Fakultät für Elektrotechnik und Informationstechnik
Technische Universität München

Automated Power Optimization of Sequential Integrated Circuits through Approximate Computing

David M. May

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr.-Ing. Ulf Schlichtmann

Prüfer der Dissertation:
1. apl. Prof. Dr.-Ing. Walter Stechele
2. Prof. Lirida de Barros Naviner, Ph.D.

Die Dissertation wurde am 16.02.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 19.06.2017 angenommen.

Abstract

Approximate Computing is a paradigm that has recently drawn interest due to its promise to substantially increase the power efficiency of integrated circuits by lowering the requirements on the precision of the calculations of a circuit. For a circuit in Approximate Computing, results that stay within a confidence interval can be sufficient as long as the power consumption can be reduced by tolerating this in-determinism. Not all applications are suitable for this approach. Especially applications from the signal and image processing domain are applicable, due to their intrinsic tolerance to imprecision. Approximate Computing proposes to reduce the power consumption for instance by removing, or switching off parts of a circuit that are only required for unnecessary high precision. The difficulty of applying the approach is to reliably and efficiently determine which parts of the circuit can be approximated, given the quality requirements of the application. Another approach in order to save power, is to over-scale the supply voltage and tolerate the appearance of timing faults. In this case it is not only necessary to determine which parts qualify for an approximation but also the degree of approximation that can be tolerated has to be determined.

In the prior art, approximations are mainly performed by a manual inspection and modification of the circuits. Only a few automated approaches exist, which limits the practical applicability of Approximate Computing. Lengthy simulations at gate-level are usually used to analyze the behavior of approximated circuits. Due to this limitation, often the proposed approximations are limited to combinational logic, i.e. sub-blocks of circuits, like arithmetic units.

In this work an approach is presented trying to overcome these limitations. Many steps of the approach are automated in order to simplify the process of the approximation. Approximations in this work are not limited to combinational sub-blocks, but can instead be applied to complex sequential logic. In order to overcome the lengthy gate-level simulations an intermediate approximation step, at register-transfer level, is used to analyze the behavior of the approximated circuits. In order to model a variety of approximation techniques, probabilistic fault injections are introduced. FPGA-based, accelerated, emulation is employed to perform the fault injections and to improve the runtime of the approximation. In order to apply voltage over-scaling without the need of lengthy simulations at transistor level, an analytical analysis approach is presented. This approach builds up on top of the results gained at the register-transfer level analysis and allows to efficiently and reliably apply voltage over-scaling to sequential circuits. With the help of a variety of case studies, the individual steps of the presented approach are demonstrated and validated.

Zusammenfassung

Approximate Computing ist ein Paradigma, dass in letzter Zeit vermehrt Interesse in der Forschung geweckt hat, bedingt durch das Versprechen den Stromverbrauch von integrierten Schaltungen substantiell zu reduzieren. Für eine Schaltung in Approximate Computing ist es unter Umständen egal, ob das berechnete Ergebnis nur innerhalb eines Konfidenzintervalls bleibt, solange durch die Inkaufnahme dieser Ungenauigkeit der Stromverbrauch gesenkt werden kann. Nicht jeder Typ Schaltung ist geeignet für eine Approximierung. Besonders Schaltungen aus dem Bereich der Signal- und Bildverarbeitung scheinen geeignet zu sein, da diese eine natürliche Toleranz gegenüber Fehlern auszeichnet. Approximate Computing schlägt vor, den Stromverbrauch zum Beispiel durch das absichtliche Entfernen, bzw. Abschalten von Schaltungsteilen zu erreichen, die nur für unnötig hohe Präzision zuständig sind. Die Schwierigkeit dabei liegt darin, zuverlässig und effizient zu bestimmen, welche Teile approximiert werden, wenn gewisse Qualitätsanforderungen an die Anwendung nicht unterschritten werden dürfen. Ein anderer Ansatz den Stromverbrauch zu reduzieren, ist es, die Versorgungsspannung soweit zu skalieren, so dass unter Umständen Zeitverletzungen in sequenzieller Logik auftreten können. Bei diesem Ansatz ist es nicht nur notwendig zu bestimmen, welche Teile approximiert werden dürfen, sondern auch noch den Grad der Approximierung, der toleriert werden kann.

