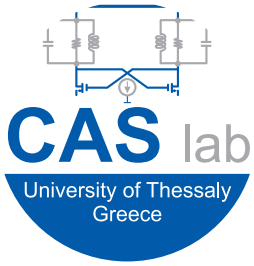

DIPLOMA THESIS

Latch STA Time-Borrowing Implementation with and without Loop Breaking

Student:
Nikolaos Blias
nblias@uth.gr

Supervisor:
Christos Sotiriou
chsotiriou@e-ce.uth.gr
Committee:
Georgios Stamoulis
georges@e-ce.uth.gr
Fotios Plessas
fplessas@e-ce.uth.gr

A thesis submitted in fulfillment of the requirements
for the degree of Integrated Master
in the
Circuits & Systems Laboratory (CASlab)
Department of Electrical and Computer Engineering

September 18, 2021

Abstract

Latch-based designs have many advantages over Flip-Flop-based designs such as timing, power, and area but their use is constrained. First, most RTL implementations nowadays are Flip-Flop-based due to the existence of Latches being level-sensitive, while Flip-Flops are edge-triggered components, allowing for easier circuit Static Timing Analysis (STA) validation. After that, state of the art STA engines are dependent on a number of hypotheses about the essence of the design and the combination of the nature of Latches may lead to timing errors and poor quality results. In more detail, the existence of combinational feedback loops in a Latch-based design may drive the STA engine to disable these feedback loops in an effort to minimize the effect of the violation-timing error on the circuit timing validation. In relation to the above, in some cases where more than one Latches clocked on different phases are transparent simultaneously, may come up with a wrong timing analysis, since a momentery combinational feedback it may be produced functionally. Another case that can cause a wrong timing report is the procedure of a setup checking on a Latch. In this work, we introduce a Latch-STA methodology that supports Synchronous, Cyclic, and Acyclic circuits and has been implemented and integrated on an Electronic Design Automation (EDA) tool called ASP. We need to mention that the combination of this work with the existing ASP ASTA tool could provide timing analysis on Bundled-Data Latch-Based designs. Eventually, we present the experimental results and emphasize the fact that this work handles effectively the above assumptions and design cases.

Περίληψη

Τα ψηφιακά κυκλώματα που στηρίζουν τη σχεδίαση τους σε Μανδαλωτές έχουν αρκετά πλεονεκτήματα σε σχέση με τα ψηφιακά κυκλώματα που στηρίζουν την υλοποίηση τους σε Καταχωρητές, σε τομείς όπως είναι ο χρονισμός, η ενέργεια και το εμβαδόν, η χρήση τους όμως είναι περιορισμένη. Αρχικά στη σημερινή εποχή οι περισσότερες κυκλωματικές υλοποιήσεις στηρίζονται σε Καταχωρητές, λόγω της φύσης τους που τα καθιστά πυροδοτούμενα στην ακμή σε σχέση με τους Μανδαλωτές που πυροδοτούνται με βάση το ενεργό χρονικό επίπεδο. Έτσι κυκλώματα με Καταχωρητές είναι πιο εύκολα διαχειρίσιμα στο γεγονός της επαλήθευσης της χρονικής λειτουργείας τους μέσω της Στατικής Χρονικής Ανάλυσης. Στη συνέχεια, τα βιομηχανικά εργαλεία Στατικής Χρονικής Ανάλυσης είναι εξαρτώμενα από διάφορες υποθέσεις σχετικά με την φύση του κυκλώματος, το οποίο σε συνδιασμό με την φύση των Μανδαλωτών, μπορεί να οδηγήσει σε λάθη χρονισμού καθώς επίσης και σε χαμηλής ποιότητας αποτελέσματα. Πιο συγκεκριμένα η ύπαρξη κύκλων ανατροφοδότησης σε κυκλώματα σχεδιασμένα με Μανδαλωτές, μπορεί να οδηγήσει το εργαλείο στο γεγονός να ¨κόψει¨ αυτούς τους κύκλους στην προσπάθεια να μειώσει τα λάθη χρονισμού κατά την διάρκεια της χρονικής επαλήθευσης του κυκλώματος. Σε συσχέτιση με το παραπάνω, σε περιπτώσεις όπου Μανδαλωτές που είναι χρονισμένοι σε διαφορετικές φάσεις ρολογιών, προάγουν δεδομένα ταυτόχρονα, μπορεί να παραχθεί μια εσφαλμένη χρονική ανάλυση. Μια ακόμη περίπτωση που μπορεί να εισάγει εσφαλμένη χρονική ανάλυση είναι η διεργασία κατά την οποία πραγματοποιείται χρονικός έλεγχος στην τιμή setup του Μανδαλωτή. Σε αυτή την πτυχιακή εργασιά παρουσιάζουμε μία μεθοδολογία Στατικής Χρονικής Ανάλυσης Μανδαλωτών, η οποία υποστηρίζει, Σύγχρονα, Κυκλικά κ Άκυκλα κυκλώματα, καθώς επίσης αξίζει να αναφερθεί πως αυτή η δουλειά έχει υλοποιηθεί και ενσωματωθεί σε ένα εργαλείο αυτοματισμού που ονομάζεται ASP. Τέλος θα παρουσιάσουμε τα πειραματικά αποτελέσματα που συλλέξαμε από αυτή την δουλειά και θα εμβαθύνουμε στο γεγονός κατά το οποίο η συγκεκριμένη μεθοδολογία που αναφέραμε παραπάνω, διαχειρίζεται με επιτυχία τις παραπάνω υποθέσεις και τους τύπους κυκλωμάτων.

Acknowledgements

First of all, I would like to thank my thesis supervisor Prof. Christos Soritiou for his guidance and his support through this research project. I would also like to thank Prof. Georgios Stamoulis and Prof. Fotios Plessas for the knowledge and guidance they provided to me throughout my academic journey.

After that, I would like to express my gratitude and thank my research fellows at CAS Lab, especially S. Simoglou, N. Sketopoulos, D. Valiantzas, C. Georgakidis, for their support, their knowledge, and the friendly environment they provided to me to complete my thesis.

Finally, I am really grateful to my family and to all my friends through these years for their support and the moments we shared.

Nikolaos Blias
Volos, 2021

Latch STA Time-Borrowing Implementation with and without Loop Breaking

Nikolaos Blias
nblias@uth.gr

Υλοποίηση Στατικής Χρονικής Ανάλυσης Δανεισμού Χρόνου Μανταλωτών, με ή χωρίς Τομή Κύκλων

Νικόλαος Μπλιας
nblias@uth.gr

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The Declarant

Nikolaos Blias

14/9/2021

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

With the increasing use of electronic systems, modern VLSI circuit implementations require smaller and more energy-efficient circuit designs. The two fundamental sequential electronic circuit structures are the Flip-Flop and the Latch. Latch-based designs have many advantages over Flip-Flop-based designs such as timing, power, and area but their use is constrained. The major difference between a latch and a flip-flop is that latches are level sensitive, meaning they operate whenever the input changes from zero binary level to one binary level or from one binary level to zero binary level. Flip-Flops are edge-triggered, meaning they turn on when the clock signal changes from low to high or high to low. As a result of this circumstance and the scenario known as time borrowing, latches tend to operate faster and contribute to better power performance as the presence of the clock is absent. [1] In the case of the area, latches take up a smaller area, but flip-flops, which is made up of two latches and a clock, take up more. However, when compared to flip-flops, the existence of level-sensitive latches might make circuit timing analysis more complicated, which is why their use is limited.

The STA engines are now one of the most significant components of EDA tools. A timing engine can easily detect timing errors in multiple parts of the EDA flow, allowing for circuit validation and verification through many stages of the design process. However, state-of-the-art STA engines are dependent on several hypotheses about the essence of the design and the combination of the nature of Latches may lead to timing errors and poor quality results. In more detail, the existence of combinational feedback loops in a Latch-based design may drive the STA engine to disable these feedback loops to minimize the effect of the violation-timing error on the circuit timing validation. [2] Concerning the above, in some cases where more than one Latches clocked on different phases are transparent simultaneously, may come up with a wrong timing analysis. Another case that can cause a wrong timing report is the procedure of a setup checking on a Latch. It's worth noting that the algorithms that make up an industrial STA engine work only with Directed Acyclic Graphs. As a result, due to the presence of combinational cycles, such engines are incompatible with asynchronous designs. Regarding the previous, one clear example of when STA engines fail is when dealing with Bundle Data designs that use the Desynchronization methodology with latches. Asynchronous controllers generate enable signals for the two Latches in the Master-Slave model in these situations.

The incorrect timing analysis in the controllers will propagate the error to the Latch data path, resulting in poor quality timing outcomes.

1.1 Aims of this Work

Considering the previous, we can infer that robust and dependable STA engines are in high demand nowadays, since they are one of the most important validation tools in EDA flows. Secondary Latches provide several advantages over Flip-Flops, as described, but they also provide a number of challenges in terms of signal and timing analysis.

This thesis aims to present and implement a Latch STA methodology named Fast ICV that can properly handle the industrial STA engine above hypotheses. In further detail, this project adopts a new Latch STA methodology that comes with an automated procedure for giving an accurate and effective Latch STA timing analysis, taking into account the proper assumptions on the timing existence of a Latch and the correct STA analysis fundamentals. It is worth mentioning that this Latch-STA methodology has been designed and integrated on an Electronic Design Automation (EDA) tool named ASP and supports Synchronous, Cyclic, and Acyclic circuits, and is also adaptable with the existing ASTA engine of the ASP tool in order to provide timing analysis on Bundled-Data designs, based on the combination of the two methodologies.

1.2 Execution Plan

Taking into account the STA fundamentals and by providing the appropriate files, meaning the Verilog netlist and the Technology Timing library, the Latch STA engine is ready to begin operating on the worst-case delay analysis. It's worth mentioning that this methodology's flow is totally automated, allowing the user to easily specify the environment parameters, such as the periods of the Latch clocks phases, their waveforms, and the appropriate feature to be applied based on the circuit's type for complete timing analysis. In more detail, having a netlist and the correct timing information for the design under analysis we can produce a complete timing report which can help us validate the Latch-based design timing rules. For further validation and correlation with the state-of-the-art STA engines, our timing engine produces an SDF file that can be used for dynamic simulation. Taking a brief view of the above steps:

- **Load Design**: Provide the netlist and the Technology Timing Library.
- **Enable Latch Analysis**: Enable the correct analysis features based on circuits type.
- **Report Timing**: Complete log of the Latch timing analysis.
- **SDF Simulation**: Proceed to Dynamic Simulation.

Finally, we should point out that in the following chapters of this thesis, we provide a detailed explanation of the results produced by our STA engine in comparison to the state-of-the-art STA industrial tool, as well as the complete experi-

mental methodology that covers all of the circuits our Latch STA methodology supports.

Chapter 2

Theoretical Background

We must first review the fundamental concepts of static timing analysis before moving on to the core of this work. **STA** is one of the most significant aspects of the EDA flow, thus having the essential understanding of the concepts on which this work is developed is necessary.

2.1 Static Timing Analysis Fundamentals

2.1.1 Static Timing Analysis

One of the numerous approaches for verifying the timing of a digital design is **Static Timing Analysis (STA)**. Timing simulation, which may check both the functionality and the timing of the design, is an alternative method for verifying the timing. As it is clear, timing analysis simply handles the timing issues of a design. When comparing the two validation techniques, STA is static since it analyzes the design statically and does not rely on input vectors being applied at the input pins. [3]

2.1.2 Setup and Hold Times

For the circuit to provide proper data propagation, we must verify various timing checks in sequential elements. Because synchronous circuits require the presence of a clock, these checks assist the designer in ensuring that the correct data is latched at the appropriate clock edge and that the correct input data is unambiguous. We refer to these checks as **Setup** and **Hold** times.Figure 2.1 [3]

- The **Setup** time is the amount of time before the active clock when the data input must stay stable.
- Similarly, the **Hold** time is the least amount of time that the data input must remain stable following the clock's active edge.

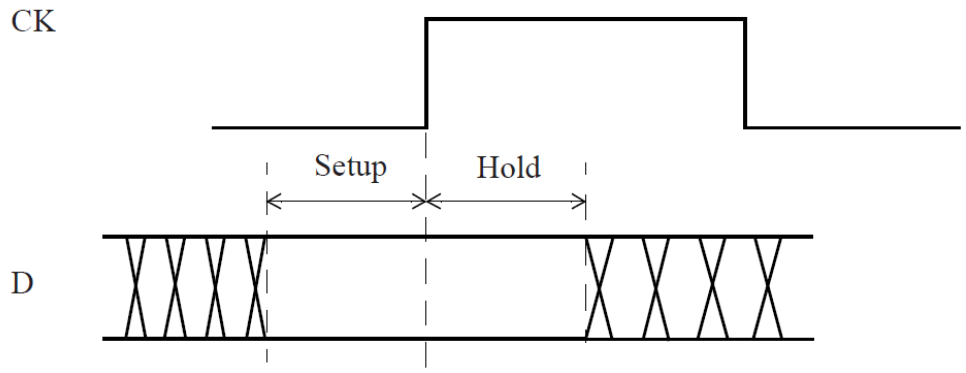

Figure 2.1: Setup Hold Checks at Flip-Flop

2.1.3 Timing Arcs and Unateness

The Static Timing Analysis is fully dependent on the existence of timing paths in the circuit. These timing paths are defined from a starting point to a boundary or an ending point. But which are the fundamental elements that a timing path is constructed on? Each circuit cell consists of several **Timing Arcs**. Timing arcs define the way the cell output is going to change at different input transitions. We refer to the above as **Unateness** or **Timing Sense**, which is a kind of relationship between input and output pins.

Each type of cell has its timing arcs. Combinational cells such as AND, OR have timing arcs from each input to each output pin. On the other hand, sequential cells such as flip-flops and latches have timing arcs from the clock to the output and timing constraints from the data pins to the output pins. [3]

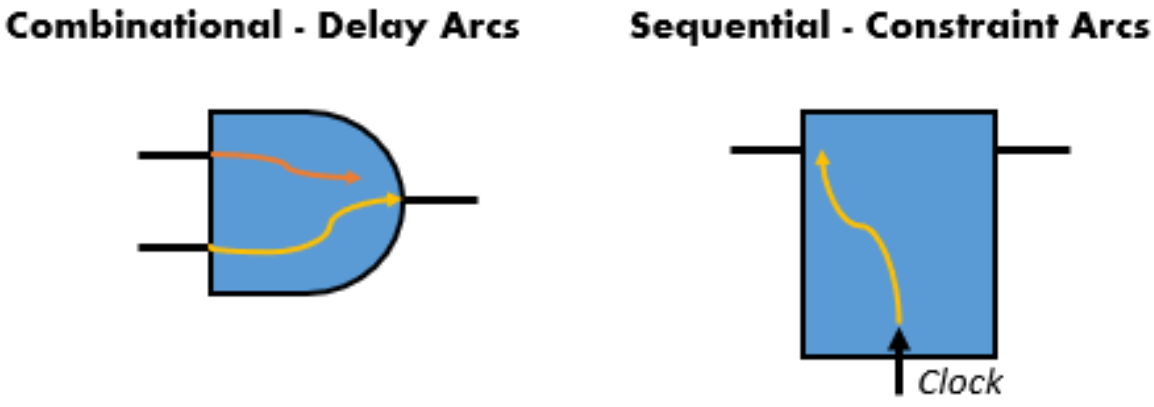

Figure 2.2: Combinational and Sequential Timing Arcs

In more detail, grouping the timing arcs based on their unateness: [3]

- If a rising transition on an input causes the output to rise or not change, and a falling transition on an input causes the output to fall or not change, the timing arc is **positive unate**.
- A **negative unate** timing arc is one in which a rising transition on an input produces a falling transition on the output, and a falling transition on an input causes a rising transition on the output.
- The output transition in a **non-unate** timing arc is not simply influenced by the direction of change of one input, but also by the state of the other inputs.

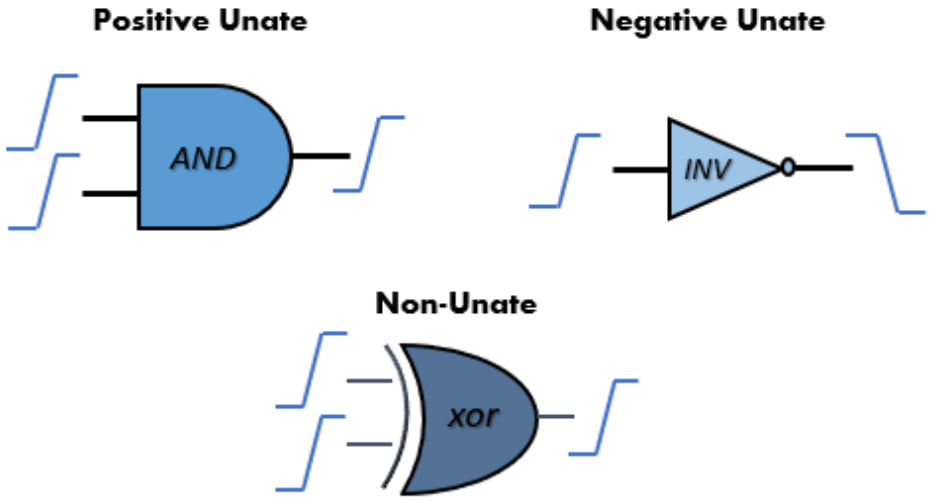

Figure 2.3: Example of Timing Sense Characterisation

2.1.4 Non-Linear Delay Model

Traditional STA engines to perform are dependant on timing libraries. Several models demonstrate the timing information from the library cells, however, in this work, we focus on the **Non-linear Delay Model (NLDM)**. In this model delay, output slew, and delay values are provided for each cell combinational or sequential (timing checks are also provided from lib files for sequential elements) through the **Look Up Tables (LUTs)**. We can imagine a LUT as a 2D array that given the input slew and the output capacitance we can compute the output slew and the delays of each cell. Having a timing arc between an input and an output pin we can use the input slew and the output capacitance as indexes in the 2D array to calculate the output slew and the delay values. It is worth mentioning, that in most cases where we cannot map exactly the indexes in the array bi-linear interpolation is used for the extraction of the corresponding values.

2.1.5 Graph-Based and Path-Based Analysis

There are two basic approaches when it comes to STA. The first is **Graph-Based Analysis (GBA)**, which is the most often utilized by STA tools. **Path-Based Analysis (PBA)** is the second one.

When it comes to setup analysis for GBA, the output slew and delay of the cell are calculated using the **worst** input slew of each gate. In the case of PBA, we pick the **actual** slew and delay for each combination of timing arcs at a cell while traversing the whole set of timing paths.

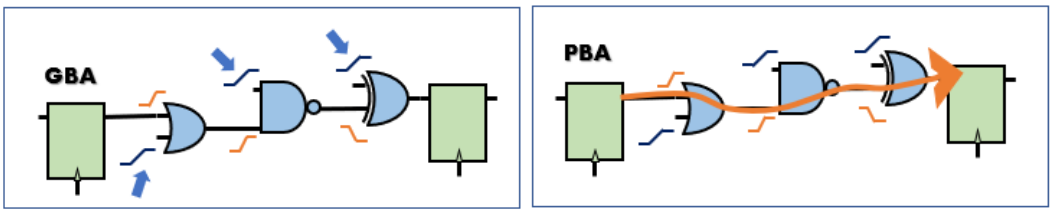

Figure 2.4: GBA and PBA Methodologies

GBA is preferred due to the polynomial algorithmic complexity in contrast to PBA having exponential algorithmic complexity in the effort to traverse all of the timing paths. However, in terms of results, PBA is more accurate due to path discovering and it adds no pessimism in the whole analysis. In terms of GBA extra pessimism is added and not all of the paths are detectable in the circuit timing analysis.

2.1.6 Delay and Slew Propagation

STA engines work with directed acyclic graphs, as previously stated. These graphs are known as **timing graphs** because they are made up of the entire set of **timing arcs** associated with the circuit. STA algorithms perform delay and transition time propagation over all possible combinations of input and output cell pins concerning **worst-case** analysis (max) and **best-case** analysis (min) to execute STA on these graphs. In all situations, delay and slew propagation on timing arcs are independent. Delay and transition time at the driver output pin are dependent on input net transition and total output net capacitance. The above assumptions are also used in the implementation of the Fast ICV method.

