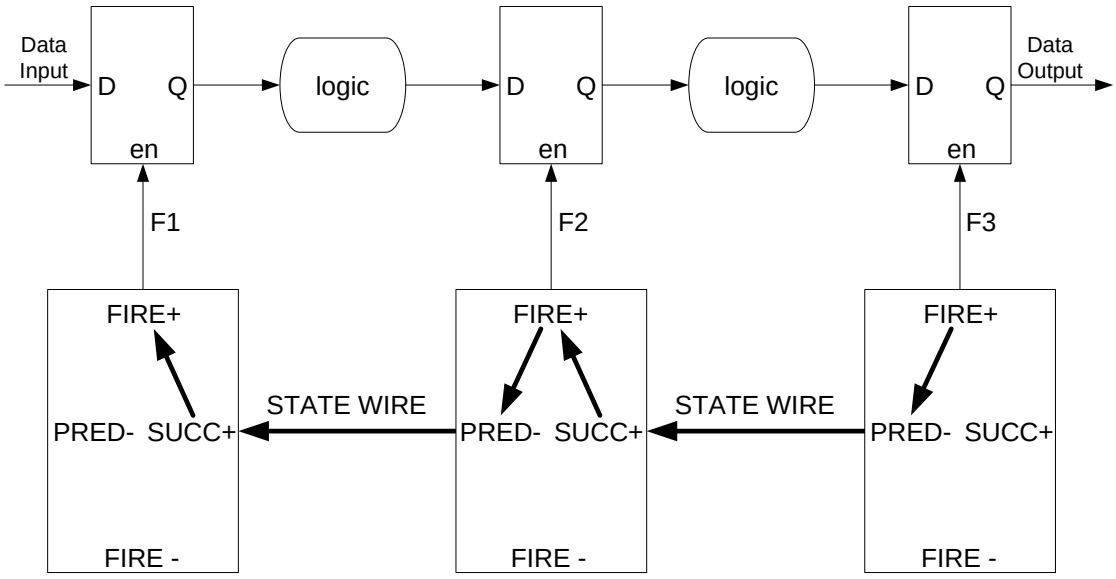

Figure 24: Measuring phase difference for hold checks

4.2.2 Part 2 of the verification flow

The last step in the verification flow is to use the phase differences between the clocks obtained from Part 1 to define the clocks in the PrimeTime script and verify the RT constraints on the data-path as shown in Figure 25. In this step the setup and hold checks are performed separately in two different runs as each run requires different clocks.

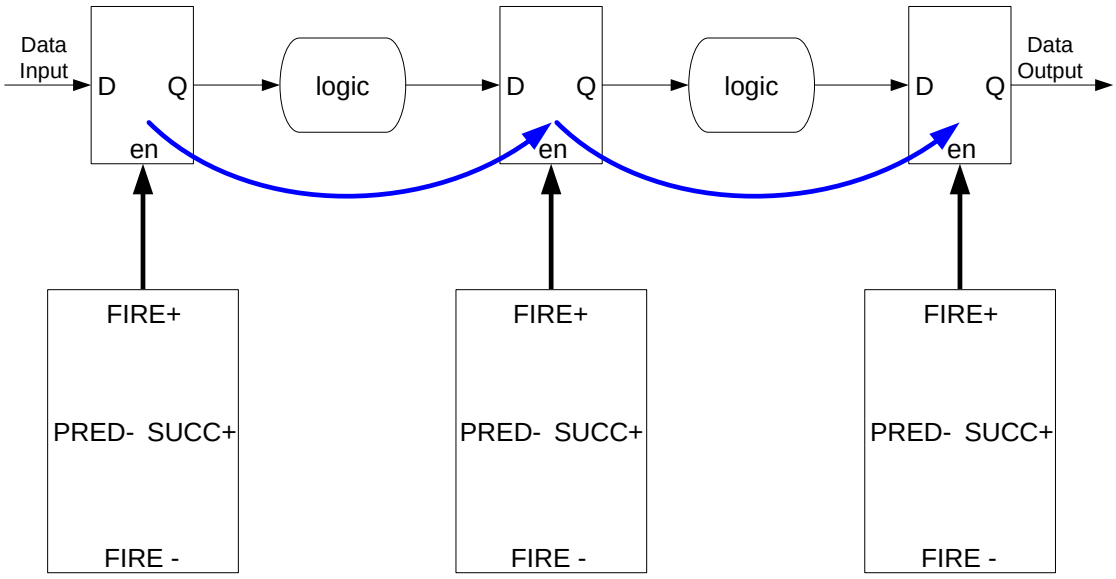

Figure 25: Verifying the RT constraints on the data-path

Chapter 5

Evaluating the effects of operating environments

After verifying the relative timing constraints using PrimeTime, the next step is to interpret the timing reports and to determine if any timing constraint was violated. Although PrimeTime verifies the timing constraints correctly, it offers very little insight for the designer in understanding the cause of timing violations in these complicated single track circuits. In this chapter, we discuss the effects of the operating environments on the performance of an asynchronous pipeline to identify the worst cases of operation for the RT constraints defined in Chapter 2. Using the analysis presented in this chapter, the designer can then simulate the violated paths using dynamic simulation to determine the source of the violation. This analysis can also be useful during the functional verification of these circuits.

The effects of the operating environments can be observed at the micro level as well as the macro level. At the micro level, the speed of the cells used in the pipeline can be affected by the arrival times of their inputs. This phenomenon has been elaborated as the Charlie Effect. At the macro level, the throughput of the pipeline is affected by the region in which they operate. In the subsequent sections, we first give a background on these two effects and then discuss their impacts on the different RT constraints.

5.1 Background

5.1.1 The Charlie Effect

The Charlie Diagram, named in the honor of the late Charles E. Molnar specifies the delay through a RendezVous element as a function of the separation time between the input arrivals [5]. For a two-input NOR gate shown in Figure 26, let t_A and t_B be the arrival times of the inputs A and B. Both the inputs have high-to-low transitions which results in the output C to have a low-to-high transition. Let the arrival time of the output be denoted by t_C.

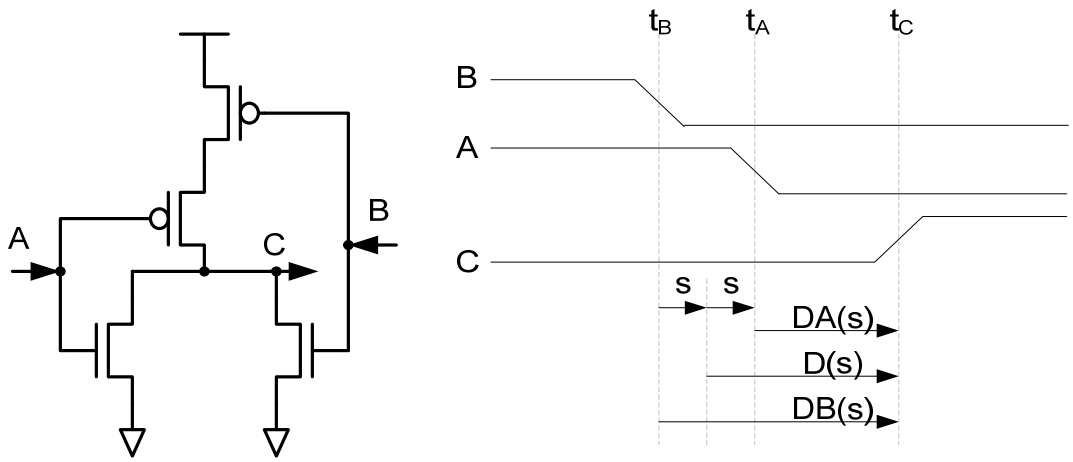

Figure 26: NOR gate

Charlie Diagrams measure the output delay of a gate from the average of the arrival times of the input events. Figure 27, shows a typical Charlie Diagram where this delay is denoted by D(s) and is plotted as a function of s, where s is the half the separation time between the input arrivals. When $s > 0$, the input A arrives last, and when $s < 0$ the input B arrives last.

In summary,

$$D(s) = t_C - (t_A + t_B)/2$$

$$s = (t_A - t_B)/2$$

As seen in Figure 27, the curve of D(s) versus s resembles a hyperbola. For large separations of the input events, the output time approaches the time of the last input plus some constant. Thus, the Charlie Diagram has asymptotes with slopes of +/- 1 [19].

The input to output delays with respect to the individual inputs is represented as follows:

$$DA(s) = t_C - t_A = D(s) - s$$

$$DB(s) = t_C - t_B = D(s) + s$$

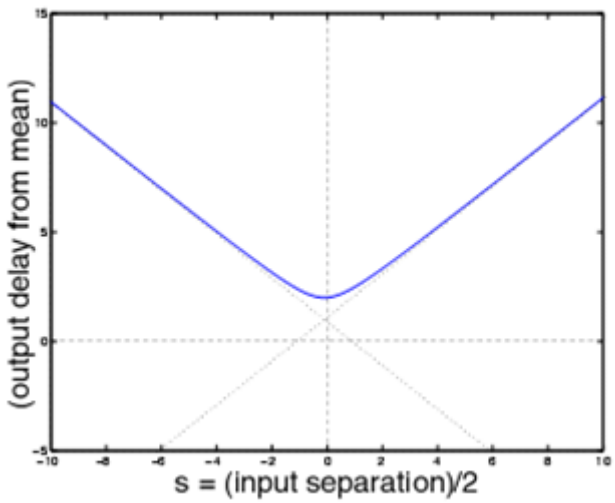

Figure 27: Charlie Diagram

Consider a scenario where input A changes after input B. If A changes a long time after B, the NMOS controlled by B will be in its cut-off region and the PMOS

controlled by B will be fully conducting as A changes. Furthermore, the node between the two PMOS transistors will be close to the power rail which allows a relatively fast transition on the output C. On the other hand, if A changes only slightly after B, then the two transistors controlled by B will both be partially conducting as A changes. This results in a greater delay from the transition of A to the transition of C. Similar effects occur if A changes before B. These are the simultaneous switching effects described in [3][4].