Im Stand der Technik werden Approximierungen meistens mit Hilfe manueller Analyse und Modifikation realisiert. Es sind nur wenige Ansätze bekannt, bei denen ein automatisierter Ansatz durchgeführt wird, was die praktische Anwendbarkeit vom Approximate Computing deutlich reduziert. Meistens sind zeitaufwendige Simulationen auf Gatterebene nötig um die Schaltung hinsichtlich der möglichen Approximierungen zu untersuchen. Durch diese Einschränkung sind approximierte Schaltungen oftmals auf kombinatorische Blöcke, kleine Teilschaltungen, wie zum Beispiel arithmetische Einheiten, limitiert.

In dieser Arbeit wird versucht, diese Einschränkungen zu überwinden. Viele Schritte des präsentierten Ansatzes sind automatisiert um den Prozess der Automatisierung zu erleichtern. Die Approximierungen sind nicht limitiert auf kombinatorische Unterblöcke, sondern können stattdessen auf gröse sequentielle Schaltungen angewandt werden. Um die zeitaufwendigen Simulationen auf Gatterebene zu umgehen, wird ein Zwischenschritt auf Registertransferebene eingeführt, um das Verhalten der approximierten Schaltung zu analysieren. Um eine Vielzahl von Approximierungstechniken zu modellieren, wird für die Analyse eine probabilistische Fehlerinjektion benutzt. Eine FPGA basierte, beschleunigte, Emulation um die Fehler in die Schaltung zu injizieren sorgt dafür, dass die Analyse schnellstmöglich durchgeführt werden kann. Um eine Approximierung durch Versorgungsspannungsüberskalierung, ohne den Einsatz von komplexen Simulationen auf

Transistorebene, zu bewerkstelligen, wird ein analytischer Ansatz vorgestellt. Dieser Ansatz baut auf den Ergebnissen aus der Registertransferebene auf, und erlaubt auf zuverlässige und effiziente Art und Weise die Approximierung von integrierten Schaltungen durch Versorgungsspannungsüberskalierung. Mit Hilfe einer Vielzahl von Fallstudien, werden die einzelnen Teilschritte des Ansatzes demonstriert und validiert.

Preface

I want to thank Walter, Andreas and Thomas, not only for giving me the opportunity to write this thesis but also for their valuable input and support throughout my time working at the institute. I want to thank Stefan who helped me a lot with the netlist manipulation presented in this work. Without his help, this work would not have been possible. Special thanks go to my wife for always supporting me and for giving me the time to finish the thesis.

The Hague, December 2016
David May

Contents

List of Figures

List of Tables

Acronyms

ALU	Arithmetic Logic Unit.
ASIC	Application Specific Integrated Circuit.
AST	Abstract Syntax Tree.
AXI	Advanced eXtensible Interface Bus.
BER	Bit Error Rate.
CMOS	Complementary Metal Oxide Semiconductor.
CPU	Central Processing Unit.
DAB	Digital Audio Broadcast.
DCT	Discrete Cosine Transformation.
DSP	Digital Signal Processing.
DVB	Digital Video Broadcasting.
DVFS	Dynamic Voltage Frequency Scaling.
EDP	Energy Delay Product.
FFT	Fast Fourier Transformation.
FIFO	First-In First-Out.
FPGA	Field Programmable Gate Array.
HDL	Hardware Description Language.
ISA	Instruction Set Architecture.
LDPC	Low Density Parity Check.
LFSR	Linear Feedback Shift Register.
LTE	Long Term Evolution.
LUT	Look-Up Table.
MBU	Multi Bit Upset.
MIMO	Multiple Input Multiple Output.
MOSFET	Metal Oxide Semiconductor Field Effect Transistor.
NMOS	N Channel Metal Oxide Semiconductor.