Each of the timing paths has a startpoint and an endpoint. Startpoints are usually referred to as starting from a sequential elements output pin or a primary input. All possible cells in the circuit would be traversed by the STA algorithms. **Arrival time (AT)** refers to the time when a signal arrives at a certain place at a specific time. The predecessors of this cell will be included in the computation since it is obvious

to compute the arrival time at a certain point. Arrival times are usually come up as a pair of the worst-case analysis and the best-case analysis, meaning the latest moment a signal can change and the earliest moment a signal can change. Suppose we have u and v where u is the predecessor of v, defining $AT(v)$ for late and early analysis. $FI(v)$ stands for the fanin of the node: [3]

$$AT_l(v) = \max_{u \in FI(v)} (AT_l(u) + d_l(u, v))$$

$$AT_e(v) = \min_{u \in FI(v)} (AT_e(u) + d_e(u, v))$$

In case v is the startpoint:

$$AT_l(v) = 0$$

$$AT_e(v) = 0$$

The second important terminology is **Required Arrival Time (RAT)**. We may relate this concept to the Arrival time, with the exception that we compute RAT by traversing backward from the endpoints of the timing paths. This idea is important because we use it to upper bound the real arrival times in such a way that the clock cycle is intended and the timing margin is not violated. In terms of computation, a backward topological sort is used so that in late analysis, the required arrival time at a pin is calculated by subtracting the timing arc delay from the minimum required arrival times of the successor pins. The early analysis uses the same idea, subtracting the maximum successors' arrival times by the timing arc delay. Defining $RAT(v)$ for late and early analysis for a pair of v and u. $FO(v)$ stands for the fanout of the node: [3]

$$RAT_l(v) = \min_{u \in FO(v)} (RAT_l(u) - d_l(u, v))$$

$$RAT_e(v) = \max_{u \in FO(v)} (RAT_e(u) - d_e(u, v))$$

In case v is the endpoint:

$$RAT_l(v) = Tclk - t_s$$

$$RAT_e(v) = t_h$$

Where t_s stands for setup time and t_h for hold time.

Now that we review the two fundamental terminologies for delay propagation, we can state that the following conditions must be preserved for proper circuit operations, at every node of the graph:

$$AT_l(v) \leq RAT_l(v)$$

$$AT_e(v) \geq RAT_e(v)$$

Slack is the value that reflects the connection between AT and RAT and is determined by the difference between these two values:

$$slack_l(v) = RAT_l(v) - AT_l(v)$$

$$slack_e(v) = AT_e(v) - RAT_e(v)$$

If slack is positive, no timing violation occurred for that pin, and we can increase AT as the value of slack is at this point, without propagating a timing error. If slack is negative, on the other hand, and we want our circuit to work within the established conditions, the path to this pin must speed up. Finally, slack is a useful indicator for determining the clock parameters in our circuit. [3]

2.2 Latch Timing

As discussed above, a latch is a sequential element that is level sensitive, meaning they operate whenever the input changes from zero binary level to one binary level or from one binary level to zero binary level. Flip-Flops are edge-triggered, meaning they turn on when the clock signal changes from low to high or high to low. As a result of this circumstance and the scenario known as time borrowing, latches tend to operate faster and contribute to better power performance. In this section, we are going to review the above terminologies in detail.

2.2.1 Latch Timing Arcs

Latches are sequential elements that are level triggered, meaning they operate whenever the input changes from zero binary level to one binary level or from one binary level to zero binary level. There are two types of timing arcs in a latch: **Data to Out** and **Enable to Out**. As it is clear, there are two possible ways for propagating data to output, change Out based on Data and change Out following Enable.

Let's assume we have a positive level-sensitive latch. In that case, data propagation will pass to Out when Enable is to one binary level. So in case enable is "1", and Data toggles, the propagation from Data to Out will directly occur. As we can see in the following picture Out follows Data changes.

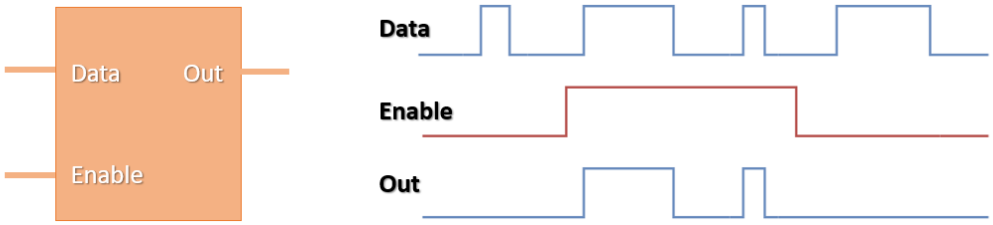

Figure 2.5: Positive Level Sensitive Latch Data to Out Timing Arc

On the other hand, if data start to change when Enable is "0" there will be no toggle at the Out till Enable switches to "1". So Out would follow Data only when Enable change. In the following picture, we can observe how Enable affects the change at the output.

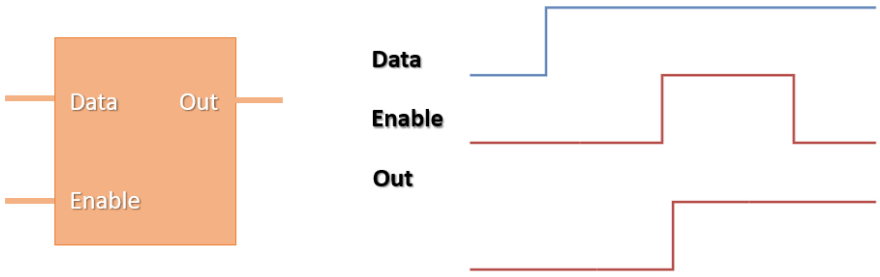

Figure 2.6: Positive Level Sensitive Latch Enable to Out Timing Arc

2.2.2 Latch Timing Checks

As seen in the above lines, the change in Output is very dependent on the relationship between Data and Enable, and because latches are sequential elements, we must apply certain conditions to ensure that data propagation does not negatively impact Output. As previously stated, latches are level-sensitive elements, implying that data propagates across the whole active enable window. So, what if Data toggles extremely near to Enable when it goes from one level to another? We may infer that setup and hold timing checks in latches are necessary, particularly at the closing enable edge to prevent data propagation errors at the output.

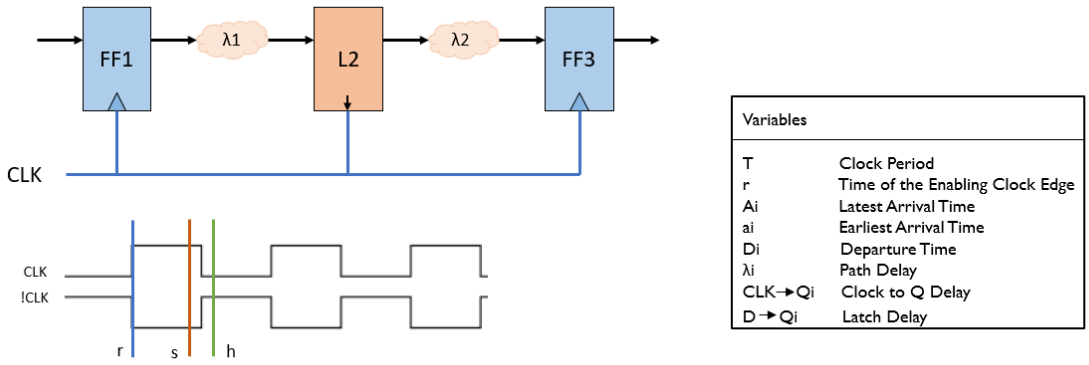

Figure 2.7: Latch Setup and Hold Check

Given a linear pipeline with two Flip-Flops and a latch at the middle we can define timing checks in a latch as follows:

$$A_L \leq T/2 - s_L$$

$$a_L \geq T/2 + h_L$$

We can guarantee accurate data propagation to the output at the next clock edge by performing this setup check. We persuade ourselves that data will not propagate to the output at the current or prior clock edge by holding the data and performing the hold check. Finally we can conclude that since latches are transparent for half of

the period's clock, setup and hold checks come up concerning latch enable **closing** edge.

2.2.3 Latch Time Borrowing

It is obvious at this point that a level-sensitive latch may propagate data during the entire time Enable is asserted. In contrast to edge-triggered flip-flops, a latch is transparent for half of at its active clock period, which might lead to a significant concept known as **Time Borrowing**. [4, 5]

First, consider how flip-flops would operate in the following design scenario. The clocks' period is `10 ns`, and all four flip-flops are positive edge triggered. *Flip-Flop 1* and *Flip-Flop 3* are clocked on clock one, whereas *Flip-Flop 2* and *Flip-Flop 4* are clocked on clock two.

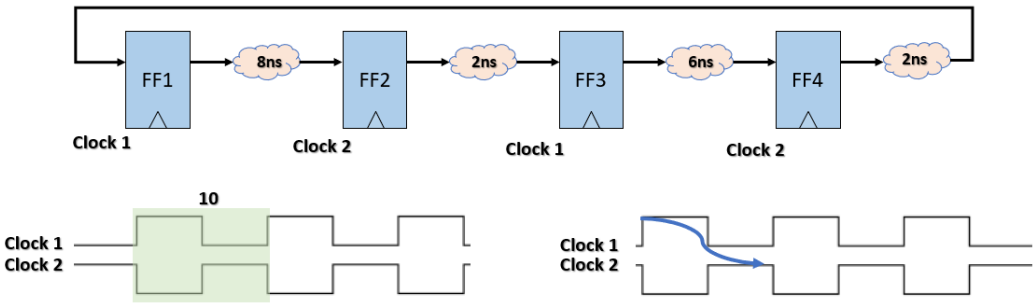

Figure 2.8: Flip-Flop circuit scenario

Examining the combinational delay at each stage of this four-stage cycle circuit shows that there is a `8ns` delay from *Flip-Flop 1* to *Flip-Flop 2*, implying that this maximum path delay forces the circuit to operate under the period of at least `8ns`. Furthermore having a period of `10 ns` and a pulse width of `5` data violations may occur at some timing paths. Using the *Flip-Flop 1* to *Flip-Flop 2* path as an example, data will arrive after the positive edge of *Flip-Flop 2* after *Flip-Flop 1* launches the data, potentially resulting in a violation.

On the other hand, how would latches respond to the above findings we made for the flip-flop-based design? Each flip-flop is replaced with a latch in the schematic below. The period, the pulse width of the clocks and path combinational delays remain unchanged. All four latches are positive level sensitive. *Latch 1* and *Latch 3* are clocked on phase one, whereas *Latch 2* and *Latch 4* are clocked on phase two.

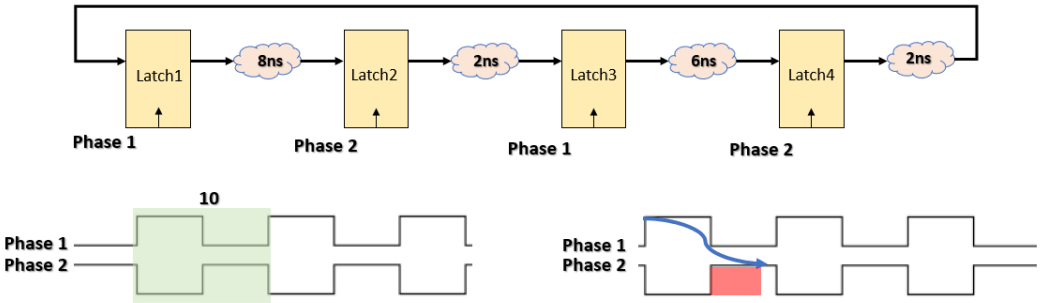

Figure 2.9: Latch circuit scenario

As a starting point, let's look at the *Latch1* to *Latch 2* path. Returning to our memory, latches are level sensitive, which implies that *Latch 2* will be transparent for half a period in our situation, in contrast to edge-triggered flip-flop. In such an instance, there will be no violation from *Latch1* to *Latch 2* since the data will be captured by Latch 2 at the *8th ns*. As a preliminary observation, we can see that timing violations may be avoided by using level-sensitive latches. The second important point to note is that when the data is captured at the *8th ns* by *Latch 2*, data propagation may proceed to path *Latch 2* to *Latch 3* as that design path is short enough to allow for appropriate data propagation. Latch usage helps the design to operate faster and not forcing the period to be at least *8ns* as it was done by flip-flops. We can refer to that technique that lets the longer combinational paths "borrow" some time from the following shorter combinational paths as **Time Borrowing**. [4]

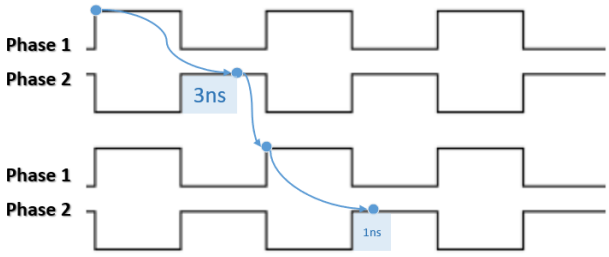

Figure 2.10: Latch Time Borrowing

More specifically, since a latch is transparent, data might come after the latch enable clock edge, allowing it to borrow time from the following cycle, reducing the time for the next stage. [3] Typically borrowing occurs in the same clock cycle. Summarizing the main goals of time borrowing we can conclude that: [4]

- Time borrowing helps us avoid timing violations.
- In a multistage circuit, the time of each stage would be reduced.

NOTE: At the above cases we made the assumption that all the sequential elements are ideal, meaning they do not have an internal delay, and the library setup time at both cases is zero.

2.3 Bundled-Data Design

Asynchronous designs have many advantages over synchronous designs in case of performance. The absence of the clock may drive better performance and power results. One significant example is the Bundled-Data design and more specifically in our case, a bundled data design with latches is the Desynchronization methodology. Through this methodology, asynchronous controllers generate the clock signals for the synchronous data paths, which indicates that Asynchronous Static Timing Analysis is required for the asynchronous part and STA for the synchronous data path. [6, 7] First consider the circuit at picure 2.11 before the Desynchronization method. [6, 7]

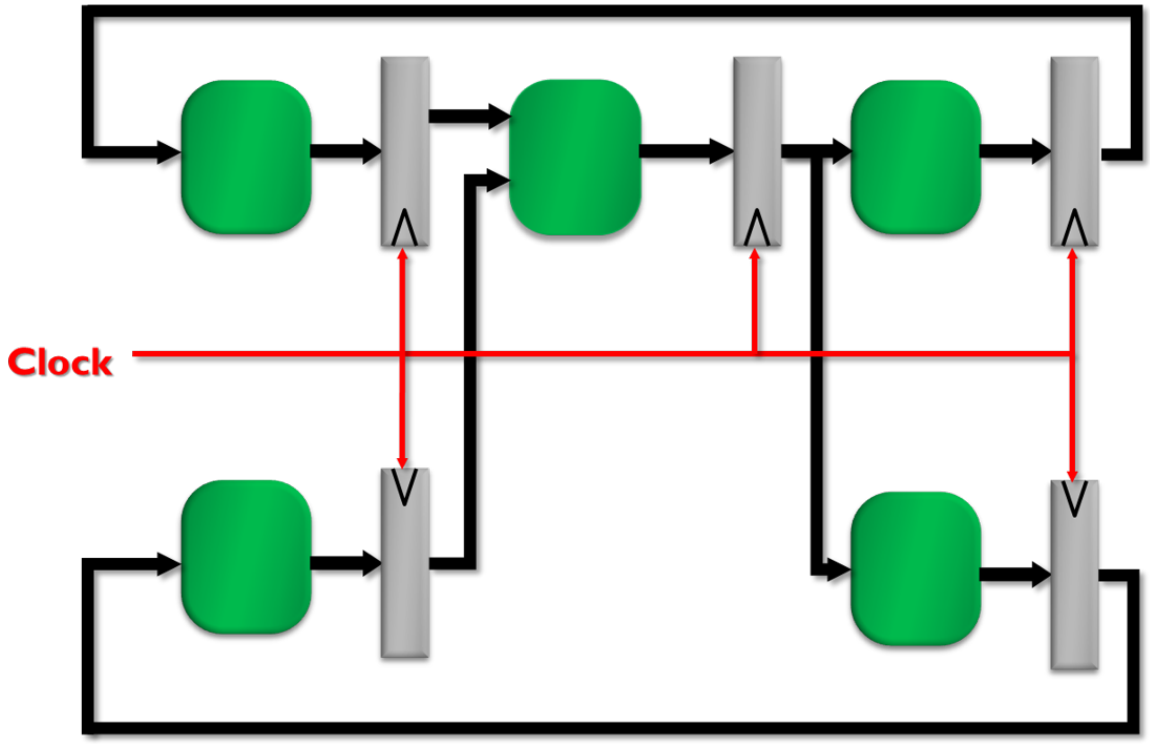

Figure 2.11: Circuit Before Desynchronization Methodology

Following up as shown in figure 2.12 [6, 7] every flip-flop is replaced with two latches, a Master and a Slave, and as it clear each controller provides the appropriate enable signal to each latch. The transformation into latches it is not deeply related to the Desynchronization methodology, however as stated at the previous section latches can contribute to better timing results and avoid violations. So the key idea is, that till the data are not stable the enable must be delayed. [6, 7] Finally in figure 2.13 we can observe the complete outcome of this process. [6, 7]

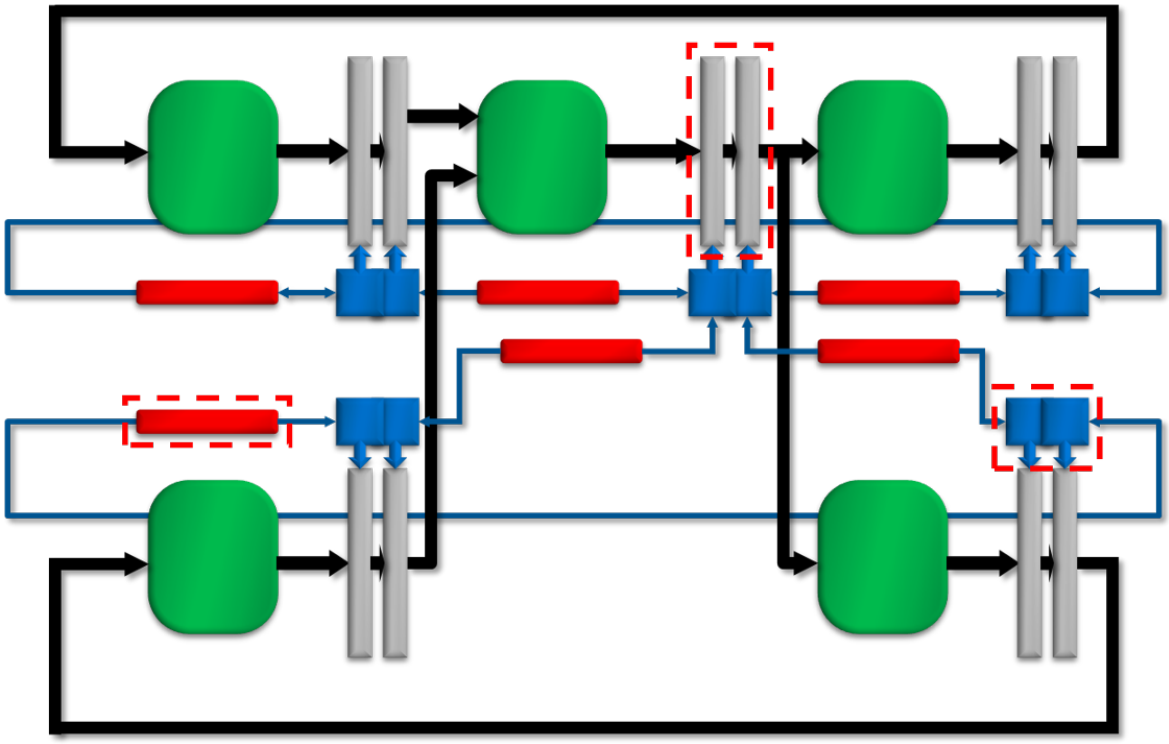

Figure 2.12: Desynchronization Methodology

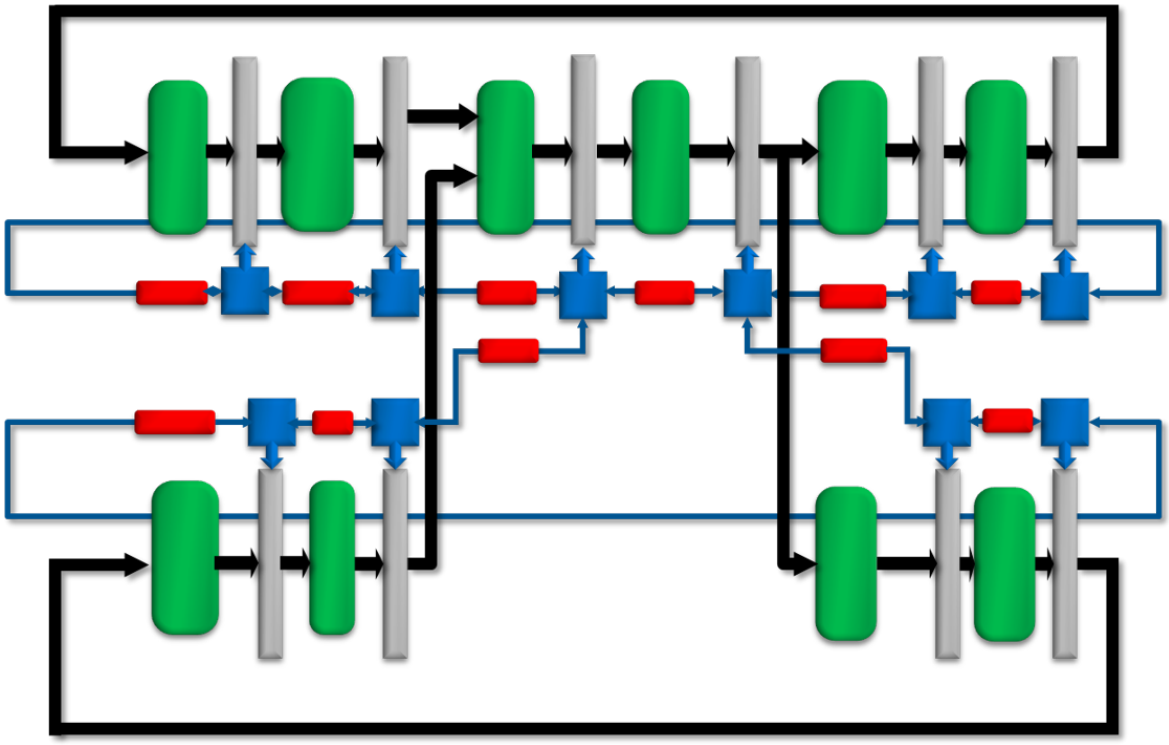

Figure 2.13: Desynchronization Outcome

Chapter 3

STA for Latch-based Design Approaches and Challenges

Let us look at the challenges that STA engines face when it comes to Latch STA and the proposed literature methodology that helps us come up with our implementation before we look at the suggested technique, Fast ICV.

