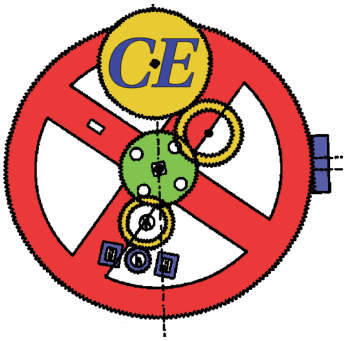


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2016

MSc THESIS

Experimental and Industrial Evaluation of Variability Resilient Schemes

Daniël Kraak

Abstract

As semiconductor devices shrink further, the effects of *process variation* become more and more pronounced. They negatively impact the speed and power consumption of Integrated Circuits (ICs). Traditionally, a worst-case based design methodology is used to compensate for their potential impact. For modern technology this method means an increased penalty in area and power consumption. Therefore, Variability Resilient Schemes (VRSs) are introduced. They are used to improve the speed of ICs with slow processing and decrease the power consumption of ICs with fast processing. Thanks to these techniques a Better-Than-Worst-Case (BTWC) design methodology can be used. It improves performance, area, and power consumption.

The work in this thesis is performed within the Variability Resilient Architectures (VRA) project at NXP Semiconductors. In this project the potential of body biasing, clock stretching, and error avoidance flip-flops as VRSs is investigated. Circuits with VRSs run closer to their optimal performance. This means more extensive testing is necessary compared to worst-case based designs. Also, the usage of error avoidance flip-flops bring new testability challenges. In order for designs with VRSs to be interesting yield and testability need to be high enough. This thesis evaluates the testability challenges of the VRSs used in the VRA project. A cost model is proposed to compare the cost of worst-case based designs with designs with VRSs. The testability of the error avoidance flip-flop is evaluated and a solution is proposed for stuck-at and path delay testing of the error output. Finally, the potential of body biasing to compensate process variation is investigated, by comparing the performance of a worst-case based design and a BTWC design, which has an area reduction of 25%. The performance of both designs is compared by measuring path delays in both simulations and measurements. It is shown that with around 0.2 V forward body biasing the performance of the BTWC design is comparable to the worst-case design.

CE-MS-2016-04

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Experimental and Industrial Evaluation of Variability Resilient Schemes

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Daniël Kraak
born in Dordrecht, The Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Experimental and Industrial Evaluation of Variability Resilient Schemes

by Daniël Kraak

Abstract

As semiconductor devices shrink further, the effects of *process variation* become more and more pronounced. They negatively impact the speed and power consumption of Integrated Circuits (ICs). Traditionally, a worst-case based design methodology is used to compensate for their potential impact. For modern technology this method means an increased penalty in area and power consumption. Therefore, Variability Resilient Schemes (VRSs) are introduced. They are used to improve the speed of ICs with slow processing and decrease the power consumption of ICs with fast processing. Thanks to these techniques a Better-Than-Worst-Case (BTWC) design methodology can be used. It improves performance, area, and power consumption.

The work in this thesis is performed within the Variability Resilient Architectures (VRA) project at NXP Semiconductors. In this project the potential of body biasing, clock stretching, and error avoidance flip-flops as VRSs is investigated. Circuits with VRSs run closer to their optimal performance. This means more extensive testing is necessary compared to worst-case based designs. Also, the usage of error avoidance flip-flops bring new testability challenges. In order for designs with VRSs to be interesting yield and testability need to be high enough. This thesis evaluates the testability challenges of the VRSs used in the VRA project. A cost model is proposed to compare the cost of worst-case based designs with designs with VRSs. The testability of the error avoidance flip-flop is evaluated and a solution is proposed for stuck-at and path delay testing of the error output. Finally, the potential of body biasing to compensate process variation is investigated, by comparing the performance of a worst-case based design and a BTWC design, which has an area reduction of 25%. The performance of both designs is compared by measuring path delays in both simulations and measurements. It is shown that with around 0.2 V forward body biasing the performance of the BTWC design is comparable to the worst-case design.

Laboratory : Computer Engineering
Codenumber : CE-MS-2016-04

Committee Members :

Advisor:	dr. ir. Said Hamdioui, CE, TU Delft
Chairperson:	dr. ir. Said Hamdioui, CE, TU Delft
Member:	dr. ir. Bram Kruseman, NXP Eindhoven
Member:	dr. ir. Andre Bossche, EI, TU Delft
Member:	dr. ir. Mottaqiallah Taouil, CE, TU Delft

Dedicated to my family and friends

Contents

List of Figures

List of Tables

List of Acronyms

ABB Adaptive Body Biasing

ATPG Automatic Test Pattern Generation

BTI Bias Temperature Instability

BTWC Better-Than-Worst-Case

CGU Clock Generation Unit

CPU Central Processing Unit

DCO Digitally Controlled Oscillator

DFT Design For Test

DVS Dynamic Voltage Scaling

GPU Graphics Processing Unit

HCI Hot Carrier Injection

IC Integrated Circuit

JTAG Joint Test Action Group

LDO Low Dropout Regulator

PVT Process, Supply Voltage, and Temperature

PVTA Process, Supply Voltage, Temperature, and Aging

SE Scan Enable

STA Static Timing Analysis

SSTA Statistical Static Timing Analysis

STIL Standard Test Interface Language

V_{th} Threshold Voltage

VRA Variability Resilient Architectures

VRS Variability Resilient Scheme

Acknowledgements

Creating this thesis has been a long journey. I would like to thank a number of people for their help and support.

First of all, I would like to thank my supervisor at NXP, Bram Kruseman. Your support and guidance during the project could not have been any better. You always made time to have meetings and gave excellent feedback on my work. I really learned a lot.

Secondly, I would like to thank my supervisor at TU Delft, Said Hamdioui. Thanks to your feedback, I was able to give the work a more academic background. The meetings were always very useful and gave several opportunities to expand and improve the work.

I would also like to thank Mottaqiallah Taouil for attending several meetings and providing feedback on my work. Your immediate understanding of the project and also the technical work helped a lot.

Finally, I would like to thank Boris Ljevar for helping me a lot with the Cadence tools and Zavarin Gagov for dedicating a lot of his time to create the measurement setup.

Daniël Kraak
Delft, The Netherlands
January 22, 2016

1 Introduction

This section serves as an introduction of this thesis. Section 1.1 discusses the impact of process variation. Section 1.2 addresses the state-of-the-art solutions. Section 1.3 describes the current shortcomings and the contributions this thesis makes. Finally, Section 1.4 presents the outline of the thesis.

1.1 Impact of Process Variation

With the ongoing shrinking of semiconductor devices, the performance of Integrated Circuits (ICs) is improved. However, as the gates shrink further and further, ICs are more and more hampered by process variations, causing electrical parameters to vary. Process variations can negatively impact the the speed and power consumption of ICs. This is illustrated in Figure 1.1, which shows the distribution of frequency and standby leakage of ICs in a wafer for a 180 nm process. It can be seen there is a spread of around 30% in IC frequency and a spread of roughly 20x in standby leakage. This variation will only increase as technology shrinks further.

Process variations are usually categorized into global and local variations[6]. With global variations device parameters, such as the oxide thickness and dopant concentrations, change in an equal manner for all transistors. Process variations that are induced wafer-to-wafer, lot-to-lot, fab-to-fab, and within-wafer fall into this category. Local variations are caused by mismatch. Within-die variations fall under local variations. These process variations may negatively impact yield, as some produced ICs may not perform within specifications.

Traditionally, designers add margins to the design to make sure slow ICs will still run at the required speed. This design method is referred to as worst-case based design. With the increasing impact of process variations in modern technology, the worst-case based design method means a higher amount of area is wasted on these margins. This added area also negatively impacts power consumption. Therefore, with modern technology it is no longer sufficient to optimize on speed, as it can introduce a severe penalty in terms of area usage and power consumption.

These problems with worst-case based design for modern technology have led to the development of new design techniques, which guarantee high yield without compromising on area and power consumption. These techniques provide resilience against process variation, so a Better-Than-Worst-Case (BTWC) design approach can be used. The state-of-the-art techniques are discussed in the next section.

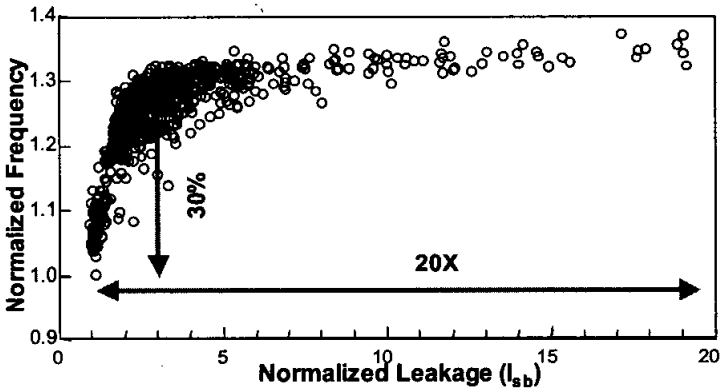


Figure 1.1: Impact of process variations on performance and power for ICs from the same wafer for a 180 nm process (from [1])

1.2 State-of-the-Art

Figure 1.2 shows a classification diagram of the most common Variability Resilient Schemes (VRSs). It can be seen that VRSs are possible in a static and a dynamic fashion.

Static schemes can be applied at transistor level, circuit level, and even by tuning the supply voltage. An example of a technique at transistor level is Variation Aware Gate Sizing, where gates are sized against process variations to optimize for high yield and a good balance between power and performance[7, 8]. Statistical Static Timing Analysis (SSTA) is a popular circuit level technique. SSTA is an improved version of Static Timing Analysis (STA). With STA a circuit's timing is analyzed based on worst-case propagation times of logic gates. With SSTA the normal deterministic timing of gates is replaced with probability distributions. This way a more realistic estimate can be made of the speed of paths in an IC. Thanks to this technique an acceptable yield can be achieved without adding excessive margins[9]. Using body biasing it is possible to let a circuit run faster. After production the silicon speed of the IC is measured. When the produced silicon is slow, internal voltage regulators are programmed *once* to apply forward body biasing, so the circuit is able to run faster[10, 11].

With the dynamic schemes the circuit's behavior is constantly being monitored during runtime and when necessary action is taken. These schemes can also be called *sense and adapt* strategies[12]. Voltage tuning techniques can also be applied in a dynamic fashion. In this case, based on, for example, temperature or aging of the IC, body biasing[13] can be applied or dynamic voltage scaling[14] to make the IC faster or more power efficient. Techniques on the architectural level, such as clock stretching[15] and time borrowing[16] are meant to give the circuit extra time in order to avoid timing errors. With clock stretching the frequency of the clock is dynamically changed. With error avoidance flip-flops, long paths can borrow time from the succeeding clock cycle. This way timing errors are avoided. Finally, process variation resilience can also be addressed at the software level. An example of a software level technique is redundant execution. With redundant execution[17] critical portions of the program are run redundantly on

multiple cores. The outputs are then compared to see if any errors are introduced. Another example is Re-execution and Recovery. Re-execution and Recovery focuses on reexecuting portions of the application that have been detected as being corrupted[18].

The work in this thesis is performed within the Variability Resilient Architectures (VRA) project[1]. The VRA project integrates and evaluates the *static* variant of body biasing, clock stretching, and error avoidance flip-flops (highlighted in Figure 1.2). Therefore, the work in this thesis will focus on these techniques only.

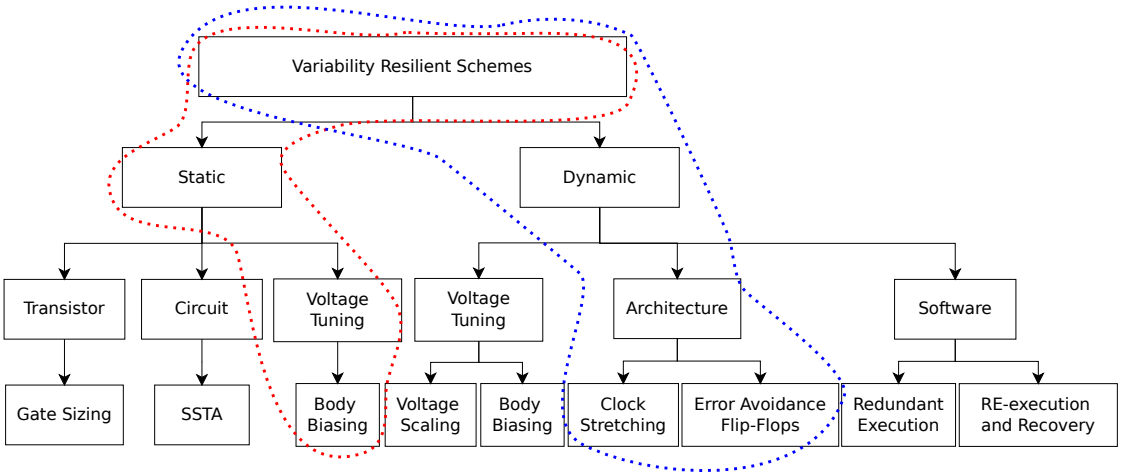


Figure 1.2: Classification diagram of Variability Resilient Schemes

1.3 Main Contributions

Circuits with VRSs run closer to their optimal performance and have less headroom. This means more extensive testing is necessary compared to conventional worst-case based designs. Also, the usage of elements that allow for time borrowing, such as error avoidance flip-flops bring new testability challenges. This thesis evaluates the three VRSs (body biasing, clock stretching, and error avoidance flip-flops) from the VRA project. Providing high quality ICs at high yield is needed for production and sales. Furthermore, increased test costs need to stay low enough. Otherwise, the money saved on area or earned on improved performance will be wasted on testing again.

This thesis makes the following contributions:

- Gives an overview of the three VRSs used in the VRA project.
- Creation of a cost model to compare cost of designs using VRSs with conventional design cost. Using this model the allowed increase of test cost can be estimated.
- Evaluation of testability of the error avoidance flip-flop. New testing methods are proposed.
- Evaluation of body biasing for a BTWC design using path delay testing.

[1]The VRA project is a research project at NXP semiconductors[19].

1.4 Thesis outline

The thesis outline is as follows:

- Chapter 2 gives an overview of the VRSs used in the VRA project. The chapter ends with a comparison of the schemes.
- Chapter 3 discusses the evaluation methodology for the three VRSs.
- Chapter 4 describes the performed experiments and the results.
- Chapter 5 concludes the thesis and gives recommendations for future work.

2 Variability Resilient Schemes

Modern CMOS technologies suffer from increased process variation. This process variation impacts functionality, performance, reliability, and yield of devices. Therefore, Variability Resilient Schemes (VRSs) are proposed. They aim at the creation of design methods and solutions with the objective to minimize or prevent the impact of process variations and, therefore, achieve higher cost-effectiveness and better reliability. In this chapter, the variability resilience concept and the used schemes are discussed and compared. This chapter serves as background information. Section 2.1 discusses the types and effects of process variation. Section 2.2 provides a motivation for the need of VRSs. After this, the main VRSs are addressed: Body biasing in Section 2.3, the error avoidance flip-flop in Section 2.4, clock stretching in Section 2.5. Then, Section 2.6 discusses the advantages and disadvantages of these techniques. Finally, Section 2.7 concludes the chapter.

2.1 Process Variation

Process variation refers to the occuring difference in device parameters of an Integrated Circuit (IC) due to fabrication, environmental, and temporal conditions. These variations affect the switching speed of the transistors in the circuit and thus the overall speed of the circuit. Therefore, these variations need to be taken into account during design. Otherwise, it cannot be guaranteed that the design will be able to run at the required speed. Process variation sources include Process, Supply Voltage, Temperature, and Aging (PVTA)[6].

Variations in the process are caused by imperfections in the manufacturing process (fabrication). This is, for instance, caused by imperfect lithographic process control and errors in alignment. This causes fluctuations in the width W and the length L of transistors, hence affecting the drive current of transistors. During the doping process atoms are randomly placed into the channels of transistors. This results in statistical variations of dopant atoms in the channels of transistors. This leads to variations of the Threshold Voltage (V_{th}), which directly affects the switching speed and drive strength of the transistor.

Environmental variations are caused by fluctuations in supply voltage and temperature. Supply voltage fluctuations are mainly caused by IR drop and di/dt noise. IR drop is caused by the voltage drop that occurs due to current going through the parasitic resistance of the power grid. Di/dt noise is caused by the parastic inductance of the power grid. The fluctuations in power supply voltage affect the delay of logic gates. The lower the supply voltage the higher the delay will be. The temperature in the IC varies due to the environment temperature. The temperature also varies on the IC, due to power dissipation in the transistors and wires. Regions in the IC with a lot

of switching activity can create a high local temperature called a hot-spot. An increase in temperature makes a circuit typically slow down, due to reduced mobility of carriers and increased interconnect resistance.

Temporal variations are caused by aging effects. The aging effects cause the transistors to wear-out. Wear-out effects, such as Hot Carrier Injection (HCI) and Bias Temperature Instability (BTI) degrade the drive current of transistors during use. During design these wear-out effects need to be taken into account. Enough margin needs to be added to the design so it will continue to work throughout its lifetime under these effects.

Besides categorizing variations into fabrication, environmental, and temporal variations, process variations are also often categorized into *local* and *global* variations[20]. With global variations, device parameters change in an equal manner for all devices on the IC. Process variations that are induced fab-to-fab, wafer-to-wafer, lot-to-lot, and within-wafer fall into this category (illustrated in Figure 2.1). Local variations refer to variations within a single IC. This variation is called within-die variation (illustrated in Figure 2.2).

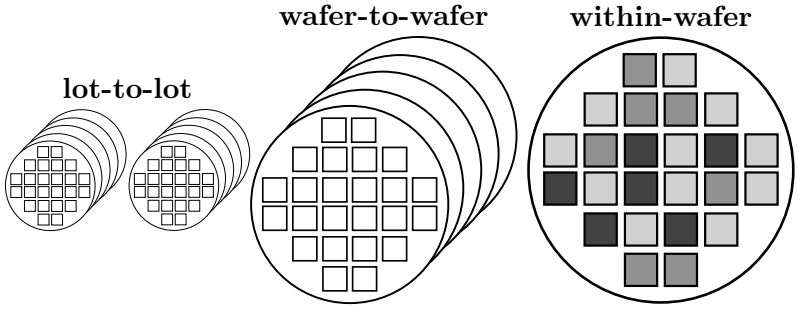


Figure 2.1: Global variations can be recognized lot-to-lot, wafer-to-wafer, and within-wafer.

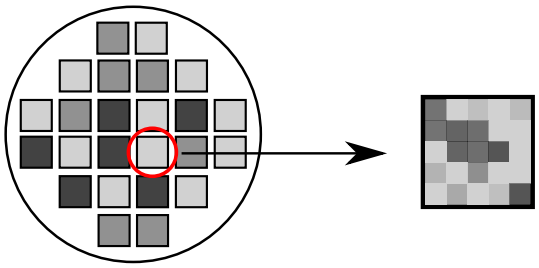


Figure 2.2: Local variation is the variation within a die, also called within-die variation

Due to the process variations some fabricated ICs will be slower and some will be faster. The speed of fabricated ICs is often modeled with a gaussian probability density function. Figure 2.3 shows an example probability density of the speed of an IC. The slow (ss) corner and the fast (ff) corner are indicated by arrows. The slow corner and the fast corner represent the most extreme cases of divergence from the nominal characteristics.

The norminal characteristics are often called the typical (tt) corner. For the slow corner the process variations are combined in such a way that the IC is the slowest possible. For the fast corner the process variations are combined in such a way so the IC is the fastest possible. The fast corner suffers from the highest leakage, so fast ICs have the highest power consumption.

At the end of the design phase of an IC, *corner lots* are usually produced. Corner lots are wafers which have been purposely skewed by a fab to a corner, such as ss or ff corner[21]. Using the corner lots it can then be verified if the design is able to run at the required specifications (speed, power consumption) for all extreme cases of process variation.

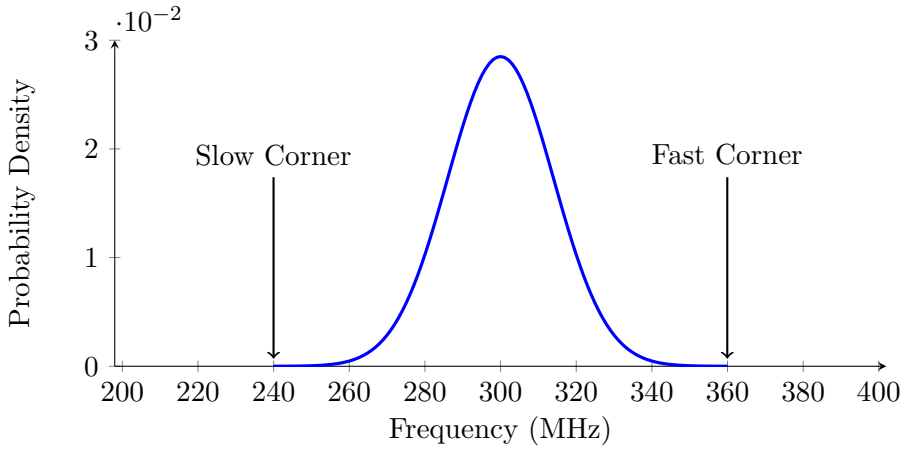


Figure 2.3: Probability density function of the speed of an IC after fabrication.

2.2 Motivation for Variability Resilient Schemes

Figure 2.4 shows example distributions of silicon speed for old and modern technologies after manufacturing. As can be seen the distribution for modern technology is wider than the distribution for old technology. This happens because the variability is higher for modern technology. On average the modern technology is faster. Slow samples, however, have a performance similar to old technology.

For some designs, such as high performance Central Processing Units (CPUs) and Graphics Processing Units (GPUs), speed binning[22] is performed. This means the same design is sold at several maximum frequencies. Chips are tested extensively after manufacturing to determine their maximum frequency. The tests to determine the maximum frequency are very expensive[23]. However, chips with a bit more performance can be sold for more money. Therefore, it pays to do this extensive testing to determine which chips can be sold for a higher maximum frequency.

Some designs do not benefit from a higher maximum frequency. For instance, an audio chip just needs to run at a minimum target speed and does not benefit from running faster. For these chips speed binning is not an option. Therefore, in order to

maximize yield, enough margin is built into the design to make sure also chips from the slow corner still meet the target frequency.

Adding enough margin to the design so even the chips from the slow corner are able to run at the target speed, is referred to as a worst-case based design. Also, the faster the silicon is, the higher the leakage, thus the higher the power consumption. Therefore, the worst-case parameters determine the specifications of the product. The slow corner determines the speed and the fast corner the power consumption. Due to the higher variability of modern technology, more margin needs to be built into the ICs to make sure chips from the slow corner will still meet the target frequency. When more margin is added to the chip it means the power consumption increases as well. Therefore, the slow corner and the fast corner are both responsible for competing specifications. With worst-case based design it is hard to optimize for both speed and power consumption at the same time.

VRSs minimize or prevent the effects of process variation, making the speed distribution of the manufactured chips more narrow again. VRSs enable slow silicon to run at higher frequencies. This way a Better-Than-Worst-Case (BTWC) design method can be used, as less margin needs to be added to make sure slow chips meet timing specifications. Furthermore, techniques are used to tune fast silicon to become less leaky, so the power consumption is reduced. With this area and power reduction a more cost effective use of modern technology is achieved.

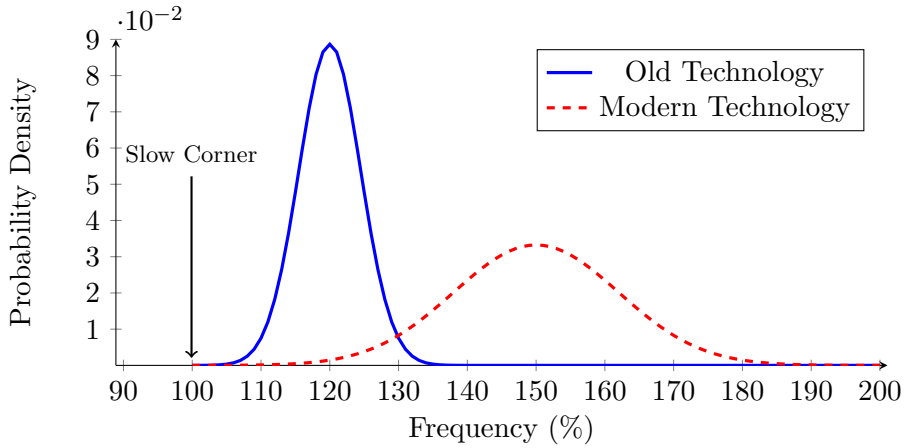


Figure 2.4: Old versus modern technology speed distributions. The frequency denotes the normalized clock speed of the circuit.

2.3 Body Biasing

Body biasing is a technique that changes the transistor's threshold voltage. Body biasing is based on the transistor's *body effect*. The body effect models the threshold voltage change based on the voltage difference between the source and body of the transistor (V_{SB}). Normally, for an NMOS transistor the body is connected to ground, while for a PMOS transistor it is connected to V_{DD}. With body biasing the transistor bodies

are connected to separate power networks instead of V_{DD} and ground. These power networks can either be supplied from off-chip or on-chip sources.

With forward body biasing the threshold voltage is lowered, making the transistor faster and leakier. Forward body biasing is done by applying a lower V_{SB} for an NMOS transistor. For a PMOS transistor the V_{SB} needs to be increased. Reverse body biasing raises the threshold voltage. This makes the transistor slower and less leaky. Reverse body biasing is done by applying a higher V_{SB} for an NMOS transistor and a lower V_{SB} for a PMOS transistor.

Body Biasing can be used to compensate the effects of global process variation. This is done by adding two extra power networks to the design that are used to control the body bias voltages of the NMOS and PMOS transistors. Furthermore, two on-chip sources are added that generate the body bias voltages. After the silicon is produced the characteristics of the silicon can be determined by measuring dedicated monitoring circuits, such as ring oscillators[24]. Based on the characteristics, for instance, fuses are set on the dies. These fuses control the on-chip sources for the body bias network. Chips with slow silicon can be made faster using forward body biasing. Chips with fast silicon can be made less leaky (and slower) using reverse body biasing.

In Figure 2.5 an example is shown of the probability distributions of silicon speed before and after body biasing. As can be seen the distribution with body biasing is a lot more narrow meaning effects of process variation are compensated.

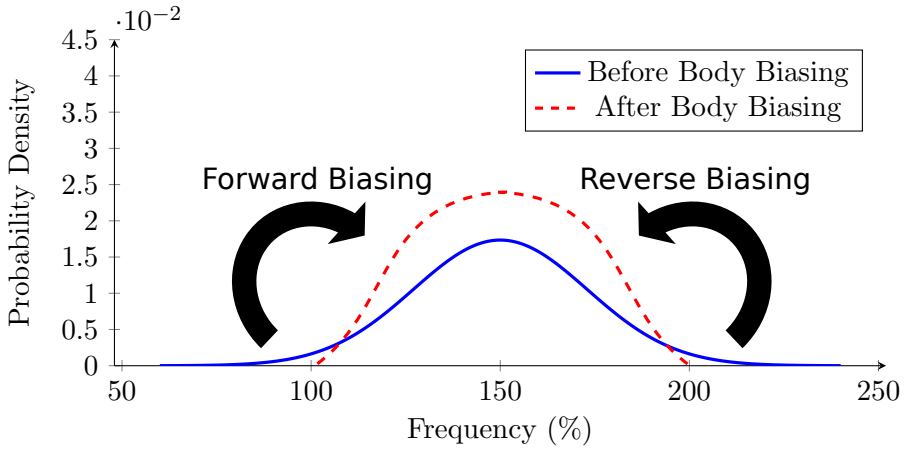


Figure 2.5: Silicon speed distributions before and after body biasing

2.4 Error Avoidance Flip-Flop

Error avoidance flip-flops can be used to counter both global and local process variation. The error avoidance flip-flop is a circuit that provides resilience against timing errors. Timing errors that arise when the propagation delay of a path exceeds the clock period are compensated by borrowing time from the successive clock cycle. The motivation for this arises from the study that only a small portion of the paths in a chip are critical paths[25]. Furthermore, the chance that two critical paths share the same flip-flop as

begin- and endpoint is small. This means most critical paths can borrow time from successive paths in the pipeline, because the successive path will still be able to finish within the reduced timing margin.

In Figure 2.6 the symbol of the error avoidance flip-flop is shown. Just like a normal D flip-flop it has a data input (D), a clock input (CK) and an output Q. The new signals are a delayed clock input (DCK) and an ERROR (ERR) output.

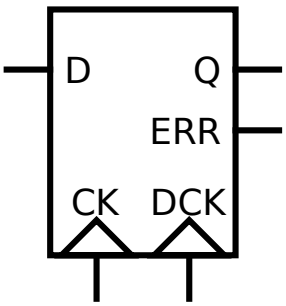


Figure 2.6: Error Avoidance Flip-Flop symbol

Figure 2.7 shows a timing diagram of the error avoidance flip-flop, illustrating how timing errors are compensated. Basically, an error avoidance flip-flop uses a delayed clock to resample the data input. By doing this, it checks for timing errors and corrects the output. At timestep *t=t1*, the D input changes within the normal clock period (CK). The D input is sampled and the sampled value is placed onto output Q. At timestep *t=t2*, a timing violation occurs: the new value for the D input is resolved too late and happens after the rising edge of the clock. The delayed clock is used to check for late transitions at the D input. The timing violation at timestep *t=t2* is corrected at timestep *t=t2'*: the data input is sampled at the rising edge of DCK and the Q output is updated with the correct value. Furthermore, the ERROR signal is raised, indicating a timing violation has occured.

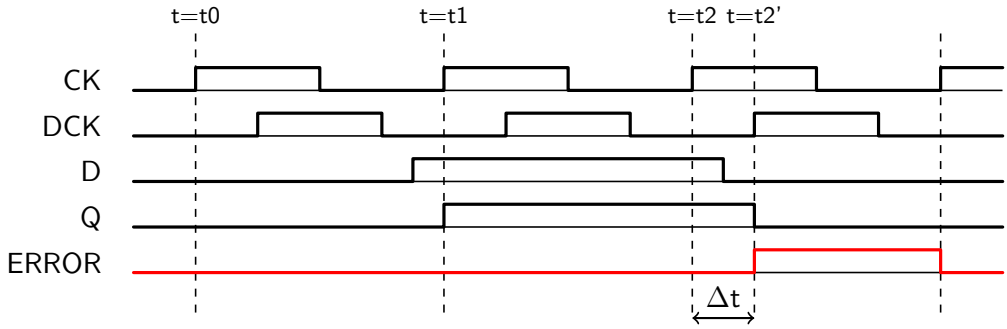


Figure 2.7: Error Avoidance Flip-Flop waveform

The amount of timing violation that can be recovered equals the skew between the normal clock and the delayed clock, shown as Δt in Figure 2.7. The sampling with the delayed clock also puts a new constraint on the hold time at the D input. The hold time now needs to be increased with Δt. This means short paths in front of the data input

need to have sufficient delay, so the data input is not updated too early. If path delays are too short, there is a chance that the D input is updated before it is resampled with the delayed clock. In this case, the Q output is updated with the wrong value and the ERROR signal is raised unjustly.

