Synchronizer Analysis and Design Tool: an Application to Automatic Differentiation

by

Justin James Reiher

BCS, University of British Columbia, 2017

B.Eng, University of Victoria, 2011

Dipl.T, British Columbia Institute of Technology, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master's of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

July 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Synchronizer Analysis and Design Tool: an Application to Automatic Differentiation

submitted by **Justin James Reiher** in partial fulfillment of the requirements for the degree of **Master's of Science** in **Computer Science**.

Examining Committee:

Additional Supervisory Committee Members:

Mark R. Greenstreet, Computer Science
Supervisor

Ian M. Mitchell, Computer Science
Additional Examiner

Abstract

In 2007, Yang and Greenstreet presented an algorithm that enables the computation of synchronizer failure probabilities, even when these probabilities are extremely small. Their approach gives a single probability number for the synchronizer but does not explain how the circuit details within the synchronizer contributes to the final result. We present an extension of their algorithm that connects the time-to-voltage gain of a synchronizer to the propagation of metastability through synchronizer circuits. This allows the designer to see what circuit features are helpful or not for synchronizer performance. There exists abundant folklore about what helps or hinders multistage synchronizer performance. We use our analysis to examine and explain such synchronizer folklore and draw novel conclusions. A brief error analysis is presented to provide evidence that the machinery used to compute our new measure of synchronizer effectiveness is accurate. The tools are exercised to objectively evaluate a handful of industry used synchronizer designs in order to compare their effectiveness against one another.

Lay Summary

Synchronizers are circuits which coordinate the transfer of information between two processors that run on different schedules. It is well known that synchronization can fail because the circuit gets stuck deciding if data transfer should occur or not. These events are very rare, thus making it difficult to analyze. The tools developed in this work provide a framework to evaluate the manifestation of those rare behaviors, delivers novel methods to systematically break down synchronization at a sub-circuit level and provides a tool to automatically optimize synchronizer circuits to minimize the probability of these failures. A consequence of this framework is that circuit designers now have a concrete way of objectively comparing synchronizer designs in a succinct and fair way.

Preface

At University of British Columbia (UBC), a preface may be required. Be sure to check the Graduate and Postdoctoral Studies (GPS) guidelines as they may have specific content to be included.

Table of Contents

List of Tables

List of Figures

Glossary

AD Automatic Differentiation

BSIM Berkeley Short-channel Insulated gate field effect transistors Model

CAD Computer Aided Design

CD Centered Differencing

CDC Clock Domain Crossing

CMG Common Multi-Gate

CMOS Complementary Metal Oxide Semiconductor

CPU Central Processing Unit

DIBL Drain Induced Barrier Lowering

EKV Enz, Krummenacher and Vittoz's model

GPS Graduate and Postdoctoral Studies

HP High Performance

IVP Initial Value Problem

MIT Massachusetts Institute of Technology

MOS Metal Oxide Semiconductor

MOSFET Metal Oxide Semiconductor Field Effect Transistor

MTBF Mean Time Between Failures

MVS MIT Virtual Source

ODE Ordinary Differential Equation

PCHIP Piecewise Cubic Hermite Interpolating Polynomial

PTM Predictive Technology Model

SPICE Simulation Program with Integrated Circuit Emphasis

VLSI Very Large Scale Integration

Acknowledgments

Thank you to Mark Greenstreet for reminding both of us on several occasions that we are computer scientists. Our conversations about the details of semi-conductor physics,electronics and countless other interesting subjects keep the research fun, engaging and meaningful.

A thank you to Ian Jones formally at Oracle Labs helped clarify aspects of synchronizer designs and always striving to explaining their behavior in simple terms.

A special thank you is in order to my loving wife Madeleine who has encouraged me to pursue this academic career and continues to support my pursuit of academia instead of pursuing lofty financial gains. It is thanks to her that I am here and able to work with the fantastic group of people here at UBC.

Chapter 1

Introduction

Modern day micro-electronic designs often have many separate computational blocks that are running in parallel. For a variety of engineering reasons, each computation block typically operates with its own localized clock, creating a clock domain. Everything from consumer grade electronics such as cell phones to the servers running cloud services are designed with multiple clock domains. In a typical server, each Central Processing Unit (CPU) can have several hundred to over a thousand clock domain crossings.

Figure 1.1 shows key trends in microprocessor designs. For the first 30 years, clock frequencies increased along a clearly exponential pathway. Around 2003, clock frequencies hit a plateau, and since then, the number of cores per CPU chip has had an increasing trend. New design challenges arise in multi-core/multi clock domain designs. Sending information between clock domains, called a Clock Domain Crossing (CDC), requires synchronization. The design of reliable synchronizers is a critical component to the correct execution of computational blocks.

The distinctive failure mode of a synchronizer is metastability – the output of a synchornizer can be logically undefined or change after an arbitrarily long time beyond the clock edge that sampled the input to the synchronizer. This can then cause failures in downstream logic circuits that depend on the output of the synchronizer. The probability of such failures can be made very small, but it cannot be driven all the way to 0. The metastable behavior of a synchronizer can be

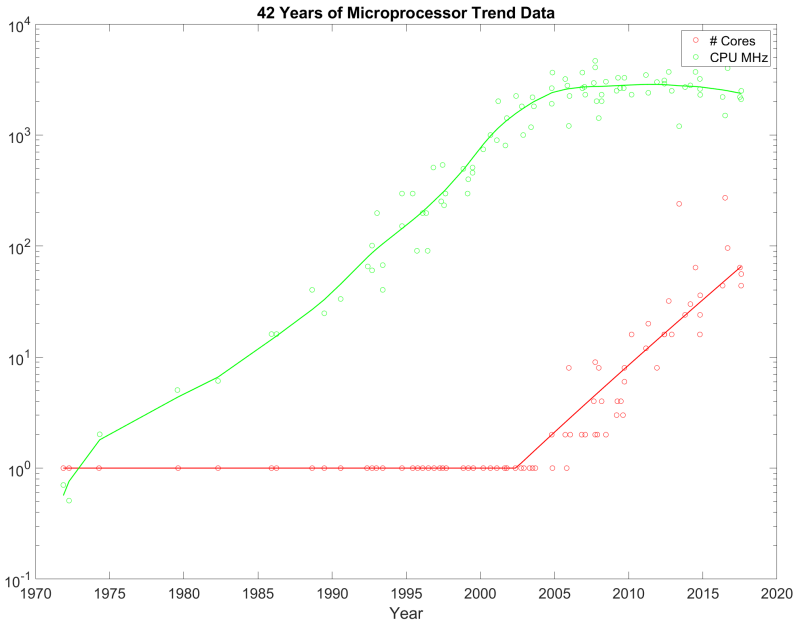

Figure 1.1: 42 years of microprocessor trend [44]

characterized as a saddle problem which has a failure probability described by an exponential law. The failure of a properly designed synchronizer is very rare. However, because of the number of synchronizers and frequency of sychronization events on modern CPU designs, a thorough understanding of synchronizer behavior is increasingly important for reliable CPU design.

This thesis introduces and develops a new tool to characterize and quantify the performance of synchronizers when metastable. This tool gives the designer new insight and quantifiable explanations for synchronizer behavior and failure probability by developing a time varying differential equation which describes synchronizer trajectories which are metastable. My implementation of this tool makes extensive use of Automatic Differentiation (AD). A benefit of this approach is that we can readily compute quantities related to the probability of synchronization failure. I present a work flow which can quantitatively compare the performance of synchronizer designs. I explore and quantify a passgate synchronizer design and

demonstrate the merits and pitfalls of proposed design modifications which aim to improve metastability resolution.

1.1 The Synchronization Problem

Synchronizers are circuits which coordinate the transfer of information across processing blocks operating on different clock domains. The digital behavior of a synchronizer is to *accept* or *reject* a synchronization request that occurs in a given clock period. The circuit designer's goal is to design a circuit which can achieve synchronization with a very large Mean Time Between Failures (MTBF) due to metastability.

Let's consider a very simple one-bit synchronizer with signals shown in Figure 1.2. Let our synchronizer's output and input bits both initially be 0. The transition to a 1 on the synchronizer's input indicates that another clock domain is offering a data transfer. The output of the synchronizer should transition to a 1 *synchronously with respect to the synchronizer's clock* to let the local computational block know that a transfer is ready. We assume that the circuitry using the output of the synchronizer requires the output to change before time t_{crit} and remain logically constant (1 or 0) over an interval of time $t \in [t_{crit}, t_{end}]$ otherwise a failure occurs.

Consider that a transfer requests may occur at any time on the interval $t \in [0, t_{end}]$. This gives rise to three distinct scenarios. The first scenario is that a request occurs at a time t_{hi} which is early enough, allowing our synchronizer's output bit to change to a 1 before t_{crit} and we accept the transfer. The second scenario is that a request arrives at a late time t_{lo} which does not change the output bit in the interval $t \in [t_{crit}, t_{end}]$, i.e. the request is rejected for the current synchronization period. The third scenario requires us to consider circuit-level models. Standard circuit models are based on ordinary differential equations, where the derivative function is C^1 or smoother. As a consequence, the output of the synchronizer must be a continuous function of the input signals and of time. Therefore, there exists times between t_{hi} and t_{lo} for which the output will have an intermediate value (neither logically 0, nor logically 1) for some $t \geq t_{crit}$ leading to a failure. This is the basic argument for why a perfect synchronizer cannot exist [33].

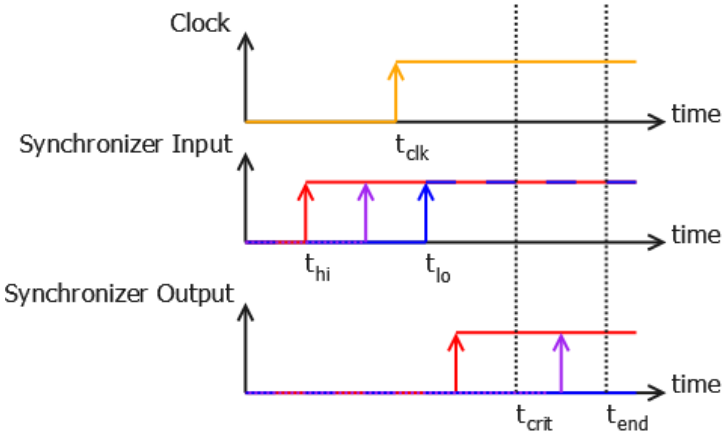

Figure 1.2: The synchronization problem

1.1.1 Saddle Problems

Synchronizers have behavior which belongs to the class of problems called *saddle problems*. Figure 1.3 illustrates that our one-bit synchronizer has two basins of attraction, if the input transition occur at or before t_{hi}, the synchronizer's output will go high which is one of its basins of attraction. Similarly for input transitions that occur at or after t_{lo} will result in the synchronizer's output to go low forming the other basin of attraction. Because the synchronizer's output changes its output from low to high (or high to low) in a continuous fashion, implies that there must exists a separator between the two basins of attraction.

A synchronizer's separator (or saddle) is an unstable equilibrium, hence the name metastable. The fact that a synchronizer circuit has an unstable equilibrium on the separator implies that for any synchronizer design, we can theoretically find an input transition which will cause the synchronizer's output trajectory to remain on the separator for an indefinite amount of time. A synchronizer whose output is on the separator is one that is metastable and depending on which side the trajectory leaves the separator will determine in which basin of attraction it will settle.

The dotted lines shown in Figure 1.3 represents that in practice there is a set of input transitions and a corresponding region around the separator which lead to metastability failures. We also in practice treat an output signal which is sufficiently close to one of the basins of attractions as a logical 0 or 1. A synchroniza-

tion failure occurs if by t_{crit} the synchronizer's output is sufficiently far from either basin of attraction.

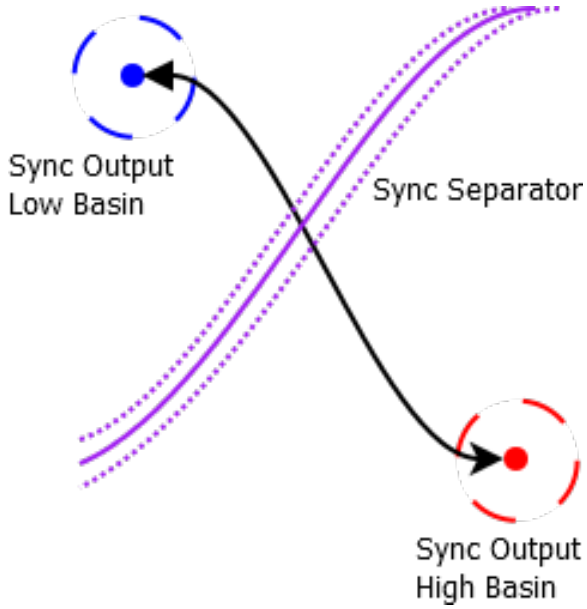

Figure 1.3: One-bit synchronizer as a saddle problem

Synchronizers are generally designed with multiple stages. The ideas of our one-bit synchronizer still applies, however, the multi-stage synchronizer give rise to a separator which is described with time varying dynamics. The details and implications are covered in this thesis. Designers often reason about the time-varying nature of synchronizer metastability as "moving" from stage $n-1$ to stage n and describe the synchronizer as metastable if any stage is behaving in a metastable fashion. Because most real synchronizers are multi-staged, have high dimensional state spaces *and* time-varying saddles, visualizing the time evolution of metastable behavior is difficult. In this thesis I provide a succinct way to both summarize and visualize their behavior.

1.2 Synchronizer Metastability Analysis Workflow

A high level overview of the work flow is shown in Figure 1.4. The highlighted green boxes in Figure 1.4 are components where I have made contributions. The

recipe to use the tool developed in this thesis is as follows:

- Pick a transistor model (details in Chapter 5)
- Design a synchronizer (example design in Chapter 6)
- Run the nested bisection algorithm (details in Chapter 2)
- Run the sychronizer analysis tool (details in Chapter 3)
- Examine results (example in Chapter 6)

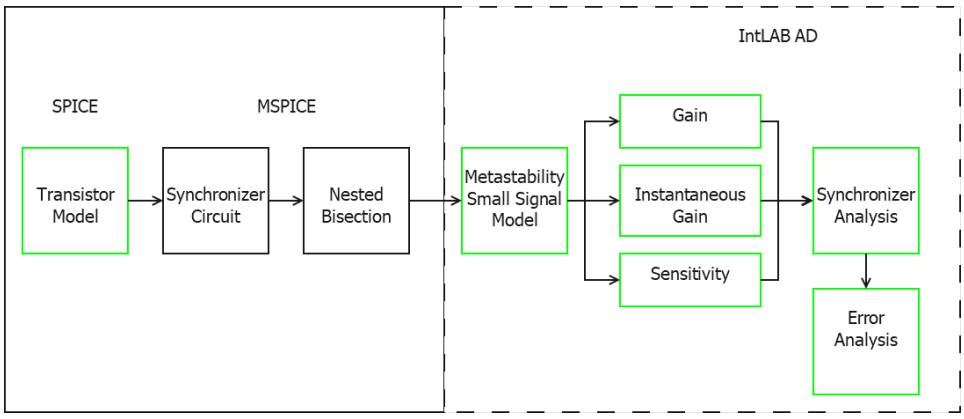

Figure 1.4: Analysis tool work flow

1.3 Contributions & Thesis Statement

In this thesis I implement Yang and Greenstreet's nested bisection algorithm [51] using Yan's MSPICE framework (MATLAB). The perturbation analysis techniques developed in this thesis are implemented using the automatic differentiation AD package in IntLAB [42]. To model deep-submicron Complementary Metal Oxide Semiconductor (CMOS) circuits, I adapted the Massachusetts Institute of Technology (MIT) Virtual Source (MVS) model [38] to work with AD and developed a simplified Enz, Krummenacher and Vittoz's model (EKV) [13]. My thesis statement is:

"I present a novel method to construct a simple, time-varying linear model of synchronizer dynamics from the non-linear circuit model. This enables new ways to

examine, explain, characterize, and explore metastable behaviour and the impact of design choices on synchronizer robustness."

This thesis is supported by four main contributions.

1. My main contribution is a method to automatically create a linear small-signal perturbation model to describe trajectories which lie near the separator. This linear model gives insight into the circuit's performance and failure probability. I have implemented this method as a new tool that provides the designer with insight into the time varying nature of the circuit's behavior as the trajectory evolves along the separator.

2. There is an enormous body of work which describes transistors to varying levels of detail. Different models focus on different aspects of semiconductor physics. My contribution is to provide a transistor model which is simple but reasonably accurate. This allows me to use my new analysis techniques without overly compromising accuracy and obtain plausible results in a timely manner. The model I have developed in this thesis is empirically fit to Simulation Program with Integrated Circuit Emphasis (SPICE) data and borrows from underlying physical principles where needed to increase the accuracy. Many transistor models are broken into operating regions with conditional statements which are problematic for AD packages. My contribution is to provide a model which by construction is AD friendly, simple enough to be easily used in experimental code, while realistic enough to capture the key properties of state-of-the-art designs.

3. Using this tool on a standard synchronizer design, I evaluate "improvements" that have been suggested by designs, and thereby found some unexpected behaviors of real synchronizer designs.

4. I also make contributions to the error analysis and implementation of AD methods through iterative algorithms such as numerical integration. I develop concrete examples with which I am able to test the accuracy of my methods with AD against straightforward numerical approximating methods and draw conclusions about the merits of using AD.

The organization of the thesis starts with background information on metastability analysis, circuit simulation and transistor modeling in Chapter 2. Chapter 3 provides the details of the mathematics behind the new tool developed for metastability and failure probability analysis in synchronizers. Chapter 4 outlines experiments performed to provide evidence that using AD to implement the mathematics in Chapter 3 provides numerically accurate results.

A brief survey in transistor modeling and the different effects that are considered are covered in Chapter 5, while also introducing two compact transistor models that have been evaluated in sychnronizer simulations. Chapter 6 goes through the analysis of a passgate synchronizer in detail to demonstrate how the new tool is applied. Chapter 7 finishes the thesis with a summary of the work presented and a list of near future work and where I believe the techniques can be further applied.

Chapter 2

Related Work

The novel work that I develop in this thesis builds off of many years of previous study in metastability, circuit simulators, transistor models and automatic differentiation. Metastable failures in synchronizers are a well known and studied phenomenon dating back to work by Kinniment and Edwards [24] and Chaney and Molnar [4] in the 1970s. In that same time frame, SPICE [37] was being developed as a tool to simulate circuit designs. Meanwhile short-channel Metal Oxide Semiconductor (MOS) effects [45] were being studied for use with Computer Aided Design (CAD) tools. The implementation of AD was also being developed in that time frame to automatically compute Jacobians, gradients and produce automatic error analysis on numerically computed solutions of Ordinary Differential Equation (ODE)s [39] [22].

This chapter is a brief overview of the work in the field of transistor modeling, circuit simulation, automatic differentiation, metastability analysis and synchronizer design.

2.1 Transistor Modeling

A plethora of work exists to model the behavior of MOS devices. There are many textbooks (e.g. [12, 15, 17, 49]). Our focus is on CMOS models for use in Very Large Scale Integration (VLSI). Modern technology processes require proper modeling of short-channel effects such as velocity saturation [45] [35]. The Berkeley

Short-channel Insulated gate field effect transistors Model (BSIM) has evolved to include many transistor effects and is generally considered to be the benchmark for MOS behavior. Compact models such as the MVS [38] and EKV model [13][10][11] seek to simplify MOS behavior to a handful of equations. By no means is this an exhaustive list. The work in this thesis does not require the full understanding of all the details of semi-conductor physics, but enough knowledge is needed in order to build a reasonable compact model.

2.1.1 BSIM and PTM model cards

The BSIM models [16] are the foundation of the SPICE family of simulators such as ngSPICE [47], HSPICE[18] and LTSPICE[9]. The BSIM group has developed models for bulk/substrate/well MOS devices which are the ones considered in this thesis. More recently, the BSIM group has developed models for FinFET and multi-gate MOS constructions [7] which are used in today's microprocessor designs[21].

The parameters used to simulate circuit designs with the BSIM models are often proprietary, which can be an impediment to public domain research. Researchers at Arizona State University provide open source model cards called the Predictive Technology Model (PTM) [20] which operate with the BSIM models. These PTM model cards are used as the basis for collecting "ground truth" MOS device data to build our compact model in Chapter 5.

2.1.2 Compact Models

The BSIM models describe the behavior of MOS devices quite accurately. However these models have many parameters and equations make them less intuitively understandable. For this reason compact MOS models which trade off accuracy for simplicity are also an area of active research. MIT researchers have developed one such model called the MVS model [38][23][48][28]. Their compact model comes with routines to automatically fit MOS device data supplied by the user. The model is semi-empirical, and describes the behavior of MOS devices in a piecewise fashion. Another compact model is the EKV model [13] which aims to describe all regions of operation for a MOS device with a single set of governing equations which apply for all operating modes. Both rely on building their models based on

physical properties such that the behavior can be explained through physics while maintaining simplicity.

2.2 Circuit Simulation

There are a number of circuit simulators which designers use to simulate their circuit designs. ngSPICE [47] is an open source SPICE program that is actively being developed. Analog Devices Inc. provides LTSPICE [9] for free which includes a graphical user interface. Often used in industry is a licensed version of SPICE program called HSPICE[18]. All of these SPICE programs provide means to simulate circuits with the BSIM models. Yan's work in [50] provides a MATLAB [31] version of SPICE called MSPICE. MSPICE is the base framework I use in this thesis. I have extended Yan's MSPICE to include the analysis tools described in this thesis. LTSPICE and ngSPICE by default use a multi-step Gear's integrator where as MSPICE uses by default MATLAB's `ode45` integrator. MSPICE has the advantage of exposing all the internal structures and make it easy to manipulate and add functionality. Future work could include the addition of these tools in ngSPICE.

2.3 Automatic Differentiation

In this work, we use automatic differentiation to automatically obtain derivatives from the transistor models. This technique for computing derivatives, gradients, Jacobians and Hessians is not new. Rall has a published set of lecture notes on using AD in a number of different ways in [39]. Details on an early implementations of AD for computer programs can be found in [22] which, within traces the ideas of recursive derivative definitions as early as the 1930s. Rump notes in [43] that "the method was found and forgotten several time, starting in the 1950s."

In this thesis, Rump's licensed IntLAB package [42] is used which implements automatic differentiation for MATLAB. Another MATLAB AD package developed by Coleman and Xu called ADMAT is found in [5]. More recent work in automatic differentiation include a package for the scientific computing language Julia [41]. Research is being conducted to develop an AD framework which also includes functionality in stochastic models and variables by Fries in [14]. This work could potentially be used to model the stochastic nature of asynchronous data signals.

The research by Eberhard and Bischof in [8] describes how to compute the derivative of parameters with respect to numerically integrated solutions. The described method of recasting this problem as a new ODE is adopted in this thesis to compute the derivative of numerically obtained trajectories with respect to various parameters. i.e.:

$$\begin{aligned} \frac{d}{dt}\left(\frac{d\mathbf{x}}{d\mathbf{p}^{\mathbf{T}}}\right) &= \frac{\partial \mathbf{f}}{\partial \mathbf{x}^{\mathbf{T}}}\nabla\mathbf{x} + \frac{\partial \mathbf{f}}{\partial \mathbf{p}^{\mathbf{T}}} \\ \nabla\mathbf{x}(t = t^0) &= \nabla\mathbf{x}^0 \end{aligned} \tag{2.1}$$

2.4 Metastability Analysis and Synchronizer Designs

Marino [30] and Mendler & Stroup [33] argue that resolving synchronizer metastability in a bounded time without error is a physically impossible endeavor. Their arguments are based on the continuous nature of the circuit models. Intuitively, these continuity arguments suggest a bisection method for finding metastable trajectories. Pick some time t_{crit} at which time the synchronizer is expected to have logically defined voltage levels. Start with finding an input t_{hi} which leads to the circuit's output to be high at t_{crit}, then find an input t_{lo} which leads to the circuit's output to be low at t_{crit}. Somewhere between these two exists an input t_{in} which leads to a metastable trajectory which is neither high or low at time t_{crit}.

2.4.1 Metastability

We now describe metastability and how it pertains to a synchronizer. Consider the scenario depicted in Figure 2.1 where we have two CPUs, A and B, where CPU A wants to send information over to CPU B. CPU A sends a signal to CPU B's synchronizer to indicate that CPU A has presented data for CPU B to consume. There are three possible outcomes from CPU B's synchronizer. The clock shown in Figure 2.1 is the clock for CPU B, and we refere to the first rising edge of the clock in this figure as clock edge 1.

- The first outcome (in Red), is that CPU A's signal arrives to CPU B's synchronizer's at a time t_{hi} well *before* CPU B's synchronizer's clock edge 1. CPU B's synchronizer outputs a high signal before clock edge 2 which triggers CPU B to prepare to receive data.

- The second outcome (in Blue) is that CPU A's signal arrives to CPU B's synchronizer at a time t_{lo} well *after* CPU B's synchronizer's clock edge 1 and only acknowledges CPU A's request on clock edge 2 which triggers CPU B to get prepared to receive data on clock edge 3. In relation to the first scenario, everything happens as is in the first scenario but delayed by one clock cycle.

- The last outcome lumps all other possible outcomes into one category. By the continuity arguments made by Marino [30] and Mendler & Stroup [33], assuming that the synchronizer is modeled by an ODE where the derivative function has at least C^1 continuity, then there must exist transition times t_{in} such that the synchronizer's output can be anywhere between high and low by time t_{crit}. This argument can be extended to propagate these non-digital values to *any* signal that depends on the synchronizer output, either in the current clock cycle or an arbitrary number of clock cycles later. For any signal that depends on the value of the synchronizer's output, there are transition times for t_{in} such that it leads to ill-defined values for the downstream signal.

The last outcome is what is called a synchronization failure due to metastability. This is problematic if the synchronizer's result branches as shown in Figure 2.2 to multiple places. For example, consider the case where the output of a synchronizer determines the next value of a CPU's program counter, e.g. whether the next instruction is fetched from the current program, or if the CPU jumps to an interrupt handler. Then some of the bits of the program counter may be set as if the synchronizer has resolved low, while others as if it had resolved high, and some bits could be invalid levels for a digital signal. This inconsistency can lead to corruption of data, system crash or other unexpected behavior. The possibility of the inconsistent interpretation of the synchronizer's output allows CPU B in our scenario to reach states that can not be explained by a purely digital model.

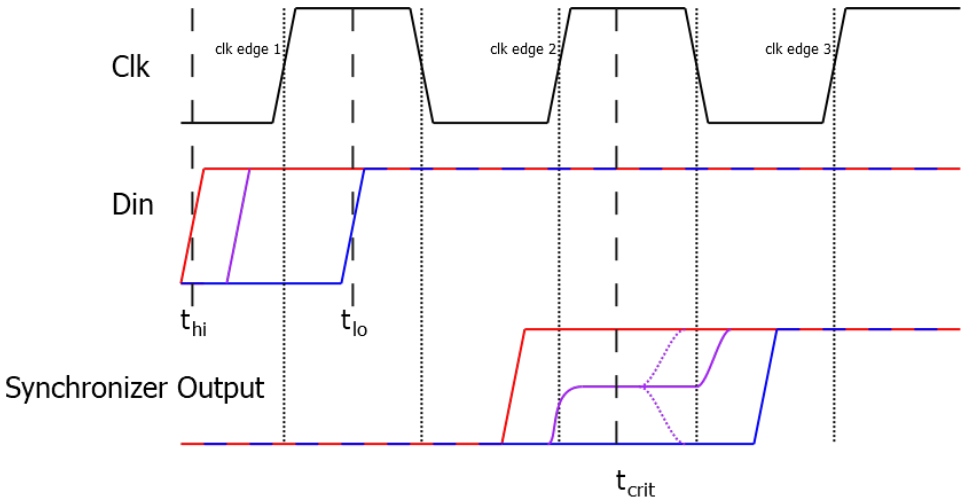

Figure 2.1: Synchronizer outcomes

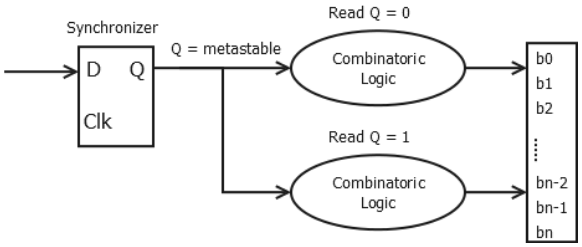

Figure 2.2: Synchronizer output branch

2.4.2 MTBF and Metastability Analysis

Intuitively, the mean-value argument from Section 2.4.1 above suggests that one could find times t_{hi} and t_{lo} for which the synchronizer resolves high and resolves low respectively. Then, an algorithm could apply bisection to find a narrow interval that contains the actual metastability failures of the synchronizer. Implementing such a bisection approach using simulators such as ngSPICE or HSPICE is hopeless in practice because the numerical errors preclude distinguishing behaviors for trajectories with nearly identical initial conditions. To achieve failure probabilities that are acceptable in real designs, the gap between t_{hi} and t_{lo} must be several orders of magnitude smaller than the resolution of the double-precision floating point representation. Yang & Greenstreet [51] [52] demonstrate an algorithm of "bisec-

tion with restarts" which has been implemented in this thesis MSPICE framework and the BlendICS MetaACE [34] tool.

Early analysis of synchronizers and metastability focused on a single latch. Couranz & Wann [6] and Kinniment & Wood [25] explored this and obtained an analytical model where:

$$P\{failure\} \quad = \quad T_w f_c e^{-t_{crit}/\tau} \tag{2.2}$$

$P\{failure\}$ is the probability of a synchronizer failure for a single transition to the synchronizer's input. T_w is the "input window" where the synchronizer is "vulnerable", f_c is the clock frequency and τ is the metastability resolution time-constant of the bistable element in the latch, e.g. the cross coupled pair of inverters (see Chapter 6). Equation 2.2 assumes that data transitions are equally likely to occur at any time within the clock period $P_{clk} = 1/f_{clk}$, i.e. the data transitions are drawn from a uniform distribution over the clock period P_{clk}.
Typically, the designer's concern is MTBF – the Mean Time Between Failure. For a single synchronizer, this is the mean time between synchronization events divided by the probability of a synchronization event failing. The rate of data transitions is written as f_{data}, and the probability of an individual synchronization event leading to failure is given by Equation 2.2. This leads to Equation 2.3:

$$MTBF \quad = \quad \frac{e^{t_{crit}/\tau}}{f_c f_{data} T_w} \tag{2.3}$$

Beer *et al* [3] outline a history of MTBF models for synchronizers and their multi-stage nature. They all follow a form similar to Equation 2.2. Yang and Greenstreet's nested bisection algorithm address the computation of the MTBF for a particular synchronizer design directly by reporting Δt_{in}, the width of the interval of input transition times for which failures can occur. By the assumption that input transition times are uniformly distributed over the clock period,

$$MTBF \quad = \quad \frac{P_{clk}}{\Delta t_{in} f_{data}} \tag{2.4}$$

of an input transition occurring within the small fraction of the clock period P_{clk}. The algorithm on its own however does not provide insight into the explanation or

causes for the different MTBF of the synchronizer. This thesis extends those tools for a better analysis and was first presented in [40].

2.4.3 The Nested Bisection Algorithm

In order to perform the synchronizer analysis, it is necessary to be able to simulate and produce metastable trajectories. Yang & Greenstreet's nested bisection algorithm [51] is a key contribution to the simulation/computation of metastable trajectories in synchronizers. Designers typcially want to have a MTBF on the order of millions of years to ensure that synchronization failures are sufficiently rare, even in designs with thousands of synchronizers. Note, the million year per-synchronizer MTBF is typical for servers and other designs with high reliability/availability requirements. Other designs (such as consumer products like cell phones) have much lower standards, with MTBFs of a few thousand years per synchronizer. However, given the exponential relationship between resolution time and failure probability, the issues remain largely the same.

Prior to the work in [51], it was not possible to simulate metastable trajectories to those kinds of MTBF via SPICE. SPICE is not capable of simultaneously placing a data transition with enough precision and simulating a synchronizer which exhibits the expected corresponding metastability behavior. This is because SPICE can not simulate a correspondingly accurate trajectory for an input that would occur within the necessary interval. An MTBF on the order of millions of years corresponds to a input time window Δt_{in} on the order of 10^{-25} to 10^{-30}.

The nested bisection algorithm solves this problem and functions in the following manner:

- Find an input transition time, t_{hi} for which the synchronizer settles high, and another time, t_{lo} for which the synchronizer settles low.

- To accelerate the search, we do an M-way bisection instead of a simple bisection. Because of vectorization, the cost of simulating M additional trajectories in MATLAB is relatively small in comparison to simulating M individual trajectories. We evenly space input transitions t_{in_i} between t_{hi} and t_{lo} in M.

- Launch M simultaneous synchronizer simulations where each synchronizer trajectory corresponds to a particular input transition time t_{in_i}. Let the k^{th} run of M simulations be called the k^{th} epoch. We write T_k to denote the start time of the k^{th} epoch.

- The user provides a function, `end_epoch(v)` that returns true when the voltages in voltage vector v indicate that all simulations have settled and/or resolved. For example, a simple criterion is to test that all voltages are within V_{tol} of V_{dd} or gnd.

- Search for the last synchronizer simulation trajectory that settles high and the first one that settles low. In order to make the classification of the trajectories robust, the before last trajectory that settled high $\vec{V}_H(t)$ and the 2nd trajectory that settles low $\vec{V}_L(t)$ are stored. We do this because if the first trajectory that settles low (or the last to settle high) happens to be very close to the separator it's possible that either of those trajectories occurred as a result of numerical artifacts. Selecting the trajectories that are once removed gives the algorithm some margin for error at the expense of a slightly longer run time (see Figure 2.3).

- At the end of epoch k, the algorithm needs a way to determine how far the overall simulation can advance in time. The goal is to determine a time interval $t \in [T_k, t_{lin}]$, such that all n simulation trajectories over that time interval can be approximated by a linear model. For each time point t_j, corresponding to the j^{th} time point in epoch k, we take each synchronizer voltage state vector $\vec{V}_i(t_j)$ across all M simulations are fit them to a cubic polynomial. If the synchronizer has N nodes, then the state for M simulations at time t_j is a vector of MN values. At each time-step, we calculate the ℓ_2 distance between this vector as determined by the integrator, and its approximation as determined by the cubic polynomial fit for that time. We find the first time-point for which this discrepancy is greater than some user specified tolerance, ε. The epoch ends at the previous time-point.

- The saved trajectories that settled high and low are used to set the initial conditions for the next set of simulations for epoch $k+1$ beginning at time

$T_{k+1} = t_{lin}$. The initial conditions for the following epoch is an affine combination of $\vec{V}_H(t_{lin})$ and $\vec{V}_L(t_{lin})$ i.e. $(1-\alpha)\vec{V}_H(t_{lin}) + \alpha\vec{V}_L(t_{lin})$ where $\alpha \in [0,1]$ split evenly in n.

- The relative time window Δt_{in} is updated corresponding to the fraction with which we have shrunk the relative time window.

- The input data transitions t_{hi} and t_{lo} are updated to correspond to the trajectories $\vec{V}_H(t)$ and $\vec{V}_L(t)$. The entire process is repeated until a user-specified end condition is reached, for example, the desired relative time window Δt_{in} is achieved.

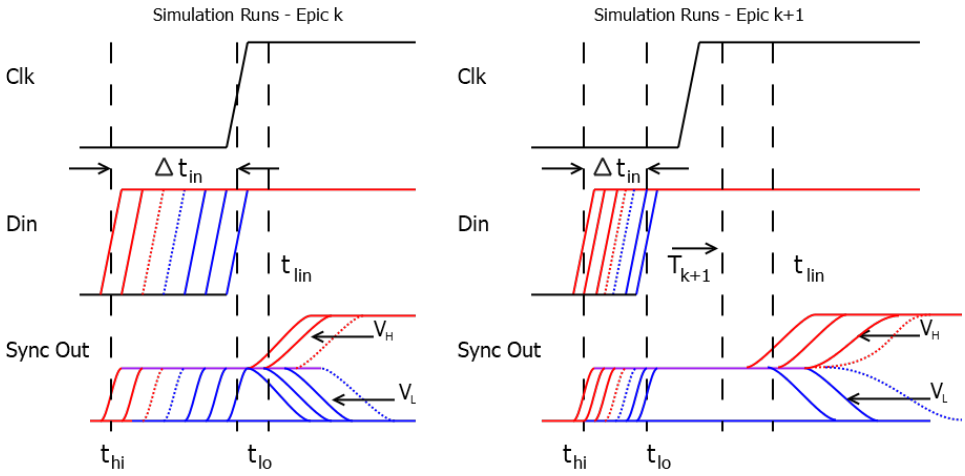

Figure 2.3: Nested Bisection Algorithm Illustration

Because the simulations are performed using MATLAB's `ode45` integrator we can restart simulations and impose stopping conditions to speed up simulation times. The linearity test via a cubic measure is a result of legacy test code which has been carried through without presenting any problems. There is certainly room for a more rigorous analysis and likely a cleaner test.

Another observation is that even though the input data transitions are indistinguishable using floating point arithmetic, the synchronizer states at each epoch boundary, T_k, clearly distinguish trajectories that settle high from those that settle

low. This is because these states include the voltages on all nodes, including those that are most relevant to resolving metastability at time T_k.

I also observed that progress is slow until the input data transitions t_{hi} and t_{lo} are narrowly placed. Once t_{hi} and t_{lo} are close to each other, the nested bisection algorithm progress much faster. Figure 2.3 illustrates a sample of the nested bisection algorithm from one epoch to the other. It is possible to generate trajectories corresponding to arbitrarily small relative input time windows Δt_{in} and overcome the limiting need to specify the input transition time to an accuracy greater than machine epsilon.

2.4.4 Synchronizers

Designers have focused on designing synchronizers to achieve large MTBF to ensure reliable computation. Zhou *et al.* [54] propose a synchronizer design which claims to be robust under voltage supply power variation and be efficient at metastability resolution. Beer and Ginosar [2] propose eleven ways to improve metastability resolution in synchronizer designs one of which is tested in this thesis. I demonstrate that it is advisable to avoid putting loads on critical nodes, an observation made in Beer and Ginosar's paper [2]. Yang *et al.* [53] performed a survey and comparison of synchronizer designs and propose a new design. The pseudo NMOS design in [53] is shown to have a larger MTBF when compared with previous synchronizer designs. More recently, Li *et al.* [27] have proposed that boosting the voltage supply when the synchronizer is metastable improves metastability resolution.

Apart from Yang *et al.* where failure probabilities were computed using the nested bisection algorithm, the ideas in these papers tend to revolve around folklore and intuition which often does not capture the entire picture as is demonstrated in this thesis.

Chapter 3

Synchronizer Metastability Analysis Tools

The details found in this chapter provides the foundation for the main contributions of this thesis. The chapter outlines the mathematics and details for synchronizer analysis in the metastable regime and clearly derives the MTBF of the synchronizer.

Equation 2.2 from Chapter 2 gives a widely used formula for modelling synchronizer failure probability. The input window T_w and resolution constant τ for a synchronizer are terms that have been either empirically fit from observation or derived for a single latch. The MTBF of a multi-stage synchronizer is not explicitly derived from the underlying circuit dynamics.