The dependence of the output delay on the relative arrival times described above is reflected in the curve of the Charlie Diagram approaching the asymptotes monotonically from above. Returning to the scenario where input A changes after B, we note that if A changes a long time after B, then s is large and positive and the delay DA(s) is small. Conversely, if A changes only slightly after B, then s is smaller and the delay DA(s) increases. Because of the dependence of gate delay on the relative arrival time of the inputs is naturally modeled by Charlie Diagram, this dependence is termed as the "Charlie Effect".

5.1.2 Operating regions

The throughput of all asynchronous linear pipelines depends on the region in which they operate. Based on the availability of a data or a space (i.e. bubble), these pipelines operate in the data-limited region, the bubble-limited region or the full throughput region. In the data-limited region, the speed of the pipeline is limited by the availability of the data and is illustrated in Figure 28. In this region the throughput of the pipeline increases as the data values are inserted more frequently into the pipeline.

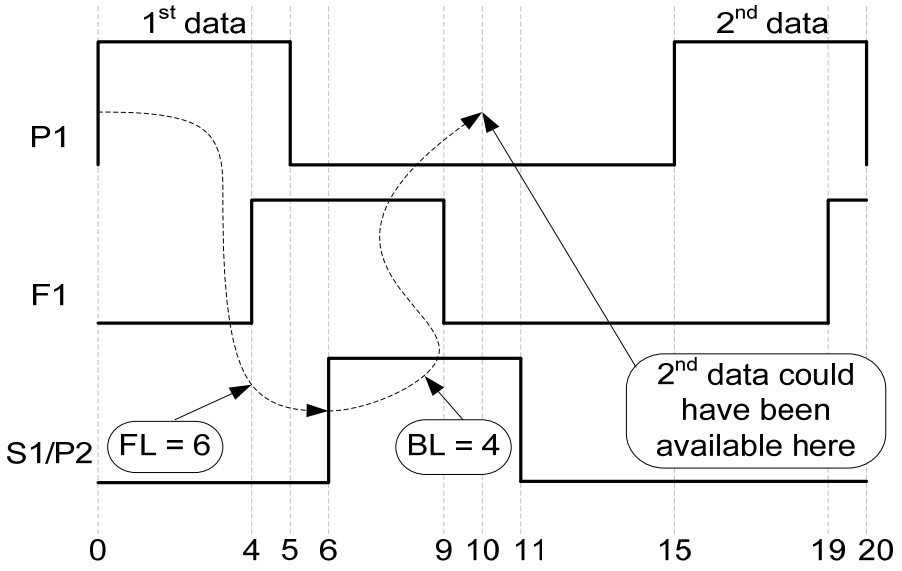

Figure 28: Data limited region

In the bubble limited region, the output environment cannot consume the data provided by the pipeline and is illustrated in Figure 29. This results in the successor state-wire remaining full for a longer duration. As a result of this, the data values are stalled at the predecessor state-wire and hence they start accumulating in the pipeline.

In the full throughput region, the next data is inserted into the first stage of the pipeline at the same time as the second stage consumes the first data and thus creates a space for the next token. This is illustrated in Figure 30. Notice that in the full throughput case both the inputs of the NOR gate switch at the same time. As a result of this, all the signals have a cycle time of 10 gate delays because the two ring oscillators having five inverting gates are running at full speed.

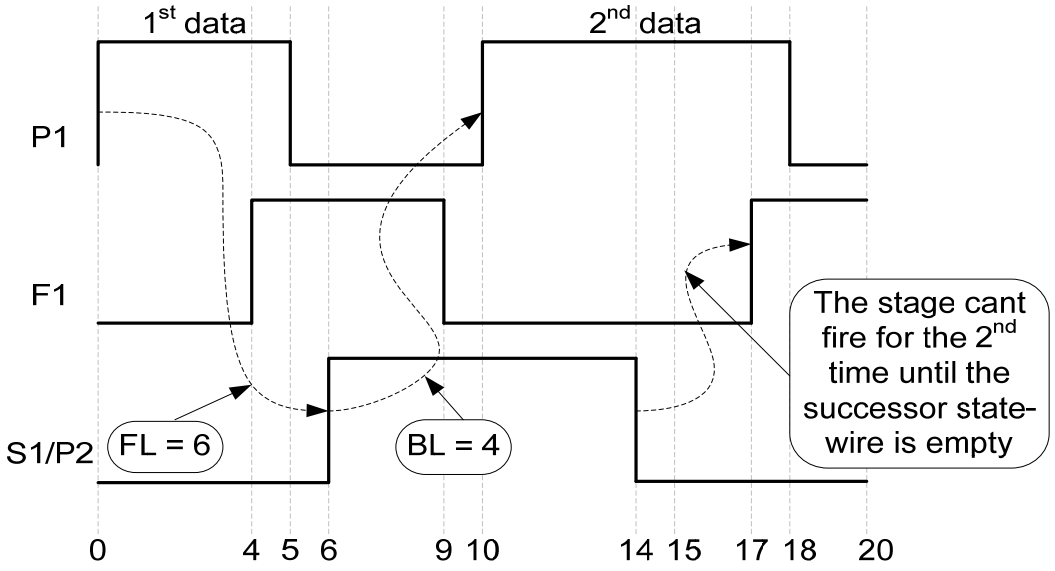

Figure 29: Bubble limited region

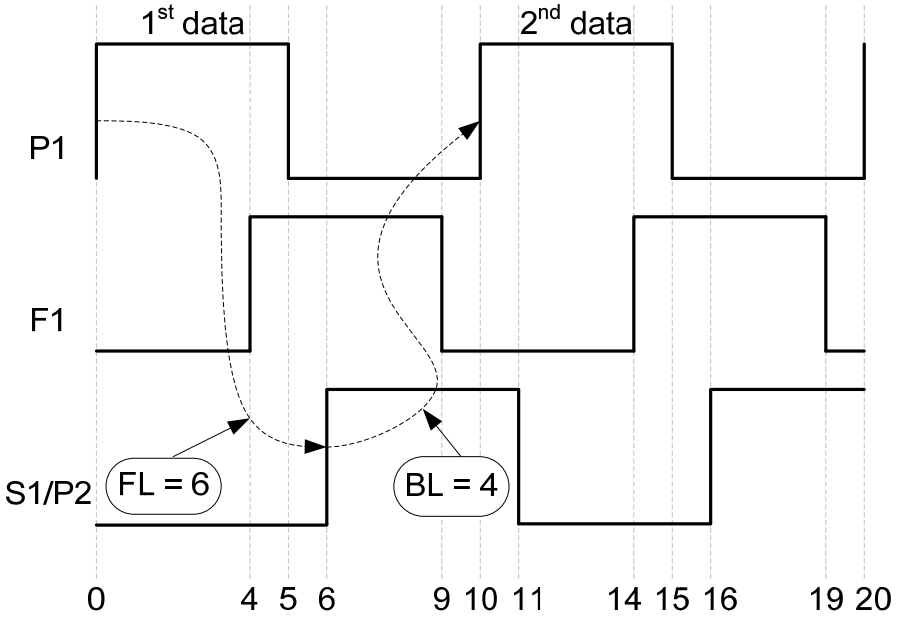

Figure 30: Full throughput region

5.2 Charlie Effects on the RT constraints for control logic

The three operating conditions result in different delays for two timing arcs namely the timing arc SUCC- to $\overline{\text{FIRE}}$ - and the timing arc PRED+ to FIRE+. The timing arc SUCC- to $\overline{\text{FIRE}}$ - is dependent on when the next instruction arrives and the timing arc PRED+ to FIRE+ is dependent on when the next bubble arrives. All the other timing arcs complete by themselves as they are either pass through the single input gates or they pass through the NOR gate for a falling transition on the output of the NOR gate. Note that the falling transition on the output of the NOR gate requires the presence of only one input.

The Charlie Effect affects only the rising transition on the output of the NOR gate where the speed of the NOR gate varies depending on the arrival times of its two inputs. The NOR gate has minimum delay when one of the inputs arrives earlier than the other input. This happens in the data limited case and the bubble limited case. The NOR gate has maximum delay when both the inputs arrive at the same time which happens in the full throughput case. Even though the timing arc SUCC- to $\overline{\text{FIRE}}$ - is dependent on the arrival of the next instruction, it has minimum delay for the bubble limited case. Similarly, though the timing arc PRED+ to FIRE+ is dependent on the arrival of the next bubble, it has minimum delay for the data limited case.