P&R	Place and Route.
PCI	Peripheral Component Interconnect.
PCMOS	Probabilistic CMOS.
PE	Probabilistic Element.
PMOS	P Channel Metal Oxide Semiconductor.
PPA	Power Performance Area.
PPAR	Power Performance Area Reliability.
PSNR	Peak Signal to Noise Ratio.
PTM	Probabilistic Transfer Matrix.
QEC	Quality Evaluation Circuit.
SEL	Single Event Latchup.
SET	Single Event Transient.
SEU	Single Event Upset.
SPICE	Simulation Program with Integrated Circuit Emphasis.
SQCC	Sequential Quality Constraint Circuit.
UMR	Universal Multi Resource.
VHDL	Very High Speed Integrated Circuit Hardware Description Language.

1 Motivation

Two main trends are dominating the development of integrated circuits over the last decades. The first one is the unabated tendency for shrinking CMOS feature sizes. The motivation for this is to realize more functionality on the same or even smaller chip size at faster clock frequencies. And the second one is the ever increasing demand for low-power applications, inevitable in our mobile society [1]. Both trends are closely connected to each other. Unfortunately, both of these trends don't come for free. They both have in common that they, if driven much further, negatively affect the overall performance of an integrated circuit. Scaled-down technology nodes result in a hard to manage manufacturing process which again results in complex process variations on the IC. That scale-down makes a circuit even more perceptible to the very same variations and to soft-errors due to smaller threshold voltages [2]. Furthermore, in today's designs the chip functionality is usually not limited by the area but often constrained by a limited power budget [3, 4]. With increasing chip density and, at the same time increasing operating frequencies, power dissipation is becoming a serious issue that has to be considered in the design phase. Similar problems can be observed when applying voltage scaling, the

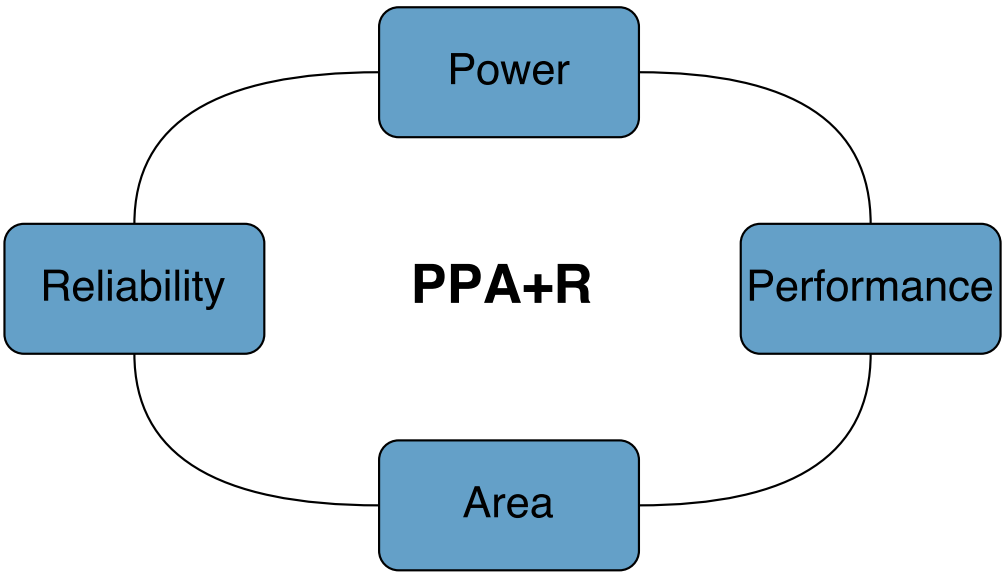


Figure 1.1: The extended "PPA" tradeoff of today's IC designs between power, performance, area and reliability

primary technique for saving power in CMOS circuits. Voltage scaling results in a degraded performance of the circuit due to the decreased speed of MOSFETs, operated at