In this chapter, we build on the small-signal, linear model that is the foundation of the nested-bisection algorithm [51]. In particular, we show that there is a scalar quantity, $\lambda(t)$ that models how the synchronizer converts small changes to its input transition time into large changes in its output voltage at a later time, t_{crit}. We show that we can model $\lambda(t)$ as a first-order, linear, inhomogeneous ODE. The vector-valued function, $\beta(t)$ gives the time-to-voltage transfer of the synchronizer at time t; in other words, $\beta(t)$ describes the sensitivity of the voltage state at time t to the input transition time, t_{in}. This analysis provides a quantitative way to explain why a synchronizer has a particular failure probability. We can quantify the impact of particular sub-circuits of the synchronizer and identify which sub-circuits are critical at what times. As a bonus, we can calculate the values of parameters such

as T_w and τ from Equation 2.2 and connect these values to the dynamics of the synchronizer circuit.

Consider a one latch synchronizer, that is transparent when the clock signal is low and opaque when it is high. There are four time points of importance in the analysis of a synchronizer. Figure 3.1 illustrates an example where the times t_{clk}, t_{in}, t_{eola}, and t_{crit} are shown for an event that leads to a failed synchronization in our one latch example, i.e. the output changes at some time after t_{crit}.

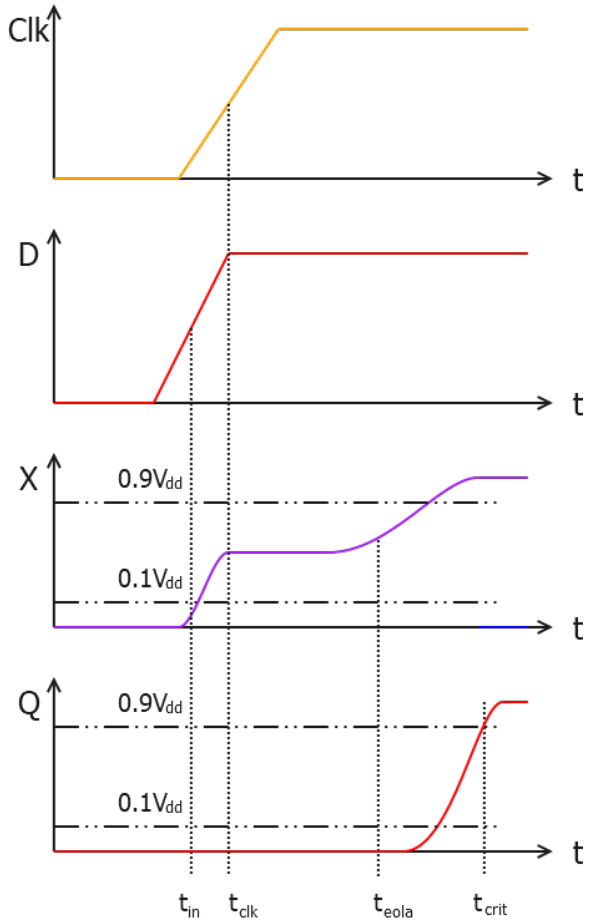

Figure 3.1: Synchronization timing analysis and failure example

- The time t_{clk} is the time at which the clock transition and makes the first stage of our synchronizer transition from being transparent to opaque. The period

of the clock is P_{clk}. In the following, we will describe other times, e.g. t_{in}, t_{eola}, and t_{crit} relative to t_{clk}. In otherwords, $t_{clk} = 0$ unless otherwise stated.

- The time t_{in} is the time at which the data transition occurs at the input of the synchronizer. This time is assumed to be uniformly distributed over the clock period P_{clk}.

- A designer will pick a time t_{crit} by which time the synchronizer's output needs to be logically defined, i.e. clearly *high* or clearly *low*, or else a synchronization failure has occurred. The time t_{crit} is relative to the transition of the clock.

- For trajectories that are metastable, the time t_{eola} is the time at which our linear analysis ends. From time t_{eola} to t_{crit} and beyond in a single period of P_{clk}, non-linear large signal behavior dictates the outcome of the synchronizer. The time t_{eola} is selected such that the dynamics of the synchronizer have a valid linear small signal behaviour but where $\vec{V}(t_{eola})$ has separated enough from the saddle to be clearly headed towards one of the basins of attraction.

To compute the failure probability of a synchronizer, we must identify what conditions constitute a "failure". Throughout this thesis, we will consider any change of the logical values of the synchronizer outputs between t_{crit} and the next rising edge of the clock to be a failure. Furthermore, if the synchronizer has multiple outputs, e.g. $\mathtt{Q}$ and $\overline{\mathtt{Q}}$, then we require these outputs to be logically consistent, e.g. $\mathtt{Q} = \neg\overline{\mathtt{Q}}$ during this interval; otherwise, the behaviour is a failure. Conversely, we place no constraints on the synchronizer outputs during the interval from a rising edge of the clock until t_{crit}.

There are three main pieces to derive the MTBF of a synchronizer. The first piece, is to compute $\beta(t) = \frac{\partial \vec{V}(t)}{\partial t_{in}}$ which represents how the voltage state $\vec{V}(t)$ of the synchronizer is affected by a change in t_{in}. Informally, $\beta(t)$ summarizes the cumulative progress the synchronizer has made at time t towards resolving metastability. The units of $\beta(t)$ are in $[V/s]$.

The second piece requires a designer to identify what components of the synchronizer state are important to metastability resolution at time t_{eola}. For exam-

ple, latches often involve two inverting gates connected in a cycle to form a state-holding element (see, e.g. Figure 6.1 or Figure 6.3). If $x(t)$ and $y(t)$ denote the voltages on the outputs of two such gates at time t, then $x(t) - y(t)$ could be such an indicator of progress from metastability. With the designer's selected measure $\tilde{u}^T$, a normalized unitless vector $u(t)^T$ is computed such that at the end of our linear analysis, $\tilde{u}^T \vec{V}(t_{eola})$ computes the designer's measure of importance to metastability resolution, e.g. $\tilde{u}^T \vec{V}(t_{eola}) = \frac{1}{\sqrt{2}}(x(t_{eola}) - y(t_{eola}))$. For times $t \in [t_{in}, t_{eola}]$, $u(t)^T$ is the component of the synchronizer state at time t such that perturbations of the state $\vec{V}(t)$ are co-linear with $\tilde{u}^T$ at time t_{eola}. $u(t)^T$ has the effect of telling the designer what will happen from time t to t_{eola}.

Lastly, the synchronizer's time to voltage gain, a scalar quantity measured in units of $[V/s]$, represents how effective the synchronizer is at resolving metastability at time t and is shown in Equation 3.1.

$$g(t) \quad = \quad u(t)^T \beta(t) \tag{3.1}$$

In Section 3.4, we show that $g(t)$ can be expressed as a time-varying, first-order, inhomogenous, differential equation of a single variable:

$$\frac{d}{dt} g(t) \quad = \quad \lambda(t) g(t) + \rho(t) \tag{3.2}$$

where, $\lambda(t)$ describes the instantaneous gain of the synchronizer and $\rho(t)$ describes the initial time-to-voltage conversion of the synchronizer. For a typical synchronizer, $\rho(t)$ rapidly approaches 0 after the initial clock event that samples $\mathtt{d}_{in}$. We write t_{homo} to indicate a time, after which $\rho(t)$ is effectively 0, and note that Equation 3.2 becomes a homogenous, linear, time-varying differential equation for $t > t_{homo}$.

In Section 3.6, we apply this analysis to a single latch and show how the quantities τ and T_w in Equation 2.2 can be obtained from our model. Section 3.7 shows how our analysis can be applied on a component-by-component basis to quantify how each transistor (or each inverter, etc.) of a synchronizer impacts the performance.

The tools developed in this chapter are exercised in Chapter 6 on a passgate

synchronizer design to illustrate the impact of design variants and how quantitatively they affect synchronizer performance. We develop a summarizing figure of merit $\lambda(t)$ called the instantaneous gain of the synchronizer which is used to compare the design variants in Chapter 6.

3.1 Linear Small Signal Model Construction

The nested bisection algorithm simulates M trajectories simultaneously and computes metastable trajectories forward in time. Once the algorithm has produced trajectories to the desired time t_{crit}, we collect those trajectories and create a single trajectory which we use for the remainder of the analysis.

Working backwards from time t_{crit}, we identify a time t_{eola} such that there is no ambiguity about how the synchronizer will resolve from time $t \in [t_{eola}, t_{crit}]$, but where the trajectories have not diverged enough to violate our linearity assumption. Therefore from time $t \in [t_{in}, t_{eola}]$ we can construct a single trajectory which describes the synchronizer as a linear small-signal perturbation model by computing $\beta(t)$, the derivative of the synchronizer state with respect to the input transition time, for all time $t \in [t_{in}, t_{eola}]$.

We note in Section 2.4.3 that simulating M circuit trajectories in the nested bisection algorithm accelerates our search. We defined N to be the number of circuit node variables per circuit, then the nested bisection algorithm at each time step is an operation which goes from $\mathbb{R}^{MN} \rightarrow \mathbb{R}^{MN}$. Computing the Jacobian for each of the M circuit models at each time point is an operation which goes from $\mathbb{R}^{MN} \rightarrow \mathbb{R}^{MN \times MN}$. For this reason we want to separate the linearization of the synchronizer to a single trajectory and not compute the Jacobian at each time step for all M circuit trajectories of the nested bisection algorithm. Doing so would be a $O(N)$ order increase per circuit in computing complexity, i.e. the complexity would go from $O(MN) \rightarrow O(MN^2)$ at each time step of simulation. We note that for simplicity, the $O(MN)$ and $O(MN^2)$ ignores the time for computing C^{-1}.

Constructing the metastable small-signal model of the synchronizer is broken into two parts. The first is to construct the "perfectly" metastable trajectory for the desired time t_{crit}. The second is to construct a linear mapping along the "perfectly" metastable trajectory. This mapping is then used to obtain a linear small-signal

perturbation model for all points in time $t \in [t_{in}, t_{eola}]$ for use in subsequent sections.

3.1.1 Constructing the "Perfectly" Metastable Trajectory

Each epoch of the nested bisection algorithm computes a pair of trajectories that bracket the "perfectly" metastable trajectory. We write t_k to denote the starting time of the k^{th} epoch, and note that t_{k+1} is the end-time of that epoch. Nested bisection computes M trajectories, and selects two that bracket the separator between trajectories that settle-high and those that settle-low: we write $\vec{V}_{H,k}$ to denote the trajectory that will ultimately settle-high, and $\vec{V}_{L,k}$ to denote the trajectory that will ultimately settle-low. These trajectories have associated values of the t_{in} parameter that we will refer to as $t_{in,H,k}$ and $t_{in,L,k}$ respectively.

For $1 \leq k \leq K$, where K is the total number of epochs computed, the initial points for trajectories $\vec{V}_{H,k}$ and $\vec{V}_{L,k}$ are an affine combination of the end points from epoch $k-1$. In particular, we define $\alpha_{H,k}$ and $\alpha_{L,k}$ such that:

$$\begin{aligned} \vec{V}_{H,k}(t_k) &= \alpha_{H,k}\vec{V}_{H,k-1}(t_k) + (1-\alpha_{H,k})\vec{V}_{L,k-1}(t_k) \\ \vec{V}_{L,k}(t_k) &= \alpha_{L,k}\vec{V}_{H,k-1}(t_k) + (1-\alpha_{L,k})\vec{V}_{L,k-1}(t_k) \end{aligned} \tag{3.3}$$

and note that $t_{in,H,k} = \alpha_{H,k}t_{in,H,k-1} + (1-\alpha_{H,k})t_{in,L,k-1}$, and $t_{in,L,k} = \alpha_{L,k}t_{in,H,k-1} + (1-\alpha_{L,k})t_{in,L,k-1}$. In the final epoch, $k = K$, we find trajectories $\vec{V}_{H,K}$ and $\vec{V}_{L,K}$ which just meet the user defined settling criteria at time t_{crit}.

The key idea of the nested bisection algorithm is to find trajectories that bound the metastable failures and are close enough to each other so that small-signal, linear analysis is applicable. In the final epoch, these trajectories diverge to the settled-high trajectory, $\vec{V}_{H,K}$, and the settled-low trajectory, $\vec{V}_{L,K}$. Clearly, these involve large-signal, non-linear behaviour. We write t_{eola} to denote the time at which the small-signal linear analysis ends, and the remainder of the analysis relies on large-signal, non-linear, methods. We now describe how we determine a value for t_{eola}. Let

$$\Delta V(t) = \tilde{u}^T \left(\vec{V}_{H,K}(t) - \vec{V}_{L,K}(t) \right) \tag{3.4}$$

$\Delta V(t)$ quantifies the amount of separation between the settles-high and settles-low trajectory. The user defines a threshold, ΔV_{eola} such that if $\Delta V(t) < \Delta V_{eola}$, then small-signal, linear methods are applicable. After $\Delta V(t) \geq \Delta V_{eola}$, then non-linear

methods should be used. In particular, we define

$$t_{eola} = \min_{t} \text{s.t.} \Delta V(t) \geq \Delta V_{eola} \tag{3.5}$$

Finally, we find $\alpha_{H,K}$ and $\alpha_{L,K}$ such that Equation 3.3 is satisfied for $\vec{V}_{H,K}(t_{eola})$ and $\vec{V}_{L,K}(t_{eola})$.

We now define the "perfectly" metastable trajectory, $\vec{V}_{K,meta}(t)$ as the one that bisects $\vec{V}_{H,K}$ and $\vec{V}_{L,K}$ – we don't actually need for it to be *exactly* on the separator between trajectories that would eventually settle-high and those that would eventually settle-low. We just need a continuous trajectory that spans the entire analysis (up until t_{crit}) and is in the region where our linearization of the model applies. To this end, we pick the trajectory that is half way between $\vec{V}_{H,K}$ and $\vec{V}_{L,K}$ in the final epoch and derive it's predecessors in earlier epochs as shown below. Working back to $k = 0$ from K, we get:

$$\begin{aligned} \tilde{\alpha}_K &= \tfrac{1}{2}(\alpha_{H,K} + \alpha_{L,K}) \\ \tilde{\alpha}_k &= \tilde{\alpha}_{k+1}\alpha_{H,k+1} + (1 - \tilde{\alpha}_{k+1})\alpha_{L,k+1}, \quad 0 \leq k < K \\ \vec{V}_{k,meta}(t) &= \tilde{\alpha}_k \vec{V}_{H,k}(t) + (1 - \tilde{\alpha}_k)\vec{V}_{L,k}(t) \end{aligned} \tag{3.6}$$

Given these interpolations, $\vec{V}_{k,meta}$ for each epoch k, we use MATLAB's Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) function to create a continuous function from the time-points of the combined epochs. This constructed trajectory, $\vec{V}_{meta}(t)$, is the "perfectly" metastable trajectory that we use in the reminder of the analysis.

We note that a synchronizer in reality has a window of times $t \in [t_{crit}, t_{\phi_{next}}]$ where if a change on the synchronizer's outputs would result in a failure. The time $t_{\phi_{next}}$ is the latest time where a change to the synchronizer's outputs would have an effect on the circuit's behaviour within the given clock period P_{clk}. In other words metastable resolution which occurs after time $t_{\phi_{next}}$ happens late enough to no longer affect the resultant outcome. Finding the window where such trajectories could exist is acknowledged as a more rigorous analysis but we only consider the first occurrence of failure at time t_{crit} in this thesis. Furthermore we note that ΔV_{eola} and t_{eola} are parameters to the analysis but as will be shown in Section 3.6 has a

range of valid choices and ultimately cancel each other out when computing the MTBF. The current implementation requires the user to provide ΔV_{eola}, however, ΔV_{eola} acts as another linearity test. We believe that the linearity test used for the nested bisection algorithm can also be used to determine t_{eola}. This would further simplify and automate the analysis for the user.

3.1.2 Small Signal Linear Mapping

A trajectory $\vec{V}(t)$ is the solution to the differential equation $\frac{d}{dt}\vec{V}(t) = f\left(\vec{V}(t), t_{in}\right) = C\left(\vec{V},(t)t_{in}\right)^{-1} I\left(\vec{V}(t), t_{in}\right)$. The Jacobian, $J_f(t) = \frac{\partial f\left(\vec{V}(t), t_{in}\right)}{\partial \vec{V}(t)}$ for all points in time from $t \in [t_{in}, t_{eola}]$, is used as a small-signal linear mapping to determine how f would change as a result of perturbing the voltage nodes $\vec{V}(t)$. Notice that if we also want to know how f would change as a result of perturbing when the input transition t_{in} occurs, we need $\frac{\partial f(t)}{\partial t_{in}}$. The Jacobian of f and $\frac{\partial f(t)}{\partial t_{in}}$ are automatically obtained by using AD for the trajectory $\vec{V}(t)$. Equation 3.7 shows the derivation for the Jacobian and $\frac{\partial f(t)}{\partial t_{in}}$ on the time interval $t \in [t_{in}, t_{eola}]$. The derivative of the inverse of a matrix is shown in Appendix A, Section A.1 for reference.

Tensor terminology and notation is at the edge of Mark and I's math knowledge, we have both attempted to get clarification from online sources with little success. We hope that Ian can confirm the terminology or set us straight here

Note that $C\left(\vec{V}(t), t_{in}\right)$ is a non-linear capacitance matrix with respect to the inputs $\vec{V}(t)$ and t_{in}. As a consequence when we compute $\frac{\partial C\left(\vec{V}(t), t_{in}\right)}{\partial \vec{V}(t)}$, an intermediate part of the Jacobian matrix, the result is a $N \times N \times N$ tensor. In order to compute the Jacobian $J_f(t)$, we need to compute $\frac{\partial C(\vec{V}(t), t_{in})}{\partial \vec{V}(t)} \otimes \left(C\left(\vec{V}(t), t_{in}\right)^{-1} I\left(\vec{V}(t), t_{in}\right)\right)$. The result of $\left(C\left(\vec{V}(t), t_{in}\right)^{-1} I\left(\vec{V}(t), t_{in}\right)\right)$ is a vector of size $N \times 1$ and so we are left with a tensor-vector product. We define $\otimes$ as the operator to denote a tensor-vector product and demonstrate our definition in Section A.3.

Observe that the synchronizer's input signal transitions from *gnd* to V_{dd} (or from V_{dd} to *gnd*) within some small time window around t_{in} and then remains constant for the remaining of the synchronization simulation. This means that $\frac{\partial f(t)}{\partial t_{in}}$ should go to zero for $t \gg t_{in}$.

The following is a list of definitions that are used in the derivation for Equa-

tion 3.7:

- $\vec{V}(t)$ is the vector of voltages in the synchronizer in units of $[V]$
- t_{in} is the time at which the input signal transition occurs in units of $[s]$
- Let $I_{nodes} = I(\vec{V}(t), t_{in})$ the current at each node of the synchronizer in units of $[A]$. Currents flowing out of a node are positive and currents flowing into a node are negative
- Let $C = C(\vec{V}(t), t_{in})$ be the capacitance matrix for the synchronizer in units of $[F]$
- Let $f = \frac{d}{dt}\vec{V}(t) = f(\vec{V}(t), t_{in}) = C^{-1} I_{nodes}$ be the derivative function for the synchronizer in units of $[V/s]$
- In order for f to be AD friendly, it is assumed in this thesis that $f \in C^{\infty}$, i.e. f is infinitely differentiable. (see Chapter 5 for details on f)
- $J_f(t)$ is in units of $[1/s]$
- $\frac{\partial f(t)}{\partial t_{in}}$ is in units of $[V/s^2]$

$$\begin{aligned} \frac{\partial f(t)}{\partial \vec{V}(t)} = J_f(t) &= -C^{-1}\left(\frac{\partial C}{\partial \vec{V}(t)} \otimes \left(C^{-1} I_{nodes}\right)\right) + C^{-1}\frac{\partial I_{nodes}}{\partial \vec{V}(t)} \\ \frac{\partial f(t)}{\partial t_{in}} &= -C^{-1}\frac{\partial C}{\partial t_{in}} C^{-1} I_{nodes} + C^{-1}\frac{\partial I_{nodes}}{\partial t_{in}} \end{aligned} \tag{3.7}$$

There are a few remarks to make with regards to $J_f(t)$ and $\frac{\partial f(t)}{\partial t_{in}}$. The first is that $\vec{V}(t)$ includes source voltages such as clock, power supply, ground and input source signals in addition to the voltage nodes of the synchronizer (i.e. the state variables of the ODE model), so when I refer to $J_f(t)$, this really is the Jacobian of the state variables – likewise when referring to $\frac{\partial f(t)}{\partial t_{in}}$. Implementing these derivatives require being able to separate out the sources and internal nodes.

3.2 Computing $\beta(t)$

Section 3.1 provides all the pieces needed to analyze synchronizers under metastability. We define a new term $\beta(t) = \frac{\partial \vec{V}(t)}{\partial t_{in}}$. What $\beta(t)$ tells us is how all the voltage nodes of the synchronizer change for a change in the input transition time t_{in}.

This means that a perturbation of t_{in}, the time at which the input signal changes, translates into a change in the synchronizer's voltage state; in other words, we are regarding the synchronizer as a time to voltage converter. What a designer wants is for $\beta(t)$ to be as large as possible because the goal is to design a circuit which is highly sensitive to t_{in}.

An intuitive way to think about this is to perform a small thought experiment where the synchronizer is metastable. If $\beta(t)$ were small, this would imply that perturbing t_{in} would result in a small change on the voltage nodes of the synchronizer. Let's say then that a large change in t_{in} is required to get the synchronizer to appreciably change its voltage state. By extension, this implies that a large change in t_{in} is required to get the synchronizer out of metastability. The opposite is also true, a very large value for $\beta(t)$ means a small perturbation to t_{in} translates into large changes in the synchronizer's voltage state.

How do we compute $\beta(t)$? From the nested bisection algorithm we can construct a "perfectly" metastable trajectory up to time t_{eola}. That trajectory is defined as $\vec{V}_{meta}(t)$. However, we are interested in $\frac{\partial \vec{V}_{meta}(t)}{\partial t_{in}}$, and $\vec{V}_{meta}(t)$ is computed via numerical integration of f; for realistic circuit models, there is no closed form solution or function that can be directly used to compute $\beta(t)$. What we can do is compute $\frac{\partial f(t)}{\partial t_{in}}$ and $J_f(t)$. We pose the problem of computing $\beta(t)$ as an Initial Value Problem (IVP) ODE shown in Equation 3.8.

$$\begin{aligned} \dot{\beta}(t) &= J_f(t)\beta(t) + \tfrac{\partial f(t)}{\partial t_{in}} \\ \beta(0) &= 0 \end{aligned} \tag{3.8}$$

To solve for $\beta(t)$, just like solving for $\vec{V}_{meta}(t)$ we numerically integrate Equation 3.8. Note that $\beta(t)$ could be computed in conjunction to computing $J_f(t)$ and $\frac{\partial f(t)}{\partial t_{in}}$. This is not done because $J_f(t)$ is used in multiple places in the overall analysis and $J_f(t)$ is the time consuming quantity to compute, therefore $J_f(t)$ and $\frac{\partial f(t)}{\partial t_{in}}$ are pre-computed. Note that $\beta(t)$ can be estimated by numerical differencing on each nested bisection epoch k as the nested bisection algorithm makes progress in finding metastable trajectories to the desired t_{crit}. Equation 3.9 shows how this estimate is obtained. We can use Equation 3.9 as a quantity to compare against our

solution obtained from Equation 3.8.

$$\begin{aligned} \hat{\beta}(t) &= \frac{\vec{V}_{H,k}(t)-\vec{V}_{L,k}(t)}{t_{in,H,k}-t_{in,L,k}} \\ t &\in [t_k, t_{k+1}] \end{aligned} \tag{3.9}$$

3.3 Sensitivity Analysis $u(t)^T$

Section 3.2 outlines how to compute $\beta(t)$ which is the sensitivity of all the voltage nodes at time t to the input signal transistion time t_{in}. Our main concern is how a perturbation to the voltage nodes of the synchronizer at time t_{in} will be reflected to the nodes of interest at time t_{eola}. We define a sensitivity matrix $S(t)$ as:

$$\begin{aligned} S(t) &= \frac{\partial \vec{V}(t)}{\partial \vec{V}(0)} \\ S(0) &= I \end{aligned} \tag{3.10}$$

Equation 3.10 answers the question of how a change in $\vec{V}(0)$ changes $\vec{V}(t)$ for all time. To generalize, let the sensitivity matrix $S(t_1,t_2)$ explain how $\vec{V}(t_2)$ changes as a result of perturbations that occur at time t_1 for all $t \in [t_1,t_2]$. In other words $S(t_1,t_2)$ is the small-signal model of the circuit. We can solve for $S(t_1,t_2)$ similarly to solving for $\beta(t)$:

$$\begin{aligned} \tfrac{d}{dt}S(t_1,t) &= J_f(t)S(t_1,t) \\ &\quad \text{in integral form:} \\ S(t_1,t) &= S(t_1,t_1) + \textstyle\int_{t_1}^{t} J_f(w)S(t_1,w)dw \\ S(t_1,t_1) &= I \end{aligned} \tag{3.11}$$

What we want is to compute $S(t,t_{eola})$ for all time $t \in [t_{in},t_{eola}]$. However Equation 3.11 would need to be solved for all time points t_i along the "perfectly" metastable trajectory, this would be prohibitively expensive to compute. Instead, we compute $S(t_1,t_2)^{-1} = S(t_2,t_1)$, which has the effect of reversing the roles of t_1 and t_2. Observe that this reversal is the same as looking backwards in time from t_2 to t_1. Thus to understand how a perturbation at $\vec{V}_{meta}(t)$ perturbs $\vec{V}_{meta}(t_{eola})$ requires knowledge of the future. Because we have already computed the "perfectly" metastable trajectory to t_{eola}, we can indeed integrate backwards in time

starting from t_{eola}. Equation 3.12 shows the derivation for $S(t,t_{eola})$:

$$\begin{aligned}
\tfrac{d}{dt}S(t,t_{eola}) &= \tfrac{d}{dt}S(t_{eola},t)^{-1} \\
&= -S(t_{eola},t)\tfrac{d}{dt}S(t_{eola},t)S(t_{eola},t)^{-1} \\
&= -S(t_{eola},t)J_f(t)S(t_{eola},t)S(t_{eola},t)^{-1} \text{ from Equation 3.11} \\
&= -S(t_{eola},t)J_f(t) \\
&\quad \text{in integral form this is:} \\
S(t,t_{eola}) &= I - \textstyle\int_{t_{eola}}^{t} S(t_{eola},w)J_f(w)dw
\end{aligned} \tag{3.12}$$

$S(t,t_{eola})$ is a matrix which is of size $N \times N$, where N is the number of circuit nodes. Solving for all $S(t,t_{eola})_{i,j}$ elements at each time step is actually overkill for what the designer is interested in. We can dramatically speed up the computation if we project $S(t,t_{eola})$ onto the nodes of interest. The designer chooses which nodes they want to investigate and creates a unit vector $\tilde{u}^T$ which is right multiplied by $S(t,t_{eola})$. The vector $\tilde{u}^T$ may consider the difference of two nodes, the sum of some nodes or possibly a single node. The designer has the flexibility to choose the measure they are looking for with this vector. To prevent numerical overflows, the resulting vector is normalized and so we define $u(t,t_{eola})^T$ in Equation 3.15:

$$u(t,t_{eola})^T = \frac{\tilde{u}^T S(t,t_{eola})}{||\tilde{u}^T S(t,t_{eola})||} \tag{3.13}$$

In short form we refer to Equation 3.15 as $u(t)^T$. To efficiently compute $u(t)^T$, Equation 3.14 puts all the pieces together:

$$\begin{aligned}
w(t)^T &= \tilde{u}^T S(t,t_{eola}) \\
w(t_{eola})^T &= \tilde{u}^T \\
\tfrac{d}{dt}w(t)^T &= \tfrac{d}{dt}\tilde{u}^T S(t,t_{eola}) \\
&= \tilde{u}^T \tfrac{d}{dt}S(t,t_{eola}) \\
&= -\tilde{u}^T S(t,t_{eola})J_f(t) \\
&= -w(t)^T J_f(t)
\end{aligned} \tag{3.14}$$

Solving the IVP above via numerical integration we get:

$$u(t)^T = \frac{w(t)^T}{||w(t)||} \tag{3.15}$$

In practice, $u(t)^T$ provides a magnifying glass that identifies which nodes are important at which time to resolving metastability by t_{eola}. Let's explore how this can be used.

3.4 Synchronizer gain $g(t)$ and Instantaneous gain $\lambda(t)$

We now have all the tools necessary to derive a precise formulation for the gain of the synchronizer $g(t)$ and how we can summarize the effectiveness of a synchronizer. We developed $\beta(t)$ which measures the sensitivity to the synchronizer's voltage nodes with respect to the input signal transition t_{in}. Next we developed a mechanism that allows us to understand how perturbations at time t manifest themselves at time t_{eola} through $u(t)^T$. Combining these two ideas, the gain of the synchronizer is:

$$g(t) \quad = \quad u(t)^T \beta(t) \tag{3.16}$$

The designer hopes that $u(t)^T$ and $\beta(t)$ align themselves exactly, i.e $\frac{\beta(t) \cdot u(t)^T}{||\beta(t)|| ||u(t)^T||} = 1$. We can reason about $g(t)$ as being the amount of useful gain the synchronizer develops as a result of $\beta(t)$. The synchronizer gain $g(t)$ is still describing a time to voltage conversion, i.e. $\beta(t)$ is in units of $[V/s]$ and $u(t)^T$ is unitless, so $g(t)$ is also in units of $[V/s]$

The last piece of the puzzle is to determine how $g(t)$ changes in time: $\frac{d}{dt} g(t) = \dot{g}(t)$. The derivation for $\dot{g}(t)$ is quite involved, thus the main points of the derivation are highlighted in Equation 3.17, where the full blown details are found in

Appendix A Section A.2.

$$\begin{aligned}
\dot{g}(t) &= \tfrac{d}{dt}\left(u(t)^T\beta(t)\right) \\
&= \tfrac{d}{dt}\left(\tfrac{\tilde{u}^T S(t,t_{eola})}{||\tilde{u}^T S(t,t_{eola})||}\beta(t)\right) \\
&= \tfrac{d}{dt}\left(\tfrac{1}{||\tilde{u}^T S(t,t_{eola})||}\right)\tilde{u}^T S(t,t_{eola})\beta(t) \\
&\quad + \tfrac{d}{dt}\left(\tilde{u}^T S(t,t_{eola})\right)\tfrac{\beta(t)}{||\tilde{u}^T S(t,t_{eola})||} \\
&\quad + \tfrac{\tilde{u}^T S(t,t_{eola})}{||\tilde{u}^T S(t,t_{eola})||}\tfrac{d}{dt}\beta(t) \\
&= u(t)^T J_f(t)u(t)g(t) \\
&\quad - u(t)^T J_f(t)\beta(t) \\
&\quad + u(t)^T\left(J_f(t)\beta(t) + \tfrac{\partial f(t)}{\partial t_{in}}\right) \\
&= u(t)^T J_f(t)u(t)g(t) + u(t)^T\tfrac{\partial f(t)}{\partial t_{in}}
\end{aligned} \tag{3.17}$$

Let $\lambda(t) = u(t)^T J_f(t)u(t)$ and $\rho(t) = u(t)^T\frac{\partial f(t)}{\partial t_{in}}$ which are time varying scalar quantities, then we obtain the time varying differential equation:

$$\dot{g}(t) = \lambda(t)g(t) + \rho(t) \tag{3.18}$$

The $\rho(t)$ term is transient, and when $t \gg t_{in}$, $\dot{g}(t) \approx \lambda(t)g(t)$. Choosing t_1 such that $\rho(t) \approx 0$ for all $t \geq t_1$, allows us to integrate $\dot{g}(t)$ and get a solution of the form:

$$g(t) \approx g(t_1)\exp\left(\int_{t_1}^{t}\lambda(s)ds\right) \tag{3.19}$$

This approximate solution shows the familiar emergence of the exponential behavior of synchronizer probability failure. If we consider the exponential term $\lambda(t)$, we can talk about how the gain is helping metastability resolution at all points in time. We define the term $\lambda(t)$ as the synchronizer's instantaneous gain and is the term of interest in analyzing synchronizer designs.

$\lambda(t)$ is the term that is used to observe a global picture of the effectiveness of the synchronizer. It is desired for this term to be large and positive. If $\lambda(t)$ becomes negative then the sychronizer is harming metastability resolution, and if it is zero the synchronizer is "stalled", in other words not doing anything useful.

In practice, concerns about $\lambda(t)$ (should) dominate synchronizer design and analysis. The $\rho(t)$ term quantifies the necessary initial "time to voltage" conversion at the input of the synchronizer. Thus, questions about the impact of "edge sharpening" and other changes to the input circuitry of the synchronizer can be quantitatively analyzed by considering $\rho(t)$.

3.5 Failure Probability and MTBF

To determine the failure probability of a synchronizer, we determine the width of the interval for t_{in} for which the synchronizer does not resolve by time t_{crit}. We write Δt_{in} to denote the width of this interval for t_{in}. Assuming that transitions on $\mathtt{d}_{in}$ are uniformly distributed over the clock period, P_{clk}, the probability of failure for a single synchronization event is:

$$P\{failure\} = \frac{\Delta t_{in}}{P_{clk}} \tag{3.20}$$

Intervals of time at t_{in} correspond to intervals in the voltage state at time t_{eola}, and we get

$$\begin{aligned} \Delta V_{eola} &= g(t_{eola})\Delta t_{in} \\ \Rightarrow \quad \Delta t_{in} &= \frac{\Delta V_{eola}}{g(t_{eola})} \\ \Rightarrow \quad P\{failure\} &= \frac{\Delta V_{eola}}{g(t_{eola})P_{clk}} \end{aligned} \tag{3.21}$$

More precisely, those intervals of time at t_{in} correspond to intervals in the separation between the settle-high and settle-low voltage from Equation 3.4 such that $\Delta V(t_{eola}) < \Delta V_{eola}$. The width of the corresponding window for t_{in} is $\Delta V_{eola}/g(t_{eola})$. Recall Equation 3.18:

$$\frac{d}{dt}g(t) = \lambda(t)g(t) + \rho(t)$$

As noted earlier, $\rho(t)$ goes rapidly to zero after the clock edge that initially samples $\mathtt{d}_{in}$. Let t_{homo} be a time, such that for $t \geq t_{homo}$, $\rho(t) \approx 0$ (more precisely $\rho(t) \ll \lambda(t)g(t)$). Solving Equation 3.18 for $t \geq t_{homo}$, we get:

$$g(t) \approx g(t_{homo}) \exp\left(\int_{t_{homo}}^{t} \lambda(s)ds\right) \tag{3.22}$$

Substituting the approximation into Equation 3.21, we get:

$$P\{failure\} = \frac{\Delta V_{eola}}{g(t_{homo})P_{clk}} \exp\left(-\int_{t_{homo}}^{t} \lambda(s)ds\right) \quad (3.23)$$

We note that $g(t_{homo})$ is a property of the synchronizer circuit, independent of t; $g(t_{homo})$ models the initial time-to-voltage sensitivity of the synchronizer to changes of t_{in}. Likewise, ΔV_{eola} is a property of the circuit; ΔV_{eola} describes the divergence from metastability required to ensure that the synchronizer settles to digitally defined values by time t_{crit}. P_{clk} is an operating condition for the synchronizer. Finally $\exp\left(-\int_{t_{homo}}^{t} \lambda(s)ds\right)$ gives the exponentially decreasing probability of failure with increased time for synchronization.

If the average rate of transitions on $\mathtt{d}_{in}$ is given by f_d, we get that the failure rate is $P\{failure\}f_d$. Thus,

$$\text{MTBF} = \frac{g(t_{homo})}{\Delta V_{eola}} \frac{P_{clk}}{f_d} \exp\left(\int_{t_{homo}}^{t} \lambda(s)ds\right) \quad (3.24)$$

3.6 Computing T_w

For a one latch synchronizer, we can derive the quantities T_w and τ from Equation 2.2. To demonstrate this, we derive τ and T_w using the passgate latch which will be covered in Chapter 6. Note that from Equation 2.2 and Equation 3.21 we get:

$$T_w = \frac{\Delta V_{eola}}{g(t_{eola})} \exp(t_{crit}/\tau) \quad (3.25)$$

We assume that t_{eola} is sufficiently far away from t_{clk} (otherwise the synchronizer is being misused) that $\rho(t) \approx 0$ and $\lambda(t)$ is nearly constant for t leading up to t_{eola}. As shown in Figure 3.4, we can find a line-of-best fit for $g(t)$ to obtain:

$$g(t) \approx g_0 \exp(\lambda_0 t) \quad (3.26)$$

Let $\tau = 1/\lambda_0$. In the region where $\lambda(t)$ is nearly constant, we can approximate $\log \Delta V(t)$ with:

$$\Delta V(t) \approx V_0 \exp(\lambda_0 t) \quad (3.27)$$

where $V_0 = \Delta V_{eola} \exp(-\lambda_0 t_{eola})$. Equivalently, we can write $\Delta V_{eola} = V_0 \exp(\lambda_0 t_{eola})$. Substituting the approximation for $g(t_{eola})$ from Equation 3.26 and the approximation for ΔV_{eola} into Equation 3.25 yeilds:

$$T_w = \frac{V_0}{g_0} \exp(t_{crit}/\tau) \tag{3.28}$$

We define

$$\Delta V_{crit} = V_0 \exp(t_{crit}/\tau) \tag{3.29}$$

Noting that ΔV_{crit} is the extrapolation of $\Delta V(t)$ to time t_{crit}. We obtain Equation 2.2:

$$\begin{aligned} P\{failure\} &\approx \frac{T_w}{P_{clk}} \exp(-t_{crit}/\tau) \\ \tau &= 1/\lambda_0 \\ T_w &= \frac{\Delta V_{crit}}{g_0} \end{aligned} \tag{3.30}$$

where λ_0 is the "steady-state" value for $\lambda(t)$, ΔV_{crit} is from Equation 3.29, and g_0 is from Equation 3.26.

This derivation has a nice graphical interpretation. We can plot $\log g(t)$ vs. t which for time t greater than a few gate delays will be linear by our approximation to $g(t)$ in Equation 3.26. We can find the line of best fit as described above for the data on the time interval $t \in [t_{lin}, t_{eola}]$ where time $t_{lin} \gg t_{clk}$. The slope of our best fit line will be $1/\tau = \lambda_0$ and the y-intercept is g_0. Similarly we can plot $\log \Delta V(t)$ vs. t and find the line of best fit on the same time interval. We find ΔV_{crit} where the line of best fit intercepts the time t_{crit}. We give an example of this approach in Section 3.6.1 below.

3.6.1 Obtaining T_w for a passgate latch

With the above derivation, I now demonstrate how we can obtain T_w for a single passgate latch synchronizer shown in Figure 6.1. The passgate latch is described in much more detail in Chapter 6. In this example I select $t_{crit} = 1.93 \times 10^{-10}$ $[s]$, and select $\tilde{u}^T = \frac{1}{\sqrt{2}}(\mathtt{y0} - \mathtt{x0})$. The resultant nested bisection algorithm trajectories are shown in Figure 3.2 and the resultant plot for $\lambda(t)$ in Figure 3.3.