Both the timing arcs SUCC- to $\overline{\text{FIRE}}$ - and PRED+ to FIRE+ are used for the evaluation of the short circuit constraints. As shown in Figure 6 and Figure 7, both these

arcs fall on the solid path which has to be larger than the dotted path. Hence the worst case for these constraints happens when these timing arcs have minimum delays. In conclusion, the data limited case and the bubble limited case must be used while verifying these short circuit constraints through dynamic simulation.

5.3 Charlie Effects on the RT constraints for the data-path

5.3.1 Finding the worst case for the setup constraint

Let us consider a transition phase of a GasP pipeline from the full throughput case into the data limited case. This will help us evaluate the difference between the two cases. As mentioned previously, the NOR gate has a maximum delay in the full throughput case. Let us take this delay to be 1 gate delay. We also know that in the data limited case, the NOR gate operates slightly faster. Let us take this delay to be 0.5 gate delay. After making these assumptions, we can make our analysis based on Figure 31.

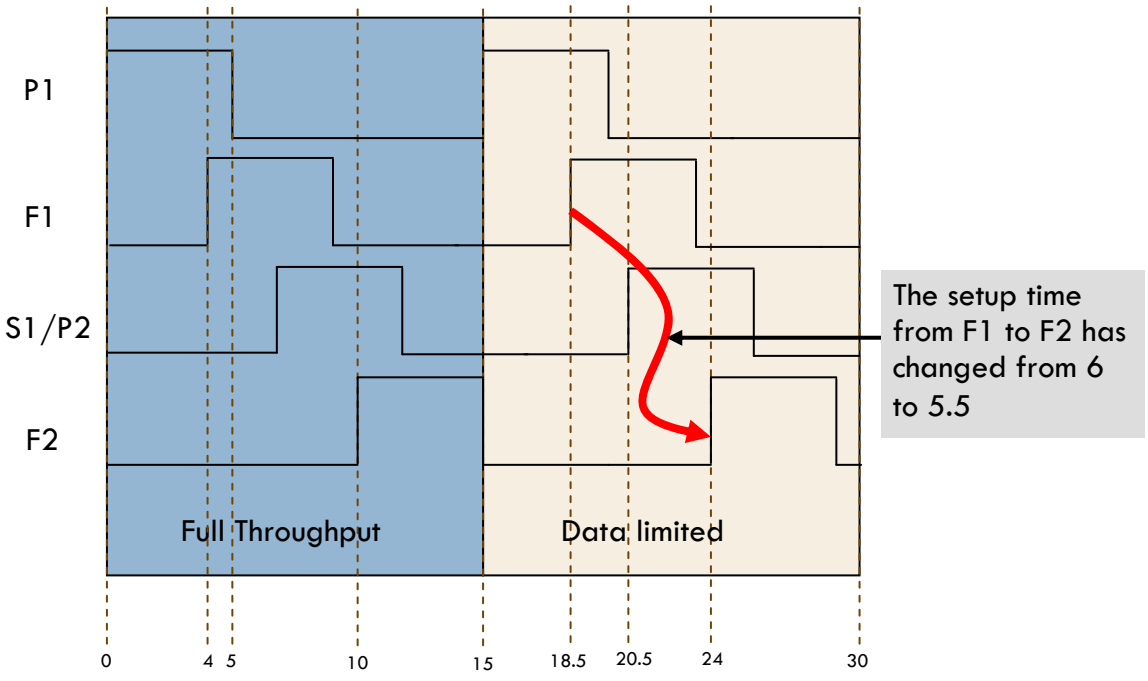

Figure 31: Worst case for setup checks

As illustrated in Figure 31, the setup check from F1 to F2 is 6 gate delays in the full throughput case and is reduced to 5.5 in the data limited case due to a slightly faster NOR gate. This leads us to the conclusion that the data limited case is worse than the full throughput case. The next question that comes into mind is “what happens in the bubble limited case?”

In the bubble limited case the successor state-wire of a particular cell is stuck at full and hence that cell can’t fire again till its successor state-wire becomes empty. As a result of this, the phase difference between F1 going high and F2 going high will always be greater than 6 gate delays. Hence we can conclude that the data limited case is the worst case for the setup checks.

5.3.2 Finding the worst case for the hold constraint

Let us now consider a transition phase of a GasP pipeline from the full throughput case into the bubble limited case. This will help us evaluate the difference between the two cases. Similar to the previous case we will assume the NOR gate delay to be 1 gate delay in the full throughput case. We also know that in the bubble limited case, the NOR gate operates slightly faster. Let us take this delay to be 0.5 gate delay. After making these assumptions, we can make our analysis based on Figure 32.

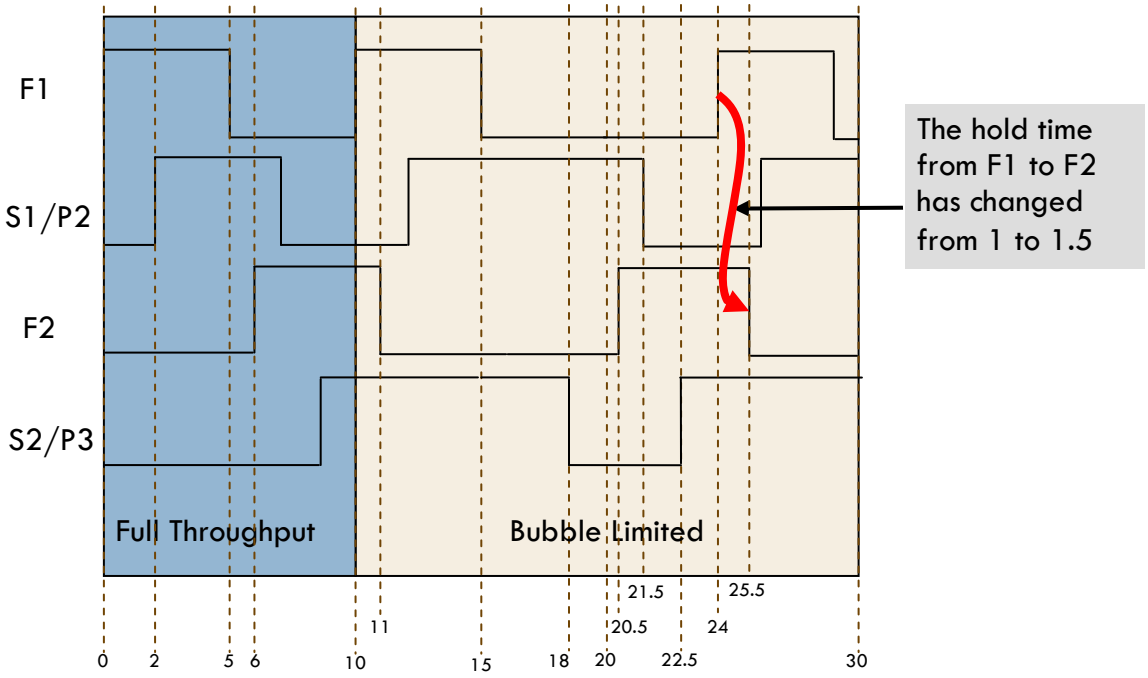

Figure 32: Worst case for hold checks

As illustrated in Figure 32, the hold time check from F1 to F2 has changed from 1 to 1.5 for the bubble limited case. Hence as shown in Figure 32, the third data launched by F1 gets more time to corrupt the second data that is getting latched by F2. This can

result in a hold time violation. Thus we can conclude that the bubble limited case is worse than the full throughput case.

The hold time violations occur because the second data tries to corrupt the first data. This happens because of the overlapping transparent periods of the adjacent fire signals. However in the data limited case, the second fire signal arrives later than the usual 5 gate delays. This results in no overlap between the transparent periods of F1 and F2. Hence the data limited case can never be the worst case for hold time violations.

Chapter 6

Example

In this chapter, we discuss an example on which the timing verification flow was successfully tested. The FLEET architecture has an on-deck stage which acts a three-way branch that passes instructions ahead. The on-deck stage decides which way to pass the incoming instructions based on the flags set by the predicate circuit. The predicate circuit has to make its decisions in time for the efficient working of the on-deck stage. This desired behavior imposes heavy timing constraints on the latches and flip-flops used in the predicate circuit. Hence we chose to verify the constraints on the predicate circuit as a part of this thesis.

6.1 The Predicate circuit

The predicate circuit uses the eighteen instruction bits received by the on-deck stage and an external flag named flag C to set the two flags, named flag A and flag B. This predicate circuit is illustrated in Figure 33. As shown in the figure, a six-bit instruction field (in[1:6]) for flag A and a separate six-bit instruction field (in[7:12]) for flag B define the function that computes the next values for the two flags. The remaining six bits of the instruction (in[13:18]) decide whether the new values of the two flags are going to be latched or the flags retain their old values. This is represented in the clock gating circuit that generates the signal Clk_DF which is used to latch the new values for the two flags.