The ERROR signal of the error avoidance flip-flop can be used to inform a control unit that a timing error has occured. Based on the incoming ERROR signals, the control unit is able to detect when error avoidance is no longer possible. In this case the control unit can inform a higher system layer. For example, the software layer can be informed. The software layer is then responsible for performing a rollback and redoing the calculation at a lower frequency for instance.

For the Variability Resilient Architectures (VRA) project the TIMBER flip-flop[2] is used. In Figure 2.8 the layout of the TIMBER flip-flop is shown. The TIMBER flip-flop consists of two master latches, M0 and M1, and a slave latch. Master latch M0 samples the D input at the rising edge of the normal clock and drives the slave latch with this value. Master latch M1 does this at the rising edge of the delayed clock.

Transmission gates control whether the value in latch M0 or M1 is passed to the slave latch. When P0 is low, latch M0 passes its captured value to the slave latch. For M1 this is the case, when P1 is low. The truth tables for the P0 and P1 are shown in Tables 2.1a and b. As can be seen latch M0 passes its value to the slave latch, when CK is high and DCK is low. M1 passes its value to the slave latch when both CK and DCK are high. This puts the constraint on the CK and DCK signals that the periods that they are high need to overlap.

The ERROR output is obtained by comparing the values saved in latches M0 and M1. If they are different, then a timing violation has occured and the ERROR signal needs to be raised. The ERROR signal is created by feeding the values in latches M0 and M1 to an XOR gate. The truth table for the ERROR signal is shown in Table 2.1c.

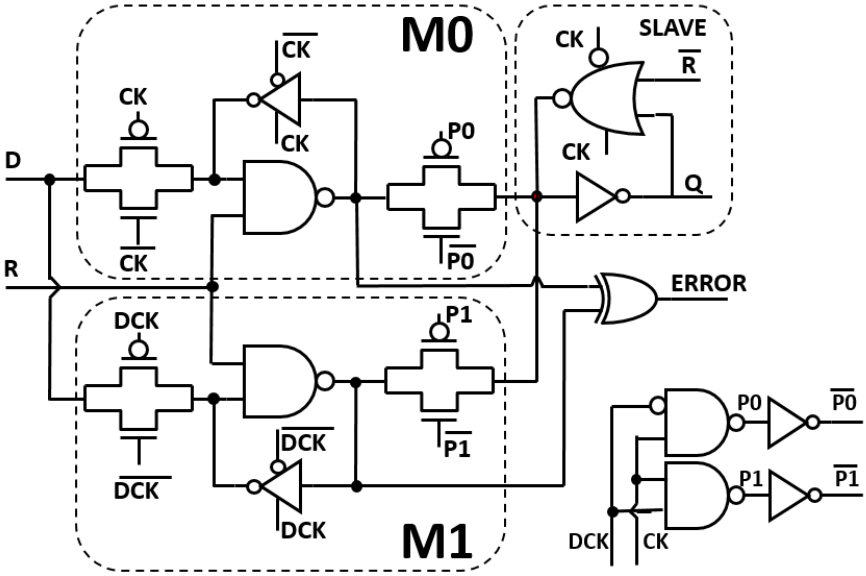


Figure 2.8: TIMBER Flip-Flop Circuit Design from([2])

CK	DCK	P0
0	0	1
0	1	1
1	0	0
1	1	1

(a)

CK	DCK	P1
0	0	1
0	1	1
1	0	1
1	1	0

(b)

M0	M1	ERROR
0	0	0
0	1	1
1	0	1
1	1	0

(c)

Table 2.1: Thruth tables for (a) P0, (b) P1, and (c) ERROR signals

2.5 Clock Stretching

Clock stretching is a technique that can be used to give critical paths in a design extra time to propagate. This is done by dynamically stretching the clock period, when a critical path is activated. In [26] clock stretching for arithmetic units, such as adders and multipliers is discussed. This is done by predicting when critical paths are activated. When this is the case the clock is stretched, giving the critical path more time to finish propagating. This approach is motivated by the fact that critical paths are rarely activated[10]. In conventional designs they do, however, determine the maximum frequency, hence maximum performance.

For the prediction of critical path activation a *decoder* is implemented. This decoder decides whether the clock should be stretched based on the input operands of the arithmetic unit. In [26] the clock stretching is mainly used to decrease power consumption, by using supply voltage scaling and reverse body biasing. This is possible, because the off-critical paths have plenty of timing margin and the critical paths have plenty of timing margin as well due to the clock stretching.

Instead of decreasing power consumption, it is also possible to save on area and/or increase performance. The VRA project investigates the potential of clock stretching to save area. Certain units in an IC can be built with less margin, meaning these units will not meet timing closure under all Process, Supply Voltage, and Temperature (PVT) conditions. A unit like this is referred to as a *relaxed unit*, indicating that the timing constraints have been relaxed. It must be noted here that the focus is not on performing clock stretching when a critical path is used, but on when a certain unit is used. This is mostly done, because it is easier to detect/predict if a certain unit is used.

In microprocessors arithmetic units such as the multiplier usually have high area overhead. These units are used relatively little. The idea is to build these units with less margin in order to save area. Using clock stretching the timing closure can still be met for all PVT conditions. This obviously occurs at a slight performance penalty, since the clock period is now longer. When only applying clock stretching when certain units in the IC are used, clock stretching can be used to provide resilience against global variation.

In Figure 2.9 an example design with a clock stretcher is shown. The design consists of a CPU and a Clock Generation Unit (CGU) that is responsible for providing the clock signal to the CPU. The CGU has an input signal *Stretch*. When this signal is

high, the CGU will stretch the clock period, giving the signals in the CPU more time to propagate. The CPU contains relaxed units such as the multiplier and divider. In the CPU a decoder is added that generates the Stretch signal. This decoder has as input the instruction that will be performed in the next cycle. If this instruction is a critical instruction that uses relaxed units, the Stretch signal is made high, indicating clock stretching is needed.

Figure 2.10 shows a waveform that shows how clock stretching works. In the waveform the CLK and stretch signal are shown. Furthermore, the *computation* signal indicates how long the execution actually takes (when all paths in the CPU have finished propagating). During cycle 1 a non-critical instruction is run, hence the stretch signal stays low. During cycle 2 a critical instruction is run. Therefore, the stretch signal is made high and the clock is stretched. In this case the clock is stretched to two times the clock period of the normal clock. During cycle 3, a non-critical instruction is run again, so the stretch signal is low again.

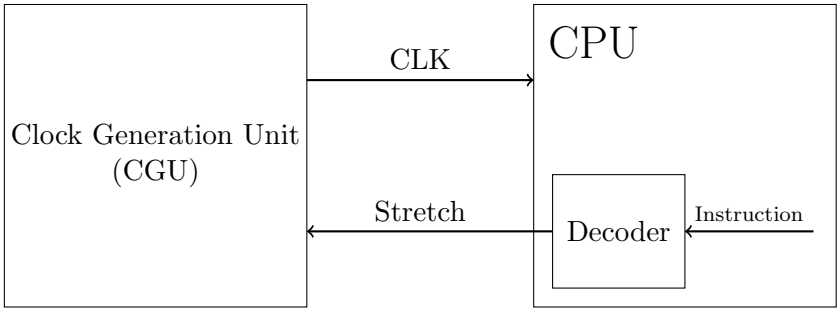


Figure 2.9: CPU design with clock stretcher

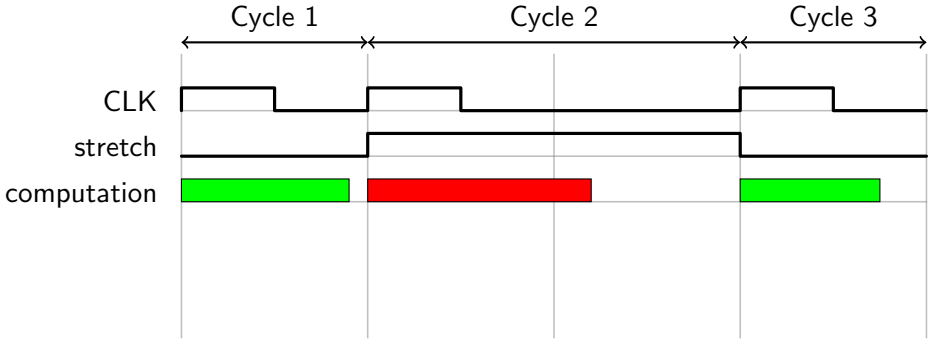


Figure 2.10: Clock stretcher operation

2.6 Advantages and Disadvantages

This section discusses the advantages and disadvantages of each scheme and compares them. A look is taken at area overhead, power consumption, and throughput.

2.6.1 Body Biasing

In terms of area overhead for body biasing two extra power networks and also two body bias generators are added to the design. The two power networks connect to the body connections of the NMOS and PMOS transistors. There will hardly be any current going through these networks, however[13]. Therefore, the metal traces for these networks can be designed at minimum width. Furthermore, during design, the p-wells and n-wells of cells are connected to each other as much as possible. Figure 2.11 shows an example of the connection of the body bias power networks to the n- and p-wells. Using so-called *well-taps* the body bias power networks are connected to the n- and p-wells. Thus, it is not the case that every cell has a separate connection for the body. Therefore, there are relatively few places that the body bias network needs to be connected to. This will make the routing of the body bias power network a lot easier. Considering this and the fact that the width of the metal for the body bias network can be made minimum width, the area overhead of the bias networks will be very minimum.

Also two body bias generators are needed to generate the body bias voltages. Usually, these body bias voltages are created with Low Dropout Regulators (LDOs). Again considering the low power requirements of the body bias network, these LDOs can be designed with very low area. In [27] a similar body biasing scheme to that of the VRA project is discussed. It uses two body bias generators as well, so body biasing can be applied for all NMOS and PMOS transistors in the IC. The area overhead of these generators and the additional routing of the body bias networks is estimated to be around 2%. Therefore, the area overhead of adding body biasing is small.

As said previously, there will hardly be any current going through the body bias networks. Therefore, the power consumption of body biasing will be very minimal.

When looking at the throughput of the design, it can be concluded that body biasing does not affect it. This is because slow silicon is tuned to become faster, so critical parts in the chips do not need extra time as with clock stretching for instance.

2.6.2 Error Avoidance Flip-Flop

In terms of area overhead of the error avoidance flip-flop, there is the area overhead of the error avoidance flip-flop itself, the overhead of handling the ERROR signal, and the overhead of the delayed clock network. In the VRA project the TIMBER flip-flop is used. Compared to a normal flip-flop it uses an extra latch (M1), there is added logic for creating the P0 and P1 signals and and XOR gate is added for the ERROR signal. With this added logic, the TIMBER flip-flop is about twice as big as a normal flip-flop. Not all flip-flops need to be replaced with error avoidance flip-flops. Only the flip-flops that are the endpoint of a critical path. According to [2] about 50% of the flip-flops are the endpoint of a critical path. Replacing all these flip-flops will create a big area overhead. Because of this, it seems more feasible to only design parts of the system with less margin and add error avoidance flip-flops there to provide resilience to process variation. Furthermore, logic needs to be added to handle the ERROR signals. The ERROR signals are most likely captured at a central place in the design, so possibly the signal needs to travel a big distance. This means the logic needs to optimized for speed and will, therefore, have an increased area.

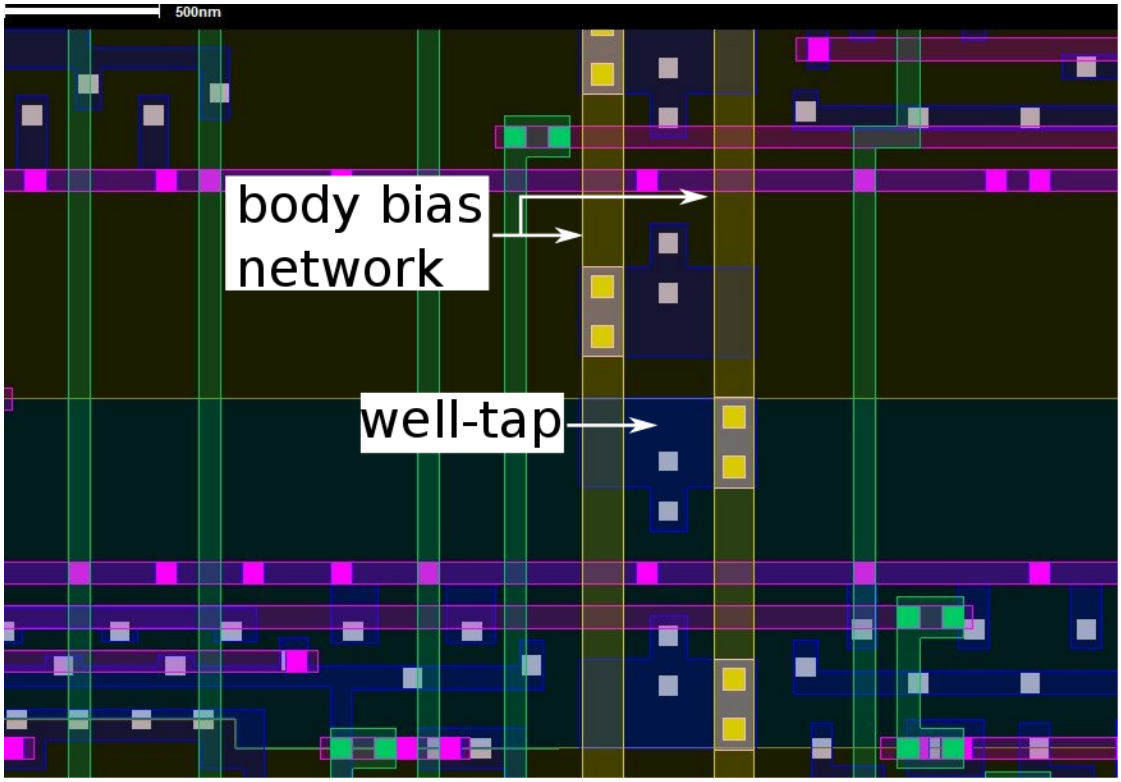


Figure 2.11: GDSII image showing the body bias power networks that connect using well-taps to the n- and p-wells

Also, a delayed clock is needed for the error avoidance flip-flop. This clock needs a certain timing relation to the normal clock. Usually it is already hard to control clock skew for one clock network. The problem of controlling the clock skew can be solved by delaying the normal clock using buffers. If every error avoidance flip-flop uses a buffer to generate the delayed clock from the normal clock, the delayed clock will have a well-defined clock skew to the normal clock. The problem of controlling the clock skew between the normal and delayed clock is solved with this. Obviously, this does cost an extra buffer per error avoidance flip-flop, which will cost a lot of area. To save some area it is possible to only use one buffer for error avoidance flip-flops that are close to each other. With this buffer a common delayed clock is then created for this group of nearby error avoidance flip-flops. Due to the area overhead of the error avoidance flip-flop itself, the overhead of handling the ERROR signal and the overhead of the delayed clock network, it can be expected that the total area overhead of the error avoidance flip-flop is rather high.

The power consumption that error avoidance flip-flops add is most likely high as well due to the high area overhead. Furthermore, the extra clock network for the delayed clock adds more switching activity increasing power consumption even more.

An advantage of error avoidance flip-flops is that one of the main goals is to only affect throughput when necessary. This is the case, because the timing error is compensated

by borrowing time from the succeeding clock stage. Only when there are too many errors and error avoidance is no longer possible (when this happens depends on the implementation), the throughput will be affected. This is because in this case the system needs to redo the calculation at, for instance, a lower frequency.

2.6.3 Clock Stretching

For clock stretching the area overhead lies in a CGU with the option to stretch the output clock and a decoder that determines when the clock needs to be stretched. The area overhead of the CGU and the decoders depends on how advanced the implementation of clock stretching is. For the CGU it depends on which precision of clock stretching is desired. Stretching the clock a whole cycle is easy, as it means the clock output of the CGU should be blocked for one cycle. If the clock needs to be stretched for, for instance, one tenth of the clock period, a clock with a higher speed than the output clock is needed to generate the output clock.

The area overhead of the decoder also depends on how advanced the implementation of clock stretching is. If the clock stretching is only based on which instruction is performed it can be implemented with a relatively low area overhead. If it is based on input vectors to arithmetic units such as the adder or multiplier a higher area overhead can be expected, as the input operands contain more bits.

Just like with the area overhead of clock stretching, the power consumption also depends on the complexity of the implementation. If clock stretching is based on the instruction that is performed the decoder will have low area overhead, thus also a low power consumption. If the clock stretching scheme is more complicated, for instance, based on the input vectors to arithmetic units the power consumption will increase.

A disadvantage of clock stretching is that it affects throughput. This happends because cycles in which the clock is stretched take longer. The penalty in performance depends on the added delay to the cycle, when clock stretching is applied.

2.6.4 Comparison

Table 2.2 compares the area overhead, power consumption, and throughput of each scheme. The more plusses for area, the higher the area overhead of the scheme. It can be seen that body biasing has the lowest area overhead. The error avoidance flip-flop is expected to have the highest area overhead of the three schemes.

In terms of power consumption body biasing performs the best. This is due to the fact that very small currents go through the body bias networks. The error avoidance flip-flop is expected to add the most power consumption due to the fact that it has a high area overhead and extra switching activity due to the extra clock network.

When looking at throughput, body biasing performs the best, as it does not affect throughput. Clock stretching decreases the throughput more than the error avoidance flip-flop. This is the case, because with clock stretching the clock is always stretched when a critical path is activated. The error avoidance flip-flop does not affect throughput, when the amount of errors is low enough, because with the error avoidance flip-flop time is borrowed from the succeeding clock stage. Only when there are too many errors and

error avoidance is no longer possible, throughput is affected, because, for instance, the calculation needs to be redone.

Table 2.2: Comparison between different Variability Resilient Schemes

	Body Biasing	**Error avoidance flip-flop**	**Clock Stretching**
Area	+	+++	++
Power	+	+++	++
Throughput	+	-	- -

2.7 Conclusions

This chapter presents background on the VRSs. It gives an overview of the different sources and the impact of process variation. Process variation sources include process, supply voltage, temperature, and aging. Due to process variation some ICs are slower and some are faster. During design, process variation needs to be taken into account to ensure a high yield.

Subsequently, a motivation for VRS is given. As modern technology is more affected by process variation, more area is lost on design margins. VRSs minimize the effects of process variation, so area and power consumption can be reduced and performance increased.

The discussed VRSs consist of body biasing, error avoidance flip-flops, and clock stretching. With forward body biasing slow ICs are made faster and with reverse body biasing fast ICs less leaky, thus reducing power consumption. The error avoidance flip-flops are flip-flops that correct timing errors by borrowing time from a successive clock cycle. With clock stretching critical paths are given extra time by dynamically stretching the clock.

Finally, a comparison is made of the VRSs. A look is taken at area overhead, throughput, and power consumption.

Evaluation Methodology for Variability Resilient Schemes

3

This chapter presents the evaluation methodology for Variability Resilient Schemes (VRSs). This methodology is focused at evaluating VRS for production and sales. Designs that use VRSs have less headroom, so better testing is needed to determine, whether ICs meet specifications. First of all, a cost model is created for VRSs. With this model the cost of design with VRSs can be compared to that of the conventional design method. Section 3.1 discusses the cost model. After this, Section 3.2 evaluates the testability challenges of the VRSs. Finally, Section 3.3 concludes the chapter.

3.1 Cost Model for Variability Resilient Schemes

With VRSs money can be saved, because Integrated Circuits (ICs) use less area. ICs that use VRSs have less headroom, however, which is discussed in Section 3.2. This makes more extensive testing during production necessary. If the added costs of testing make design with VRSs more expensive than conventional designs, then it is no longer interesting. Depending on how much area is saved versus how much test cost is added, design with VRSs is either cheaper or more expensive. By creating a model to compare the cost of the conventional design method versus design using VRSs, quick insight can be obtained how much test cost can be added for different amounts of area reduction. Formulas 3.1, 3.2, and 3.3 [28] show the formulas to express the cost of an IC.

$$Cost_{chip} = \frac{Cost_{wafer\_fab} + Cost_{wafer\_test}}{Y \times Chips_{wafer}} \tag{3.1}$$

$$Chips_{wafer} = \frac{A_{wafer}}{A_{die}} \tag{3.2}$$

$$Y = Y_{defect} = \frac{1}{1 + (Defects_{unit\_area} \times \frac{A_{die}}{2})^2} \tag{3.3}$$

Y denotes the yield, $Chips_{wafer}$ the amount of ICs that fit on a wafer, A_{die} the area of the die, A_{wafer} the total area of the wafer, $wafer\_cost_{fab}$ the cost of fabrication of a single wafer with ICs, $wafer\_cost_{test}$ the cost of testing all ICs on the wafer, and $Defects_{unit\_area}$ the average amount of defects per unit area. These equations can be used to compare the cost per IC for conventional design and design with VRSs.

3.1.1 Taking Variability Resilient Schemes into account

Formulas 3.1, 3.2, and 3.3 can be used directly for the conventional design method. For design with VRSs the yield calculation (Formula 3.3) is different. For design with VRSs

parametric yield also needs to be taken into account. With the conventional design method a worst-case-based design is created. This means the parametric yield is 100%, so it does not need to be taken into account.

Designs with VRSs are built with a Better-Than-Worst-Case (BTWC) design methodology, which means less margin is built into the ICs. The VRSs minimize process variation effects, so ICs with slow and fast silicon still meet specifications. However, if the design margin is made very low, then there will be cases that the VRSs can no longer compensate all levels of variation. In this case the parametric yield is no longer 100%. In Formula 3.4 the formula to calculate yield for design with VRSs is given.

$$Y_{VRS} = Y_{parametric} \times Y_{defect} \tag{3.4}$$

3.1.2 Variability Resilient Schemes versus conventional design

The cost model needs to compare the cost of design with VRSs versus conventional design. Therefore the following inequality must hold:

$$Cost_{chip,VRS} < Cost_{chip,conventional} \tag{3.5}$$

These formulas can now be filled in using Formula 3.1:

$$\frac{Cost_{wafer\_fab} + Cost_{wafer\_test,VRS}}{Y_{VRS} \times Chips_{wafer,VRS}} < \frac{Cost_{wafer\_fab} + Cost_{wafer\_test,conventional}}{Y \times Chips_{wafer,conventional}} \tag{3.6}$$

Further simplification gives:

$$\frac{Cost_{wafer\_fab} + Cost_{wafer\_test,VRS}}{Y_{VRS} \times \dfrac{A_{wafer}}{A_{die,VRS}}} < \frac{Cost_{wafer\_fab} + Cost_{wafer\_test,conventional}}{Y \times \dfrac{A_{wafer}}{A_{die,conventional}}} \tag{3.7}$$

$$\frac{Cost_{wafer\_fab} + Cost_{wafer\_test,VRS}}{yield_{VRS} \times \dfrac{1}{A_{die,VRS}}} < \frac{Cost_{wafer\_fab} + Cost_{wafer\_test,conventional}}{Y \times \dfrac{1}{A_{die,conventional}}} \tag{3.8}$$

The model needs to show how much extra money can be spent on testing for different amounts of area reduction. Therefore, $A_{die,VRS}$ needs to be expressed as a fraction of $A_{die,conventional}$ and $Cost_{wafer\_test,VRS}$ as a fraction of $Cost_{wafer\_test,conventional}$. This gives:

$$A_{die,VRS} = \alpha \times A_{die,conventional} \tag{3.9}$$

$$Cost_{wafer\_test,VRS} = \beta \times Cost_{wafer\_test,conventional} \tag{3.10}$$

These new formulas for $A_{die,VRS}$ and $Cost_{wafer\_test,VRS}$ can now be filled into Formula 3.8:

$$\frac{Cost_{wafer\_fab} + \beta \times Cost_{wafer\_test,conventional}}{Y_{VRS} \times \frac{1}{\alpha \times A_{die,conventional}}} < \frac{Cost_{wafer\_fab} + Cost_{wafer\_test,conventional}}{Y \times \frac{1}{A_{die,conventional}}} \tag{3.11}$$

Writing this out gives:

$$\frac{Cost_{wafer\_fab} + \beta \times Cost_{wafer\_test,conventional}}{Y_{VRS} \times \frac{1}{\alpha}} < \frac{Cost_{wafer\_fab} + Cost_{wafer\_test,conventional}}{Y} \tag{3.12}$$

Formula 3.12 shows the final formula that is used for the cost model.

3.1.3 Case Study

Using the cost model, it is now possible to see how much test cost can be increased for a certain area reduction. In Table 3.1 the used values for the parameters are shown. These values represent typical values for NXP ICs. The cost of wafer fabrication and testing are taken as percentages and combined they form the total cost. As can be seen only 10% of the total cost goes towards testing for conventional designs. The wafer diameter is 300 mm.

In Figure 3.1 the inequality from Formula 3.12 is plotted for test cost increase (β) versus die area increase (α). Several graphs are shown for different parametric yields. The lines show when the costs of conventional design and vra design are equal. The area below the lines marks when design with VRSs is cheaper and the area above the line when conventional design is cheaper. In the graph it can be seen that if the parametric yield is 100% and the area is reduced to 90%, the test cost may increase by about 250% before conventional design is cheaper again. This shows that for a relatively small area reduction the test cost increase can be very high. This makes sense, because most cost lies in the wafer fabrication (90% total).

Parameter	Value
$Cost_{wafer\_fab}$	90%
$Cost_{wafer\_test,conventional}$	10%
A_{die}	0.5 cm^2
$Defects_{unit\_area}$	0.25/cm^2
d_{wafer}	300 mm

Table 3.1: Paramater values used in the case study for the cost comparison of conventional design versus design with VRSs

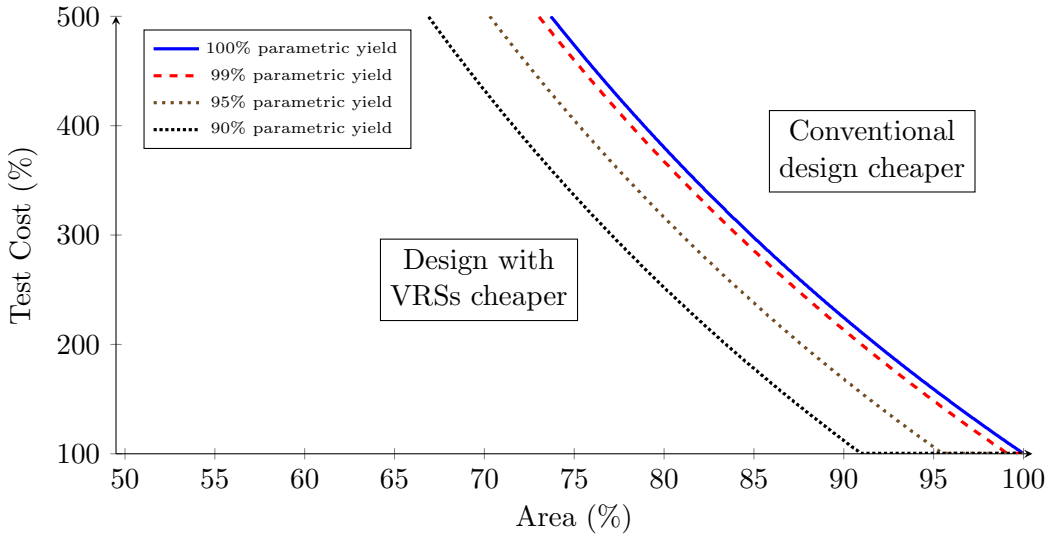


Figure 3.1: Cost model results for the case study. Results are shown for 100%, 99%, 95%, and 90% parametric yield. When below the line, design with VRSs is cheaper, above the line, conventional design is cheaper.

3.1.4 Validation

In the proposed cost model the calculation of the number of ICs per wafer (Formula 3.2) is very simplistic: the assumption is made that all area of the wafer can be used for the production of ICs. In reality this is not the case. This is due to the fact that a wafer is round and the dies are rectangular. Because of this, some areas on the edges cannot be used for IC production. Furthermore, edge exclusion also reduces the amount of usable area. When the edge exclusion is 3 mm, roughly 4% of effective area is lost[3].

In order to check, whether the assumption that all area of the wafer can be used for IC production, is sufficient, the cost analysis from Figure 3.1 has been performed again with a more accurate way of calculating the number of dies that fit on a wafer. The used method for this is the *gross die per wafer two term* formula from [3]. During calculation, it is assumed that the edge exclusion is 3 mm, so the effective radius of the wafer is 147 mm.

Figure 3.2 shows the outcome of the cost analysis, when the *gross die per wafer two term* is used to calculate the number of dies per wafer. When comparing the obtained graphs with the graphs in Figure 3.1 it can be seen that they are very similar. Therefore, it can be concluded that the approximation for the number of dies per wafer is good enough, when Formula 3.2 is used.

The MATLAB code for both the simplistic cost model, which uses Formula 3.2, and the cost model that uses the *gross die per wafer two term* can be found in Appendix A.

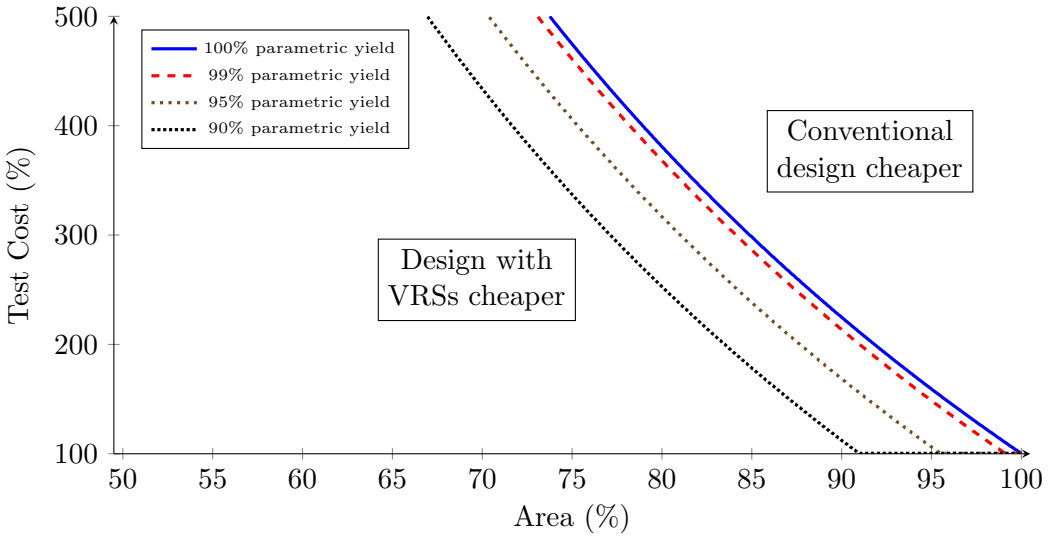


Figure 3.2: Cost model results for the case study, but now using the *gross die per wafer two term* from [3].