We observe that in Figure 3.3 for times $t > 5 \times 10^{-11}$ s, $\lambda(t)$ indeed remains nearly constant at a value of $\lambda_0 = 14 \times 10^{10}$ s^{-1}. Figure 3.4 shows $\log g(t)$ vs. t,

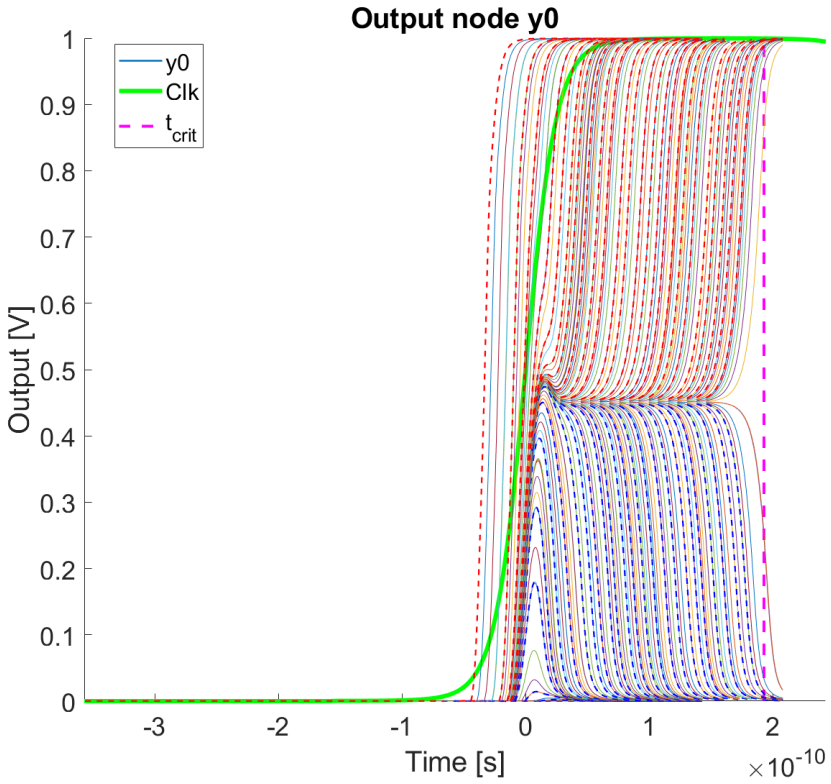

Figure 3.2: Nested bisection trajectories for passgate latch

the linear best fit line and the y-intercept $g_0 = 2.09 \times 10^{10}\ V/\text{s}$. Figure 3.5 shows $\log \Delta V(t)$ vs. t, the linear best fit line and the x-intercept at t_{crit} where $\Delta V_{crit} = 2.845\ V$. Finally we can compute $T_w = \frac{\Delta V_{crit}}{g_0} = 1.36 \times 10^{-10}$ s for the passgate latch and this particular value of t_{crit}. Note that computing $\Delta V(t)$ in Figure 3.5 for times beyond t_{eola} includes several approximations because of the large signal non-linear behaviour. On the other hand we see in Figure 3.5 that we have a lot of freedom in picking ΔV_{eola} as our criteria to determine t_{eola} before the linearity assumption fails. We also do not compute $g(t)$ beyond t_{eola} in Figure 3.4 for similar reasons.

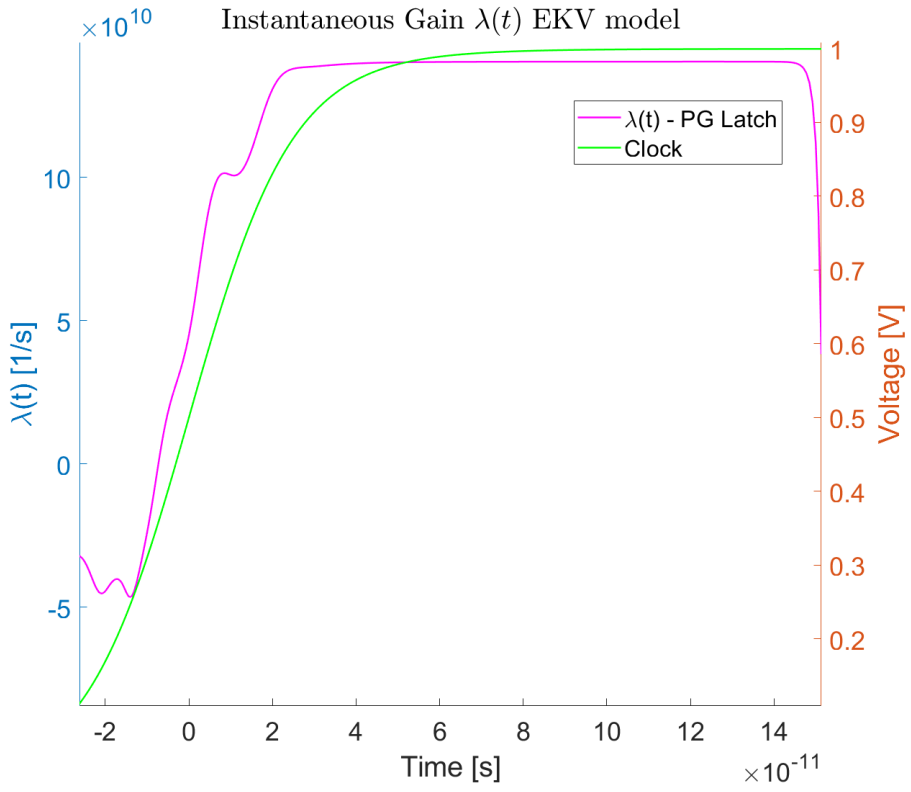

Figure 3.3: $\lambda(t)$ for passgate latch

3.7 Component Gain Decomposition Tool

The function $u(t)^T$ as developed in Section 3.3 gives a precise mathematical formulation of the notion that metastability is a time varying quantity which "moves" from one latch stage to the next during the operation of the synchronizer. Metastability arises in latch $i+1$ because it becomes opaque before latch i fully leaves metastability, thus landing latch $i+1$ in a metastable voltage state. The term $u(t)^T$ points to the nodes of importance which are currently contributing to the synchronizer's ability to resolve at time t_{eola}. Here, we build upon the derivations for $g(t)$, $\lambda(t)$ and $u(t)^T$ to show how one can isolate sub-circuits and components of the synchronizer and determine how those parts are contributing to the overall instantaneous gain.

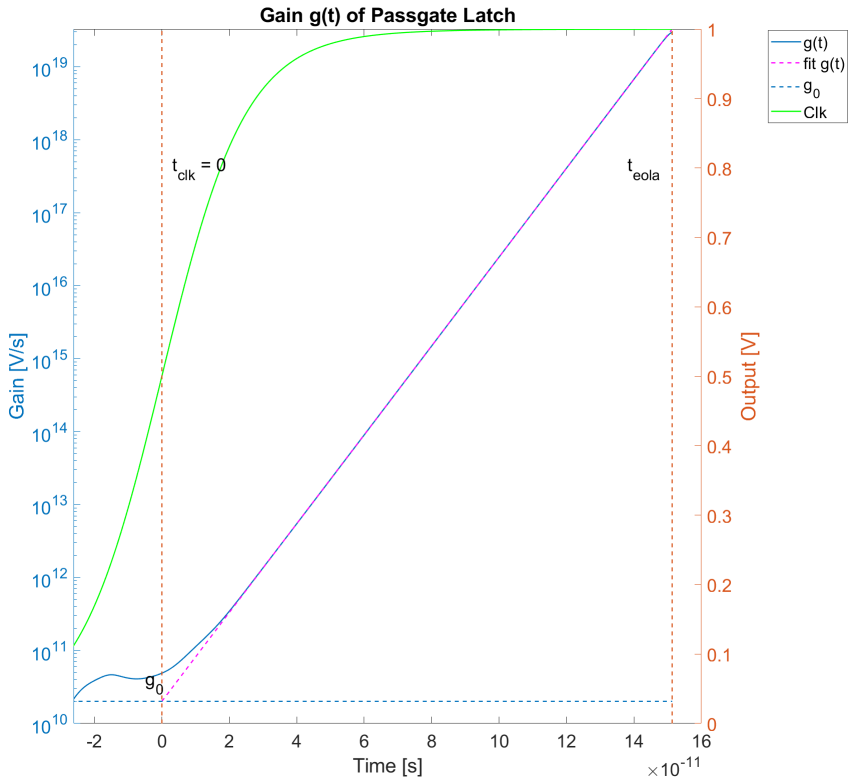

Figure 3.4: Linear fit line and plot of $\log g(t)$ vs. t for passgate latch

As described in Chapter 5, our ODE models for synchronizers are derived using modified-nodal analysis of the synchronizer circuit. This allows us to decompose the Jacobian $J_f(t)$, into the sum of contributions from each of the devices:

$$J_f(t) \quad = \quad \sum_{d \in devices} J_d(t) \tag{3.31}$$

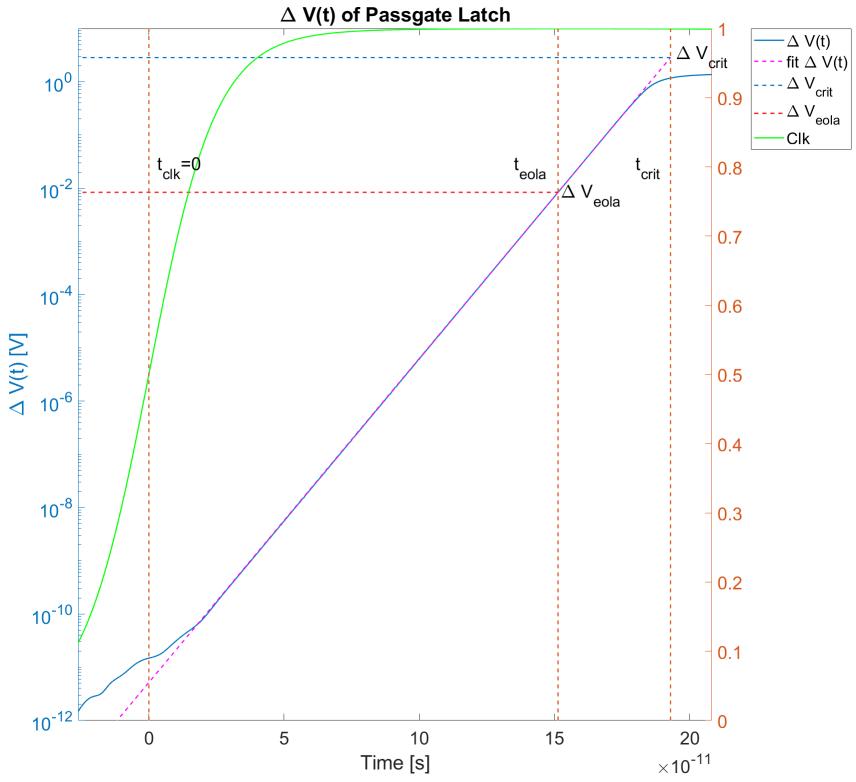

Figure 3.5: Linear fit line and plot of $\log \Delta V(t)$ vs. t for passgate latch

Therefore,

$$\begin{aligned} \lambda(t) &= u(t)^T J_f(t) u(t) \\ &= u(t)^T \left(\sum_{d \in devices} J_d(t) \right) u(t) \\ &= \sum_{d \in devices} u(t)^T J_d(t) u(t) \end{aligned} \tag{3.32}$$

If we have some set of "interesting" devices, e.g. a particular transistor, or the transistors forming a particular inverter, passgate or other structure, we get:

$$\lambda_{dev}(t) = \sum_{d \in dev} u(t)^T J_d(t) u(t) \tag{3.33}$$

Where *dev* is the set of interesting devices.

Define t_{homo} as described in Section 3.5: $\rho(t) \approx 0$ for $t \geq t_{homo}$. Then, $\dot{g}(t) \approx \lambda(t)g(t)$, which implies:

$$\begin{aligned} g(t) &= g(t_{homo}) \exp\left(\int_{t_{homo}}^{t} \lambda(s)ds\right) \\ \lambda(s) &= \sum_{d \in dev} \lambda_d(s) \\ g(t) &= g(t_{homo}) \prod_{d \in dev} \exp\left(\int_{t_{homo}}^{t} \lambda_d(s)ds\right) \\ g_d(t) &= \exp\left(\int_{t_{homo}}^{t} \lambda_d(s)ds\right) \\ g(t) &= g(t_{homo}) \prod_{d \in dev} g_d(t) \end{aligned} \tag{3.34}$$

We show in Chapter 6 how $\lambda_d(t)$ can be very helpful for understanding how particular circuit modifications impact synchronizer performance.

Note that for "most" components in the synchronizer (i.e. anything not in the first latch), $\lambda_d(t)$ is close to 0 for values of t in the neighborhood of the initial clock transition. Thus we can safely pick any reasonable value for t_{homo} for times after the first clock transition and characterize the impact of devices after the first stage of the synchronizer. However, analysis of the first stage is more complicated because the contribution of $\rho(t)$ cannot be neglected. The case studies in this thesis do not require the decomposition of components in the first stage.

Chapter 4

Automatic Differentiation Analysis

Differentiation plays a key role in synchronizer analysis methods developed in this thesis. This chapter examines and compares the three most common approaches to differentiation in scientific computing: symbolic, numerical differencing, and automatic differentiation. Section 4.1 and Section 4.2 present quantitative comparisons based on simple "toy" models, and we conclude that automatic differentiation provides a combination of numerical accuracy and practical implementation that make it our choice for synchronizer analysis. Section 4.3 examines issues that arise when using automatic differentiation in combination with numerical solutions to systems of non-linear equations, which is part of our synchronizer analysis when used with realistic transistor models. We present our solutions to these challenges.

Symbolic differentiation starts with the formula for a function, and uses standard methods from calculus to derive a formula for the derivative of the function with respect to some specified variable or variables. An appealing property of symbolic methods is that they are *mathematically exact*. Software packages such as Maple[29], Mathematica[19] and MATLAB's symbolic toolbox [32] are examples where symbolic mathematics and differentiation are used. For our purposes, the critical limitation of symbolic approaches is that we must have a formula to differentiate. Synchronizer analysis involves the solutions of non-linear, differential equations. Typically, such equations do not have a closed form solution; thus

there is no explicit formula available for subsequent differentiation. Fortunately, this saves us from having to go down the tangent of discussing whether or not symbolic methods are tractable for our work – they are simply inapplicable.

At the other extreme is approximating the derivative by finite differencing. The simplest approach uses the approximation:

$$\frac{d}{dx}f(x) \approx \frac{f(x+h)-f(x)}{h} \tag{4.1}$$

for some h sufficiently small, but not zero. The derivative estimated by equation Equation 4.1 is easily shown to have an error term that is $o(h^2)$. A simple improvement is to take a symmetric difference

$$\frac{d}{dx}f(x) \approx \frac{f(x+\frac{h}{2})-f(x-\frac{h}{2})}{h} \tag{4.2}$$

which has an error of $o(h^3)$. Better approximations can be achieved by evaluating f at more points, fitting a polynomial to the results, and differentiating the polynomial at the desired value of x. All of these formulations suffer from the trade-off that if h is too large, then higher-order derivatives of f corrupt the estimate of the first derivative. On the other hand, if h is too small, then we end up with a small-difference of large numbers (e.g. in the numerators of Equations Equation 4.1 and Equation 4.2). As a result, the estimate of $\frac{d}{dx}f(x)$ is inherently less accurate than the computation of $f(x)$ itself. We refer to these methods as being *numerically approximate*.

Automatic differentiation (AD, also known as "source code differentiation") provides a third approach. The basic idea is to compute both the value and the derivative for each term in an expression. Let x be a variable, and e, e_1 and e_2 be expressions. The observations are that

$$\begin{array}{rcl|rcl} \frac{d}{dx}x & = & 1 & \frac{d}{dx}e & = & 0, \text{ if } x \text{ does not appear in } e \\ \frac{d}{dx}(e_1+e_2) & = & \frac{d}{dx}e_1+\frac{d}{dx}e_2 & \frac{d}{dx}e_1 * e_2 & = & e_1\frac{d}{dx}e_2+e_2\frac{d}{dx}e_1 \\ \frac{d}{dx}\sin(e) & = & cos(e)\frac{d}{dx}e & & & \end{array} \tag{4.3}$$

and so on for other arithmetic operators and standard math library functions. In many programming languages, AD can be implemented by operator overloading.

The value for an expression, e, is now a tuple: the value and its derivative. If derivatives are taken with respect to several variables, then AD computes the value of e and a gradient vector. The number of floating point operations to compute the derivative is at most the number of operations required to evaluate e times the number of derivatives requested times five (because five operations are needed for division). Thus AD is reasonably efficient.

For our purposes, the biggest strength of AD is that, unlike numerical differencing, AD does not introduce any approximation errors beyond floating point round-off errors and the errors introduced by any approximations in evaluating the original expression (such as discretization for approximating an integral). We say that AD is *numerically accurate*.

For the reasons described above, AD is the workhorse for differentiation in this thesis. That noted, numerical differencing also had an important role in this research: code testing. We often had to do careful derivations to put the problem into a form that minimized the overhead of AD. Doing so produced several, big-$\mathcal{O}$ efficiency improvements. To check our approach, we compared the results of our optimized implementations with brute-force numerical differencing. There are many confounding issues in doing such checks including higher-order terms for numerical differencing, round-off errors, and the approximation errors of the numerical integrator. To better understand the accuracy issues for AD and numerical differencing I developed a model for a "toy" synchronizer. While this model does not correspond to a VLSI circuit, I designed the model to "act like a synchronizer" *and* to admit closed form solutions. This provides an exact reference that I use for comparing AD with numerical differencing in Section 4.1 and Section 4.2. In Section 4.3 we address one more implementation issue with AD: root solving algorithms. A naïve implementation of AD does not play nicely with iterative algorithms –"`if`" is not differentiable. Section 4.3 describes how I adapted the root solver in the MVS model to work with AD.

4.1 Numerical Differentiating Comparisons

In the synchronizer analysis of Chapter 3, we require the derivative of $\vec{V}(t, t_{in})$ with respect to t_{in}. We obtain $\vec{V}(t, t_{in})$ by numerical integration of our derivative func-

tion $\frac{d}{dt}\vec{V}(t,t_{in})$. In an attempt to verify our approach, we numerically estimated the derivative $\frac{\partial \vec{V}(t,t_{in})}{\partial t_{in}}$ through brute-force numerical differencing. Our expectation was that a close match would arise between our method and brute-force numerical differencing. However, a non-trivial difference remained between the two approximating methods was observed that appeared to be independent of our delta for t_{in} for a wide range of small deltas. Our prejudice is that the AD results are more accurate, but let's find out.

Because there is no known closed form solution for $\vec{V}(t,t_{in})$ for realistic models, I develop a toy example function $f(t,w)$ with known derivatives $\frac{d}{dt}f(t,w)$, $\frac{\partial}{\partial w}f(t,w)$ and outline five different ways to numerical approximate $\frac{\partial}{\partial w}f(t,w)$. The investigation aims to uncover and separate the sources of error introduced by *numerical differencing*, *Automatic Differentiation* and *numerical integration*. The five methods of approximating the derivative $\frac{\partial}{\partial w}f(t,w)$ are as follows:

- Method 1: If we assume that we have a formula for the function $f(t,w)$, then we can compute numerically accurate results by setting the parameter w as an AD variable and compute the tuple $\left(f(t,w), \frac{\partial}{\partial w}f(t,w)\right)$.

- Method 2: Following the assumption that we have the function $f(t,w)$, then we can compute numerically approximate results by a centered difference (CD):

$$\frac{\partial f(t,w)}{\partial w} \approx \frac{f(t,w+\Delta)-f(t,w-\Delta)}{2\Delta} \tag{4.4}$$

  The question arises of what value of Δ to use in order to minimize the error. If Δ is too large, then the higher order terms which are present in the approximation dominate the error. On the other hand if Δ is too small, then round-off errors are being amplified and thus dominate the error. Picking Δ to minimize the error in practice is often a best guess.

- Method 3: If we now assume that we only have the derivative function $\frac{d}{dt}f(t,w)$, then we can compute numerically approximate results by numeri-

cal integration of $\frac{d}{dt}f(t,w)$ for $w \pm \Delta$ followed by a centered difference:

$$\begin{array}{rcl} f(t,w+\Delta) & = & \int_0^t \frac{d}{dr} f(r,w+\Delta)dr \\ f(t,w-\Delta) & = & \int_0^t \frac{d}{dr} f(r,w-\Delta)dr \\ \frac{\partial f(t,w)}{\partial w} & \approx & \frac{f(t,w+\Delta)-f(t,w-\Delta)}{2\Delta} \end{array} \tag{4.5}$$

Once again, picking Δ is not obvious, furthermore in this method we note that numerical integration may produce $f(t,w \pm \Delta)$ to about 6 or 7 digits of accuracy and our derivative approximation is good to $o(\Delta^3)$ which means that our approximation reduces the accuracy by approximately another 3 digits. This approximation to the derivative is thus likely only 3 or 4 digits accurate. Higher order differencing methods can improve the errors introduced, but it illustrates the fact that numerically approximating derivatives compounds errors. Therefore to approximate further derivatives from already approximated derivatives becomes a dubious method to obtain derivatives upon derivatives.

- Method 4: Another approach is to numerically integrate $\frac{\partial}{\partial w}\frac{d}{dt}f(t,w)$ to get a numerically approximate result of $\frac{\partial}{\partial w}f(t,w)$. We assume that Equation 4.6 is integrable. For circuit models this is reasonable.

$$\frac{\partial}{\partial w}\int_0^t \frac{d}{dr} f(r,w)dr \quad = \quad \int_0^t \frac{\partial}{\partial w}\frac{d}{dr} f(r,w)dr \tag{4.6}$$

  Seeing as we know the formula for $\frac{d}{dt}f(t,w)$, we can write down a mathematically exact formula for $\frac{\partial}{\partial w}\frac{d}{dt}f(t,w)$ which we then numerically integrate. This approach has the advantage that computing $\frac{\partial}{\partial w}\frac{d}{dt}f(t,w)$ is numerically accurate, thus the only errors introduced are those produced by numerical integration.

- Method 5: Suppose that $\frac{d}{dt}f(t,w)$ is complicated and includes many secondary functions to compute $f(t,w)$ such as those found in transistor models, then writing down $\frac{\partial}{\partial w}\frac{d}{dt}f(t,w)$ may be difficult. Our last method is to compute a numerically approximate result as described in Method 4 by computing $\frac{\partial}{\partial w}\frac{d}{dt}f(t,w)$ via AD. We note that obtaining the tuple $\left(\frac{d}{dt}f(t,w), \frac{\partial}{\partial w}\frac{d}{dt}f(t,w)\right)$ via AD is numerically accurate. Thus, the value we get for $\frac{\partial}{\partial w}f(t,w)$ is nu-

merically approximate due to the numerical integration.

4.1.1 Example function $f(t,w)$

The typical scenario, such as in the synchronizer analysis, is that we have a derivative function $\frac{d}{dt}f$ and have no closed form solution for f. In order to evaluate all the approximating techniques, I want to compare the results against a known closed form solution to $\frac{d}{dt}f(t,w)$, where the dynamics of $\frac{d}{dt}f(t,w)$ are time varying. Let $f(t,w)$, $\frac{d}{dt}f(t,w)$, $\frac{\partial}{\partial w}f(t,w)$ and $\frac{\partial}{\partial w}\frac{d}{dt}f(t,w)$ be:

$$\begin{aligned} f(t,w) &= \tanh\left(wt+aw^2t\right) \\ \tfrac{d}{dt}f(t,w) &= \left(w+aw^2\right)\operatorname{sech}^2\left(wt+aw^2t\right) \\ \tfrac{\partial}{\partial w}f(t,w) &= \left(t+2awt\right)\operatorname{sech}^2\left(wt+aw^2t\right) \\ \tfrac{\partial}{\partial w}\tfrac{d}{dt}f(t,w) &= \left(1+2aw\right)\operatorname{sech}^2\left(wt+aw^2t\right) \\ &\quad -2\left(w+aw^2\right)\operatorname{sech}^2\left(wt+aw^2t\right)\tanh\left(wt+aw^2t\right)\left(t+2awt\right) \end{aligned} \tag{4.7}$$

In the above example if we let $\dot{x} = \frac{d}{dt}f(t,w)$ and $x = f(t,w)$ then using the identity that $\operatorname{sech}^2 x = 1-\tanh^2 x$ we get a non-linear differential equation of the form:

$$\begin{aligned} \dot{x} &= \left(w+aw^2\right)\left(1-\tanh^2\left(wt+aw^2t\right)\right) \\ &= \left(w+aw^2\right)\left(1-x^2\right) \\ \tfrac{\partial}{\partial w}\dot{x} &= \left(1+2aw\right)\left(1-x^2\right)-2\left(w+aw^2\right)x\tfrac{\partial x}{\partial w} \end{aligned} \tag{4.8}$$

4.1.2 Approximating $\frac{\partial}{\partial w}f(t,w)$

To begin evaluating our approximations to $\frac{\partial}{\partial w}f(t,w)$, we arbitrarily choose values for the parameters used for Equation 4.7 and Equation 4.8 as follows:

$$\begin{aligned} w &= 1.19 \\ a &= 0.65 \\ t &\in [0,10] \end{aligned} \tag{4.9}$$

The integrator used to perform the numerical integration is MATLAB's `ode45` integrator and we use a `'RelTol' = 'AbsTol'` $= 1\times10^{-6}$.

We start by addressing the issue of picking Δ for Methods 2 and 3. In order to

give the numerical methods the best results, we investigate the approximation error of Method 2 and 3 against the numerically accurate computation of $\frac{\partial}{\partial w} f(w,t)$. We summarize $\frac{\partial}{\partial w} f(t,w)$ by computing $||\frac{\partial}{\partial w} f(t,w)||_2$. We compare the approximating methods for $\frac{\partial}{\partial w} f(t,w)$ against $||\frac{\partial}{\partial w} f(t,w)||_2$ for different values of Δ to characterize the error. Figure 4.1 shows the error as we vary Δ. We can conclude from Figure 4.1 that Δ should be $\approx 1 \times 10^{-5}$ to minimize the error for approximating $\frac{\partial}{\partial w} f(t,w)$ with $f(t,w)$ and anywhere between 1×10^{-6} to 1×10^{-10} when approximating $\frac{\partial}{\partial w} f(t,w)$ with numerically integrated trajectories of $\frac{d}{dt} f(t,w)$.
Thus we pick $\Delta = 1 \times 10^{-6}$.

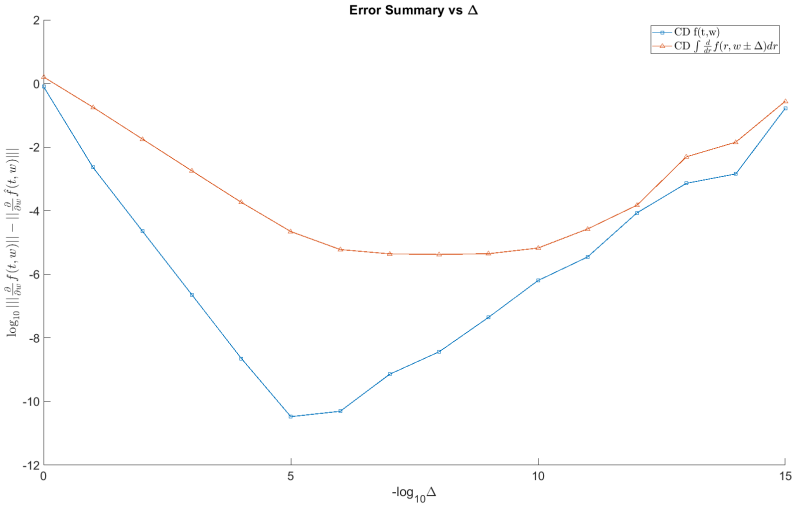

Figure 4.1: Error analysis with respect to Δ

With Δ selected, we proceed to evaluate the other three methods. Figure 4.2 shows all five approximating methods in addition to the numerically accurate computation of $\frac{\partial}{\partial w} f(t,w)$ from Equation 4.7. There is certainly no obvious discrepancy that appears in Figure 4.2; thus we investigate the error over time in Figure 4.3. I make several observations about Figure 4.3:

- Overall, the integration of $\frac{\partial}{\partial w}\frac{d}{dt} f(t,w)$ outperforms the centered differencing of numerically integrated results of $\frac{d}{dt} f(t,w \pm \Delta)$ for different values of Δ. The advantage is not so large that AD outperforms the centered differencing

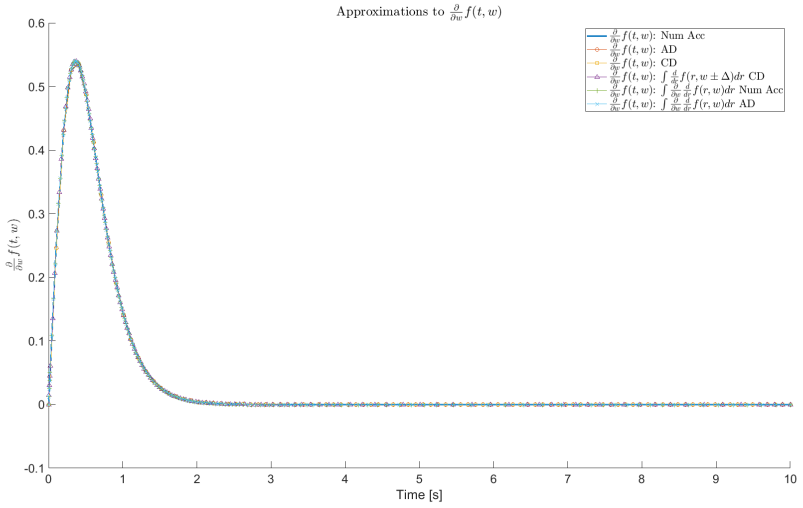

Figure 4.2: Approximating methods for $\frac{\partial}{\partial w} f(t,w)$

approach at every time point.

- Method 3 has errors which are not strongly sensitive to the choice of Δ. We show in Figure 4.3 the errors for $\Delta = 10^{-3}$, $\Delta = 10^{-6}$ and $\Delta = 10^{-9}$ and see minor differences. This suggests that choosing a good Δ may not be overly difficult. On the other hand, it's unlikely that Method 3 will match Method 5 by simply tuning Δ.

- We observe that Methods 4 and 5 produce the *same* results, i.e. they are indistinguishable from each other. The hand derived version of $\frac{\partial}{\partial w} \frac{d}{dt} f(t,w)$ and computing it via AD produce numerically equivalent results. Thus, even thought the two methods are evaluated with two separate calls to `ode45`, the sequence of time points and state values are identical for both calls.

- Using a centered difference numerical approximation is a better approximating method for $\frac{\partial}{\partial w} f(t,w)$ provided that we are given the function $f(t,w)$. The centered differencing approach on $f(t, w \pm \Delta)$ directly outperforms Methods 3 through 5. This further supports our hypothesis that the integrator is the largest cause of error.

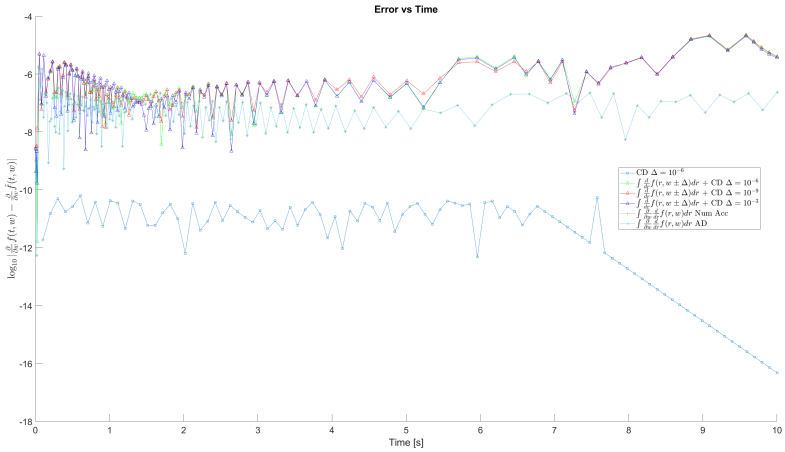

Figure 4.3: Approximating methods error $\log_{10}|\frac{\partial}{\partial w}f(t,w) - \frac{\partial}{\partial w}\hat{f}(t,w)|$

- Computing $\frac{\partial}{\partial w}f(t,w)$ via AD is numerically equivalent to the hand written implementation of $\frac{\partial}{\partial w}f(t,w)$. Comparing the two results agree to within a few digits of least precision, i.e. relative errors $< 10^{-15}$, and thus is much more accurate than centered differencing. Concluding that either AD or the hand-written derivative is better than the other is not a conclusion that one can make. Either result can be used as our measure to compare against. As a result, an error plot is not included. For consistency the hand written version of $\frac{\partial}{\partial w}f(t,w)$ is used.

The experiments performed here are preliminary. We observe that changing the simulation time span yields varying error results. Our experiments show that our AD methods overall outperform centered differencing. We also note that `ode45` produces more time points at the tail end of the simulation for Method 3. The errors observed in Methods 3 through 5 are sensitive to the settings applied for `'RelTol'` and `'AbsTol'`. Furthermore as we tighten `'RelTol'` the seperation of AD and centered differencing shrinks, however, AD always maintains an advantage. Over the course of running these experiments, these anomalies and unexplained behavior emerged. Further exploratory experiments are required to better understand the anomalous results. On the other hand, these experiments per-

sistently suggest that our initial belief that the AD approach has a higher accuracy as compared to numerical differencing is supported and that integrator error is the dominant error that arises in the approaches which use integration.

4.2 Linear Switched Dynamics Synchronizer

The example developed in Section 4.1 does not have a saddle such as in our synchronizers. In this section, I present a set of switched linear equations with behavior that qualitatively corresponds to a two latch synchronizer shown in Figure 4.4.

The system is broken into 4 phases and has two input variables x_0 and x_1 and two output variables q_0 and q_1. The outputs $q_0(t) = \tanh(a_2 x_0(t))$ and $q_1 = \tanh(a_2 x_1(t))$ bound the output of the toy synchronizer to values $[-1,1]$, whereas x_1 and x_2 are unbounded. To avoid unreasonably large results for x_1 and x_2, time is restricted to $t \in [0,1]$. The dynamics, solution and imposed conditions are broken into each phase individually shown in Table 4.1. In phase ϕ_0 a signal transition occurs at time t_{in} according to a function $din(t, t_{in})$. Analogous to a real synchronizer, t_{in} can drive the dynamics such that q_1 can be 0, -1 or +1 as $t \to \infty$. In this toy example $q_1 = 0$ would correspond to the toy synchronizer which is forever metastable.

Real synchronizers are made as a chain of latches. A latch is described as being either transparent or opaque. When a latch is opaque, the input has no effect on the latch's output. Conversely when a latch is transparent, then the latch's input has a direct effect on its output. As a latch transitions from being transparent to opaque the influence the input has on the output is also transitioning. Thus we can talk about the phases as modeling the behavior of the latches in a synchronizer.
In the terminology of latches:

- Phase ϕ_0 simulates latch 0 as being transparent and captures the input signal $din(t, t_{in})$.
- Phase ϕ_1 simulates both latch 0 and latch 1 as opaque, this phase simulates latch 0 transitional period. The transition can leave output $q_0 \in [-1,1]$. It is this transitional period which captures the possibility of latch 0 becoming metastable depending on latch 0's internal state at the time of the phase switch.

- Phase ϕ_2 simulates latch 1 as transparent and obtaining signal x_0 from latch 0. The signal x_0 of latch 0 is kept constant in this phase, but in a real synchronizer this would not be the case.

- Finally phase ϕ_3 simulates latch 1's transitional period, at which point our switched dynamics ends.

Mathematically, the phases can be described as: phase ϕ_0 is a tracking phase for the variable x_0 which is asymptotically approaching the value of $din(t, t_{in})$ while variable x_1 is staying constant at its initial value. In phase ϕ_1 variable x_0 is exponentially growing either positively or negatively depending on the sign of x_0 at the time we switch between phase ϕ_0 and ϕ_1. This has the effect of driving q_0 to either +1 or -1 in the limit. If $x_0 = 0$ at the phase switch, then $q_0 = 0$. Variable x_1 remains constant.

In phase ϕ_2, variable x_0 stays constant from where it left off at the switch between phase ϕ_1 and ϕ_2. Variable x_1 is asymptotically approaching the value of x_0, i.e. x_1 is converging to the value of x_0. Finally, in phase ϕ_3, x_0 stays constant and variable x_1 is growing exponentially which has the effect of driving q_1 to +1 or -1 depending on the sign of x_1 at the time of the switch between phase ϕ_2 and ϕ_3. If $x_1 = 0$ at the start of phase ϕ_3 then the $q_1 = 0$ forever. The switched dynamics described here captures the behavior of a saddle system and is cast in the terminology of a synchronizer.

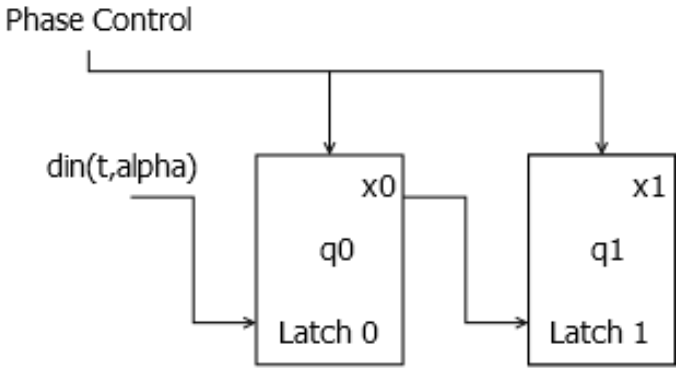

Figure 4.4: Two Latch Synchronizer

4.2.1 The Dynamics of the Toy Synchronizer

The conditions imposed on the constants C_x are to maintain continuity. Notice that the conditions on C_1 are a function of t_{in} which cascades throughout the rest of the switched dynamics. The variables a_0, a_1 and a_2 are parameters which dictate how fast the signals change in the switched dynamics model. As shown in Table 4.1, the simple dynamics of the toy synchronizer allow us to solve for x(t) in closed form.

Table 4.1: Switched Dynamics Toy Synchronizer Equations

Phase	Dynamics	Solutions
ϕ_0 :	$din(t,t_{in}) = \begin{cases} -1, & t \le t_{in} \\ 1, & t > t_{in} \end{cases}$	
$0 \le t \le t_{\phi_1}$	$\dot{x_0}(t) = a_0(din(t,t_{in}) - x_0(t))$	$x_0(t) = \begin{cases} C_0\exp(-a_0t) - 1, & t < t_{in} \\ C_1\exp(-a_0t) + 1, & t \ge t_{in} \end{cases}$
$t_{in} < t_{\phi_1}$	$\dot{x_1}(t) = 0$	$x_1(t) = C_2$
		$C_0 = x_0(0) + 1$
		$x_0(t_{in}) = C_0\exp(-a_0t_{in}) - 1$
		$C_1 = \frac{x_0(t_{in})-1}{\exp(-a_0t_{in})}$
		$C_2 = x_1(0)$
ϕ_1 :	$\dot{x_0}(t) = a_1x_0(t)$	$x_0(t) = C_3\exp(a_1t)$
$t_{\phi_1} < t \le t_{\phi_2}$	$\dot{x_1}(t) = 0$	$x_1(t) = C_4$
		$C_3 = \frac{x_0(t_{\phi_1})}{\exp(a_1t_{\phi_1})}$
		$C_4 = x_1(t_{\phi_1})$
ϕ_2 :	$\dot{x_0}(t) = 0$	$x_0(t) = C_4$
$t_{\phi_1} < t \le t_{\phi_3}$	$\dot{x_1}(t) = a_0(x_0(t) - x_1(t))$	$x_1(t) = C_5 + C_6\exp(-a_0t)$
		$C_5 = x_0(t_{\phi_2})$
		$C_6 = \frac{x_1(t_{\phi_2})-C_5}{\exp(-a_0t_{\phi_2})}$
ϕ_3 :	$\dot{x_0}(t) = 0$	$x_0(t) = C_7$
$t_{\phi_3} < t$	$\dot{x_1}(t) = a_1x_1(t)$	$x_1(t) = C_8\exp(-a_1t)$
		$C_7 = x_0(t_{\phi_3})$
		$C_8 = \frac{x_1(t_{\phi_3})}{\exp(-a_1t_{\phi_3})}$

4.2.2 Derivation of the Gain of the Toy Synchronizer

Chapter 3 describes the gain of a synchronizer as a measurement of how the state of the synchronizer changes as a result of a change to when its input signal transition occurs. i.e. $\beta(t) = \frac{\partial \vec{V}(t)}{\partial t_{in}}$. For this toy synchronizer, the function $\beta(t)$ can be solved in closed form as summarized in Table 4.2. The derivation for $\beta(t)$ is with the initial conditions of $x_0(0) = x_1(0) = -1$. Recall that C_1 is a functions of t_{in} and the coefficient in the derivation of Table 4.2 would change based on the value of $x_0(t_{in})$. In this derivation $x_0(t_{in}) = -1$ which means that $\beta(t_{in}) = \begin{bmatrix} -2a_0 \\ 0 \end{bmatrix}$.