3.1 Industrial Tools STA Latch-based Design Challenges

As previously said, STA engines have become one of the most important components of EDA tools. A timing engine can quickly detect timing errors in various portions of the EDA flow, providing circuit validation and verification at various stages of the design process. However, modern STA engines are based on numerous assumptions about the design's essence, and the nature of Latches combined with timing mistakes can result in poor quality outcomes.

3.1.1 Latch Loop Breakers

Traditional STA engines operate on the level of directed acyclic graphs. In that case gate pins are the nodes, timing arcs are the edges of the graph, and sequential elements represent the boundary points in the analysis. More specifically, the presence of combination feedback loops in a design may cause the STA engine to disable these loops in order to reduce the impact of the violation-timing error on circuit timing validation. When this cycle cutting technique is used, the timing arc or disabled data point may have an unrealistic transition time, resulting in erroneous slew propagation in separate timing arcs and inaccurate slew calculation analysis across cycles. In more detail, as already stated output transition and output delay are directly dependent on input transition, so by cutting a data arc an unrealistic delay would be calculated. [2, 8] Figure 3.1 demonstrates cycle cutting and how it affects delay propagation.

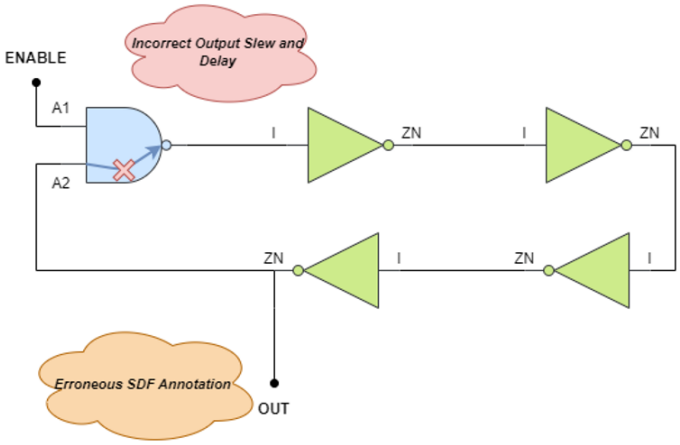

Figure 3.1: Cycle Cutting

In some cases, designers come up with setting the transition time at the cut point manually, however that technique still may lead to timing errors. The impact of this error is also carried to SDF simulation, where incorrect delay values are annotated and the designer gets a false image of the design. In conclusion, the performance and results may be completely misleading. [2, 8]

Moving on, let's add some sequential latch logic to the design above as shown in figure 3.2 to see how cycle cutting affects STA analysis. It is possible to create sequential loops of transparent latches when latches are spotted in a cyclic design. These loops would be identified by the industrial tool, which would then construct a set of latch pins that were involved in the loop. The user may examine the sequential loop using a specific reporting command, which shows which pins each loop contains as well as certain unique arguments for each pin. [9]

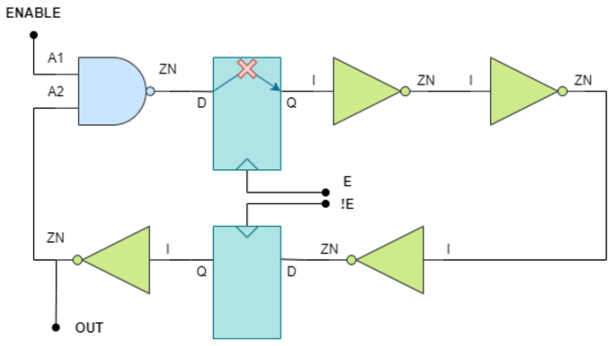

Figure 3.2: Latch Cycle Cutting

One of the arguments reported for some latch data pins is that a loop breaker is set on those pins, suggesting that the industrial tool performs cycle cutting through sequential latch loops as well. In the above case, transition time propagation would

proceed from G to Q as the D to Q arc is disabled. As in combinational loop cases, an unrealistic slew and delay value may be computed at the point loop breaker is set and the worst transition time from D to Q will not be propagated. In some situations overall circuit timing performance won't be affected, however, there is a chance that may lead to erroneous results. Timing analysis failure by cycle cutting is directly dependent on the nature of the clocks latches are clocked on and in which timing windows the computation of the delay values is performed. Figure 3.3 demonstrates three cases of clock waveforms. In the first case, the two phases do not have an overlap, meaning that each time latches of one phase will launch data and latches of the second phase will capture data. In more detail, latches clocked on different phases will not be simultaneously transparent and the delay and slew computation would be procced from the E to Q window as it seems to be the dominant timing window, so cycle cutting at D to Q won't affect timing analysis.

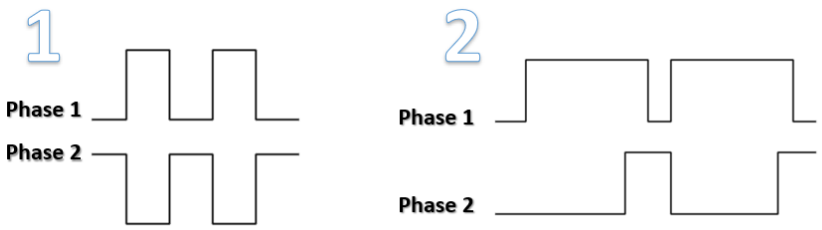

Figure 3.3: Non Overlapping and Overlapping Clock Waveforms

Cases where the clock phases have an overlap as is demonstrated at the second waveform in figure 3.3 may drive to situations where latches from different phases would be transparent at the same time. In that case, delay and slew propagation would proceed from D to Q timing window, thus it is clear that cycle cutting would affect the whole timing analysis of the circuit in a negative way.

One key example of a latch-based design that cycle cutting would affect negatively timing analysis is Bundled-Data Design. As already stated, Bundled Data design consists of asynchronous cyclic controllers and the synchronous latch data path. Cutting the cycles in the asynchronous part would lead to false enable signal propagation at the data path which indicates a disastrous outcome.

All of the outcomes of the above challenges have been analyzed in detail in the experimental section.

3.1.2 Latch Timing Checks

Latches are level-sensitive elements, which means that data propagates over the whole active enabling window, as previously mentioned in the background section. So, what if Data toggles very close to Enable as it progresses from one level to the next? To prevent data propagation problems at the output, setup and hold timing checks in latches are required, particularly near the closing enable edge. The point we choose to make the setup check for a latch is critical as it would affect directly the slack computation. Refreshing our memories, slack is the value that reflects the connection between AT and RAT. If slack is positive, there was no timing violation for that pin, and we can raise AT as long as slack is positive at this moment without propagating a timing mistake. If slack is negative, on the other hand, and we want our circuit to operate as expected, the path to this pin must accelerate. As it is clear, slack is a valuable indicator for determining our circuit's clock parameters. [3]

The industrial tool, on the other hand, suggests that slack may be calculated in two ways. The first is to look for a setup violation at the closing edge, which indicates the proper way to latch the data. In the second case, the industrial tool checks the setup with the worst window between closing and virtual, referred to as the opening window or virtual window. Figure 3.5 demonstrates the above mentioned.

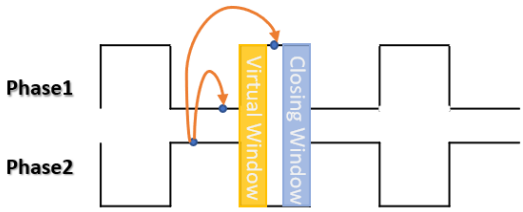

Figure 3.4: Virtual and Closing Window for Setup Checking

The above calculation may cause an inexperienced designer to have a distorted view of their work. As previously indicated, margin slack can help the designer in determining the clock parameter in which the design will operate. Having a much tighter slack than the slack that will be computed at the closing edge confuses the designer in such a way that he may not put as much pressure on the clock as he can. In the experimental section, we summarize the previous, noting how this calculation impacts the overall overview of the timing analysis.

3.2 Iterative Constraint Verification Algorithm (ICV)

In this section, we are going to provide a detailed overview of the Static Timing Analysis algorithm we used as a basis to construct the proposed methodology. Iterative Contraint Verification (ICV) is the approach under examination, and its major contribution is to properly handle the STA case for latch controlled circuits. [10] Taking a quick look before digging deeper into the above-mentioned algorithm, we picked that technique as the core of our work for the following reasons. ICV works with graph-based circuit transformations each is easily adaptable to the needs of our STA engine. Furthermore, as we are going to study in detail at the following lines, the above graph transformation and the iterative nature of that methodology helped us come up with a methodology that operates in reduced iterations than the existing methodology without dealing with sequential loops by cycle cutting. Finally, this approach is ready to be used as long as the latch-based design and clock parameters are provided, thus it is not dependent on a large number of input resources to generate the desired results.

3.2.1 Basic Timing Formulations

Latch-controlled circuits are constructed basically of latches separated by combinational logic between them. This methodology adopts a simple graph transformation in order to demonstrate the above relationship. We can refer to that graph representation as Reduced Timing Graph (RTG) or Latch Graph. The two fundamental structures of that graph, are the latches which are represented by the nodes of the RTG, and the edges, which represent the combinational delay between a stage of one latch to another. Figure 3.6 demonstrates a simple example of RTG [10, 11]

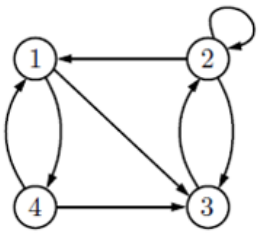

Figure 3.5: Reduced Timing Graph

Every path has a starting node and an ending node. We can refer to the successors of a node as the fanin nodes. Node 1 is the fanin node of the node 2. Note that in the case of a loop the starting point and the endpoint would be the same, like the $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$ loop in the above figure. Furthermore regarding the analysis we working on, meaning the setup or the hold analysis each of the edges would contain the related values. In cases of setup analysis, the edges would represent worst-case combinational delay, and in hold analysis best-case combinational delays would be stored in the graph's edges.

3.2.2 Timing Constraints of Latch-Controlled Circuits

Before moving to the core part of the algorithm we first need to study the timing constraints that are going to be utilized at the presentation of the algorithms and some fundamental assumptions regarding the computation of the related timing values. Figure 3.7 contains the whole set of variables that this methodology uses to perform Latch STA. Timing values are provided for both best and worst-case analysis and it is worth mentioning that most of the values are also mapped in our implementation. [10]

TIMING CONSTRAINTS OF LATCH-CONTROLLED CIRCUITS

	Variables
n	Number of latches in the circuit.
T	Clock period.
r_i	Time of the enabling clock edge in local time zone.
$E_{j,i}$	Phase shift, time difference between starting times of clock phases j and i.
D_i	Latest departure time from latch i.
A_i	Latest arrival time at latch i.
$\Delta_{j,i}$	Maximum combinational delay from latch j to i.
$\Lambda_{j,i}$	Edge weight in the latest RTG.
s_i	Setup time of latch i.
d_i	Earliest departure time from latch i.
a_i	Earliest arrival time at latch i.
$\delta_{j,i}$	Minimum combinational delay from latch j to i.
$\lambda_{j,i}$	Edge weight in the earliest RTG.
h_i	Hold time of latch i.

Figure 3.6: Timing Constraints of Latch-Controlled Circuits Variables

We investigated the timing nature of latches being transparent when the enable signal is up to its active level and when the data must be latched or not in the background section. We know that the timing values of latch STA are connected to the fundamental terminologies that are hidden at the clock's representation, and most of them are calculated with respect to the enable signal. The authors refer to the relation between the enable signal and the calculation of the timing values in the analysis as the local time zone. Having already reviewed the fundamental knowledge regarding when a latch is transparent and when the data must be latched, is simple to understand that the local time zone is a hyper set that contains all those terminologies, and indicates the proper timing value calculation. Figure 3.8 demostrates local time zone between two successive latches j and i [10].

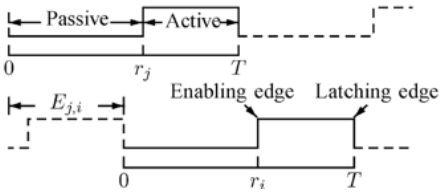

Figure 3.7: Local Time Zone

The edge that activates the latch is referred to as the enabling clock edge r_i, and the edge that closes the latch is referred to as the latching edge in this work. When the latch is transparent, we indicate the start of the local time zone at the enabling clock edge, and when data propagation is not feasible, we mark the end of the local time zone at the latching edge. As previously stated, the latching edge is always the closing edge concerning the period T. As can be seen in the figure above, there is one more parameter, E_{ji}. This parameter, named phase shift, is utilized to transform the latch clocks' starting points, in our case clocks j and i. This conversion aligns the clocks, allowing each arrival time to be calculated according to the local time zone definition. [10]

3.2.3 Setup Time Constraints

In the next lines, we'll focus on the basic setup constraints that this work applies. In the variables table, setup timing constraints are indicated as the latest values. [12]

The moment the signal from the launching latch is ready to be delivered at the capturing latches is referred to as the latest Departure time or D_i. There will be two latest departure times, one for the rise and one for the fall, as is reasonable. The following formula is used to calculate the above value: [10]

$$D_i = \max\{A_i, r_i\}$$

At each capture latch, A_i stands for the signal's latest arrival time. Behind the max condition, it is specified in which timing window the D_i will be computed, i.e., in the D to Q window or the Enable to Q window. Note that if the data arrives before the enable signal, propagation will not begin until r_i arrives. Using one pair of latches, one launch latch j, and one capture latch i as an example, the latest Arrival time can be easily defined. A combinational path must exist between the two latches, as is obvious. The latest arrival time is defined as the time when the data signal reaches the capture latch after taking into account the time when data departs the launching latch and the maximum combination delay between the two latches. When more than one latches fanin at latch i the maximum combination between the latest departure time of launching latches and combinational max path must be taken into account, for computing A_i. Looking into the equation for computing A_i: [10]

$$A_i = \max_{j \to i}\{D_j + \Lambda_{ji}\}$$

Figure 3.9 demonstrates the application of the equations above on RTG.

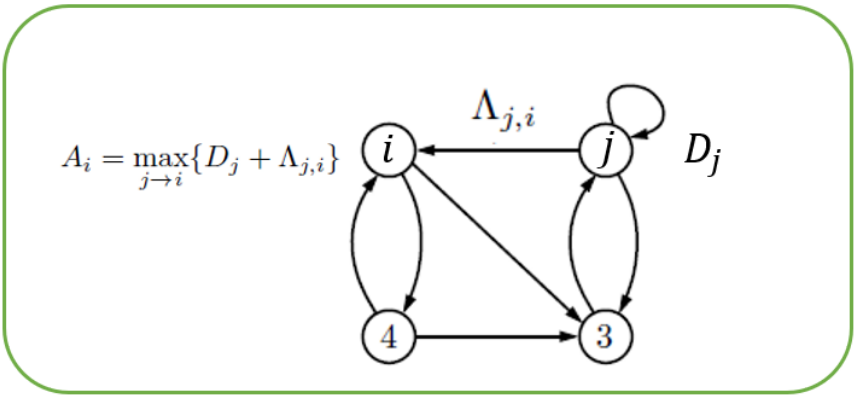

Figure 3.8: A_i and D_i Computation Example

Data propagation is always accompanied by the necessary conditions to guarantee that no violations occur and that data propagation proceeds normally. We must ensure that the data or A_i is stable for setup or s_i time, knowing that the check for

the data to be latched must be at the latching edge. Taking a look at the condition for arrival time: [10]

$$A_i \leq T - s_i$$

We may move on to the following part after examining the necessary timing formulations and variables, where we'll take a closer look at the proposed literature latch-controlled based timing algorithm. In the following lines, the Iterative Constraint Algorithm (ICV) is described in depth.

3.2.4 Iterative Constraint Verification ICV

The ICV algorithm is an iterative algorithm that works with the latch-controlled design's latch graph transformation. The signal propagation is unfolded and the concept of a simulation-based method is given through this iterative approach. It efficiently extracts setup timing constraints in a way that ensures that timing values have converged after a certain number of iterations and allows for the evaluation of timing violations in the design under timing analysis. Initialization, time constraint calculation, and constraint verification are the three parts of the ICV algorithm. Algorithm 1 represents the complete ICV algorithm.

Algorithm 1 Iterative Constraint Verification

1: **for each** node i in RTG **do**
2: $A_i^0 = -\infty$
3: $D_i^0 = r_i$
4: **end for**
5: **for** $m = 1$ **to** n **do**
6: **for each** node i in RTG **do**
7: $A_i^m = \max_{j \to i}\{D_j^{m-1} + \Lambda_{ji}\}$
8: $D_i^m = \max\{A_i^m, r_i\}$
9: **end for**
10: **end for**
11: **for each** node i in RTG **do**
12: **if** ($D_i^n \neq D_i^{n-1}$ or $A_i^n > T - s_i$) **then**
13: **return** *false*
14: **end if**
15: **end for**
16: **return** *true*

In the initialization part, we set both the A_i and D_i at the appropriate value for each node in the RTG. We set A_i to $-\infty$ as data propagation has not started yet, and D_i at r_i as data propagation for each latch would start corresponding to the enable signal.

The A_i and D_i computations are done in the next part, lines 5-10. At each iteration, the algorithm updates A_i for the relevant latch and then updates D_i. This iterative method is repeated n times, where n is the number of latches in the RTG. Each iteration may be thought of as a clock cycle in which data is propagated from launching points to capturing points. Note that each time delay values are updated incrementally based on the values of the previous iteration.