3.2 Variability Resilient Schemes vs Testability

Design For Test (DFT) stands for the set of IC design techniques that add testability features. Extra hardware (scan-chains) is added that enables better and faster testing of the IC. Hence, extra area is added to the IC to ensure good testability. This added area for testing pays off as it enables faster testing, as well as higher quality products for the customer.

When introducing new design methods, such as VRSs, that offer new advantages it is still possible that these advantages are outweighed by the fact that the testability is not high enough or it is too expensive to make it high enough. In this case it is hard to guarantee that a high quality product is sold.

In this section the testability challenges of each VRS are evaluated. First of all, the testing method of conventional designs is discussed. After this, body biasing, clock stretching, and the error avoidance flip-flop are discussed.

3.2.1 Conventional Design Testing Method

For digital ICs designed with a worst-case based methodology the following tests are performed[1]:

- Stuck-at-fault testing
- Transition-fault testing

With stuck-at-fault-testing the design is tested for faults that make signals stay stuck at a logic '0' or '1'. With stuck-at-fault-testing defects, such as shorts can be detected[29].

[1]Description of conventional design testing is based on NXP.

With transition-fault testing, it is tested whether signals are *slow-to-rise* or *slow-to-fall*. Using this technique delay defects can be detected. The transition fault model assumes that the defect adds a large amount of delay comparable to or even higher than the period of the target clock speed[30]. Because of this, it suffices to sensitize short paths for detection of transition faults. Therefore, this test is only suitable to detect large amounts of added delay along paths. The advantage of transition-fault testing, is that the computational effort to generate tests is a lot lower, as well as the time needed to perform the tests.

Subtle manufacturing defects that cause a very small added delay in the circuit are usually not detected by transition-fault tests. This is due to the fact that transition-fault tests test for delay faults through sensitization of short paths. These defects that cause a very small added delay are referred to as *small-delay defects*[31]. Small-delay defects in long paths have a chance of adding enough additional delay, so the IC is no longer able to run at the required speed. Detection of small-delay defects requires more expensive tests, such as *Timing-Aware ATPG*[32], which aims to detect delay faults through long paths or *faster-than-at-speed* testing[33], which tests the circuit at a higher speed than the functional speed.

Figure 3.3 shows an example probability density function of the speed of ICs manufactured using older technology. The worst-case corner is indicated with an arrow. What can be seen is that the majority of the ICs run faster than the worst-case corner. Thanks to this, cheap tests such as stuck-at-fault-testing, and transition-fault testing are enough for a high quality[2] product. This is due to the fact that when an IC is affected by a small-delay defect, it is usually still able to run at the minimum speed. This is illustrated in Figure 3.4. As can be seen, the parametric speed and the actual speed are indicated by arrows. The parametric speed is the speed that the IC is able to run at based on its silicon properties, which are determined by the global and local process variation. Due to a small-delay defect the actual speed of the IC is lower, however. The actual speed is, however, still faster than the worst-case corner speed, for which the IC has been designed. Therefore, this IC is still able to run at the minimum speed specification.

Once ICs have a maximum speed that is close to the worst-case corner, small-delay defects are able to make the IC slower than the minimum specification. When looking at the distribution a very small amount of ICs has a speed close to the worst-case speed, however. Therefore, there will be a very small amount of ICs that could be affected by a small-delay defect that makes it actual speed fall below the minimum specification. Because of this, doing stuck-at-fault testing and transition-fault testing is enough for high quality products.

[2]Quality is the fraction of faulty chips amongst the chips that pass the tests, expressed as parts per million (ppm)[34].

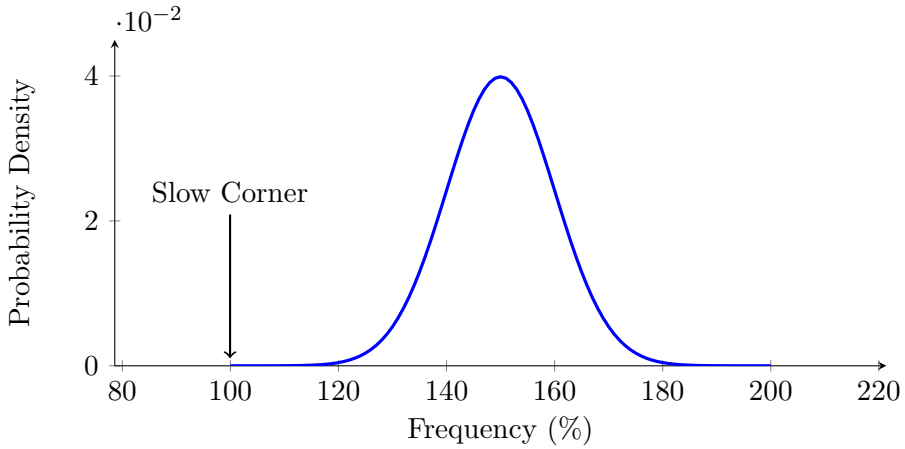


Figure 3.3: Example of probability distribution of IC speed for older technology.

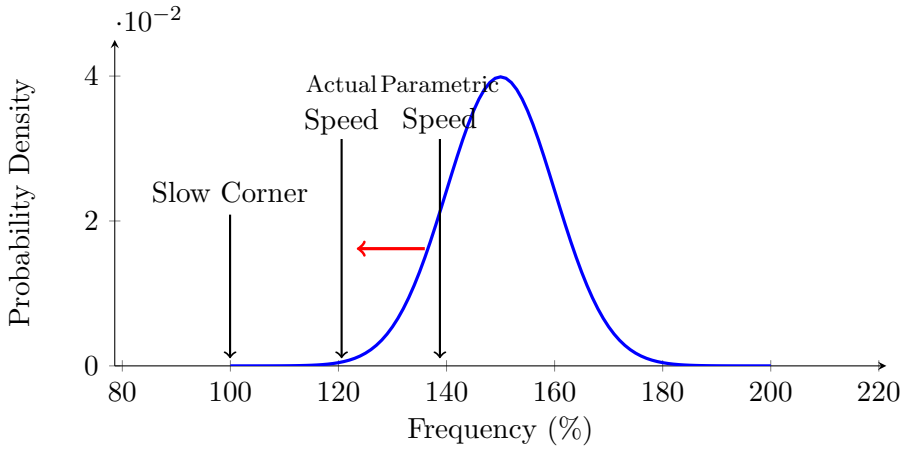


Figure 3.4: Example of effect of small-delay defect in older technology. As can be seen the affected IC still has a speed above the worst-case corner, which determines the target speed.

3.2.2 Body Biasing

As discussed in Section 2.3, body biasing is a technique that allows to change the transistor's threshold voltage. By lowering the treshold voltage an IC is made faster, and by lowering the threshold voltage it is made slower, but less leaky. ICs with slow silicon are made faster using forward body biasing, and ICs with fast silicon are made slower, but less leaky with backward body biasing.

Figure 3.5 shows an example of the impact of body biasing on the IC speed distribution. By applying forward biasing for slow ICs and backward biasing for fast ICs the distribution becomes more narrows. This narrow distribution has the advantage that the specifications of the product improve. The disadvantage is, however, that ICs become more susceptible to small-delay defects. The arrow 'parametric speed' indicates the performance of an IC, however, because of a small-delay defect the actual speed is

lower. This actual speed is marked with the arrow 'actual speed'. As a result of the small-delay defect, the IC no longer meets the minimum speed requirement.

Because the distribution with body biasing is so steep, there are suddenly a lot of ICs that will perform below specification, when affected by a small-delay defect in long paths. Therefore, this is a serious threat for the quality level. In this case more extensive, and expensive testing is necessary.

Instead of doing more extensive testing it is also possible to add to add more margin to the design in order to ensure the slowest samples after performing forward body biasing are further removed from the target speed. From a design point of view this is less ideal, as it means speed increase, and/or area savings, and/or power usage reduction are lost due to testing.

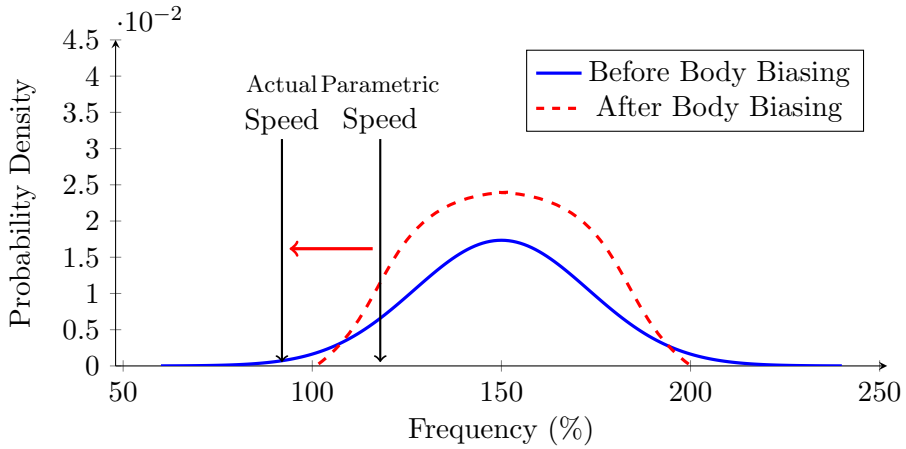


Figure 3.5: Effect of small-delay defect on body biased circuit

Evaluation Experiment

In order to evaluate the ability of body biasing to compensate process variation, the following experiment is proposed: A design with the same functionality is created twice. One is created with a worst-case based design methodology and the other with a BTWC based design methodology. Once both designs are finished, they are placed onto the same die, which is then fabricated. Furthermore, corner lots are produced, so samples with typical, slow, and fast processing are available. Having both designs on the same die, means they will both suffer from the same global variation. Because of this, a fair comparison can be made between the worst-case and BTWC designs on the same die.

Once the ICs are fabricated, the performance of the worst-case based design and the BTWC design are compared. A look is taken at the speed and power consumption of both designs. The speed of the worst-case based design functions as a reference speed, as it should still run at the speed specification under worst-case Process, Supply Voltage, and Temperature (PVT) conditions. The BTWC design will most likely not meet this speed under worst-case PVT conditions. Using forward biasing it can be attempted to make the BTWC as fast as the worst-case based design. For the fast corner samples the potential of decreasing power consumption using reverse biasing can be investigated.

In order to determine the speed of the designs, *functional testing* or *structural testing* can be used. With functional testing an application is run multiple times at increasing clock speeds. For every clock speed it is verified if the test ran correctly. The highest clock speed of the design is then the last frequency at which the application ran correctly. A disadvantage of functional testing is that it is hard to selectively target specific paths. Therefore, getting a high coverage is very challenging[35]. Also, in case the test does not sensitize critical paths in the design, the measured maximum speed of the design is higher than the actual maximum speed. Another disadvantage is that it does not give any information about the speed of shorter paths.

Structural testing uses the DFT network in the IC. With the DFT network it is possible to perform *path delay tests*. With a path delay test a transition is created along the *path under test*. At the next cycle, this transition is captured in a scan flip-flop at the output of the path. By performing a scan-out and checking the value in the scan flip-flop it can be checked if the path finished propagation. By running the path delay test at increasing clock frequencies the propagation delay time of the path can be measured. In this case, the highest frequency at which the path did not fail is the maximum frequency the path can run at. An advantage of structural testing is that when critical paths in the design are tested, the maximum speed of the IC can be determined more accurately. Another advantage is that the delay of shorter paths can be measured as well. Therefore, it can be determined for a big selection of paths how their propagation delays are affected by body biasing. Furthermore, the test time is shorter than functional testing and high quality can be achieved. Because of this, path delay testing is chosen over functional testing.

There are two kinds of path delay tests: *robust* and *non-robust* path delay tests[34]. With a non-robust path delay test, the output of the path can be controlled by off-path inputs. Therefore, a transition at the output can also be caused by another path. This makes it impossible to guarantee that the observed transition at the output of the path is caused by the path of interest. Because of this the delay of the path cannot be measured reliably. With a robust path delay test only the signals on the path itself control the output. So the transition at the output is caused by the path of interest, which is the desired behavior. Therefore, only robust path delay tests are used.

For the experiment path delay tests are created for both the worst-case and BTWC design. For both designs the set of tested paths consists of long, medium length, and short paths. The speed of the paths can then be compared between both designs. The experiment described above is performed in Chapter 4.

3.2.3 Clock Stretching

A design that uses clock stretching uses a decoder that determines based on input operands/instruction, whether the clock is stretched. This decoder is just added digital logic, so it does not provide any new testability challenges.

With clock stretching, the clock is stretched when a critical path is activated. This means there is a slight drop in performance of the IC. In order to compensate for this drop in performance, it is possible to make the design run at a higher frequency. This is possible due to the fact that for the critical paths the clock is stretched. Therefore,

the clock speed is no longer determined by the propagation delays of the critical paths. The shorter paths will now run at a higher speed. This means the shorter paths will run closer to their maximum speed.

Typically, in a design only a small amount of all paths is critical. Instead, medium length paths dominate[4]. Small-delay defects are especially a risk on the critical paths, as small amounts of added delays can introduce timing errors. Considering there are relatively few critical paths, the chance is also small that a small-delay defect will affect a critical path. When the shorter paths become critical paths due to clock stretching, however, suddenly a lot more paths qualify as being critical. This is illustrated in Figure 3.6, which shows an example of a path delay distribution in an IC. For a normal design the area accentuated with horizontal stripes would be the set of critical paths. With clock stretching this set of paths gets more time to propagate and, therefore, it is no longer critical. When the clock speed is increased to compensate for the lost performance, however, the area accentuated with vertical stripes becomes the set of critical paths. This area is bigger than the area of critical paths when no clock stretching is used. Thus, the amount of critical paths becomes bigger. Due to the bigger amount of critical paths, the chance is bigger that one of these paths is affected by a small-delay defect. Therefore, it will be necessary to test paths for small-delay defects.

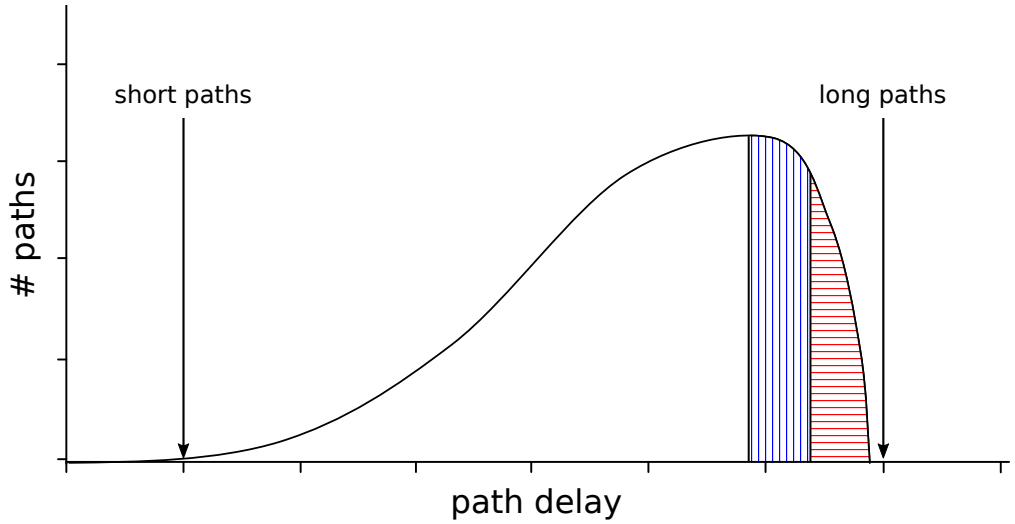


Figure 3.6: Example of path delay distribution in an IC (from [4])

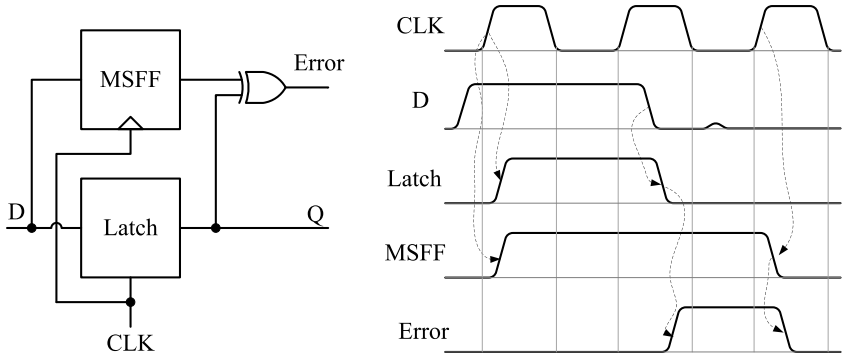
3.2.4 Error Avoidance Flip-Flop

The error avoidance flip-flop is a new circuit element, which replaces existing flip-flops. For testability most flip-flops are replaced by scan flip-flops. These scan flip-flops are connected into a chain. Through this chain, the values in the scan flip-flops can be controlled and observed. By controlling the values in the scan flip-flops, the logic connected to them can be sensitized. Finally, by storing the values that the logic generated with the scan flip-flops, the circuit's response can be observed. In case there are faults in the circuit, these can then be caught, because the circuit's response is incorrect.

With scan flip-flops the Q output can be controlled. This is also needed for the error avoidance flip-flop, so the connected logic can be tested. Furthermore, the error avoidance flip-flop has an extra output: the ERROR signal. For good testability it needs to be possible to control this output as well. Sensitizing this output to a logic '1' is difficult. As explained in Section 2.4, the error avoidance flip-flop samples the D input using a normal clock and a delayed clock. When the sampled values differ, the ERROR output becomes high. Therefore, in order to sensitize the ERROR output to a high value, the error avoidance flip-flop will need to sample different values on the D input at the rising edge of the normal and the delayed clock.

In [5] the process of adding testability to a latch based timing speculator is described. The design and timing diagram of the timing speculator are shown in Figure 3.7. Basically, the timing speculator consists of a flip-flop (MSFF) and a latch. The latch will capture the value on input D as long as the clock signal is high. If the value on input D changes during this period (due to a timing error) this value is captured and propagated to the XOR gate. Because the input values for the XOR gate differ, the ERROR output becomes high.

In [5] it is assumed that the ERROR signals coming from the timing speculators are accumulated into one ERROR signal using an OR-gate tree, shown in Figure 3.8. The accumulated ERROR signal is then captured by a *final error flip-flop*. It is possible that there are multiple OR-gate trees, for instance one per pipeline stage.



(a) Timing speculator (b) Timing diagram

Figure 3.7: Latch based timing speculator (from [5])

In [5] the following cases are considered:

- Stuck-at testing OR-gate tree
- Delay testing OR-gate tree

The same cases will be considered for the error avoidance flip-flop.

Stuck-at testing OR-gate tree

For this case, it needs to be possible to control the ERROR output of each error avoidance flip-flop to a logic '0' and a logic '1'.

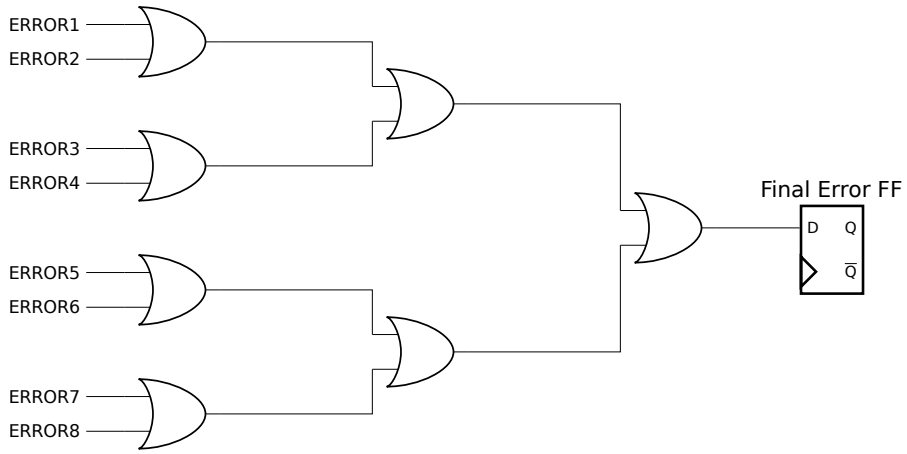


Figure 3.8: OR-gate tree with final error flip-flop.

Figure 3.9a shows the scan-chain setup with timing speculators from [5]. The ERROR signal of the timing speculator is sensitized to a logic '1' by purposely introducing a hold time violation. This is illustrated in Figure 3.9b. A $0 \rightarrow 1$ transition is propagated through the consecutive scan cells. When the time that the clock is high (denoted by T_H) is long enough, Latch1 will already capture the consecutive value through the scan chain coming from MSFF0. MSFF1 will then hold a '0', but LATCH1 will hold a '1'. Therefore, Error1 becomes high. This signal can be observed in the final error flip-flop at the end of the OR-gate tree.

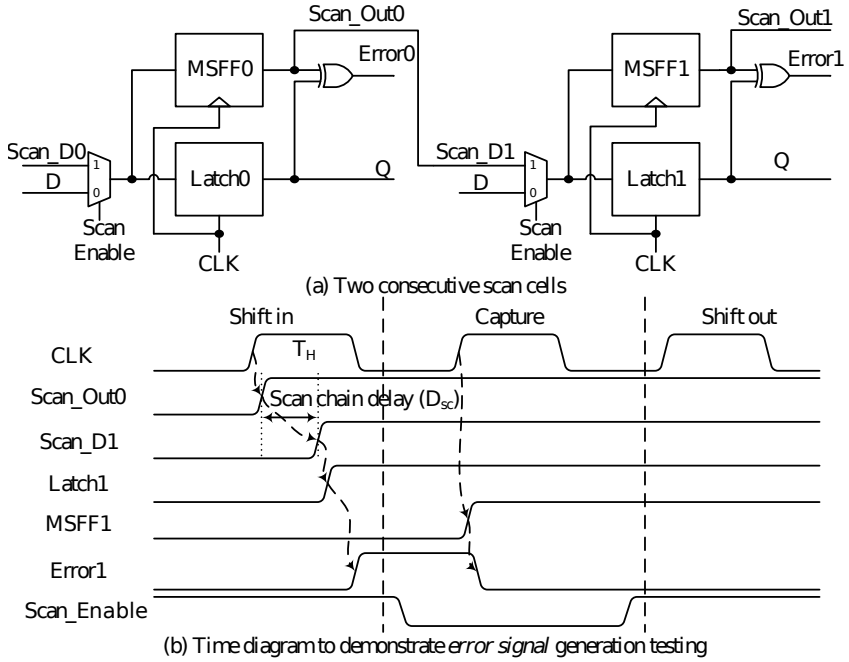
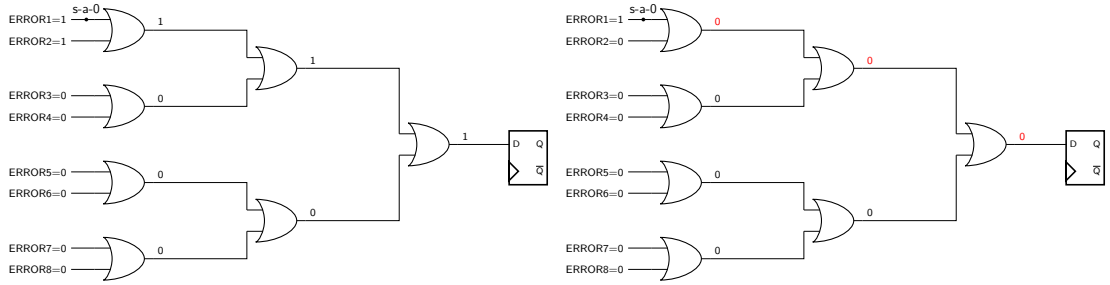


Figure 3.9: Error signal generation for timing speculator (from [5])

One thing to note is that it is only possible to test the ERROR output of one timing speculator belonging to the same OR-gate tree per scan vector. Otherwise, fault masking occurs. This happens when there is a stuck-at-0 fault in the OR-gate tree, which is illustrated in Figure 3.10. In Figures 3.10a and b there is a stuck-at-0 fault at ERROR1. This fault can be detected, by sensitizing ERROR1 to a logic '1' and capturing the faulty response of the circuit with the final error flip-flop. In Figure 3.10a ERROR2 is also a logic '1'. Because of this a logic '1' is still captured in the final error flip-flop, so the fault goes undetected. In Figure 3.10b only ERROR1 is sensitized to a logic '1'. The final error flip-flop now captures a logic '0', which is a faulty response of the circuit. Hence, the fault is detected.



(a) Masking occurs (b) No masking

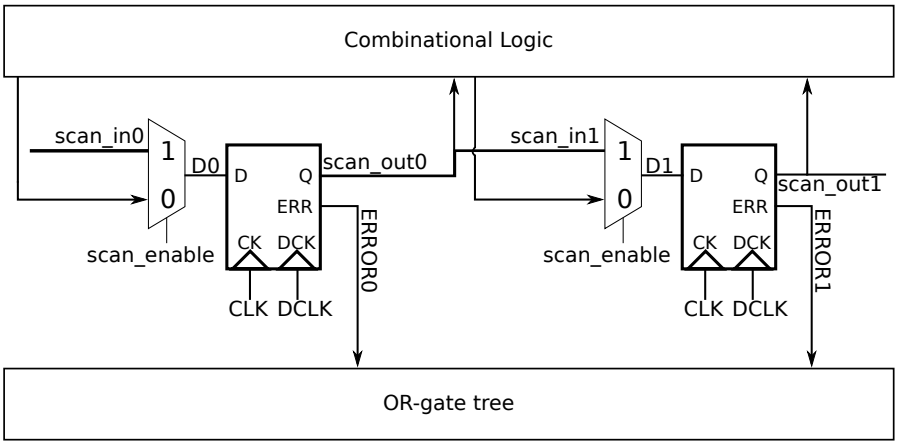
Figure 3.10: Error masking in OR-gate tree

For the error avoidance flip-flop a similar method of controlling the ERROR output to '1' is possible. The error avoidance flip-flop is edge-triggered on both the normal and the delayed clock. The delay between the normal and the delayed clock is fixed by design, as the delayed clock is created by adding a delay element to the normal clock. However, if the delay between the normal clock and the delayed clock is long enough, it is possible to also introduce a hold time violation during scan in. In this case the delay of the scan chain between two error avoidance flip-flops needs to be low enough, so the new value for the previous flip-flop will already be on the input of the next flip-flop before the rising edge of the delayed clock.

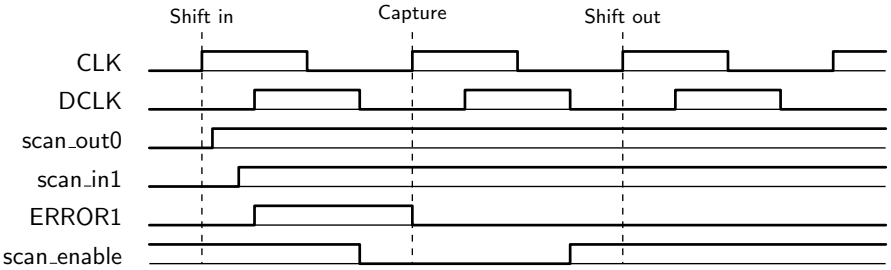
Using hold time violations to test the ERROR output of the error avoidance flip-flop is illustrated in Figure 3.11, which shows an example of a scan chain with two error avoidance flip-flops and the corresponding waveform for stuck-at-testing of the ERROR output. A $0 \rightarrow 1$ transition is propagated through the consecutive scan cells. When the delay between the normal and the delayed clock is high enough, scan_in1 will capture a '0' on the rising edge of the normal clock and capture a '1' coming from scan_out0 on the rising of the delayed clock. Because the captured values differ, ERROR1 becomes high.

The design of the error avoidance flip-flop needs to be altered slightly to use the scheme in 3.11. The propagation of values from latch M1 to the slave latch needs to be disabled in test mode. In case this is not done, the output value of the error avoidance flip-flop will be updated on the rising edge of the delayed clock. This happens because of the hold time violation that is used to control the ERROR output. In this case the

consecutive value in the test vector will be captured by latch M1. If the value in latch M1 is also propagated to the slave latch, output Q will be updated with this value. Hence, the test vector gets corrupted during scan-in. The revised design of the TIMBER flip-flop is shown in Figure 3.12. A NAND gate is added, so P1 is always low, when scan enable is high. Therefore, the propagation of values captured on the delayed clock to the output Q of the error avoidance flip-flop is now prevented.



(a) Scan chain design with error avoidance flip-flops



(b) Stuck-at-'0' testing of error avoidance flip-flop's ERROR output.

Figure 3.11: Stuck-at-'0' testing of error avoidance flip-flop's ERROR output using hold time violations during scan in.

Delay testing OR-gate tree

For the timing speculators[5] two methods are proposed to delay test the OR-gate tree: *short path sensitization* and *long path sensitization*. For short path sensitization, it is assumed that the clock generator has the option to control the period that the clock is high. By making the period that the clock is high longer, a short path will finish propagation within this period. With this a hold time violation can be created, because the latch will capture the value from the short path on the D input. Therefore, the ERROR signal will make a rising transition.

Long path sensitization is focused on creating a timing fault on one of the paths connected to the D input of the timing speculator. By doing this, the timing speculator will detect a timing error and it will raise the ERROR signal. If this procedure is done

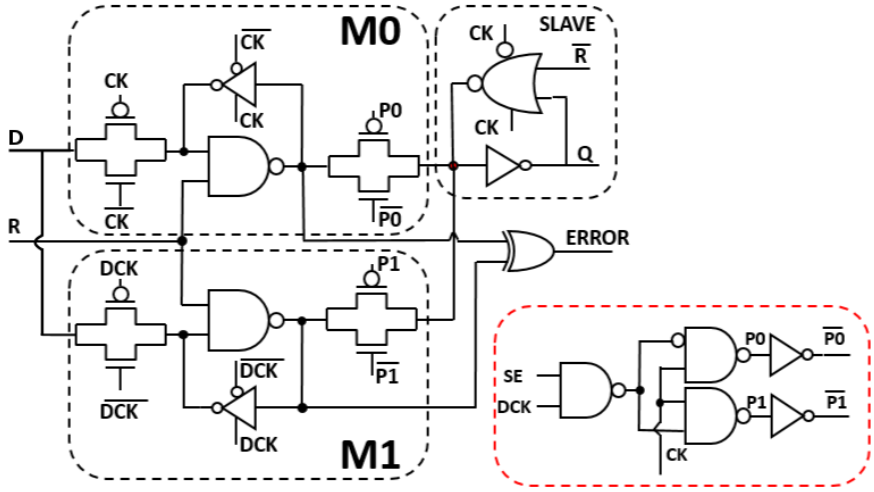


Figure 3.12: Proposed DFT circuit of error avoidance flip-flop. A NAND gate is added, so the propagation of the value in M1 to the slave latch is blocked, when scan enable is high.

at functional speed and the ERROR signal is observed at the end, it can be checked if there are delay faults in the OR-gate tree or the timing speculator's ERROR output.