Table 4.2: Derivation of Toy Synchronizer Gain $\beta(t)$ with initial conditions $x_0(0) = x_1(0) = -1$

Phase	$\beta(t)$
$\phi_0 : 0 \leq t < t_{\phi_1}$	$\beta_0(t) = \begin{cases} 0, & t < t_{in} \\ -2a_0 \exp\left(a_0(t_{in} - t)\right), & t \geq t_{in} \end{cases}$ $\beta_1(t) = 0$
$\phi_1 : t_{\phi_1} \leq t < t_{\phi_2}$	$\beta_0(t) = -2a_0 \exp\left(a_0\left(t_{in} - t_{\phi_1}\right) + a_1\left(t - t_{\phi_1}\right)\right)$ $\beta_1(t) = 0$
$\phi_2 : t_{\phi_2} \leq t < t_{\phi_3}$	$\beta_0(t) = -2a_0 \exp\left(a_0\left(t_{in} - t_{\phi_1}\right) + a_1\left(-t_{\phi_1} + t_{\phi_2}\right)\right)$ $\beta_1(t) = \beta_0(t) + 2a_0 \exp\left(a_0\left(t_{in} - t_{\phi_1} + t_{\phi_2} - t\right) + a_1\left(-t_{\phi_1} + t_{\phi_2}\right)\right)$
$\phi_3 : t_{\phi_3} \leq t$	$\beta_0(t) = \beta_0(t_{\phi_3^-})$ $\beta_1(t) = \Big(\beta_0(t) + 2a_0 \exp\Big(a_0\left(t_{in} - t_{\phi_1} + t_{\phi_2} - t_{\phi_3}\right) + a_1\left(-t_{\phi_1} + t_{\phi_2}\right)\Big)\Big) \exp\left(a_1\left(-t_{\phi_3} + t\right)\right)$

4.2.3 The Results

As in Section 4.1, we evaluate our AD approach against a centered difference numerical approximation of $\beta(t)$. Given that we have a piecewise equation for $f(t, t_{in}) = \dot{x}(t, t_{in})$ from Table 4.1, we can compute the Jacobian of $f(t, t_{in})$ as $J_f(t) = \frac{\partial f(t, t_{in})}{\partial x}$ via AD, and solve the differential equation $\dot{\beta}(t) = J_f(t)\beta(t)$ with an

initial value of $\beta(0) = 0$. Therefore $\beta(t) = \int_0^t J_f(w)\beta(w)dw$. We can also numerically approximate $\beta(t) \approx \frac{x(t,t_{in}+\Delta)-x(t,t_{in}-\Delta)}{2\Delta}$ by numerical integration of $\dot{x}(t,t_{in} \pm \Delta)$. The parameters used for the toy synchronizer are as follows:

$$\begin{aligned} t_{in} &\approx 0.12336 \\ t_{\phi_1} &= 0.25 \\ t_{\phi_2} &= 0.50 \\ t_{\phi_3} &= 0.75 \\ a_0 &= a_1 = a_2 = 6 \end{aligned} \tag{4.10}$$

Numerical integration is performed using MATLAB's `ode45` integrator with `'RelTol'` = `'AbsTol'` = 1×10^{-6}. Figure 4.5 shows the resultant trajectory for x_0, x_1, q_0 and q_1. A simple bisection on t_{in} is performed such that $q_1(t = 1) \leq 1 \times 10^{-2}$. The values for $t_{hi} \approx 0.12333$ and $t_{lo} \approx 0.12338$ which translates to $\Delta \approx 2.5 \times 10^{-5}$.

The numerically accurate derived solution, the numerically integrated one via AD and the numerically approximate result for $\beta(t)$ are shown in Figure 4.6. Figure 4.7 shows the absolute error between the numerically accurate solution and the two approximations. The max absolute error $\max_{err} \log_{10} |||\beta(t) - \hat{\beta}(t)||_2| \approx 0.7$ for the centered difference approximation and ≈ -5 for the AD method. The AD method has an absolute error about an order of magnitude worse than what we set for MATLAB's `ode45` error tolerance, however the numerical approximation is several orders of magnitude worse when compared to the AD method. I want to also observe what happens as $t \to \infty$. I simulated the toy synchronizer on a time interval $t \in [0, 100]$ to let $\beta(t)$ grow exponentially. Because $\log_{10} ||\beta(t)||$ is growing exponentially, I compute the relative error between $\beta(t)$ and my approximation method $\hat{\beta}(t)$. Figure 4.8 shows the relative error with respect to the numerically accurate derivation of $\beta(t)$ and $\hat{\beta}(t)$. I also show in Figure 4.8 the $\log_{10} ||\beta(t)||$ as a reference to show the magnitude of $\beta(t)$.

We observe that the relative error of our AD method is slowly growing, however it is several orders of magnitude more accurate as compared to the centered difference approximation. We also observe that $\log_{10} ||\beta(t)|| \approx 250$ and our AD method is able to achieve a relative error of $\approx 1 \times 10^{-4}$ while the numerical approximation has a relative error of $\approx 1 \times 10^{-1}$ to $10^{-1.5}$ throughout.

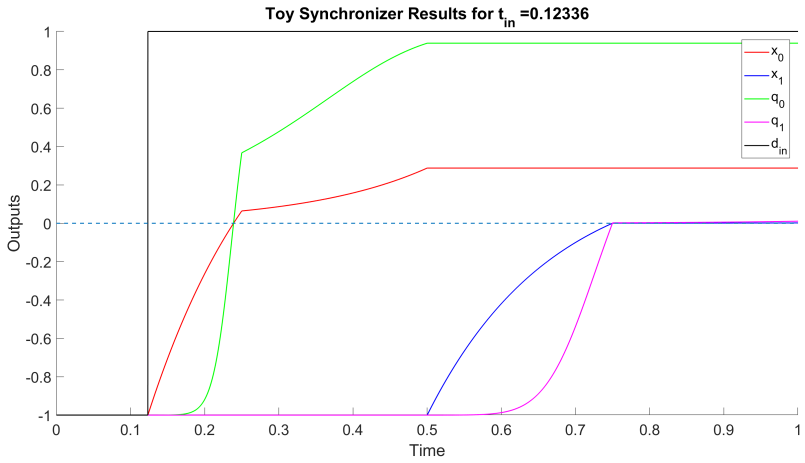

Figure 4.5: Two Latch Toy Synchronizer Simulation Results

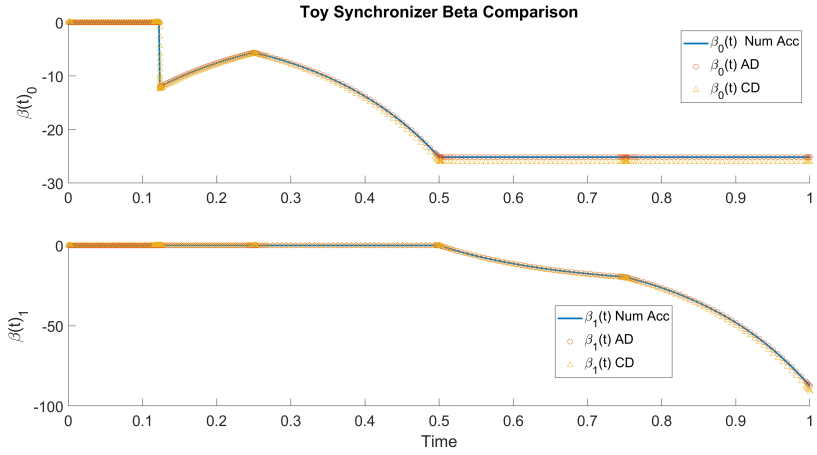

Figure 4.6: Two Latch Toy Syncronizer $\beta(t)$ Simulation Results

The experiments performed in Section 4.1 and Section 4.2 suggests that our initial belief in the AD methods of approximating derivatives from Chapter 3 do indeed have a higher degree of accuracy as compared to brute-force numerical approximating methods. This is not a proof and we note that higher order differentiating approximation schemes may have comparable results to our AD methods. Preliminary tests of our analysis tool from Chapter 3 using the non-linear transistor

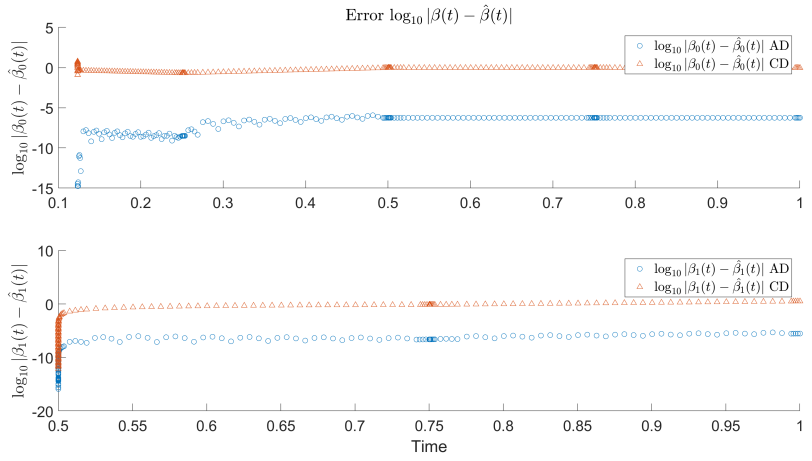

Figure 4.7: Two Latch Toy Synchronizer $\beta(t)$ Error: $\log_{10}|\beta(t)-\hat{\beta}(t)|$

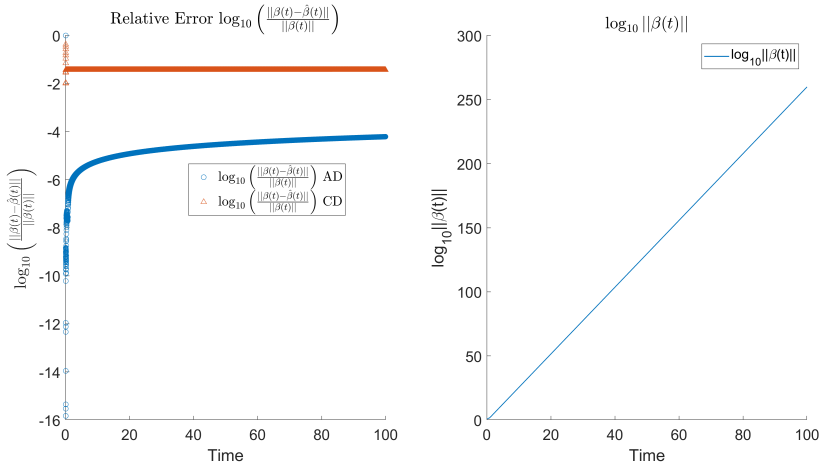

Figure 4.8: Relative error analysis $\log_{10}\frac{||\beta(t)-\hat{\beta}(t)||}{||\beta(t)||}$ as $t\to\infty$,i.e. $\beta(t)$ exponentially growing.

models described in Chapter 5, showed that a higher order numerical differentiating scheme produced results between the centered differencing scheme explored here and our AD method. We can conclude from the experiments in this chapter that our AD method outperform the centered difference approach.

4.3 Derivative of Solution to an Iterative Algorithm

The MVS model described in Chapter 5 includes an iterative solver to obtain internal quantities to the model. The focus of this section is to present a solution to the problem of computing the derivative(s) of the result(s) obtained from an iterative solver with respect to its input(s).

Iterative algorithms are problematic with AD because they involve a series of conditional steps. An `if-cond-then-else` construction is not differentiable at the point(s) where `cond` changes. Often, AD is used ignoring such details because it is correct in most places, or because the derivative of the `then` and `else` terms match at the point(s) where `cond` changes. Iterative algorithms are approximating fix points; conceptually, the algorithm produces the limit solution to an infinite number of iterations. If each iteration involves non-differentiable terms such as `if`, then there can be an infinite number of such discontinuities, and as a result, a direct application of AD is meaningless.

In particular to the MVS iterative solver, we want to compute $\frac{\partial I_{ds}}{\partial \vec{V}_{tx}}$, where $\vec{V}_{tx}$ are the terminal voltages of the transistor in question and I_{ds} is its current flowing from the drain to the source. The problem is that I_{ds} depends on the values obtained from the iterative solver for the virtual source and drain which are internal variables to the MVS model.

The concept can be generalized by considering a function $z = f(x,y)$ where we have some way of finding y such that $f(x,y) = 0$. Then we want to find $\frac{\partial y}{\partial x}$. This section presents our method and demonstrates how it applies to the MVS model covered in Chapter 5.

4.3.1 Fixed point algorithm $f(x,y) = 0$

With several assumptions, we present a solution to compute $\frac{\partial y}{\partial x}$ for a fixpoint algorithm f which for a given input x finds y such that $f(x,y) = 0$. We note that a full treatment of AD for fixpoint algorithms is beyond the scope of this thesis. We assume that $f(x,y) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$, i.e. y and the range of f have the same dimensions. In the case of our non-linear function solver for circuit model evaluation, this restriction holds. Thus $\frac{\partial y}{\partial x} \in \mathbb{R}^{m \times n}$. We will also assume that there is

some open neighborhood of x, $X \subseteq \mathbb{R}^n$ such that:

$$\forall x \in X.\ \exists y \in \mathbb{R}^m.\ f(x,y) = 0 \tag{4.11}$$

Let $g(x) : \mathbb{R}^m \to \mathbb{R}^m$ be a differentiable function such that $\forall x \in X.\ f(x,g(x)) = 0$. Therefore the total derivative of f with respect to x is:

$$\frac{d}{dx} f(x,g(x)) \quad = \quad \left[\frac{\partial}{\partial x} f(x,y)\right]_{y=g(x)} + \left[\frac{\partial}{\partial y} f(x,y)\right]_{y=g(x)} \frac{\partial}{\partial x} g(x) \tag{4.12}$$

Because we assume that $x \in X$, $f(x,g(x)) = 0$ then Equation 4.12 becomes:

$$\begin{array}{rcl} \frac{d}{dx} f(x,g(x)) & = & 0 \\ \left[\frac{\partial}{\partial y} f(x,y)\right]_{y=g(x)} \frac{\partial}{\partial x} g(x) & = & -\left[\frac{\partial}{\partial x} f(x,y)\right]_{y=g(x)} \end{array} \tag{4.13}$$

We assume that:

$$\left[\frac{\partial}{\partial y} f(x,y)\right]_{y=g(x)} \tag{4.14}$$

is non-singular. Then by multiplying both sides of Equation 4.13 by the inverse of the above Jacobian matrix yields:

$$\frac{\partial}{\partial x} g(x) \quad = \quad -\left[\frac{\partial}{\partial y} f(x,y)\right]^{-1}_{y=g(x)} \left[\frac{\partial}{\partial x} f(x,y)\right]_{y=g(x)} \tag{4.15}$$

Which gives us a way to compute $\frac{\partial y}{\partial x}$ given our assumptions.

4.3.2 Derivative of VS in MVS Model

The MVS model described in Chapter 5 Section 5.2 has an iterative solver in its transistor model. More details on the MVS model can be found in Section 5.2, but here I focus on the iterative solver that computes the virtual source and drain of the model. These internal nodes dictate the I_{ds} current flowing in the transistor (refer to Figure 5.9) and it depends on $\vec{V}_{tx}$, the bias voltages applied to the terminal nodes of the transistor, i.e. $\vec{V}_{tx} = \left[drain \quad gate \quad source \quad body\right]$. The iterative solver within the MVS model takes $\vec{V}_{tx}$ and an initial guess I_0 (a guess for the current flowing from the source to drain) in order to find a solution for the internal virtual source and drain. The MVS implementation computes I_0 based on the user supplied $\vec{V}_{tx}$

input. We define a new function $I_{ds0}(\vec{V}_{tx}, I_0)$, which computes the virtual source and drain quantities based on the supplied guess I_0 without the iterative solver. From those results we then computes the resultant I_{ds} current. Let

$$f(\vec{V}_{tx}, I_0) \quad = \quad I_0 - I_{ds0}(\vec{V}_{tx}, I_0) \tag{4.16}$$

In order to compute the Jacobian of I_{ds}, i.e. $\frac{\partial I_{ds}}{\partial \vec{V}_{tx}}$, we compute I_{ds} as described by the MVS model with an input $\vec{V}_{tx}$. We then compute $f(\vec{V}_{tx}, I_{ds})$ where I_{ds} is the current we just computed using the original solver. The way in which we define I_{ds0} means that computing $I_{ds0}(\vec{V}_{tx}, I_{ds})$ effectively computes a new $\tilde{I}_{ds}$ with one more round of the original iterative solver. The assumption is that the result of $I_{ds} - I_{ds0}(\vec{V}_{tx}, I_{ds})$ is near zero. Therefore if we let $x = \vec{V}_{tx}$ and $y = I_{ds}$ and follow the steps in Section 4.3.1, then $\frac{\partial y}{\partial x}$ is the desired Jacobian: $\frac{\partial I_{ds}}{\partial \vec{V}_{tx}}$. The assumption that $I_{ds} - I_{ds0}(\vec{V}_{tx}, I_{ds})$ is near zero is validated in Figure 4.9 by testing an NMOS device with a sinusoidal input. The results show that the computed residual is on the order of 10^{-14}. Also shown in Figure 4.9 is the computed Jacobian elements for the transistor.

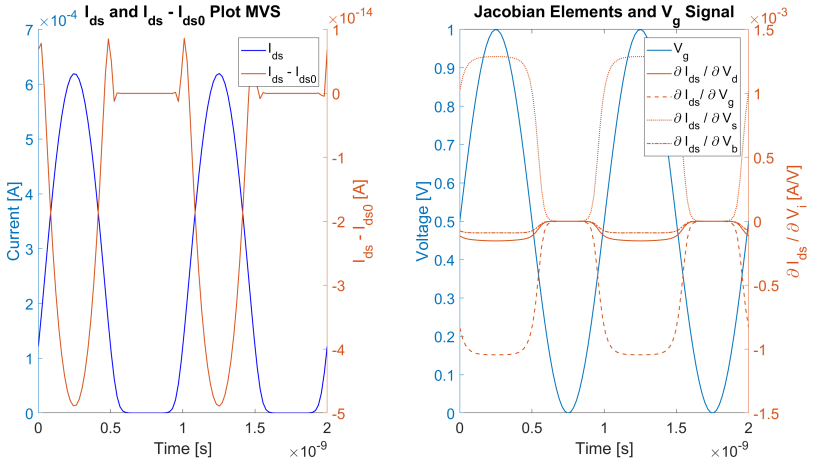

Figure 4.9: $I_{ds} - I_{ds0} = 0$ Assumption for an NMOS device in the MVS model and Jacobian Elements for a Sinusoidal Input Signal

Chapter 5

Transistor and Circuit Modeling

At the heart of any circuit simulator is the models that describe how the devices behave. Of importance in this thesis are the device models for Metal Oxide Semiconductor Field Effect Transistor (MOSFET)s, often abbreviated as MOS devices. There are two kinds of MOS devices, the PMOS and NMOS devices.

SPICE is generally regarded as the gold standard for VLSI circuit simulation. SPICE was first released in the 1970s [37], and has been updated many times. These updates include frequent releases of new and revised device models. As transistors have gotten smaller, a growing number of effects and parameters are added to model the correct physical and observed behavior of MOS devices.

Today the SPICE models are generally based on the BSIM models [16]. In this thesis, I exclusively use the BSIM4 model which has on the order of a 100 parameters. There are other BSIM models such as the BSIM-Common Multi-Gate (CMG) models which are used to simulate FinFET technology[7]. These are not discussed in this thesis. Despite the accuracy the BSIM4 model offers, it is such a large code base that it makes it cumbersome to use with AD packages.

For this reason, this chapter briefly introduces the main ideas behind short-channel transistor modeling and outlines two different model implementations: the MVS model [23][48] and a model developped by C. C. Enz, F. Krummenacher and E. A. Vittoz known as the EKV model [13]. These MOS models are used to simulate synchronizer behavior in subsequent chapters.

The MVS model has approximately 20 parameters and derived from physical

principles and is known as a *source* referenced model, while the simplified EKV model I develop in this thesis has 6 parameters and is known as a *body* referenced model.

This chapter describes how the model parameters are fit and how their respective capacitance model is derived. This chapter may be skipped if the reader is comfortable with transistor models or does not wish to dive into the inner details of MOS modeling. The subsequent simulations in future chapters make use of these models but an understanding of their inner workings is not required. It is sufficient to know that the current flowing out of a circuit node for a particular device is a function of the transistor's terminal voltages, i.e. $I_{ds}(V_d, V_g, V_s, V_b)$ and that each device has an associated capacitance matrix that models the capacitance between each pair of device terminal nodes. The compact models have been vectorized for use in MATLAB and fully differentiable for all combinations of node voltages for friendly use with AD.

5.1 Transistor Modeling Basics

Before diving into the MVS or EKV transistor models, we can talk about transistors in a general way. The models are specific implementations of the following details, but their underlying principles are similar. The discussion on modeling will only consider short channel MOSFET devices. The models are fit with empirical data gathered from LTSPICE [9] and ngSPICE[47]. This section outlines the main effects for short channel MOSFET devices as background for the implementations of the MVS and EKV models.

5.1.1 Transistor Construction

This section is a brief and simplified summary of transistor construction. This is a deep field in of itself, but in order to understand the effects that are being modeled, a picture to illustrate the construction of a transistor is helpful. A MOS device has 4 terminals: the *drain*, *gate*, *source* and *body*. The gate of a transistor is separated from the rest of the transistor by a thin, insulating layer. For simplicity we assume that no DC current flows into the gate terminal, e.g. we are neglecting quantum tunnelling through the gate-insulator. The area directly underneath the

gate is called the channel. The gate terminal of a transistor is the control which determines how open or closed the channel is, and current flows between the source and drain through the channel.

MOS devices come in two types, PMOS and NMOS shown in Figure 5.1. The difference between the two is the doping of the source, drain and channel. Doping in this context means injecting impurities into the silicon which change the conducting characteristics of the silicon in that region. N-doping introduces extra electrons into the conduction band to allow a flow of current. Conversely, P-doping results in an absence of electrons in the valence band. These "missing" electrons are called "holes", and these holes move around the semiconductor crystal structure as if they were positively charged. In a NMOS device, the channel is p-doped and the source and drain terminals are n-doped and vice versa for a PMOS device. When the gate terminal of an NMOS device is biased positively it attracts a layer of electrons (negative charge) that sit under the gate oxide. Depending on the source and drain terminal bias voltages, current either flows from the drain to the source, or the source to the drain. This current is called I_{ds} where a positive current is defined to be current flowing from the drain to the source and a negative current would be the current flowing the other way (see Figure 5.2). Similarly for a PMOS device, except that the gate is negatively biased to attract a layer of holes (positive charge) under the gate oxide.

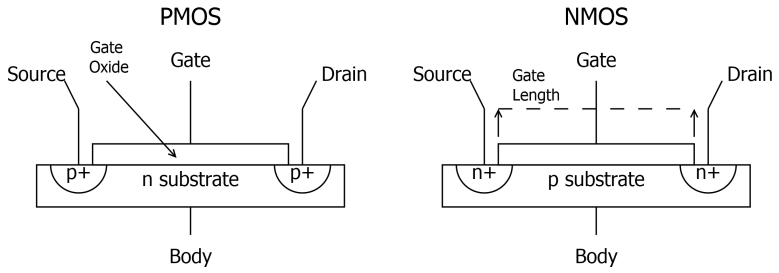

Figure 5.1: PMOS and NMOS simplified device construction

The NMOS and PMOS devices sit on top of what is called the bulk/substrate (or well) which is either doped p for NMOS devices or n for PMOS devices. The 4th and last terminal in a MOSFET devices is called the body terminal which connects to

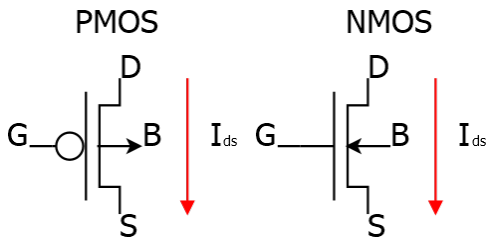

Figure 5.2: PMOS and NMOS device symbols and I_{ds} current

this substrate. For all subsequent discussions the substrate is connected to *gnd* for the NMOS devices and the supply voltage V_{dd} for the PMOS devices as is common for digital circuits.

The geometry of a transistors dictates how much current can flow in the channel and how large its associated parasitic capacitances are. The "process size" refers to the minimum feature size of the process. So a 45nm process would mean that the smallest feature in that process is 45nm. For all discussions in this thesis, it will be assumed that the transistor gate length will be the size of the process, i.e. 45nm would be the length for the transistor's gate unless otherwise specified.

The transistor width however is varied to change the parasitic capacitance and amount of current the device in question can provide. For fitting purposes a $10:1$ ratio is used, that is to say that the width is $10\times$ the length of the transistor.

For SPICE simulations, MOS devices parameters are provided via a model card. These model card parameters come from the process by which the devices are made and include the appropriate values to accurately simulate the device based on observed measurements. These model cards are generally proprietary, and require non-disclosure agreements in order to use them in the development of microelectronic circuits. Thankfully, researchers at the University of Arizona have came up with a set of open-source models. These model cards are called the PTM[20] model. The PTM model cards are the ones used to generate simulated measured transistor data from LTSPICE[9] and ngSPICE[47]. Transistor model fitting in this thesis is derived from the PTM 45nm High Performance (HP) model card. There are many others to choose from.

5.1.2 IV Characteristics

To characterize a function for the I_{ds} current flowing in a transistor, there are a number of tests performed to get current-voltage (IV) characteristic curves. These measured data points are then used to find the appropriate parameters to the transistor model which best describe those curves. There are two kinds of IV characteristic curves that are important to the behavior of MOS devices.

The first is to understand how the channel current I_{ds} changes as a function of the drain-source voltage $V_{ds} = V_d - V_s$ while holding $V_{gs} = V_g - V_s$ constant. The second is to understand how I_{ds} changes as a function of the gate-source voltage $V_{gs} = V_g - V_s$ while holding V_{ds} constant.

For the purposes of demonstration, the process will be described for a PMOS device with the PTM 45nm HP model card. Classic textbooks such as [49] will generally describe the process for an NMOS device. The experimental setup is shown in Figure 5.3. Three voltage sources are placed for V_g, V_d and V_s. The body terminal V_b is connected to $V_s = V_{dd}$ for a PMOS device.

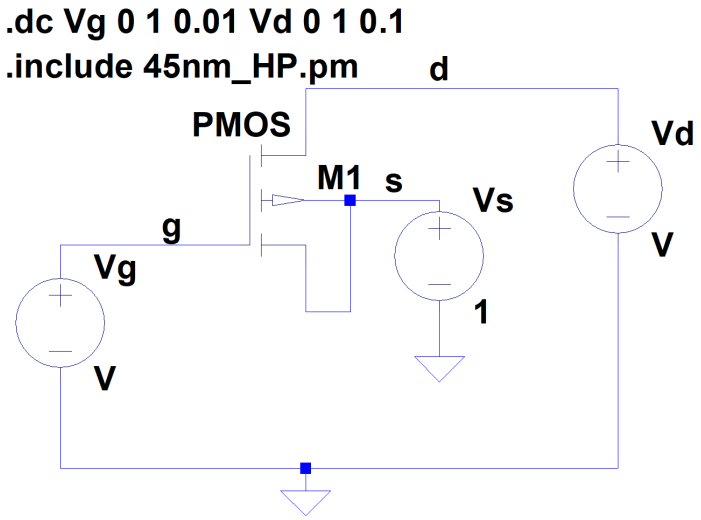

Figure 5.3: PMOS experiment test setup for the PTM 45nm HP model card

The 45nm HP process is a 1.0V process which means that the supply voltage V_{dd} is 1.0V. The 1.0V supply voltage will be carried out throughout subsequent

simulations. In the PMOS device, the source terminal is connected to V_{dd}, in an NMOS experiment the test setup remains the same but the source terminal gets connected to *gnd*. Eleven different values for V_g are chosen, starting from 0V to 1V in increments of 0.1V. Notice that V_{gs} is in fact negative and goes from -1.0V to 0V. V_d is swept from 0 to 1 in increments of 0.01V for each value of V_{gs}. Again note that V_{ds} sweeps from -1.0V to 0V and so I_{ds} is negative as shown in Figure 5.4.

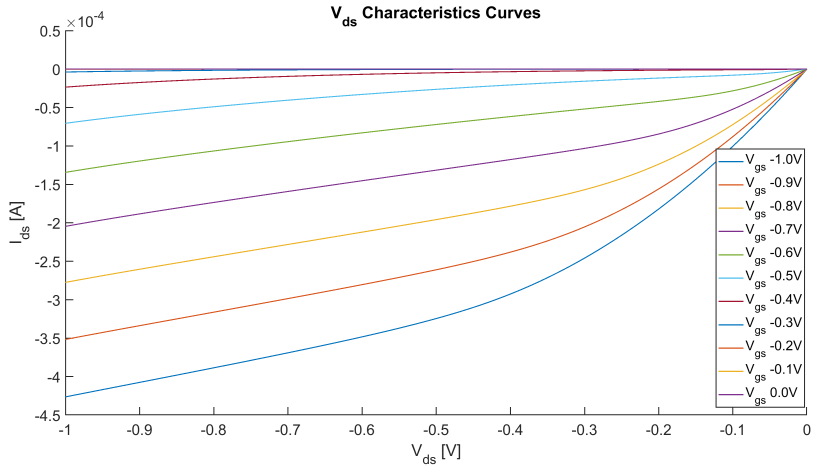

Figure 5.4: PMOS V_{ds} sweep for $V_{gs} = [-1, 0]V$ in increments of 0.1V

The other IV characteristics curve is when V_d is kept constant and V_g is varied for various values of V_d. Shown in Figure 5.5 is the SPICE simulation results for sweeping V_g in 0.01 increments for V_d of 1.0 to 0 in 0.1 steps.

The SPICE simulation data from Figure 5.4 and Figure 5.5 is saved, formatted and used as the dataset for model fitting.

5.1.3 Transistor Effects

Transistors simplistically are devices that control current flow from the drain to source via the gate. Well known physical properties that arise from short-channel transistor construction get incorporated into state-of-the-art transistor models. Introducing these parameters helps the model fitter have more degrees of freedom to better describe the IV characteristics discussed in Section 5.1.2. In this section, the main effects are discussed, but by no means is exhaustive. The goal is to be able to

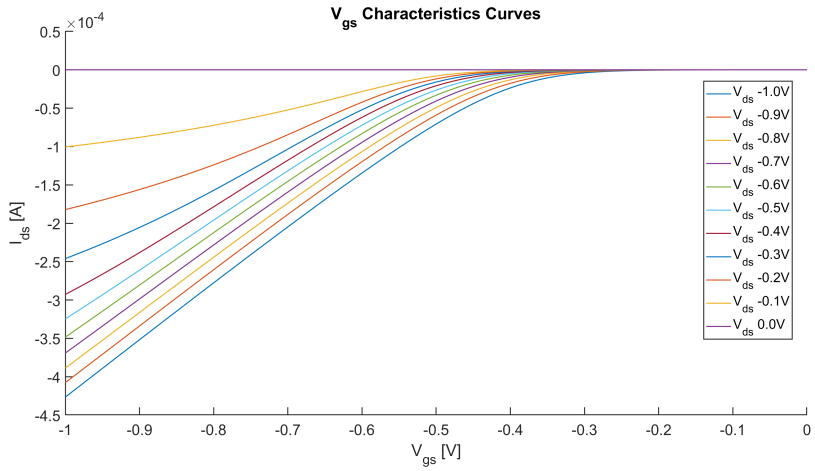

Figure 5.5: PMOS V_{gs} sweep for $V_{ds} = [-1.0, 0]V$ in increments of 0.1V

understand the main effects which contribute to the observations from SPICE.

Velocity Saturation

Velocity saturation is the first main point to understand in short-channel transistor models. Velocity saturation of the carriers means that under the electric field that is applied at either the drain or source node of the transistor causes the charge carriers to reach their maximum speed in the channel. When this occurs the terminology in literature[35] says that the semiconductor is velocity saturated. This has implications on the IV characteristics shown in Section 5.1.2. It is not a terrible assumption to just assume that the charge carriers are always velocity saturated in short channel devices. Although there is some loss of accuracy when the electric field across the drain and source is sufficiently low.

Body Effect, Threshold Voltage and Pinchoff Voltage

The threshold voltage for a transistor is the minimum gate voltage which allows current to flow from the drain to source. While the pinchoff voltage is the voltage at which the transistor transitions from the linear region to the saturation region.

The body effect in a transistor is the effect present due to the bulk/substrate/well/body which changes the effective threshold voltage of a transistor and has the fol-

lowing form[49]:

$$V_{th} = V_{th0} + \gamma\left(\sqrt{\phi_s + (V_s - V_b) \times type} - \sqrt{\phi_s}\right) \tag{5.1}$$

The terms in Equation 5.1 are V_{th0} a fit parameter to describe the threshold voltage in $[V]$, ϕ_s the surface potential which is a physical property of the device in units of $[V]$ and γ – the body effect coefficient in units of $[\sqrt{V}]$. NMOS devices get a *type* parameter of 1 while PMOS devices get a *type* of -1 because the body is connected to V_{dd}.

Note that the body effect does not impact the model for circuits such as inverters, where the sources of NMOS devices are connected to *gnd* and the sources of PMOS devices are connected to V_{dd}. On the other hand, our analysis tool shows that the body effect has an impact on circuits such as the passgates in a passgate latch where the source terminals of the transistors in the passgate are connected to the outputs of inverters. The inverter output can be at an intermediate value between *gnd* and V_{dd}, especially when a latch is in a metastable condition.

Drain Induced Barrier Lowering (DIBL)

Drain Induced Barrier Lowering (DIBL) is a significant effect in short channel devices. This is the effect of the bias voltage on the drain node and its influence on the transistor effective threshold voltage. An interesting observation is that without this term, the synchronizers considered in this thesis would have very high failure rates. DIBL is essential to ensure that the inverters of a metastable latch are biased into their high gain regions rather than being "stuck" with all transistors in their cut-off regions.

Channel Length Modulation

Channel length modulation models how the the reverse bias of the drain-to-channel junction creates a depletion region that effectively reduces the length of the overall channel. This increases the conductance of the device.

5.1.4 Device Capacitance

The most dominant capacitance for a MOS device is the parallel plate capacitor created from the geometry of the transistor. C_{ox} is the capacitance from the gate insulating oxide and channel which is measured in capacitance per units area of the device $[F/m^2]$. The total parallel plate capacitance C_{ox} is distributed to the drain, source and gate nodes according to the voltages on the terminals.

The distribution of the capacitance gives rise to the Miller effect which is due to the capacitance between the gate and drain/source nodes and lastly there is the *junction capacitance* at the source and drain which also contribute to the overall capacitance of the MOS device. To get the capacitance from the transistor model directly, it is necessary to be able to compute:

$$C_{ij} = \begin{cases} -\frac{\partial Q_i}{\partial V_j}, & i \neq j \\ \frac{\partial Q_i}{\partial V_j}, & i = j \end{cases} \tag{5.2}$$

If the transistor model provides a charge model, then the capacitance matrix can be automatically obtained using automatic differentiation. The full capacitance Matrix for any individual device is then:

$$\mathbf{C}_{Tx} = \begin{bmatrix} C_{dd} & C_{dg} & C_{ds} & C_{db} \\ C_{gd} & C_{gg} & C_{gs} & C_{gb} \\ C_{sd} & C_{sg} & C_{ss} & C_{sb} \\ C_{bd} & C_{bg} & C_{bs} & C_{bb} \end{bmatrix} \tag{5.3}$$

Where the relevant elements of C_{xx} is label in Figure 5.6. For simplicity, the matrix shown in Equation 5.3 is assumed to be symmetric[1], the off diagonal terms are equal to each other, i.e. $C_{xy} = C_{yx}$. Our model thus only requires the computation of 6 unique capacitance values for each device. Furthermore it is reasonable to assume $C_{ds} = 0$ for the purposes of the work presented in this thesis.

[1] SPICE models can generate non-symmetric $\mathbf{C}_{Tx}$. Although this can lead to non-physical behaviours (e.g. violation of charge conservation), for many circuits, these models can be a "best fit" for modeling the distributed structure of an actual transistor to a four-terminal device.

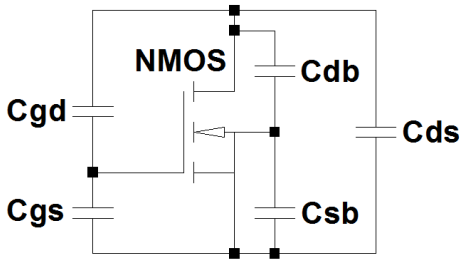

Figure 5.6: NMOS parasitic capacitors (not shown C_{gb})

C_{ox} Model

For both models considered in this thesis, having an accurate C_{ox} is crucial for proper simulation. Nominally C_{ox} is defined as:

$$C_{ox} = \frac{\varepsilon_0 \varepsilon_r}{t_{ox}} \tag{5.4}$$

Where $\varepsilon_0 = 8.854 \times 10^{-12} \frac{F}{m}$ is the vacuum permittivity, ε_r is the relative permittivity of the gate oxide material and t_{ox} is the oxide thickness. From the model card provided by the PTM 45nm HP process, $\varepsilon_r = 3.9$ and $t_{ox} = 1.25 \times 10^{-9} m$. Alternatively, one can extract C_{ox} by setting up a small signal experiment with SPICE (experimental setup show in Figure 5.8) and an AC signal sweep of a voltage source applied on the gate. In the 45nm HP PTM model card this was done for transistor gate lengths of 32nm, 45nm, 60nm, 90nm and 120nm with a fixed width of 450nm. The results shown in Figure 5.7 show the linear relationship and demonstrate that approximating the gate capacitance as a parallel plate capacitor is a reasonable assumption.

Comparing the results obtained from the SPICE experiment and the parallel plate capacitance yileds a difference of $\approx 2.71\%$. For both the MVS and EKV models C_{ox} is used as a base unit of capacitance from which the gate capacitance is derived from.