Finally, at the remaining lines, there's constraint verification, which works as long as delay propagation is complete at the final iteration. Two conditions are checked for each latch. The second one is simpler since it represents a common setup check for each latch's final latest arrival time:

$$A_i \leq T - s_i$$

The first condition, on the other hand, is more complicated since it examines if the latest departure times between the two final iterations have different values:

$$D_i^n \neq D_i^{n-1}$$

The method concludes whether the latch delay values have converged and indicates that if the latest departure times continue to increase after a specified threshold of iterations, setup violation will occur frequently from that point forward. In more detail, the algorithm, as stated in the paper, tests for the presence of a positive loop in the circuit. The circuit would be unable to function if such a loop exists. The proof that the presence of a positive loop is destructive for the circuit's operations is given as follows:

Let a loop $\{j_0, j_1, ..., j_n - 1, j_n\}$ with $j_0 = j_n$

For a given m iteration:

$$D_{jk}^m = max\{A_{jk}^m, r_{jk}\} \Rightarrow D_{jk}^m \geq A_{jk}^m \quad (1)$$

$$A_{jk}^m = \max_{j_{k-1} \to j_k} \{D_{j_{k-1}}^{m-1} + \Lambda_{j_{k-1}, j_k}\} \Rightarrow D_{j_{k-1}}^{m-1} + \Lambda_{j_{k-1}, j_k} \quad (2)$$

From (1) and (2):

$$D_{jk}^m \geq A_{jk}^m \geq D_{j_{k-1}}^{m-1} + \Lambda_{j_{k-1}, j_k} \quad (3)$$

Given (3) for a total number of n iterations:

$$D_{jn}^n \equiv D_{j0}^n \geq D_{j0}^0 + \sum_{k=1}^n \Lambda_{j_{k-1}, j_k} \quad (4)$$

Having:

$$D_{j0}^n \geq D_{j0}^0 + \sum_{k=1}^n \Lambda_{j_{k-1}, j_k} \quad (4)$$

After n iterations the D of j_0 increases by:

$$\sum_{k=1}^n \Lambda_{j_{k-1}, j_k}$$

Therefore, in order to **avoid having violation**:

$$\sum_{k=1}^n \Lambda_{j_{k-1}, j_k} \leq 0 \quad (5)$$

ICV, as we can see, functions on the n iteration boundary. Because the biggest loop would contain all of the latches, we check it at the last iteration to ensure that there are no positive loops in the RTG. If there is a positive loop in the circuit, the algorithm will fail. If the delay values do not increase after n iterations, on the other hand, we may infer that no violation occurred during the timing validation under the provided environment parameters. On the following pages a scenario of a worst-case loop is provided, that visualizes also the above proof.

Consider the following example:

- 4 Latches ($n = 4$) are connected in a circle.
- Latches $\{L1, L3\}$ are clocked on $\phi 1$ and $\{L2, L4\}$ on $\phi 2$.
- The clocks are non overlapping.

This specific scenario is considered the worst case as all of the latches form the biggest possible cycle. Figure 3.10.

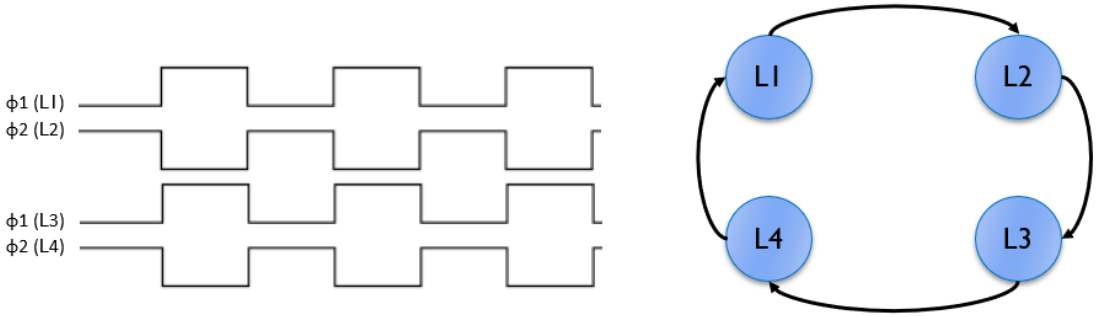

Figure 3.9: Constraint Verification (A)

All departure times D_i are set to $T - r_i$. Recall that each latch has its own time zone. Figure 3.11.

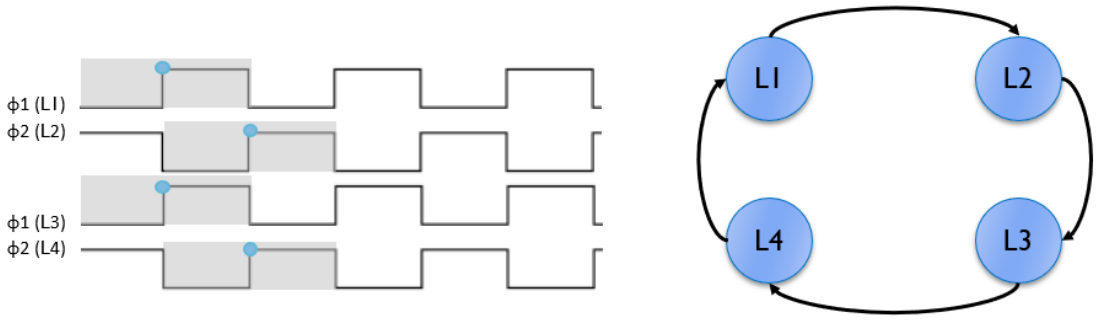

Figure 3.10: Constraint Verification (B)

Figures 3.12, 3.13, 3.14, 3.15 demonstrate the update of D_i accross iterations. Consider on that iteration only D_1 for $Latch1$ changed.

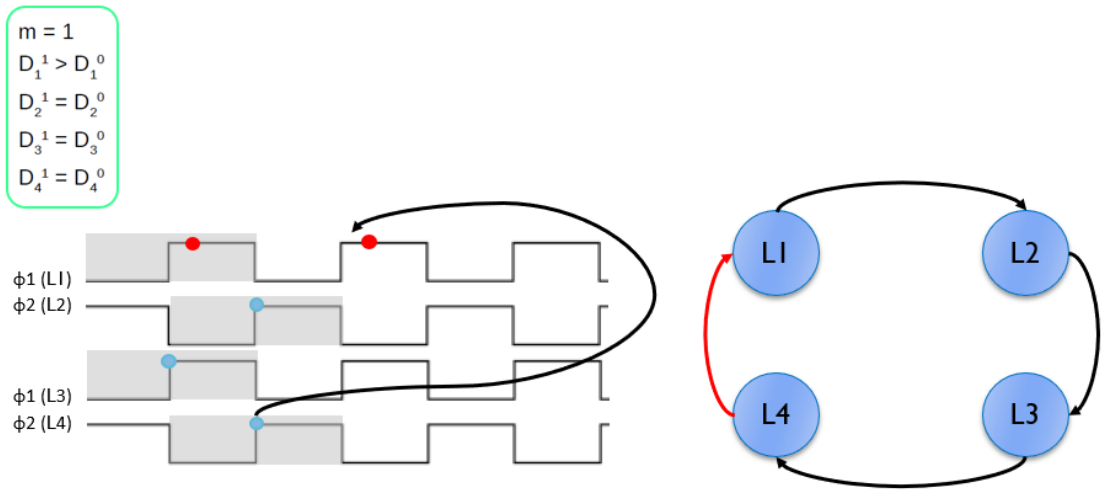

Figure 3.11: Constraint Verification (C)

Moving on we have an increase at D_2 at second iteration. Figure 3.13.

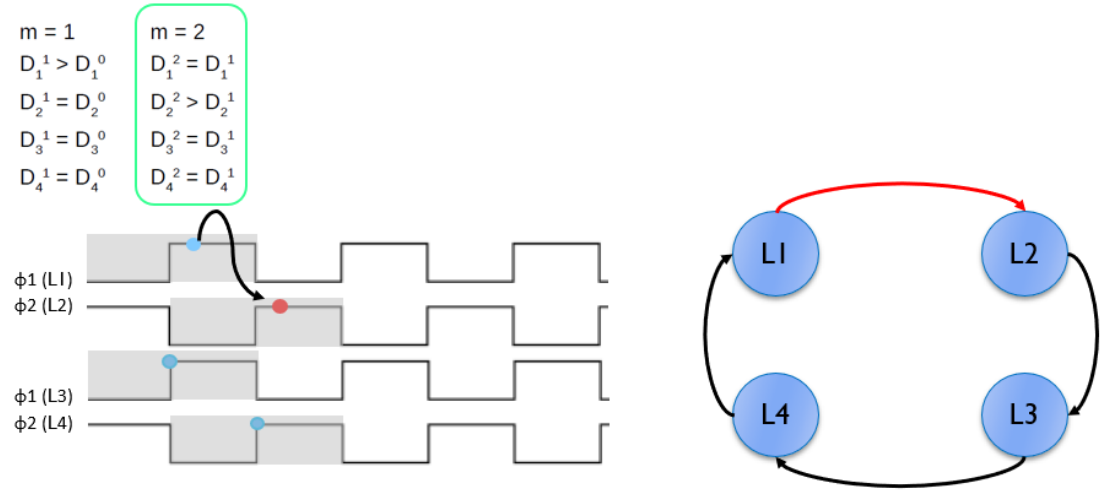

Figure 3.12: Constraint Verification (D)

D_3 increases next. Figure 3.14.

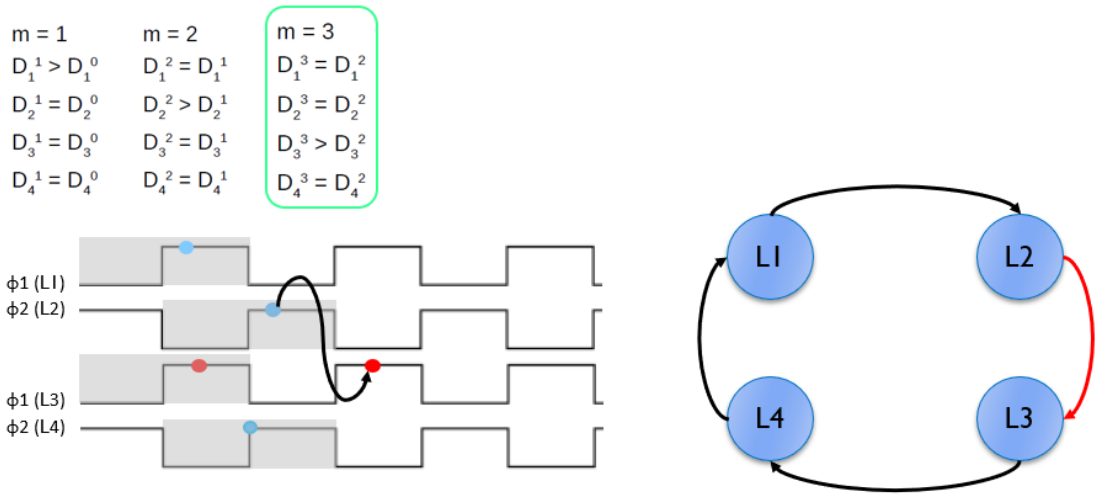

Figure 3.13: Constraint Verification (E)

Finally we observe a change in D_4 value at the last iteration, so what would happen if we proceed to more iterations? Figure 3.15.

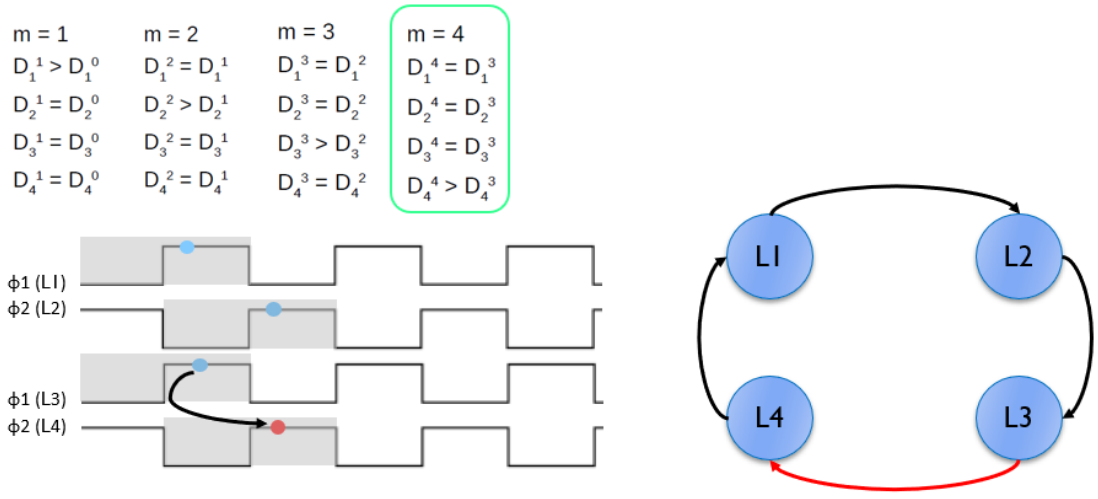

Figure 3.14: Constraint Verification (F)

At this point we can understand that the final and increased D_4 value will increase D_1 even more which will lead into a **setup violation** in the future. Figure 3.16.

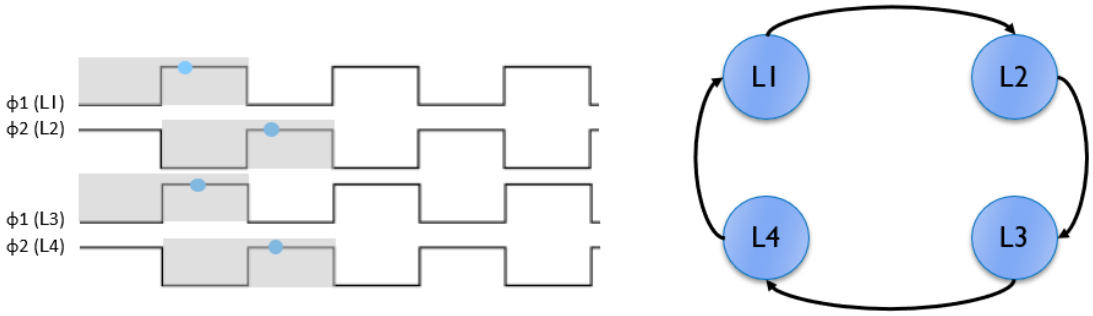

Figure 3.15: Constraint Verification (G)

In more detail, we computed $D_1^1 > D_1^0$ with D_4^0. If we proceed to a further iteration and without departure times converge we will have $D_1^5 = D_1^0 + (D_4^4 - D_4^0)$, as the cumulative departure time for *Latch*4 increased from $D_4^0 \rightarrow D_4^4$. At this point we can map $\sum_{k=1}^{n} \Lambda_{j_{k-1},j_k} \leq 0$ (5) condition into our case, as the cumulative D_4 value across iterations. In conclusion, that increment would increase D_1 even more so in following possible iterations setup violation will occur. One important observation is that this algorithm checks for setup violation at the final iteration, however it would be impossible in many cases to detect which latch cause the error as it would be propagate across iterations.

Chapter 4

A Latch-based STA Methodology Our Contribution

At that point in the work, we'll move on to the implementation section and the algorithmic contribution, but before, let's take a quick look at the above. To summarize the important points of the preceding chapters, after studying the appropriate Latch STA principles and the challenges that typical STA tools face with latch timing analysis, we identified a latch STA algorithm in the literature that satisfies the requirements of our STA tool. The implementation of the regular ICV began our quest to provide the functionality of Latch STA in our timer. We gradually uncover some of the algorithm's hidden assumptions as well as certain weaknesses. After that, we come up with Fast ICV as a variation of the current method, which covers all of the functions of the normal ICV algorithm, with the addition of certain additional features and the major advantage of operating in less iterations.

4.1 Implementation of ICV

First of all, let's look at how to put the current algorithm into action. To begin, we performed the identical procedures as in Algorithm 1, with the exception that we first updated the latest departure time for each latch, and then we updated the latest arrival time at each of the capturing latches based on the latest departure time of the launching latch. Because the algorithms bound delay values independently of traversal order, the result will not change. We make this adjustment for the sake of simplicity since we built Latch Graph in such a manner that each node holds its successors:

$$D^m_{launch\_latch} = \max\{A^m_{launch\_latch}, r_{launch\_latch}\}$$
$$A^m_{capture\_latches} = \max_{launch \rightarrow capture}\{D^m_{launch\_latch} + \Lambda_{launch\_latch \rightarrow capture\_latch}\}$$

4.1.1 Internal Latch Delay

Following up, according to the paper, the delay equations applied for the latest values suggest that the latches in the RTG are ideal, meaning they have no internal delay. To compute the latest departure time and account for the latch's internal delay, we must first determine which timing window we will work in. If $D \rightarrow Q$ is

used, the $D \rightarrow Q$ arc delay must be included in the latest departure time calculation. If, on the other hand, *Enable* $\rightarrow Q$ is the dominant method, the *Enable* $\rightarrow Q$ arc must be taken into account throughout the computation. So the above equation for departure time would change into:

$$D_i^m = \max\{A_i^m + D \rightarrow Q_{arc\_delay}, r_i + E \rightarrow Q_{arc\_delay}\}$$

4.1.2 Transition Time Propagation

Although the ICV method performs well and explains the delay propagation technique in detail, transition time propagation information is not provided in the current work. In the previous subsection, we stated that based on the timing window we work in, latch internal delay must be included in the delay calculation. As we reviewed in the background chapter latch as a sequential element possesses two-timing arcs the D to Q and the enable to Q timing arcs too. Slew propagation is an independent procedure that must be taken into account for proper timing analysis. As with delay, we choose to propagate the worst transition time each time for rising timing arcs and with respect for falling timing arcs. In respect to the latest arrival time and the timing window we choose to work in, we compute D to Q transition time or Enable to out transition time for the latch output pin as we compute latch latest departure time. Keep in mind that we first update the latest departure time for the launching latch so the slew that is computed in the current output latch pin would indicate as a starting point for slew propagation at the capture latches.

We stand for two transition time calculation cases. In the first case when the Enable to Q timing window is activated we calculate and propagated the slew from G to Q arc which is dominant. On the other hand, when the D to Q timing window is activated the D to Q slew must be propagated. When at least one combinational loop is found in the circuit, most industrial STA engines execute cycle cutting. As a result of the lack of cycles in the resultant gate pin graph, the disabled arc will have an unrealistic transition time and will not affect other gate pin slews. This results in overly optimistic results, as well as timing problems. To compute the proper slew across cycles before starting the delay and slew propagation across iterations the first step, to begin with when dealing with overlapping clocks that lead to concurrent transparent latches and the activation of D to Q timing window is the computation of an equilibrium cyclic slew for all circuit pins. We introduce the following algorithm that indicates a worst cyclic equilibrium slew for each one of the circuit gate pins before ICV starts its operations. Note that this is an iterative procedure that no delay propagation is required for this step.

Algorithm 2 Cyclic Equilibrium Slew Computation

Input: Gate Pin Timing Graph $G = (V, E, Load(V), Slew(V))$, Slew Threshold *slew_delta*
Output: Slew Annotated Timing Gate Pin Graph G

```
1: Slew(V) = 0; // Initialise Gate Pin Pin Slews to 0 //
2: Q = ∅; // FIFO Queue Q //

3: for (each Primary Input Gate Pin pi in V) do
4:    set_slew(pi, pi_slew_constraint); // Set PI Slew (user-defined) //
5:    for (each Successor Gate Pin s in pi →) do
6:       enqueue(Q, s); // Add pi Successor Gate Pins to Queue //
7:    end for
8: end for

9: while (Q ≠ ∅) do
10:   q = dequeue(Q); // next Gate Pin for Slew Computation //
11:   slew = calculate_slew(q, get_slew(q →), get_load(q →)); // Compute q Slew, based on
      input Slew and output Load //

12:   if (|slew − get_slew(q)| < slew_delta) then
13:      continue; // Continue to Another Gate Pin from Queue //
14:   end if

15:   prev_slew = get_slew(q);
16:   new_slew = max(prev_slew, slew);
17:   set_slew(q, new_slew); // Update Slew Value //

18:   for (each Successor Gate Pin s in q →) do
19:      enqueue(Q, s); // Add q Successor Gate Pins to Queue //
20:   end for
21: end while
```

Slew propagation begins at the Primary Input *PI* pins and continues across the circuit's timing arcs. The operation is repeated until the slew difference between consecutive propagation iterations of each pin approaches zero. Algorithm 2 demonstrates the cyclic slew computation procedure. In the first place, the algorithm takes as an input the cyclic gate pin timing graph G, and the slew threshold *slew_delta* which indicates the minimum slew difference between iterations. G consists of nodes V (or gate pins), E edges and each node is characterized by the load and slew calculation functions $Load(V)$ and $Slew(V)$. The algorithm then does a *BFStraversal* with a First In First Out *FIFO* queue. Slews, for instance, are user-defined. If the traversal finds an empty queue, no more slew updates are necessary; if the queue is not empty, the next pin is dequeued, and the new slew is computed and compared to the previous iteration's value. If the slew value still hasn't converged, we go through further iterations and enqueue the successor pins at the *FIFO*. Finally, the Algorithm produces as output the cyclic equilibrium slews in $Slew(V)$. [2, 7]

4.1.3 Iteration Reduction

The technique concludes if the latch delay values have converged and show that if the latest departure times continue to increase beyond a given threshold of iterations, setup violation will occur frequently from that point forward, as we discussed in detail in the previous chapter. That technique requires multiple iterations, which can be time-consuming in circuits where values converge after the first iterations or for linear circuits where one iteration is usually sufficient. We must clarify that for each latch, an incremental run must be performed in order to update each delay value depending on the previous iterations' delay values. Every circuit would have m iterations, and if n latches synthesize the RTG, the current technique would necessitate $n * m$ incremental runs, resulting in a complexity of $O(n^2 * (E + V))$.