Introducing hold time violations using short path sensitization is not an option for delay testing the ERROR output of an error avoidance flip-flop. This is due to the fact that it is edge triggered on both the normal and delayed clock and not level sensitive like the timing speculator. The delay of the delayed clock compared to the normal clock is fixed by design. Therefore, paths with an error avoidance flip-flop at the end also cannot have a propagation delay lower than the difference in time between the delayed and the normal clock. Otherwise, hold time violations will also occur during normal use.

Long path sensitization is an option for delay testing the ERROR output of an error avoidance flip-flop. In this case three clock cycles are needed. The first two cycles are needed to launch the transition along the long path and capture the response with the error avoidance flip-flop on its D input. By controlling the period between these two cycles a timing error can be created on the D input of the error avoidance flip-flop. When the value captured on the rising edge of the normal and delayed clock differs, the ERROR output will rise. The rising ERROR output can then be observed in the third cycle. This process is illustrated in Figure 3.13. It can be seen that a timing error occurs at the D input (highlighted in the figure), because input D is updated after the rising edge of the normal clock, but before the rising edge of the delayed clock. This timing error is created by generating a transition along a long path. In the third cycle the ERROR output is then captured.

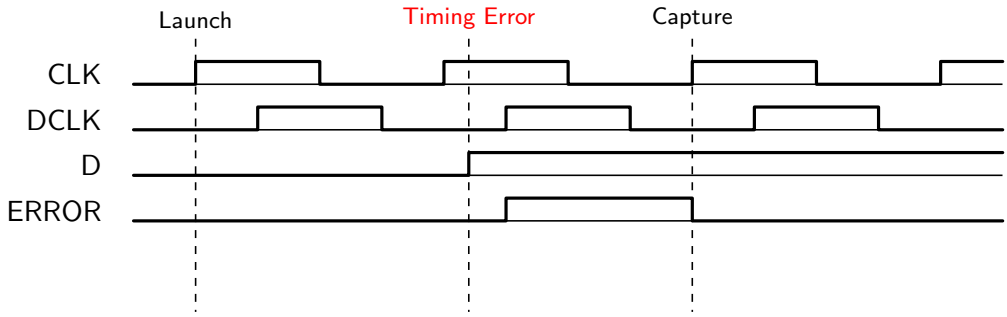


Figure 3.13: Delay testing of error avoidance flip-flop's ERROR output.

Limitations

There are some limitations to the proposed DFT solution for the error avoidance flip-flop:

- The delay between the normal clock and the delayed clock for the error avoidance flip-flop needs to be high enough. Otherwise, it is not possible to use the above described method of stuck-at-testing the ERROR output using hold time violations.
- Only one ERROR output of an error avoidance flip-flop can be tested per OR-gate tree per test vector. This goes for both stuck-at as well as delay fault testing. If the OR-gate tree is very big this means a lot of test vectors are needed, which will have a negative impact on test time.
- For delay testing of ERROR outputs, a frequency first needs to be found at which the path fails. This is due to the fact that it is hard to predict the precise propagation delay due to process variation. Finding this frequency is an iterative process, which means the test needs to be performed multiple times, until the path fails. This negatively affects the test time.
- One ERROR output of an error avoidance flip-flop per OR-gate tree can be tested per test vector. Because of this, it may be hard to generate tests along long paths that will only create a transition at the input of one of the error avoidance flip-flops. Therefore, it is likely that it is not possible to do delay tests for all ERROR outputs.

Based on these limitations, it can be concluded that test time can become an issue and also that it is most likely not possible to do delay tests for every ERROR output of the error avoidance flip-flops. Considering the fact that it is most important to get a good coverage of stuck-at-faults, a scheme which allows for quick testing of stuck-at-faults, but does not allow for delay fault testing, may be interesting. This can be done by modifying the design of the error avoidance flip-flop, so that when scan enable is high, the ERROR output is *always* high. This way using an extra NAND gate (which is only used for testing) the whole OR-gate tree can be tested for stuck-at-0 faults. This is illustrated in Figure 3.14. When all ERROR signals from the error avoidance flip-flops are high, then all inputs of the NAND gate should be high and, therefore, the output

of the NAND gate will be low. If there is a stuck-at-0 fault in the OR-gate tree, then one of the inputs to the NAND gate will be low. In that case, the output of the NAND gate is high. When using this scheme, only one vector is needed to test for all stuck-at-0 faults in the OR-gate tree. As future work, it would be interesting to further investigate this scheme.

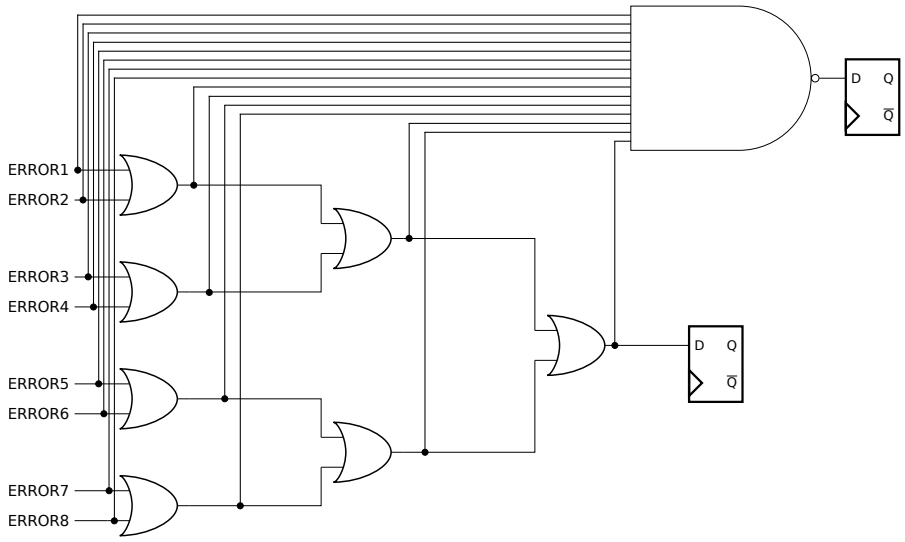


Figure 3.14: Alternative DFT solution for the error avoidance flip-flop.

3.3 Conclusions

This chapter presents the evaluation methodology for VRSs. First of all, a cost model is presented that can be used to compare the cost of conventional worst-case design with design with VRSs. With the cost model it can be checked if design with VRSs is cheaper, when the area of the IC is reduced, but the test costs increase.

Next, the testability challenges of each VRS are discussed. For body biasing the more narrow distribution of IC speeds is a challenge. Due to this more narrow distribution there are more ICs that might not meet the target speed when there is a small-delay defect. An experiment is proposed to evaluate body biasing. This experiment uses path delay tests in order to evaluate the speed of ICs.

For clock stretching a testability challenge is that more critical paths are created. This is due to the fact that the design can be run at a higher clock speed to compensate the drop in performance due to clock stretching. Typically, in a design there are only relatively few critical paths. Instead, there are more slightly shorter paths. These paths will now become critical paths. Therefore, the design will have more critical paths which can be affected by delay defects.

Finally, a look is taken at the testability challenges of the error avoidance flip-flop. For the error avoidance flip-flop it is a challenge to control the ERROR output. A solution is proposed with a low area overhead. A disadvantage of the proposed solution is that test time can become high when the design has a lot of error avoidance flip-flops.

Furthermore, it is hard to get a good coverage of delay faults and test time can become high for them.

Experimental and Industrial Evaluation of Variability Resilient Schemes

4

This chapter discusses the performed experiments and obtained results. Path delay measurements are performed to evaluate the performance of a worst-case design and a Better-Than-Worst-Case (BTWC) design. With these expirements both designs are compared. Furthermore, the potential of body biasing to compensate process variation is investigated. Section 4.1 describes the experimental setup. Section 4.2 discusses the path delay test generation results. Section 4.3 presents the simulation results and Section 4.4 the measurement results. Section 4.5 compares the simulation results with the measurement results. Section 4.6 evaluates the potential of the BTWC design using the cost model from Chapter 3. Finally, Section 4.7 concludes the chapter.

4.1 Experimental Setup

This section describes the experimental setup. First, the test platform is discussed. After this, the path delay test generation process is discussed. Finally, the performed experiments are discussed.

4.1.1 Test Platform

For the experiments the Beetle test platform[1] is used. The Beetle is a test chip designed in a 40 nm process, created to evaluate several Variability Resilient Schemes (VRSs). Figure 4.1 depicts the schematic of the Beetle test chip. The Beetle chip exists of four ARM Cortex M3[36] processors. Each core is designed differently. The differences can be found in used VRSs and the amount of margin against process variation.

Other than being on the same die, the cores are completely individual and do not share any other functionality whatsoever: there are no internal communication channels between the cores and each core has its own set of pins needed to get a working core, such as power pins, clock pins, and Joint Test Action Group (JTAG)[37] pins. One could take the design of one of the cores and produce it as a chip with only one core relatively easily. There is a good reason for the cores being on the same die, however: the cores will have similar process variation. Therefore, a fair comparison can be made between cores on the same die.

In Table 4.1 the specifications that the cores have in common are given. As can be seen, the cores are designed to run at a clock speed of 180 MHz. The clock network, however, has been designed to be able to achieve frequencies of around 500 MHz. The speed of the clock network is higher, so it can be tested if the cores can run at a higher speed without failures. Furthermore, for the characterization of the silicon using path

[1]The Beetle is a test chip designed by NXP, used to evaluate VRSs for the VRA project.

delay testing, shorter, non-critical paths need to be tested as well. In order to measure the speed of a path, the first frequency at which it fails needs to be found. Therefore, the clock network needs to be able to handle higher clock speeds.

Corner lots with typical, but also slow and fast processing are produced for the Beetle. This way it can be checked for each core if it is able to achieve its target specifications under the worst-case process and operating conditions.

In Table 4.2 the relative area usage of each core compared to core 1 is listed. It can be seen that core 2 saves the most area (25%), as the whole circuit is designed with a BTWC methodology. All cores have separate power networks for the body bias voltages, which are supplied through external pins.

Next, the special features of the cores and the differences between them are described. Afterwards, sigma design is explained.

Table 4.1: Common specifications between four cores on the Beetle test chip

Specification	**Value**
Process	40 nm CMOS
$F_{signoff}$	180 MHz
$F_{ClockNetwork}$	500 MHz
$VDD_{nominal}$	1.1 V
T_{MIN}	−40 °C
T_{MAX}	125 °C
Path Delay Test Technique	Launch-on-Capture

Table 4.2: Area of each core of the Beetle platform. The area usage is expressed relative to the area of core 1.

Core	**Area**
Core 1	100%
Core 2	75%
Core 3	90%
Core 4	90%

Core 1

Core 1 has been designed with a conventional worst-case based design methodology. This means a lot of margin is added to make sure all process corners will meet specifications. Core 1 is meant to serve as a reference for the other cores for comparing performance (speed), power consumption, and area usage. Especially comparing the performance of core 1 with other cores is important. The other cores are designed with a BTWC based design methodology. Core 1 should still meet the 180 MHz specification under worst-case Process, Supply Voltage, and Temperature (PVT) conditions. For the other cores the added VRSs, hopefully, compensate process variation enough to also meet the 180 MHz specification.

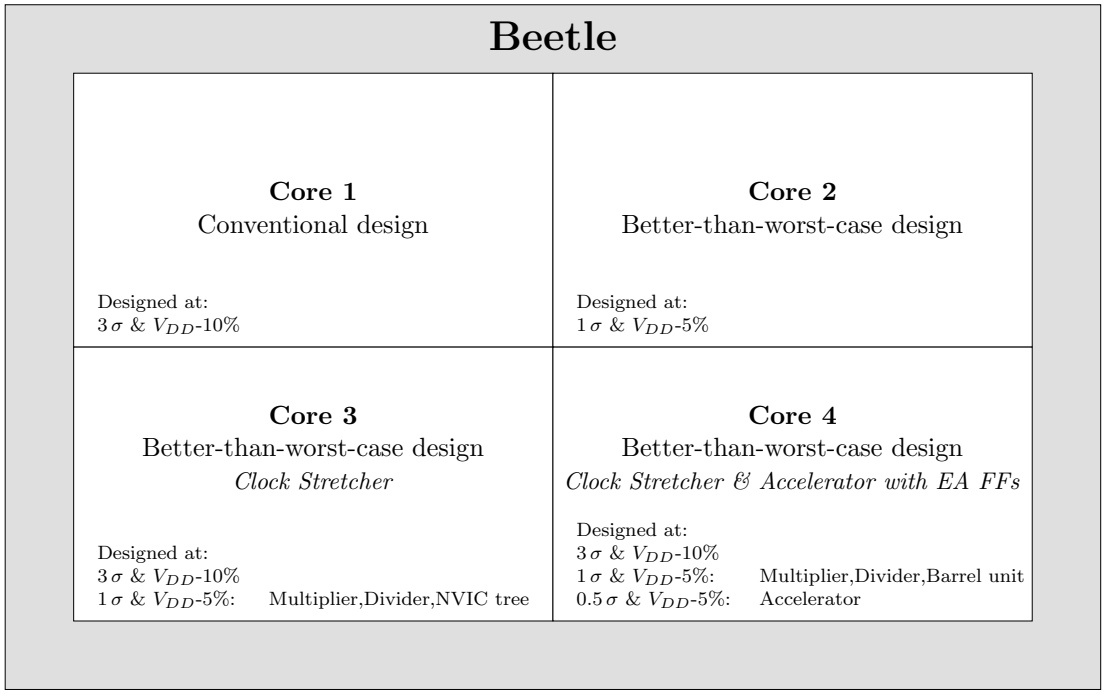


Figure 4.1: Beetle Platform, showing its four cores and synthesis constraints.

Core 2
Core 2 has been designed with a BTWC based design methodology. The whole core has been designed with a 1 sigma BTWC library. Furthermore, it is assumed that V_{DD} will only vary by 5% from the nominal value of 1.1 V.

Core 3
In core 3 conventional worst-case based design and BTWC based design are combined, meaning certain parts are designed with full margin and certain parts with less margin. The multiplication, division, and NVIC tree units have been designed with a 1 sigma library with an allowed variation of 5% for V_{DD}. Thanks to this, roughly 10% of area is saved compared to core 1.

Furthermore, core 3 features a clock stretcher. This clock stretcher will activate based on the operation that is about to be performed. It is possible to activate the clock stretcher for any operation: through a set of registers the operations for which clock stretching is used can be programmed. For core 3 it makes sense to activate the clock stretcher for multiplication, division and NVIC tree operations, as these may not be able to run at the target frequency.

Core 4
Just like core 3, core 4 is a combination of worst-case based design and BTWC based design. In core 4 the multiplication, division, and barrel shifter units are designed with a 1 sigma library with an allowed variation of 5% for V_{DD}. It as well features a clock stretcher.

Also, core 4 has an accelerator featuring error avoidance flip-flops. This accelerator implements a simple pipeline circuit that performs a multiply-accumulate[38] operation. Between the stages of the pipeline, error avoidance flip-flops are used. The accelerator is accessible for the Cortex M3 through an internal bus. The accelerator is designed with very little margin: a 0.5 sigma library is used. The error avoidance flip-flops are used to avoid timing errors that may arise due to this very low design margin.

Sigma Design
At NXP the amount of margin against process variation in a design is denoted with n sigma, with n reflecting the amount of margin. A higher n means more margin is added. In order to create a design at n sigma, an n sigma Static Timing Analysis (STA) library is used during synthesis.

These libraries are created by performing analog simulations with the gates that are used in the design. The propagation delay is determined for all kinds of load, input slopes, temperatures, and supply voltages during these simulations. It is possible to vary the amount of local and global variation in the simulations. The simulation files of the gates contain paramaters through which the amount of local and global variation that occurs can be set. These parameters can be set from 0 to 1. For a 3 sigma design these parameters are set to 1. In this case, the worst-case of local and global variations are activated. Therefore, with a 3 sigma library a worst-case based design is created.

The parameters to set the amount of global and local variations are set to only 0.33 for the 1 sigma libraries. Therefore, the worst-case variation is not accounted for with this library. Thus, with this library a BTWC based design is created.

4.1.2 Path Delay Test Generation

For the path delay Automatic Test Pattern Generation (ATPG), Mentor Graphics' **Tessent**[39] is used. Tessent needs a list of paths as input data. A path description consists of the begin- and endregister, the gates/nodes in the path and the transitions at every node. Using this list of paths, Tessent will try to generate delay tests for them. As Tessent needs a list of paths to perform path delay ATPG, a tool is also needed to generate these paths. For the path generation Cadence's **Tempus**[40] is used, which is an STA tool. It is possible to instruct Tempus where to look for paths, as you can give a list of begin- and endregisters between which it needs to analyze paths. Tempus will then look for the most critical paths between these registers.

The goal of the path delay test generation is to obtain a comparable set of paths for each core. As the cores are very similar and the difference lies mostly in synthesis constraints and added hardware such as a clock stretcher, the cores still have a lot of the same registers. The logic between these registers will implement similar functionality. So, if there is a long path between a certain begin- and register pair for one core, it is likely the other cores will have a long path as well. Therefore, if every core has paths for the same set of begin- and endregister pairs, a comparable set of test paths is obtained. Furthermore, paths with the same begin- and endregister can be compared directly between cores.

In order to get a set of test paths for every core with the same set of begin- and

endregister pairs, one could generate a lot of paths with Tempus and then feed these to Tessent to make it try to create path delay tests for them. Once Tessent is done creating the tests, it can be checked per core for which begin- and endregister pairs a path delay test has been created. Finally only the paths with the same begin- and endregister are selected that have a test for every core. A more efficient way is to first generate a lot of paths for a single core, the *reference core*, and perform path delay ATPG for these paths. Once the path delay ATPG is done, it can be checked which paths are covered. The begin- and endregisters can then be extracted from these paths to create a list of begin- and endregister pairs. Next, paths are generated for every core between these begin- and endregister pairs and path delay ATPG is performed for them. The idea behind this is that when a path with the same begin- and endregister for one core can be tested, there is a good chance that a path with the same begin- and endregisters of the other cores can be tested as well. This is because the implemented logic is similar.

Figure 4.2 shows a flowchart of the path delay test generation. The process consists of three stages. In the first stage path delay ATPG is performed for the reference core. For this process a list of begin- and endpoints is needed. Tempus will generate then generate paths between this list of begin- and endpoints. These paths are then fed to Tessent, which will try to generate path delay tests for them. Finally, it is checked for which paths a robust test is generated. From these paths the begin- and endpoints are extracted.

In the second stage path delay ATPG is performed for all cores. Paths are generated using the list of begin- and endpoints from stage one this time. Again, for these paths path delay ATPG is performed. Finally, the robustly testable paths are extracted. This will yield in total four files containing paths (one file for each core).

In the last stage, the final test set is created. In this stage, it is checked which paths have a test for each core. If this is the case, The path is added to the final test set. Table 4.3 shows an example of when a path is added to the final test set. Paths are indicated with a number. In the implementation paths with the same begin- and endpoint share the same number between the cores. It can be seen that only paths 2 and 3 have a test for every core. Thus, only these two paths are added to the test set.

Table 4.3: Example of when a path is added to the final test set. Only paths that have a test for every core are added to the test set.

	Path Testable?				
Path	**Core 1**	**Core 2**	**Core 3**	**Core 4**	**Test added**
Path 1	Yes	Yes	No	Yes	No
Path 2	Yes	Yes	Yes	Yes	Yes
Path 3	Yes	Yes	Yes	Yes	Yes
Path 4	No	Yes	No	No	No

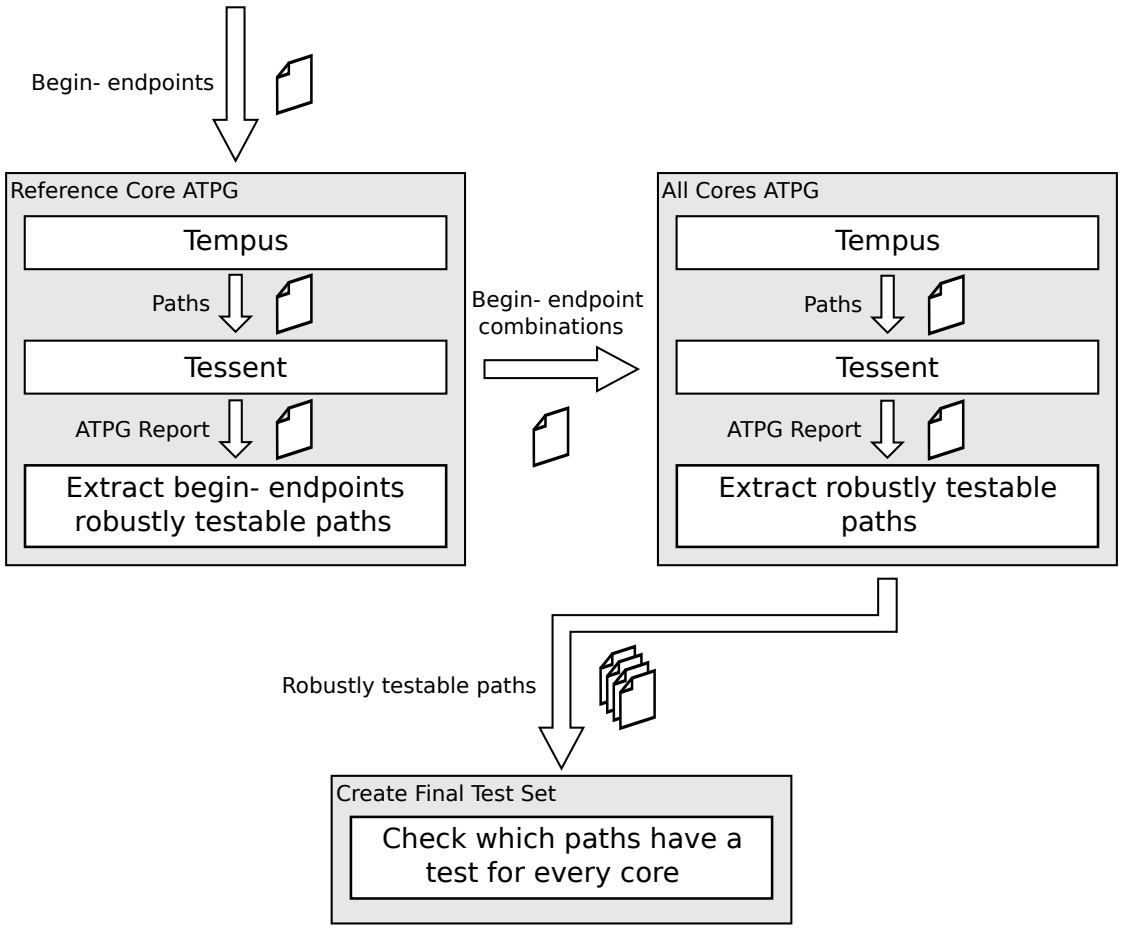


Figure 4.2: Flowchart of the Path Delay ATPG process

Optimizations

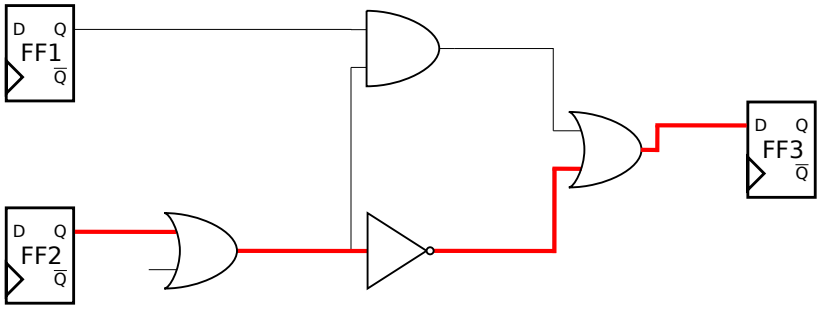
The target objective is to have in the order of 100 paths for delay measruements. In general, this is not an issue. However, the following constraints exist:

- The start and end point of the path needs to be a scan flip-flop. Only the scan flip-flops can be controlled and observed, which is required for path delay measurements. Since the Beetle is a research chip, only half of the flip-flops are scannable. This affects the testability.
- The path needs to be robustly testable. A robust test has the most constraints when it comes to the state of side inputs of the paths. Therefore, it is often not possible to generate a robust test for a path.
- For each core a robustly testable path needs to exist between the same begin and endregisters. Otherwise, the path is discarded.

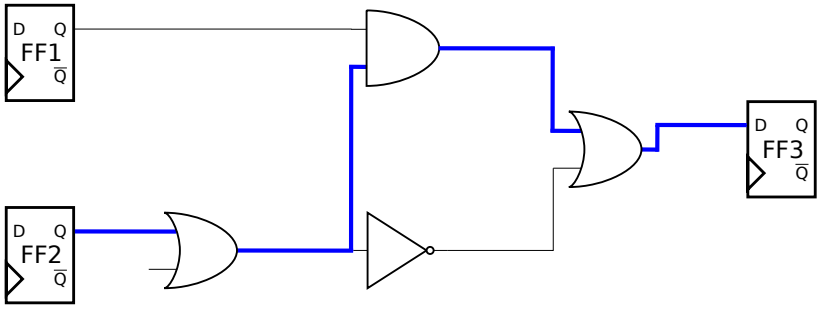
To efficiently generate suited paths the following optimizations are done:

- For the path generation only paths with scan flip-flops as begin- and endpoint are taken. Paths without this requirement, cannot be observed nor controlled, hence it is impossible to generate a test for them. Therefore, such paths are discarded before performing ATPG. This optimization is done by supplying the list of scan flip-flops as the list of begin- and endpoints for the first stage of the path delay ATPG flow (Figure 4.2).

- In real circuits there are usually multiple paths between a begin- and endregister. This is illustrated in Figure 4.3. Normally, Tempus will only generate one path per begin- and endregister. In that case, it always picks the path with the least amount of slack. It is possible to force Tempus to generate multiple paths between the same begin- and endregister. This is done by using the *nworst* option when performing STA. Tempus will then generate multiple paths between the same begin- and endregister sorted on increasing slack. When there are multiple paths with the same begin- and endregister, the chance is bigger that Tessent is able to generate a robust path delay test for one of them. When multiple of these paths with the same begin- and endregister are covered, the one with the lowest slack is selected and the other paths are discarded.



(a) A possible path between beginregister FF2 and endregister FF3.



(b) Another path between beginregister FF2 and endregister FF3.

Figure 4.3: Multiple paths exist between beginregister FF2 and endregister FF3.

- An opposite transition is included for every generated path. When Tempus generates a timing report for a path, it includes per node whether a falling or a rising transition occurs. An opposite transition is also possible on the path. Opposite

means that a rising transition becomes a falling transition and vice versa. Paths that are generated by Tempus are duplicated and all transitions are inverted. By doing this, the set of input paths for Tessent becomes twice as big and therefore there will be more paths for which a robust test can be created. This optimization is done in the second stage in the path delay ATPG flow (Figure 4.2). For all paths generated by Tempus in the second stage, an opposite transition is included as well.

Path Delay Test Set Evaluation Criteria
The following criteria exist for the path delay test set:

- The path delay test set will be used to characterize the silicon of the chips. This is done by measuring the speed of paths. This means paths of varying lengths should be included: long paths, medium length paths, and short paths.
- It is interesting to get an indication of the maximum clock speed of the cores. This can be observed by measuring the speed of the critical path. It may not be possible to measure the actual critical path. Firstly, because not every flip-flop in the design is a scan flip-flop and also because creating a robust test is not always possible. Therefore, a tradeoff would be to include paths with a speed that is close to that of the critical path.
- There is a restriction to the allowed delay of the paths. The reason for this is that the clock and the clock network are designed for a maximum frequency of 500 MHz. Because of this, it is possible that paths with a delay of less than 2 ns cannot be measured, because of a failing clock network. Therefore, it is preferred that the majority of paths is slower than 500 MHz.
- The test set is meant to be used on a lot of samples, including different corner lots. Furthermore, the test set will be used at different temperatures, power supply voltages, and body biasing voltages. Also, the test set needs to be run for every core. Therefore, a number of around 100 paths is wanted (each core 100 paths), so the test time is kept low.

4.1.3 Performed Experiments

With the path delay test set, both simulations and measurements are performed. These tests are used to evaluate the performance of each core. Furthermore, the effect of body biasing on path propagation time is investigated and whether it can be used to compensate process variation.

Simulations
Analog simulations are performed using SPICE decks[41]. The SPICE decks allow for a more accurate simulation than STA. Furthermore, they allow one to adjust voltages for V_{DD} and body biasing.

The SPICE decks are generated with Tempus. SPICE decks for the typical, slow, and fast corners are created for every path. The SPICE decks contain parasitics based on

actual layout, such as for instance cross capacitance between neighbouring interconnects. A SPICE deck for a path contains the gates of the path itself. Furthermore, it can be selected how many levels of gates connected to the off-path inputs are added to the SPICE deck. For the simulations a level of one is chosen. The SPICE decks contain stimuli that will introduce the same transitions along the path as with the generated path delay test. Using *TRIGGER* statements the propagation delay of the path is measured, which is written to a log file. The SPICE decks are simulated using **Spectre**[42].

The following simulations are performed:

- Transient analysis for all paths under varying conditions such as corner, power supply voltage, and body bias voltages.
- Monte Carlo simulations with local and global variation for a long path.

With the transient analysis a path is simulated once. With Monte Carlo simulations the circuit is simulated multiple times, each time with slightly different process parameters ($V_t, L, W, ...$) for each transistor. These variations reflect the actual veriations that occur during the manufacturing process. These simulations will show how process variation affects the propagation delay of the path. Monte Carlo simulations are very time consuming, so they are only performed for one path for both cores.

The specifications of a design are determined by its performance under the worst-case manufacturing and operating conditions. Therefore, the simulations are performed for slow processing, the minimum supply voltage, and the temperature at which the circuit is the slowest. Core 1 is a conventional worst-case based design, meaning it should still meet the target frequency. Therefore, it is interesting to compare the performance of core 1 with cores 2, 3, and 4. The other three cores need to be able to meet the same performance as core 1. However, these cores are designed with far less margin, so the VRSs need to be applied, such as body biasing, to achieve this. In the simulations (and in the measurements) it will be investigated if with forward body biasing the other cores are able to achieve a similar performance as the reference performance of core 1.