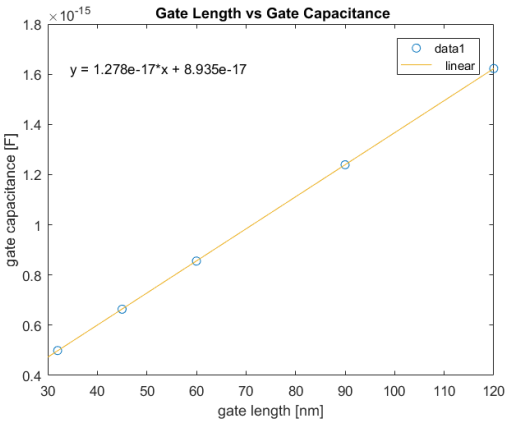

Figure 5.7: Linear Gate Capacitance Results

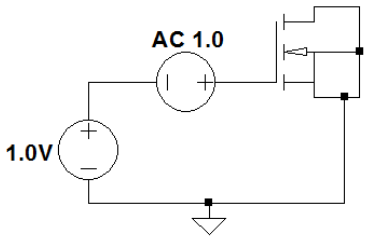

Figure 5.8: Gate Capacitance Experimental Setup

Junction Capacitance

Neither the MVS or EKV model have explicit junction capacitance models. In order to model this, I have extracted from the BSIM4 model [16] manual the equations to compute junction capacitance. Recall that the 45nm HP model card is available to us, therefore it is possible to find all the necessary parameters to compute the junction capacitance. The junction capacitance is the capacitance that exists between the source and body or drain and body, i.e. C_{sb} or C_{db}.

The junction capacitance model shown in Equation 5.5 is used to improve the accuracy of the simulations. Table 5.1 lists all the parameters needed to properly compute the junction capacitance for the MOS model. The names of the parameters and how they are derived is not of significant importance. Note that junction capacitance is the capacitance which accounts for a circuits ability to go to voltage levels slightly above V_{dd} or below *gnd* voltage levels.

Although the BSIM4 model for computing the junction capacitance is comprehensive, a simpler diode model with fewer parameters is likely to perform just as

well and is left as a future model simplification.

$$
\begin{aligned}
C_{jbd} &= \begin{cases} C_{jd}(T)\cdot\left(1-\frac{V_{bd}}{P_{bd}(T)}\right)^{-M_{jd}}, & V_{bd}<0 \\ C_{jd}(T)\cdot\left(1+M_{jd}\cdot\frac{V_{bd}}{P_{bd}(T)}\right), & \text{otherwise} \end{cases} \\
C_{jd}(T) &= C_{jd}(T_{nom})\cdot(1+T_{cj}\cdot(T-T_{nom})) \\
P_{bd}(T) &= P_{bd}(T_{nom})-T_{pb}\cdot(T-T_{nom}) \\
C_{jbdsw} &= \begin{cases} C_{jswd}(T)\cdot\left(1-\frac{V_{bd}}{P_{bswd}(T)}\right)^{-M_{jswd}}, & V_{bd}<0 \\ C_{jswd}(T)\cdot\left(1+M_{jswd}\frac{V_{bd}}{P_{bswd}(T)}\right), & \text{otherwise} \end{cases} \\
C_{jswd}(T) &= C_{jswd}(T_{nom})+T_{C_{jsw}}\cdot(T-T_{nom}) \\
P_{bswd}(T) &= P_{bswd}(T_{nom})-T_{pbsw}(T-T_{nom}) \\
C_{jbdswg} &= \begin{cases} C_{jswgd}(T)\cdot\left(1-\frac{V_{bd}}{P_{bswgd}(T)}\right)^{-M_{jswgd}}, & V_{bd}<0 \\ C_{jswgd}\cdot\left(1+M_{jswgd}\cdot\frac{V_{bd}}{P_{bswgd}(T)}\right), & \text{otherwise} \end{cases} \\
C_{jswgd}(T) &= C_{jswgd}(T_{nom})(1+T_{C_{jswg}}(T-T_{nom}) \\
P_{bswgd}(T) &= P_{bswgd}(T_{nom})-T_{pbswg}(T-T_{nom}) \\
C_{bd} &= A_dC_{jbd}+P_dC_{jbdsw}+W_{active}C_{jbdswg} \\
A_d &= W_{active}\times L_{active} \\
P_d &= X_j\times(W_{active}+L_{active}) \\
L_{active} &= L+X_L-2L_{int} \\
W_{active} &= W+X_W-W_{int}
\end{aligned}
\tag{5.5}
$$

5.2 MVS Model

The main idea behind the MVS model[23][48][28] is to create internal virtual nodes for the source and drain of each device which are determined based on the current that would need to flow in order to create an equilibrium. An iterative algorithm is used to find the solution to the virtual source and drain by starting with an initial guess for those virtual nodes to be the same potential as the potential actually applied to the drain and source nodes. Solving for the virtual source and drain dictates the current that must be flowing in the device (see Figure 5.9).

A semi-empirical model is used to capture all the effects described in Section 5.1.3. The main effects discussed in the previous section are derived from

Table 5.1: Junction Capacitance List of Paramaters

Parameters	Description	45nmHP Model Card
T_{nom}	Temperature for parameters	27^oC
W_{int}	Channel-width offset parameter	$5.0 \times 10^{-9} m$
L_{int}	Channel-length offset parameter	$3.75 \times 10^{-9} m$
X_L	Channel Length offset due to mask/etch effect	$-20 \times 10^{-9} m$
X_W	Channel Width offset due to mask/etch effect	$0.0m$
X_j	Source/Drain junction Depth	$1.4 \times 10^{-8} m$
$C_{jd}(T_{nom})$	Bottom junction capacitance per unit area at zero bias	$5 \times 10^{-4} F/m^2$
M_{jd}	Bottom junction capacitance grating coefficient	0.5 unitless
$C_{jswd}(T_{nom})$	Isolation-edge sidewall junction capacitance per unit area	$5 \times 10^{-10} F/m$
M_{jswd}	Isolation-edge sidewall junction capacitance grading coefficient	0.33 unitless
$C_{jswgd}(T_{nom})$	Gate-edge sidewall junction capacitance per unit length	$5 \times 10^{-10} F/m$
M_{jswgd}	Gate-edge sidewall junction capacitance grading coefficient	0.33 unitless
$P_{bd}(T_{nom})$	Bottom junction built-in potential	$1.0V$
T_{pb}	Temperature coefficient of P_{bd}	$0.005V/K$
T_{cj}	Temperature coefficient of C_{jd}	$0.001K^{-1}$
$P_{bswd}(T_{nom})$	Bottom junction built-in potential	$1.0V$
T_{pbsw}	Temperature coefficient of P_{bswd}	$0.005V/K$
$T_{C_{jsw}}$	Temperature coefficient of C_{jswd}	$0.001K^{-1}$
$P_{bswgd}(T_{nom})$	Bottom junction built-in potential	$1.0V$
T_{pbswg}	Temperature coefficient of P_{bswgd}	$0.005V/K$
$T_{C_{jswg}}$	Temperature coefficient of C_{jswgd}	$0.001K^{-1}$

physical principles. This model is a source referenced model which means that the assumption is that the drain node is at a higher potential then the source node, otherwise the source and drain nodes and the direction of the current is flipped.

This model also calculates the charge in the device which means that the node

capactiances can be calculated directly with automatic differentiation by applying equations Equation 5.2 and Equation 5.5. The body of work found in [38] includes software and instructions on how to fit the measured transistor data to the model parameters. Figure 5.10 and Figure 5.11 show the results of fitting LTSPICE data to the MVS model. The default C_{ox} value in the parameter fitting needed to be modified to that extracted from the PTM 45nm HP model card to get a good fit.

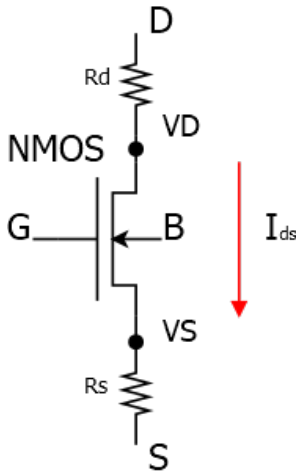

Figure 5.9: MVS n-device Transistor with Virtual source and drain

I used the three-stage ring oscillator circuit shown in Figure 5.12 to evaluate how well the fit parameters and capacitance model perform. In Figure 5.13 a comparison between the results obtained using SPICE and the MVS model is observed. The MVS model and its associated capacitance model extracted from AD yields results that agree quite closely to what is obtained using SPICE. This little experiment gives us confidence in the model's ability to properly simulate synchronizers.

Note however that there is a potential weakness in this test for the capacitance model. The three-stage ring oscillator does not exhibit any body effect because the circuit is purely made of inverters. A better test circuit such as a C-element oscillator[46] (micropipeline ring) would be a more comprehensive check.

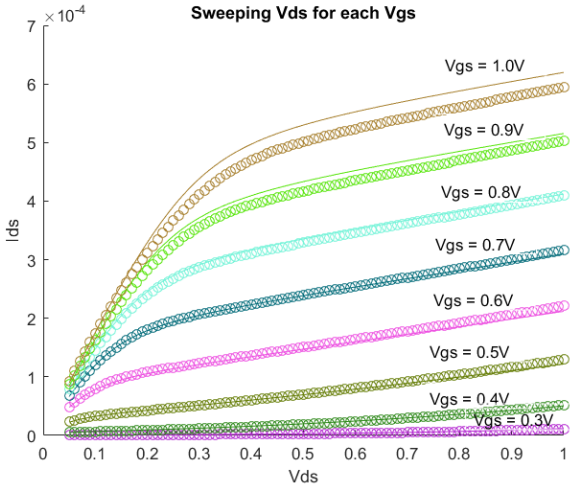

Figure 5.10: MVS fit for NMOS IV characteristic sweeping V_{ds}, bubbles are SPICE data points

5.2.1 MVS model deficiencies

Although the MVS model is a nice and compact model, there are some workarounds that need to be addressed. The first is that because the model is source referenced, and requires V_{ds} to be a positive quantity, there arise difficulties when $V_{ds} \approx 0$. A physically realistic model should be infinitely differentiable everywhere. Experiments in [38] demonstrate that the MVS model is at least C^2 at $V_{ds} = 0$, but not C^∞ due to the piecewise definition at $V_{ds} = 0$. A practical problem arose because the iterative solver in the MATLAB implementation provided by the MVS model authors [38] checked V_{ds} and iteratively updated estimates of the virtual source and drain. When V_{ds} is close to zero, intermediate steps of the solver can flip the sign of the virtual source and drain which caused the model evaluation to fail when attempting to take the log of negative quantities. A correction to the code from [38] was applied to mitigate this problem.

The iterative solver introduces challenges for use with AD. While the iterative solver is finding solutions for the virtual source and drain, we suspend the use of AD. We then need a meaningful way to re-introduce the AD functionality after the virtual source and drain have been finalized. The special care that is required to

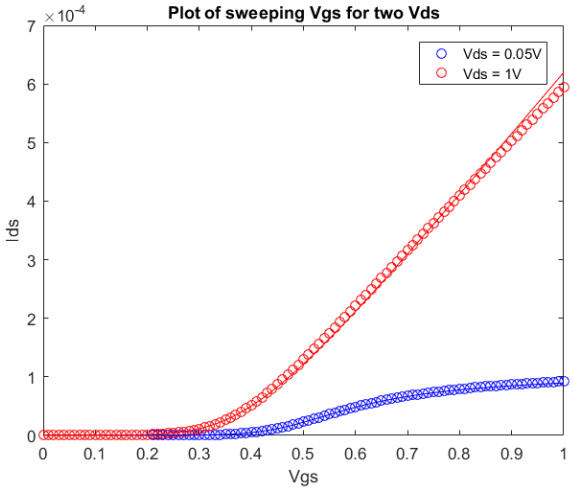

Figure 5.11: MVS fit for NMOS IV characteristic sweeping V_{gs}, bubbles are SPICE data points

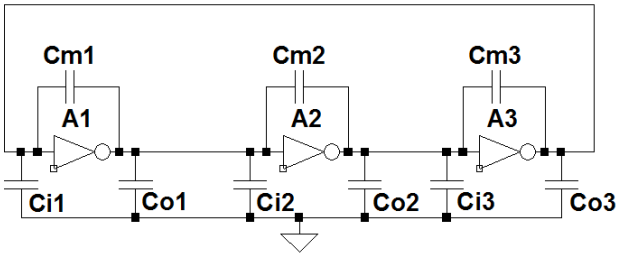

Figure 5.12: Three Ring Oscillator circuit with Miller capacitance

preserve meaningful derivatives makes the implementation less flexible. Computing the derivative of the current with respect to the drain or source nodes requires new methods because they are functions of the solution to the virtual source and drain. In order to compute $\frac{\partial I_{ds}}{\partial V_x}$ where x are the terminal voltages, a new technique is employed which is discussed in Chapter 4 Section 4.3.

The final remarks are that despite the MVS model being a nicely compact model which produces accurate results as demonstrated in Figure 5.13, Figure 5.10 and Figure 5.11 its piecewise model is problematic with AD; `if` is not differentiable. The MATLAB code has been revised to eliminate all but one of the conditional

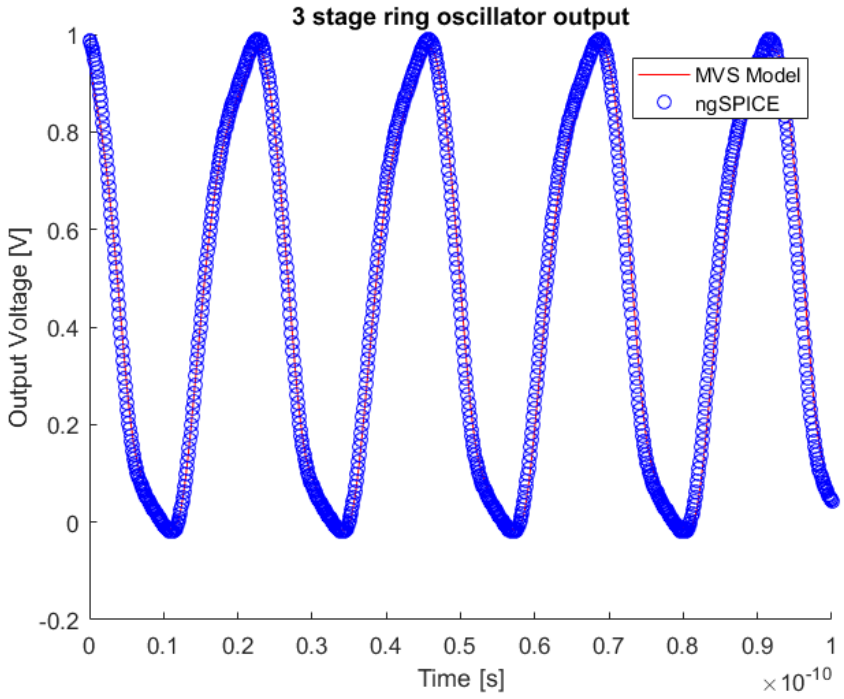

Figure 5.13: Three stage ring oscillator MVS model comparison with SPICE simulation

statements in the model (outside of the root solver, which was handled separately as described above). The remaining `if` arises from the requirement that $V_s \leq V_d$ (for an NMOS device). This is a consequence of the MVS model being a source-referenced model. These concerns of simplicity, efficiency and differentiability motivated examining the EKV model as described in the next section. Furthermore, model evaluation dominated the time in the analysis algorithms of Chapter 3.

5.3 Simplified EKV Model

The EKV model, named after its original creators, Enz, Krummenacher, and Vittoz is a body-referenced model that has gone through many revision over the past 25 years [13][10][12][11]. Similarly to the BSIM and MVS models, the EKV model is physics based. By taking a body-referenced approach, the symmetry of the source

and drain is built into the design of the formulas of the model. For the work in this thesis, we are primarily interested in a model that is numerically tractable and produces results that are close enough to the physics based model to enable meaningful analysis of circuit design trade-off. Thus, we derive a simplified, empirical version of the model. The form of our model is motivated by the underlying physics, e.g. using exponentials to reflect Boltzmann distributions, but the choice of parameters is purely empirical: based on curve fitting to get a "good enough" fit.

In this section I introduce a simplified EKV model with a few parameters that lump together many of the physical parameters described in Section 5.1.3. Similarly to the MVS model, the EKV model is compact and has a handful of parameters. The main idea behind the EKV model is that the current flowing in the channel of a transistor is the sum of the *forward* current and *backward* current. This simple idea is attractive because the function to compute both the forward and backward currents are continuous and infinitely differentiable functions i.e. C^{∞}. The EKV model is a body referenced model which means that there is symmetry about the MOS device, thus there is no case split for V_{ds} when the source is larger than the drain. In that scenario all that happens is the backward current overtakes the forward current and I_{ds} becomes negative.
I start by writing out our proposed formula for computing I_{ds}:

$$\begin{aligned} u &= \alpha\left\{V_g - \beta V_d - V_s - \left(V_{th0} + \gamma\left(\sqrt{\phi_f + (V_s - V_b) \times type} - \sqrt{\phi_f}\right)\right)\right\} \\ v &= \alpha\left\{V_g - \beta V_s - V_d - \left(V_{th0} + \gamma\left(\sqrt{\phi_f + (V_s - V_b) \times type} - \sqrt{\phi_f}\right)\right)\right\} \\ I_{ds} &= W I_0 \left(\log\left[1 + \exp\left(u\right)\right] - \log\left[1 + \exp\left(v\right)\right]\right) \end{aligned} \tag{5.6}$$

A few terms that are familiar from Section 5.1.3 are present, such as the body effect term from Equation 5.1 and W the width of the transistor. That leaves α, a term to capture velocity saturation, β, a term to capture DIBL and I_0 a scaling term that gives the optimizer some flexibility to scale the output as it sees fit. The units associated with these quantities are not derived from physical properties of the device like the MVS model.

5.3.1 EKV Model Fitting

The fitting algorithm to find the parameters for the EKV model simply uses gradient descent (Equation 5.7) and takes a step size according to the Wolfe condition shown

in Equation 5.9. I allow for a weighted sum of the error with w_i because otherwise the fit will distribute all the error to regions where the current is relatively small. Therefore the weighting provides a mechanism to distribute the error a little better. Automatic differentiation can easily be used to compute the gradient with respect to the parameters, however it is also nearly as easy and more illuminating to compute the derivative manually shown in Equation 5.8. The tabulated I_{ds} from Section 5.1.2 is taken as ground truth, thus the objective is to come up with parameters that minimize the error from the EKV model and the tabulated I_{ds} from SPICE.

$$\begin{aligned} \nabla_{Params} I_{ds} &= \tfrac{\partial I_{ds}}{\partial Params} \\ \nabla_{Err_{Params}} &= \Sigma_{i=1}^{n} 2 \tfrac{\partial I_{ds}}{\partial Params} \cdot w_i \cdot \left(I_{ds_i} - I_{ds_{meas_i}} \right) \\ Params_{i+1} &= Params_i - \gamma_{step} \nabla_{Err_{Params}} \end{aligned} \tag{5.7}$$

$$\begin{aligned} \text{Let} \quad \phi_M &= \sqrt{\phi_f + (V_s - V_b) \times type} - \sqrt{\phi_f} \\ V_{mod} &= \gamma \times \phi_M \\ u &= \alpha \left(V_g + \beta V_d - V_s - (V_{th0} + V_{mod}) \right) \\ v &= \alpha \left(V_g + \beta V_s - V_d - (V_{th0} + V_{mod}) \right) \end{aligned}$$

$$\frac{\partial I_{ds}}{\partial Params} = \begin{bmatrix} \frac{\partial I_{ds}}{\partial I_0} \\ \frac{\partial I_{ds}}{\partial \alpha} \\ \frac{\partial I_{ds}}{\partial \beta} \\ \frac{\partial I_{ds}}{\partial V_{th0}} \\ \frac{\partial I_{ds}}{\partial \gamma} \\ \frac{\partial I_{ds}}{\partial \phi_f} \end{bmatrix} = \begin{bmatrix} W \left(\log\left[1 + \exp(u)\right] - \log\left[1 + \exp(v)\right] \right) \\ W I_0 \left(\frac{e^u u}{\alpha(1+e^u)} - \frac{e^v v}{\alpha(1+e^v)} \right) \\ W I_0 \left(\alpha V_d \frac{e^u}{1+e^u} - \alpha V_s \frac{e^v}{1+e^v} \right) \\ W I_0 \left(-\alpha \frac{e^u}{1+e^u} + \alpha \frac{e^v}{1+e^v} \right) \\ W I_0 \left(-\alpha \phi_M \frac{e^u}{1+e^u} + \alpha \phi_M \frac{e^v}{1+e^v} \right) \\ W I_0 \left(-\alpha \gamma \frac{e^u}{(1+e^u)} \left(\frac{1}{2\sqrt{\phi_f + (V_s - V_b) \times type}} - \frac{1}{2\sqrt{\phi_f}} \right) \right. \\ \left. + \alpha \gamma \frac{e^v}{1+e^v} \left(\frac{1}{2\sqrt{\phi_f + (V_s - V_b) \times type}} - \frac{1}{2\sqrt{\phi_f}} \right) \right) \end{bmatrix} \tag{5.8}$$

$$\gamma_{step} = \frac{\left| \left(Params_{i+1} - Params_i \right)^T \left(\frac{\partial I_{ds_{i+1}}}{\partial Params} - \frac{\partial I_{ds_i}}{\partial Params} \right) \right|}{\left| \left| \frac{\partial I_{ds_{i+1}}}{\partial Params} - \frac{\partial I_{ds_i}}{\partial Params} \right| \right|^2} \tag{5.9}$$

Using gradient descent as shown in Equation 5.7, results in a fit shown in Figure 5.14 and Figure 5.15. The resultant fit is not as good as that from Section 5.2 but the simplicity and body referenced model make this a very attractive model which captures the behavior reasonably well to explore synchronizer designs. The

PMOS device fits better in this EKV model than the NMOS device. See Figure 5.16 and Figure 5.17

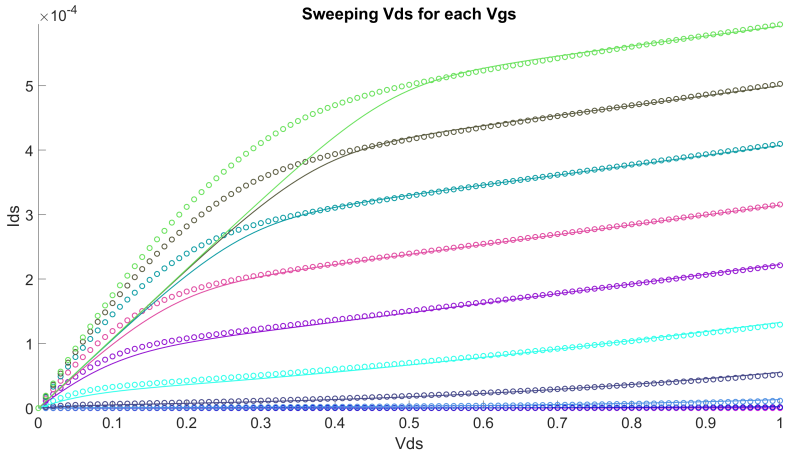

Figure 5.14: EKV fit for NMOS IV characteristic sweeping V_{ds}, bubbles are SPICE data points

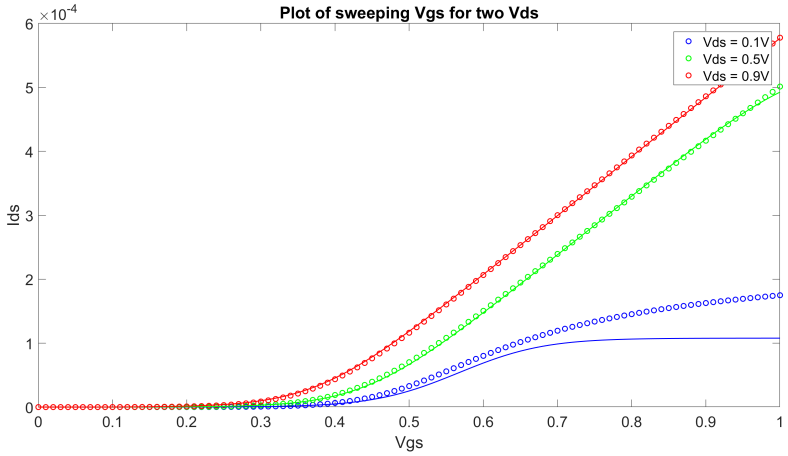

Figure 5.15: EKV fit for NMOS IV characteristic sweeping V_{gs}, bubbles are SPICE data points

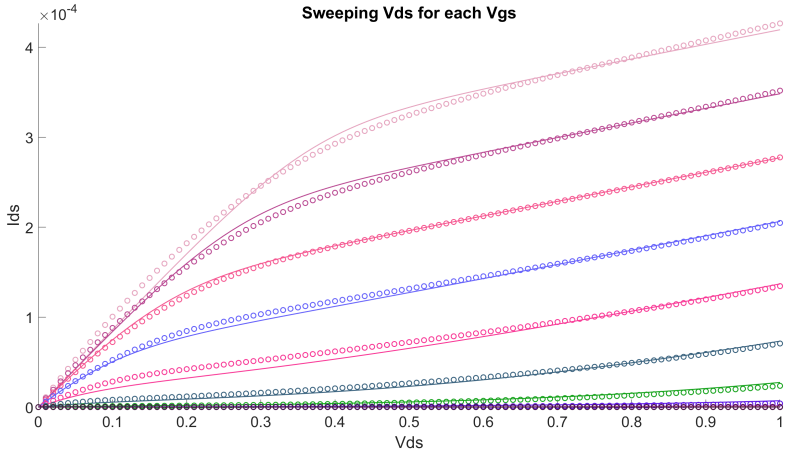

Figure 5.16: EKV fit for PMOS IV characteristic sweeping V_{ds}, bubbles are SPICE data points

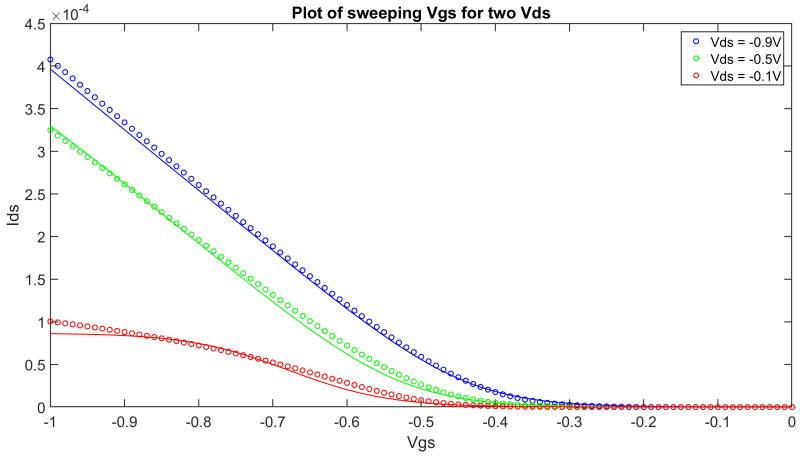

Figure 5.17: EKV fit for PMOS IV characteristic sweeping V_{gs}, bubbles are SPICE data points

5.3.2 EKV Capacitance Model

This EKV model uses the capacitance model from [10] which distributes C_{ox} computed by the geometry of the transistor in proportion to the normalized currents i_f and i_r as in Equation 5.10. Using the computed capacitance as shown in Equa-

tion 5.10 with the junction capacitance from Equation 5.5 yields a result that has too little capacitance when comparing against the result of the three ring oscillator. Through trial and error, n the shape factor in Equation 5.10 for the PTM 45nm HP model is set to 1.3. The overall computed capacitance from Equation 5.10 and Equation 5.5 is scaled up by 2.2 which produces results that are similar in accuracy as those shown in Figure 5.13. Figure 5.18 demonstrates that it is possible to get reasonable results as compared to SPICE by distributing C_{ox} as described in Equation 5.10.

$$\begin{aligned}
i_f &= \log(1+e^u) \\
i_r &= \log(1+e^v) \\
q_f &= \frac{\sqrt{1+4i_f}-1}{2} \\
q_r &= \frac{\sqrt{1+4i_r}-1}{2} \\
c_s &= (1/3)\frac{(q_f\cdot(2q_f+4q_r+3)}{(q_f+q_r+1)^2} \\
c_d &= (1/3)\frac{(q_r\cdot(2q_r+4q_f+3)}{(q_r+q_f+1)^2} \\
C_{gs} &= C_{ox}\cdot c_s \\
C_{gd} &= C_{ox}\cdot c_d \\
C_{gb} &= \frac{n-1}{n}\cdot(C_{ox}-C_{gs}-C_{gd})
\end{aligned} \tag{5.10}$$

5.3.3 EKV model deficiencies

My EKV model lumps the physical behaviors of certain transistor effects into a single parameter without any physical explanation unlike the BSIM, MVS, and published EKV models. The model makes no attempt to explain the parameters but simply provides them as a means to make a better fit.

The fit for the NMOS device is noticeably worse for small values of V_{ds} and V_{gs} as compared to the PMOS device. The NMOS device computes currents that are smaller than they should be. This means that my NMOS devices in those regions will be slower responding or have worse than expected drive strength.

The assumption of always having a velocity saturated model is obviously apparent for low V_{ds} in the NMOS devices. We believe that a near future improvement can be made to obtain a better term for velocity saturation which takes into account the field strength.

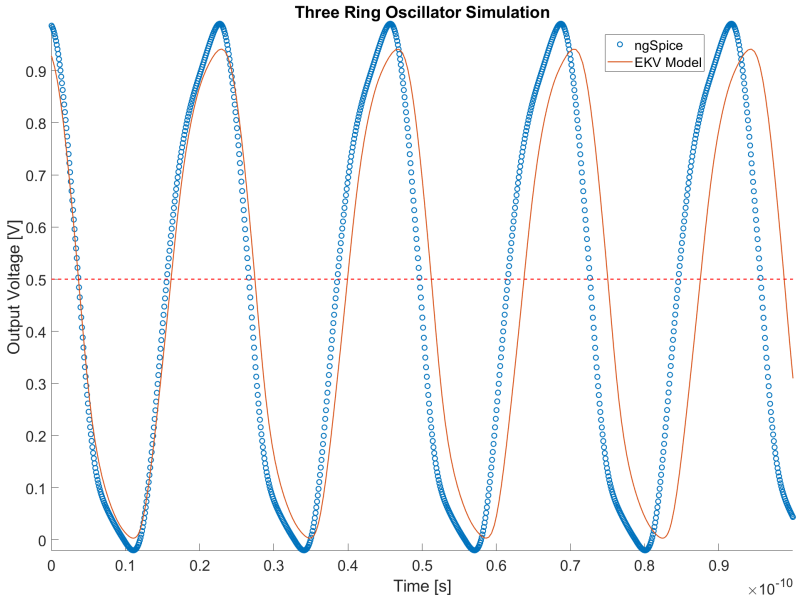

Figure 5.18: Three stage ring oscillator EKV model comparison with SPICE simulation

On the other hand, the subjective notice in simulation speed-up time by using a simpler model is attractive for analysis. In the interest of getting simulation results faster and obtaining results that are likely to be within 20% is considered to be good enough to demonstrate the effectiveness of the new tools in this thesis.

Chapter 6

Passgate Synchronizer and Variants: A Running Example

Chapter 3 developed the mathematics and tools to analyze synchronizer designs, and Chapter 5 described the underlying models which are used to simulate circuits. In this chapter, we turn our attention on a particular design to see how the models and tools produce results that enable designers to understand their synchronizer's behaviour and thereby optimize their designs in ways that could not be done with previous analysis tools. The synchronizers analyzed in this chapter consist of chains of two or three flip-flops based on pass-gate latches. The methods presented are readily applicable to synchronizers with more or fewer latches, and other design topologies which are not a linear chain such as the wagging based synchronizer design by Alshaikh *et al.* in [1]. A flip-flop can be constructed to be "edge" triggered by connecting two latches together that are enabled/disabled by the clock signal and its logical inverse. By enabling latches on opposite polarities of the clock, the latches in a chain alternate between transparent and opaque as described in Section 4.2.

The simplified EKV model from Chapter 5 is used to perform all of the simulations without loss of generality. The simplified EKV model can be evaluated much faster when running the tools as compared to the more accurate MVS model. The results from Chapter 5 gives us confidence that when a designer switches to a more accurate model, that the analysis becomes more accurate as dictated by the model

but will not lead to qualitatively different conclusions in most cases.

We start with the passgate latch synchronizer because it is a design that is widely used in practice. The goal is to establish a baseline design with which we can use as a benchmark in order to compare different synchronizer designs and variants. In this chapter, the details of the passgate latch and passgate synchronizer designs are described with explanations of their digital and continuous behaviors. We then cover in detail the results obtained by using the tools from Chapter 3 on the passgate synchronizer design and demonstrates how to interpret $g(t)$, $u(t)^T$ and $\lambda(t)$ as measures to establish benchmark characteristics. With a benchmark design to compare against, I investigate three passgate synchronizer design variants: one with scan circuitry, the kicked synchronizer and the offset synchronizer. The kicked and offset synchronizers are variants that aim to improve metastability resolution. I explore their consequence and impact to metastability resolution against the benchmark design.

The analysis tools leads us to a new discovery. The passgate element, one often though of as being a passive device can be exploited as a small signal amplifier for a brief period of time during metastability resolution of a passgate synchronizer. This subtle behavior is discovered as a result of the new tools at our disposal.

6.1 The Passgate Latch & Synchronizer

This section describes the components and functionality of a passgate latch and outlines how the composition of four passgate latches makes a two flip-flop passgate synchronizer. No new or novel results are presented in this section, on the other hand this section is present to keep the thesis a self-contained document. We present the functionality of the building blocks of a passgate latch by first describing their digital abstraction followed by an intuition into their analog and continuous behavior. This section is intended to be referred to as a reference when I discuss various parts of the passgate synchronizer in the following sections.

6.1.1 The Passgate Latch

The passgate latch, shown in Figure 6.1, is made of two passgates, a cross-coupled pair of inverters and a buffer inverter which minimizes loading the nodes of the

cross-coupled pair when the latch is connected to additional circuitry. The passgate latch uses the clock and its logical inverse to turn on and off the passgates which produce the transparent and opaque phases of the latch. We describe how the latch and each of its components function in three steps:

- By describing the component's digital abstraction behavior
- By an intuitive analog argument and its implications
- And finally we provide some key insights into the detailed non-linear behavior of transistors

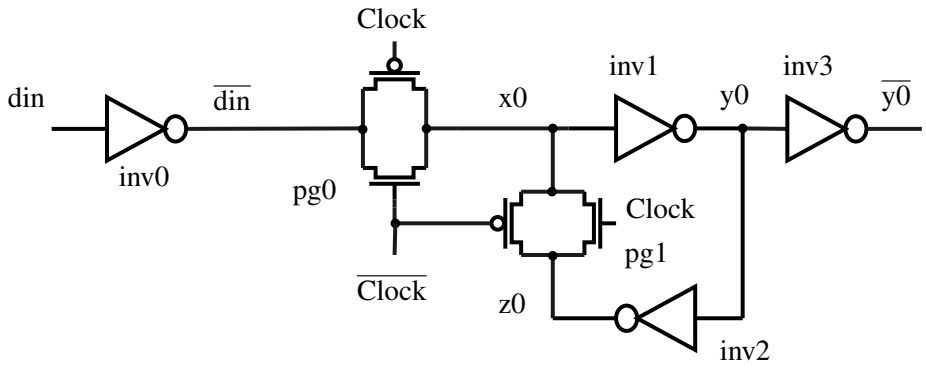

Figure 6.1: The passgate latch

Passgates

A passgate is a device which has two input signals `x` and `y` and a control input `enable`. The CMOS implementation consists of two transistors, a NMOS and PMOS shown in Figure 6.2 and control signals `enable` and $\overline{\texttt{enable}}$. The `enable` and $\overline{\texttt{enable}}$ ports are mapped to the `clock` signal and its logical inverse, $\overline{\texttt{clock}}$ in the passgate latch. Let us consider the passgate as a digital device. When `enable` is true, the passgate acts like a closed switch connecting `x` to `y`, and conversely when `enable` is false the passgate acts like an open switch. In the passgate latch from Figure 6.1, when the `clock` signal is high, then `pg0` connects nodes $\overline{\texttt{din}}$ and `x0`. When the `clock` signal is low, then `pg1` connects nodes `z0` and `x0`. The pass-

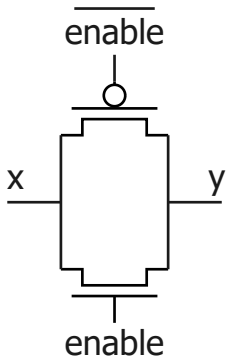

Figure 6.2: Passgate

gates in the passgate latch work together as a multiplexer to drive node `x0` either to the signal at node `z0` or $\overline{\texttt{din}}$.

Designers construct passgates with two transistors as shown in Figure 6.2 because the NMOS devices are effective at conveying 0s (low levels) but not 1s (high levels) and vice versa for PMOS devices. To see why, let's consider just the NMOS device from Figure 6.2. In order for the NMOS device to be well conducting, the gate voltage on the `enable` node needs to be at least the transistor's threshold voltage above the source node. Let `x` be the source node, then in order to convey $\texttt{x} = V_{dd}$ to node `y`, would require the `enable` node to be V_{dd} + the threshold voltage. This is not practical. Fortunately, PMOS devices behave in a complementary fashion. Therefore the NMOS and PMOS devices together form a practical passgate.

In more detail, transistor behavior can be thought of as a variable resistor whose resistance is controlled by the transistor's gate signal at `enable`. The resultant resistance also dependent on the input signals `x` and `y`, i.e. the source and drain nodes as described above. Thus the passgate is a non-linear device. Section 6.4 describes how the non-linearity of the passgate can lead it to behave as a small signal amplifier. As a result we uncover a surprising outcome when comparing passgate synchronizer variants.

Inverters & Cross-Coupled Pairs

A cross-coupled pair of inverters is simply two inverters connected in a loop as shown in Figure 6.3. From a digital point of view, if we let $\mathtt{x} = 1$ then $\mathtt{y} = 0$ and conversely if we let $\mathtt{x} = 0$ then $\mathtt{y} = 1$. These two states are stable and thus the cross-coupled pair is a digital circuit which can hold state.

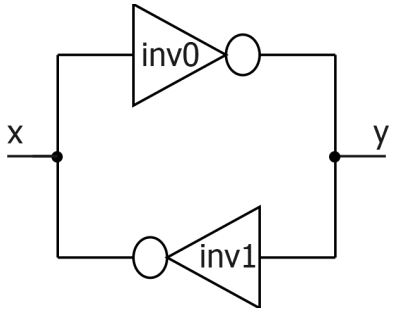

Figure 6.3: Cross-Coupled pair

The continuous model of an inverter defines a DC transfer function as shown in Figure 6.4. The transfer function shows the voltage level the inverter would asymptotically approach for a fixed input voltage. In practice, after a few nanoseconds, the output will have settled to well within the tolerances of any practical measurement.

Designers want digital logic gates to be *restoring* circuits. In other words, the output of a digital logic gate should be closer to *gnd* or V_{dd} than its inputs. For the inverter, this means that the slope of the DC transfer function should be negative everywhere and close to 0 near the points where $\mathtt{x} = 0$ and $\mathtt{y} = 1$ or where $\mathtt{x} = 1$ and $\mathtt{y} = 0$. A designer wants a high gain region where the slope $\ll -1$ when transitioning between these two states. These design requirements suggest the negated sigmoid shape depicted in Figure 6.4.