Two key actions were implemented in order to minimize the number of iterations. The first is a basic adaptation where we update the latest values, and the second is the variation of Fast ICV that will be discussed in the following chapter. The idea behind the new adaptation is that if all of the latches' latest departure timings have converged, there is no need for more iterations. Taking a brief look at the above concept:

Algorithm 3 Iteration Reduction

```
1: for m = 1 to n do
2:    D_values_converged = 1
3:    for each node i in RTG do
4:       D_i^m = max{A_i^m, r_i}
5:       if D_i^m ≠ D_i^(m-1) then
6:          !D_values_converged
7:       else
8:          continue to next latch
9:       end if
10:   end for
11:   if D_values_converged then
12:      break
13:   end if
14: end for
```

We start the iterative procedure by assuming that the delay values have reached a point of convergence. Then, for each latch, we update the D value; if it is the same as the previous iteration's, we move on to the next latch; otherwise, further iterations are required. There is no need for additional iterations if all of the latest departure times have converged, therefore we skip the remaining iterations.

4.1.4 Constraint Verification

The constraint verification, which examines if two conditions for each latch are violated, is the final part of this method. We already analyzed the first condition for the latest departure time convergence. The second condition is easier to understand since it reflects a standard setup check for each latch's final latest arrival time. In more detail, it is clarified that if the latest arrival time grows enough to be larger than the latching edge, setup violation has occurred:

$$A_i \leq T - s_i$$

However, as discussed in the previous chapter, points setup violates are difficult to detect over iterations because they are propagated from latch to latch and across iterations. Every time data arrives at a capture latch, a setup violation might occur, as we aim to calculate the latest departure time. Despite the fact that the violation may have happened earlier iterations, the preceding condition checks for setup violation at the last iteration. This scenario causes extra iterations and makes it more difficult to discover the point of setup violation.

We noticed that we may identify setup violations from the first iterations when doing latch STA properly and preventing setup violations. It's worth mentioning that the existing method uses r_i and A_i to update the latest departure time. We keep the maximum value between the two, which simply specifies the timing window in which we will work, i.e. the Enable to Out or Data to Out time window. A setup check must be performed if the data signal comes after the latching edge, which is not done in the current method. We perform the following check before updating latest departure time:

$$A_i \leq latching\_edge - s_i$$

Because the arrival time might grow quickly in the early stages of iterations, we must verify every time the data signal arrives at a capture latch to avoid incorrectly calculating the latest departure time. Checking for setup violations in the last iteration will increase execution time and conduct numerous needless computations in circumstances when there are multiple latches and a setup violation has occurred at an early step. The Fast ICV variation also has that adaptation.

Finally, we progressively reviewed some of the algorithm's underlying assumptions, as well as some of its drawbacks. Then, as a variant of the existing approach, we propose Fast ICV, which performs all of the functions of the standard ICV algorithm while adding a few extra characteristics and having the significant benefit of requiring fewer iterations. We'll look into Fast ICV in-depth in the next section.

4.2 Implementation of Fast ICV

Implementing the existing ICV algorithm allowed us to gain a deeper understanding of the delay propagation approach used by this algorithm, as well as how the constraint verification part impacts the algorithm's overall performance. That method also benefited us in identifying some of the algorithm's underlying assumptions and drawbacks in terms of delay and transition time propagation. After reviewing the fundamental latch STA principles and identifying some of the challenges that industrial STA engines face with latch STA, we were able to combine our knowledge with the existing ICV algorithm to create a methodology that successfully hadles all of the challenges that industrial tools face while also being more robust than the existing ICV methodology. The name Fast ICV refers to a variation of the current approach ICV. This algorithm incorporates all of the aforementioned adaptations of regular ICV, as well as operates as an adaptation of an industrial tool known as ASP. In the following lines, we will analyze the entire view of the suggested approach in-depth and provide a comprehensive picture of this work through an example.

The Fast ICV algorithm is an iterative technique that works with the latch graph transformation in latch-controlled designs as regular ICV. It extracts setup timing constraints quickly in a way that assures that timing values have converged after a given number of iterations and facilitates for the evaluation of timing violations in the design under timing analysis, in a faster way than the existing algorithm. The Fast ICV algorithm has three parts: initialization, time constraint calculation, and constraint verification. The Fast ICV algorithm is represented by Algorithm 4.

Algorithm 4 Fast Iterative Constraint Verification

1: **for each** node i in *RTG* **do**
2: $A_i^0 = -\infty$
3: $D_i^0 = r_i$
4: **end for**
5: $n = clock\_num * clock\_num$
6: *D_values_converged* = 1 // decide if D values converged //
7: *A_values_update* = 0 // decide if more updates in delay values required //
8: **for** $m = 1$ **to** n **do**
9: **for each** *clock_phase* in *clock_set* **do**
10: **for each** node i in *RTG* clocked on *clock_phase* **do**
11: $D_i^m = \max\{A_i^m, r_i\}$ // Update D for launching latches //
12: **if** $D_i^m \neq D_i^{m-1}$ **then**
13: !*D_values_converged*
14: !*A_values_update*
15: **else**
16: continue to next latch
17: **end if**
18: **end for**
19: **if** *A_values_update* = 1 **then**
20: Update A^m for all capture latches // Update A for all successor latches of the current phase //
21: **else**
22: continue to next clock phase
23: **end if**
24: **end for**
25: **if** *D_values_converged* **then**
26: break // no more iterations needed //
27: **end if**
28: **end for**
29: **for each** node i in *RTG* **do**
30: **if** ($D_i^n \neq D_i^{n-1}$ or $A_i^n > T - s_i$) **then**
31: **return** *false*
32: **end if**
33: **end for**
34: **return** *true*

4.2.1 Fast ICV General Overview

When we take a brief look at Algorithm 4, we can see that it has the same three parts as normal ICV: initialization, time constraint computation, and constraint verification. The initialization and constraint verification sections are identical to those found in ICV. Following that, we should emphasize that all of the relevant changes and adaptations made for normal ICV and reviewed in-depth in the previous section found adaptation in the implementation of Fast ICV too. In more detail, accurate STA analysis requires consideration of the latch's internal delay at the latest departure time computation, as well as the worst equilibrium slew before the algorithm's operations. In the following lines, we'll look at how this technique differs from the previous methodology in terms of alternative delay and slew propagation, and how, with the right modifications, we may get accurate timing results in a reliable and fast approach.

4.2.2 Delay and Transition Time Propagation

We'll look at the primary algorithmic computing part in detail in this section. Regular ICV, to refresh our memory, executes a traversal in one RTG latch at a time, and for n iterations for all latches. Because each time a launching latch sets an incremental update run to its corresponding capture latches, this traversal is thought of as a point-to-point traversal. It's important to note that running an incremental run for each latch for n iterations can increment a lot the execution time of this algorithm. Another disadvantage of this point-to-point traversal may be seen in the situation below. When two latches combine to join to a third one, we must save the maximum value of arrival time and slew at the capture latch since the traversal may overwrite the existing worst-case values. As a result, more memory is required at each latch to store the previous delay values.

On the other hand, we can identify the fundamental difference between normal and Fast ICV by looking at lines 9-10 in Algorithm 4. It's worth noting that this modification traverses the latches according to the relevant clock phase. But how does this affect the delay and transition time propagation? Rather than traversing each latch one at a time, Fast ICV traverses each clock phase or the latches that are clocked on the same phase. In further detail, this technique bounds latch delays at the latch group level rather than considering connected latches cartesian product as in regular ICV. We can consider the first group of latches the launching latches of one phase, and the second group of latches the capture latches clocked on the rest of the clock phases. First and foremost, the latest departure time of the launching latches is updated. Following that, the latest arrival time for all capturing latches will be updated as stated in lines 11-21 in Algorithm 4. By this grouping procedure delay and transition time propagation would proceed in a level way, from phase to phase, and not point-to-point, from latch to latch. This modification affects positively both execution time and the propagation of the worst values. Keep in mind that each iteration requires one incremental update run for the latches of one phase to start propagating to the rest of the phases' capture latches. Many point-to-point incremental runs may be avoided in this manner, and the overall iterative method can be improved in terms of the number of iterations. Following up, another benefit of level to level propagation is that each iteration ensures that the worst delay and

slew are propagated from each phase without the need for comparisons between the latest arrival time produced of latches of the same phase.

As already stated, Fast ICV indicates an iterative procedure that guarantees delay value convergence after a number of iterations. Working with a different way of traversal, meaning the level to level procedure, a new upper bound was necessary for that technique so we could persuade ourselves that we can provide a formal number of iterations, under that delay values would converge. We define n in this algorithm as the product of the total phases with itself (line 5). We conclude to that as the upper bound of iterations as we consider the following scenario as the worst-case. In a worst-case situation, every phase would launch data to all the other phases and would capture data from the latches of all the other phases. As it is clear the number of the corresponding combinations is $clocks * clocks$.

4.2.3 Iteration Reduction

In order to reduce the number of iterations, two essential steps were taken. The first is a standard adaptation in which we update the latest departure times if they have not converged, and the second is the variation of Fast ICV traversing the latches according to the phase on which they are clocked. The proposed adaption assumes that after all of the latches' latest departure times have converged, no further iterations are required. We had previously looked at this adaptation in the previous section, so it was simple to adapt it to Fast ICV. We start the iterative procedure by assuming that the delay values have reached a point of convergence with *D_values_converged* (line 6). Then, for each latch, we update the D value; if it is the same as the previous iteration's, we move on to the next latch; otherwise, further iterations are required (lines 11-13, 23-27). The difference with the regular ICV is that as we work with latch groups, in case the latest Departure times of all the latches of a phase converge we move to the next clock phase. That manner is controlled by *A_values_update*, as more iterations are required if the departure times of one phase have not converged (line 7, 19-23). We skip the remaining iterations since there is no need for more iterations if all of the latest departure times of all phases have converged.

On the other hand, the structure of this method, which includes many latch to latch delay propagations, at one level traversal to another in each iteration, allows for quicker results. One important factor is that, regardless of how many latches join at a capture latch, this method would offer the worst arrival time. Finally, each delay propagation from a phase to another covers and gives the entire set of data required by the following phase. The computation of the worst delay value may need additional iterations in the case of the standard ICV, which traverses one latch at a time.

4.2.4 Fast ICV Example

We can move on to an example using our STA engine after studying the Fast ICV algorithm's implementation. Consider the following circuit of figure 4.1, four latches form a cycle where the first one and the third one are clocked on phase one and the second and the fourth one are clocked on phase two. The period of the two clocks is

both $10ns$, and the waveforms are given as $\{0, 5\}$ and $\{5, 10\}$.

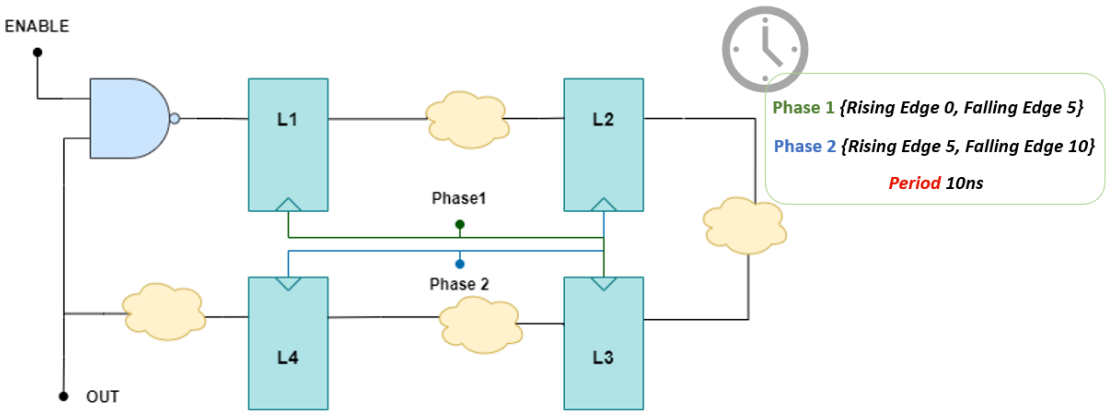

Figure 4.1: Fast ICV Example - Circuit Specification

Figure 4.2 demonstrates the first part of the algorithm each is the initialization.

```
Latch latch_cycle_4stage_2phi/LATCH2, Latest Arrival Time (r) = -1.0000000
Latch latch_cycle_4stage_2phi/LATCH2, Latest Arrival Time (f) = -1.0000000
Latch latch_cycle_4stage_2phi/LATCH2, Latest Departure Time (r) = 5.0000000
Latch latch_cycle_4stage_2phi/LATCH2, Latest Departure Time (f) = 5.0000000
Latch latch_cycle_4stage_2phi/LATCH2, incoming Rise Slew = 0.0000000
Latch latch_cycle_4stage_2phi/LATCH2, incoming Fall Slew = 0.0000000
Gatepin latch_cycle_4stage_2phi/LATCH3/E, Clock Index = 0
Latch latch_cycle_4stage_2phi/LATCH3, Latest Arrival Time (r) = -1.0000000
Latch latch_cycle_4stage_2phi/LATCH3, Latest Arrival Time (f) = -1.0000000
Latch latch_cycle_4stage_2phi/LATCH3, Latest Departure Time (r) = 0.0000000
Latch latch_cycle_4stage_2phi/LATCH3, Latest Departure Time (f) = 0.0000000
Latch latch_cycle_4stage_2phi/LATCH3, incoming Rise Slew = 0.0000000
Latch latch_cycle_4stage_2phi/LATCH3, incoming Fall Slew = 0.0000000
Gatepin latch_cycle_4stage_2phi/LATCH4/E, Clock Index = 1
Latch latch_cycle_4stage_2phi/LATCH4, Latest Arrival Time (r) = -1.0000000
Latch latch_cycle_4stage_2phi/LATCH4, Latest Arrival Time (f) = -1.0000000
Latch latch_cycle_4stage_2phi/LATCH4, Latest Departure Time (r) = 5.0000000
Latch latch_cycle_4stage_2phi/LATCH4, Latest Departure Time (f) = 5.0000000
Latch latch_cycle_4stage_2phi/LATCH4, incoming Rise Slew = 0.0000000
Latch latch_cycle_4stage_2phi/LATCH4, incoming Fall Slew = 0.0000000
Gatepin latch_cycle_4stage_2phi/LATCH1/E, Clock Index = 0
Latch latch_cycle_4stage_2phi/LATCH1, Latest Arrival Time (r) = -1.0000000
Latch latch_cycle_4stage_2phi/LATCH1, Latest Arrival Time (f) = -1.0000000
Latch latch_cycle_4stage_2phi/LATCH1, Latest Departure Time (r) = 0.0000000
Latch latch_cycle_4stage_2phi/LATCH1, Latest Departure Time (f) = 0.0000000
Latch latch_cycle_4stage_2phi/LATCH1, incoming Rise Slew = 0.0000000
Latch latch_cycle_4stage_2phi/LATCH1, incoming Fall Slew = 0.0000000
```

Figure 4.2: Fast ICV Example - Initialization

Notice that the Arrival time and the slew values, are initialized in a negative and at a zero value as the propagation of delay and transition time has not started

yet. All of the Departure times are set to the enable clock edge with respect to the algorithm (lines 1-4).

Figure 4.3 demonstrates the start of the update of the delay values. Notice that in the first place we create the latch group based on the clock phase. Taking as an example Latch 3 of phase one we first update the latest departure time based on the max value of the latest arrival time and the enable clock edge. Keep in mind that we work in the G to Q timing window so we make use of the latch enable clock edge value. As it is clear we take into account the internal delay of the latch (G to Q arc delay) and we propagate at the output pin the appropriate transition time. We update independently the value for both rise and fall delays, and we work individually for delay and transition time. We follow the same procedure for Latch 1. After we update the timing information for the two latches of phase one the appropriate message is reported, and we can move on to the update of the latest arrival time of the capture latches. Figure 4.4 represents the delay and slew propagation at the following phase.

```
Latch latch_cycle_4stage_2phi/LATCH1 phi1 (latch_cycle_4stage_2phi/phi1) added to Latch Group.
Restore latch_cycle_4stage_2phi/LATCH1/D slews, rise slew = 0.000000000000000000, fall slew = 0.000000000000000000
Latch D->Q, latch_cycle_4stage_2phi/LATCH1/D->latch_cycle_4stage_2phi/LATCH1/Q Delays (r, f) = (0.2643581, 0.3709675)
Latch G->Q, latch_cycle_4stage_2phi/LATCH1/E->latch_cycle_4stage_2phi/LATCH1/Q Delays (r, f) = (0.2347870, 0.3651775)
Latch latch_cycle_4stage_2phi/LATCH1 - Arrival Times for Computing Departure (r, f) = -1.0000000, -1.0000000, Enabling Edge = 0.0000000
latch_cycle_4stage_2phi/LATCH1/D -> latch_cycle_4stage_2phi/LATCH1/Q, rise slew = 0.1095111111111111146, fall slew = 0.0829543396226415097
latch_cycle_4stage_2phi/LATCH1/E -> latch_cycle_4stage_2phi/LATCH1/Q, rise slew = 0.1107210062893081715, fall slew = 0.0837444444444444408


                    Latch latch_cycle_4stage_2phi/LATCH1, rise delays/slews from G->Q.
                    Latch latch_cycle_4stage_2phi/LATCH1, Propagated Rise Slew = 0.1107210
                    Latch latch_cycle_4stage_2phi/LATCH1, Latest Departure Time (r) = 0.2347870
                    Latch latch_cycle_4stage_2phi/LATCH1, fall delays/slews from G->Q.
                    Latch latch_cycle_4stage_2phi/LATCH1, Propagated Fall Slew = 0.0837444
                    Latch latch_cycle_4stage_2phi/LATCH1, Latest Departure Time (f) = 0.3651775
                    *** New Departure Time (r, f) = 0.2347870, 0.3651775, Original Departure Time (r, f) = 0.0000000, 0.0000000
                    Latch latch_cycle_4stage_2phi/LATCH1 Output Timing Annotated.
                    Latch latch_cycle_4stage_2phi/LATCH3 Output Timing Annotated.
```

Figure 4.3: Fast ICV Example - Departure Time Update

```
Dumping Longest Path to Gatepin latch_cycle_4stage_2phi/LATCH4/D (unateness checking)
1: latch_cycle_4stage_2phi/LATCH3/Q (Delay = 0.2347870 r) (r: 0.1107210, f: 0.0837444) (Load: 0.003800) (Wireload: 0.000000)
2: latch_cycle_4stage_2phi/BUFFER3/I (Delay = 0.2347870 r) (r: 0.1107210, f: 0.0837444) (Load: 0.000000) (Wireload: 0.000000)
3: latch_cycle_4stage_2phi/BUFFER3/Z (Delay = 0.4088531 r) (r: 0.0281617, f: 0.0221971) (Load: 0.001900) (Wireload: 0.000000)
4: latch_cycle_4stage_2phi/LATCH4/D (Delay = 0.4088531 r) (r: 0.0281617, f: 0.0221971) (Load: 0.000000) (Wireload: 0.000000)
---------------------------------------
Dumping Longest Path to Gatepin latch_cycle_4stage_2phi/LATCH4/D (unateness checking)
1: latch_cycle_4stage_2phi/LATCH3/Q (Delay = 0.3651775 f) (r: 0.1107210, f: 0.0837444) (Load: 0.003800) (Wireload: 0.000000)
2: latch_cycle_4stage_2phi/BUFFER3/I (Delay = 0.3651775 f) (r: 0.1107210, f: 0.0837444) (Load: 0.000000) (Wireload: 0.000000)
3: latch_cycle_4stage_2phi/BUFFER3/Z (Delay = 0.5051337 f) (r: 0.0281617, f: 0.0221971) (Load: 0.001900) (Wireload: 0.000000)
4: latch_cycle_4stage_2phi/LATCH4/D (Delay = 0.5051337 f) (r: 0.0281617, f: 0.0221971) (Load: 0.000000) (Wireload: 0.000000)

Dumping Longest Path to Gatepin latch_cycle_4stage_2phi/LATCH2/D (unateness checking)
1: latch_cycle_4stage_2phi/LATCH1/Q (Delay = 0.2347870 r) (r: 0.1107210, f: 0.0837444) (Load: 0.003800) (Wireload: 0.000000)
2: latch_cycle_4stage_2phi/BUFFER1/I (Delay = 0.2347870 r) (r: 0.1107210, f: 0.0837444) (Load: 0.000000) (Wireload: 0.000000)
3: latch_cycle_4stage_2phi/BUFFER1/Z (Delay = 0.4088531 r) (r: 0.0281617, f: 0.0221971) (Load: 0.001900) (Wireload: 0.000000)
4: latch_cycle_4stage_2phi/LATCH2/D (Delay = 0.4088531 r) (r: 0.0281617, f: 0.0221971) (Load: 0.000000) (Wireload: 0.000000)
---------------------------------------
Dumping Longest Path to Gatepin latch_cycle_4stage_2phi/LATCH2/D (unateness checking)
1: latch_cycle_4stage_2phi/LATCH1/Q (Delay = 0.3651775 f) (r: 0.1107210, f: 0.0837444) (Load: 0.003800) (Wireload: 0.000000)
2: latch_cycle_4stage_2phi/BUFFER1/I (Delay = 0.3651775 f) (r: 0.1107210, f: 0.0837444) (Load: 0.000000) (Wireload: 0.000000)
3: latch_cycle_4stage_2phi/BUFFER1/Z (Delay = 0.5051337 f) (r: 0.0281617, f: 0.0221971) (Load: 0.001900) (Wireload: 0.000000)
4: latch_cycle_4stage_2phi/LATCH2/D (Delay = 0.5051337 f) (r: 0.0281617, f: 0.0221971) (Load: 0.000000) (Wireload: 0.000000)
```