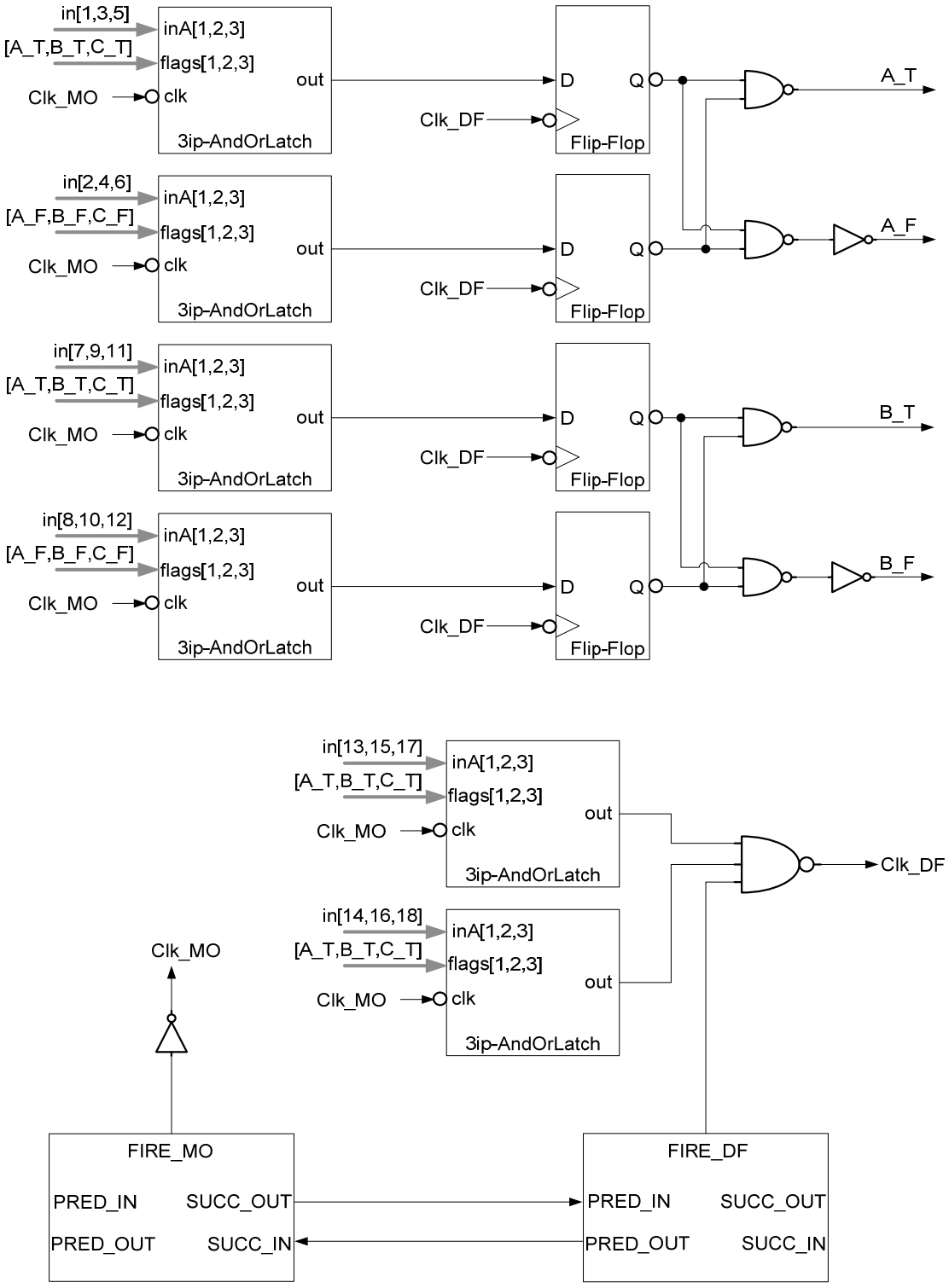

Figure 33: The Predicate circuit

6.1.1 Three input AndOrLatch

One of the important components in the predicate circuit is the three-input AndOrLatch. The internal details of this component are shown in Figure 34. The three instructions bits are allowed to pass through the latches during the transparent phase of the enable signal. The output of these latches is combined with the three flags by the three NAND gates. The output of the NAND gates is then combined with a three input NAND gate which causes the entire function to be of the And-Or form and hence the name AndOrLatch.

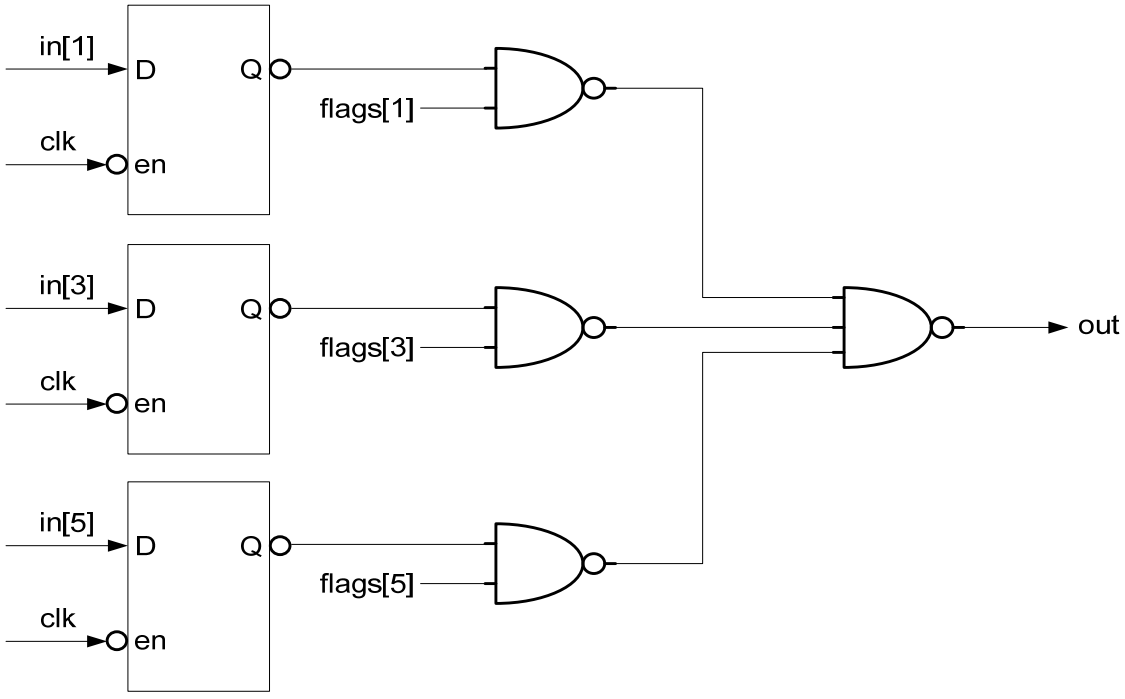

Figure 34: 3ip-AndOrLatch

6.2 Creating timing libraries and the Verilog netlist

The fast and slow timing libraries were created using Sun's internal CAD tool Electric. The fast timing library (fast.lib) is illustrated in Appendix A. Due to the presence of Sun's proprietary information in these libraries; we have chosen to use fixed values for the GasP Plain cell used in the fast.lib. Notice that we have put the timing information pertaining to the falling edge of the FIRE pin in the timing section of the pseudo pin FIRE_PS. The Verilog netlist shown in Appendix B takes this into account. A pseudo load has also been added in the Verilog netlist so that the pin FIRE_PS has the same load as the FIRE pin.

6.3 PrimeTime scripts

The two PrimeTime scripts used for the two-part timing verification flow are shown in Appendix C and Appendix D. The PrimeTime script of Appendix C is used to verify the RT constraints on the GasP control cells and generate the necessary timing information required for verify the RT constraints on the data-path. This information is then used by the PrimeTime script of Appendix D to verify RT constraints on the data-path. The script shown in Appendix C takes on-chip variation into account by using the notion of max and min libraries as shown in Figure 35.

```
set_operating_conditions -analysis_type on_chip_variation -min
fast -max slow -min_library fleet_fast -max_library fleet_slow
set_min_library fleet_slow -min_version fleet_fast
```

Figure 35 : On-chip variation in PrimeTime

A small subset of the script in Appendix C which is used to verify the predecessor loop constraint on the successor state wire is shown in Figure 36. Here the timing path from SUCC_IN to FIRE is disabled in order to break the loop shown in Figure 18. In the next step, we create a clock on the FIRE pin which is the point of divergence for this constraint. Thereafter we issue a data-to-data check between the constrained pin PRED_IN and the related pin FIRE_PS. The results of this data-to-data check are then reported in a text file.

```
#predecessor loop constraint on the successor state-wire
set_disable_timing -from SUCC_IN -to FIRE {MO DF}
create_clock -period 0.4 MO/FIRE
set_data_check -clock MO/FIRE -rise_to DF/PRED_IN -fall_from MO/FIRE_PS -
setup -0.04
report_timing -from MO/FIRE -max_paths 1 > fwd_path_report1.txt
```

Figure 36: Verification of RT constraints on the GasP control cells

After the verification of the RT constraints on the GasP control cells, the script in Appendix C measures the phase difference between clocks for the setup and hold checks. This is shown in Figure 37. As seen previously, we first disable the timing path SUCC_IN to FIRE to break the loop and then we report the phase difference in a text file.

```
#obtaining phase difference between fire signals for setup checks
set_disable_timing -from SUCC_IN -to FIRE {MO DF}
create_clock -period 0.4 MO/FIRE
set_max_delay 1 -from MO/FIRE -to DF/FIRE
report_timing -from MO/FIRE -max_path 1 > setup_phase_report.txt
```

Figure 37: Measuring phase difference between fire signals for setup checks

The phase information thus received is then used by the script in Appendix D to verify the setup and hold checks on the data-path. As shown in Figure 38, we first define the clocks using the phase information. We then define the input conditions and generate the various timing reports.

```
#defining the clocks for setup checks
create_clock MO/FIRE -period 0.400 -waveform { 0.2 0.4 }
create_clock DF/FIRE -period 0.400 -waveform { 0.43 0.63 }
#setting the input conditions
set_input_delay -0.1 -clock MO/FIRE {in*}
#generating timing reports for setup checks
report_timing -group **clock_gating_default** -max_paths 3
-nworst 3 -delay_type max > setup_report2.txt
```

Figure 38: Verification of RT constraints on the data-path

6.4 Interpreting Timing Reports

The timing reports generated by PrimeTime for all the constraints are listed in Appendix E through Appendix K. These reports enable margin analysis and allow the designer to identify the paths where the timing constraints were violated. A subset of the timing report presented in Appendix H is shown in Figure 39. Note that all the delay values are in nano-seconds. PrimeTime first calculates the delay for the two timing paths, namely the constrained path and the relative path. Then it verifies whether the delay on the constrained path is less than the delay on the relative path.