When we consider the superposition of the DC transfer functions of Figure 6.4 for the cross-coupled pair of inverters, we get Figure 6.5. The points where the two curves intersect are the equilibrium points of the circuit. The equilibria at $\mathtt{x} = 0$ and $\mathtt{y} = 1$ and at $\mathtt{x} = 0$ and $\mathtt{y} = 1$ are familiar from the digital description of the circuit. These two equilibria must be stable because we assume that at these points $|slope| < 1$ for the DC transfer functions of the inverters. On the other hand, there is

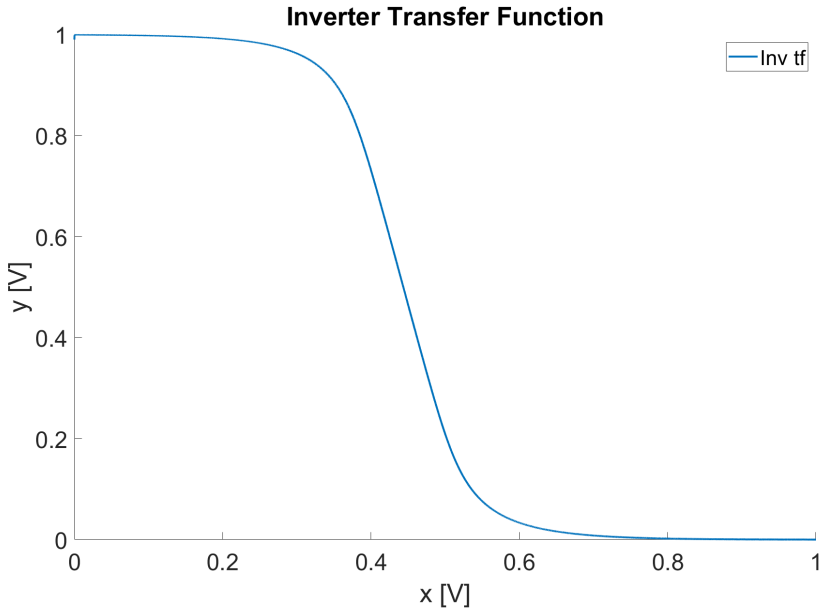

Figure 6.4: Inverter DC transfer function

a third equilibrium, when $\mathrm{x} = \mathrm{y} \approx 0.47V$ for the inverter that we simulated. This is a *metastable equilibrium*. At this point, the DC transfer function has a $|slope| > 1$ which leads to an unstable equilibrium. Small perturbation about the metastable equilibrium will cause nodes x and y to diverge and settle to one of the two stable equilibria.

We can analyze the inverter circuit at the metastable equilibrium depicted in Figure 6.5, and describe its behavior as a linear small-signal system. Each inverter of the cross-coupled pair can be thought of as an amplifier with gain $G < -1$ driving an RC circuit as shown in Figure 6.6. For this small-signal analysis, we will write $v_{ms}(x)$ to denote the voltage on x minus the voltage of the balance point, i.e. we are shifting our voltage reference to the metastable voltage. Similarly we will define $v_{ms}(y)$ as the voltage on y minus the metastable point. Therefore we have:

$$\begin{aligned} \frac{d}{dt} v_{ms}(x) &= \frac{G}{RC}\left(v_{ms}(y) - v_{ms}(x)\right) \\ \frac{d}{dt} v_{ms}(y) &= \frac{G}{RC}\left(v_{ms}(x) - v_{ms}(y)\right) \end{aligned} \tag{6.1}$$

The eigenvalues of the system are $-1 \pm G$. Because $G < -1$, the eignvalue at

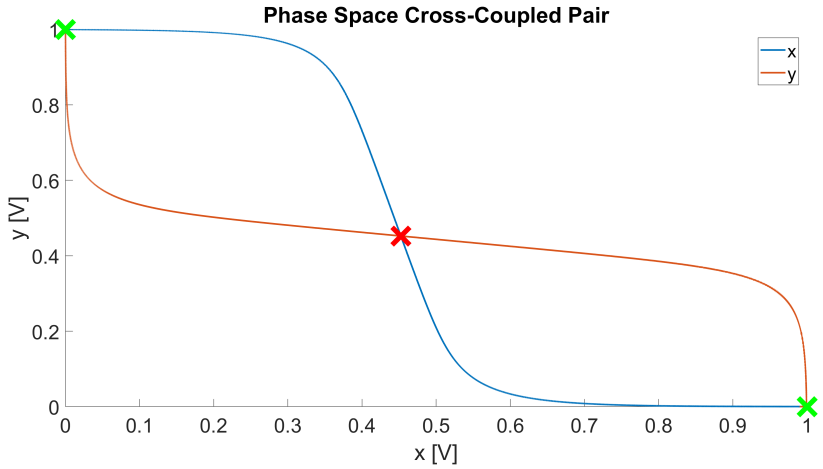

Figure 6.5: Cross-coupled pair phase space with the attractors marked in green and the unstable saddle in red

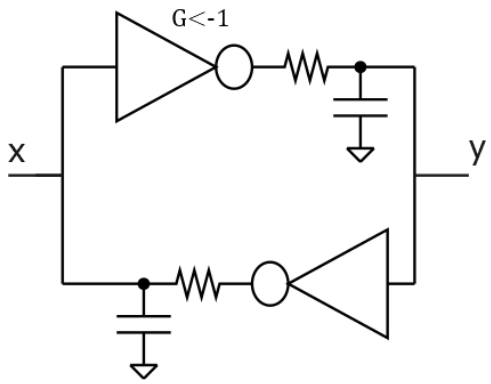

Figure 6.6: Cross-coupled pair as amplifier circuit

$-1+G$ is negative and corresponds to the decay of the "commond mode" component of the state (i.e. $x+y$) as time evolves. The eigenvalue at $-1-G$ is positive and gives rise to the divergence from metastability – the differential component of the state (i.e. $x-y$) is exponentially increasing with time. This same exponential divergence is the one that is used to described the probability failure of a synchronizer. Here the "synchronizer" is a single latch, however, keeping this in mind, the

synchronizer designer's goal is to maximize this positive eigenvalue to maximize the rate of divergence from the metastable equilibirum and minimize the failure probability.

A succinct way of demonstrating the dynamics of a cross-coupled pair of inverters is shown in Figure 6.5 where the green marks illustrate the attractors of the system and the red mark illustrates the unstable saddle at the point $x = y$. We can change the shape of the curves by changing the design of the inverters, but there must exist between the two stable attractors a point where the transfer functions of the inverters intersect and it is at this intersection where metastability arise.

6.1.2 The Two Flip-Flop Passgate Synchronizer

A two flip-flop passgate synchronizer is constructed by connecting four passgate latches in a chain as shown in shown in Figure 6.7. In order to make the chain of latches a synchronizer, the clock signals `clk` and `clkB` are connected to the chain of latches in an alternating pattern. With this alternating phasing of the clocks, the first latch, often called the "master", of each flip-flop is transparent when the clock is low and opaque when the clock goes high. Conversely, the second latch, often called the "slave", is transparent when the clock is high, and opaque when the clock is low. Thus, when the clock makes a low-to-high transition, the master latch retains the value of $\mathtt{d}_{in}$ from just before the clock transition, and the slave propagates this value the output, `y1`, just after the clock transition. This sampling of the input and updating of the output on the rising edge of the clock makes the flip-flop "positive edge triggered". When used as a synchronizer, we observe that the latches can exhibit metastable behaviour when they are opaque. Thus, the master latch can be metastable when the clock is high, and the slave can be metastable when the clock is low. The metastability "moves" from one latch to the next when the clock transitions. In our analysis, this notion of metastability moving is quantified by the function $u(t)^T$.

The digital behaviour of the passgate synchronizer shown in Figure 6.8 illustrates how a data transition which occurs well before the first rising edge of the clock at t_1 propagates through the synchronizer to the output at `y3` within two clock cycles. For the period of time between t_0 and t_1, `master1` is transparent, `slave1`

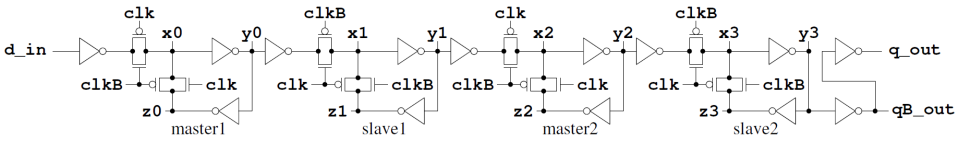

Figure 6.7: Two flip-flop passgate synchronizer

is opaque, and `master1`'s output `y0` tracks the synchronizer's input at `din`. When the clock signal transitions at time t_1, `slave1` becomes transparent while `master1` becomes opaque. On the time interval $t \in [t_1, t_2]$, `slave1`'s output `y1` tracks the output of `master1`, `y0`. After the falling edge of the clock at time t_2 `slave1` becomes opaque and `master2` becomes transparent and the signal propagation continues. We can think of the synchronizer as a shift register where at each clock transition the input signal shift down the chain by 1 latch.

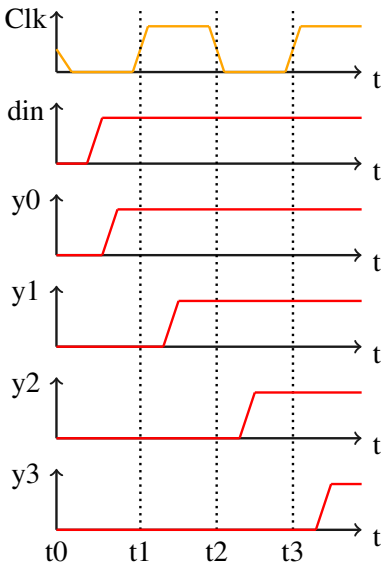

Figure 6.8: Digital behaviour and propagation of a two flip-flop passgate synchronizer

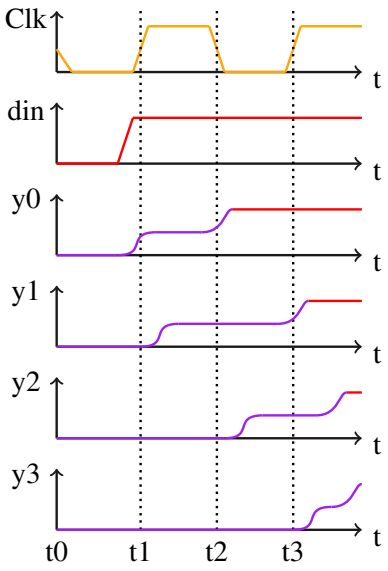

Figure 6.9: Metastability behaviour and propagation of a two flip-flop passgate synchronizer

Figure 6.9 considers the continuous nature of the underlying electronics. If we allow `din` to transition around the neighbourhood of t_1, i.e. as the clock is also transitioning, then `master1`'s passgate may become opaque at a point in time

where nodes x0, y0 and z0 happen to be at a voltage levels which are very close to the cross-coupled pair's metastable point x = y. Having previously established that between times $t \in [t_1, t_2]$ slave1's output y1 tracks master1's output y0, then if node y0 is at or near the metastable point then slave1's output y1 will correspondingly track y0. Observe that in the event that x0 $\approx$ y0 at time t_2, then it's possible that as slave1 becomes opaque its output y1 causes master2's output y2 to head towards the metastable point. The ideas behind the digital propagation of the input signal down the chain of latches at logic levels through the synchronizer also applies to metastable behaviour.

The difference is that the probability that x0 $\approx$ y0 at time t_2 as a result of x0 $\approx$ y0 at time t_1 is decreasing exponentially in time. Intuitively we can think about the time varying nature of metastability in a synchronizer "moving" from stage to stage, but in order for y1 to exhibit metastable behaviour requires y0 to have been metastable long enough such that at time t_2 node y1 $\approx$ x2 and so on. The remaining sections in this chapter illustrates, quantifies and explains the manifestation of metastability in the passgate synchronizer design and explores the impact on design variants.

6.2 The Passgate Latch Synchronizer Benchmark

This section explains the details in analyzing the passgate synchronizer described in Section 6.1 using the tools developed in Chapter 3 to establish a baseline benchmark. The benchmark will be used to compare design variants in the subsequent sections.

The key definitions established from Chapter 3 are summarized:

- t_{crit} is the time by which the outputs of the synchronizer must have stable digital values.
- t_{eola} is the time at which the small-signal linear analysis ends. The synchronizer has large signal, non-linear behaviours that are distinct for the settle-low and the settle-high outcomes for $t > t_{eola}$.
- Δt_{in} is the width of the time window for which the synchronizer output is not guaranteed to settle by t_{crit}.

- In the analysis of the passgate synchronizer, we define t_{settle} as the time at which all signals in the synchronizer have settled to within 10% of the voltage rails. The conditions that define t_{settle} are used to accelerate the nested bisection algorithm, but t_{settle} itself plays no role in the analysis.

To establish our benchmark design, we start by running Yang & Greenstreet's nested bisection algorithm [51] on the passgate synchronizer design shown in Figure 6.7. The resultant trajectories shown in Figure 6.10 are collected as described in Section 3.1 to create a single "perfectly" metastable trajectory up to time t_{eola}.

A small-signal linear model of the synchronizer describes all simulated trajectories from when the data transition t_{in} occurs to the time t_{eola}. This model is used to compute $g(t)$ and $\lambda(t)$ which are derived from $\beta(t)$ and $u(t)^T$. We use $\lambda(t)$ to observe the overall performance characteristics of the synchronizer and take the resultant curve as our benchmark measure. I demonstrate how $u(t)^T$ is used as our magnifying glass in Figure 6.14 and Figure 6.15 to show which latch and its internal nodes matters towards metastability resolution at any given time illustrating the time varying nature of metastability in a synchronizer.

6.2.1 Nested Bisection Trajectories

To begin the analysis of the passgate synchronizer, we run the nested bisection algorithm to find the "perfectly" metastable trajectories which just meet our selected critical time $t_{crit} = 7.5 \times 10^{-10}$s. All our examples unless otherwise stated use this timing criterion. Our benchmark example employs a transistor sizing scheme such that all inverters and passgates in the design have a 1:1 p-to-n ratio. Figure 6.10 shows the results obtained from the nested bisection algorithm to at least time t_{crit}. In Figure 6.10 we show latch output nodes `y0`, `y1`, `y2` and `y3`. One can visualize the "perfectly" metastable trajectory by following the path that is between these trajectories that settle high and low. Figure 6.11 illustrates the resultant "perfectly" metastable trajectory up to time t_{eola} which is computed from the collection of the nested bisection trajectories.

We have previously claimed in Section 3.2 that we can compute $\beta(t)$ by numerical differencing as the nested bisection algorithm progress. Because $\beta(t)$ is a vector of size $N \times 1$, we report it as $\log_{10}(||\beta(t)||_2)$. Figure 6.12 summarizes the

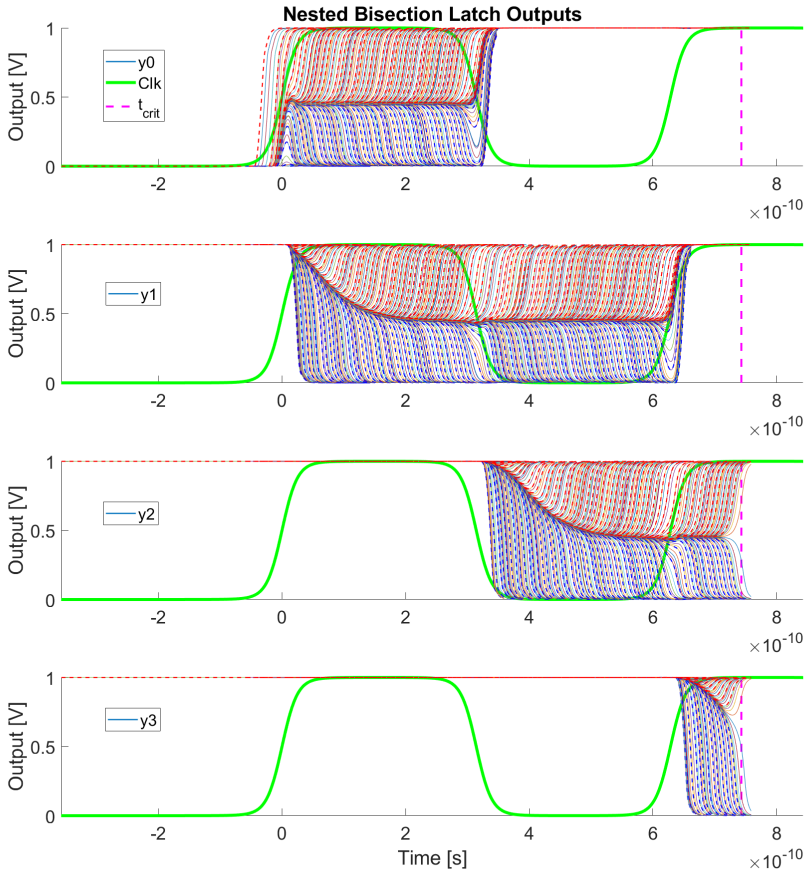

Figure 6.10: Passgate synchronizer latch outputs from nested bisection runs

numerical differentiated result of $\hat{\beta}(t)$ from the K epochs of the nested bisection algorithm using Equation 6.2. Note that in Equation 6.2 we divide by 3Δ because trajectories $\vec{V}_{H,k}(t)$ and $\vec{V}_{L,k}(t)$ at epoch k are in fact 3Δs away from each other for trajectory classification robustness as discussed in Section 2.4.3.

$$\begin{aligned} \Delta &= \frac{t_{in,H,k} - t_{in,L,k}}{M} \\ \hat{\beta}(t) &= \frac{\vec{V}_{H,k}(t) - \vec{V}_{L,k}(t)}{3\Delta} \end{aligned} \tag{6.2}$$

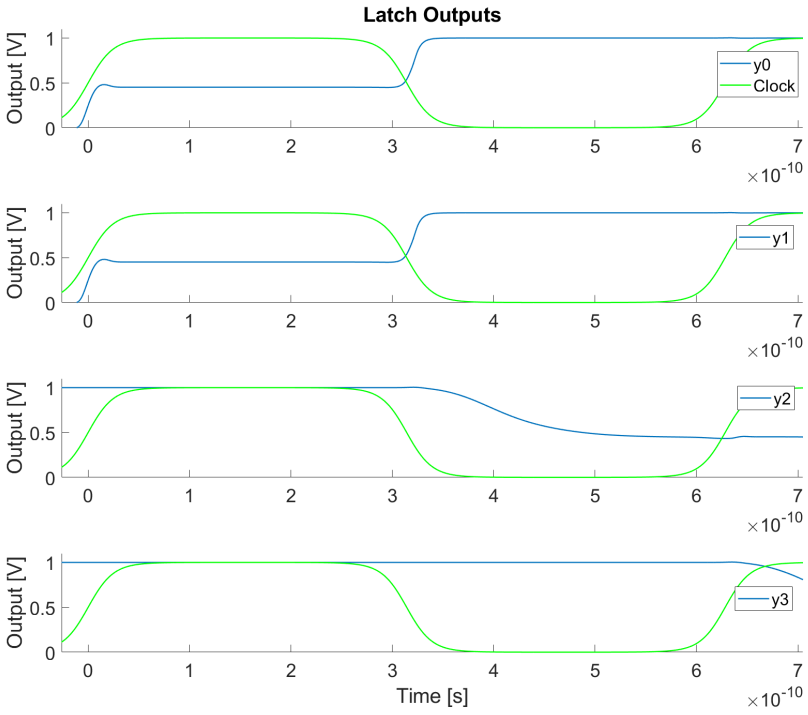

Figure 6.11: Passgate synchronizer latch outputs for "perfectly" metastable trajectory

6.2.2 Analysis Results

With the "perfectly" metastable trajectory $\vec{V}_{meta}(t)$, we can now compute $\beta(t)$, $u(t)^T$ and $\lambda(t)$. Our selection of $\tilde{u}^T$ is influenced by the observations made in Figure 6.10 and Figure 6.11. We note that node `y3` begins to track `master2`'s output node `y2` as `slave2` becomes transparent after the 2^{nd} rising edge of the clock signal shown in Figure 6.10 and Figure 6.11. We observe that as metastability is resolving itself in `master2`, `slave2` follows. For this reason, I select $\tilde{u}^T = \frac{1}{\sqrt{2}}(\texttt{x2} - \texttt{y2})$, the difference on `master2`'s cross-coupled pair element. This is an example of how the designer states the intended behaviour of the circuit to set up the analysis. The

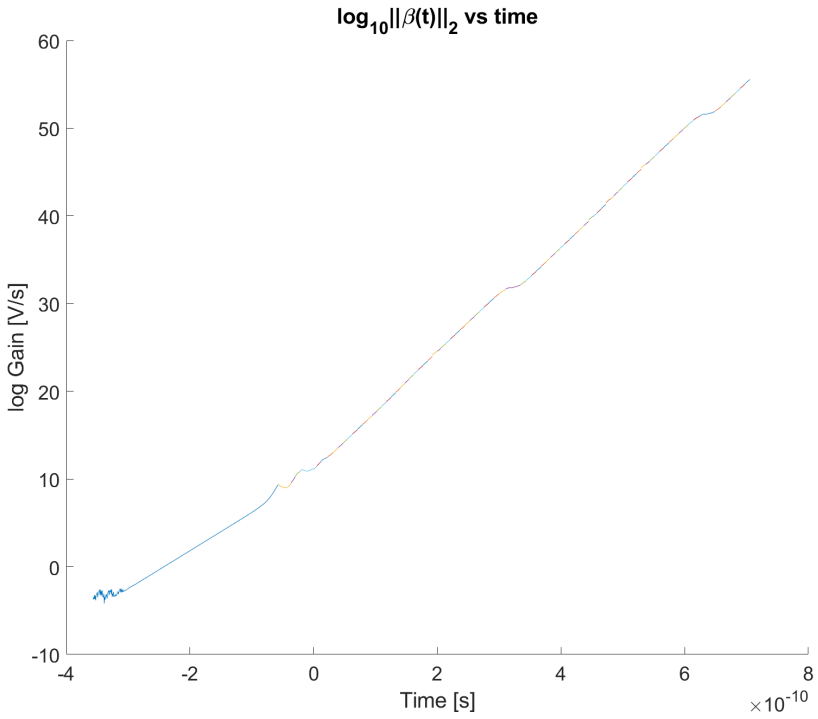

Figure 6.12: Estimate of $\log_{10}||\hat{\beta}(t)||_2$ as the nested bisection simulates forward in time

particular selection of $\tilde{u}^T$ implies by construction of $u(t)^T$ that at t_{eola}:

$$u(t_{eola})^T \vec{V}(t_{eola}) \quad = \quad \frac{1}{\sqrt{2}}(\mathtt{x2}(t_{eola}) - \mathtt{y2}(t_{eola})) \tag{6.3}$$

With $\tilde{u}^T$ selected, we continue our analysis by computing $\beta(t)$ as outlined in Section 3.2. Figure 6.13 superimposes the summary of $\beta(t)$ obtained as the nested bisection algorithm progress using Equation 6.2 and our AD method. We observe that qualitatively they both exhibit the expected exponential growth and both have similar characteristic shapes, but otherwise there is a clear divergence between the two.

The experiments and discussion found in Section 4.1 and Section 4.2 are motivated by the diverging results between the two methods of computing $\beta(t)$ shown

in Figure 6.13. The main observation in Figure 6.13 is that $\log_{10}||\beta(t)||_2$ as computed via AD grows at a slower rate than the one obtained from numerical differentiating. Section 4.1 and Section 4.2 outline experiments we conducted between the two methods. The results from Chapter 4 provide support that our AD method of computing $\beta(t)$ is more accurate.

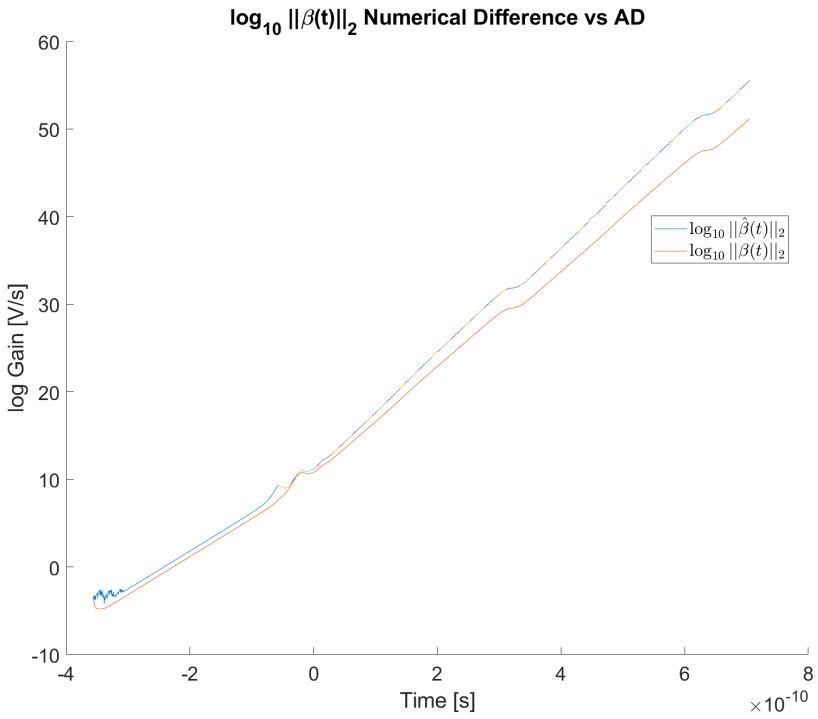

Figure 6.13: Comparing numerical estimated $\hat{\beta}(t)$ vs. $\beta(t)$ via AD

Prior descriptions of metastability in synchronizers often include a discussion of metastability "moving" from one stage to the next, and the circuitry between stages is usually neglected, except for a remark that it somehow contributes to T_w. Our analysis views the synchronizer as a time varying, linear system. The term $u(t)^T$ identifies what nodes in the synchronizer are most significant at time t. $u(t)^T$ highlights very clearly what nodes matter most to metastability resolution at time t_{eola} and how those nodes of importance change in time. Thus, $u(t)^T$ demonstrates that at some times, the cross-coupled pair of a particular latch is the most critical

subcircuit of the synchronizer and at other times, $u(t)^T$ may "point" to another latch. For t near a clock edge, $u(t)^T$ quantifies the impact of circuitry that couples latches. Figure 6.14 and Figure 6.15 show the time varying nature of metastability while $u(t)^T$ highlights what nodes matter at which time for each latch in the passgate synchronizer.

Notice that in Figure 6.14 the weight of $u(t)^T$ only moves to `slave1` as `slave1` becomes opaque, prior to that effectively all of the weight is on `master1` because any perturbation on `master1` will directly influence `slave1` and not the other way around. Figure 6.15 illustrates that `slave2` never has any significant weight associated with it because metastability is resolving itself in `master2` while `slave2` is still transparent.

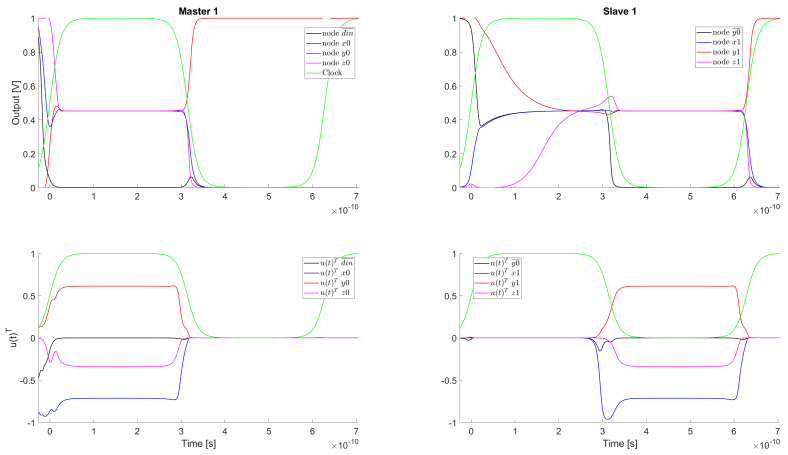

Figure 6.14: $u(t)^T$ highlighting which latch is in play in the first flip-flop for all time t during metastability

$u(t)^T$ provides designers a tool which focuses the attention to the parts of the circuit which matter at a particular time t. This is not to say that making changes to a stage will not affect the overall performance, but $u(t)^T$ provides a magnifying glass to highlight the order in which nodes matter which may influence how a designer thinks about making design improvements. In other words $u(t)^T$ highlights which nodes of the circuit at the present time are most significant in determining the synchronization outcome at time t_{crit}. When the clock signal is close to 0 or 1, then

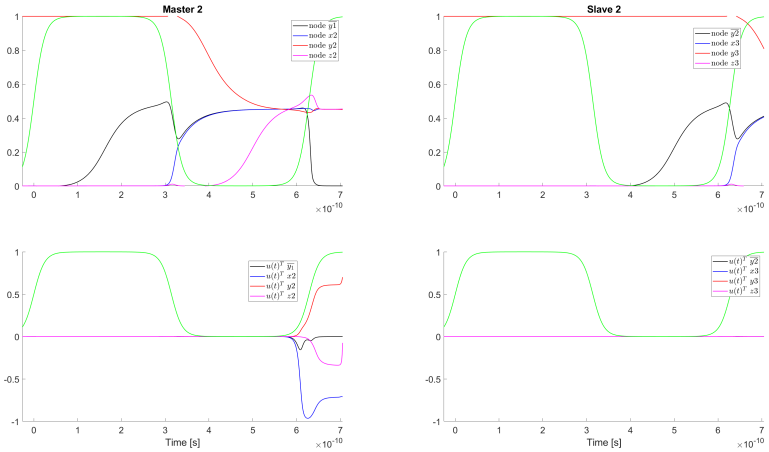

Figure 6.15: $u(t)^T$ highlighting which latch is in play in the second flip-flop for all time t during metastability

$u(t)^T$ will identify the nodes in the cross-coupled pair of inverters for a particular latch as being the most significant. Informally, we can say that the metastability is located at that latch. In the same manner, $u(t)^T$ tracks how metastability propagates through the coupling circuitry and into the next latch when the clock changes. The big advantage of our formulation, is that $u(t)^T$ gives a clear, quantitative description of what it means for metastability to "move".

Finally, the instantaneous gain $\lambda(t)$ of the synchronizer is shown in Figure 6.16. The clock is plotted as an anchoring time reference with which the designer can reference in order to remind the designer what is happening to the circuit state $\vec{V}_{meta}(t)$ at a particular time t while looking at this new figure of merit. The first most obvious feature of Figure 6.16 is that the gain of the synchronizer drops dramatically during the clock transitions. Figure 6.17 shows that in the neightbourhood of the first clock transition, i.e. for t close to t_{clk}, $\rho(t)$ is responsible for establishing the first "time-to-voltage" conversion in order for the synchronizer to enter metastability. The contribution of $\rho(t)$ does not appear in Figure 6.16, but Figure 6.17 shows that the contribution of $\rho(t)$ in that time is positive. The following clock transition shows how transferring metastability from one stage to the next results in performance degradation. The instantaneous gain $\lambda(t)$ does not go to zero but

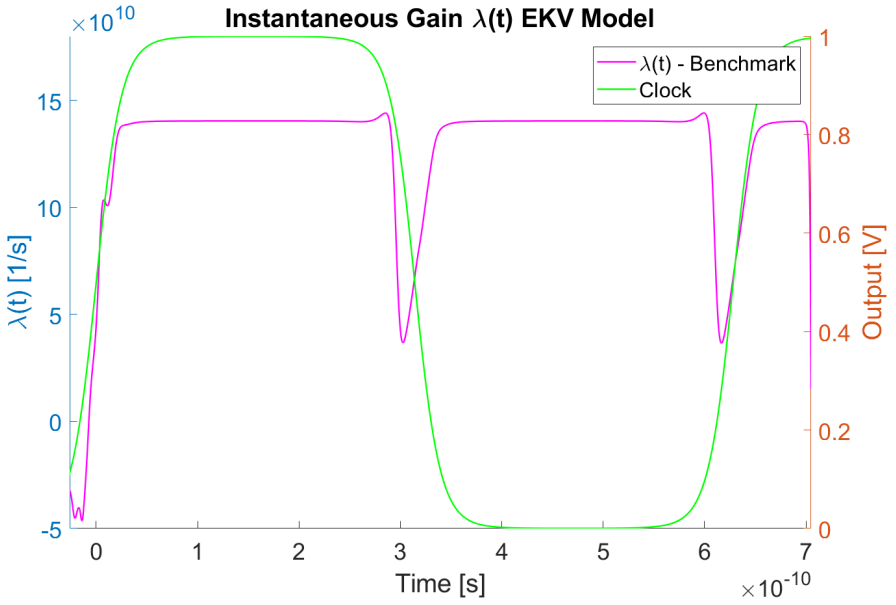

Figure 6.16: Instantaneous gain for passgate synchronizer with p:n ratio of 1:1

the performance profile tells us that the design is not particularly efficient at transferring metastability from stage to stage. Synchronizer designers have appreciated that the coupling logic is less effective than the cross-coupled pairs of inverters at resolving metastability. For example, by suggesting that the logic delay associated with coupling inverters and passgates should be added to T_w. We are not aware of any prior work that quantifies the impact of the coupling logic.

At this point one might ask why we would desire an "efficient" transition of metastability from one stage to the next. From Equation 3.23 we have:

$$P\{failure\} = \frac{\Delta V_{eola}}{g(t_{homo})P_{clk}} \exp\left(-\int_{t_{homo}}^{t} \lambda(s)ds\right)$$

Thus, larger values of $\lambda(t)$ lead to smaller probabilities of synchronization failure. The dips in $\lambda(t)$ show a decrease in the rate of divergence from the metastable condition at the clock edges. If the clock period is sufficiently long, such small dips have little overall impact. At higher clock frequencies, these losses of gain can be more seriously detrimental and must be considered to obtain a robust synchronizer.

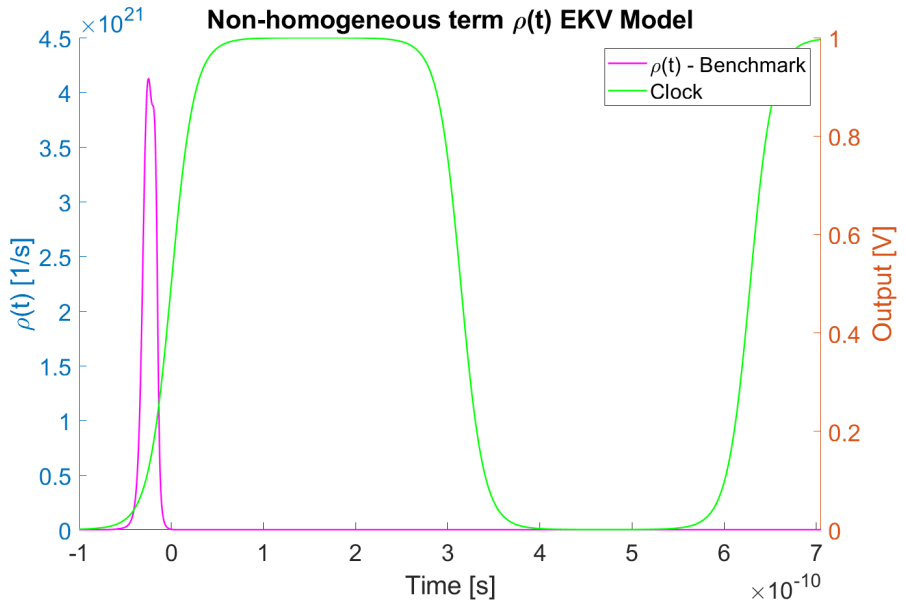

Figure 6.17: Non-homogenous term $\rho(t)$ for passgate synchronizer with p:n ratio of 1:1

We use the terms $g(t)$ and $\lambda(t)$ as characteristics of the benchmark design which we use for comparing the benchmark with variants.

6.3 Passgate Synchronizer Variants

Over the years as metastability in synchronizers has become better understood, designers have proposed improvements to synchronizer designs which aim to reduce synchronizer failures due to metastability, examples can be found in [27][54][2]. Without tools to quantify synchronizer failure probabilities, these "enhancements" have been largely untested.

In this section I explore three design variants and compare them against the non modified benchmark design from Figure 6.7. The first design variant investigates the impact that scan circuitry has on the benchmark synchronizer. The second is to investigate if applying a large perturbation to one of the nodes of the cross-coupled pair of inverters can improve the probability failure and lastly I investigate the impact of modifying the coupling circuitry between stages within the synchronizer.

For each design variant I compare the instantaneous gains $\lambda(t)$ with respect to the benchmark design and compute the log-ratio of the gains: $\log_{10}\left(\frac{g_{Variant}(t)}{g_{Benchmark}(t)}\right)$. The log-ratio of the gains provides a clear comparison between the benchmark and a particular variant to identify which design has a better resultant probability failure. If the log-ratio is positive the design variant is outperforming the benchmark, likewise if it is negative then the benchmark is outperforming the variant, while zero means they are performing identically the same.

6.3.1 Synchronizer with Simulated Scan Circuitry

To improve testability of clock-domain crossing circuits, synchronizers are usually included in testing scan-chains. At the circuit level, this involves adding multiplexors, i.e. passgates, to select whether the latches are configured in "scan mode" or "mission mode" (i.e. normal operations). We simulate the extra capactive load by adding the capacitance equivalent to two inverters to nodes `y1` and `y3` from the schematic shown in Figure 6.7. The expectation is that adding scan circuity will degrade the synchronizer's instantaneous gain profile $\lambda(t)$ because adding capacitance to any node means more current is required to change the voltage on said nodes.

Figure 6.18 shows a comparison between the benchmark and scan loaded instantaneous gains $\lambda(t)$. Indeed, the scan loaded synchronizer has a performance decrease as expected and a resultant probability failure which is greater than that observed by the unmodified benchmark design shown in Figure 6.19. This experiment gives an intuitive feel for how to view the instantaneous gain summary $\lambda(t)$ of the synchronizer design. It also highlights that careful placement of scan circuitry may preserve synchronizer performance and is a suggested synchronizer improvement technique in [2].

6.3.2 Synchronizer Kicking

The first bit of folklore that we test is that synchronizer performance can, allegedly, be improved by applying a major perturbation during the settling time. The ideas is to "knock" it out of metastability. The rationale behind the idea is that if the synchronizer is metastable, applying a large perturbation to one of the nodes of

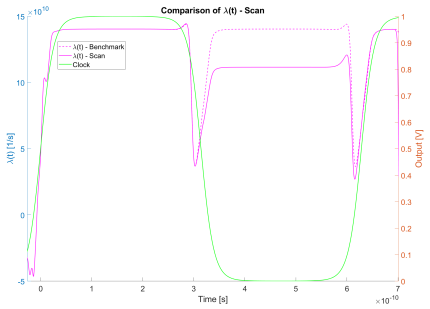

Figure 6.18: Instantaneous gain comparison between the passgate synchronizer without and with scan loading

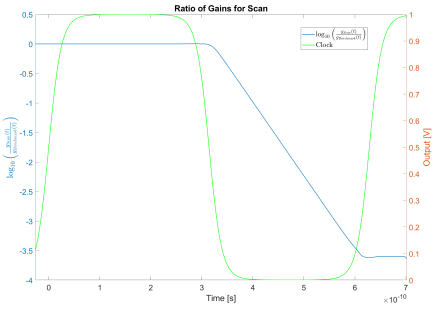

Figure 6.19: log-ratio of the gain between the scan loaded vs. benchmark: $\log_{10}\left(\frac{g_{Scan}(t)}{g_{Benchmark}(t)}\right)$

the cross-coupled pair will drive the synchronizer out of metastability thus "solving" the metastability failure mode. I call this idea kicking the synchronizer. The investigation of kicking a synchronizer was presented in [40].

The perturbation is done as part of the simulation in the middle of the clock period during which `master1` is opaque and `slave1` is transparent. Furthermore, my experiment was performed by stopping the integration of the circuit at the time the perturbation is supposed to be applied (i.e. in the middle of the clock period), modifying the value of `x1` (and correspondingly `z1` because that node would equivalently be affected), and then resuming integration. This analysis ignores the extra capacitances that would be introduced by a real "kick" circuit, thus I name this a "magic kick". Applying the "magic kick" clearly favours the kicking approach.