Figure 4.4: Fast ICV Example - Delay and Slew Propagation

One incremental update run is performed for all of the capture latches of the next clock phase. Again for each launching latch propagation would start from both rise and fall values. Our STA engine reports for the worst timing path to each input data gate pin of each one of the capture latches. In figure 4.5 we get to obtain the latest arrival time of the capture latches. This value has been calculated based on the maximum combinational delay from the output pin of the launching latch to the input pin of the capture latch. Both the launching latch which is indicated as a starting point and the combinational delay is reported. For the sake of simplicity, we'll bypass the next delay propagation steps because the process is the same for latches clocked on phase two. Latch 2 and *Latch*4 would form a group, and their departure time would be updated accordingly. Following that, an incremental run would be performed to update the arrival times at phase one's latches.

```
Latch latch_cycle_4stage_2phi/LATCH4, Latest Arrival Time (r) = 0.4088531
Latch latch_cycle_4stage_2phi/LATCH4, Latest Arrival Time (f) = 0.5051337
Latch latch_cycle_4stage_2phi/LATCH4, incoming Rise Slew = 0.0281617
Latch latch_cycle_4stage_2phi/LATCH4, incoming Fall Slew = 0.0221971
Launch Latch latch_cycle_4stage_2phi/LATCH3, Maximum Delay (r) at Gatepin latch_cycle_4stage_2phi/LATCH4/D = 0.1740661
Launch Latch latch_cycle_4stage_2phi/LATCH3, Maximum Delay (f) at Gatepin latch_cycle_4stage_2phi/LATCH4/D = 0.1399562


Latch latch_cycle_4stage_2phi/LATCH2, Latest Arrival Time (r) = 0.4088531
Latch latch_cycle_4stage_2phi/LATCH2, Latest Arrival Time (f) = 0.5051337
Latch latch_cycle_4stage_2phi/LATCH2, incoming Rise Slew = 0.0281617
Latch latch_cycle_4stage_2phi/LATCH2, incoming Fall Slew = 0.0221971
Launch Latch latch_cycle_4stage_2phi/LATCH1, Maximum Delay (r) at Gatepin latch_cycle_4stage_2phi/LATCH2/D = 0.1740661
Launch Latch latch_cycle_4stage_2phi/LATCH1, Maximum Delay (f) at Gatepin latch_cycle_4stage_2phi/LATCH2/D = 0.1399562
```

Figure 4.5: Fast ICV Example - Arrival Time Update

In figure 4.6 we proceed to the next iteration.

```
Latch latch_cycle_4stage_2phi/LATCH1, rise delays/slews from G->Q.
Latch latch_cycle_4stage_2phi/LATCH1, Propagated Rise Slew = 0.1107210
Latch latch_cycle_4stage_2phi/LATCH1, Latest Departure Time (r) = 0.2347870
Latch latch_cycle_4stage_2phi/LATCH1, fall delays/slews from G->Q.
Latch latch_cycle_4stage_2phi/LATCH1, Propagated Fall Slew = 0.0837444
Latch latch_cycle_4stage_2phi/LATCH1, Latest Departure Time (f) = 0.3651775
*** New Departure Time (r, f) = 0.2347870, 0.3651775, Original Departure Time (r, f) = 0.2347870, 0.3651775

Latch latch_cycle_4stage_2phi/LATCH3, rise delays/slews from G->Q.
Latch latch_cycle_4stage_2phi/LATCH3, Propagated Rise Slew = 0.1107210
Latch latch_cycle_4stage_2phi/LATCH3, Latest Departure Time (r) = 0.2347870
Latch latch_cycle_4stage_2phi/LATCH3, fall delays/slews from G->Q.
Latch latch_cycle_4stage_2phi/LATCH3, Propagated Fall Slew = 0.0837444
Latch latch_cycle_4stage_2phi/LATCH3, Latest Departure Time (f) = 0.3651775
*** New Departure Time (r, f) = 0.2347870, 0.3651775, Original Departure Time (r, f) = 0.2347870, 0.3651775

Latch latch_cycle_4stage_2phi/LATCH2, rise delays/slews from G->Q.
Latch latch_cycle_4stage_2phi/LATCH2, Propagated Rise Slew = 0.1107210
Latch latch_cycle_4stage_2phi/LATCH2, Latest Departure Time (r) = 5.2347870
Latch latch_cycle_4stage_2phi/LATCH2, fall delays/slews from G->Q.
Latch latch_cycle_4stage_2phi/LATCH2, Propagated Fall Slew = 0.0837444
Latch latch_cycle_4stage_2phi/LATCH2, Latest Departure Time (f) = 5.3651775
*** New Departure Time (r, f) = 5.2347870, 5.3651775, Original Departure Time (r, f) = 5.2347870, 5.3651775


Latch latch_cycle_4stage_2phi/LATCH4, rise delays/slews from G->Q.
Latch latch_cycle_4stage_2phi/LATCH4, Propagated Rise Slew = 0.0920631
Latch latch_cycle_4stage_2phi/LATCH4, Latest Departure Time (r) = 5.2115092
Latch latch_cycle_4stage_2phi/LATCH4, fall delays/slews from G->Q.
Latch latch_cycle_4stage_2phi/LATCH4, Propagated Fall Slew = 0.0753556
Latch latch_cycle_4stage_2phi/LATCH4, Latest Departure Time (f) = 5.3484553
*** New Departure Time (r, f) = 5.2115092, 5.3484553, Original Departure Time (r, f) = 5.2115092, 5.3484553
```

Figure 4.6: Fast ICV Example - Next Iteration

The algorithm groups the latches depending on the phase they are clocked on and updates their departure time in the following iteration. Taking a quick look at these numbers, we can see that the latches' latest departure times have converged in both phases. As we saw in the previous section, Fast ICV would not update the latest arrival time in subsequent iterations in any incremental run. The departure time values have converged in this example, and our algorithm has not identified a setup violation, suggesting that we may proceed to the algorithm's final output logs.

Looking at figures 4.7 and 4.8, we can see a detailed output log that includes information on the latest values as well as the worst path analysis. Note that our tool reports the worst negative slack and the delay values and worst path analysis is performed for all of the circuit latches. In our case we for simplicity reasons we present only Latch 4 as it is the latch we the worst negative slack.

Figure 4.7 is the first thing we'll look at. We may simply observe that this collection of data comprises no more than all of the relevant data found in the latch graph for each of the latch nodes. We obtain the latest departure and arrival times, as well as the enabling clock edge, for each of the nodes. The edges-connections to the corresponding nodes are then obtained for each node. Note that for each latch to latch connection, the weight or combination delay for both rise and fall values is provided for each edge.

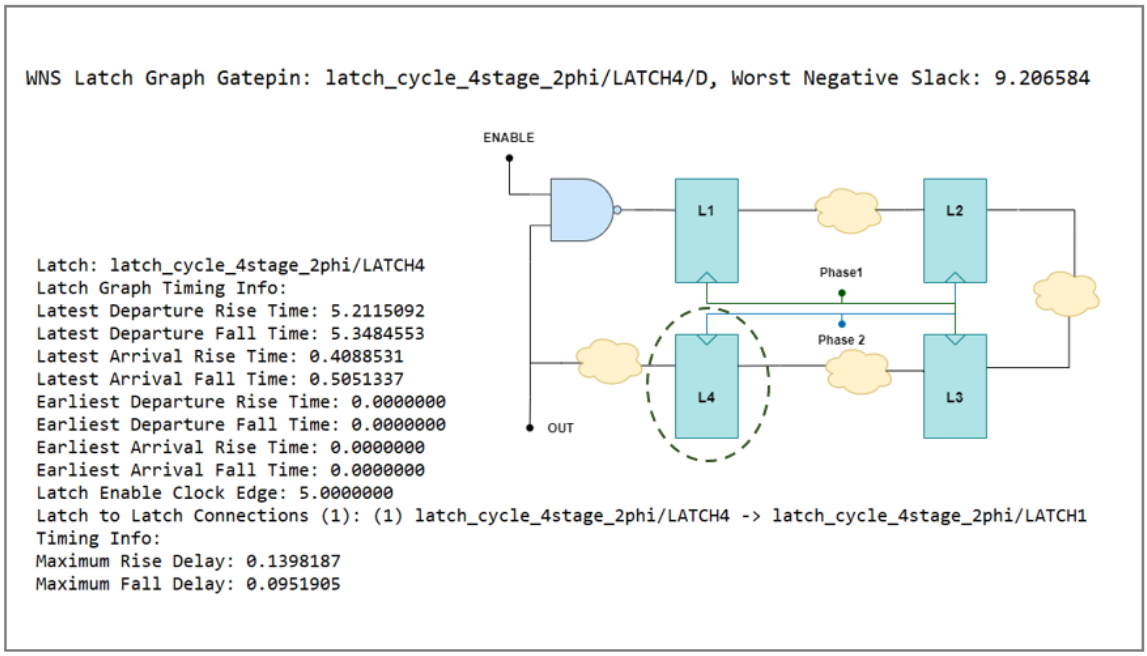

Figure 4.7: Fast ICV Example - Delay Values

We get to see the worst path analysis for each of the circuit latches in the final step. In this analysis, we find the worst slack for each capture latch for rising and falling starting launching points. The launch latch marks the beginning point, and as we go through the combination delay, we finally arrive at the proper information at the capture latch's input pin.

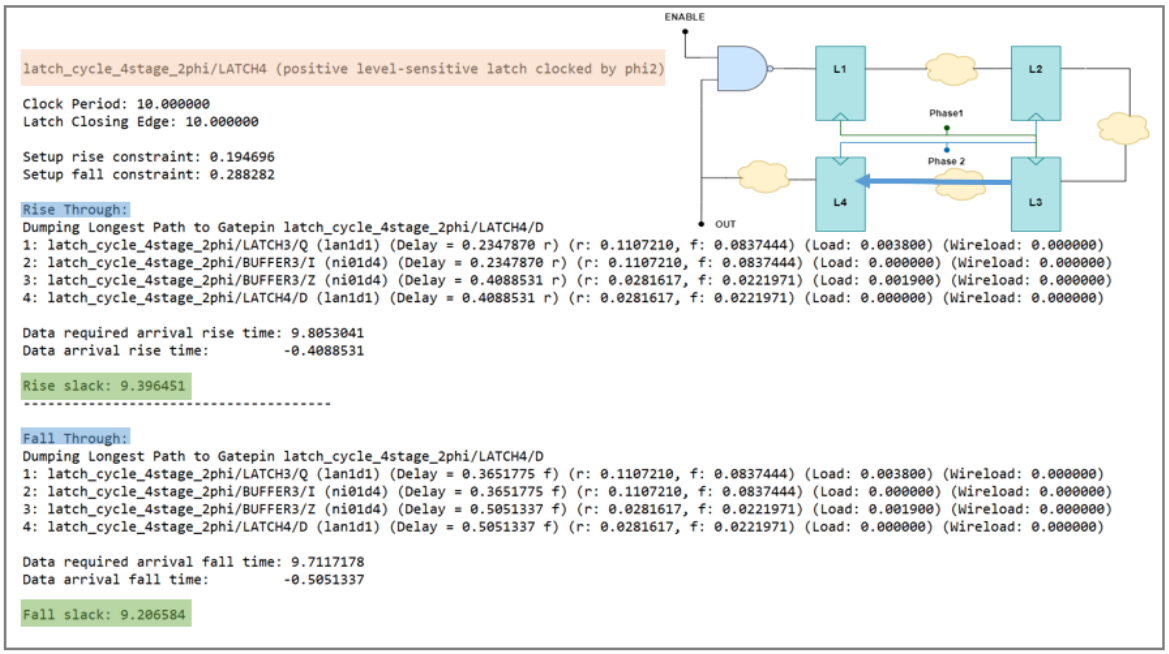

Figure 4.8: Fast ICV Example - Worst Paths

4.3 Bundled-Data Design

As already stated in previous chapters, the level of operation for traditional STA engines is directed acyclic graphs. Gate pins are the nodes in this example, timing arcs are the graph's edges, and sequential elements are the analyses' boundary points. More precisely, the STA engine may deactivate combinational feedback loops in a design to decrease the influence of the violation-timing error on circuit timing validation. The timing arc or disabled data point may have an incorrect transition time when this cycle cutting approach is utilized, resulting in erroneous slew propagation in different timing arcs and inaccurate slew computation analysis across cycles.

Bundled-Data Design is a good example of a latch-based design that would be badly affected by cycle cutting in timing analysis. Bundled Data is made up of asynchronous cyclic controllers and a synchronous latch data path, as previously indicated. Cutting the cycles in the asynchronous section would result in erroneous enable signal propagation in the data path, which would be devastating. In more detail through this methodology, asynchronous controllers generate the clock signals for the synchronous data paths, which indicates that Asynchronous Static Timing Analysis is required for the asynchronous part and STA for the synchronous data path. So a hybrid methodology is needed in order to perform timing analysis in that complex form of the circuit without additional timing errors. Before we proceed, we must refresh our knowledge of all those structural pieces that make up a bundled data design. We may take a deeper look at the exact essential structures of such a circuit in the figure below.

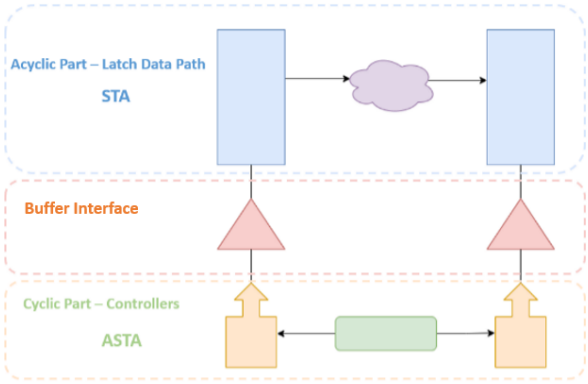

Figure 4.9: Bundled Data Design Structural Parts

As it is clear from the figure above, we would need to follow a separate timing methodology for the asynchronous controllers' portion and a different timing approach for the synchronous data path. The ASTA engine is also implemented and integrated into the same tool (ASP) as this study. Regarding the ASTA analysis, we are going to take a brief look at the analysis steps:

- **Cyclic Equilibrium Slew Computation:** When dealing with combinational loops in cyclic circuits, a worst or best slew must be established over cycles. To accomplish so, we use algorithm 2 described above, which runs through these cycles, annotating a worst or best case slew for each circuit component until it converges.
- **Event Timing Graph (ETG) Construction:** Following the cyclic slew computation, the event graph is built or the user provides it to the engine. ETG is a term used to define the timing connection between the events in it. The transitions in the event model are then mapped to netlist module ports.
- **ETG T2T Delay Derivation in Netlist:** It is worth mentioning that each arc in ETG demonstrates a timing arc. So at that point, we perform delay annotation to netlist paths based on ETG Event to event arcs. The computation of the worst and best path is achieved through STA.
- **ETG Period and Critical Cycle(s) Computation:** The calculation of the critical cycle is the final stage in this flow. The critical cycle is the component of timing analysis that will have the most impact on the circuit since the critical cycle's period will directly affect the circuit's period.

When it comes to the synchronous portion, the controllers give the clock signals, which Fast ICV may use to execute Latch STA at the latch data path since the clock periods and waveforms are defined. Most of the controllers (if not all) operate on the worst-period, but for each sequential stage a different arbitrary clock phase is produced. Regarding the delay computation part, the process is already defined for the latch data path. Fast ICV would come up with its iterative procedure to give the answers regarding the synchronous portion timing analysis.

However, we must keep in mind that this communication between the ASTA tool and Fast ICV, in which the first one propagates the enable signals from the

asynchronous portion to the synchronous portion must be performed under specific circumstances where several conditions must not be violated. In the following steps, we are going to introduce some timing constraints so the proper timing analysis for a bundled data design is achieved.

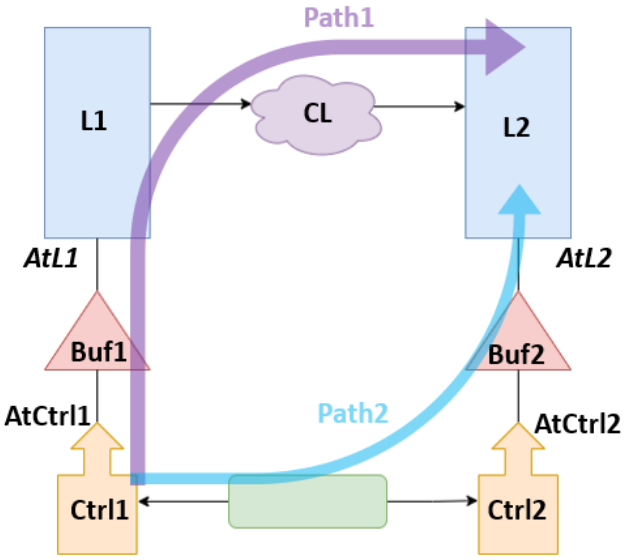

Figure 4.10: Bundled Data Design Timing Constraints

*Ctrl*1 and *Ctrl*2 will give produced clocks to *Latch*1 and *Latch*2 in the example above. We must ensure that clock signal propagation proceeds without causing any setup violations in any of the latches during data propagation. We must confirm that *Path*1 will be slower than *Path*2 since the two latches create a pair of launching and a capturing latch, and the first one launches data to the second one. In the first place, the following relative timing constraint must be taken into account:

$$At_{L1} + \Delta_{CL} + s_{L2} < At_{L2}$$

At stands for the arrival time at the latches, Δ_{CL} for the combinational delay at the data path and *s* for the library setup time. The key factor here is to understand that the arrival time at the latches is an outcome of the asynchronous part, so if we want to dig deeper into the above equation we can conclude to the following equation:

$$(At_{CTRL1} + \Delta_{BUF1}) + \Delta_{CL} + s_{L2} < (At_{CTRL2} + \Delta_{BUF2})$$

Both STA and ASTA must contribute to the proper circuit validation so the circuit can work correctly.

Furthermore, when we generate a clock in this manner, we must ensure that the generated clock has the necessary properties for a sequential element to function under it. It is necessary to guarantee that every latch receives a clock pulse with a width larger than the specified minimum pulse width. The "minimum pulse width check" ensures that the pulse width is kept to a minimum. By default, all latches in a design should have a minimum pulse width set in the liberty file.

Chapter 5

Experimental Methodology

We describe the whole Latch STA flow in this chapter, which can be utilized for timing validation in Latch-based designs. At this part of this work we also examine the methodology we followed in order to validate the operations of the Latch STA procedure, through gate-level simulation and correlate its functionality with the latch timing verification methodology that is provided by the industrial tool.

5.1 Latch STA Flow

Figure 5.1 demonstrates the whole Latch STA process. The following process has been applied to produce timing reports at both STA tools, at our tool, and at the industrial tool.