As seen from Figure 39, the data on the constrained path at the inA pin of the three input NAND gate is required to arrive before the latching signal on the relative path arrives on the inC pin. As indicated by the last line of the timing report, the data on the inA pin arrives 40ps before the data on the pinC pin and hence the timing constraint is

met. Similarly the other timing reports can also be interpreted. Note that even though the short circuit constraints produced a violation of 10ps, the design was passed as the designer had chosen to set a margin of 50ps for the short circuit violation.

```
Startpoint: andOr3wL_5/latch10_1 (negative level-sensitive latch clocked
            by MO/FIRE')
Endpoint: nand3_1 (rising clock gating-check end-point clocked by DF/FIRE)
Path Group: **clock_gating_default**
Path Type: max

Point                                                 Incr       Path
-----------------------------------------------------------------------
clock MO/FIRE' (fall edge)                            0.20       0.20
clock network delay (propagated)                      0.05       0.25
andOr3wL_5/latch10_1/clk (latch10)                    0.00       0.25 f
andOr3wL_5/latch10_1/Q (latch10)                      0.03       0.28 r
andOr3wL_5/andOr3_1/inA[1] (andOr3)                   0.00       0.28 r
andOr3wL_5/andOr3_1/nand10sy_1/out (nand10sym)        0.03       0.32 f
andOr3wL_5/andOr3_1/nand3in6_1/out (nand3in6_6)       0.07       0.39 r
andOr3wL_5/andOr3_1/out (andOr3)                      0.00       0.39 r
andOr3wL_5/out (andOr3wLat)                           0.00       0.39 r
nand3_1/inA (nand3in6_6)                              0.00       0.39 r
data arrival time                                                0.39

clock DF/FIRE (rise edge)                             0.43       0.43
clock network delay (propagated)                      0.00       0.43
nand3_1/inC (nand3in6_6)                                         0.43 r
clock gating setup time                               0.00       0.43
data required time                                               0.43
-----------------------------------------------------------------------
data required time                                               0.43
data arrival time                                               -0.39
-----------------------------------------------------------------------
slack (MET)                                                      0.04
```

Figure 39 : Timing Report

Chapter 7

Conclusions and Future Work

A timing verification flow for single track asynchronous circuits is a pre-requisite for incorporating these circuits in standard ASIC designs. An efficient timing analysis flow can then be used to enable synthesis, timing driven place and route and also ECO flows of substantially larger designs. This thesis presents the issues and solutions encountered in establishing such a verification flow for a single track circuit family like GasP. First, we identified the timing constraints which are necessary to be satisfied to ensure correct operation of the GasP control cells and the associated data-path logic. Second, we characterized the timing information pertaining to the GasP control cells in the industry standard Liberty format. Third, we used a static timing analysis tool, Synopsys PrimeTime, to verify the identified timing constraints. Finally, we identified the worst cases of operation which can be used to simulate the timing violations.

The work presented in this thesis is promising as it enables ASIC designers to use the GasP control cells in the standard ASIC flow. The developed flow is a pre-cursor for timing sign-off of Sun's FLEET architecture which uses the GasP control cells for local handshaking. The availability of GasP timing libraries has also made it possible to perform back annotated Verilog simulations which consume substantially less time for larger designs as compared to Spice simulations. Another extension to this flow is to enable timing driven place and route using commercial CAD tools like Cadence Encounter. The liberty characterization flow can also be enhanced to take power consumption into account and thereby enable power analysis. Such a liberty file can then

be used for power driven place and route which is gaining importance with the reducing technologies.

As a part of future work, we believe that the flow presented in this thesis can be easily extended to other asynchronous circuit families like USC's Static Single Track Full Buffer (SSTFB) and Intel's Asynchronous Bus Connector (ABC). Both these circuit families have different relative timing constraints that guide their operation and identifying them would be the only change required to the current flow. The current flow can thus be made generic by establishing connection with a tool like ANALYZE [1] that automatically generates the relative timing constraints for a given circuit template.

Bibliography

[1] Analyze website: http://www.ece.utah.edu/~kstevens/analyze-page.html

[2] Coates, W., Lexau, J., Jones, I., Fairbanks, S., Sutherland, I.: FLEETzero: An Asynchronous Switching Experiment. Proceedings of the 7th IEEE International Symposium on Asynchronous Circuits and Systems, pages 173-182, 2001.

[3] Chandromouli, V., Sakallah, K.: Modeling the effects of temporal proximity of input transitions on gate propagation delay and transition time. Proceedings of the 33rd ACM/IEEE Design Automation Conference, pages 617-622, June 1996.

[4] Dartu, F., Menzes, N., Qian, J., Pillage, L.: A gate delay model for high speed CMOS circuits. Proceedings of the 31st ACM/IEEE Design Automation Conference, pages 576-580, June 1994.

[5] Ebergen, J.C., Fairbanks, S., Sutherland, I.E.: Predicting Performance of Micropipelines Using Charlie Diagrams. Proceedings of the 4th IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 238–246, April 1998.

[6] Electric website: http://www.staticfreesoft.com

[7] Ferretti, M., Beerel, P.A.: High Performance Asynchronous Design Using Single-Track Full-Buffer Standard Cells. IEEE Journal of Solid-State Circuits, Vol. 41, No. 6, pages 1444–1454, June 2006.

[8] Fleet website: http://research.cs.berkeley.edu/class/fleet/

[9] Golani, P.: Area Efficient High Performance Asynchronous Design, Qualifying Proposal, July 2008.

[10] Handshake Solutions website: http://www.handshakesolutions.com/partners/ARM996HS_press.html

[11] Joshi P., Beerel P., Roncken M., Sutherland I.: “Timing Verification of GasP Asynchronous Circuits: Predicted Delay Variations Observed by Experiment”, 2008 Festschrift Series of Springer LNCS for the Festschrift of Willem-Paul de Roever.

[12] Molnar, C., Jones, I., Coates, W., Lexau, J.: A FIFO Ring Performance Experiment. Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 7-10, April 1997.

[13] Prakash, M.: Library Characterization and Static Timing Analysis of Asynchronous Circuits. Master's Thesis, University of Southern California, December 2007.

[14] PrimeTime User Guide: Advanced timing analysis, Version Y-2006.06, Synopsys Inc., June 2006.

[15] PrimeTime User Guide: Fundamentals, Version Y-2006.06, Synopsys Inc., June 2006.

[16] Stevens, K., Ginosar, R., Rotem, S.: Relative Timing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 11, No. 1, pages 129–140, February 2003.

[17] Sutherland, I., Fairbanks, S.: GasP: A Minimal FIFO Control. Proceedings of the 7th IEEE International Symposium on Asynchronous Circuits and Systems, pages 46-53, 2001.

[18] van Berkel, K., Bink, A.: Single-Track Handshake Signaling with Application to Micropipelines and Handshake Circuits. 2nd IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 122–133, March 1996.

[19] Winstanley, A.J., Garivier, A., Greenstreet, M.R.: An Event Spacing Experiment. Proceedings of the 8th IEEE International Symposium on Asynchronous Circuits and Systems, pages 47 – 56, April 2002.