In this particular example, I select the time at which the kick occurs to be at $t_{kick} \approx 4.4 \times 10^{-10}$s. Figure 6.20 shows the trajectories produced by the nested bisection algorithm for the kicked synchronizer and Figure 6.21 shows the resultant "perfectly" metastable trajectory that the nested bisection finds which quantifies the metastability failures of the kicked synchronizer. In this trajectory, the first latch diverges from its balance point prior to the kick in precisely a way such that the kick returns the latch to nearly perfect balance. In essence the nested bisection algorithm "anticipates" the kick and places the synchronizer in a state such that after t_{kick}, the circuit state is put back to where $\vec{V}_{meta}(t)$ would have been if the kick

had not occurred.

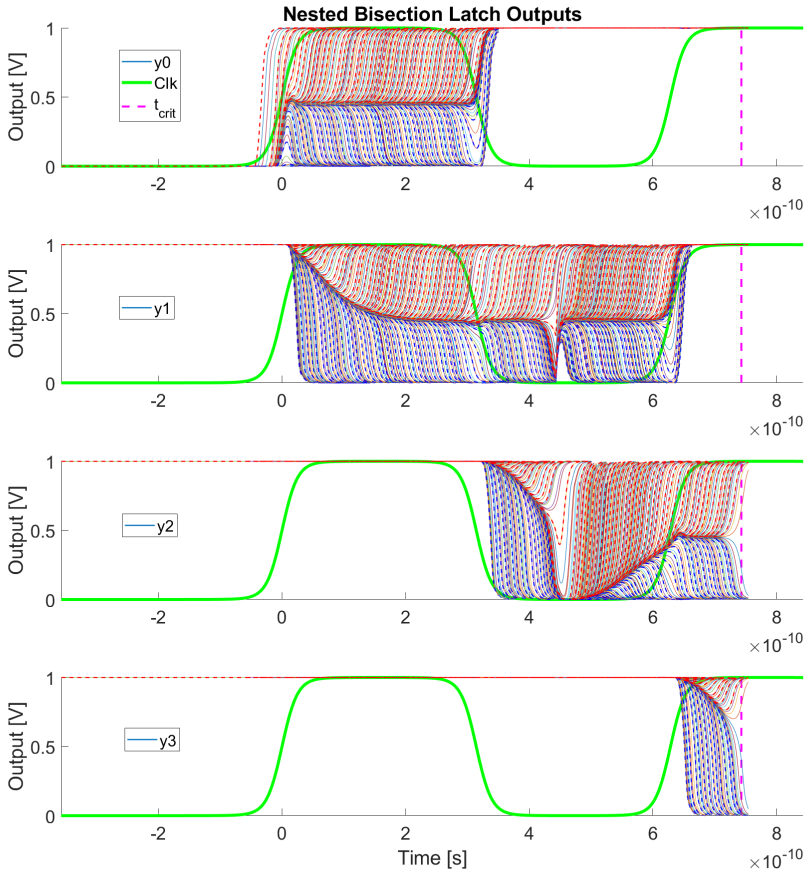

Figure 6.20: Nested bisection simulation runs for Synchronizer with kick

Figure 6.21 investigates voltage nodes `x1` and `y1` of the cross-coupled pair of inverters in `slave1` to better observe the effect of this large perturbation. In order to evaluate the effectiveness of the kicked synchronizer, it is compared against the benchmark design. I make several observation from these plots:

- A synchronizer needs to have the ability to output a logical 1 or 0 at t_{crit}, which means perturbing the synchronizer at time t_{kick} with the goal of having

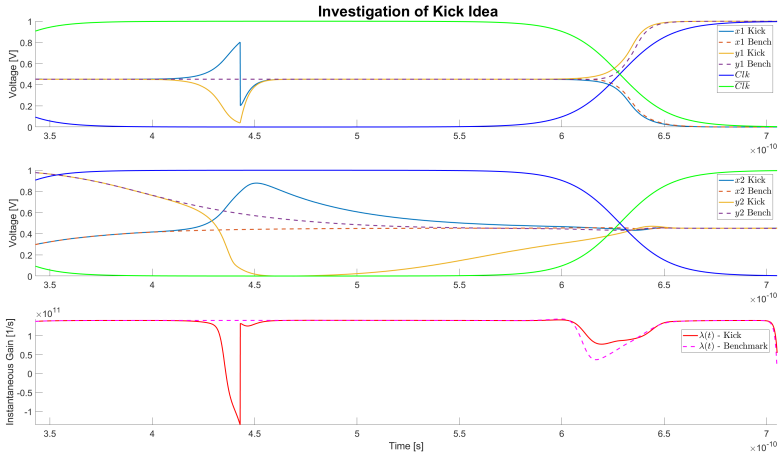

Figure 6.21: Synchronizer kicking results

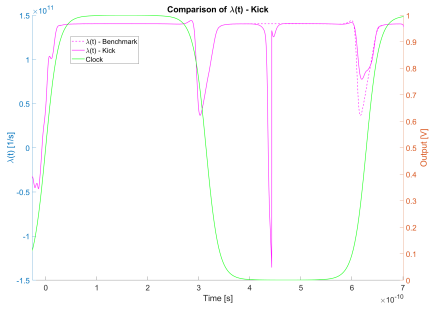

Figure 6.22: Instantaneous gain comparison between kicked and non-kicked synchronizer

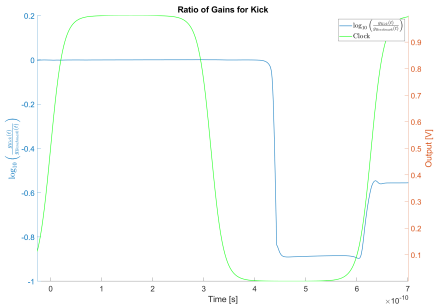

Figure 6.23: log-ratio of the gain between the kick design vs. benchmark: $\log_{10}\left(\frac{g_{Kick}(t)}{g_{Benchmark}(t)}\right)$

a deterministic outcome defeats the intended functionality of a synchronizer.

- Because of the previous point which says that at t_{crit} we must have the ability to output a 0 or 1, there must be a metastable trajectory for any synchronizer design with or without the kick that will produce any value between 0 and 1 by time t_{crit}. This means that the nested bisection algorithm *will* find a trajectory such that there is a metastable trajectory across the kick.

- Kicking the synchronizer becomes a very obviously bad idea. In order to have the synchronizer be metastable across the kick, the synhronizer is required to nearly resolve such that the kick knocks the synchronizer *back* into metastability, doing *exactly* the opposite of what is intended.

The comparison of instantaneous gain profiles between the kicked and benchmark designs in Figure 6.22 immediately shows that kicking the synchronizer is a bad idea. There is a large gain penalty at the time of the kick which harms the performance of the synchronizer. The results shown in Figure 6.23 demonstrate that the benchmark synchronizer design has a better gain ratio as compared to the kick design implying that the kicked synchronizer has a smaller MTBF. Observe that Figure 6.23 and Figure 6.22 show that the kick design variant has a smaller gain penalty at the following clock edge as compared to the benchmark. However, the penalty incurred due to the "magic kick" does not make up for this increase in gain at that later time.

6.3.3 Synchronizer with Offset

Another idea that designers have contemplated is to introduce an offset in the coupling inverter between synchronizer stages. In this case, I introduce an offset by changing the coupling inverter's p:n ratio which goes between the nodes `y0` and $\overline{\texttt{y0}}$ while maintaining the same overall total transistor width (we assume that this will roughly preserve the overall capacitance), i.e. the inverter that connects `master1` and `slave1` of the first flip-flop. The rationale behind introducing the offset is that it shifts the metastable point on the output of the coupling inverter away from the nominal metastable level. This has the effect of requiring the signal on node $\overline{\texttt{y0}}$ to swing further in order to maintain metastability as `slave1` transitions from transparent to opaque. The idea is that this has a net effect of decreasing the probability of metastability to propagate through to the next stage.

The hypothesis is that this idea will hurt the synchronizer's performance for reasons similar to what we observed with the kicked synchronizer: the latch whose state is diverging from its balance point experiences a decrease in gain. The nested bisection algorithm trajectories in Figure 6.24 displays the effects of introducing the offset in the coupling inverter between `master1` and `slave1`. The output `y1` of

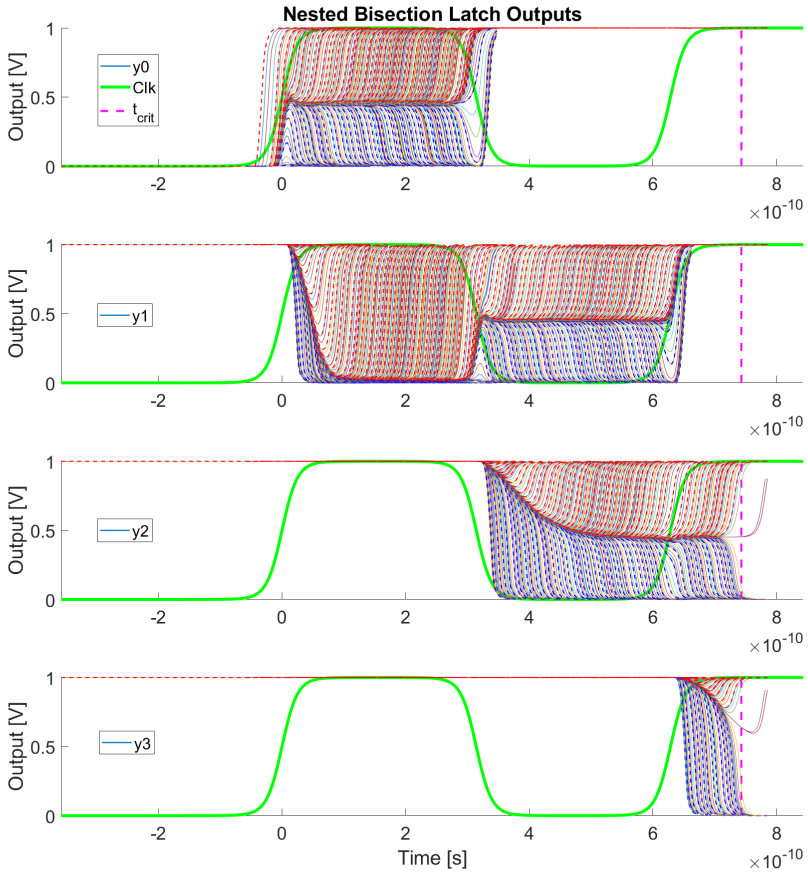

Figure 6.24: Nested bisection trajectories for offset synchronizer

`slave1` as a result is nearly at *gnd* and illustrates how the offset has dramatically shifted the metastable voltage level for the output of that particular stage. However despite this output level seemingly at *gnd*, this signal is not stable. To observe this, Figure 6.25 shows that node `x0` which drives node `y0` is metastable. The output $\overline{\texttt{y0}}$ is the input to `slave1`, and we can see that node $\overline{\texttt{y0}}$ is metastable but at a higher voltage level because of the offset introduce by the inverter. Because of this higher voltage level, `x1` is closer to V_{dd} which drives node `y1`, the output inveter of `slave1`

which has no offset, near *gnd*. Thus `x0`'s metastability is propagating to the output `y1` in such a way that makes `y1` appear to be near a logical 0 level. This illustrates another potential trap that designers may fall into, which is to say that a signal may appear to have logically defined behavior but is in fact subject to change at any moment as a result of metastability resolution.

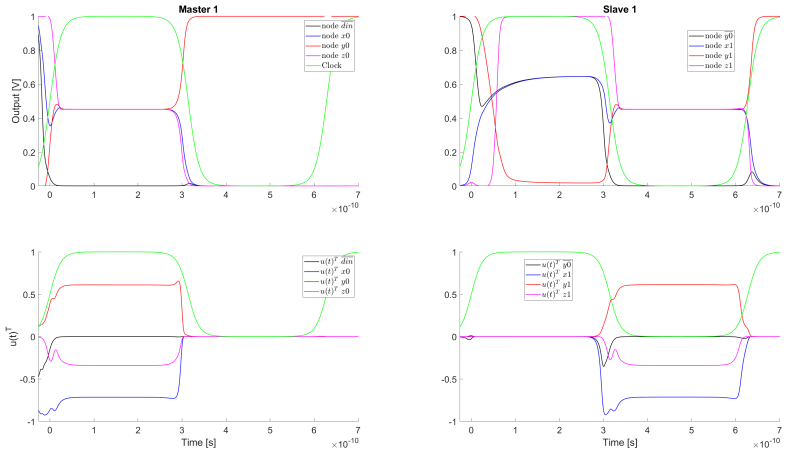

Figure 6.25: $u(t)^T$ highlighting which latch is in play in the first flip-flop for all time t during metastability

Investigating the instantaneous gain profiles in Figure 6.27 and the log-gain ratio in Figure 6.28 produce results that are not expected. Figure 6.28 shows that the offset design has a briefly lived strong benefits as compared to the benchmark when `slave1` transitions from transparent to opaque. We also observe that on the time interval between the first rising and falling edge of the clock signal that the offset synchronizer outperforms the benchmark design. Furthermore there is another small advantage given to the offset design at the last rising edge which suggests that introducing this offset has changed the circuit in ways that had not been anticipated. We investigate this design variant in greater detail in the following section using the decomposition tool developed in Section 3.7.

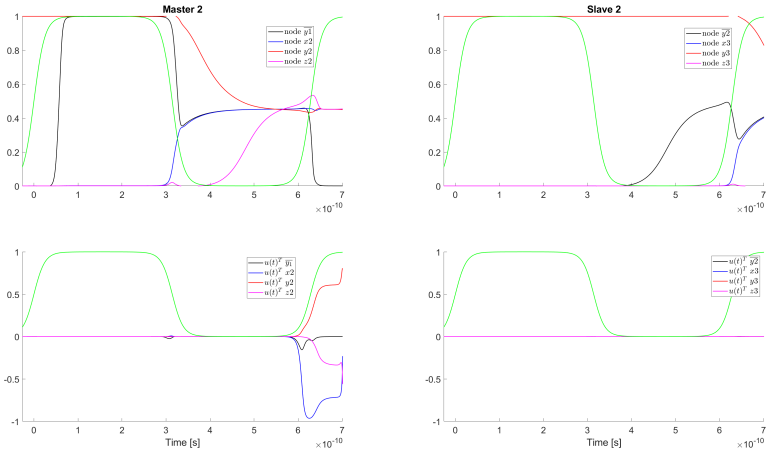

Figure 6.26: $u(t)^T$ highlighting which latch is in play in the second flip-flop for all time t during metastability

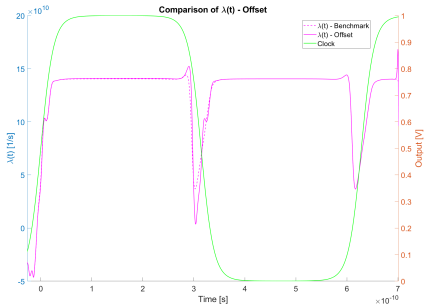

Figure 6.27: Instantaneous gain comparison between offset and non-offset syncronizer

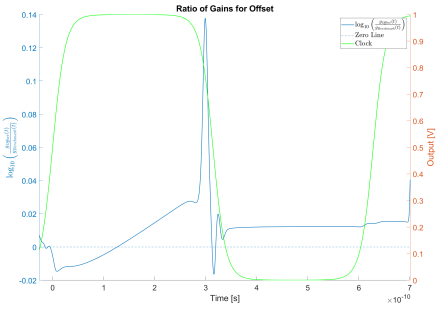

Figure 6.28: log-ratio of the gain between the offset design vs. benchmark: $\log_{10}\left(\frac{g_{Offset}(t)}{g_{Benchmark}(t)}\right)$

6.4 Investigating the Synchronizer with Offset

In the previous section, our results suggest that the two flip-flop synchronizer with an offset in the coupling inverter slightly improves the design. Figure 6.28 shows that there is a significant gain contribution in the offset synchronizer as compared to the benchmark when `slave1` becomes opaque and `master2` becomes transpar-

ent (i.e. at the falling edge of the clock). Figure 6.27 does not provide an obvious explanation for the difference between the two designs so we employ our decomposition tool discussed in Section 3.7 to further investigate what differences exist between the offset and benchmark synchronizers.

The two flip-flop synchronizer design muddles the gain decomposition analysis between the offset and benchmark. We noted previously that $\rho(t)$ is responsible for launching the synchronizer into metastability, which means that $\rho(t)$ is responsible for initially putting `master1` into metastability. In Section 6.3.3, I changed the p:n ratio of the inverter which connects `master1` to `slave1`. To eliminate the possibility of $\rho(t)$ contributing to different startup conditions, we analyze a three flip-flop passgate synchronizer shown in Figure 6.29 and introduce the offset in the coupling inverter `inv9` which connects `master2` and `slave2`.

Analyzing the three flip-flop design shown in Figure 6.29 allows us to fully isolate and investigate the differences between the two designs by removing the possibility of any differences introduced by the initial onset of metastability. Figure 6.30 shows that the gains between the offset and benchmark design leading up to the point where `master2` becomes the important subcircuit to metastability resolution are identical. We note that the characteristics that appear in Figure 6.30 and Figure 6.28 are very similar. This first qualitative comparison suggests consistency in the behaviour of introducing an offset into the design.

For each component I show for both the benchmark and offset synchronizers the following:

- The voltage node at the component's input and output and the clock signal.
- The importance those nodes have on the overall circuit as dictated by $u(t)^T$.
- The gain contribution of the component as described in Section 3.7.
- The log-ratio comparison $\log_{10}\left(\frac{g_{Offset}(t)}{g_{Benchmark}(t)}\right)$.

I superimpose in each of the subplots $u(t)^T$ for the nodes in question to show how much influence those nodes have towards metastability resolution on the circuit as a whole.

By considering the component-wise contributions to the gain, we can see how each part of the synchronizer contributes differently to the overall gain for the two

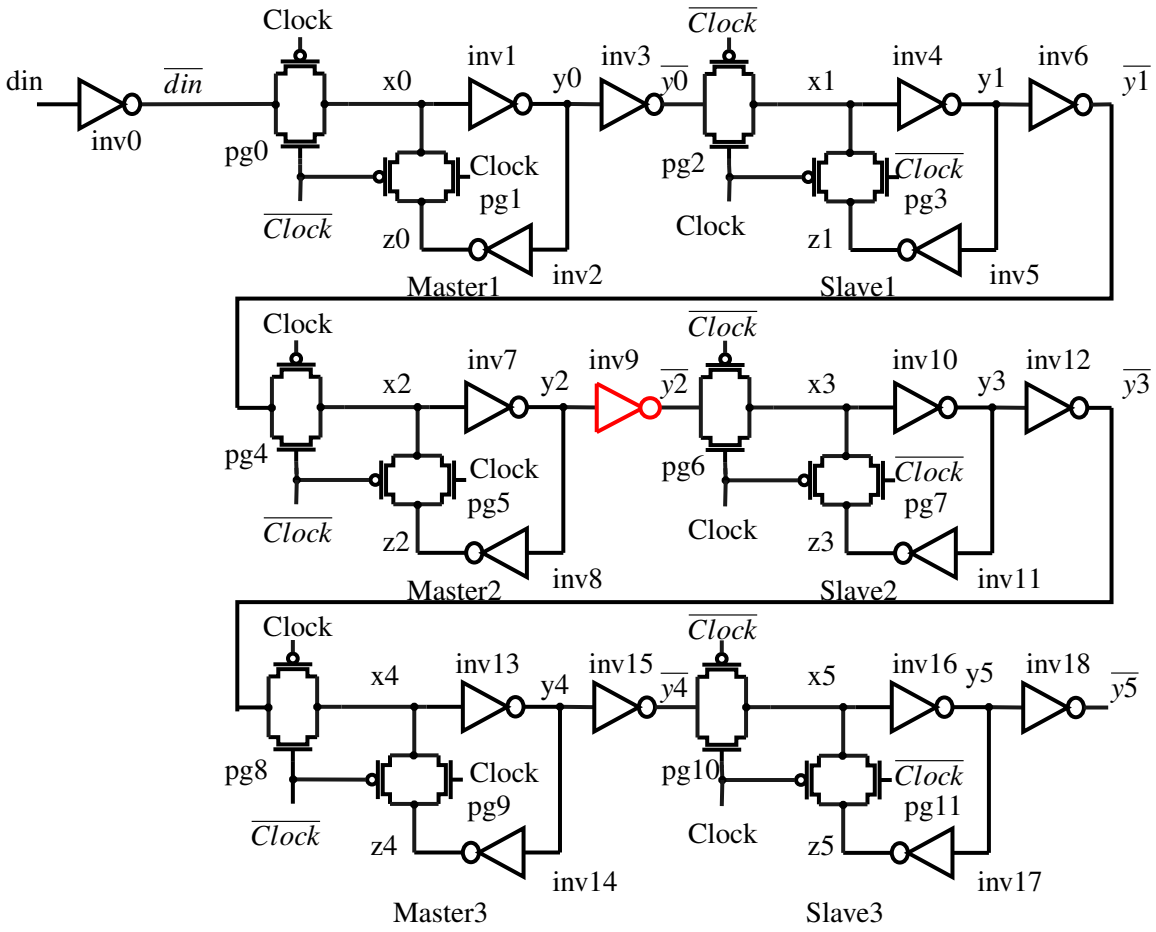

Figure 6.29: 3 flip-flop passgate synchronizer

designs. We observed many second-order effects due to the change in operating points caused by the offset and non-linearities in the circuits themselves. Some of these increase the gain of the offset design relative to the benchmark, while others cause a decrease. The combined effect is a slight increase in the total gain that cannot be properly attributed to any single cause. We describe below the impact of the offset design on each affected inverter and passgate.

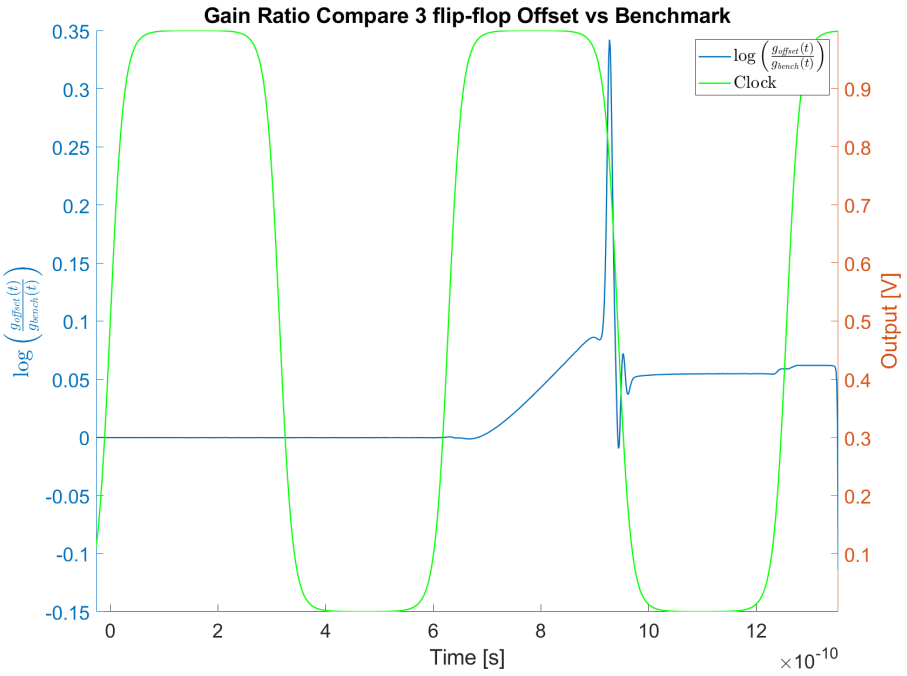

Figure 6.30: Three flip-flop offset vs. benchmark log-ratio: $\log_{10}\left(\frac{g_{Offset}(t)}{g_{Benchmark}(t)}\right)$

6.4.1 The offset coupling inverter and passgate

The immediate observation from Figure 6.30 (which is similar in Figure 6.28) is that there is a sharp rise in the effectiveness of the offset design immediately followed by a sharp drop which settles to a net advantage for the offset design. We home in on that behaviour by starting our investigation with the component that has changed: `inv9`. Figure 6.31 shows a sharp rise in gain at the clock edge of interest followed by a sharp fall in the gain. As seen in Figure 6.30, the total gain contribution of `inv9` settles to a value for the offset synchronizer that is *less* than the gain for the benchmark. Thus we must look at other components to find the cause of the offset synchronizer's advantage. We also note that $u(t)^T$ only considers `inv9`'s output $\overline{\texttt{y2}}$ for a brief period of time.

We then investigate `pg6`, the passgate that connects `master2` and `slave2`. Fig-

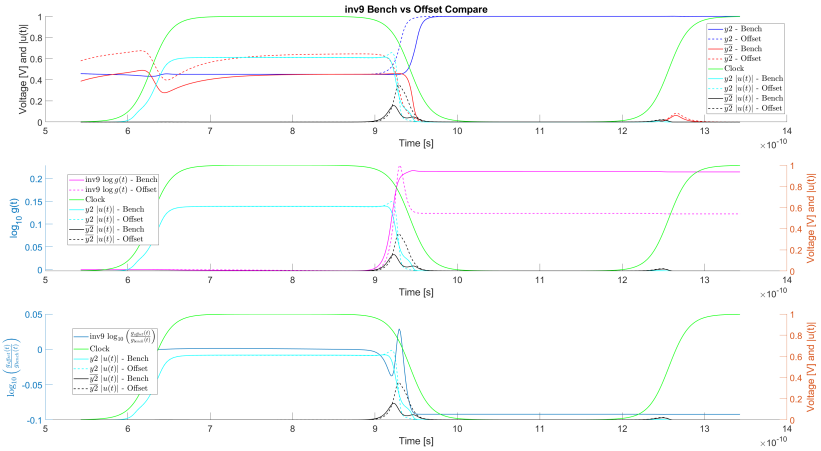

Figure 6.31: Three flip-flop component analysis: `inv9`

ure 6.32 shows that this passgate contributes a clear advantage for the offset design as compared to the benchmark. Throughout the time period between clock edge transitions, node `x3` matters. As with `inv9`, a small but non-zero amount of weight is given to node $\overline{\text{y2}}$ at the rising edge of the clock at around $t = 12.5 \times 10^{-10}$s, however, no appreciable change in the gain is observed.

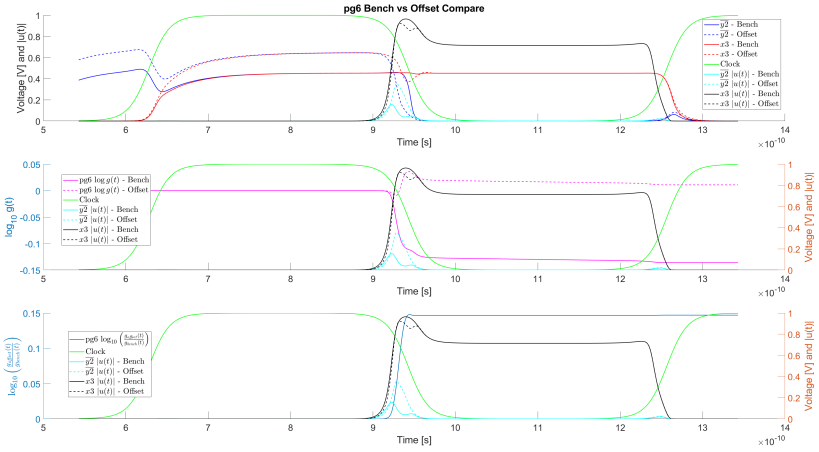

Figure 6.32: Three flip-flop component analysis: `pg6`

We can further decompose pg6 into its individual transistors to determine which transistor is responsible for this improvement. Looking at Figure 6.33 and Figure 6.34 we see that the PMOS device in both the offset and benchmark designs are contributing negatively and insignificantly (on the order of 10^{-3} difference between the offset and benchmark) to the circuit whereas the NMOS device accounts for the observed gain in Figure 6.32. We see this behaviour because as the clock signal is changing, for a brief moment in time the circuit is configured as a common gate amplifier shown in Figure 6.35. In more detail, with the offset design, node $\overline{\texttt{y2}}$ drops earlier in the offset design. This increases the gate-to-source voltage of the NMOS device in pg6, and thus increase the transconductance, g_m, of that transistor. In other wrods, the drain-to-source current of this transistor has a greater sensitivity to the voltage on node $\overline{\texttt{y2}}$ in the offset design than in the benchmark. This is how the passgate acts as an active device and makes a positive contribution to the overall gain in the offset design but not in the benchmark.

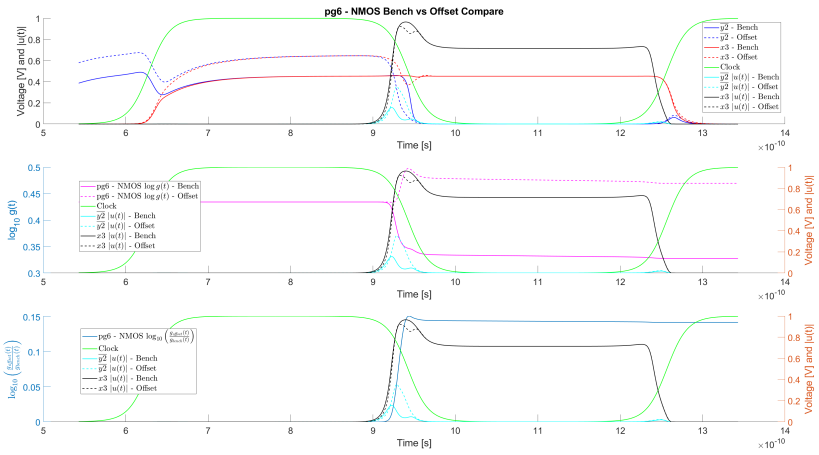

Figure 6.33: Three flip-flop component analysis: pg6 – NMOS

6.4.2 The master2 cross-coupled loop

The plots for inv7 (Figure 6.37) show a higher instantaneous gain for the offset design than the benchmark when the master2 latch is opaque. We conjectured

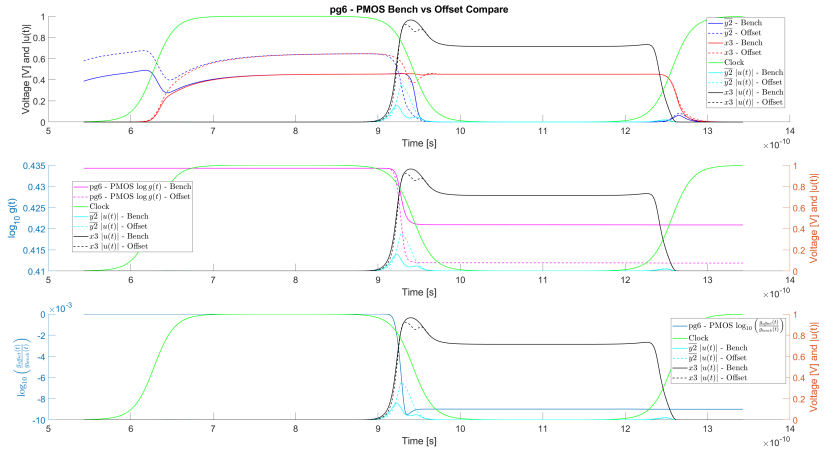

Figure 6.34: Three flip-flop component analysis: `pg6` – `PMOS`

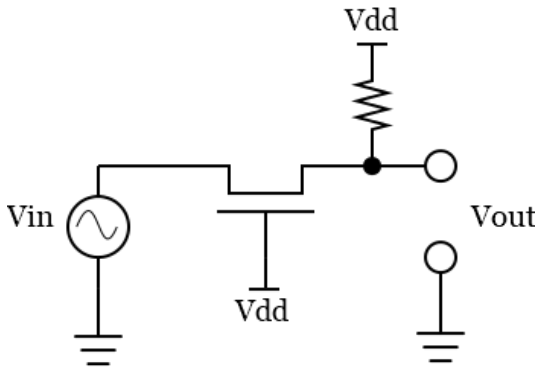

Figure 6.35: Common Gate Amplifier circuit with NMOS

that the offset design has less total capacitance on node `y2` than the benchmark, even though the total transistor width for `inv9` is the same in both designs. Figure 6.36 confirms this conjecture. We suspect that this is a consequence of the non-linear gate-capacitance of the MOSFET transistors, but need to perform further experiments to test this hypothesis. At the clock edge, `slave2` goes opaque, we see a sharp loss of gain for inverter `inv7`. This is because the `master2` latch exits metastability earlier for the offset design than for the benchmark. The combined effect of these two phenomena is that `inv7` has lower overall gain in the offset

design than the benchmark.

Figure 6.38 and Figure 6.39 show the contributions of `inv8` and `pg5`. Both provide better performance for the offset synchronizer than the benchmark. The differences can be attributed to their $u(t)$ functions. Inverter `inv8` makes a positive contribution to the overall gain, and $u(t)$ shows that `inv8` remains relevant slightly longer in the offset design. Conversely, the passgate `pg5` causes a reduction in gain, and its $u(t)$ term shows that it is relevant for slightly less time than in the benchmark design. We have not yet determined a circuit-design-friendly explanation for why $u(t)$ functions are different: for now, they just are.

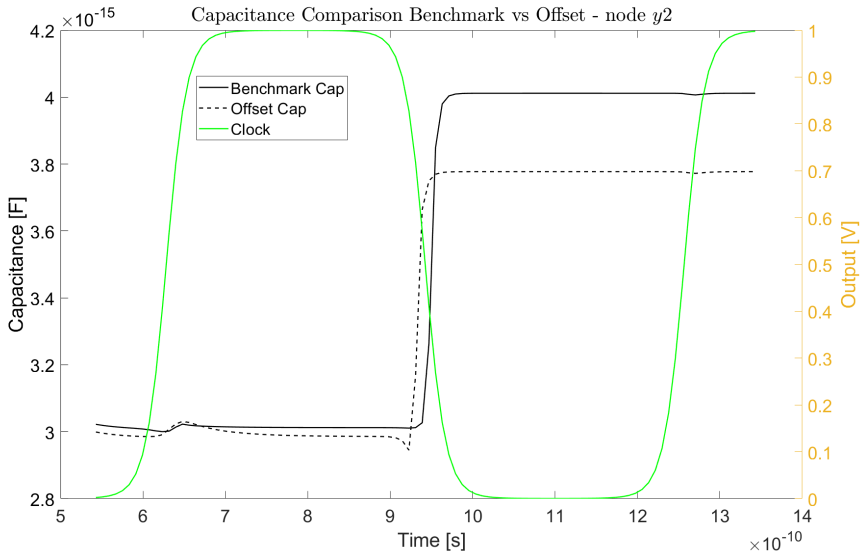

Figure 6.36: Comparison of effective capacitance on node `y2`

6.4.3 The `slave2` cross-coupled loop

Finally we investigate the components in the `slave2` cross-coupled loop: `inv10`, `inv11` and `pg7` with corresponding plots Figure 6.40, Figure 6.41 and Figure 6.42. All three components in the offset synchronizer have a net negative gain contribution at the clock transition, noting that `inv10` contributes the most towards the lower gain for the offset synchronizer than the benchmark.

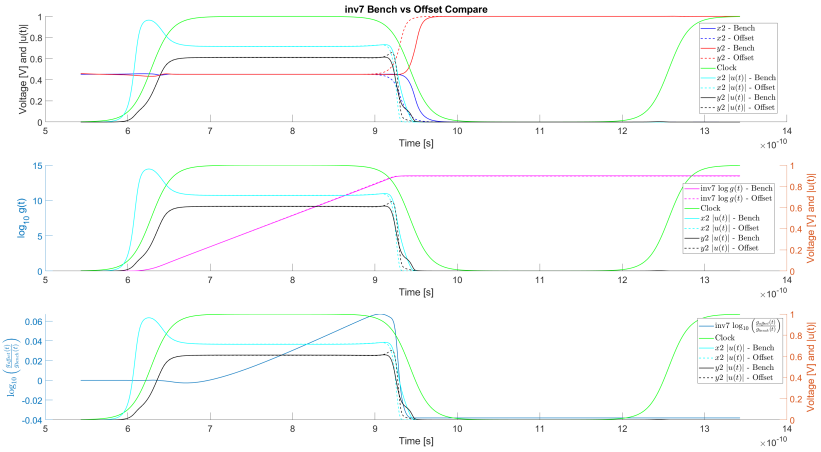

Figure 6.37: Three flip-flop component analysis: `inv7`

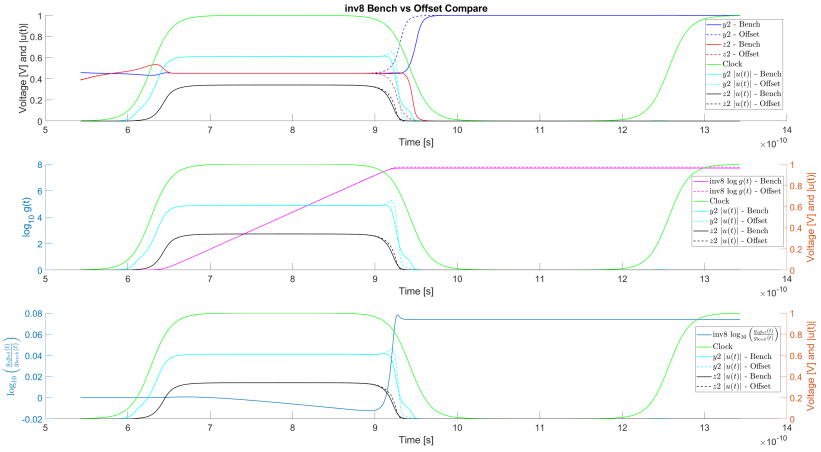

Figure 6.38: Three flip-flop component analysis: `inv8`

6.4.4 Summary of results

Table 6.1 summarizes our results. We see that the combination of `inv9` (the offset coupling inverter), `pg6`, `inv8` and `pg5` in the cross-coupled loop of `master2` contribute positively to the offset design at the falling edge of the clock, while `inv7` and all the components in the cross-coupled loop of `slave2` contribute negatively

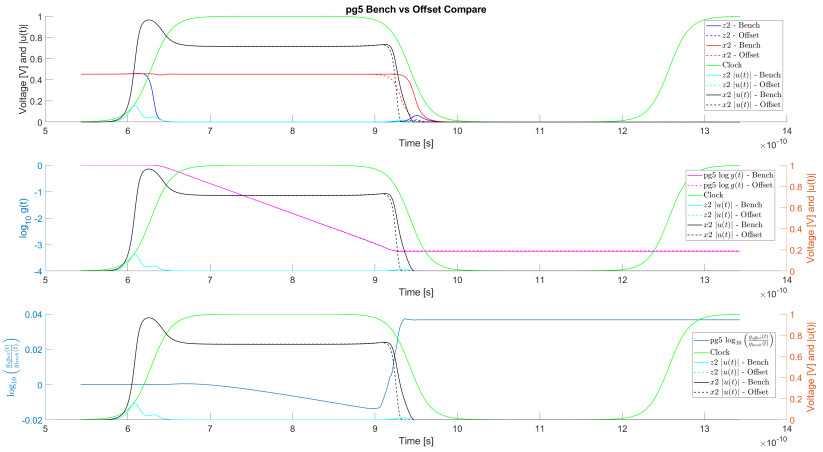

Figure 6.39: Three flip-flop component analysis: `pg5`

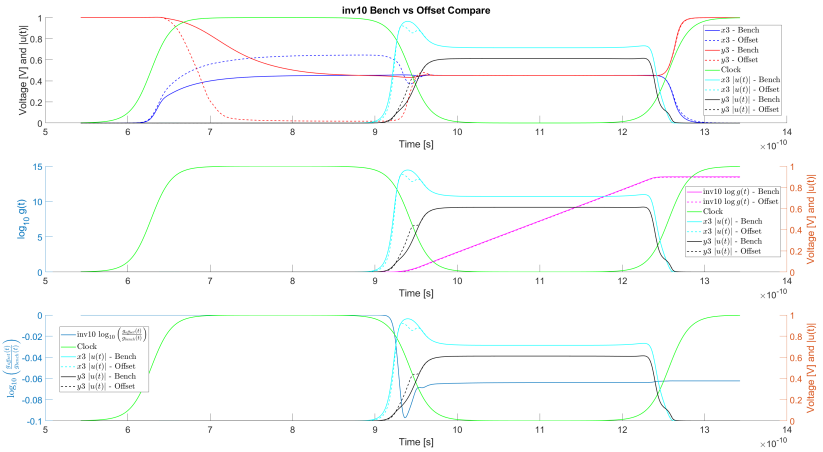

Figure 6.40: Three flip-flop component analysis: `inv10`

to the offset design. The net result is a wild variation in gain between the offset and benchmark design at the clock transition, with a settled net gain favouring the offset design. We note that the net gain that is observed in the two flip-flop design from Figure 6.28 is a factor of roughly $10^{0.02}$ over the benchmark which is small as compared to the scan loaded synchronizer (decrease in performance by over $10^{3.5}$)

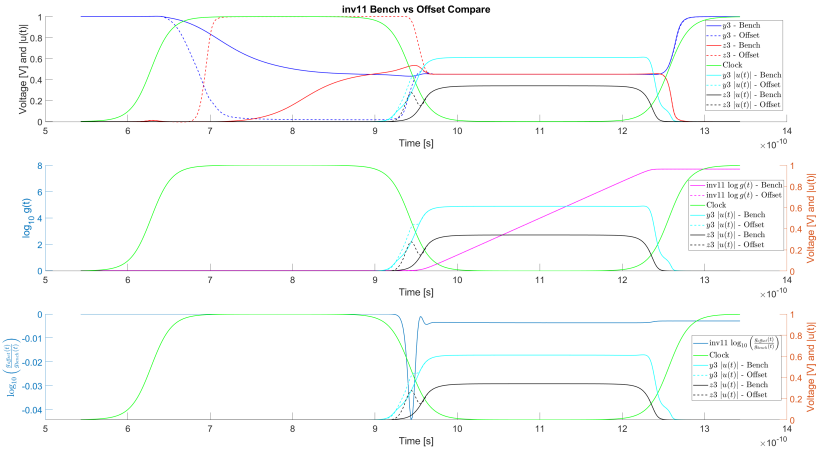

Figure 6.41: Three flip-flop component analysis: `inv11`

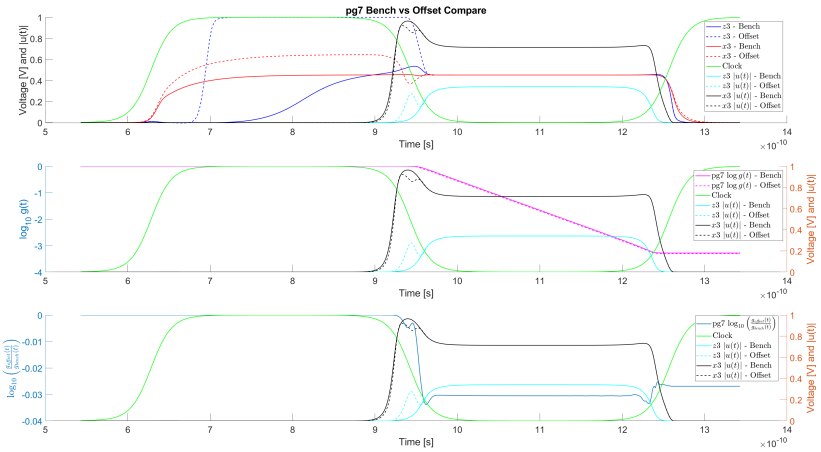

Figure 6.42: Three flip-flop component analysis: `pg7`

and the kicked design (decrease in performance by $\approx 10^{0.6}$).