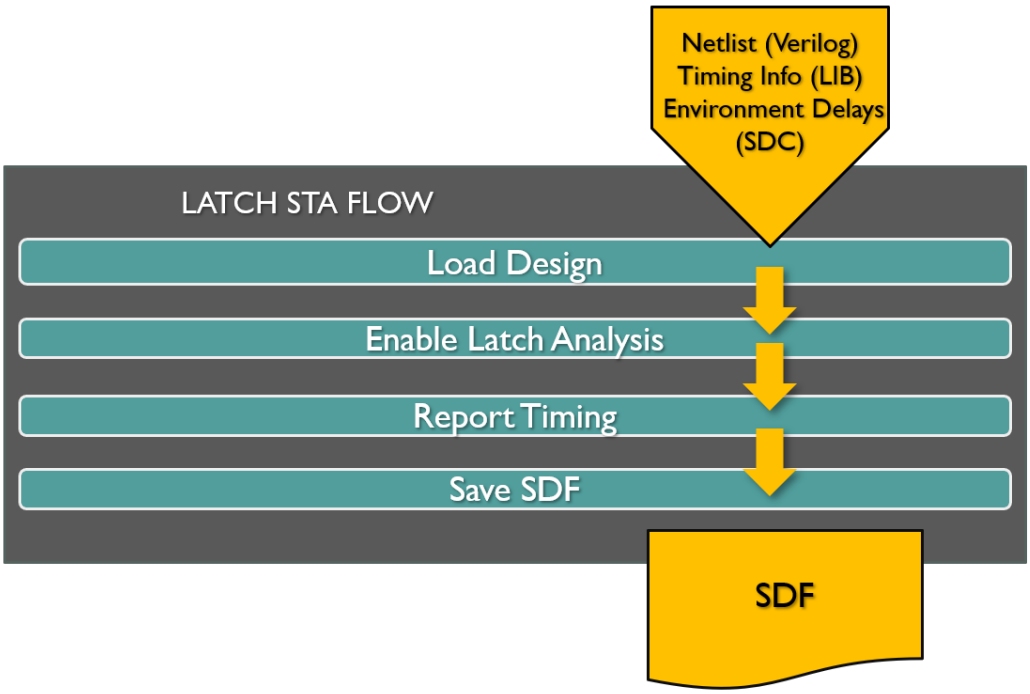

Figure 5.1: Complete Latch STA Flow

Coming up with the first step, the user must provide at the engine the appropriate files that demonstrate the circuit, and under which timing technology the design was constructed, to achieve loading the design successfully. So the user must feed the engine with a netlist, the technology timing library, and specify the environment delays through the SDC format. Looking into a brief description of the above:

- **Netlist**: A netlist is a list of electronic components in a circuit and the nodes to which they are connected in its most basic form. This file is given in Verilog (.v) format.
- **Technology Timing Library**: The technology timing library is represented through the Liberty file format (.lib). The (.lib) file is an ASCII representation of the timing and power parameters associated with any cell in particular semiconductor technology.
- **Synopsis Design Constraint (SDC)**: SDC is a format that specifies the design's environment-intent including the timing and the power of the design. SDC format consists basically of TCL commands. [9]

The following step in both processes is to enable their settings so that the Latch Static Timing analysis may begin. To do Latch STA analysis on the industrial tool, a particular variable must be set. [9] On the other hand, after specifying the appropriate periods and waveforms for the different clock phases of the latches, we can use the Fast ICV algorithm to compute the latch delay values at our engine. Concerning the above, a specific variable must be set in our tool when the clock phases of the latches overlap, implying that latches of various clock phases are transparent at the same time, and the circuit is cyclic. Before the ICV algorithm runs, an equilibrium cyclic slew is computed for all circuit gate pins when this variable is enabled. The preceding approach takes into consideration the presence of cycles in the circuit and computes a convergent equilibrium transition time for all circuit gate pins using an iterative technique through the cycles. [2] At this point, we must note that the industrial tool does not take into account the above slew computation. When at least one combinational loop is found in the circuit, most industrial STA engines execute cycle cutting. As a result, the disabled data to output latch arc will have wrong slew and will have no effect on propagating the correct slew at the following pins. This results in erroneous timing outcomes.

Finally, when the timing tools have completed their computations, we look at the latch delay values from the tools' output logs. The extraction of the SDF file is another operation that should be considered. This file will help us in moving on to the next phase, the SDF simulation flow, which will give us a clearer validation of the two-timing engines' outputs. Let's take a closer look at what an SDF file is:

- **SDF**: The Standard Delay Format (SDF) is a timing data description and analysis format that may be utilized at any stage of the electrical design process. It bridges the gap between dynamic and static timing analysis. [9]

The flow we maintained in order to perform SDF simulation at our test cases will be examined in-depth in the next section.

5.2 SDF Simulation Flow

The approach we used to accomplish the gate-level SDF simulation of latch-based designs is now outlined in detail. The goal of the gate-level simulation is to compare our Latch STA engine to the industry tool golden-reference model and to confirm that our Latch STA approach is successful by using the SDF simulation procedure to validate our results.

The entire SDF simulation procedure is depicted in Figure 5.2. As previously stated, the following procedure is used to generate timing data in order to correlate and verify the operations of Latch STA engines.

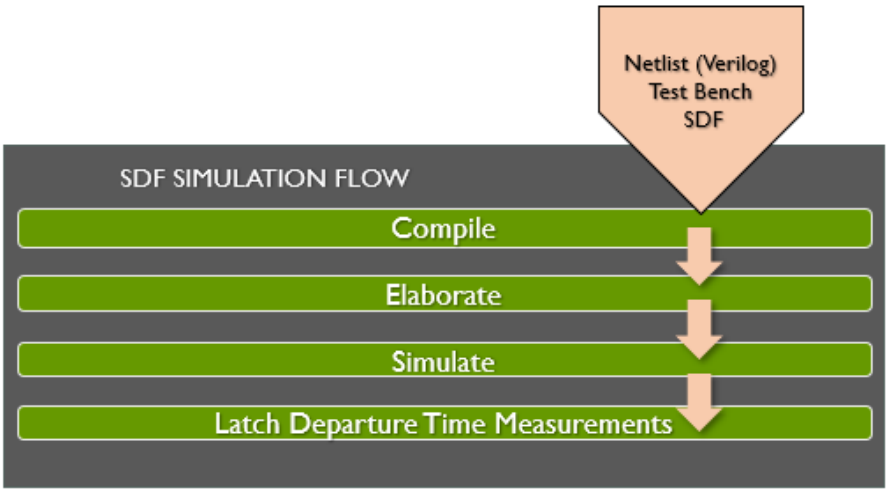

Figure 5.2: SDF Simulation Flow

As previously stated, SDF Simulation serves as a link between static and dynamic timing analysis. As can be seen at this stage, the extraction of the SDF file and simulation using it were done individually for both engines in order to correlate and validate their results. Looking at the first step of the flow, in order to simulate our gate-level test cases and annotate the delay values and the timing checks of the SDF file we constructed different Verilog test benches for each one of them. It's worth noting that each of the test benches follows a semi-automated method that generates results without requiring the user to look at the signal waveforms in the majority of circumstances.

The steps-utilities that we performed using the simulation program to arrive at the final results are described in the following lines. At the compilation stage, we must compile the netlist Verilog file, the test bench, and the technology file. Through this step, the compilation tool performs syntax verification and static semantics checks at the Verilog code. After that comes the elaboration step. The responsible tool for this task elaborates the design hierarchy and determines signal connectivity.The back annotation of the delay values provided in the SDF file is also done at this point. Through the test bench, the simulator reads the SDF file automatically. We gather and examine the simulation flow's outcomes in the last stage. We may infer the outcomes of the simulation flow by looking at the waveforms or

looking at the terminal at the data supplied by the semi-automated measurements by the test bench.

5.3 Bundled Data Design Experimental Methodology

Moving forward, we'll go over the complete timing analysis flow for Bundled Data Design. As previously said, we would follow a separate flow for the asynchronous part, and because the clock signals are provided by the controllers, Fast ICV may take these clock signals and execute Latch STA at the latch data path. The ASTA engine is also implemented and integrated into the same tool (ASP) as this work, and related publications by current and former engineers who contributed to the tool's development detail its development and scientific achievements. Figure 5.3 demonstrates the complete ASTA flow [2, 7, 13], the files needed in order to perform ASTA are the same with the STA flow, meaning the Verilog Netlist, the .lef, .lib and the SDC files. Looking into a brief explanation of the bellow demonstrated steps:

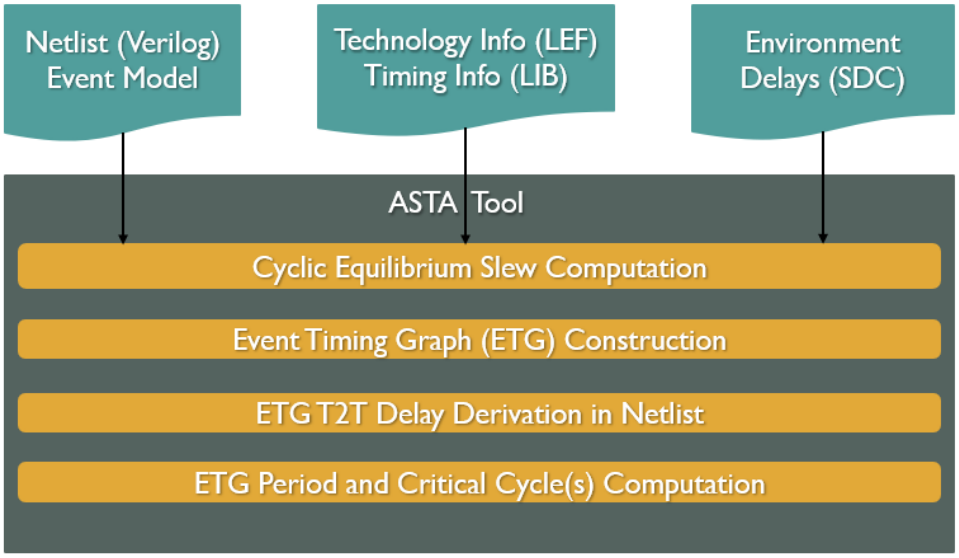

Figure 5.3: ASTA Flow

- **Cyclic Equilibrium Slew Computation:** A worst or best slew must be determined across cycles when dealing with combinational loops in cyclic circuits. To do this, an iterative algorithm performs through these cycles, annotating a worst or best case slew for each circuit component till it converges.
- **Event Timing Graph (ETG) Construction:** After cyclic slew computation, the construction of the event graph takes place or the user provides it to the engine. Basically, ETG is a concept that describes the timing relationship between the events in it. After that, event model's transitions are mapped on netlist module ports.
- **ETG T2T Delay Derivation in Netlist:** It is worth mentioning that each arc in ETG demonstrates a timing arc. So at that point, we perform delay annotation

to netlist paths based on ETG Event to event arcs. The computation of the worst and best path is achieved through STA.

- **ETG Period and Critical Cycle(s) Computation:** The final step of this flow comes up with the computation of the critical cycle. The critical cycle is the component of timing analysis that is going to affect the circuit the most, as the critical cycle's period will affect directly the period of the whole circuit.

The two engines, namely the ASTA engine and the last STA engine, are ready to interact as the critical cycle is computed. Since the asynchronous controllers generate the clocks for the Latch data path, the method is already specified as long as the enable signals for the synchronous part are available. Fast ICV adapts instantly to the ASTA flow and begins operating as soon as the ASTA part generates clocks' periods.

We may move on to the following chapter after having a close look at the whole Latch static and dynamic experimental flow. We'll summarize and examine the abovementioned in depth in the following chapter as we collect the results of the latch delay values and the observations we made.

Chapter 6

Experimental Results

A particular set of test cases has been created to validate the above methodology's operations so that we can examine in detail the complete design cases that our methodology handles effectively and compare its results and performance to the state-of-the-art STA engine and the existing ICV (Iterative Constraint Verification) algorithm. The netlist technology is 0.25*um* IHP. We should point out that we focus on the nature of the circuit in terms of how the latches' clock phases are represented, rather than the number of elements in the circuit because we want to look at the correct circuit handling, timing propagation, and timing reports in Acyclic, Cyclic, and the exceptional case of the Bundle Data design. The tables below show the entire range of our showcases. Linear latch pipelines and classic ring oscillator cyclic circuits are examples of Acyclic and Cyclic circuits having overlapping and non-overlapping clocks. The waveforms of the clocks indicate which timing windows will be active throughout the delay propagation, hence the above clock specification is critical.

6.1 Latch-Based Design Experimental Results

Tables 6.1 and 6.2 show the results of two acylcic testcases, with latches clocked on non overlapping and overlapping clocks.

Testcase: latch_pipeline_4stage_and_2phi								
Latches: 4	**Clock Period (ns): 10**			**Clocks Waveforms: {0 5} {5 10}**	**Number of Clocks: 2**			
	Latest Arrival Rise Time				**Latest Arrival Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1								
Latch2	0.459586	0.4596	0.459586	0.4596	0.291028	0.2910	0.291028	0.2910
Latch3	5.459586	5.4596	5.459586	5.4596	5.291028	5.2910	5.291028	5.2910
Latch4	0.488274	0.4883	0.488274	0.4883	0.3067	0.3067	0.3067	0.3067
	Clock Period (ns): 10			**Clocks Waveforms: {0 5} {1 6}**	**Number of Clocks: 2**			
	Latest Arrival Rise Time				**Latest Arrival Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1								
Latch2	0.459586	0.4596	0.459586	0.4596	0.291028	0.2910	0.291028	0.2910
Latch3	1.459586	1.4596	1.459586	1.4596	1.291028	1.2910	1.291028	1.2910
Latch4	1.80052	1.8005	1.80052	1.8005	1.821163	1.8212	1.821163	1.8212

Table 6.1: Four Stage Latch Pipeline Case

Testcase: latch_unconnected_pipelines								
Latches: 4	**Clock Period (ns): 10**		**Clocks Waveforms: {0 5} {5 10}**		**Number of Clocks: 2**			
	Latest Arrival Rise Time				**Latest Arrival Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1								
Latch2								
Latch3	0.45941	0.4594	0.45941	0.4594	0.273704	0.2737	0.273704	0.2737
Latch4	5.45941	5.4594	5.45941	5.4594	5.273704	5.2737	5.273704	5.2737
	Clock Period (ns): 10		**Clocks Waveforms: {0 5} {1 6}**		**Number of Clocks: 2**			
	Latest Arrival Rise Time				**Latest Arrival Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1								
Latch2								
Latch3	0.45941	0.4594	0.45941	0.4594	0.273704	0.2737	0.273704	0.2737
Latch4	1.45941	1.4594	1.45941	1.4594	1.273704	1.2737	1.273704	1.2737

Table 6.2: Two Latch independent Pipelines Case

Table 6.3 demonstrates the results of cyclic Latch circuits, where latches phases are clocked on non-overlapping clocks.

Testcase: latch_cycle_5stages_ring_enb								
Latches: 2	**Clock Period (ns): 10**		**Clocks Waveforms: {0 5} {5 10}**			**Number of Clocks: 2**		
	Latest Arrival Rise Time				**Latest Arrival Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1	5.50679	5.5068	5.50679	5.5068	5.63473	5.6347	5.63473	5.6347
Latch2	0.61181	0.6118	0.61181	0.6118	0.42944	0.4294	0.42944	0.4294
Testcase: latch_cycle_4stage_2phi								
Latches: 4	**Clock Period (ns): 10**		**Clocks Waveforms: {0 5} {5 10}**			**Number of Clocks: 2**		
	Latest Arrival Rise Time				**Latest Arrival Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1	5.48827	5.4883	5.48827	5.4883	5.30670	5.3067	5.30670	5.3067
Latch2	0.40885	0.4089	0.40885	0.4089	0.50513	0.5051	0.50513	0.5051
Latch3	5.40885	5.4089	5.40885	5.4089	5.50513	5.5051	5.50513	5.5051
Latch4	0.40885	0.4089	0.40885	0.4089	0.50513	0.5051	0.50513	0.5051
Testcase: latch_join_one_stage								
Latches: 4	**Clock Period (ns): 10**		**Clocks Waveforms: {0 5} {5 10}**			**Number of Clocks: 2**		
	Latest Arrival Rise Time				**Latest Arrival Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1	5.48827	5.4883	5.48827	5.4883	5.30670	5.3067	5.30670	5.3067
Latch2	5.48827	5.4883	5.48827	5.4883	5.30670	5.3067	5.30670	5.3067
Latch3	0.79865	0.7986	0.79865	0.7986	0.63851	0.6385	0.63851	0.6385
Latch4	0.79865	0.7986	0.79865	0.7986	0.63851	0.6385	0.63851	0.6385
Testcase: latch_join_two_stages								
Latches: 4	**Clock Period (ns): 10**		**Clocks Waveforms: {0 5} {5 10}**			**Number of Clocks: 2**		
	Latest Arrival Rise Time				**Latest Arrival Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1	5.51750	5.5175	5.51750	5.5175	5.68384	5.6838	5.68384	5.6838
Latch2	5.51750	5.5175	5.51750	5.5175	5.68384	5.6838	5.68384	5.6838
Latch3	0.79865	0.7986	0.79865	0.7986	0.63851	0.6385	0.63851	0.6385
Latch4	0.79865	0.7986	0.79865	0.7986	0.63851	0.6385	0.63851	0.6385

Table 6.3: Cyclic Latch Circuits Non-Overlapping Clocks

All of the above designs are simple cyclic showcases that helped us validate the operations of our methodology. In more detail, a ring oscillator circuit with two latches, a latch cycle with four latches, and two cases where more than one cycle is composed in the circuit are demonstrated at the above table. Regarding all cases, latches are clocked on two different phases alternately.

Because the clocks do not overlap in this scenario, each phase will launch and capture data without latches from different phases being transparent at the same time. Delay propagation would proceed from the Enable to Output timing window rather than the Data to Output timing window as a result of this. So the point in

that, is that even though cycle cutting occurs, Enable to Output arc is dominant so the Data to Output arc does not affect slew propagation.

The lines that follow focus on cyclic latch-based designs in which the clocks of different phases overlap. The 6.4 table has an overview of all of the results; however, we will go through certain of them in further depth.

Testcase: three_phase_latch_cycle								
Latches: 3	**Clock Period (ns): 20**			**Clocks Waveforms: {0 10} {5 12} {14 19}**				Number of Clocks: 3
	Latest Departure Rise Time				**Latest Departure Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1	0.21461	0.2146	0.21461	0.2146	0.35068	0.3507	0.35068	0.3507
Latch2	5.21461	5.2146	5.21461	5.2146	5.35068	5.3507	5.35068	5.3507
Latch3	14.23479	14.2348	14.23479	14.2348	14.36518	14.3652	14.36518	14.3652
Testcase: three_phase_latch_cycle_per_5								
Latches: 3	**Clock Period (ns): 20**			**Clocks Waveforms: {0 10} {5 15} {10 20}**				Number of Clocks: 3
	Latest Departure Rise Time				**Latest Departure Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1	0.21461	0.2146	0.21461	0.2146	0.35068	0.3507	0.35068	0.3507
Latch2	5.21461	5.2146	5.21461	5.2146	5.35068	5.3507	5.35068	5.3507
Latch3	10.21461	10.2146	10.21461	10.2146	10.35068	10.3507	10.35068	10.3507
Testcase: two_phase_latch_cycle_d_to_q								
Latches: 4	**Clock Period (ns): 20**			**Clocks Waveforms: {0 17} {14 20}**				Number of Clocks: 2
	Latest Departure Rise Time				**Latest Departure Fall Time**			
	Fast ICV	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**	**Fast ICV**	**SDF Sim**	**Industrial Tool**	**Industrial Tool Sim**
Latch1	Setup	Setup	17.0576	0.2639	Setup	Setup	17.0761	0.3704
Latch2	Violation	Violation	18.2406	0.2696	Violation	Violation	19.4531	0.3872
Latch3	At	At	17.8765	1.3209	At	At	17.4801	3.5396
Latch4	LATCH3	LATCH3	17.8015	13.4149	LATCH3	LATCH3	18.3601	13.2569

Table 6.4: Overlapping Clocks Cases

In cases where the clocks have an overlap, we notice two scenarios. First, suppose as a reference the second test case in table 6.4. In that case, the waveforms have the following view as is demonstrated in the following picture 6.2. All clocks have a period of *20ns* and waveform overlap per *5ns*.