Appendix A

Fast timing library

```
library (fleet_fast) {
  technology (cmos) ;
  delay_model : table_lookup ;
  library_features(report_delay_calculation);
  capacitive_load_unit (1.0, pf) ;
  time_unit : 1ns ;
  voltage_unit : 1V ;
  current_unit : 1A ;
  input_threshold_pct_rise : 50.0 ;
  input_threshold_pct_fall : 50.0 ;
  output_threshold_pct_rise : 50.0 ;
  output_threshold_pct_fall : 50.0 ;
  slew_lower_threshold_pct_rise : 20.0 ;
  slew_upper_threshold_pct_rise : 80.0 ;
  slew_lower_threshold_pct_fall : 20.0 ;
  slew_upper_threshold_pct_fall : 80.0 ;
  pulling_resistance_unit : 1ohm ;
  default_fanout_load : 1.0 ;
  default_inout_pin_cap : 1.0 ;
  default_input_pin_cap : 1.0 ;
  default_output_pin_cap : 0.0 ;
  operating_conditions (fast) {
    voltage : 0.85 ;
    temperature : 110.0 ;
    process : 1.0 ;
  }
  default_operating_conditions : fast;
lu_table_template(delay_template_cap_P_plain_6x1) {
            variable_1 : total_output_net_capacitance;
            index_1 ("0.010,0.015,0.020,0.025,0.030,0.035");
}

lu_table_template(delay_template_P_rise_plain_6x1) {
            variable_1 : input_net_transition;
            index_1 ("0.1,0.2,0.3,0.4,0.5,0.6");
}

lu_table_template(delay_template_P_fall_plain_6x1) {
            variable_1 : input_net_transition;
            index_1 ("0.1,0.2,0.3,0.4,0.5,0.6");
}

lu_table_template(delay_template_S_rise_plain_6x1) {
            variable_1 : input_net_transition;
            index_1 ("0.1,0.2,0.3,0.4,0.5,0.6");
}

lu_table_template(delay_template_S_fall_plain_6x1) {
```

```
        variable_1 : input_net_transition;
        index_1 ("0.1,0.2,0.3,0.4,0.5,0.6");
}

lu_table_template(delay_template_cap_S_plain_6x1) {
        variable_1 : total_output_net_capacitance;
        index_1 ("0.010,0.015,0.020,0.025,0.030,0.035");
}
cell(GASP_PLAIN) {

        cell_leakage_power : 0.0;

        pin (PRED_IN) {
                direction : input;
                capacitance : 0.010;
                }

        pin (PRED_OUT) {
                direction : output;
                capacitance : 0.010;

                timing() {

                        related_pin : "FIRE";
                        timing_type : combinational_fall;
                        timing_sense : negative_unate;
                        cell_fall(delay_template_cap_P_plain_6x1) {
                                values("0.04,0.04,0.04,0.04,0.04,0.04");
                                }
                        fall_transition(delay_template_cap_P_plain_6x1) {
                                values("0.1,0.2,0.3,0.4,0.5,0.6");
                                }
                        }
                }

        pin (FIRE) {
                direction : output;
                capacitance : 0.010;

                timing() {

                        related_pin : "PRED_IN";
                        timing_type : combinational_rise;
                        timing_sense : positive_unate;

                        cell_rise(delay_template_P_rise_plain_6x1) {
                                values("0.15,0.15,0.15,0.15,0.15,0.15");
                                }
                        rise_transition(delay_template_P_rise_plain_6x1) {
                                values("0.1,0.2,0.3,0.4,0.5,0.6");
                                }
                        }
```

```
        timing() {

                related_pin : "SUCC_IN";
                timing_type : combinational_rise;
                timing_sense : negative_unate;
                cell_rise(delay_template_S_rise_plain_6x1) {
                        values("0.11,0.11,0.11,0.11,0.11,0.11");
                        }
                rise_transition(delay_template_S_rise_plain_6x1) {
                        values("0.1,0.2,0.3,0.4,0.5,0.6");
                        }
                }
        }

pin (FIRE_PS) {
        direction : output;
        capacitance : 0.010;


        timing() {

                related_pin : "PRED_OUT";
                timing_type : combinational_fall;
                timing_sense : positive_unate;

                cell_fall(delay_template_P_fall_plain_6x1) {
                        values("0.16,0.16,0.16,0.16,0.16,0.16");
                        }
                fall_transition(delay_template_P_fall_plain_6x1) {
                        values("0.1,0.2,0.3,0.4,0.5,0.6");
                        }

                }

        timing() {

                related_pin : "SUCC_OUT";
                timing_type : combinational_fall;
                timing_sense : negative_unate;

                cell_fall(delay_template_S_fall_plain_6x1) {
                        values("0.12,0.12,0.12,0.12,0.12,0.12");
                        }
                fall_transition(delay_template_S_fall_plain_6x1) {
                        values("0.1,0.2,0.3,0.4,0.5,0.6");
                        }
                }
        }

pin (SUCC_IN) {
        direction : input;
        capacitance : 0.010;
        }
```

```
        pin (SUCC_OUT) {
                direction : output;
                capacitance : 0.010;

                timing() {

                        related_pin : "FIRE";
                        timing_type : combinational_rise;
                        timing_sense : positive_unate;
                        cell_rise(delay_template_cap_S_plain_6x1) {
                                values("0.08,0.08,0.08,0.08,0.08,0.08");
                                }
                        rise_transition(delay_template_cap_S_plain_6x1) {
                                values("0.1,0.2,0.3,0.4,0.5,0.6");
                                }
                        }

                }
        }
}
```

Appendix B

Verilog netlist for the predicate circuit

```
module inv20B(in, out);
  input in;
  output out;
endmodule   /* inv20B */

module inv30(inA, out);
  input inA;
  output out;
endmodule   /* inv30 */

module nand10sym(inA, inB, out);
  input inA;
  input inB;
  output out;
endmodule   /* nand10sym */

module nand3in6_6(inA, inB, inC, out);
  input inA;
  input inB;
  input inC;
  output out;
endmodule   /* nand3in6_6 */

module nor10sym(inA, inB, out);
  input inA;
  input inB;
  output out;
endmodule   /* nor10sym */

module latch10(D, clk, mc, Q);
  input D;
  input clk;
  input mc;
  output Q;
endmodule   /* latch10 */

module flip_flop10(D, clk, mc, Q);
  input D;
  input clk;
  input mc;
  output Q;
endmodule   /* flip_flop10 */

module andOr3(inA, inB, out);
  input [1:3] inA;
  input [1:3] inB;
  output out;
  wire net_1, net_2, net_3;
```

```
  nand10sym nand10sy_1(.inA(inA[1]), .inB(inB[1]), .out(net_1));
  nand10sym nand10sy_2(.inA(inA[2]), .inB(inB[2]), .out(net_2));
  nand10sym nand10sy_3(.inA(inA[3]), .inB(inB[3]), .out(net_3));
  nand3in6_6 nand3in6_1(.inA(net_1), .inB(net_2), .inC(net_3), .out(out));

endmodule   /* andOr3 */

module andOr3wLat(clk, flag_A, flag_B, flag_C, \in[1] , \in[2] ,
     \in[3] , mc, out);
  input clk;
  input flag_A;
  input flag_B;
  input flag_C;
  input \in[1] , \in[2] , \in[3] ;
  input mc;
  output out;
  wire [1:3] Q;

  latch10 latch10_1(.D( \in[1] ), .clk(clk), .mc(mc), .Q(Q[1]));
  latch10 latch10_2(.D( \in[2] ), .clk(clk), .mc(mc), .Q(Q[2]));
  latch10 latch10_3(.D( \in[3] ), .clk(clk), .mc(mc), .Q(Q[3]));
  andOr3 andOr3_1(.inA(Q[1:3]), .inB({flag_A, flag_B, flag_C}),
     .out(out))
endmodule   /* andOr3wLat */

module GASP_PLAIN (PRED_IN,PRED_OUT,FIRE,FIRE_PS,SUCC_IN,SUCC_OUT);
input PRED_IN;
input SUCC_IN;
output FIRE;
output FIRE_PS;
output PRED_OUT;
output SUCC_OUT;
endmodule   /* GASP_PLAIN */


module predicate(Pin, Sin, flag_C_F, flag_C_T, in, mc,  Pout, Sout, predBar);
  input Pin;
  input Sin;
  input flag_C_F;
  input flag_C_T;
  input [1:18] in;
  input mc;
  output Pout;
  output Sout;
  output predBar;

  wire FIRE_MO,FIRE_MO_PS,S_MO_IN,S_MO_OUT,FIRE_DF,FIRE_DF_PS;
  wire Clk_MO, ok_A,ok_B,Clk_DF;
  wire flag_A_F, flag_A_T, flag_B_F, flag_B_T;
  wire net_1, net_2, net_3, net_4, net_5, net_6, net_7, net_8, net_9, net_10;
  wire net_11,net_12,net_13,net_14;
```

```
  GASP_PLAIN MO
(.PRED_IN(Pin),.PRED_OUT(Pout),.FIRE(FIRE_MO),.FIRE_PS(FIRE_MO_PS),.SUCC_IN(S_MO_IN),
.SUCC_OUT(S_MO_OUT));
  GASP_PLAIN DF
(.PRED_IN(S_MO_OUT),.PRED_OUT(S_MO_IN),.FIRE(FIRE_DF),.FIRE_PS(FIRE_DF_PS),.SUCC_I
N(Sin),.SUCC_OUT(Sout));

  inv20B inv20B_1(.in(FIRE_MO), .out(Clk_MO));
  inv20B inv20B_2(.in(FIRE_MO_PS), .out(net_11));

  nand3in6_6 nand3_1(.inA(ok_A), .inB(ok_B), .inC(FIRE_DF), .out(Clk_DF));
  nand3in6_6 nand3_2(.inA(net_12), .inB(net_13), .inC(FIRE_DF_PS), .out(net_14));

  andOr3wLat andOr3wL_1(.clk(Clk_MO), .flag_A(flag_A_T),
     .flag_B(flag_B_T), .flag_C(flag_C_T), .\in[1] (in[1]), .\in[2]
     (in[3]), .\in[3] (in[5]), .mc(mc), .out(net_1));
  andOr3wLat andOr3wL_2(.clk(Clk_MO), .flag_A(flag_A_F),
     .flag_B(flag_B_F), .flag_C(flag_C_F), .\in[1] (in[2]), .\in[2]
     (in[4]), .\in[3] (in[6]), .mc(mc), .out(net_2));

  andOr3wLat andOr3wL_3(.clk(Clk_MO), .flag_A(flag_A_T),
     .flag_B(flag_B_T), .flag_C(flag_C_T), .\in[1] (in[7]), .\in[2]
     (in[9]), .\in[3] (in[11]), .mc(mc), .out(net_3));
  andOr3wLat andOr3wL_4(.clk(Clk_MO), .flag_A(flag_A_F),
     .flag_B(flag_B_F), .flag_C(flag_C_F), .\in[1] (in[8]), .\in[2]
     (in[10]), .\in[3] (in[12]), .mc(mc), .out(net_4));

  andOr3wLat andOr3wL_5(.clk(Clk_MO), .flag_A(flag_A_T),
     .flag_B(flag_B_T), .flag_C(flag_C_T), .\in[1] (in[13]), .\in[2]
     (in[15]), .\in[3] (in[17]), .mc(mc), .out(ok_A));
  andOr3wLat andOr3wL_6(.clk(Clk_MO), .flag_A(flag_A_F),
     .flag_B(flag_B_F), .flag_C(flag_C_F), .\in[1] (in[14]), .\in[2]
     (in[16]), .\in[3] (in[18]), .mc(mc), .out(ok_B));

  flip_flop10 flop_1(.D(net_1), .clk(Clk_DF), .mc(mc), .Q(net_5));
  flip_flop10 flop_2(.D(net_2), .clk(Clk_DF), .mc(mc), .Q(net_6));
  flip_flop10 flop_3(.D(net_3), .clk(Clk_DF), .mc(mc), .Q(net_7));
  flip_flop10 flop_4(.D(net_4), .clk(Clk_DF), .mc(mc), .Q(net_8));

  nand10sym nand10sy_1(.inA(net_5), .inB(net_6), .out(flag_A_T));
  nand10sym nand10sy_2(.inA(net_5), .inB(net_6), .out(net_9));
  inv30 inv30_1(.inA(net_9), .out(flag_A_F));

  nand10sym nand10sy_3(.inA(net_7), .inB(net_8), .out(flag_B_T));
  nand10sym nand10sy_4(.inA(net_7), .inB(net_8), .out(net_10));
  inv30 inv30_2(.inA(net_10), .out(flag_B_F));

  nor10sym nor10sym_1(.inA(ok_A), .inB(ok_B), .out(predBar));

endmodule   /* predicate */
```