a lower voltage. Usually, the performance loss is not acceptable for most applications as performance requirements are becoming constantly more and more demanding. However, even for those applications where voltage scaling can be applied, the gain is limited. The margin between supply voltage and threshold voltage is becoming so small to that logic gates are becoming more and more perceptible to small variations and noise in the circuit and in power supplies but also to soft-errors due to radiation. Hence, IC designers have to decide in which direction a circuit has to be optimized. The existing design trade-off between performance, power and area, sometimes referred to as the *PPA* trade-off, extends by the factor "reliability", as shown in Figure 1.1. It seems that a circuit cannot be fast, have a low power consumption, a small footprint and be reliable at the same time. A fast circuit needs, in order to operate reliable and correct, a sufficiently high supply voltage. And in reverse, a circuit operated at low power cannot be fast. Small circuits can usually not spend the area for redundancy mechanisms which would be required to provide sufficient reliability for circuits operated at near threshold or regular circuits e.g. in space applications. And finally, the reliability, as seen before, is depending on the power and area budget of a circuit. A reliable circuit can basically neither be small, nor power efficient.
However, recently one approach has appeared in the literature trying to tackle this problem, namely "Approximate Computing". The main idea of Approximate Computing is to relax the need for fully precise or completely deterministic operations [5], in order to substantially improve the energy efficiency. That means, Approximate Computing proposes to tolerate faults or imprecision in a circuit in order to save power. Various means on how the power can be saved exist and they vary a lot. The two most common approaches are *aggressive voltage over-scaling* and *functional approximation*. In the former case, the supply voltage is scaled down so far, so that timing violations occur in the critical paths of the circuit. When recalling the formula for the dynamic power consumption in CMOS devices, as shown in Equation 1.1, one can see that this method is ideal for saving power due to the quadratic relation of the supply voltage V_{dd} on the power consumption.

$$P_{dyn} = \alpha_{01} f_{clk} C_{load} {V_{dd}}^2 \tag{1.1}$$

The latter method, functional approximation, instead proposes to switch off or remove those parts of the circuit that are not absolutely required for the operation, respectively the desired quality of the calculated result. The pruning can be performed either dynamically for instance by clock or power gating techniques or statically at synthesis time. Clearly, this method promises to save even more power, as the gates are completely switched off and therefore also the static power consumption is tackled.
Naturally, not all applications are suitable for Approximate Computing. Safety critical applications for instance, can never be approximated. There are mainly two types of applications that seem to suitable for such an approach. The first class of applications are those that have an *imprecision tolerance*. These are e.g. applications that somehow interfere with the human perception. For instance applications from the image, video or audio processing domain. The second class of applications are those that have an *intrinsic reliability*. These are applications that anyhow have to deal with noisy or imprecise data and know how to deal with it, like signal processing applications in wireless