Measurements

Standard Test Interface Language (STIL)[43] files are used to apply the path delay test vectors to the Beetle platform. These STIL files are generated using Tessent. The STIL files are run on the Beetle platform using a custom developed LabVIEW[44] program that parses the STIL files and applies the stimuli to the Beetle cores and observes the output signals. One STIL file contains several test vectors. The LabVIEW program reports per vector, whether it passed or failed. Multiple test vectors can be combined, which is beneficial for test time during production testing, however, in this case, it was ensured that each path has a separate vector. This way, it is clear which paths failed by looking at which test vectors failed.

With the measurements the speed of paths will be measured. In order to do this the the first frequency at which the path fails, needs to be found. This means the tests need to be run at a low frequency first and then rerun each time with increasing frequency. The cores on the Beetle all contain a Digitally Controlled Oscillator (DCO) that can be programmed through JTAG. One of the first things the STIL file will do is initialize

certain registers in the cores through JTAG. The DCO registers are also programmed during this procedure. One of the registers that is programmed is the register that controls the output frequency that the DCO needs to make the Beetle core run at. Thus, the frequency sweep can be performed by creating multiple STIL files that each program the DCO for a different output frequency.

The following measurements are performed:

- Path delay measurements for multiple samples with typical processing for normal body biasing and 0.1 V, 0.2 V, and 0.3 V forward body biasing.
- Path delay measurements for a sample with slow processing for normal body biasing and 0.1 V, 0.2 V, and 0.3 V forward body biasing.

4.2 Path Delay Test Generation Results

Core 2 is used as the *reference core* for the path delay test generation. The motivation for this is that it is completely designed at 1 sigma. The STA tool is focused on finding critical paths. As core 2 is designed completely at 1 sigma, the STA tool will find paths spread over the whole design.

In Table 4.4 the frequencies of the critical paths are listed for each core. In Figure 4.4 histograms are shown of the frequencies of the paths in the test set for each core. Table 4.5 summarizes the results of Figure 4.4. The test set contains 150 paths, which is well above the target of around 100 paths.

The values for the frequencies of the paths are based on the STA results from Tempus, using tt corner models with V_{DD} at 1.1 V. As discussed, the clock network in the Beetle cores is designed for frequencies up to 500 MHz. It can be seen that for all cores the average frequency of the paths is well below this number. Furthermore, when looking at the histograms, a vast majority of paths has a frequency below 500 MHz. Thus, only for a few paths it needs to be taken into account that it may not be possible to measure them correctly. It can be seen that for core 2 the average frequency of the paths is a lot lower than that of the other cores. Therefore, the effect of the reduced margin of core 2 is well visible.

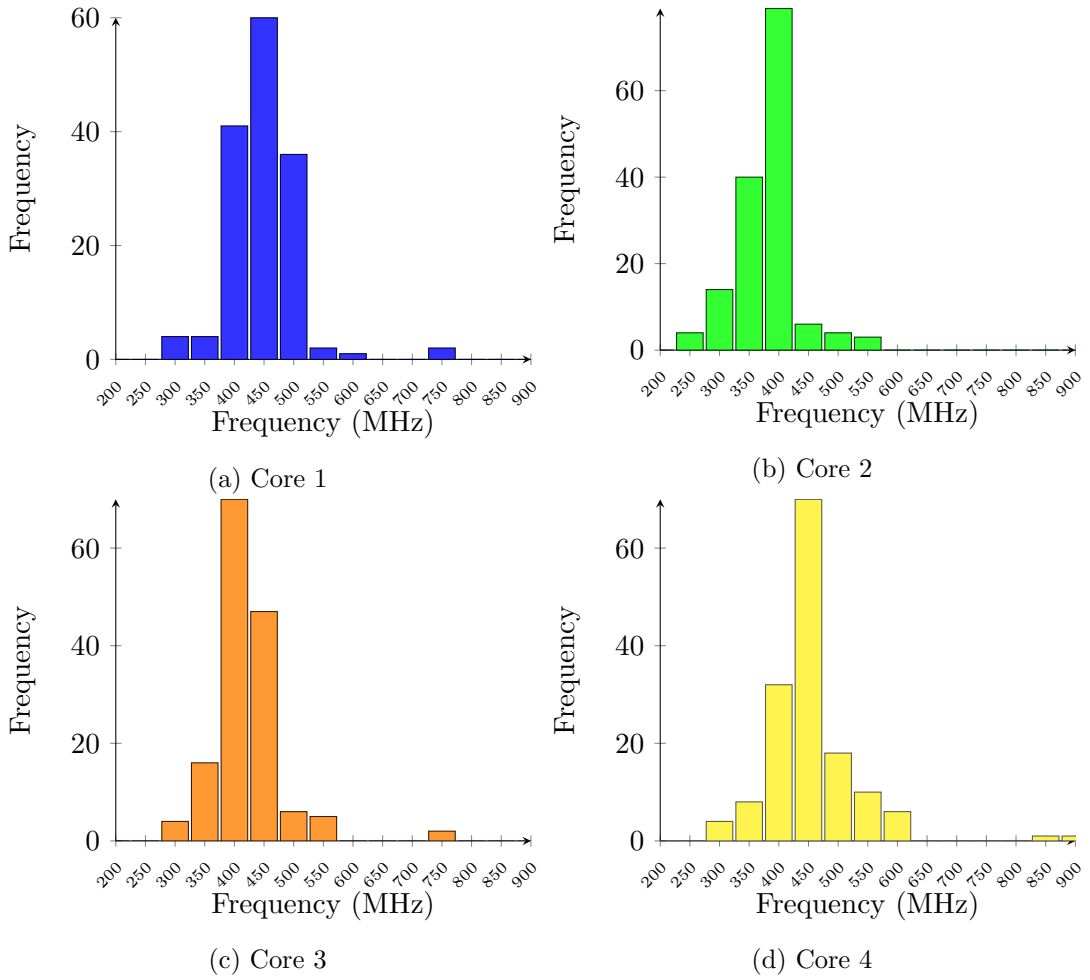
When comparing the critical paths with the minimum frequency in the test set, it can be seen that for cores 1 and 2 the minimum frequencies are very close to the frequency of the critical path. For cores 3 and 4 this is not the case. The minimum frequency in the test set for these cores is close to that of core 1. This can be explained by the fact that the parts of cores 3 and 4 that have been designed with a BTWC approach, do not have any scan flip-flops. Therefore, only 3 sigma parts can be tested with path delay testing for cores 3 and 4. Considering core 1 has been designed at 3 sigma, performing the experiments with cores 3 and 4 will not give any new information, as only 3 sigma parts can be measured for these cores. Therefore, the experiments (simulations and measurements) will *only* be performed for cores 1 and 2. With this a comparison between the performance of 3 sigma versus 1 sigma design can be made.

Table 4.4: Critical paths for each core, based on STA results for tt corner, V_{DD}=1.1 V, T=25 °C.

	Core 1	**Core 2**	**Core 3**	**Core 4**
$Frequency_{critical\_path}$ (MHz)	277	209	235	239

Table 4.5: Final path delay test set specifications, based on STA results for tt corner, V_{DD}=1.1 V, T=25 °C.

	Core 1	**Core 2**	**Core 3**	**Core 4**
Number of paths		150		
$Frequency_{average}$ (MHz)	416	349	389	417
$Frequency_{min}$ (MHz)	286	211	285	290
$Frequency_{max}$ (MHz)	732	515	740	864



(a) Core 1

(b) Core 2

(c) Core 3

(d) Core 4

Figure 4.4: Histograms of frequencies of the paths in the test set for each core. The frequencies are based on STA for tt corner, V_{DD}=1.1 V, T=25 °C.

4.3 Simulation Results

This section describes the simulation results. As discussed, transient analysis and Monte Carlo simulations are performed.

4.3.1 Transient Analysis

Transient simulations are performed for all paths. Figures 4.5 and 4.6 show the path speeds of cores 1 and 2 under typical conditions. For both cores 1 and 2 the results without body biasing and with 0.3 V forward body biasing are displayed. The paths are ordered based on ascending speed for the 'normal biasing' case of core 1. The paths for core 2 have been ordered the same way. Therefore, path 1 for core 1 corresponds (same begin- and endregister) to path 1 for core 2, path 2 for core 1 corresponds with path 2 for core 2, etc. Because of this correspondence, core 2's graphs do not show the same smooth increasing frequency as those of core 1. A linear trend can still be recognized, however: paths tend to get faster as their number increases. It can be seen for both cores 1 and 2 that all paths get a very consistent and predictable speedup with forward body biasing. Table 4.6 lists the average frequencies of paths for both cores under typical conditions. The averages are shown for the first 4 paths (1-4) and for all 150 paths (1-150). The average of the first 4 paths is interesting, because these paths are very close to the critical paths of the designs and, therefore, give a good indication of the maximum speed.

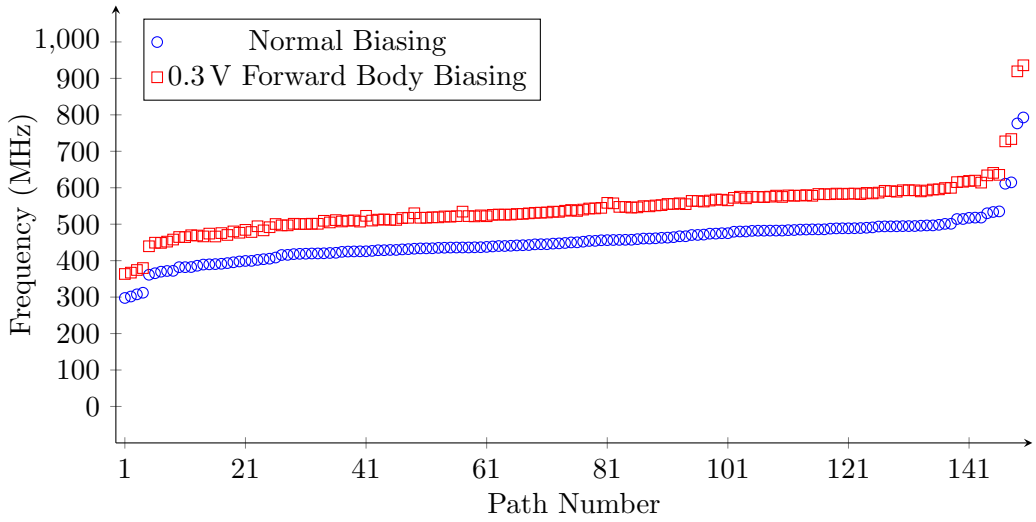


Figure 4.5: Simulated path frequencies for core 1 under typical conditions: tt corner, V_{DD}=1.1 V, T=25 °C

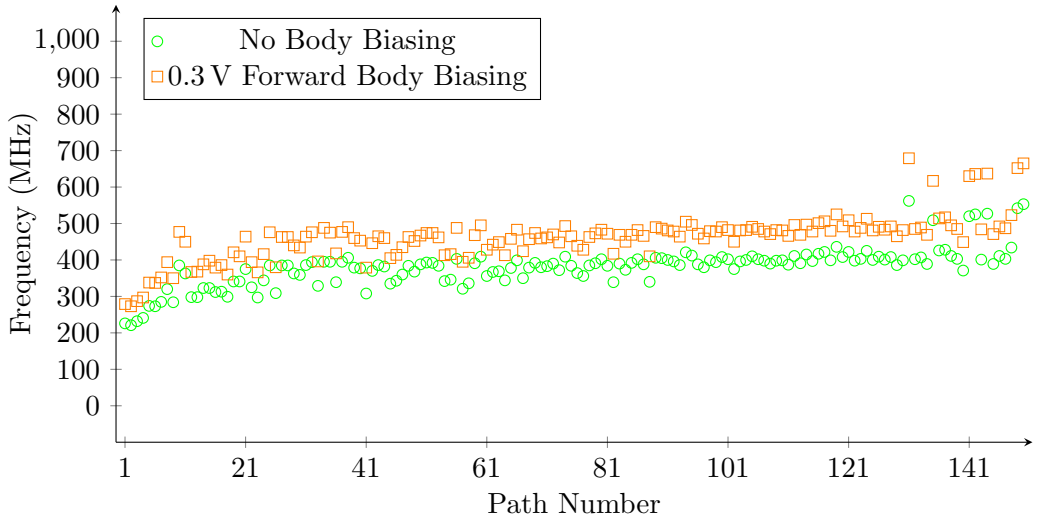


Figure 4.6: Simulated path frequencies for core 2 under typical conditions: tt corner, V_{DD}=1.1 V, T=25 °C

Table 4.6: Average frequencies of simulated paths for cores 1 and 2 under typical conditions: tt corner, V_{DD}=1.1 V, T=25 °C

Core	**Paths**	**Average Frequency** (MHz)			
		Forward Body Biasing			
		0.0 V	0.1 V	0.2 V	0.3 V
Core 1	1-4	305	324	346	371
	1-150	454	481	510	544
Core 2	1-4	230	246	264	284
	1-150	381	404	431	460

In Figures 4.7 and 4.8 the results for cores 1 and 2 for typical conditions are plotted in the same graph. Figure 4.7 shows a comparison without body biasing. It can be seen that almost all paths from core 1 are faster. This is in line with the results in Table 4.6. When normal biasing is applied, the average frequency of all paths is 454 MHz for core 1 and 381 MHz for core 2. In Figure 4.8 core 2 has 0.3 V forward body biasing. When looking at Table 4.6, the paths of core 2 now have an average frequency of 460 MHz, which is slightly higher than core 1 with normal biasing. When looking at the average of the first 4 paths, however, it is only 284 MHz for core 2 and 305 MHz for core 1. Therefore, it can be expected that core 2 will still be slightly slower with 0.3 V forward body biasing under typical conditions.

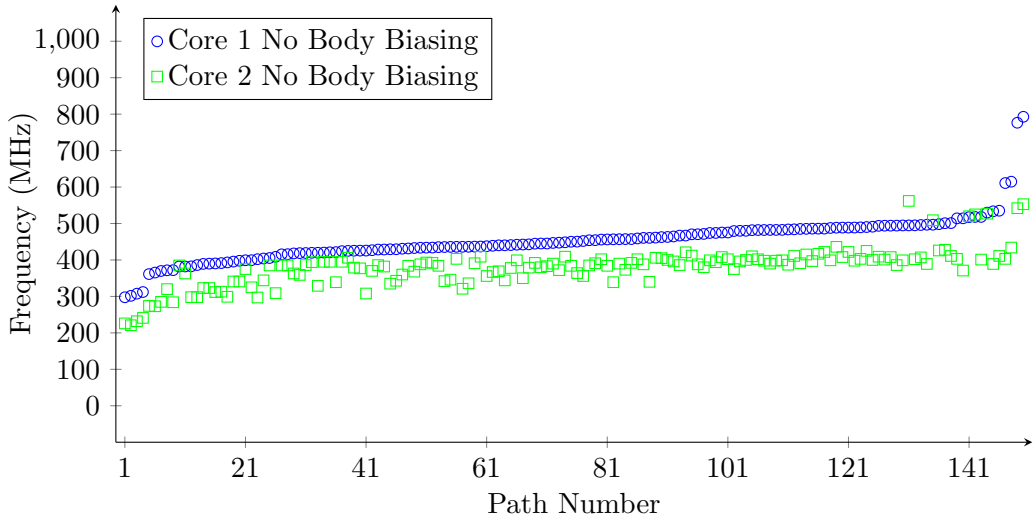


Figure 4.7: Comparison of simulated path frequencies for cores 1 and 2 under typical conditions: tt corner, V_{DD}=1.1 V, T=25 °C

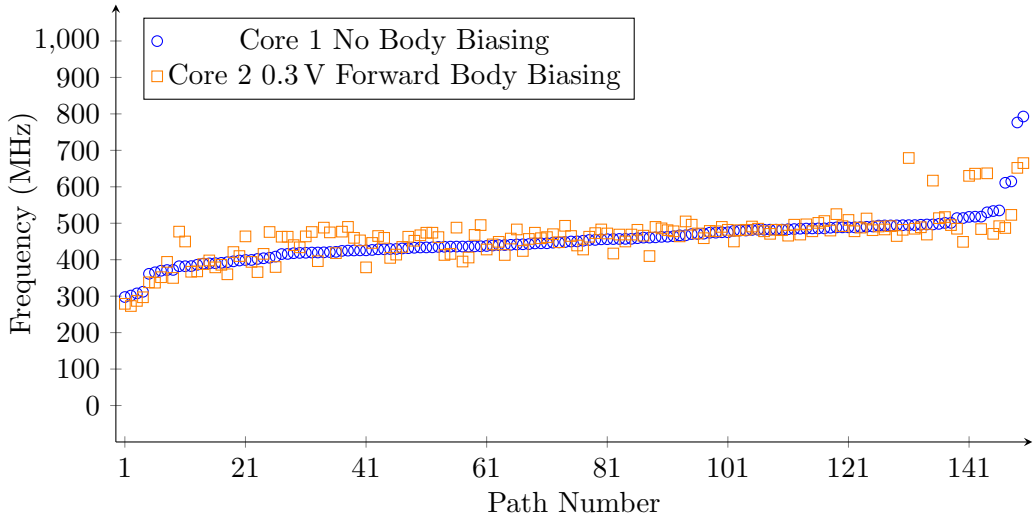


Figure 4.8: Comparison of simulated path frequencies for cores 1 and 2 under typical conditions: tt corner, V_{DD}=1.1 V, T=25 °C. Core 2 has 0.3 V forward body biasing.

The Beetle cores are designed for temperatures between −40 °C and 125 °C. During synthesis the assumption was made that the circuit runs slower at higher temperatures. Because of this assumption, it was checked if the design would meet the required speed under worst-case conditions using an STA library that models the propagation delay of gates at 125 °C. Figures 4.9 and 4.10 show the results for the simulations under worst-case conditions. The slow corner model is used for the paths, V_{DD} is set to the lowest specification, and the temperature is set to 125 °C. Table 4.7 lists the average frequencies of paths for both cores under these worst-case conditions. In Figure 4.9 core 1 and core

2 are both compared without body biasing. It can be seen that for both cores 1 and 2 the performance drops compared to the simulations under typical conditions (Figure 4.7) Mainly for core 1 there is a big drop in performance. This happens, because V_{DD} makes a significantly higher drop for core 1 than for core 2. For core 1 the minimum V_{DD} specification is 0.99 V, as a 10% variation is allowed from the nominal V_{DD} of 1.1 V. Core 2, however, has an allowed variation of 5%, so it is set to 1.045 V. The performance for core 2 is still very close to the performance under typical conditions. One would expect a bigger drop in performance, as the operating voltage has been lowered by 5% and the slow corner is simulated. There is a simple explanation why the degradation in performance is so small: during synthesis the worst-case temperature has been chosen incorrectly. For old technology a higher temperature means the circuit becomes slower. With modern technology the circuit actually becomes faster. This is due to an effect called *temperature inversion*[45]. Therefore, for worst-case signoff the used temperature should have been −40 °C. When looking at Table 4.7, it can be seen that for normal biasing the average frequency of all paths is 293 MHz for both core 1 and core 2. The average frequency of the first 4 paths is lower for core 2 than for core 1 (186 MHz versus 208 MHz). Therefore, under worst-case conditions with normal biasing the performance of core 2 is lower. This is in line with the expectations, since core 2 is designed with less voltage margin and under the assumption of less process variation. Based on simulations it is determined that 0.1 V forward body biasing makes the performance comparable with core 1, as shown in Figure 4.10.

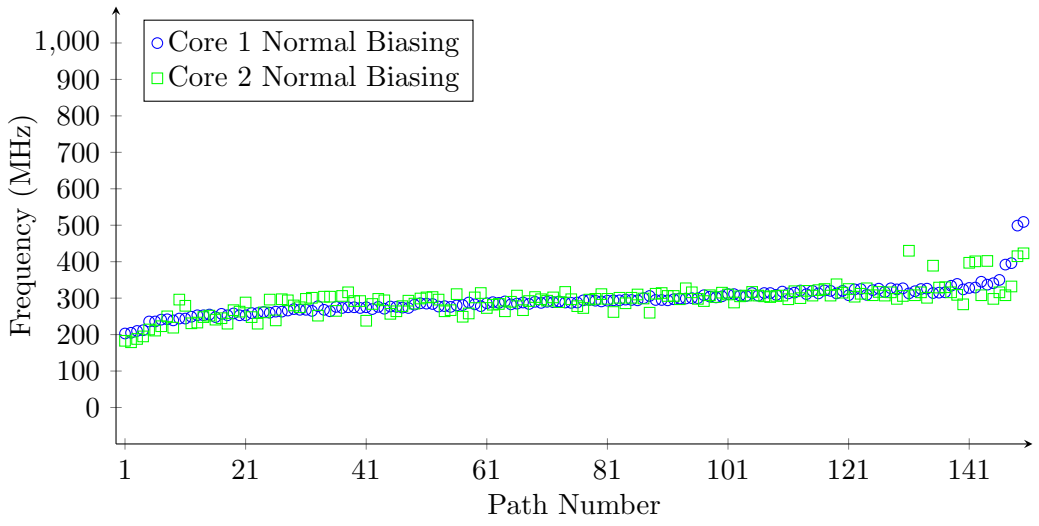


Figure 4.9: Comparison of simulated path frequencies for cores 1 and 2 under worst-case conditions: ss corner, V_{DD} at minimum specification of the specific core, T=125 °C

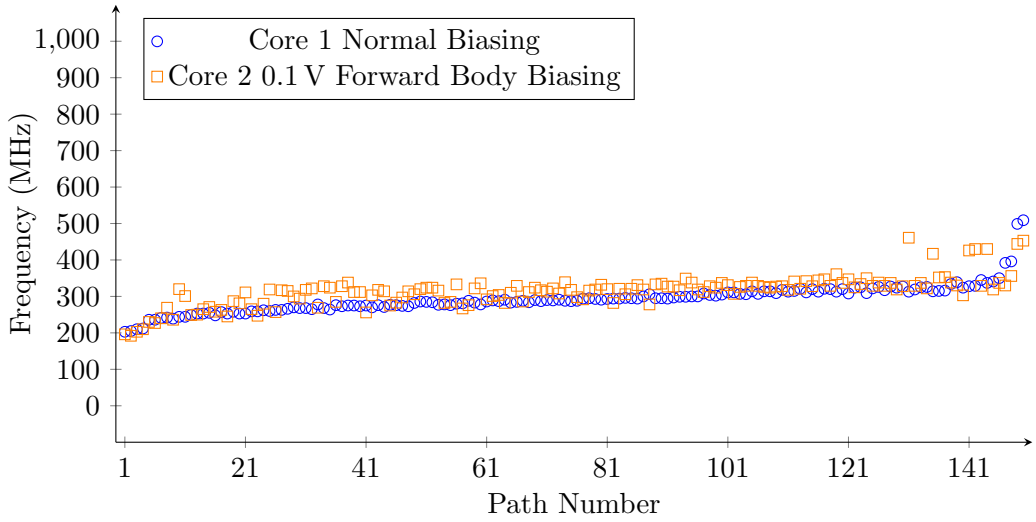


Figure 4.10: Comparison of simulated path frequencies for cores 1 and 2 under worst-case conditions: ss corner, V_{DD} at minimum specification of the specific core, T=125 °C Core 2 has 0.1 V forward body biasing.

Table 4.7: Average frequencies of simulated paths for cores 1 and 2 under worst-case conditions: ss corner, V_{DD} at minimum specification, T=125 °C

Core	**Paths**	**Average Frequency** (MHz)			
		Forward Body Biasing			
		0.0 V	0.1 V	0.2 V	0.3 V
Core 1	1-4	208	224	242	262
	1-150	293	316	341	369
Core 2	1-4	186	200	216	233
	1-150	293	314	337	362

Figure 4.11 shows the results for the simulations with both cores having normal biasing under worst-case conditions, but then for a temperature of −40 °C. Table 4.8 lists the average frequencies of paths under these conditions. When comparing the average frequencies with the results of the simulations at a temperature of 125 °C, it can be seen that there is a big drop in performance. For instance, with normal biasing, the average frequency of all paths is only 200 MHz for core 1 at −40 °C, where it is 293 MHz at 125 °C. Furthermore, the first four paths of core 1 now have an average frequency of 143 MHz, which is also a lot lower than the signoff of 180 MHz. This shows that the assumption that the circuit would run slowest at high temperatures was incorrect and that −40 °C should have been chosen. What is interesting, is that core 2 is faster than core 1 under these conditions. Both the average frequency of all paths is higher (200 MHz for core 1, 231 MHz for core 2), as well as the average frequency of the first 4 paths (143 MHz for core 1, 147 MHz for core 2).

All results from the transient simulations can be found in Appendix C.

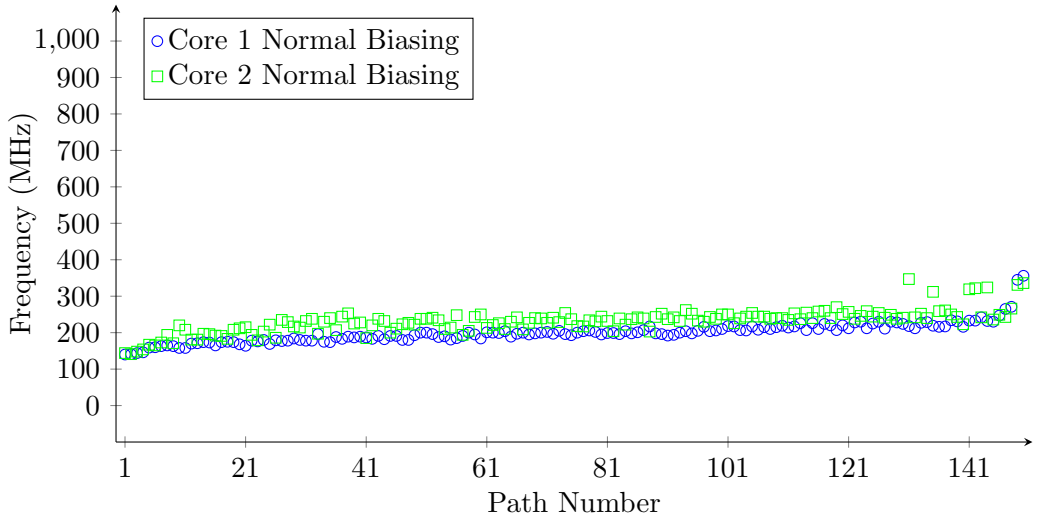


Figure 4.11: Comparison of simulated path frequencies for cores 1 and 2 under worst-case conditions: ss corner, V_{DD} at minimum specification of the specific core, T=−40 °C

Table 4.8: Average frequencies of simulated paths for cores 1 and 2 under worst-case conditions: ss corner, V_{DD} at minimum specification, T=−40 °C

Core	**Paths**	**Average Frequency** (MHz)			
		Forward Body Biasing			
		0.0 V	0.1 V	0.2 V	0.3 V
Core 1	1-4	143	160	179	202
	1-150	200	224	252	283
Core 2	1-4	147	162	179	199
	1-150	231	254	280	309

4.3.2 Monte Carlo Simulations

For each combiation of applied parameters (V_{DD} and temperature) 500 Monte Carlo runs have been performed. Figure 4.12 shows the histogram of simulated speeds for path 1 at a nominal V_{DD} of 1.1 V and a temperature of 25 °C. Path 1 is a path that is very close to the actual critical path for both cores. It can be seen that the performance of path 1 for core 2 is quite a bit lower. The slowest samples of core 2 have a speed of around 200 MHz, where the slowest samples of core 1 have a speed of around 260 MHz. Figure 4.13 shows the histogram for the same conditions, except that for core 2 now 0.3 V forward body biasing is applied. Just like with the transient simulations the performance of cores 1 and 2 is now comparable. Core 2 does have a lot more samples in the lower corner, so it can be expected that the worst-case performance of core 2 is still slightly lower than for core 1.

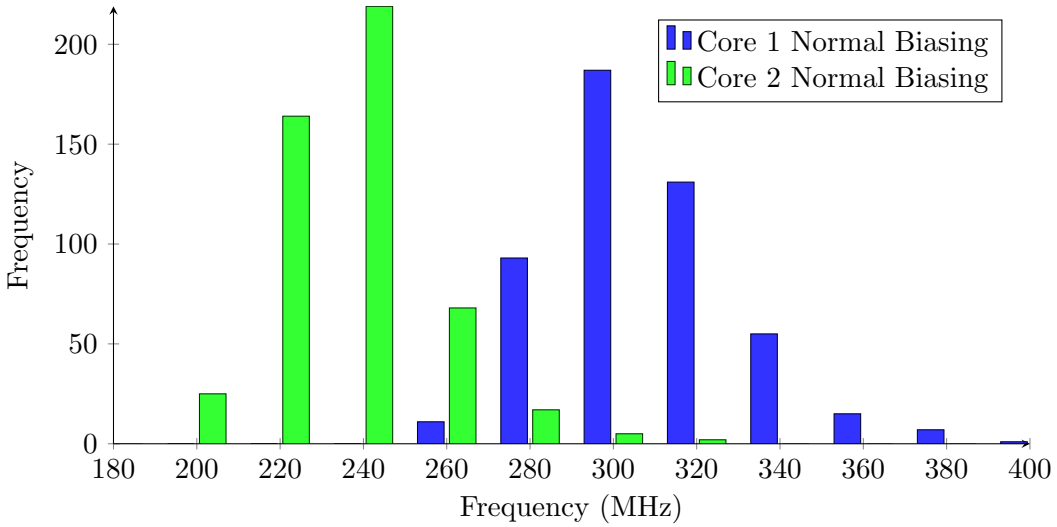


Figure 4.12: Path 1 Monte Carlo simulation results for cores 1 and 2 with T=25 °C, V_{DD}=1.1 V.

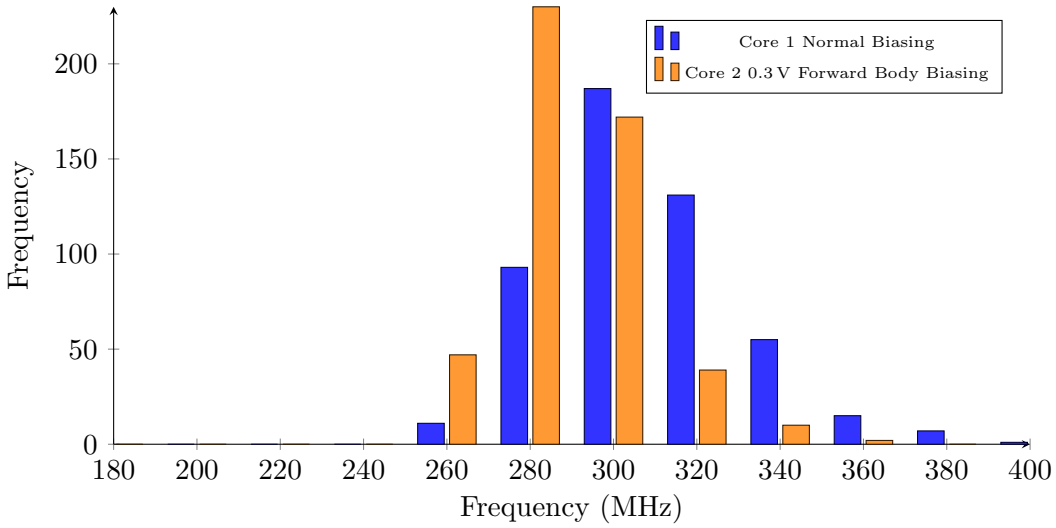


Figure 4.13: Path 1 Monte Carlo simulation results for cores 1 and 2 with T=25 °C, V_{DD}=1.1 V. Core 2 has 0.3 V forward body biasing.