Appendix C

PrimeTime script for verifying the RT constraints on the control logic and obtaining phase differences for verifying the RT constraints on the data-path

```
#reading the design
set netlist predicate.v
set top predicate
read_lib fleet_slow.lib
read_lib fleet_fast.lib
read_verilog $netlist
set link_path "fleet_slow $netlist"
link_design -keep_sub_designs $top
set_operating_conditions -analysis_type on_chip_variation -min fast -max slow -min_library fleet_fast -
max_library fleet_slow
set_min_library fleet_slow -min_version fleet_fast

#predecessor loop constraint on the successor state-wire
set_disable_timing -from SUCC_IN -to FIRE {MO DF}
create_clock -period 0.4 MO/FIRE
set_data_check -clock MO/FIRE -rise_to DF/PRED_IN -fall_from MO/FIRE_PS -setup -0.04
report_timing -from MO/FIRE -max_paths 1 > fwd_path_report1.txt
remove_data_check -clock MO/FIRE -rise_to DF/PRED_IN -fall_from MO/FIRE_PS
remove_clock -all

#short circuit constraint on the successor state-wire
create_clock -period 0.4 MO/SUCC_OUT
set_data_check -clock MO/SUCC_OUT -fall_to MO/FIRE_PS -rise_from DF/FIRE  -setup 0.04
report_timing -from MO/SUCC_OUT -max_paths 1 > fwd_path_report2.txt
remove_data_check -clock MO/SUCC_OUT -fall_to MO/FIRE_PS -rise_from DF/FIRE
remove_clock -all
remove_disable_timing -from SUCC_IN -to FIRE {MO DF}

#successor loop constraint on the predecessor state-wire
set_disable_timing -from PRED_IN -to FIRE {MO DF}
create_clock -period 0.4 DF/FIRE
set_data_check -clock DF/FIRE -fall_to MO/SUCC_IN -fall_from DF/FIRE_PS -setup 0.0
report_timing -from DF/FIRE -max_paths 1 > rev_path_report1.txt
remove_data_check -clock DF/FIRE -fall_to MO/SUCC_IN -fall_from DF/FIRE_PS
remove_clock -all

#short circuit constraint on the predecessor state-wire
create_clock -period 0.4 DF/PRED_OUT
set_data_check -clock DF/PRED_OUT -fall_to DF/FIRE_PS -rise_from MO/FIRE  -setup -0.04
report_timing -from DF/PRED_OUT -max_paths 1 > rev_path_report2.txt
remove_data_check -clock DF/PRED_OUT -fall_to DF/FIRE_PS -rise_from MO/FIRE
remove_clock -all
remove_disable_timing -from PRED_IN -to FIRE {MO DF}

#using the fast.lib for obtaining phase information
```

```
remove_lib fleet_slow
set link_path "fleet_fast $netlist"
link_design -keep_sub_designs $top

#obtaining phase difference between fire signals for setup checks
set_disable_timing -from SUCC_IN -to FIRE {MO DF}
create_clock -period 0.4 MO/FIRE
set_max_delay 1 -from MO/FIRE -to DF/FIRE
report_timing -from MO/FIRE -max_path 1 > setup_phase_report.txt
remove_disable_timing -from SUCC_IN -to FIRE {MO DF}
remove_clock -all

#obtaining phase difference between fire signals for hold checks
set_disable_timing -from PRED_IN -to FIRE {MO DF}
create_clock -period 0.4 DF/FIRE
set_max_delay 1 -from DF/FIRE -to MO/FIRE
report_timing -from DF/FIRE -max_path 1 > hold_phase_report.txt
```

Appendix D

PrimeTime script to verify the RT constraints on the data-path

```
#reading the design
set netlist predicate.v
set top predicate
read_lib fleet_fast.lib
read_verilog $netlist
set link_path "fleet_fast $netlist"
echo $link_path
link_design -keep_sub_designs $top

#defining the clocks for setup checks
create_clock MO/FIRE -period 0.400 -waveform { 0.2 0.4 }
create_clock DF/FIRE -period 0.400 -waveform { 0.43 0.63 }
set_propagated_clock {MO/FIRE DF/FIRE}
set_clock_transition 0.01 -rise {MO/FIRE DF/FIRE}
set_clock_transition 0.01 -fall {MO/FIRE DF/FIRE}

#setting the input conditions
set_input_delay -0.1 -clock MO/FIRE {in*}
set_input_transition 0.01 in*
set_false_path -from flag_C_F
set_false_path -from flag_C_T
set timing_slew_propagation_mode worst_arrival

#generating timing reports for setup checks
report_timing -from flop*/clk -to flop*/D -max_paths 3 -nworst 3 -delay_type max > setup_report1.txt
report_timing -group **clock_gating_default** -max_paths 3 -nworst 3 -delay_type max >
setup_report2.txt
report_timing -from andOr*/latch*/clk -to flop*/D -max_paths 3 -nworst 3 -delay_type max >
setup_report3.txt

#defining the clocks for hold checks
create_clock MO/FIRE -period 0.400 -waveform { 0.2 0.4 }
create_clock DF/FIRE -period 0.400 -waveform { 0.45 0.65 }
set_propagated_clock {MO/FIRE DF/FIRE}
set_clock_transition 0.01 -rise {MO/FIRE DF/FIRE}
set_clock_transition 0.01 -fall {MO/FIRE DF/FIRE}

#generating timing reports for hold checks
report_timing -from flop*/clk -to flop*/D -max_paths 3 -nworst 3 -delay_type min > hold_report1.txt
report_timing -group **clock_gating_default** -delay_type min -max_paths 3 -nworst 3 > hold_report2.txt
report_timing -from andOr*/latch*/clk -to flop*/D -max_paths 3 -nworst 3 -delay_type min >
hold_report3.txt
```

Appendix E

PrimeTime report for the predecessor loop constraint on the successor state-wire

```
****************************************
Report : timing
         -path_type full
         -delay_type max
         -max_paths 1
Design : predicate
Version: Z-2006.12-SP1
Date   : Tue Oct 14 02:49:39 2008
****************************************
```

Startpoint: MO/FIRE (clock source 'MO/FIRE')
Endpoint: DF (falling edge-triggered data to data check clocked by MO/FIRE)
Path Group: MO/FIRE
Path Type: max

Point	Incr	Path
clock MO/FIRE (rise edge)	0.00	0.00
clock source latency	0.00	0.00
MO/FIRE (GASP_PLAIN)	0.00	0.00 r
MO/SUCC_OUT (GASP_PLAIN)	0.08	0.08 r
DF/PRED_IN (GASP_PLAIN)	0.00	0.08 r
data arrival time		0.08
clock MO/FIRE (rise edge)	0.00	0.00
clock source latency	0.00	0.00
MO/FIRE (GASP_PLAIN)	0.00	0.00 r
MO/PRED_OUT (GASP_PLAIN)	0.04	0.04 f
MO/FIRE_PS (GASP_PLAIN)	0.16	0.20 f
data check setup time	0.04	0.24
data required time		0.24
data required time		0.24
data arrival time		-0.08
slack (MET)		0.16