communication systems. For such systems it might be irrelevant whether the imprecision in the processed data is coming from the communication channel or from the hardware. The full potential of Approximate Computing can be unlocked if the approximations are applied dynamically. Compared to state of the art optimization mechanisms, e.g. like fixed-point arithmetic, which is usually applied statically at design time or in the best case at synthesis time, Approximate Computing proposes to be applied at run-time. Based on actual conditions or the actual need, the precision of the computed result, and hence its power efficiency, can be scaled. In order to elaborate these relations more in detail, in the following two motivational examples are given. Consider any arbitrary wireless communication system. Such systems are usually working with the same precision independent of the quality of the received wireless signal. These systems are designed to deliver fault free data at the output of the decoding chain, no matter if the signal-to-noise ratio is high, or if it is low. However, in case the channel quality is very good, the decoder would not be required to operate with the same high precision as in a bad channel environment in order to deliver the same output bit error rate. For instance, when looking at a convolutional decoder, such as a *Viterbi decoder*, it could be operated with less precision while consuming less power. Alternatively, an iterative decoder could be operated with less iterations. If the receiver is close to the transmitter, it could switch to a resource- and power-saving, but less precise state, as the signal quality itself is good enough to deliver a fault-free data output. However, if the receiver is moving away from the transmitter, and if the channel quality degrades, it could switch on the "precision" of the decoding chain, so that the output bit-error-rate remains constantly low.
The precision of an application could also be adapted based on the actual need in order to save power either preemptive or reactive. For instance in live video streaming the video feed is usually compressed in order to reduce the data rate. The compression is often done using dedicated hardware accelerators, due to its complexity. However, even hardware accelerated, this is a complex and power consuming task. For battery-powered mobile applications, it might be conceivable to reduce the precision of the compression, hence the resulting video quality, in order to increase the energy efficiency when running out of battery. With today's techniques it is already possible to reduce the resolution or quality (e.g. based on macroblocks per second) which can also be seen as an operation trading precision with power consumption. Originally however, these features are designed to reduce the data-rate. Again, Approximate Computing introduces the adaptivity into the field. Furthermore, resolution and quality profile can remain the same. If, e.g. during a live transmission, the mobile device is running out of battery, it could reduce the precision of the encoder. This in turn could allow to continue the stream, while accepting some artifacts in the stream, that are barely noticeable by the user.
Clearly, some existing optimization techniques and Approximate Computing can go hand in hand with each other, and can support each other. Some of them are closely related. Approximate Computing is a field that can, and has to be, applied on all levels of abstraction in order to get the most benefit of it. In Section 2.2, the various possibilities of approximation will be evaluated. However, the main focus of Approximate Computing up to now and its novelty is the approximation at hardware level. Approximate Computing tries to loosen the restriction on "Boolean equivalence" of the hardware, i.e.

the need for predictability of the hardware. This work focuses on the approximation of generic sequential integrated circuits. Previous works, operating on hardware level, as we will see in Section 2.3, either do not consider sequential hardware and do not propose a generic approach, but only application specific optimizations. Whereas this work researched the possibilities, difficulties and limitations to apply "Approximate Computing" to any existing circuit in a generic manner. The benefit of a generic approach is that most of the approximations can be automated, as much less specific knowledge about the circuit and its functionality is required. This is very important in order to make this paradigm widely applicable. The difficulty is to develop approximation algorithms that can be, on the one hand, generally applied to any kind of circuit and, on the other hand, are efficient in terms of approximation at the same time. Approximation algorithms have to determine the location where approximations can be made within a circuit and the degree of approximation that is possible for a desired quality constraint. Furthermore, as we will see later in detail, approximation speed, i.e. the time it takes to analyze and approximate a circuit, is also very important and hard to maintain. These considerations were influencing the decisions throughout most of the thesis.

2 Introduction

In order to perform the approximation of integrated circuits, one first has to understand the basic relations between imprecision in a circuit and its power consumption. In this chapter the fundamentals of CMOS power consumption and how approximations can be used to reduce it will be explained. A comprehensive introduction into the domain of Approximate Computing will be given. Furthermore, the different domains where Approximate Computing can be applied, the different types of approximation, circuit reliability in general, as well as metrics to measure approximation will be presented. Finally, a detailed overview about prior and related works will be given.

2.1 Power Consumption Dilemma of Integrated Circuits

The main goal of the techniques presented in this thesis is to reduce the overall power consumption of integrated CMOS circuits. Even though some approaches can be also used to reduce the area of an integrated circuit or increase the clock frequency, as we will see later, this can be seen more as a secondary optimization goal.
The total power consumption of CMOS circuits can simplified be expressed as:

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}}, \tag{2.1}$$

the sum of the dynamic power consumption and the static power consumption, where the former consumes power only when actually "switching" and the latter at any time when connected to a supply voltage.

2.1.1 Dynamic Power Consumption

Dynamic power consumption is defined as the portion of the total power consumption that originates from switching the state of a circuit, e.g. a logic gate. It itself again consists of two portions.