Figure 4.14 shows the histogram of simulated speeds for path 1 at the minimum V_{DD} specification (core 1 at 0.99 V, core 2 at 1.045 V) and a temperature of 125 °C. The temperature of 125 °C was chosen as the temperature at which the silicon performs slowest during design. As already discussed, this assumption is incorrect. Running the simulations at this temperature, however, does show best how the reduced margin of core 2 affects its performance, since the signoff for path propagation delays happened at this temperature. It can be seen that the performance of core 2 is now slightly lower than the performance of core 1, as the slowest samples of core 1 have a speed of around 220 MHz and the slowest samples of core 2 a speed of around 200 MHz. Figure 4.15 depicts the simulations with core 2 having 0.2 V forward body biasing. It can be seen that the slow samples of core 2 are now around 220 MHz and there are less samples in this bin than for core 1. Thus, the performance of core 2 is now slightly better than that of core 1.

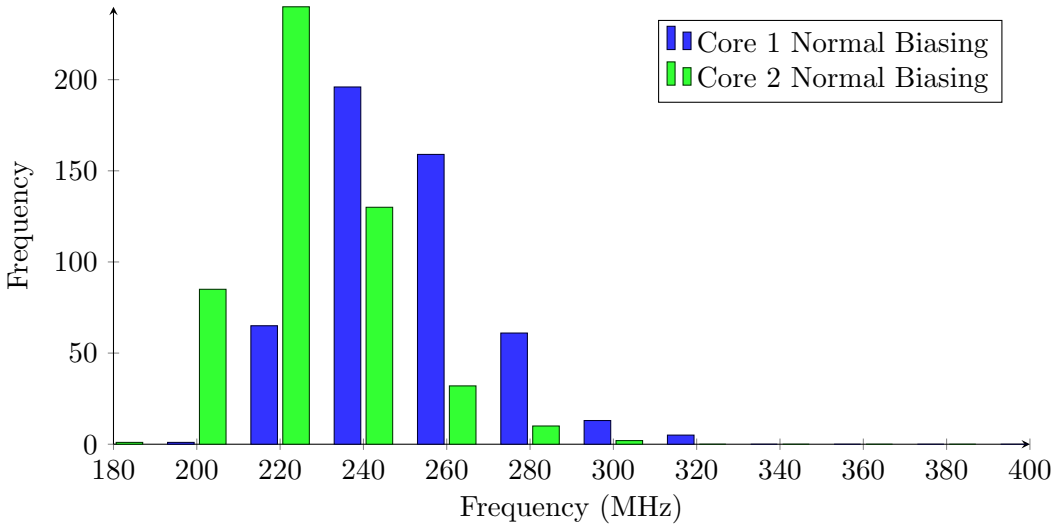


Figure 4.14: Path 1 Monte Carlo simulations for cores 1 and 2 with T=125 °C, V_{DD} at minimum specification of the specific core.

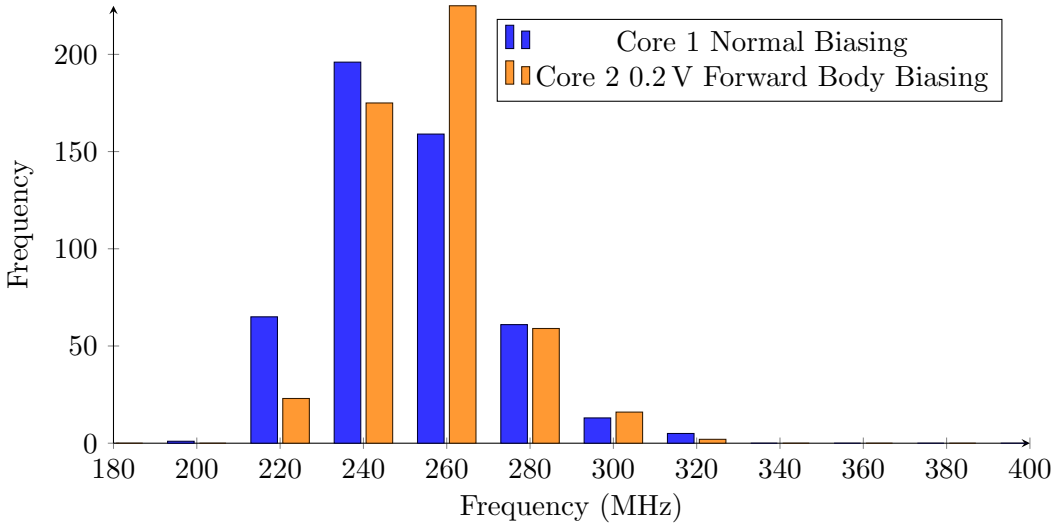


Figure 4.15: Path 1 Monte Carlo simulations for cores 1 and 2 with T=125 °C, V_{DD} at minimum specification of the specific core. Core 2 has 0.2 V forward body biasing.

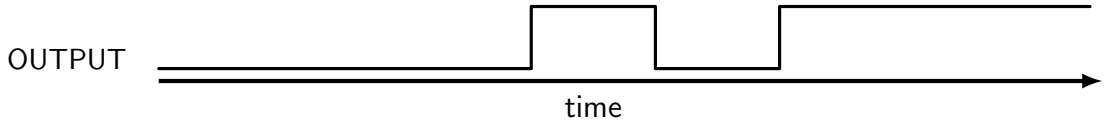
4.4 Measurement Results

This section describes the measurement results. First of all, some difficulties were encountered with the measurements, which is dicussed first. After this, the typical corner measurement results are discussed and finally the slow corner measurement results.

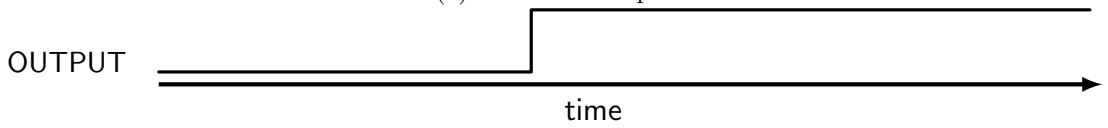
4.4.1 Measurement Difficulties

During the measurements unexpected behavior was observed for a lot of paths: a lot of paths did not show the expected robust behavior. Figure 4.16 shows both an example waveform of the measured and the expected path output. In the example of the measured waveform it can be seen that there are glitches in the output. This is not what is expected for robust tests. Instead, there should only be one transition at the output of the path and after this the output should remain stable, as shown in the expected waveform.

When looking at the definition that Mentor Graphics gives for robust tests, the glitches still cannot be explained: according to Mentor Graphics all off-path inputs are stable and only the on-path inputs make a transition. Therefore, one would expect no glitches at the output of the path.



(a) Measured output

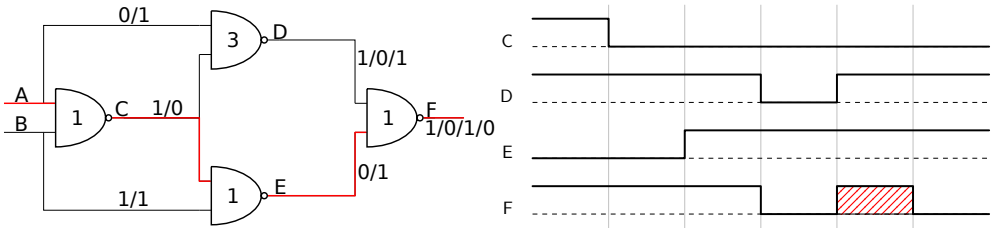


(b) Expected output

Figure 4.16: Waveform showing an example of the unexpected glitches at the output observed for a lot of paths during measurements. In this example the path has a rising transition at the output.

The most likely explanation for the glitches in the path measurements is that glitches occur at the off-path inputs. Mentor Graphics does not analyze, whether glitches occur. Instead, it only checks the final state of the off-path inputs after the transition is launched on the path. An example of a hazard is given in Figure 4.17. The hazard shown in this example depends on the timing relation of the two input signals that control output D. This timing relation is affected by process, temperature and supply voltages, hence depending on the conditions the hazard might occur or not.

Due to the glitches at the output, some path measurements will give an output similar to the example shown in Table 4.9. As can be seen at a measurement frequency of 140 MHz the test passes, indicating that the path has finished propagating. At 160 MHz the test fails, indicating that the path did not have enough time to fully propagate. At 240 MHz and 260 MHz the test passes again, however. Because of this, the maximum measured frequency of the path could be 140 MHz or 260 MHz. Using the simulation



(a) Circuit (b) Waveform

Figure 4.17: Example of a transition on a path with glitches on off-path inputs that control the output of the path. The gate delay is marked inside the gate. The path that the transition passes through is highlighted. On A a $0 \rightarrow 1$ transition is created. This will create a $1 \rightarrow 0$ transition on the output F. On D, however, a hazard occurs, pulling output F high for a short time.

results, it is possible to choose between these, by selecting the frequency that is closest to the frequency from simulation. It may also be possible that the speed of the output path cannot be measured reliably at all. This is the case when the path's transition is masked by a glitch. As a result, some paths cannot be measured at all and need to be discarded. Most of these paths will have a list of possible measured frequencies of which none is close to the simulated frequency. Therefore, by making sure that the selected measured frequency is close enough to the simulated frequency and otherwise discarding the path, most paths that cannot be measured due to masking are removed.

A Python[46] script is written to process the measurements in the way that is mentioned above. The script reads in the measurement data and the simulated frequencies. It will then identify for every path which frequencies are possible. For Table 4.9 these frequencies would be 140 MHz and 260 MHz. Next, it will determine which one of these frequencies is closest to the simulated frequency. If this frequency is within a certain range of the simulated frequency, the frequency is stored as the measured frequency of the path. In case the frequency is not within the range, the path will be omitted from the measurements. The code for the script is added in Appendix B.

Table 4.9: Critical paths for each core, based on STA results for tt corner, V_{DD}=1.1 V, T=25°C.

$F_{Measure}$ (MHz)	140	160	180	200	220	240	260	280	300
Result	Pass	Fail	Fail	Fail	Fail	Pass	Pass	Fail	Fail

4.4.2 Typical Corner Results

Measurements are performed for three samples from the same wafer with typical processing. These samples are labeled A010, A028, and A029. All measurements are done for a V_{DD} of 1.0 V instead of the nominal 1.1 V. The voltage is lowered, so the path delay will decrease. Therefore, lower clock speeds are needed to measure path delays, making the chance lower that paths cannot be measured because of a failing clock network.

Table 4.10 lists the average measured frequencies for samples A010, A028, and A029. In Figure 4.18 the measured path frequencies for core 1 of sample A010 with normal body biasing and with 0.3 V forward body biasing are shown. Figure 4.19 shows the measured path frequencies of core 1 for samples A010, A028, and A029 in the same graph. When looking at the average frequencies in Table 4.10, it can be seen that samples A028, and A029 are slightly faster than sample A010. For instance, the average frequency of all paths with normal biasing is 273 MHz for sample A010, 289 MHz for sample A028, and 290 MHz for sample A029. This could be due to within-wafer variation. During measurements, putting new samples into the socket on the measurement setup gave problems from time to time. Most likely due to contact issues between the socket and the sample. So, the difference could also be caused by the measurement setup.

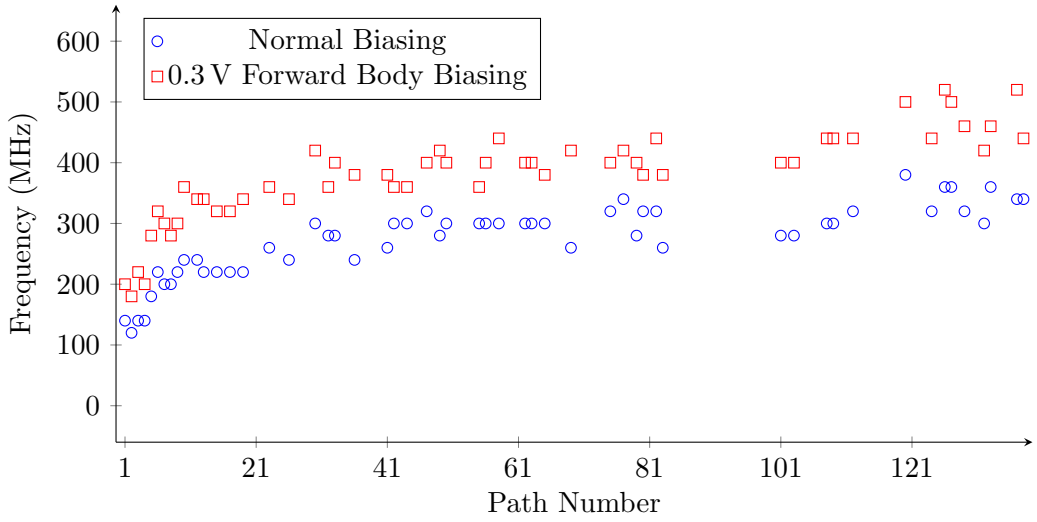


Figure 4.18: Measured path frequencies for core 1 of sample A010 (tt corner). Room temperature, V_{DD}=1.0 V

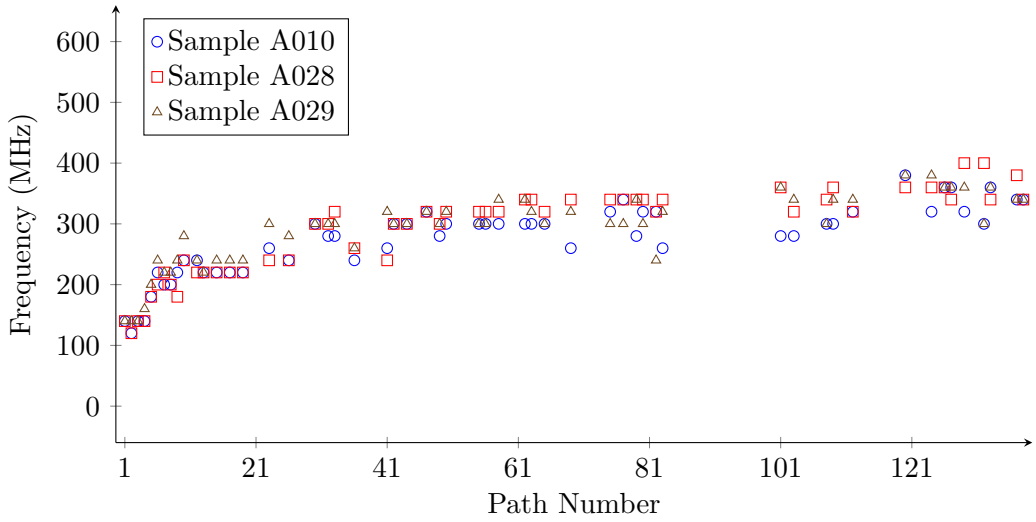


Figure 4.19: Comparison of measured path frequencies for core 1 of samples A010, A028, and A029 (tt corner). Room temperature, V_{DD}=1.0 V

Table 4.10: Average frequencies of measured paths for samples A010, A028, and A029 (tt corner). Room temperature, V_{DD}=1.0 V.

Core	**Sample**	**Paths**	**Average Frequency** (MHz)			
			Forward Body Biasing			
			0.0 V	0.1 V	0.2 V	0.3 V
Core 1	A010	1-4	135	145	165	200
		1-150	273	302	337	379
Core 1	A028	1-4	135	155	185	205
		1-150	289	320	357	394
Core 1	A029	1-4	145	160	185	215
		1-150	290	320	359	399
Core 2	A010	1-4	100	110	125	145
		1-150	183	208	225	250
Core 2	A028	1-4	100	110	125	145
		1-150	183	199	215	247
Core 2	A029	1-4	115	120	145	160
		1-150	195	213	243	269

In Figures 4.20 and 4.21 the measured path frequencies of cores 1 and 2 are shown in the same graph for sample A010. Figure 4.20 shows the case where both cores 1 and 2 without body biasing. When looking at the average frequencies in Table 4.10, it can be seen that the average frequencies of all paths and the first 4 paths is higher for core 1, which is as expected. In Figure 4.21 the path delay frequencies are shown for core 1 without body biasing and for core 2 with 0.3 V forward body biasing. When looking at the average frequencies in Table 4.10, it can be seen that the average frequency of the first 4 paths is higher for core 2 (135 MHz for core 1 and 145 MHz for core 2). However, the average frequency of all paths is lower for core 2 (273 MHz for core 1 and 250 MHz for core 2). This can most likely be explained by the fact that less paths could be measured for core 2. Especially long paths could be measured and, thus, the average frequency becomes lower. An explanation for the fact that mostly long paths can be measured is that hazards are mainly caused by short paths. Therefore, shorter paths are more likely to be masked by hazards.

All measurement results for the samples with typical processing can be found in Appendix D.

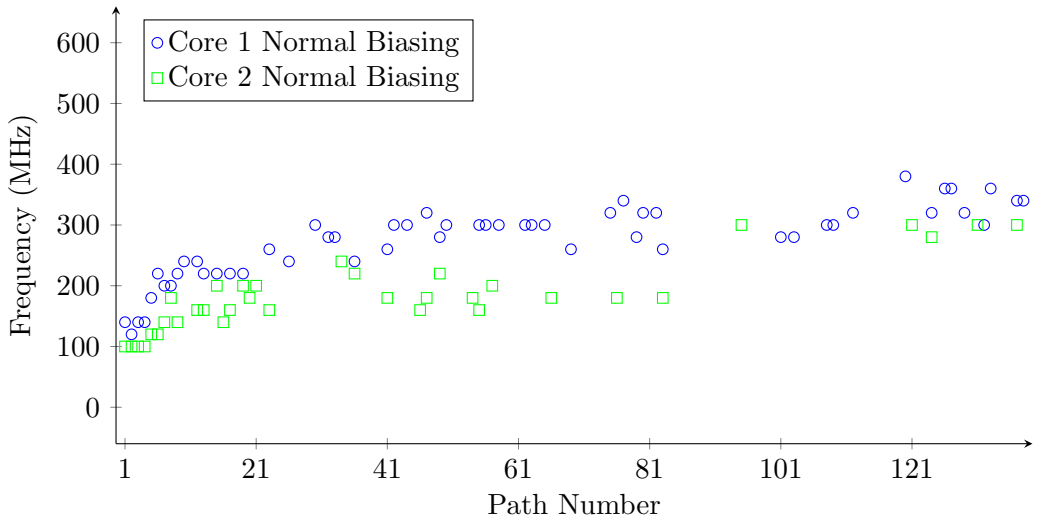


Figure 4.20: Comparison of measured path frequencies for cores 1 and 2 of sample A010 (tt corner). Room temperature, V_{DD}=1.0 V

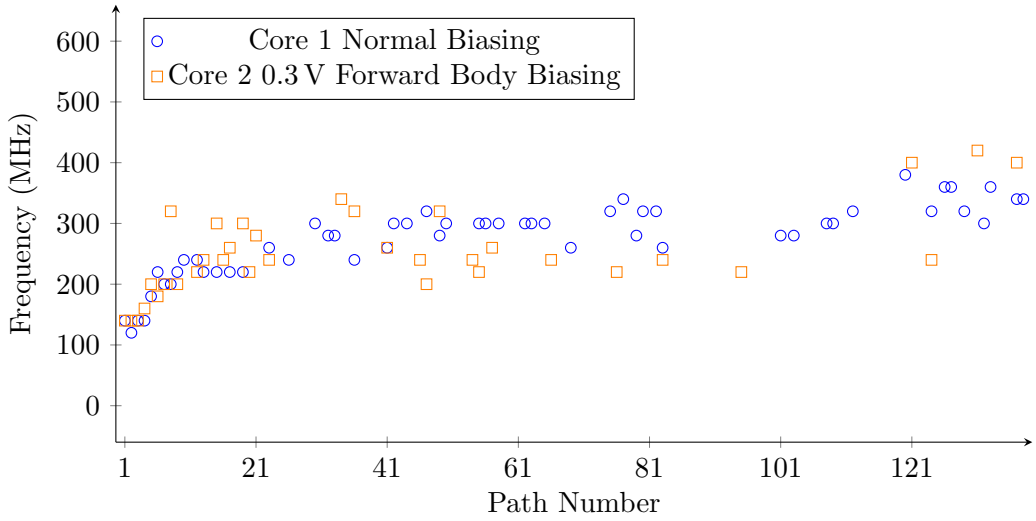


Figure 4.21: Comparison of measured path frequencies for cores 1 and 2 of sample A010 (tt corner). Room temperature, V_{DD}=1.0 V. Core 2 has 0.3 V forward body biasing.

4.4.3 Slow Corner Results

Measurements are performed for one sample with slow processing, which is labeled C001. Table 4.11 contains the average measured frequencies for sample C001. Figure 4.22 shows the measured path frequencies fore core 1 with normal biasing and 0.3 V forward biasing. When comparing this result with the results for sample A010 in Figure 4.18, it can be seen that C001 has a lot more data points, meaning more paths can be measured. This is most likely only the case, because for the slow corner only one sample is measured. For the typical corner three samples are measured and only the paths that can be measured for all samples are shown. If more samples with slow processing are measured, it is likely that a similar amount of paths remains. Figure 4.23 shows the measured path frequencies fore core 2 with normal biasing and 0.3 V forward biasing.

Table 4.11: Average frequencies of measured paths of sample C001 (ss corner). Room temperature, V_{DD}=1.0 V

Core	**Paths**	**Average Frequency** (MHz)			
		Forward Body Biasing			
		0.0 V	0.1 V	0.2 V	0.3 V
Core 1	1-4	95	105	130	150
	1-30	179	207	228	267
	1-150	234	262	294	333
Core 2	1-4	80	95	105	120
	1-30	122	143	163	188
	1-150	152	176	187	213

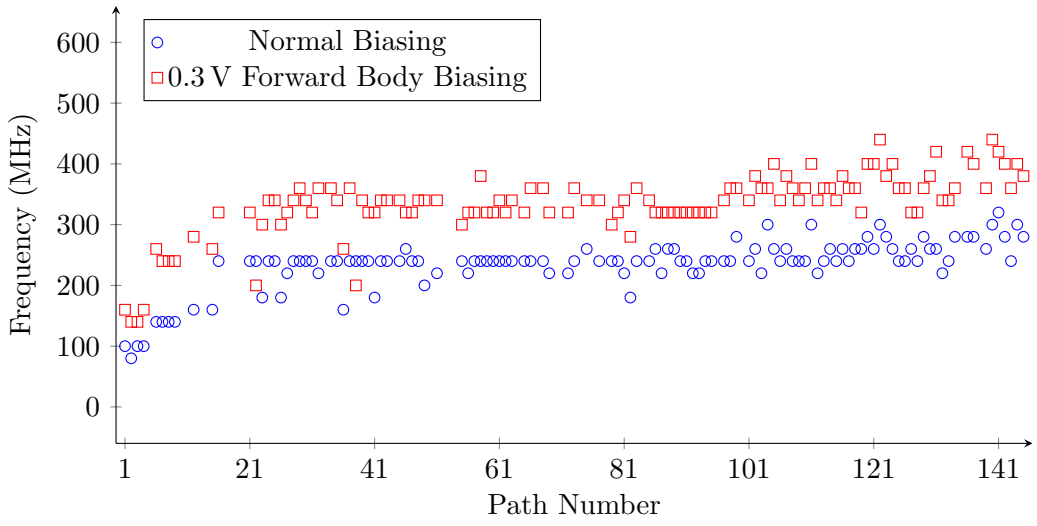


Figure 4.22: Measured path frequencies for core 1 of sample C001 (ss corner). Room temperature, V_{DD}=1.0 V

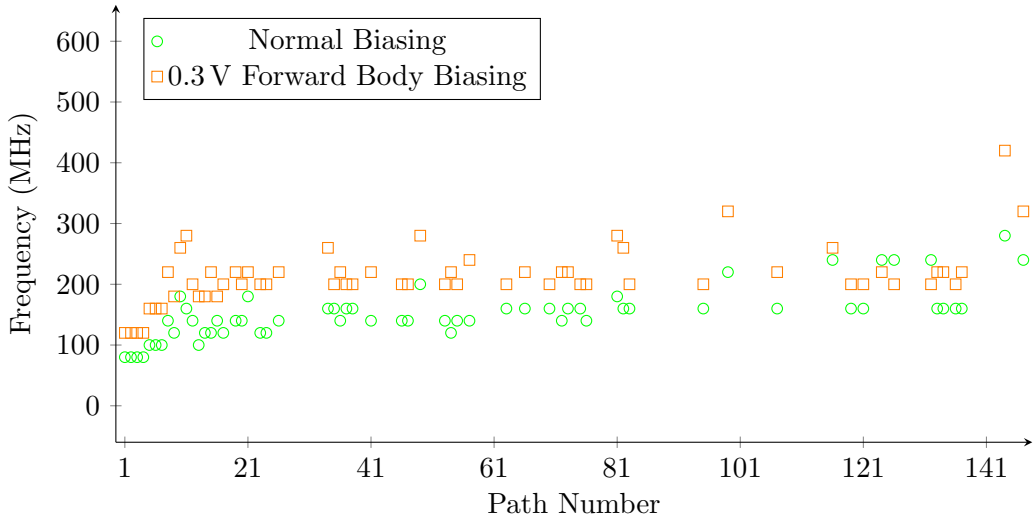


Figure 4.23: Measured path frequencies for core 2 of sample C001 (ss corner). Room temperature, V_{DD}=1.0 V

In Figures 4.24 and 4.25 the measured path frequencies of cores 1 and 2 are shown in the same graph. Core 1 is shown with normal biasing. Core 2 has 0.2 V forward biasing in Figure 4.24 and 0.3 V in Figure 4.25. In Table 4.11 it can be seen that the first four paths of core 1 have an average frequency of 95 MHz with normal biasing and those of core 2 an average speed of 105 MHz with 0.2 V forward body biasing. So, it can be expected that core 2 is already faster with 0.2 V forward body biasing. With 0.3 V forward biasing the average frequency of the first 4 paths for core 2 is 120 MHz and for core 1 with normal biasing 95 MHz. Therefore, core 2 is quite a bit faster with 0.3 V forward body biasing than core 1. The average frequency of all paths is 234 MHz for core 1 and 213 MHz for core 2, however. Just like with the typical measurements, this is most likely caused by the fact that less paths could be measured for core 2 and mostly long paths could be measured. Table 4.11 also contains the average frequency of the first 30 paths. It can be seen that when core 2 has 0.3 V forward body biasing, the average frequency of the first 30 paths is higher than that of core 1 with normal biasing: 179 MHz for core 1 and 188 MHz for core 2. Therefore, most long paths are faster for core 2.

All measurement results for sample C001 can be found in Appendix E.

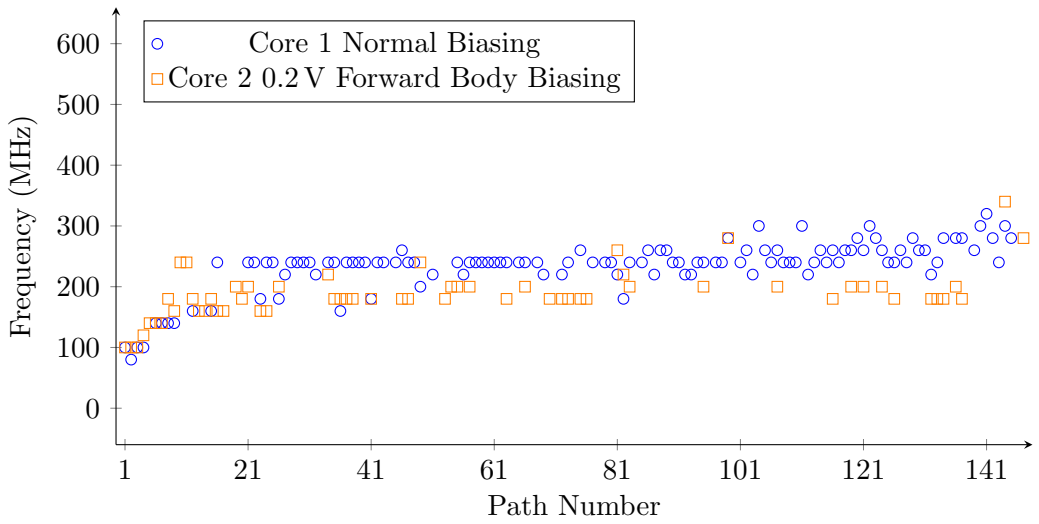


Figure 4.24: Comparison of measured path frequencies for cores 1 and 2 of sample C001 (ss corner). Room temperature, V_{DD}=1.0 V. Core 2 has 0.2 V forward body biasing.

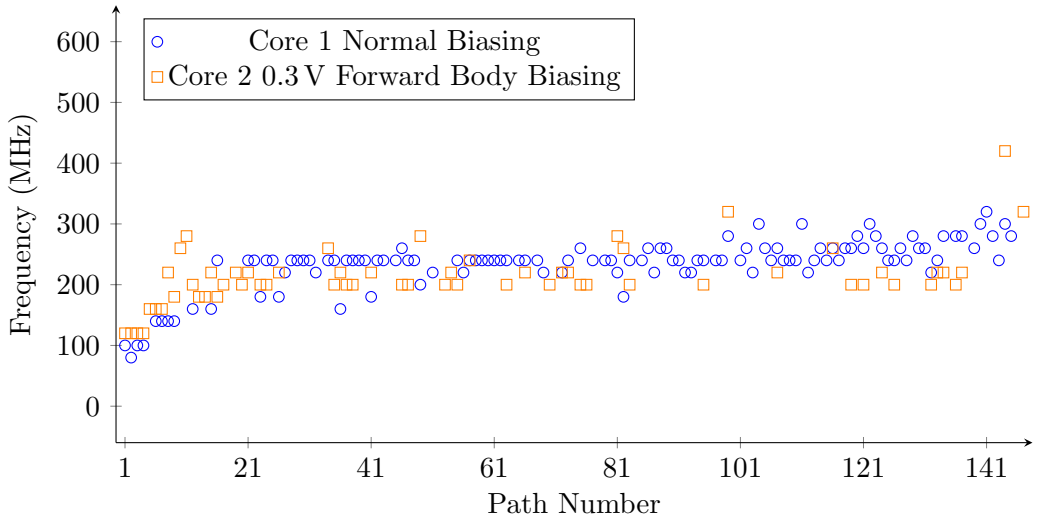


Figure 4.25: Comparison of measured path frequencies for cores 1 and 2 of sample C001 (ss corner). Room temperature, V_{DD}=1.0 V. Core 2 has 0.3 V forward body biasing.