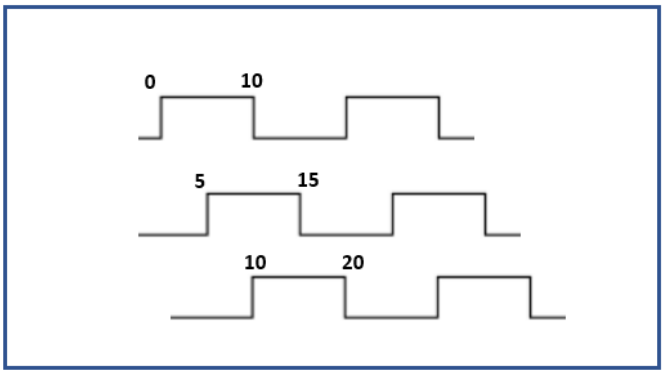

Figure 6.1: Latch Enable per 5ns

As can be seen in the snapshot, two of them have overlap each time, but the third will not have overlap with the first. *Phase one* and *Phase two* overlap, while *Phase three* does not overlap with *Phase one*. This situation will go on endlessly. So despite the fact that the clocks overlap, Enable to Output will be the dominant way of data propagation. In that case cycle cutting would no affect slew and delay, even if it applied arbitrary on among overlapping phases by the industrial tool.

Figure 6.2 demonstrates the phases as they were given at the latches in the third testcase of the table 6.4. In that cases clocks overlap in such a way that the Data to Output timing window is enabled, meaning that delay and slew computation will proceed from that window.

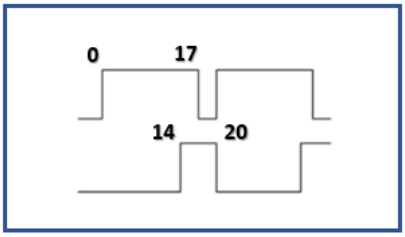

Figure 6.2: Overlapping Phases

Given the above clock waveforms, we start examining the analysis performed on the design demonstrated in figure 6.3. both at our tool and the industrial tool. With the appropriate command, we get to know that a loop breaker is set at the input pin of Latch 3, by the industrial timer.

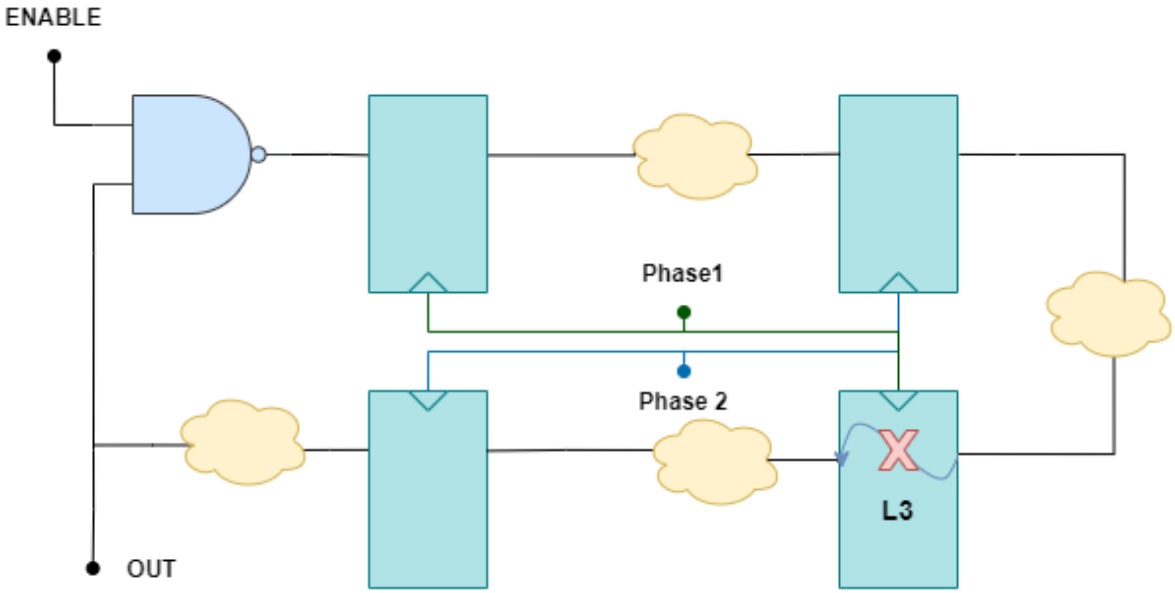

Figure 6.3: Cycle Cutting Testcase

Looking in table 6.4 as demonstrated for Latch3 a setup violation is detected by our methodology which is not detected by the industrial engine. To validate the above result mismatch, we proceed to the SDF simulation. As it can be observed again by table 6.4, in our case SDF timing check annotation indicates that a setup violation occurs in Latch3, however simulation with the SDF from the industrial tool violation detection is absent. By setting a loop breaker at Latch3 an unrealistic value was calculated for both delay and transition time. This unrealistic small value in slew drives into an erroneous delay annotation in which the setup violation was never detected. It is worth mentioning that SPICE measurements were performed at

the circuit under test for further validation. The average relative slew error is 0.3% and the feedback arc slew relative error was 1.82%.

6.2 ICV vs Fast ICV Iterations

Figure 6.4 shows a comparison of the total number of iterations required by ICV versus Fast ICV in a set of test scenarios where the latest departure times converge. Fast ICV, which performs iterations based on clock phases and checks each time if the latest departure time converges before doing all iterations, requires 2.75 less iterations than regular ICV in a set of test cases with a range of clock phases and circuit types.

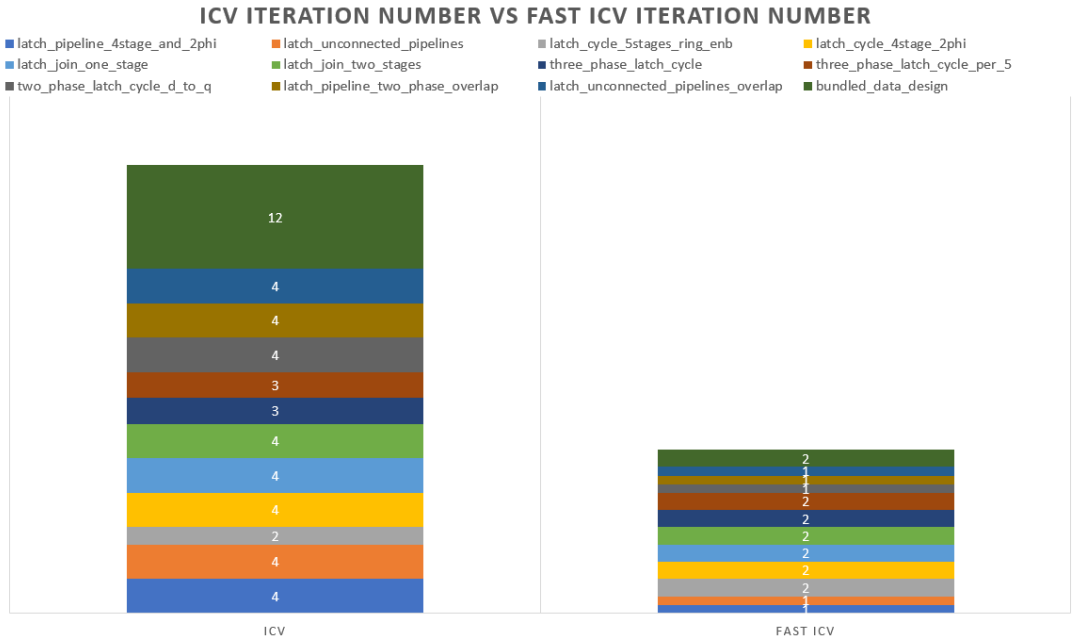

Figure 6.4: ICV vs ICV Fast Iterations

6.3 Industrial Tool vs Fast ICV Slack Computation

The comparison of slack computation between the industrial tool and Fast ICV is shown in Figure 6.5. Refreshing our memory, the industrial tool employs two methods to verify for setup, at the opening and closing edges, as stated in the background part. Slack computed at the opening edge is tighter than slack computed at the closing edge, as seen in the graph. This might lead an inexperienced designer to conclude that there is no room for additional time improvements.

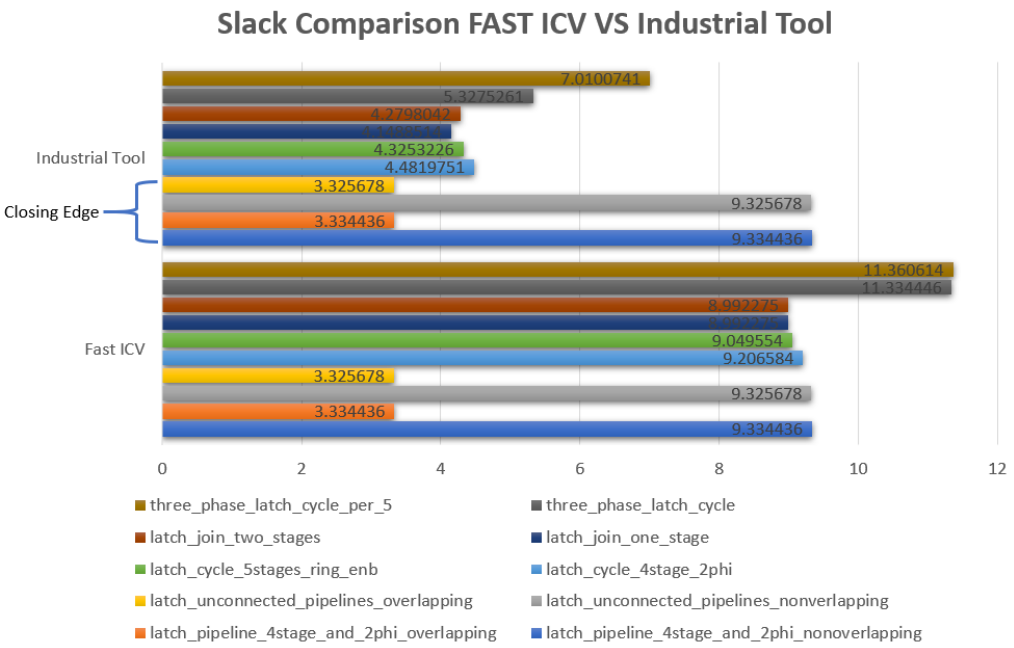

Figure 6.5: Industrial Tool vs ICV Fast Slack Computation

6.4 Bundled-Data Design Experimental Results

The Bundled-Data design on which we will do time analysis is shown in Figure 6.6. As it is clear this design consists of two asynchronous controllers and a cyclic synchronous latch data path. The enable signals would be provided by the two controllers to the respective latches. Only one latch receives the clock signal from *Ctrl*2, while the rest of the latches receive it from *Ctrl*1. The data path consists of twelve latches in total, where some of the latches of phase one join to the latch of phase two, and the latch of phase two fan outs at some latches of phase one, indicating that a cycle is created.

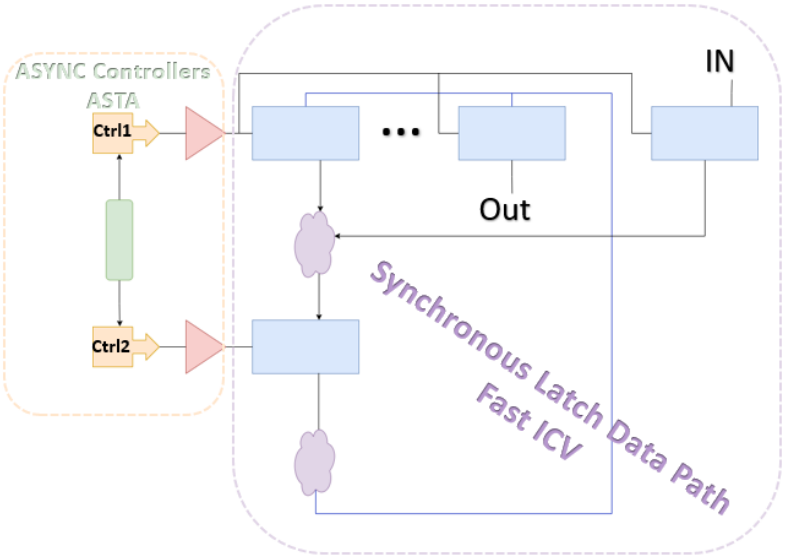

Figure 6.6: Bundled-Data Experiment

The asynchronous timing analysis would initially take place on the controllers' side. Keep in mind that the ETG is constructed and the necessary cyclic slew and delay annotation is done on it during the early phases of the ASTA flow. Let's concentrate on the last section of the ASTA flow, which clarifies the link between the asynchronous and synchronous portions. The critical cycle and critical period are computed in the final stage of this flow. The critical cycle is a component of timing analysis that is built using critical arcs and will have the most impact on the circuit since the critical cycle's period will directly affect the circuit's period. The period of the critical cycle and the delay values of the components of the controllers that will provide the enable signals at the latches are of particular interest table 6.5.

ASTA Period (ns)	**Ctrl1/INV/ZN**		**Ctrl2/INV/ZN**	
	RISE	**FALL**	**RISE**	**FALL**
2.45907	1.9352843	3.0471899	0.7555204	1.9101156

Table 6.5: ASTA Period - Controllers Delay Values

The ASTA period has converged at 2.45907ns and we can obtain the delay values of the (+) and the (−) controllers' events that would help us indicate the clock signals.

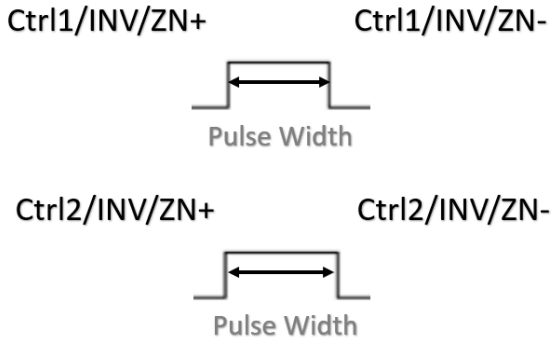

Figure 6.7: Bundled-Data Experiment Clock Generation

For proper timing analysis, we must perform two crucial checks while maintaining generated clocks. As these are relative offsets for the occurrence of these events, the reference edge timing offset and the other edge delay increment value may go beyond the period value. The adjustment is made by determining the smallest clock offset among all asynchronously produced clocks and then adjusting all clock edges by that offset. It's worth noting that this adjustment is also done to specify a certain clock reference starting point. To avoid data loss or violations, we must compare the produced clocks' pulse width with the library's minimum clock pulse width, as discussed in earlier chapters. So, after verifying that the minimum clock pulse width is not exceeded and after making the necessary clock adjustments, the final clock waveforms are:

$$CTRL1\_clk \quad \{1.1797639 \quad 2.2916695\}$$

$$CTRL2\_clk \quad \{0.0000000 \quad 1.1545953\}$$

Fast ICV receives these clocks as an input and produces Latch STA in the circuit's data path. The method has previously been thoroughly evaluated. The results of Fast ICV are shown in Table 6.6.

Testcase: Bundled-Data Latch-Based Datapath			
Latches: 12	**Clock Period (ns): 10**	**Clocks Waveforms: {0.0000000 1.1545953} {1.1797639 2.2916695}**	
Number of Clocks: 2	**Number of Iterations: 2**		
Fast ICV			
	Latest Arrival Rise Time	Latest Arrival Fall Time	
lat	0.80419	0.92180	
lat2	0.811559	0.597173	
lat3	1.73291	1.56887	
lat21	0.811559	0.597173	
lat22	0.811559	0.597173	
lat23	0.811559	0.597173	
lat24	0.811559	0.597173	
lat25	0.811559	0.597173	
lat26	0.811559	0.597173	
lat27	0.811559	0.597173	
lat28	0.811559	0.597173	
lat29	0.811559	0.597173	

Table 6.6: Bundled-Data Latch Data-Path Fast ICV Outcomes

Finally, we can note that in the data path shown above, Fast ICV performed in two iterations, but conventional ICV produced accurate results following twelve iterations.

Chapter 7

Conclusion

In conclusion, it is clear that this Latch STA approach, Fast ICV, can successfully perform STA on Synchronous, Cyclic, and Acyclic Latch-based designs, and that it is significantly correlated with the state-of-the-art industrial STA engine and the existing ICV algorithm, in terms of quality timing results and execution time. We need to mention that the combination of this work with the existing ASTA tool could provide timing analysis on Bundled-Data Latch-Based designs.

More specifically, our methodology comes up with a completely automated flow that can provide a comprehensive timing report as well as an SDF file that can be utilized for dynamic simulation, as long as the steps below are followed. After providing the STA engine with the necessary files, such as the Verilog netilst and the technology timing library, the next step is to enable the tool's relevant utilities based on the circuit type, period, and clock phase waveforms. In cases where the circuit is acyclic or cyclic with latches clocked on non overlapping clocks, we could let Fast ICV operate as it is. In a cyclic circuit, however, equilibrium slew computation and propagation must be enabled in cases where the circuit's clocks come from asynchronous elements or the clocks of the latches overlap in such a way that the Data to Output timing window is active at the same time for latches clocked on at different clock phases.

Finally, this technique validates that it functions under the core STA principles in the delay calculation and at the right setup checking by providing and performing accurate timing checks at this worst-case delay analysis.

Chapter 8

Future Work

As it is clear, that work proves how important is for a timing engine to perform efficient Latch STA and after that how useful may prove to use latches as the core structure of your design as long as proper timing verification can be achieved.

First of all, in order to define a fully-featured STA engine, the existing tool must handle the best case or hold analysis as a future step. In a similar approach to the longest analysis, the above methodology may do hold analysis and early delay propagation. Furthermore extending the proposed methodology to work under mixed sequential based designs, i.e., designs based on flip-flops and latches, and allowing latches to be clocked on several clock domains, will significantly increase the tool's usability.

Finally, as long as our tool successfully performs latch timing analysis, further stages can be considered latch-based design applications. As previously stated, latches operate faster and contribute to better power performance than flip-flops, therefore converting from flip-flops to latch-based designs would result in improved performance, and timing validation would not be an issue. [1] The same transformation may be used in Optimization Loop operations as long as the tool tends to fix the slack violation, in that case using latches on the flip-flop side may improve the results faster in a mix design.

Appendix A

Acronyms

STA Static Timing Analysis

ASTA Asynchronous Static Timing Analysis

EDA Electronic Design Automation

RTL Register Tranfer Level

SDC Synopsys Design Constraint

ICV Iterative Constraint Verification

SDF Standard Delay Format

VLSI Very Large Scale Integration

PBA Path Based Analysis

GBA Graph Based Analysis

AT Arrival Time

RAT Required Arrival Time

NLDM Non-Linear Delay Model

LUT Look Up Table

ETG Event Timing Graph

RTG Reduced Timing Graph

FIFO First In First Out

Bibliography

[1] H. Cheng, X. Li, Y. Gu, and P. A. Beerel, "Saving power by converting flip-flop to 3-phase latch-based designs," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 574–579.

[2] N. Xiromeritis, S. Simoglou, C. Sotiriou, and N. Sketopoulos, "Graph-based sta for asynchronous controllers," in *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2019, pp. 9–16.

[3] R. Bhasker, J. Chadha, *Static Timing Analysis for Nanometer Designs*. Springer Science & Business Media, 2009.

[4] S. Lin, C. Changfan, Y. Hsu, and F. Tsai, "Optimal time borrowing analysis and timing budgeting optimization for latch-based designs," *ACM Trans. Design Autom. Electr. Syst.*, vol. 7, pp. 217–230, 2002.

[5] S. Paik, L.-e. Yu, and Y. Shin, "Statistical time borrowing for pulsed-latch circuit designs," in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2010, pp. 675–680.

[6] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1904–1921, 2006.

[7] S. Stavros, "Composite current source static timing analysis cad tool implementation for mixed cyclic, acyclic circuits," Master's thesis, Volos, Greece, 2020.

[8] S. Simoglou, C. Sotiriou, and N. Blias, "Timing errors in sta-based gate-level simulation," in *2020 26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2020, pp. 1–2.

[9] S. Inc., *Synopsys Inc., PrimeTime GCA Tool Commands Manual, Version L-2016.06-SP2 ed.* Synopsys Inc., 2016.

[10] B. Li, N. Chen, and U. Schlichtmann, "Statistical timing analysis for latch-controlled circuits with reduced iterations and graph transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 11, pp. 1670–1683, 2012.

[11] L. Zhang, J. Tsai, W. Chen, Y. Hu, and C.-P. Chen, "Convergence-provable statistical timing analysis with level-sensitive latches and feedback loops," in *Asia and South Pacific Conference on Design Automation, 2006.*, 2006, pp. 6 pp.–.

[12] K. Sakallah, T. Mudge, and O. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 3, pp. 322–333, 1992.

[13] X. Nikolaos, "Static timing analysis algorithms for sequential cyclic circuits," Master's thesis, Volos, Greece, 2018, https://ir.lib.uth.gr/xmlui/handle/11615/50030.