Short Circuit Power Dissipation The short circuit current is the current that flows from supply voltage via the pull-up path of a gate (PMOS) through the pull-down path (NMOS) to ground, in case both blocks, pull-up and pull-down are conducting at the same time. This can happen for instance when the input voltage V_{GS} is not using the full voltage hub between V_{DD} and GND, resulting in one MOSFET operating in saturation, and the other in the linear region. However as this effect is arising from miss-dimensioning of the circuit or variation of the fabrication it cannot be counted as a dynamic component. However, short circuit current does also arise, when regular

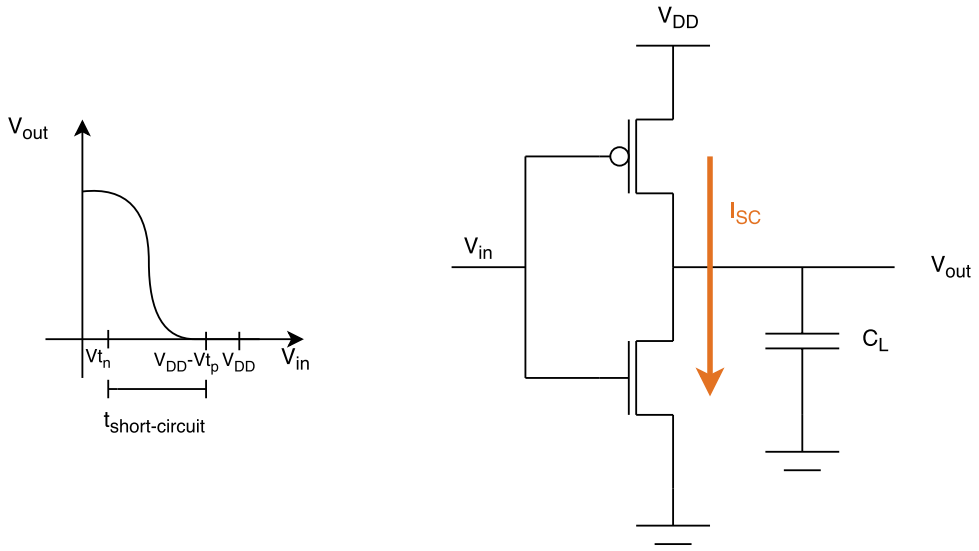


Figure 2.1: Short circuit power of a CMOS inverter

loading or unloading a load capacitance. This effect is shown in Figure 2.1. For instance in a CMOS inverter, when $V_{IN} = V_{DD}/2$, both NMOS and PMOS are operating in saturation. It hence depends on the slope of the input signal how long both transistors are in saturation. This interval is defined as $t_{\text{short-circuit}}$ in the Figure. Furthermore, it also depends on the ratio $v = V_t/V_{DD}$, where V_t corresponds to the threshold voltage. For values $v > 0.5$ short circuit current is eliminated, as NMOS and PMOS cannot be switched on at the same time. In today's technology generations V_t is in the range of $0.3V$ while the supply voltage V_{dd} is at about $1V$ [3]. Hence, short circuit current is negligible.

Capacitance Switching Power Dissipation The main factor of the dynamic power consumption is generated by loading and unloading the load capacitance of a gate. The load capacitance consists of the gate capacitance of all gates that are connected to the output of a gate (fanout). Additionally, the capacitance of the wires, as well as the intrinsic capacitance between source and drain are contributing as well. Every time the load capacitance is loaded and unloaded a charge of $Q = C_L V_{DD}$ is transferred from V_{dd} to GND, as depicted in Figure 2.2. Hence the current that flows in each complete charge-discharge cycle can be calculated as:

$$I_{cap} = C_L V_{DD} f \tag{2.2}$$

The resulting power consumption due to loading and unloading the load capacitance can hence be calculated as:

$$P_{cap} = \alpha_{01} C_L {V_{DD}}^2 f, \tag{2.3}$$

where α_{01} corresponds to the switching factor, as usually a gate is not switching in every clock cycle. As the short circuit power dissipation is negligible compared to the power

dissipation resulting from un-/loading the load capacitance, one usually focuses on the latter in order to minimize the power consumption of integrated circuits. In the following