4.5 Simulations versus Measurements

Figure 4.26 shows the measured and simulated path frequencies for core 1 with normal biasing in the same graph. The measurements are from typical sample A010. The simulation values are from simulations under typical conditions: V_{DD} at 0.99 V, a temperature of 25 °C and typical processing. Table 4.12 contains the average frequencies of the paths for these simulations. In general there is agreement between the simulated frequencies and the measured frequencies. However what is striking is that the longest path are considerably slower in the measurements than in simulation. The average of the first 4 paths in simulation is 207 MHz while the measurements have an average of 135 MHz. A reason for this difference might be IR drop. IR drop is not taken into account in the simulations and long paths will be more affected by dynamic IR drop than shorter paths. Furthermore, it is known that the amount of decoupling capacitor cells in the power network of the Beetle is smaller than in a typical design. These decoupling capacitors ensure that dynamic power drops are compenstated. Therefore, dynamic IR drop most likely explains the difference between the measurements and simulations. The average frequency of all paths is 305 MHz in simulations and 273 MHz for the measurements.

Figure 4.27 shows the measured and simulated path frequencies in the same graph for core 1 with 0.3 V forward body biasing. As can be seen very clearly, a lot of measured values are now higher than the simulated values. This can be confirmed when looking at the average frequencies. The average frequency of all paths is 388 MHz in simulations and 379 MHz for the measurements. The average frequency of the measurements is a lot closer now and is only lower, because the long paths are slow. Therefore, the actual increase in path speed with forward body biasing is higher than in simulation.

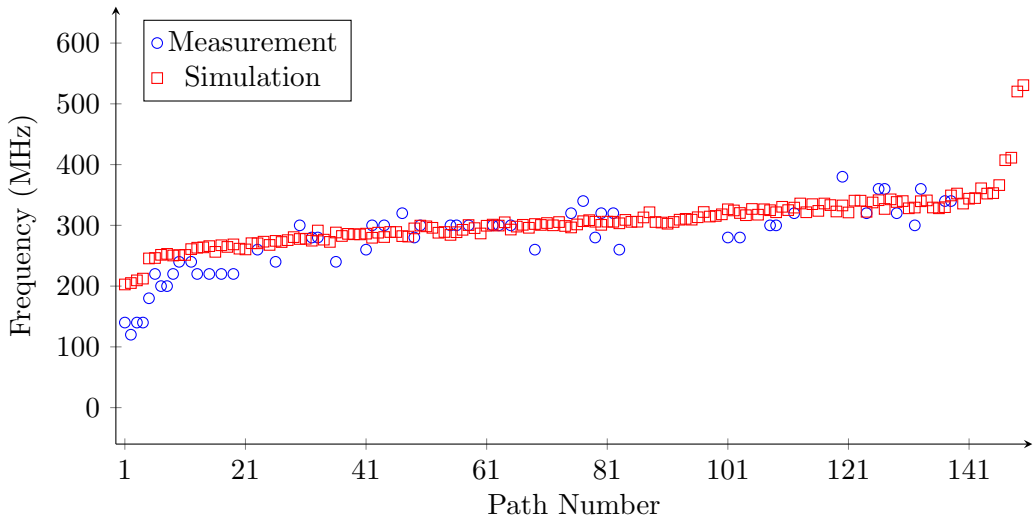


Figure 4.26: Comparison of measured and simulated path frequencies for core 1 with normal biasing. Measurements results are from sample A010 and are done at room temperature, V_{DD}=1.0 V. Simulation results from tt corner, T=25 °C, and V_{DD}=0.99 V.

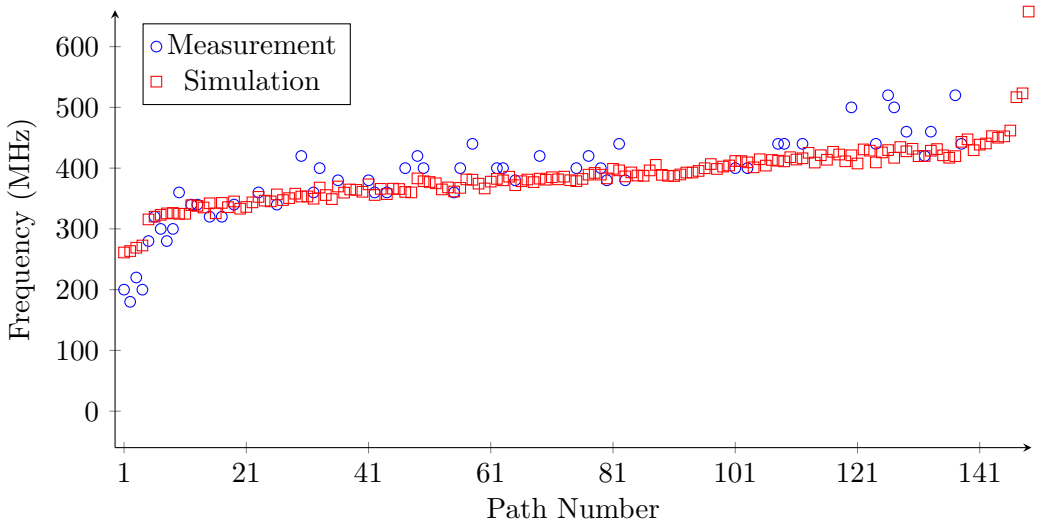


Figure 4.27: Comparison of measured and simulated path frequencies for core 1 with 0.3 V forward body biasing. Measurements results are from sample A010 and are done at room temperature, V_{DD}=1.0 V. Simulation results from tt corner, T=25 °C, and V_{DD}=0.99 V.

Table 4.12: Average frequencies of simulated paths for core 1 with tt corner, V_{DD} at 0.99 V, T=25 °C

Core	**Paths**	**Average Frequency** (MHz)			
		Forward Body Biasing			
		0.0 V	0.1 V	0.2 V	0.3 V
Core 1	1-4	207	225	244	267
	1-150	305	330	357	388

4.6 Cost Analysis

In both simulations and measurements it can be seen that with forward body biasing a good speedup can be obtained. When looking at the measurement results from C001 (slow processing), it was seen that when core 2 has 0.2 V forward biasing its performance is slightly better than that of core 1. This means there is still room for extra forward body biasing in order to make core 2 even faster. This added speed can be used to provide extra margin against small-delay defects, as core 2's worst-case samples will be further removed from the minimum target speed.

Furthermore, there is also a chance that even less forward biasing is needed to give core 2 a similar performance to core 1. The measurements have been performed at room temperature (around 20 °C). In the simulations, however, it could be seen that the Integrated Circuit (IC) is slowest at low temperatures. When looking at the simulation results at −40 °C (Figure 4.11), it was seen that core 2 actually has a better performance with normal biasing than core 1. Most likely when measurements for both cores are

performed at this temperature, core 1 will be faster. This is because core 2 has been synthesized under the assumption that there are less fluctuations in V_{DD} (due to IR drop for instance). The simulations have been done at the minimum V_{DD} that is assumed to occur. This minimum is lower for core 1 and because of this, core 2 is faster at $-40\,°C$ in the simulations. Most likely the V_{DD} fluctuations are higher during the measurements and, therefore, the performance of core 2 will be lower than in simulation and, thus, also lower than core 1. In the simulations with normal biasing at 125 °C (Figure 4.9) core 1 is faster than core 2. Because of this, it can be expected that core 2 will be closer to core 1, when measurements at $-40\,°C$ are performed, than with the current measurements at room temperature. In that case, it is likely that less forward biasing is needed for core 2 to achieve a similar performance to core 1.

Core 2 has an area reduction of roughly 25% compared to core 1. When looking at the results of the cost model case study in Figure 3.1, it can be seen that around 5 times as much money can be spent on testing (assuming there is no parametric yield loss). Forward biasing core 2's samples further, so slow silicon can be made faster than the target speed, will give more margin against small-delay defects. This eliminates part of the more extensive testing that would be necessary otherwise. Considering this, core 2 has a lot of potential to be cheaper.

4.7 Conclusions

This chapter discusses the performed experiments to evaluate the ability of body biasing to compensate process variation. The performance of a worst-case based design and a BTWC based design is compared. It is investigated if the BTWC based design can get a similar performance as the worst-case based design when forward body biasing is applied.

The chapter starts with discussing the experimental setup. The measurements are performed on the Beetle platform, which is an IC consisting of four ARM Cortex M3 cores. Core 1 is a worst-case based design, core 2 is a BTWC based design and cores 3 and 4 have parts that are mostly worst-case based designed, but some parts are designed BTWC. The performance of each core is evaluated using path delay tests. The flow for the path delay test generation is described. For each core a set of paths is selected that is comparable to the selected paths of the other cores.

Next, the path delay test generation results are discussed. The test set consists of 150 paths per core. It is observed for cores 3 and 4 that it is only possible to measure paths in the parts that are designed worst-case based. Therefore, these cores are not tested and only cores 1 and 2 are compared. With this a worst-case based design and a BTWC based design are compared.

With the path delay test set both simulations and real silicon measurements are performed. The simulations are performed under worst-case conditions. In the simulations it is observed that under the worst-case conditions core 2 is actually slightly faster than core 1. This is because core 2 is synthesized with the assumption that the supply voltage fluctuates less. During the measurements it is concluded that around 0.2 V forward body biasing is needed for core 2 to have a similar performance. Therefore, most likely the supply voltage fluctuates more than assumed during design.

Finally, the chapter performs a cost analysis of core 2 using the created cost model. It is shown that around 5 times as much money can be spent on testing as core 2 saves roughly 25% of area compared to core 1. Considering only 0.2 V forward biasing is needed for a similar performance for core 2, it is possible to add extra margin against small-delay defects by making the core faster by applying a higher forward body biasing voltage. Therefore, core 2, the BTWC based design, has a lot of potential to be cheaper.

Thesis Summary and Future Work

5

This chapter presents the conclusion of this thesis. Section 5.1 presents the summary of this thesis, organized per chapter. Finally, Section 5.2 gives recommendations for future work.

5.1 Summary

Chapter 2 presents background on the Variability Resilient Schemes (VRSs). It gives an overview of the different sources and the impact of process variation. Process variation sources include process, supply voltage, temperature, and aging. Due to process variation some Integrated Circuits (ICs) are slower and some are faster. During design, process variation needs to be taken into account to ensure a high yield.

Subsequently, a motivation for VRS is given. As modern technology is more affected by process variation, more area is lost on design margins. VRSs minimize the effects of process variation, so area and power consumption can be reduced and performance increased.

The discussed VRSs consist of body biasing, error avoidance flip-flops, and clock stretching. With forward body biasing slow ICs are made faster and with reverse body biasing fast ICs less leaky, thus reducing power consumption. The error avoidance flip-flops are flip-flops that correct timing errors by borrowing time from a successive clock cycle. With clock stretching critical paths are given extra time by dynamically stretching the clock.

Finally, a comparison is made of the VRSs. A look is taken at area overhead, throughput, and power consumption.

Chapter 3 presents the evaluation methodology for VRSs. First of all, a cost model is presented that can be used to compare the cost of conventional worst-case design with design with VRSs. With the cost model it can be checked if design with VRSs is cheaper, when the area of the IC is reduced, but the test costs increase.

Next, the testability challenges of each VRS are discussed. For body biasing the more narrow distribution of IC speeds is a challenge. Due to this more narrow distribution there are more ICs that might not meet the target speed when there is a small-delay defect. An experiment is proposed to evaluate body biasing. This experiment uses path delay tests in order to evaluate the speed of ICs.

For clock stretching a testability challenge is that more critical paths are created. This is due to the fact that the design can be run at a higher clock speed to compensate the drop in performance due to clock stretching. Typically, in a design there are only relatively few critical paths. Instead, there are more slightly shorter paths. These paths

will now become critical paths. Therefore, the design will have more critical paths which can be affected by delay defects.

Finally, a look is taken at the testability challenges of the error avoidance flip-flop. For the error avoidance flip-flop it is a challenge to control the ERROR output. A solution is proposed with a low area overhead. A disadvantage of the proposed solution is that test time can become high when the design has a lot of error avoidance flip-flops. Furthermore, it is hard to get a good coverage of delay faults and test time can become high for them.

Chapter 4 discusses the performed experiments to evaluate the ability of body biasing to compensate process variation. The performance of a worst-case based design and a Better-Than-Worst-Case (BTWC) based design is compared. It is investigated if the BTWC based design can get a similar performance as the worst-case based design when forward body biasing is applied.

The chapter starts with discussing the experimental setup. The measurements are performed on the Beetle platform, which is an IC consisting of four ARM Cortex M3 cores. Core 1 is a worst-case based design, core 2 is a BTWC based design and cores 3 and 4 have parts that are mostly worst-case based designed, but some parts are designed BTWC. The performance of each core is evaluated using path delay tests. The flow for the path delay test generation is described. For each core a set of paths is selected that is comparable to the selected paths of the other cores.

Next, the path delay test generation results are discussed. The test set consists of 150 paths per core. It is observed for cores 3 and 4 that it is only possible to measure paths in the parts that are designed worst-case based. Therefore, these cores are not tested and only cores 1 and 2 are compared. With this a worst-case based design and a BTWC based design are compared.

With the path delay test set both simulations and real silicon measurements are performed. The simulations are performed under worst-case conditions. In the simulations it is observed that under the worst-case conditions core 2 is actually slightly faster than core 1. This is because core 2 is synthesized with the assumption that the supply voltage fluctuates less. During the measurements it is concluded that around 0.2 V forward body biasing is needed for core 2 to have a similar performance. Therefore, most likely the supply voltage fluctuates more than assumed during design.

Finally, the chapter performs a cost analysis of core 2 using the created cost model. It is shown that around 5 times as much money can be spent on testing as core 2 saves roughly 25% of area compared to core 1. Considering only 0.2 V forward biasing is needed for a similar performance for core 2, it is possible to add extra margin against small-delay defects by making the core faster by applying a higher forward body biasing voltage. Therefore, core 2, the BTWC based design, has a lot of potential to be cheaper.

5.2 Future Work

The following future work is recommended:

- **Measure more Beetle samples**
  Only a few samples were used due to time constraints for the path delay measurements. 100 Samples with typical processing and 50 samples with slow and fast processing are produced. By measuring more samples the within-wafer variation can be determined. Based on these results, it can be concluded, whether the speed of every IC needs to be measured to determine the body biasing voltages or if it is enough to perform only one measurement per wafer and apply the same body biasing voltages to all ICs on it.

- **Perform measurements under extreme temperatures**
  The path delay measurements have only been performed at room temperature, because the measurement setup was not ready yet to control the temperature. Performing measurements at $-40\,^{\circ}\mathrm{C}$ is interesting, because at this temperature the gates are slowest. According to the simulations the performance of core 2 with normal biasing is closer to the performance of core 1 at $-40\,^{\circ}\mathrm{C}$ than at $125\,^{\circ}\mathrm{C}$. If this is the case for the measurements as well, then it can be expected that core 2's performance with normal biasing at $-40\,^{\circ}\mathrm{C}$ is closer to core 1 than it is in the measurements done at room temperature. This would mean a lower forward biasing voltage is needed for core 2 to achieve a similar performance.

- **Investigate potential of reverse body biasing**
  With reverse body biasing the Threshold Voltage (V_{th}) of the transistor can be increased. By doing this ICs are made less leaky at the cost of switching speed. Thanks to this ICs with fast processing, which have the highest leakage, can be made less leaky, thus the power consumption is decreased. This thesis did not look at the potential of reverse body biasing to decrease power consumption.

- **Delay Testing of Error Avoidance Flip-Flop's ERROR output**
  This thesis proposed a delay test method of the ERROR output of the Error Avoidance Flip-Flop with quite a few limitations. As future work a better method could be investigated. Furthermore, it would be possible to investigate if the testability of the whole design is high enough when delay testing of the ERROR outputs is omitted.

Bibliography

[1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Design Automation Conference, 2003. Proceedings*, June 2003, pp. 338–342.

[2] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken, "Timber: Time borrowing and error relaying for online timing error resilience," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, March 2010, pp. 1554–1559.

[3] D. de Vries, "Investigation of gross die per wafer formulas," *Semiconductor Manufacturing, IEEE Transactions on*, vol. 18, no. 1, pp. 136–139, Feb 2005.

[4] S. Ghosh, P. Ndai, S. Bhunia, and K. Roy, "Tolerance to small delay defects by adaptive clock stretching," in *On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International*, July 2007, pp. 244–252.

[5] F. Yuan, Y. Liu, W.-B. Jone, and Q. Xu, "On testing timing-speculative circuits," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, May 2013, pp. 1–6.

[6] M. Wirnshofer, *Variation-Aware Adaptive Voltage Scaling for Digital CMOS Circuits*, first edition ed. Springer, 2013.

[7] V. Mahalingam, N. Ranganathan, and J. Harlow, "A fuzzy optimization approach for variation aware power minimization during gate sizing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 8, pp. 975–984, Aug 2008.

[8] C. Zhuo, D. Blaauw, and D. Sylvester, "Variation-aware gate sizing and clustering for post-silicon optimized circuits," in *Low Power Electronics and Design (ISLPED), 2008 ACM/IEEE International Symposium on*, Aug 2008, pp. 105–110.

[9] A. Mutlu, K. Le, M. Celik, D. sun Tsien, G. Shyu, and L.-C. Yeh, "An exploratory study on statistical timing analysis and parametric yield optimization," in *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, March 2007, pp. 677–684.

[10] K. Bowman, J. Tschanz, S. Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik, and V. De, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 1, pp. 194–208, Jan 2011.

[11] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De, "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," in *Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International*, vol. 1, Feb 2002, pp. 422–478 vol.1.

[12] M. Ottavi, S. Pontarelli, D. Gizopoulos, C. Bolchini, M. Michael, L. Anghel, M. Tahoori, A. Paschalis, P. Reviriego, O. Bringmann, V. Izosimov, H. Manhaeve, C. Strydis, and S. Hamdioui, "Dependable multicore architectures at nanoscale: The view from europe," *Design Test, IEEE*, vol. 32, no. 2, pp. 17–28, April 2015.

[13] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Mitigating parameter variation with dynamic fine-grain body biasing," in *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, Dec 2007, pp. 27–42.

[14] Y.-W. Yang and K. Shu-Min Li, "Temperature-aware dynamic frequency and voltage scaling for reliability and yield enhancement," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, Jan 2009, pp. 49–54.

[15] V. Mahalingam, N. Ranganathan, N. Ahmed, and T. Haider, "A variation aware circuit design using dynamic clock stretching," in *Electronic System Design (ISED), 2010 International Symposium on*, Dec 2010, pp. 125–130.

[16] M. Choudhury, V. Chandra, R. Aitken, and K. Mohanram, "Time-borrowing circuit designs and hardware prototyping for timing error resilience," *Computers, IEEE Transactions on*, vol. 63, no. 2, pp. 497–509, Feb 2014.

[17] J. Smolens, B. Gold, B. Falsafi, and J. Hoe, "Reunion: Complexity-effective multicore redundancy," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, Dec 2006, pp. 223–234.

[18] N. Kandasamy, J. Hayes, and B. Murray, "Transparent recovery from intermittent faults in time-triggered distributed systems," *Computers, IEEE Transactions on*, vol. 52, no. 2, pp. 113–125, Feb 2003.

[19] "NXP Semiconductors," https://www.nxp.com, [Online; accessed 05-Januari-2016].

[20] K. Bowman, A. Alameldeen, S. Srinivasan, and C. Wilkerson, "Impact of die-to-die and within-die parameter variations on the throughput distribution of multi-core processors," in *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, Aug 2007, pp. 50–55.

[21] "Understanding Process Corner (Corner Lots)," http://anysilicon.com/understanding-process-corner-corner-lots/, [Online; accessed 30-November-2015].

[22] "Product binning," https://en.wikipedia.org/wiki/Product_binning, 2008, [Online; accessed 7-July-2015].

[23] J. Zeng, M. Abadir, G. Vandling, L.-C. Wang, S. Karako, and J. Abraham, "On correlating structural tests with functional tests for speed binning of high performance design," in *Microprocessor Test and Verification (MTV'04), Fifth International Workshop on*, Sept 2004, pp. 103–109.

[24] K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, W.-k. Shih, S. Sivakumar, G. Taylor, P. VanDerVoorn, and K. Zawadzki, "Managing process variation in intel's 45nm cmos technology." *Intel Technology Journal*, vol. 12, no. 2, 2008.

[25] M. Choudhury and K. Mohanram, "Masking timing errors on speed-paths in logic circuits," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, April 2009, pp. 87–92.

[26] S. Ghosh and K. Roy, "Exploring high-speed low-power hybrid arithmetic units at scaled supply and adaptive clock-stretching," in *Design Automation Conference, 2008. ASPDAC 2008. Asia and South Pacific*, March 2008, pp. 635–640.

[27] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De, "Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage," in *Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. 2002 IEEE International*, vol. 2, Feb 2002, pp. 344–539.

[28] J. Patterson, D.A. & Hennessy, *Computer Organization and Design: the Hardware/-Software Interface*, fourth edition ed. Morgan Kaufmann Publishers, 2009.

[29] J. Patel, "Stuck-at fault: a fault model for the next millennium," in *Test Conference, 1998. Proceedings., International*, Oct 1998, pp. 1166–.

[30] K.-T. Cheng, "Transition fault testing for sequential circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 12, no. 12, pp. 1971–1983, Dec 1993.

[31] T. M. . P. K. . C. K., *Test and Diagnosis for Small-Delay Defects*, first edition ed. Springer, 2012.

[32] X. Lin, K.-H. Tsai, C. Wang, M. Kassab, J. Rajski, T. Kobayashi, R. Klingenberg, Y. Sato, S. Hamada, and T. Aikyo, "Timing-aware atpg for high quality at-speed testing of small delay defects," in *Test Symposium, 2006. ATS '06. 15th Asian*, Nov 2006, pp. 139–146.

[33] T. Yoneda, K. Hori, I. Inoue, and H. Fujiwara, "Faster-than-at-speed test for increased test quality and in-field reliability," in *Test Conference (ITC), 2011 IEEE International*, Sept 2011, pp. 1–9.

[34] B. M. . A. V., *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, first edition ed. KLUWER ACADEMIC PUBLISHERS, 2002.

[35] R. Kannah and C. Ravikumar, "Functional testing of microprocessors with graded fault coverage," in *Test Symposium, 2000. (ATS 2000). Proceedings of the Ninth Asian*, 2000, pp. 204–208.

[36] "Cortex-M3 Processor," http://www.arm.com/products/processors/cortex-m/cortex-m3.php, [Online; accessed 21-November-2014].

[37] "Joint Test Action Group," https://en.wikipedia.org/wiki/Joint_Test_Action_Group, [Online; accessed 21-November-2014].

[38] M. Sai Kumar, D. Kumar, and P. Samundiswary, "Design and performance analysis of multiply-accumulate (mac) unit," in *Circuit, Power and Computing Technologies (ICCPCT), 2014 International Conference on*, March 2014, pp. 1084–1089.

[39] "The Tessent Product Suite," https://www.mentor.com/products/silicon-yield/, [Online; accessed 23-November-2014].

[40] "Tempus Timing Signoff Solution," http://www.cadence.com/products/mfg/tempus/pages/default.aspx, [Online; accessed 23-November-2014].

[41] "SPICE," https://en.wikipedia.org/wiki/SPICE, [Online; accessed 17-March-2014].

[42] "Spectre Circuit Simulator," http://www.cadence.com/products/cic/spectre_circuit/pages/default.aspx, [Online; accessed 11-November-2015].

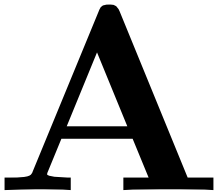
[43] T. Taylor and G. Maston, "Standard test interface language (stil) a new language for patterns and waveforms," in *Test Conference, 1996. Proceedings., International*, Oct 1996, pp. 565–570.

[44] "LabVIEW System Design Software," http://www.ni.com/labview/, [Online; accessed 11-November-2015].

[45] A. Sassone, A. Calimera, A. Macii, E. Macii, M. Poncino, R. Goldman, V. Melikyan, E. Babayan, and S. Rinaudo, "Investigating the effects of inverted temperature dependence (itd) on clock distribution networks," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, March 2012, pp. 165–166.

[46] "Python," https://www.python.org/, [Online; accessed 11-November-2015].

Cost Model Code



A.1 Simplistic Cost Model

cost_analysis.m:

```
clear all;
close all;

cost = zeros(2, 1000);
cost_par_yield_99 = zeros (2, 1000);
cost_par_yield_95 = zeros (2, 1000);
cost_par_yield_90 = zeros (2, 1000);

area_axis = zeros(1,1000);
test_cost_axis = zeros(1, 1000);


for n = 1:1000 % iterate over x-value: area
    area_axis(n) = (0.5+(n*0.0005))*100;
    test_cost_axis(n) = (1+(n*0.004))*100;

    for m = 1:1000 %iterate over y-value: test cost
        cost(m, n) = cost_vra(1+(m*0.004), 1.0, 0.5+(n*0.0005));
        cost_par_yield_99(m, n) = cost_vra(1+(m*0.004), 0.99, 0.5+(n
            *0.0005));
        cost_par_yield_95(m, n) = cost_vra(1+(m*0.004), 0.95, 0.5+(n
            *0.0005));
        cost_par_yield_90(m, n) = cost_vra(1+(m*0.004), 0.90, 0.5+(n
            *0.0005));

    end

end

% Solve inequalities
ineq1 = cost < cost_normal();
ineq2 = cost_par_yield_99 < cost_normal();
ineq3 = cost_par_yield_95 < cost_normal();
ineq4 = cost_par_yield_90 < cost_normal();

% Plot inequalities
figure, hold on
c = 1:4;                                    %# Contour levels
contour(area_axis, test_cost_axis, c(1) * ineq1, [c(1), c(1)], 'b')
contour(area_axis, test_cost_axis, c(2) * ineq2, [c(2), c(2)], 'g')
contour(area_axis, test_cost_axis, c(3) * ineq3, [c(3), c(3)], 'r')
```

```
contour(area_axis, test_cost_axis, c(4) * ineq4, [c(4), c(4)], 'm')


xlabel('area␣(%)');
ylabel('test␣cost␣increase␣(%)');

legend('100%␣parametric␣yield','99%␣parametric␣yield','95%␣parametric␣
    yield', '90%␣parametric␣yield')
```

cost_normal.m:

```
function cost = cost_normal( )
% Function to calculate cost of IC designed with conventional method

cost = (0.9 + 0.1)/(yield_defect(0.25, 0.5));

end
```

cost_vra.m:

```
function cost = cost_vra( beta, yield_parametric, alfa )
% Function to calculate cost of IC designed with VRA

cost = (0.9 + beta * 0.1)/(yield_parametric*yield_defect(0.25, alfa*0.5)
    *(1/alfa));

end
```

yield_defect.m:

```
function y = yield_defect(defect_per_area, die_area)
    y = 1/(1 + (defect_per_area * (die_area/2)))^2;
end
```

A.2 Cost Model with GDW Two Term

cost_analysis.m:

```
clear all;
close all;

cost = zeros(2, 1000);
cost_par_yield_99 = zeros (2, 1000);
cost_par_yield_95 = zeros (2, 1000);
cost_par_yield_90 = zeros (2, 1000);

area_axis = zeros(1,1000);
```

```
test_cost_axis = zeros(1, 1000);


for n = 1:1000 % iterate over x-value: area
    area_axis(n) = (0.5+(n*0.0005))*100;
    test_cost_axis(n) = (1+(n*0.004))*100;

    for m = 1:1000 %iterate over y-value: test cost
        cost(m, n) = cost_vra(1+(m*0.004), 1.0, 0.5+(n*0.0005), 0.5, 147)
            ;
        cost_par_yield_99(m, n) = cost_vra(1+(m*0.004), 0.99, 0.5+(n
            *0.0005), 0.5, 147);
        cost_par_yield_95(m, n) = cost_vra(1+(m*0.004), 0.95, 0.5+(n
            *0.0005), 0.5, 147);
        cost_par_yield_90(m, n) = cost_vra(1+(m*0.004), 0.90, 0.5+(n
            *0.0005), 0.5, 147);

    end

end

% Solve inequalities
ineq1 = cost < cost_normal(0.5, 147);
ineq2 = cost_par_yield_99 < cost_normal(0.5, 147);
ineq3 = cost_par_yield_95 < cost_normal(0.5, 147);
ineq4 = cost_par_yield_90 < cost_normal(0.5, 147);

% Plot inequalities
figure, hold on
c = 1:4;                                    %# Contour levels
contour(area_axis, test_cost_axis, c(1) * ineq1, [c(1), c(1)], 'b')
contour(area_axis, test_cost_axis, c(2) * ineq2, [c(2), c(2)], 'g')
contour(area_axis, test_cost_axis, c(3) * ineq3, [c(3), c(3)], 'r')
contour(area_axis, test_cost_axis, c(4) * ineq4, [c(4), c(4)], 'm')


xlabel('area␣(%)');
ylabel('test␣cost␣increase␣(%)');

legend('100%␣parametric␣yield','99%␣parametric␣yield','95%␣parametric␣
    yield', '90%␣parametric␣yield')
```

cost_normal.m:

```
function cost = cost_normal( Area, R_eff )
% Function to calculate cost of IC designed with conventional method

cost = (0.9 + 0.1)/(yield_defect(0.25, 0.5)* ((pi*R_eff^2)/Area - ( pi
    *1.16*R_eff/(sqrt(Area))    )  ));

end
```

cost_vra.m:

```
function cost = cost_vra( beta, yield_parametric, alfa, Area, R_eff )
% Function to calculate cost of IC designed with VRA

cost = (0.9 + beta * 0.1)/(yield_parametric*yield_defect(0.25, alfa*0.5)
    *((pi*R_eff^2)/(Area*alfa) - ( pi*1.16*R_eff/(sqrt(Area*alfa)))  ) );


end
```

yield_defect.m:

```
function y = yield_defect(defect_per_area, die_area)
    y = 1/(1 + (defect_per_area * (die_area/2)))^2;
end
```