Appendix F

PrimeTime report for the short circuit constraint on the successor state-wire

```
****************************************
Report : timing
         -path_type full
         -delay_type max
         -max_paths 1
Design : predicate
Version: Z-2006.12-SP1
Date   : Tue Oct 14 02:49:39 2008
****************************************


  Startpoint: MO/SUCC_OUT
         (clock source 'MO/SUCC_OUT')
  Endpoint: MO (rising edge-triggered data to data check clocked by MO/SUCC_OUT)
  Path Group: MO/SUCC_OUT
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock MO/SUCC_OUT (rise edge)            0.00       0.00
  clock source latency                     0.00       0.00
  MO/SUCC_OUT (GASP_PLAIN)                 0.00       0.00 r
  MO/FIRE_PS (GASP_PLAIN)                  0.12       0.12 f
  data arrival time                                   0.12

  clock MO/SUCC_OUT (rise edge)            0.00       0.00
  clock source latency                     0.00       0.00
  MO/SUCC_OUT (GASP_PLAIN)                 0.00       0.00 r
  DF/FIRE (GASP_PLAIN)                     0.15       0.15 r
  data check setup time                    -0.04      0.11
  data required time                                  0.11
  -----------------------------------------------------------
  data required time                                  0.11
  data arrival time                                   -0.12
  -----------------------------------------------------------
  slack (VIOLATED)                                    -0.01
```

Appendix G

PrimeTime report for the successor loop constraint on the predecessor state-wire

```
****************************************
Report : timing
         -path_type full
         -delay_type max
         -max_paths 1
Design : predicate
Version: Z-2006.12-SP1
Date   : Tue Oct 14 02:49:39 2008
****************************************
```

Startpoint: DF/FIRE (clock source 'DF/FIRE')
Endpoint: MO (falling edge-triggered data to data check clocked by DF/FIRE)
Path Group: DF/FIRE
Path Type: max

Point	Incr	Path
clock DF/FIRE (rise edge)	0.00	0.00
clock source latency	0.00	0.00
DF/FIRE (GASP_PLAIN)	0.00	0.00 r
DF/PRED_OUT (GASP_PLAIN)	0.04	0.04 f
MO/SUCC_IN (GASP_PLAIN)	0.00	0.04 f
data arrival time		0.04
clock DF/FIRE (rise edge)	0.00	0.00
clock source latency	0.00	0.00
DF/FIRE (GASP_PLAIN)	0.00	0.00 r
DF/PRED_OUT (GASP_PLAIN)	0.04	0.04 f
DF/FIRE_PS (GASP_PLAIN)	0.16	0.20 f
data check setup time	0.00	0.20
data required time		0.20
data required time		0.20
data arrival time		-0.04
slack (MET)		0.16

Appendix H

PrimeTime report for the short circuit constraint on the predecessor state-wire

```
****************************************
Report : timing
         -path_type full
         -delay_type max
         -max_paths 1
Design : predicate
Version: Z-2006.12-SP1
Date   : Tue Oct 14 02:49:39 2008
****************************************


  Startpoint: DF/PRED_OUT
         (clock source 'DF/PRED_OUT')
  Endpoint: DF (rising edge-triggered data to data check clocked by DF/PRED_OUT)
  Path Group: DF/PRED_OUT
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock DF/PRED_OUT (fall edge)            0.20       0.20
  clock source latency                     0.00       0.20
  DF/PRED_OUT (GASP_PLAIN)                 0.00       0.20 f
  DF/FIRE_PS (GASP_PLAIN)                  0.16       0.36 f
  data arrival time                                   0.36

  clock DF/PRED_OUT (fall edge)            0.20       0.20
  clock source latency                     0.00       0.20
  DF/PRED_OUT (GASP_PLAIN)                 0.00       0.20 f
  MO/FIRE (GASP_PLAIN)                     0.11       0.31 r
  data check setup time                    0.04       0.35
  data required time                                  0.35
  -----------------------------------------------------------
  data required time                                  0.35
  data arrival time                                   -0.36
  -----------------------------------------------------------
  slack (VIOLATED)                                    -0.01
```

Appendix I

PrimeTime report showing the phase relation between FIRE signals

```
****************************************
Report : timing
         -path_type full
         -delay_type max
         -max_paths 1
Design : predicate
Version: Z-2006.12-SP1
Date   : Tue Oct 14 02:49:39 2008
****************************************

# Phase relation for setup checks

  Startpoint: MO/FIRE (clock source 'MO/FIRE')
  Endpoint: DF/FIRE (internal path endpoint)
  Path Group: **default**
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock source latency                     0.00       0.00
  MO/FIRE (GASP_PLAIN)                     0.00       0.00 r
  MO/SUCC_OUT (GASP_PLAIN)                 0.08       0.08 r
  DF/FIRE (GASP_PLAIN)                     0.15       0.23 r
  data arrival time                                   0.23


# Phase relation for hold checks

  Startpoint: DF/FIRE (clock source 'DF/FIRE')
  Endpoint: MO/FIRE (internal path endpoint)
  Path Group: **default**
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock source latency                     0.00       0.00
  DF/FIRE (GASP_PLAIN)                     0.00       0.00 r
  DF/PRED_OUT (GASP_PLAIN)                 0.04       0.04 f
  MO/FIRE (GASP_PLAIN)                     0.11       0.15 r
  data arrival time                                   0.15
```

Appendix J

PrimeTime report for setup checks on the data-path

```
****************************************
Report : timing
         -path_type full
         -delay_type max
         -nworst 3
         -max_paths 3
         -group **clock_gating_default**
Design : predicate
Version: Z-2006.12-SP1
Date   : Tue Oct 14 02:50:30 2008
****************************************
```

Startpoint: andOr3wL_5/latch10_1
(negative level-sensitive latch clocked by MO/FIRE')
Endpoint: nand3_1 (rising clock gating-check end-point clocked by DF/FIRE)
Path Group: **clock_gating_default**
Path Type: max

Point	Incr	Path
clock MO/FIRE' (fall edge)	0.20	0.20
clock network delay (propagated)	0.05	0.25
andOr3wL_5/latch10_1/clk (latch10)	0.00	0.25 f
andOr3wL_5/latch10_1/Q (latch10)	0.03	0.28 r
andOr3wL_5/andOr3_1/inA[1] (andOr3)	0.00	0.28 r
andOr3wL_5/andOr3_1/nand10sy_1/out (nand10sym)	0.03	0.32 f
andOr3wL_5/andOr3_1/nand3in6_1/out (nand3in6_6)	0.07	0.39 r
andOr3wL_5/andOr3_1/out (andOr3)	0.00	0.39 r
andOr3wL_5/out (andOr3wLat)	0.00	0.39 r
nand3_1/inA (nand3in6_6)	0.00	0.39 r
data arrival time		0.39
clock DF/FIRE (rise edge)	0.43	0.43
clock network delay (propagated)	0.00	0.43
nand3_1/inC (nand3in6_6)		0.43 r
clock gating setup time	0.00	0.43
data required time		0.43
data required time		0.43
data arrival time		-0.39
slack (MET)		0.04

Appendix K

PrimeTime report for hold checks on the data-path

```
****************************************
Report : timing
         -path_type full
         -delay_type min
         -nworst 3
         -max_paths 3
         -group **clock_gating_default**
Design : predicate
Version: Z-2006.12-SP1
Date   : Tue Oct 14 02:51:13 2008
****************************************
```

Startpoint: flop_4 (falling edge-triggered flip-flop clocked by DF/FIRE')
Endpoint: nand3_1 (rising clock gating-check end-point clocked by DF/FIRE)
Path Group: **clock_gating_default**
Path Type: min

Point	Incr	Path
clock DF/FIRE' (fall edge)	0.05	0.05
clock network delay (propagated)	0.04	0.09
flop_4/clk (flip_flop10)	0.00	0.09 f
flop_4/Q (flip_flop10)	0.05	0.15 r
nand10sy_3/out (nand10sym)	0.04	0.19 f
andOr3wL_5/flag_B (andOr3wLat)	0.00	0.19 f
andOr3wL_5/andOr3_1/inB[2] (andOr3)	0.00	0.19 f
andOr3wL_5/andOr3_1/nand10sy_2/out (nand10sym)	0.04	0.23 r
andOr3wL_5/andOr3_1/nand3in6_1/out (nand3in6_6)	0.04	0.27 f
andOr3wL_5/andOr3_1/out (andOr3)	0.00	0.27 f
andOr3wL_5/out (andOr3wLat)	0.00	0.27 f
nand3_1/inA (nand3in6_6)	0.00	0.27 f
data arrival time		0.27
clock DF/FIRE (fall edge)	0.25	0.25
clock network delay (propagated)	0.00	0.25
nand3_1/inC (nand3in6_6)		0.25 f
clock gating hold time	0.00	0.25
data required time		0.25
data required time		0.25
data arrival time		-0.27
slack (MET)		0.02