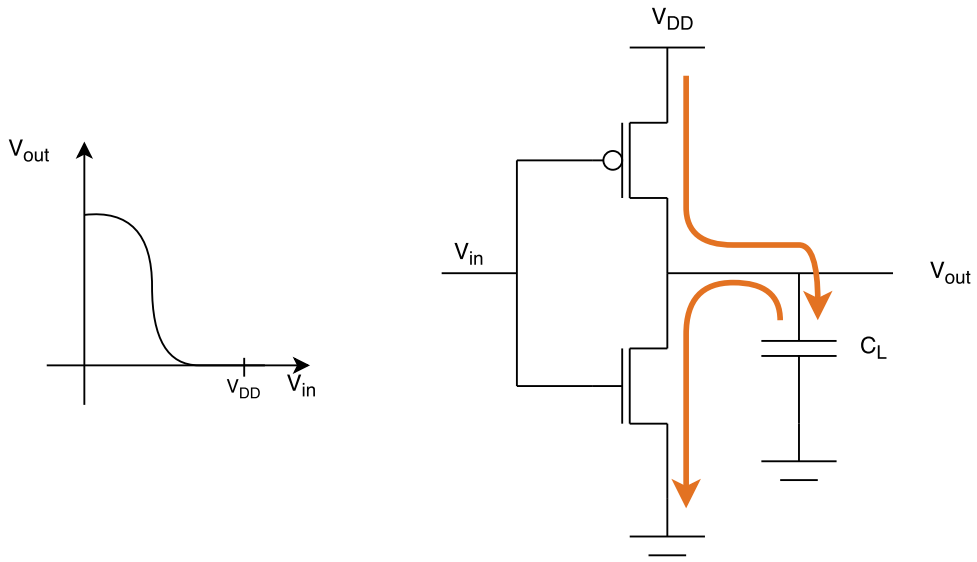


Figure 2.2: Capacitance switching power of a CMOS inverter

we will see, how each of the factors of Equation 2.3 can be used to reduce the overall power consumption of integrated circuits. Starting from the left, the first term that could be reduced in order to minimize the dynamic power consumption is the switching factor α_{01}. The switching factor is defined as the probability of a signal transition in one clock cycle. Reducing the switching factor by architectural changes is not simple and not intended, as this also correlates with the circuit efficiency. A circuit, by definition, cannot be efficient if it changes its state rarely. However, one very efficient technique in order to reduce the dynamic power consumption based on the switching factor is "Clock Gating". By disabling the clock of a circuit, the switching factor reduces to zero in sequential circuits. Applying this technique dynamically to sub-blocks of a circuit, that are not used all the time, can save a tremendous amount of power. Especially in the age of "Dark Silicon" this is a widely used technique. The second factor of Equation 2.3 is the capacitance C_L. Capacitance can be reduced by either reducing the gate capacitance C_g of a technology, or by reducing the wire capacitance of the circuit. The former can only reduced by introducing a new CMOS technology that usually results in a shorter gate length and hence a smaller gate capacitance C_L. However, a reduction of technology node sizes is a complex task that requires tremendous research effort. It is not a parameter the designer can choose. Additionally, today a reduction of the technology size is usually accompanied with an increase of parameter variations like threshold voltage, which requires a certain delta between V_{th} and V_{DD}, large enough to tolerate these variations. This in turn makes it difficult to save power by reducing the supply voltage. The third parameter is the supply voltage V_{DD}. Reducing the supply voltage in order to reduce the dynamic power consumption of an integrated circuit seems to be the primary choice due to the

quadratic relationship. However, the most important drawback is that the by decreasing the supply voltage of CMOS gates, the propagation delay is increasing. The propagation delay can be approximated as follows:

$$t_p = \ln(2) C_L R, \tag{2.4}$$

where R corresponds to:

$$R = \frac{V_{DS,p}}{I_{DS,p}} \approx \frac{1}{\beta(|V_{GS,p}| - |V_{th,p}|}, \tag{2.5}$$