Measurement Processing Script B

```
#!/usr/bin/python

##########################################################################
# Author: Daniel Kraak 2015
# Purpose: reads in list of measurement log files and
#          simulation frequencies.
#
# __-_---- is filtered to __------(most likely a glitch)
#
# For every sweep the frequencies at which the error signal rises
# is stored, then it is checked which of the stored frequencies
# is closest to simulated value. If the frequencies are not close
# enough to simulation they are discarded
#
# Run: ./filter_frequency_sweep_robust.py {measurement log files}
#      {simulation frequencies}
#
# Measurement and simulation files need to be given in incremental
# forward biasing order
##########################################################################
import sys
import re


allowed_deviations=[0.25, 0.25, 0.27, 0.3]

sim_freq_scale=float(sys.argv[len(sys.argv)-1])

##########################################################################
# read in log files
##########################################################################

num_of_sweeps=int((len(sys.argv)-1)/2) #number of body bias sweeps

freqs = [] #stores used frequencies per file
fields = []
for bias in range(1, num_of_sweeps+1): #read in sweep data

    freqs.append([])
    fields.append([])
    fileHandle = open(sys.argv[bias], 'r')

    #read in line with frequencies, creates a list of [80 Mhz, 100 Mhz, ...]
    freqs[bias-1]=re.split(';',fileHandle.readline().rstrip('\n'));
    for j,freq in enumerate(freqs[bias-1]):
        freqs[bias-1][j] = float(re.split('␣MHz', freq)[0])*1e6
```

```
    fileHandle.readline(); #read in line with sample

    for line in fileHandle:
        fields[bias-1].append(re.split(';',line.rstrip('\n')))

    fileHandle.close()

#########################################################################
# read in simulation file
#########################################################################
sim_freq = []

for bias in range(1, num_of_sweeps+1): #iterate over body bias
    sim_freq.append([])
    fileHandle = open(sys.argv[num_of_sweeps+bias], 'r') #open file
    for line in fileHandle:
        sim_freq[bias-1].append(float(re.split('\t',line.rstrip('\n'))[1]))
    fileHandle.close()

# read in path names
path_names = []
fileHandle = open(sys.argv[num_of_sweeps+1], 'r') #open first body bias file
for line in fileHandle:
        full_string=re.split('\t',line.rstrip('\n'))[0] #contains "$path_name frequency"
        full_name=re.split("␣frequency",full_string)[0]
        path_name=re.split('_',full_name)[0] + '␣' + re.split('_',full_name)[2]
        path_names.append(path_name)
fileHandle.close()

#########################################################################
# filter out glitches ___---_--------- => ___----------
#########################################################################
for bias in range(len(fields)): #iterate over body bias
    for vector in range(len(fields[bias])): #iterate over test vectors
        for freq in range(1, len(fields[bias][vector])-1): #sweep over frequencies
            if(re.split('\s',fields[bias][vector][freq-1])[1]!='#' and
                re.split('\s',fields[bias][vector][freq])[1]=='#'
                and re.split('\s',fields[bias][vector][freq+1])[1]!='#'): #detect -_-
                fields[bias][vector][freq]=fields[bias][vector][freq+1]

#########################################################################
# create freq_transitions array
#########################################################################
freq_transitions = [] #holds the frequencies at which error transition occurs

for bias in range(len(fields)): #iterate over body bias sweep
    freq_transitions.append([])
    for vector in range(len(fields[bias])):
        freq_transitions[bias].append([]) #create for every vector
        for freq in range(len(fields[bias][vector])): #sweep over frequencies
            if (re.split('\s',fields[bias][vector][freq])[1]!='#'): #sweep has error
                if (freq==0): #first freq sweep immediately gives error
                    freq_transitions[bias][vector].append(freqs[bias][freq])
                else:
```

```
                    #check previous freq, if no error occurred,
                    #then transition occurs
                    if(re.split('\s',fields[bias][vector][freq-1])[1]=='#'):
                        freq_transitions[bias][vector].append(freqs[bias][freq])

########################################################################
# Long paths such as T2 are a lot slower than simulation
# The idea is to give long paths more allowed deviation-
# from the measurements.
# Simulation value of T2 is taken as reference
########################################################################
long_path_thresshold = []
for bias in range(len(sim_freq)): #iterate over body bias sweep
    long_path_thresshold.append(1.1*sim_freq[bias][1])

########################################################################
# take closest freq to simulation from freq_transitions
########################################################################
filtered_freq = []
for bias in range(len(freq_transitions)): #iterate over body bias
    filtered_freq.append([])

    for vector in range(len(freq_transitions[bias])): #iterate over vectors
        filtered_freq[bias].append("NaN")
        expected_freq=sim_freq[bias][vector]*sim_freq_scale
        cur_percentage=100

        #iterate over transition frequencies
        for k, freq in enumerate(freq_transitions[bias][vector]):
            is_long_path=expected_freq < long_path_thresshold[bias]
            diff=(freq-10e6)-expected_freq
            percentage=abs(diff/expected_freq)

            if(is_long_path):
                allowed_deviation=0.5
            else:
                allowed_deviation=allowed_deviations[bias]

            if(percentage < cur_percentage and percentage < allowed_deviation):
                if((is_long_path and sim_freq[bias][vector] > freq-10e6)
                    or not is_long_path):
                    filtered_freq[bias][vector] = freq
                    #update cur_percentage with new lowest percentage
                    cur_percentage = percentage

########################################################################
# print final results
########################################################################
for vector in range(len(filtered_freq[0])): #iterate over vectors
    print(path_names[vector],";T", vector+1, ";", end="")
    for bias in range(len(filtered_freq)):
        print(float(filtered_freq[bias][vector])-20e6, ";", end="")
    for bias in range(len(filtered_freq)):
        print(sim_freq[bias][vector], ";", end="")
    print()
```

C Transient Simulations

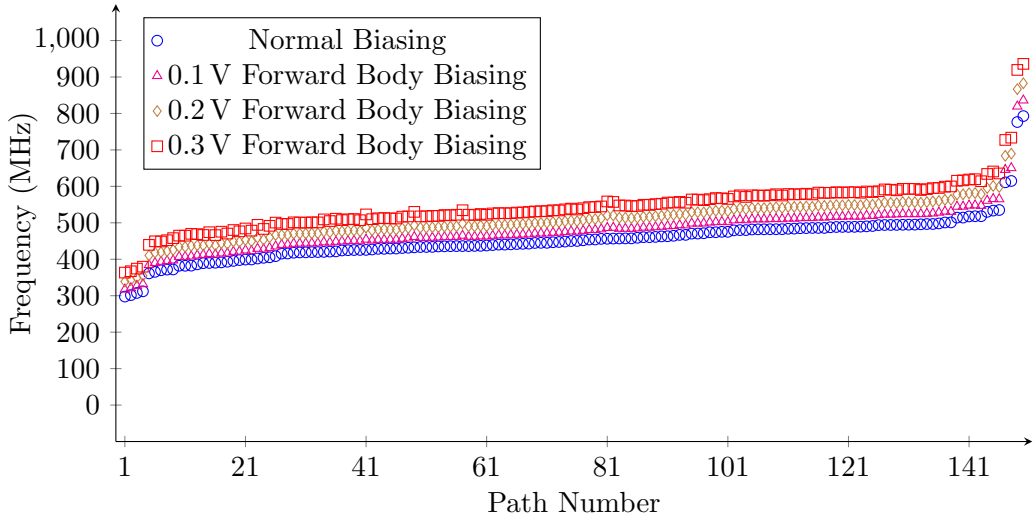


Figure C.1: Simulated path frequencies for core 1 under typical conditions: tt corner, V_{DD}=1.1 V, T=25 °C

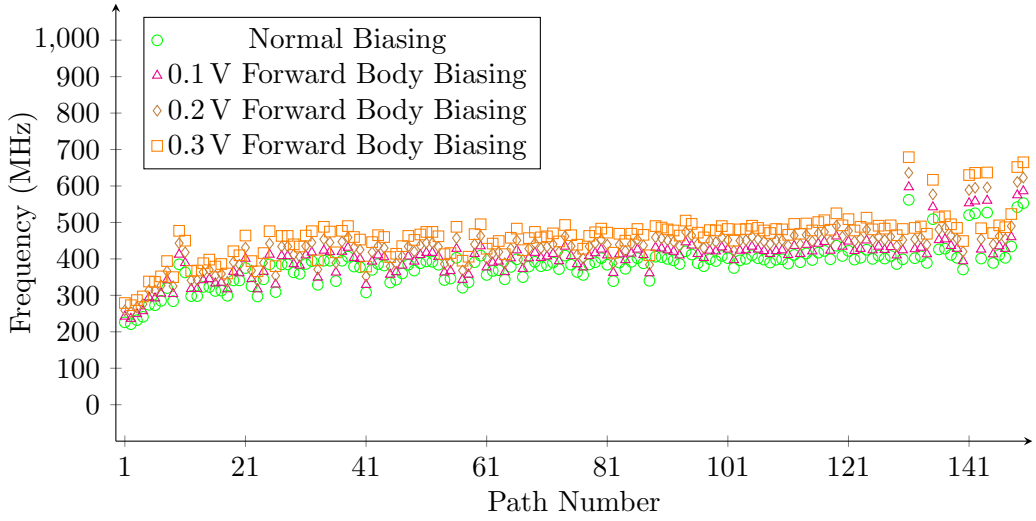


Figure C.2: Simulated path frequencies for core 2 under typical conditions: tt corner, V_{DD}=1.1 V, T=25 °C

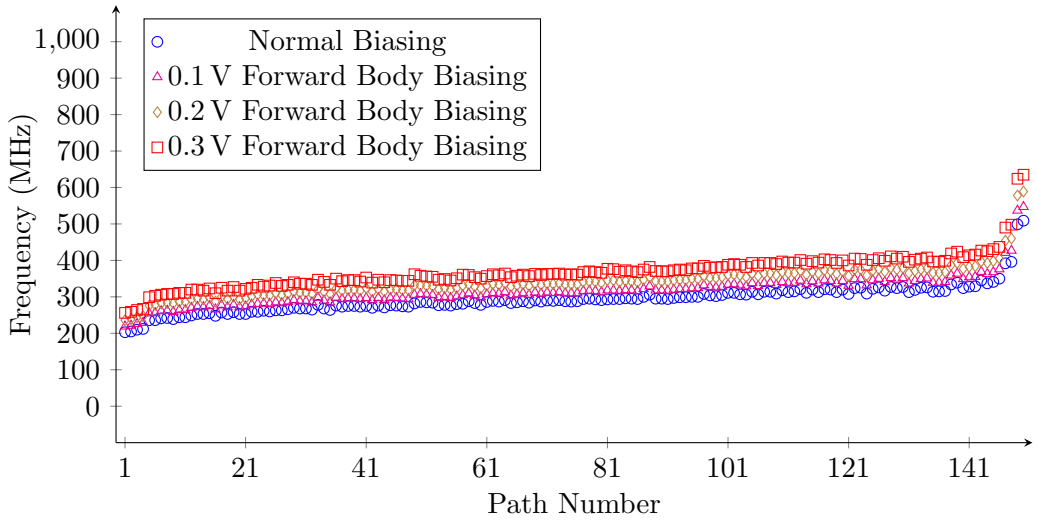


Figure C.3: Simulated path frequencies for core 1 under worst-case conditions: ss corner, V_{DD}=0.99 V, T=125 °C

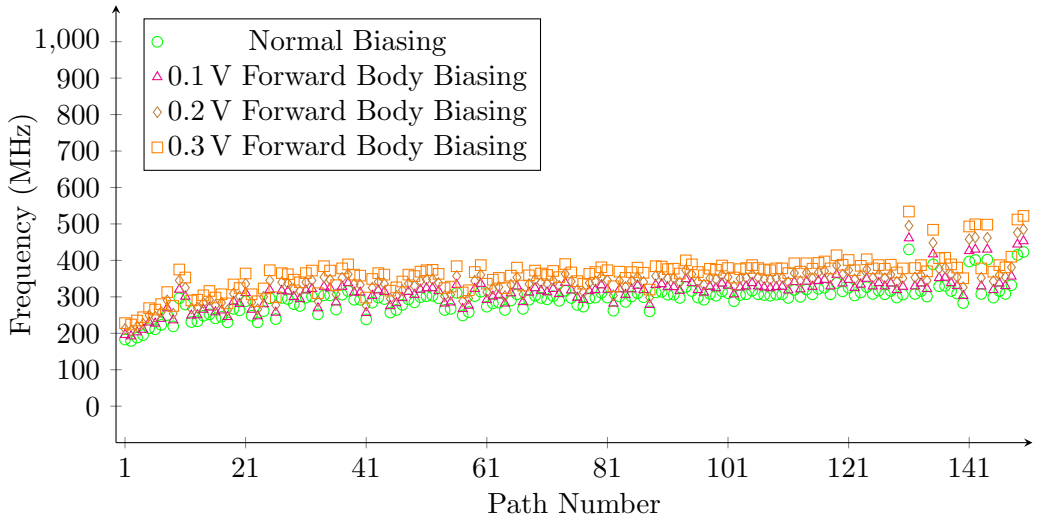


Figure C.4: Simulated path frequencies for core 2 under worst-case conditions: ss corner, V_{DD}=1.045 V, T=125 °C

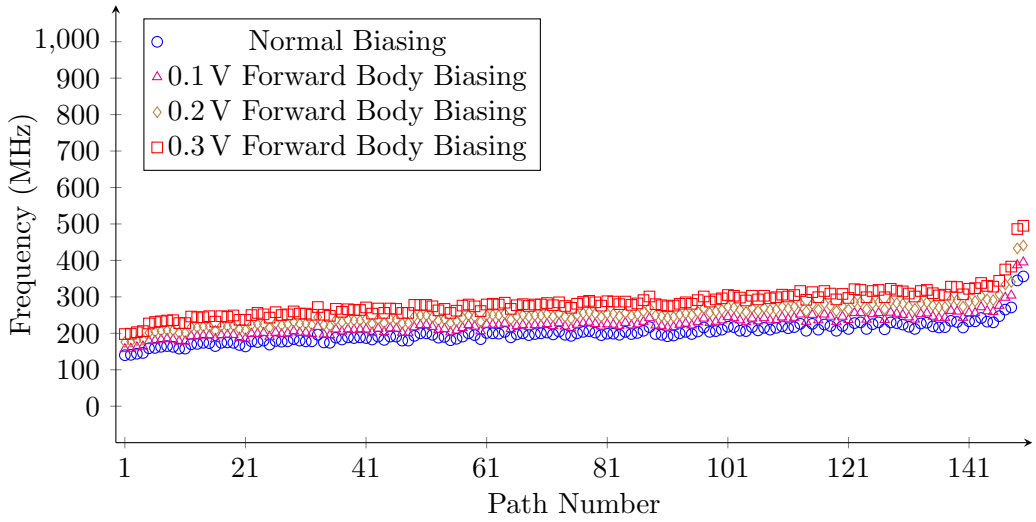


Figure C.5: Simulated path frequencies for core 1 under worst-case conditions: ss corner, V_{DD}=0.99 V, T=−40 °C

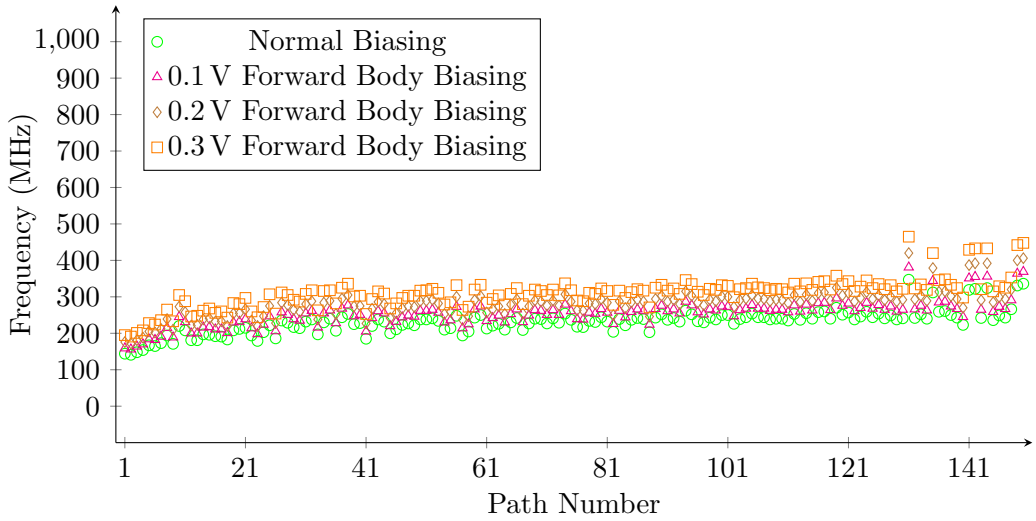


Figure C.6: Simulated path frequencies for core 2 under worst-case conditions: ss corner, V_{DD}=1.045 V, T=−40 °C

Measurement Results Typical Processing

D

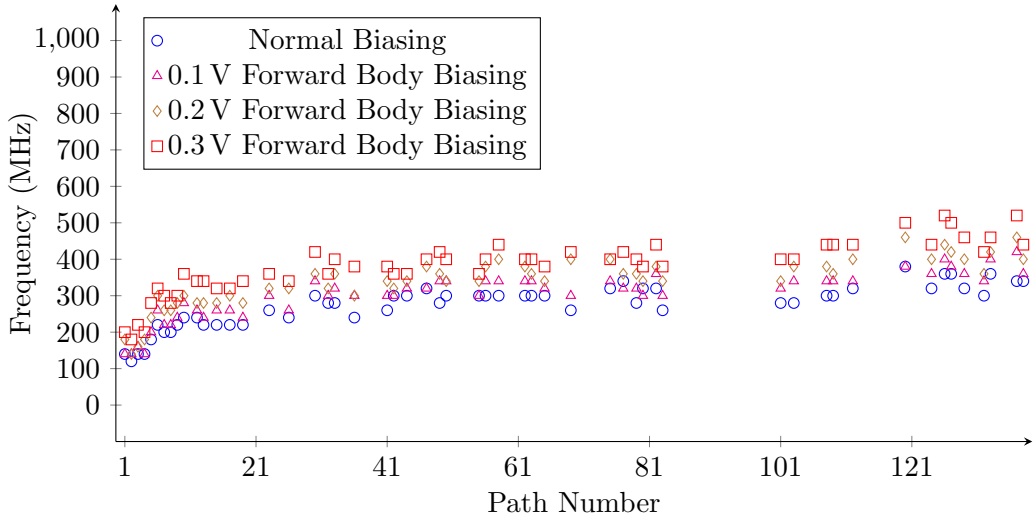


Figure D.1: Measured path frequencies for core 1 of sample A010 (tt corner). Room temperature, V_{DD}=1.0 V

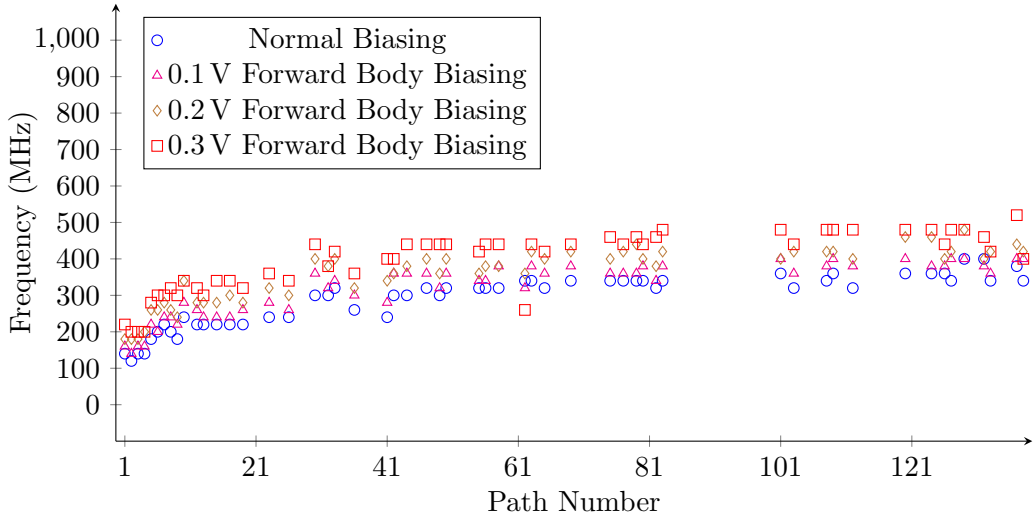


Figure D.2: Measured path frequencies for core 1 of sample A028 (tt corner). Room temperature, V_{DD}=1.0 V

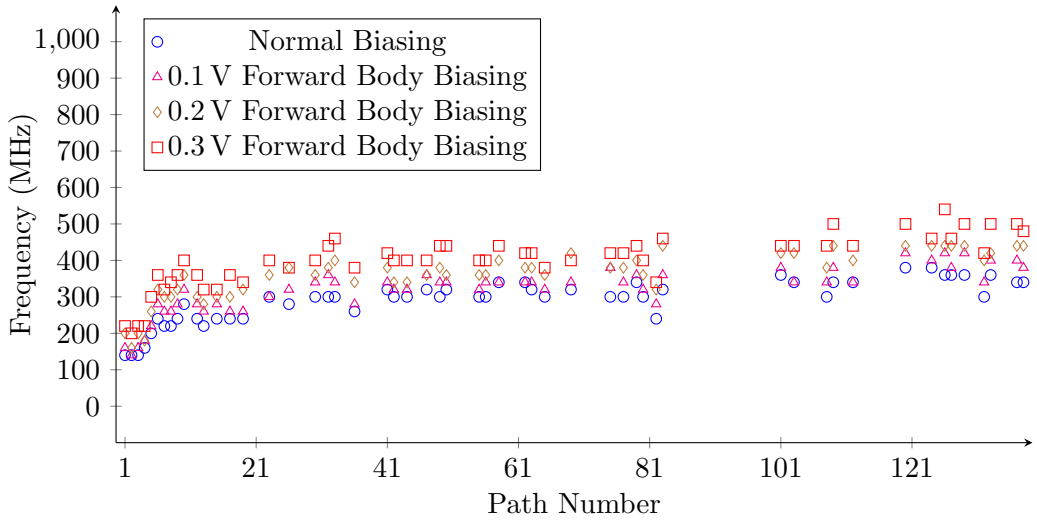


Figure D.3: Measured path frequencies for core 1 of sample A029 (tt corner). Room temperature, V_{DD}=1.0 V

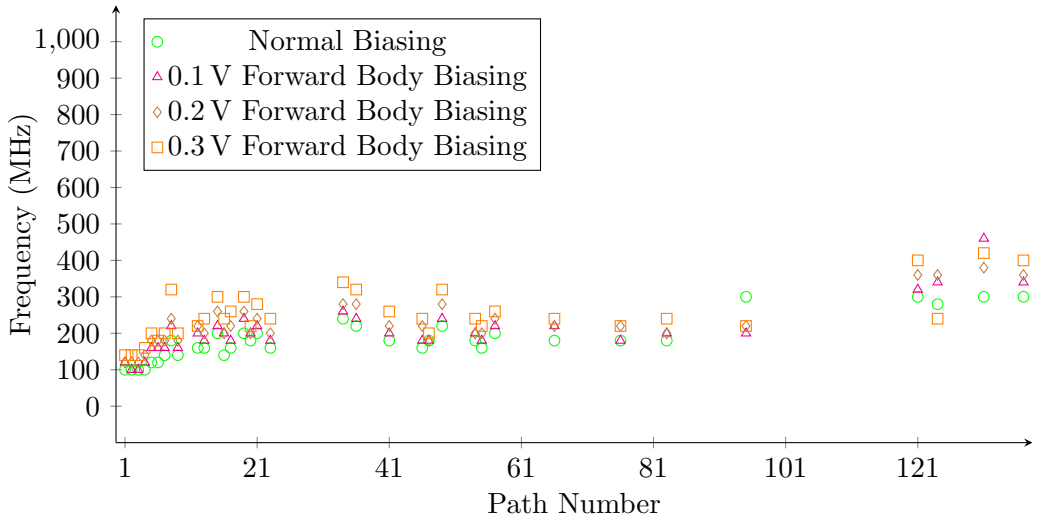


Figure D.4: Measured path frequencies for core 2 of sample A010 (tt corner). Room temperature, V_{DD}=1.0 V

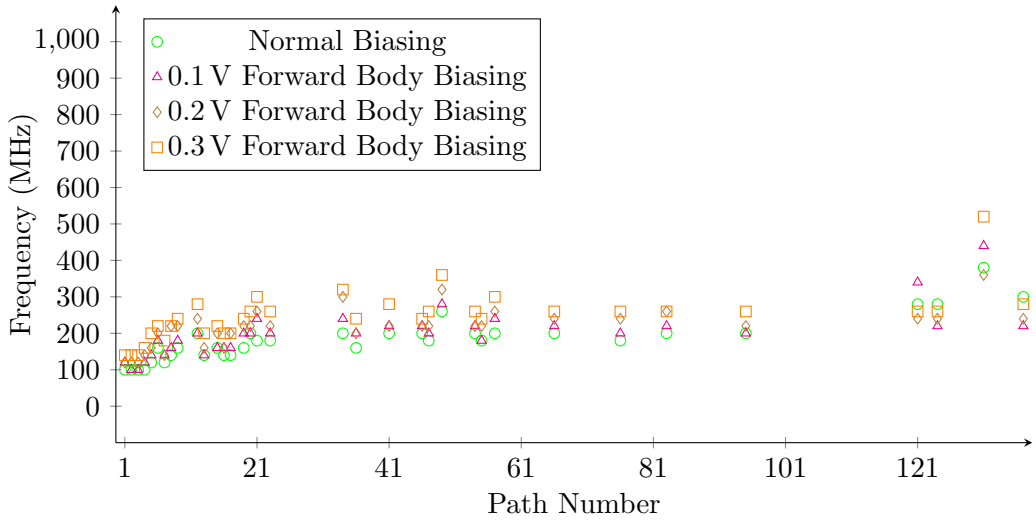


Figure D.5: Measured path frequencies for core 2 of sample A028 (tt corner). Room temperature, V_{DD}=1.0 V

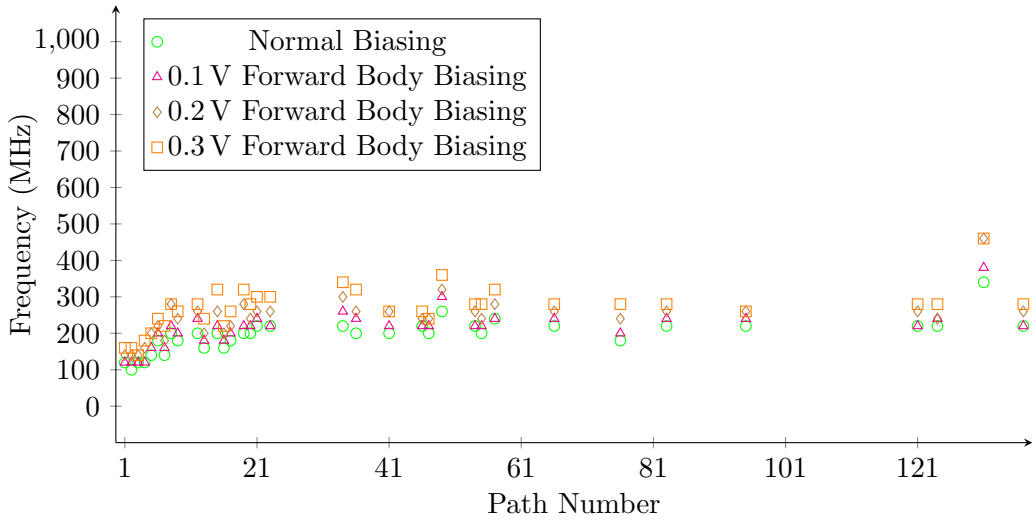


Figure D.6: Measured path frequencies for core 1 of sample A029 (tt corner). Room temperature, V_{DD}=1.0 V

Measurement Results Slow Processing

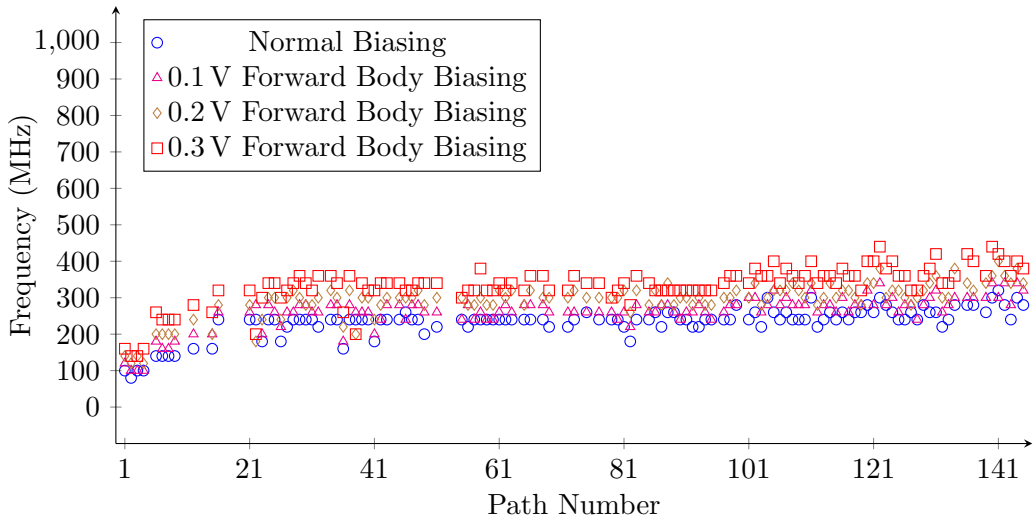


Figure E.1: Measured path frequencies for core 1 of sample C001 (ss corner). Room temperature, V_{DD}=1.0 V

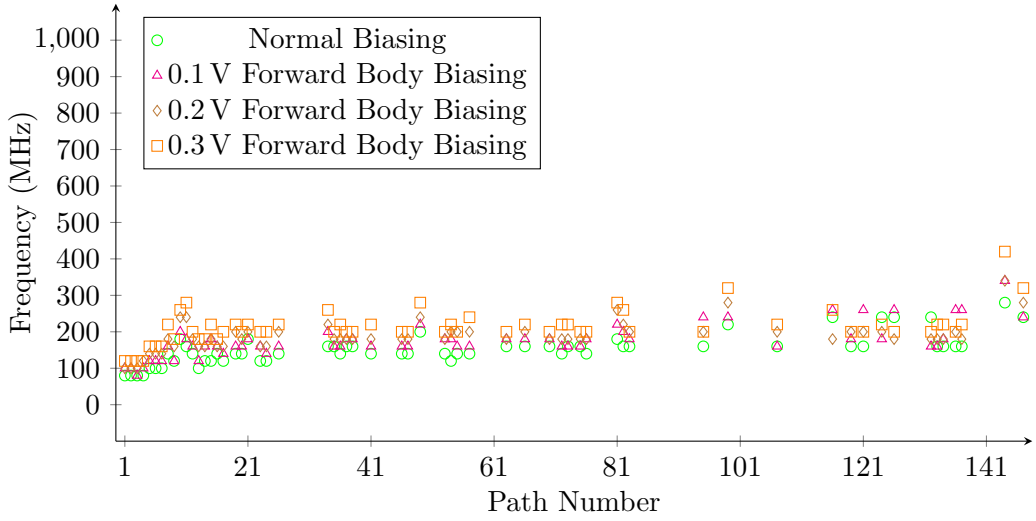


Figure E.2: Measured path frequencies for core 2 of sample C001 (ss corner). Room temperature, V_{DD}=1.0 V