We note that because of the difference in the effective capacitance depicted in Figure 6.36 on node `y2`, the offset design accumulates gain over the clock period when the `mater` stage associated with the offset coupling inverter is metastable. Figure 6.28 and Figure 6.30 show that despite the significant gain from the passgate

Table 6.1: Summary of offset and benchmark gain contributions

Component	$\log_{10}\left(\frac{g_{Offset}(t_{eola})}{g_{Benchmark}(t_{eola})}\right)$
`master2`	
`inv7`	-0.0381
`inv8`	0 .0738
`pg5`	0 .0457
`inv9`	-0.0922
`slave2`	
`pg6`	0 .1470
`inv10`	-0.0623
`inv11`	-0.0381
`pg7`	-0.0339
Σ Components	0 .0019
Overall 3 flip-flop design	0 .0269
Difference	-0.025

acting as an amplifier at the falling clock transition, the offset design only comes out marginally ahead. We believe that if the clock period was shortened sufficiently, that the offset design would be worse than benchmark. On the other hand if the clock period is larger then the offset design improvement is made larger. The total of the per-component log-gains does not quite match the overall difference between the two designs. This suggests looking deeper into the analysis and the circuits to find an explanation, or maybe we should just change the clock period to get a clear distinction between the two designs!

In the course of performing this analysis, I made two distinct discoveries. The first, is that generally, designers think of passgates as being passive devices where the introduction of the offset makes the passgate a small-signal amplifier for a brief period of time. The second, is that we assumed that introducing the offset in `inv9` (and the coupling inverter of `master1` in the two flip-flop design) by keeping the total width of the transistors fixed would preserve the overall capacitance of the circuit. In reality, this assumption was violated and brought to light subtle secondary effects which were not considered. Our tool allowed us to discover and explain the discrepancy in the behaviour of the passgate between the two designs

and expose that our underlying assumption about capacitance was incorrect.

We have shown how our new tool can be systematically used to explain big picture behaviour in the synchronizer. However, there are other aspects to Figure 6.30 which remain a mystery and motivates future investigation. One such unexplained anomaly is seen in Figure 6.30 at the last rising edge of the clock where a small step in performance is suggested to be present in the offset design which continues to elude us. Given what we have already discovered, we believe these subtle behaviours are continued secondary effects from the non-linear capacitance model we employ.

6.4.5 Offset investigation remarks

The tools developed in Chapter 3 have been fully exercised to uncover why the offset has the observed behaviour in the passgate synchronizer design at the clock transition. The reason is subtle, one which has not been argued before to my knowledge. The improvement observed in the offset design as compared to the benchmark is repeatable but small. This leads me to believe that the benchmark synchronizer with all transistors set with a p:n ratio of 1:1 at its current operating condition is potentially not optimal. The way the information is presented with this new set of analysis tools allowed us to systematically narrow down the reason for the major discrepancy between the benchmark and offset designs leading us to a new discovery! A circuit component which designers often perceive as a passive circuit element (the passgate in this case) turned out to behave in an active way for a brief moment. This discovery explained the observed behaviour after some investigation in textbook circuit designs. We also discover that a small difference in the effective capacitance on the node between the cross-coupled loop and the offset coupling inverter accounts for the accumulation of gain in the offset design as compared to the benchmark. We believe that this small accumulation in gain is key to the offset design outperforming the benchmark.

Recall that in the kicked synchronizer variant shown in Figure 6.23 we observed an increase in the log-gain ratio at the last rising edge of the clock (not enough to overcome the harm done by the kick). I believe that this boost in gain at that last clock transition is also due to the passgate element acting as an amplifier

because as the synchronizer recovers from the kick its output voltage is in the process of returning to where it should be to maintain metastability – thus at the clock transition we get a boost from the small-signal amplifier present in the circuit.

My analysis tool has put into question common intuitive arguments that designers make about synchronizers and metastability. Some of the intuitive notions have been confirmed while others have been debunked with an accompanying explanation. The conclusion that can be drawn is that the set of tools outlined in Chapter 3 are useful for synchronizer analysis.

Our analysis has not accounted for all the outstanding anomalies. We observe in Figure 6.30 and Figure 6.28 a small improvement in the offset design at the last clock transition for which we have not accounted for. We also assumed that keeping the total transistor width constant in both designs would yield a constant capacitance matrix for the two designs. Figure 6.36 shows that this assumption is violated. We acknowledge that the full capacitance model described in Section 5.3.2 is non-linear and may account for the other unexplained discrepancies between the designs.

Further analysis is required to understand the tradeoffs in sizing transistors differently. However, our new tools not only provide a mathematical model to quantify the synchronizer's probability failure, but can be used to quantify performance characteristics and compare design ideas. This has lead to new discoveries about synchronizer behaviour and provides a foundation for synchronizer optimization.

Chapter 7

Conclusion and Futurework

7.1 Summary

This thesis has presented novel methods for the analysis of synchronizer circuits. We started with Yang and Greenstreet's nested bisection algorithm which they used to compute failure probabilities [51] and produce failure trajectories [52]. In Chapter 3, we showed (see Equation 3.8) how to compute $\frac{\partial}{\partial t_{in}}\vec{V}(t)$ where t_{in} is the time of an input transition of the synchronizer, and $\vec{V}(t)$ is the synchronizer state at time t. We then presented a compuation for $u(t)$, where $u(t)$ is a unit-vector such that perturbation of the synchronizer state in durection $u(t)$ at time t map to perturbations in the direction of synchronizer resolution at the end of the synchronization process (see Equation 3.14 and Equation 3.15). Intuitively, $\beta(t)$ connects input transitions to the circuit state at time t, and $u(t)$ connects the circuit's state at time t to the final state at the deadline for synchronization. From these, we derive a notion of cumulative gain, $g(t) = u(t)^T\beta(t)$ that describes how much "progress" the circuit has achieved in resolving metastability at time t. Crucially, $g(t)$ is a scalar, and we derive a simple, first-order, linear ODE for $g(t)$ where (see Equation 3.16):

$$\frac{d}{dt}g(t) \quad = \quad \lambda(t)g(t)+\rho(t)$$

Where $\lambda(t)$ is the "instantaneous gain" of the synhcronizer, and the inhomogenous component, $\rho(t)$ describes the initial time-to-voltage conversion of the syn-

chronizer. For real-world synchronizers, $\rho(t)$ rapidly converges to zero after the clock edge that samples the data input.

The formulation for $g(t)$ enables detailed comparison of synchronizer designs. Whereas the methods from [51, 52] allow a designer to compare failure probabilities (and thus MTBFs) of synchronizer designs, $g(t)$ lets a designer see *when* in the synchronization process the differences arise. This added information can provide valuable insight into understanding the trade-offs in the circuit designs. Furthermore, we note that the standard modified-nodal-analysis approach to circuit modeling naturally lends itself to decomposing the time-derivative function for the circuit state, and the Jacobian function for this derivative into a sum of contributions from individual components, e.g. transistors, inverters, logic gates, ect. As described in Section 3.7, this allows our tool to be used to identify both *where* performance differences arise (i.e. in which sub-circuit) as well as *when*. In Chapter 6, I analyzed several synchronizer designs and showed how the new methods can be used to understand their performance characteristics.

To apply the methods from Chapter 3 to the circuits in Chapter 6, we addressed two main issues. Our implementation make extensive use of automatic differentiation (AD). Chapter 4 provides a comparison of the accuracy of automatic differentiation (AD) with numerical approximation using Centered Differencing (CD) for a simple model with a fix-point attractor and a system with switched dynamics that behaves like a synchronizer. Unlike real-world circuit models, both models in Chapter 4 were formulated to allow closed-form solutions. For both models, AD was more overall accurate than CD, with a modest overall advantage for the system with a fixed point attractor, and a clear advantage with the "synchronizer". While this is not a rigorous proof, these examples strongly suggest that AD is advantageous for problems like the ones in this dissertation where we are differentiation the result of numerically approximate algorithm (in this case integration) with respect to one or more parameters of the algorithm. Further investigation is warranted, but the current results justify our use of AD.

Implementing our synchronization analysis methods requires circuit models that are amenable to automatic differentiation. To that end, in Chapter 5, I implemented an AD friendly version of the MVS MOSFET model [23]. In the process, I replaced `if`-statements with algebraically equivalent, differentiable formulas, and

I fixed an error in the MVS root-solver for determining the virtual source and drain voltages. Not surprisingly, model-evaluation time dominated the run-time for the metastability analysis. I developed a simplified EKV model that provides a tolerable approximation of the more detailed MVS or BSIM models while enabling much faster execution of my metastability analysis algorithms.

I tested my approach to metastability analysis using real synchronizer designs in Chapter 6. In particular, I examined a simple passgate based synchronizer along with three variants: adding a scan chain, "kicking" the synchronizer and adding an offset between stages. Using the functions computed by my algorithms, in particular, $\beta(t)$, $u(t)$, and $g(t)$, I showed how my metastability analysis can be used to compare the designs and explain their differences. The analysis also revealed some unexpected behaviours. Most notably, my tool showed how the passgate coupling two latches can be an *active* device, providing gain as a common-gate amplifier.

7.2 Future Work

Over the course of developing the transistor models, the implementation of the nested bisection algorithm and investigating AD methods I noted many near future improvements to be made.

- The current implementation of the nested bisection algorithm expects the user to specify a parameter to test the linearity assumption and a ΔV term to establish t_{eola}. These are two tests for linearity, and it should be possible to use the same condition for both.

- In Chapter 4, we perform some preliminary experiments to determine the trade off of numerical integration of $\int_{t_0}^{t} \frac{\partial}{\partial w}\frac{d}{dr} f(r,w)dr$ to estimates $\frac{\partial}{\partial w} f(t,w)$ compared to numerical differencing of trajectories produced by $\int_{t_0}^{t} \frac{d}{dr} f(r, w \pm \Delta)dr$. AD appears to be more accurate, but some anomalies need further investigation. We also only investigated the performance against a centered difference approximating scheme and did not consider higher order numerical differencing methods.

- Chapter 4 develops a solution to compute $\frac{\partial y}{\partial x}$ via AD where y comes from a

fixpoint algorithm which solves $f(x,y) = 0$. The described solution makes assumptions about f and y. I note that we could relax those assumptions and derive a more general treatment of computing $\frac{\partial y}{\partial x}$ for fixpoint algorithms.

- Section 4.3 provides preliminary ground work to perform an end-to-end error analysis of the nested bisection algorithm. Although the error analysis is not covered in this thesis, doing an end-to-end error analysis is likely to be beneficial for other applications that have multi-dimensional saddle point problems.

- The simplified EKV model in Chapter 5 can be further improved. Classical derivations of velocity saturation models (e.g. [15]) consider a "vertical" term related to V_{gs} and a "horizontal" term,related to V_{ds} with the horizontal term being more significant. We realize in retrospect, that our model captures the vertical term, but not the horizontal one. We expect that a simple change to the model should improve the accuracy of our simplified EKV model without increasing its complexity. Furthermore, our model for attributing gate capcitance between the source, drain, and body is *ad hoc*, and we expect that improvements should be possible here as well. Another correction term computed as a $\log(\sum_i)$ of three operating regions is one solution to explore to get a better fit as the transistor goes from the cut off, to linear to velocity saturated regions. The addition of such a term is likely to make computing the gradient in the fitting routine be more easily readable with the incorporation of AD. The end result would remain as one equation to compute I_{ds} in the same form as shown in Equation 5.6.

- Currently the implementation of the junction capacitance is taken out of the BSIM manual. However, it is likely that a simple, junction diode model with a few parameters should greatly simplify the computation.

- Designs like the boost [27] and robust synchronizer in [54] concern themselves with the power consumption of the synchronizer design when it is metastable. Bounding the total power consumption may be a potential constraint of importance. An extension to my tools could include keeping track

of the net current flowing from nodes connected to V_{dd} which by proxy determines the power consumption of the design in question.

- This thesis ignores process and temperature variation in micro-electronics design. This is known to have a large impact on the jamb latch designs. Finding methods that take process variation into consideration should be explored.

- In this thesis I only consider the two and three flip-flop passgate latch synchronizers. Preliminary work has been started to investigate the jamb-latch. Additional synchronizer designs such as the strongARM latch [26, 36] design is the next logical design to explore. Other designs of interest to compare against would be, Li *et al* boosting voltage supply synchronizer[27], Yang *et al* pseudo NMOS synchronizer[53] and Zhou *et al* robust synchronizer[54]. The proposed designs by these researchers revolve around circuit modifications around the passgate, jamb and strong arm latch designs. Exercising my tools in this thesis to perform a thorough quantitative analysis of these designs would be of interest to the community.

- We have begun investigating ways to leverage AD to automatically apply gradient descent to transistor sizing on the two flip-flop passgate synchronizer design to automatically maximize its gain $g(t)$ by computing $\frac{\partial \beta(t)}{\partial w}$ where w is the vector of transistor widths. Initial results look promising.

7.3 Other Applications

The methodology of selecting a non-linear model, simulating trajectories of interest, automatically develop a linear small-signal perturbation model, and develop a summarizing performance metrics I believe to be applicable to many fields other than circuit analysis. One such field of particular interest to me is in control theory and robotics. Model predictive control and adaptive control schemes in robotics are candidate use cases for this design methodology. In synchronizers the performance criteria is MTBF/resolution time, in robotics it could be time-to-reach, position variance, velocity tracking among many others.

Circuit designers use SPICE in order to perform circuit analysis. Developing a similar analytical framework in the field of robotics is an area I believe contributions can be made.

Another equally interesting application would be to create a framework which can automatically generate correct scientific computing code to compute the desired mathematics. Using AD is very powerful when setup properly. However, as implemented in common scientific computing frameworks such as MATLAB, it is easy to make indexing errors or produce confounding results with mixed units. Developing a language which translates mathematical equations into scientific computing code automatically would go a long way to accelerate the development of new analysis techniques and reduce human error in the implementation of AD objects to achieve the correct mathematical operations.

Bibliography

[1] M. Alshaikh, D. Kinniment, and A. Yakovlev. A synchronizer design based on wagging. In *2010 International Conference on Microelectronics*, pages 415–418, Dec 2010. doi:10.1109/ICM.2010.5696176. → page 84

[2] S. Beer and R. Ginosar. Eleven ways to boost your synchronizer. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(6): 1040–1049, June 2015. ISSN 1063-8210. doi:10.1109/TVLSI.2014.2331331. → pages 19, 102, 103

[3] S. Beer, J. Cox, T. Chaney, and D. M. Zar. MTBF bounds for multistage synchronizers. In *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*, pages 158–165, May 2013. doi:10.1109/ASYNC.2013.18. → page 15

[4] T. J. Chaney and C. E. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Transactions on Computers*, C-22(4):421–422, April 1973. ISSN 0018-9340. doi:10.1109/T-C.1973.223730. → page 9

[5] T. Coleman and W. Xu. *Automatic Differentiation in MATLAB Using ADMAT with Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2016. doi:10.1137/1.9781611974362. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611974362. → page 11

[6] G. R. Couranz and D. F. Wann. Theoretical and experimental behavior of synchronizers operating in the metastable region. *IEEE Transactions on Computers*, C-24(6):604–616, June 1975. ISSN 0018-9340. doi:10.1109/T-C.1975.224273. → page 15

[7] M. V. Dunga, C.-H. Lin, A. M. Niknejad, and C. Hu. *BSIM-CMG: A Compact Model for Multi-Gate Transistors*, pages 113–153. Springer US, Boston, MA, 2008. ISBN 978-0-387-71752-4. doi:10.1007/978-0-387-71752-4_3. URL https://doi.org/10.1007/978-0-387-71752-4_3. → pages 10, 61

[8] P. Eberhard and C. Bischof. Automatic differentiation of numerical integration algorithms. *Mathematics of Computation*, 68(226):717–731, 1999. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2585052. → page 12

[9] L. M. engelhardt. *version XVII*. Linear Techology Corporation - now part of Analog Devices, 2019. URL https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html. → pages 10, 11, 62, 64

[10] C. Enz. An MOS transistor model for RF IC design valid in all regions of operation. *IEEE Transactions on Microwave Theory and Techniques*, 50(1): 342–359, Jan 2002. ISSN 0018-9480. doi:10.1109/22.981286. → pages 10, 77, 81

[11] C. C. Enz. A short story of the EKV MOS transistor model. *IEEE Solid-State Circuits Society Newsletter*, 13(3):24–30, Summer 2008. ISSN 1098-4232. doi:10.1109/N-SSC.2008.4785778. → pages 10, 77

[12] C. C. Enz and E. A. Vittoz. Charge-based MOS transistor modeling - the EKV model for low-power and RF IC design. 2006. URL http://infoscience.epfl.ch/record/149582. → pages 9, 77

[13] C. C. Enz, F. Krummenacher, and E. A. Vittoz. An analytical MOS transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications. *special issue of the Analog Integrated Circuits and Signal Processing Journal on Low-Voltage and Low-Power Design*, 8:83–114, 1995. URL http://infoscience.epfl.ch/record/149574. → pages 6, 10, 61, 77

[14] C. P. Fries. Stochastic automatic differentiation: automatic differentiation for monte-carlo simulations. *Quantitative Finance*, 19(6):1043–1059, 2019. doi:10.1080/14697688.2018.1556398. URL https://doi.org/10.1080/14697688.2018.1556398. → page 11

[15] P. R. Gray and R. G. Meyer. *Analysis and Design of Analog Integrated Circuits*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 1990. ISBN 0471874930. → pages 9, 127

[16] B. Group. Berkeley short-channel IGFET model (BSIM) group, Feb 2017. URL https://bsim.berkeley.edu/models. → pages 10, 61, 71

[17] D. Hodges, H. G. Jackson, and R. Saleh. Analysis and design of digital integrated circuits : In deep submicron technology. 01 2004. → page 9

[18] HSPICE. *the gold standard for accurate circuit simulations*. Synopsis Inc., Mountain View, California, 2019. URL https://www.synopsys.com/verification/ams-verification/hspice.html. → pages 10, 11

[19] W. R. Inc. *Mathematica, Version 12.0*. Champaign, IL, USA, 2019. URL https://www.wolfram.com/mathematica/. → page 42

[20] N. Integration and M. N. Group. Predictive technology model, June 2011. URL http://ptm.asu.edu/. → pages 10, 64

[21] D. James. Intel Ivy bridge unveiled — the first commercial tri-gate, high-k, metal-gate CPU. In *Proceedings of the IEEE 2012 Custom Integrated Circuits Conference*, pages 1–4, Sep. 2012. doi:10.1109/CICC.2012.6330644. → page 10

[22] G. Kedem. Automatic differentiation of computer programs. *ACM Trans. Math. Softw.*, 6(2):150–165, June 1980. ISSN 0098-3500. doi:10.1145/355887.355890. URL http://doi.acm.org/10.1145/355887.355890. → pages 9, 11

[23] A. Khakifirooz, O. M. Nayfeh, and D. Antoniadis. A simple semiempirical short-channel MOSFET current –voltage model continuous across all regions of operation and employing only physical parameters. *IEEE Transactions on Electron Devices*, 56(8):1674–1680, Aug 2009. ISSN 0018-9383. doi:10.1109/TED.2009.2024022. → pages 10, 61, 72, 125

[24] D. J. Kinniment and D. B. G. Edwards. Circuit technology in a large computer system. *Radio and Electronic Engineer*, 43(7):435–441, July 1973. ISSN 0033-7722. doi:10.1049/ree.1973.0068. → page 9

[25] D. J. Kinniment and J. V. Woods. Synchronisation and arbitration circuits in digital systems. *Proceedings of the Institution of Electrical Engineers*, 123 (10):961–966, October 1976. ISSN 0020-3270. doi:10.1049/piee.1976.0212. → page 15

[26] T. Kobayashi, K. Nogami, T. Shirotori, Y. Fujimoto, and O. Watanabe. A current-mode latch sense amplifier and a static power saving input buffer for low-power architecture. In *1992 Symposium on VLSI Circuits Digest of Technical Papers*, pages 28–29, June 1992. doi:10.1109/VLSIC.1992.229252. → page 128

[27] Y. Li, P. I.-J. Chuang, A. Kennings, and M. Sachdev. Voltage-boosted synchronizers. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 307–312, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3474-7. doi:10.1145/2742060.2742075. URL http://doi.acm.org/10.1145/2742060.2742075. → pages 19, 102, 127, 128

[28] M. S. Lundstrom and D. A. Antoniadis. Compact models and the physics of nanoscale FETs. *IEEE Transactions on Electron Devices*, 61(2):225–233, Feb 2014. ISSN 0018-9383. doi:10.1109/TED.2013.2283253. → pages 10, 72

[29] Maple. *2019*. Maplesoft, Waterloo ON, Canada, 2019. URL https://www.maplesoft.com/. → page 42

[30] L. R. Marino. General theory of metastable operation. *IEEE Transactions on Computers*, C-30(2):107–115, Feb 1981. ISSN 0018-9340. doi:10.1109/TC.1981.6312173. → pages 12, 13

[31] MATLAB. *version 9.3.0.713579 (R2017b)*. The MathWorks Inc., Natick, Massachusetts, 2017. URL https://www.mathworks.com/products/matlab.html. → page 11

[32] MATLAB. *Symbolic Math Toolbox*. Mathworks, Natick, Massachusetts, 2017b. URL https://www.mathworks.com/products/symbolic.html. → page 42

[33] M. Mendler and T. Stroup. Newtonian arbiters cannot be proven correct. *Formal Methods in System Design*, 3(3):233–257, Dec 1993. ISSN 1572-8102. doi:10.1007/BF01384075. URL https://doi.org/10.1007/BF01384075. → pages 3, 12, 13

[34] MetaACE. BlendICS Inc., St. Louis, Missouri, 2016. URL http://blendics.com. → page 15

[35] W. Müller and I. Eisele. Velocity saturation in short channel field effect transistors. *Solid State Communications*, 34(6):447 – 449, 1980. ISSN 0038-1098. doi:https://doi.org/10.1016/0038-1098(80)90648-1. URL http://www.sciencedirect.com/science/article/pii/0038109880906481. → pages 9, 67

[36] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. Donahue, J. Eno, W. Hoeppner, D. Kruckemyer, T. H.

Lee, P. C. M. Lin, L. Madden, D. Murray, M. H. Pearce, S. Santhanam, K. J. Snyder, R. Stehpany, and S. C. Thierauf. A 160-mhz, 32-b, 0.5-w CMOS RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 31(11): 1703–1714, Nov 1996. ISSN 1558-173X. doi:10.1109/JSSC.1996.542315. → page 128

[37] L. W. Nagel and D. Pederson. SPICE (simulation program with integrated circuit emphasis). Technical Report UCB/ERL M382, EECS Department, University of California, Berkeley, Apr 1973. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html. → pages 9, 61

[38] S. Rakheja and D. Antoniadis. MVS nanotransistor model (silicon), Dec 2015. URL https://nanohub.org/publications/15/4. → pages 6, 10, 74, 75

[39] L. B. Rall. *Automatic Differentiation: Techniques and Applications*, volume 120 of *Lecture Notes in Computer Science*. Springer, 1981. ISBN 3-540-10861-0. doi:10.1007/3-540-10861-0. URL https://doi.org/10.1007/3-540-10861-0. → pages 9, 11

[40] J. Reiher, M. R. Greenstreet, and I. W. Jones. Explaining metastability in real synchronizers. In *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 59–67, May 2018. doi:10.1109/ASYNC.2018.00024. → pages 16, 104

[41] J. Revels, M. Lubin, and T. Papamarkou. Forward-mode automatic differentiation in Julia. *CoRR*, abs/1607.07892, 2016. URL http://arxiv.org/abs/1607.07892. → page 11

[42] S. M. Rump. *INTLAB — INTerval LABoratory*, pages 77–104. Springer Netherlands, Dordrecht, 1999. ISBN 978-94-017-1247-7. doi:10.1007/978-94-017-1247-7_7. URL https://doi.org/10.1007/978-94-017-1247-7_7. → pages 6, 11

[43] S. M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010. doi:10.1017/S096249291000005X. → page 11

[44] K. Rupp. 42 years of microprocessor trend data. https://github.com/karlrupp/microprocessor-trend-data, 2018. → pages xi, 2

[45] D. Smith and J. Linvill. An accurate short-channel IGFET model for computer-aided circuit design. In *1971 IEEE International Solid-State*

Circuits Conference. Digest of Technical Papers, volume XIV, pages 40–41, Feb 1971. doi:10.1109/ISSCC.1971.1154896. → page 9

[46] I. E. Sutherland. Micropipelines. *Commun. ACM*, 32(6):720–738, June 1989. ISSN 0001-0782. doi:10.1145/63526.63532. URL http://doi.acm.org/10.1145/63526.63532. → page 74

[47] H. Vogt, M. Hendrix, and P. Nenzi. *ngSPICE - version 30*. 2019. URL http://ngspice.sourceforge.net/. → pages 10, 11, 62, 64

[48] L. Wei, O. Mysore, and D. Antoniadis. Virtual-source-based self-consistent current and charge FET models: From ballistic to drift-diffusion velocity-saturation operation. *IEEE Transactions on Electron Devices*, 59 (5):1263–1271, May 2012. ISSN 0018-9383. doi:10.1109/TED.2012.2186968. → pages 10, 61, 72

[49] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 3rd edition, 2004. ISBN 0321149017, 9780321149015. → pages 9, 65, 68

[50] C. Yan. *Reachability Analysis Based Circuit-Level Formal Verification*. PhD thesis, The University of British Columbia, 2011. → page 11

[51] S. Yang and M. Greenstreet. Computing synchronizer failure probabilities. In *2007 Design, Automation Test in Europe Conference Exhibition*, pages 1–6, April 2007. doi:10.1109/DATE.2007.364487. → pages 6, 14, 16, 20, 94, 124, 125

[52] S. Yang and M. R. Greenstreet. Simulating improbable events. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 154–157, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-627-1. doi:10.1145/1278480.1278518. URL http://doi.acm.org.ezproxy.library.ubc.ca/10.1145/1278480.1278518. → pages 14, 124, 125

[53] S. Yang, I. W. Jones, and M. R. Greenstreet. Synchronizer performance in deep sub-micron technology. In *2011 17th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 33–42, April 2011. doi:10.1109/ASYNC.2011.19. → pages 19, 128

[54] J. Zhou, D. Kinniment, G. Russell, and A. Yakovlev. A robust synchronizer. volume 2006, pages 2 pp.–, 04 2006. ISBN 0-7695-2533-4. doi:10.1109/ISVLSI.2006.12. → pages 19, 102, 127, 128

Appendix A

Supporting Materials

A.1 Derivative of Matrix A^{-1}

Let A be a non-singular matrix, then the derivative of $\frac{d}{dt}A^{-1}$ is shown in Equation A.1:

$$\begin{aligned} I &= AA^{-1} \\ 0 &= \tfrac{d}{dt}\left(AA^{-1}\right) \\ &= \left(\tfrac{d}{dt}A\right)A^{-1} + A\left(\tfrac{d}{dt}A^{-1}\right) \\ -A\left(\tfrac{d}{dt}A^{-1}\right) &= \left(\tfrac{d}{dt}A\right)A^{-1} \\ \tfrac{d}{dt}A^{-1} &= -A^{-1}\left(\tfrac{d}{dt}A\right)A^{-1} \end{aligned} \tag{A.1}$$

A.2 Derivation of $\dot{g}(t)$

Found here are all the details on the derivation to $\dot{g}(t)$. Of particular lenght is the derivative of $\frac{1}{||\vec{u}^T S(t,t_{eola})||_2}$. Each piece is derived individually and pieced back together for the entire result:

$$\begin{aligned} \tfrac{d}{dt}S(t,t_{eola}) &= \tfrac{d}{dt}S(t_{eola},t)^{-1} \\ &= -S(t_{eola},t)\tfrac{d}{dt}S(t_{eola},t)S(t_{eola},t)^{-1} \\ &= -S(t_{eola},t)J_f(t)S(t_{eola},t)S(t_{eola},t)^{-1} \\ &= -S(t_{eola},t)J_f(t) \end{aligned} \tag{A.2}$$

$$\begin{aligned}\frac{d}{dt}\left(\tilde{u}^T S(t,t_{eola})\right) &= \tilde{u}^T \frac{d}{dt}\left(S(t,t_{eola})\right) \\ &= -\tilde{u}^T S(t,t_{eola}) J_f(t)\end{aligned} \tag{A.3}$$

$$\begin{aligned}\frac{d}{dt}\left(\frac{1}{||\tilde{u}^T S(t,t_{eola})||_2}\right) &= \frac{-1}{||\tilde{u}^T S(t,t_{eola})||_2^2}\frac{d}{dt}\left(||\tilde{u}^T S(t,t_{eola})||_2\right) \\ &= \frac{-1}{||\tilde{u}^T S(t,t_{eola})||_2^2}\frac{d}{dt}\left(\sqrt{\tilde{u}^T S(t,t_{eola})\left(\tilde{u}^T S(t,t_{eola})\right)^T}\right) \\ &= \frac{-1}{||\tilde{u}^T S(t,t_{eola})||_2^2}\frac{\frac{d}{dt}\left(\tilde{u}^T S(t,t_{eola})\left(\tilde{u}^T S(t,t_{eola})\right)^T\right)}{2\sqrt{\tilde{u}^T S(t,t_{eola})(\tilde{u}^T S(t,t_{eola}))^T}} \\ &= \frac{-1}{2||\tilde{u}^T S(t,t_{eola})||_2^3}\frac{d}{dt}\left(\tilde{u}^T S(t,t_{eola}) S(t,t_{eola})^T \tilde{u}\right) \\ &= \frac{-1}{2||\tilde{u}^T S(t,t_{eola})||_2^3}\Bigg(\tilde{u}^T \frac{d}{dt} S(t,t_{eola}) S(t,t_{eola})^T \tilde{u} \\ &\quad + \tilde{u}^T S(t,t_{eola}) \frac{d}{dt} S(t,t_{eola})^T \tilde{u}\Bigg) \\ &= \frac{-1}{2||\tilde{u}^T S(t,t_{eola})||_2^3}\Bigg(-\tilde{u}^T S(t,t_{eola}) J_f(t) S(t,t_{eola})^T \tilde{u} \\ &\quad + \tilde{u}^T S(t,t_{eola})\left(\frac{d}{dt} S(t,t_{eola})\right)^T \tilde{u}\Bigg) \\ &= \frac{-1}{2||\tilde{u}^T S(t,t_{eola})||_2^3}\Bigg(-\tilde{u}^T S(t,t_{eola}) J_f(t) S(t,t_{eola})^T \tilde{u} \\ &\quad - \tilde{u}^T S(t,t_{eola})\left(S(t_{eola},t) J_f(t)\right)^T \tilde{u}\Bigg) \\ &= \frac{-1}{2||\tilde{u}^T S(t,t_{eola})||_2^3}\Bigg(-\tilde{u}^T S(t,t_{eola}) J_f(t) S(t,t_{eola})^T \tilde{u} \\ &\quad - \tilde{u}^T S(t,t_{eola}) J_f(t)^T S(t_{eola},t)^T \tilde{u}\Bigg) \\ &= \frac{\tilde{u}^T S(t,t_{eola}) J_f(t) S(t,t_{eola})^T \tilde{u} + \tilde{u}^T S(t,t_{eola}) J_f(t)^T S(t_{eola},t)^T \tilde{u}}{2||\tilde{u}^T S(t,t_{eola})||_2^3}\end{aligned} \tag{A.4}$$

$$\frac{d}{dt}\beta(t) = J_f(t)\beta(t) + \frac{\partial f(t)}{\partial t_{in}} \tag{A.5}$$

$$\begin{aligned}u(t)^T &= \frac{\tilde{u}^T S(t,t_{eola})}{||\tilde{u}^T S(t,t_{eola})||_2} \\ u(t) &= \frac{\left(\tilde{u}^T S(t,t_{eola})\right)^T}{||\tilde{u}^T S(t,t_{eola})||_2} \\ &= \frac{S(t,t_{eola})^T \tilde{u}}{||\tilde{u}^T S(t,t_{eola})||_2}\end{aligned} \tag{A.6}$$

$$g(t) = u(t)^T \beta(t) \tag{A.7}$$

$$
\begin{aligned}
\dot{g}(t) &= \frac{d}{dt}\left(\frac{\tilde{u}^T S(t,t_{eola})}{||\tilde{u}^T S(t,t_{eola})||_2}\beta(t)\right) \\
&= \frac{d}{dt}\left(\tilde{u}^T S(t,t_{eola})\right)\frac{\beta(t)}{||\tilde{u}^T S(t,t_{eola})||_2} + \tilde{u}^T S(t,t_{eola})\frac{d}{dt}\left(\frac{1}{||\tilde{u}^T S(t,t_{eola})||_2}\right)\beta(t) \\
&\quad + \frac{\tilde{u}^T S(t,t_{eola})}{||\tilde{u}^T S(t,t_{eola})||_2}\frac{d}{dt}\left(\beta(t)\right) \\
&= \frac{-\tilde{u}^T S(t,t_{eola}) J_f(t)\beta(t)}{||\tilde{u}^T S(t,t_{eola})||_2} \\
&\quad + \tilde{u}^T S(t,t_{eola})\frac{\tilde{u}^T S(t,t_{eola}) J_f(t) S(t,t_{eola})^T \tilde{u} + \tilde{u}^T S(t,t_{eola}) J_f(t)^T S(t_{eola},t)^T \tilde{u}}{2||\tilde{u}^T S(t,t_{eola})||_2^3}\beta(t) \\
&\quad + \frac{\tilde{u}^T S(t,t_{eola})}{||\tilde{u}^T S(t,t_{eola})||_2}\left(J_f(t)\beta(t) + \frac{\partial f(t)}{\partial t_{in}}\right) \\
&= -u(t)^T J_f(t)\beta(t) + u(t)^T\left(\frac{u(t)^T J_f(t) u(t) + u(t)^T J_f(t)^T u(t)}{2}\right)\beta(t) \\
&\quad + u(t)^T\left(J_f(t)\beta(t) + \frac{\partial f(t)}{\partial t_{in}}\right) \\
&= u(t)^T\left(u(t)^T J_f(t) u(t)\beta(t) + \frac{\partial f(t)}{\partial t_{in}}\right)
\end{aligned}
$$

Note that $u(t)^T J_f(t) u(t)$ is a scalar, then:

$$
\begin{aligned}
&= u(t)^T J_f(t) u(t) u(t)^T \beta(t) + u(t)^T \frac{\partial f(t)}{\partial t_{in}} \\
&= u(t)^T J_f(t) u(t) g(t) + u(t)^T \frac{\partial f(t)}{\partial t_{in}}
\end{aligned}
\tag{A.8}
$$

A.3 Tensor - Vector Product

This section defines the details for computing a tensor vector product. We use the operator $\otimes$ to denote a tensor-vector product. Consider a tensor A to be an $N \times N \times K$ tensor and $\vec{b}$ be a $N \times 1$ vector. Then each sub element A_i is a $N \times N$ matrix then, thus:

$$
\begin{aligned}
A &= \begin{bmatrix} A_1 & A_2 & \cdots & A_k \end{bmatrix} \\
A \otimes \vec{b} &= \begin{bmatrix} A_1 \cdot \vec{b} & A_2 \cdot \vec{b} & \cdots & A_k \cdot \vec{b} \end{bmatrix}
\end{aligned}
\tag{A.9}
$$

The result of $A_i \cdot \vec{b}$ element is a $N \times 1$, therefore overall $A \otimes \vec{b}$ is a $N \times K$ matrix.