In CMOS circuits $V_{GS,p}$ equals V_{DD}. Hence one can see that a reduction of the supply voltage increases the propagation delay. This again would require, in order to maintain the functionality of a circuit, to reduce the operating frequency of the circuit in order to meet the timing constraints. For most applications this performance loss is not acceptable. One approach to scale the supply voltage anyway is "Dynamic Voltage Frequency Scaling". This technique however is usually only applicable to CPUs. The idea of this technique is to scale the supply voltage in conjunction with the operating frequency depending on the actual work load. Hence, in case a CPU has a low workload, the operating voltage and frequency can be reduced in order to save power. When the workload increases, the voltage and operating frequency is increased as well in order to offer the performance required for the workload. This technique is widely applied in today's CPUs. In today's technologies where chip size is cheap, one can think of another method as well. Instead of dynamically adjusting the voltage-frequency operating point based on the workload, one could simply add another low-voltage CPU to the system. This slow, but power efficient CPU can then be used for simple computational tasks like background calculations, while the faster variant is used for peak workload. This technique is usually more power efficient than DVFS as the variants can be optimized exactly for their operating point. This heterogeneous technique is widely applied in today's mobile devices, where most of the workload can be seen as background tasks, and fast and responsive calculations are only required when the user interacts with the device.

Reducing the supply voltage close to the threshold voltage does not only impose a performance loss. Unfortunately, supply voltage noise is not scaling with the voltage. This effect is similar to the parameter variations as we have seen before. Usually, a certain "guard band" between V_{th} and V_{DD} is required in order to be robust against thermal noise and parameter variations. One possibility to overcome this problem could be to equally reduce the threshold voltage. However, this results in yet another problem. A reduced threshold voltage results in increased sub-threshold leakage, which in turn results in an increased, not negligible, static power dissipation, as we will see in following.

2.1.2 Static Power Consumption

The static power consumption is the power that is consumed even if no load capacitance is switched. It is therefore independent of the operating frequency f as well as the switching activity α_{01} of a circuit. There are multiple sources for static power consumption from

which the two most important ones will be shortly explained here. Simplified, the leakage current can be described as:

$$I_{\text{leak}} = I_{\text{sub}} + I_{\text{ox}} \tag{2.6}$$

where I_{sub} is the "subthreshold leakage" component and I_{ox} is the "gate leakage"component.

Subthreshold Leakage Subthreshold leakage is defined as the current that flow between source and drain, when the MOSFET channel is supposed to be off. This can happen when the channel is in "weak inversion". Even if the gate-source voltage V_{GS} is below the threshold voltage V_{th} free charge carriers can be observed between source and drain. These carriers can create a current mainly due to diffusion. The intensity of the weak inversion is depending on two factors. On the one hand the current is depending on the channel width W. In a wide channel the current increases while in a narrow one it decreases. And on the other hand the intensity is depending on the threshold voltage V_{th}. A smaller threshold voltage is drastically increasing the weak inversion as even small noise on the gate voltage will have a large effect on the inversion. The relations are again shown simplified in Equation 2.7 [6].

$$I_{\text{sub}} = K_1 W e^{-V_{th}/nV_\theta}(1 - e^{V_{\text{dd}}/V_\theta}), \tag{2.7}$$

where K_1 and n are experimentally derived values and V_θ is a value depending on the temperature.

Gate Leakage During the last decade a new leakage source arose, the gate leakage. Gate leakage became the dominant factor of static power consumption for technology generations $<$ 65nm [7]. Due to the fact that the oxide is becoming thinner every technology generation, more and more electrons can tunnel through the oxide. Gate leakage current can be observed between gate-source, gate-drain, gate-channel-source, gate-channel-drain and gate-bulk, as shown in Figure 2.3.

$$I_{\text{ox}} = I_{GS} + I_{GD} + I_{GCS} + I_{GCD} + I_{GB} \propto \frac{1}{t_{\text{ox}}} \tag{2.8}$$

The effects of gate leakage are much worse in NMOS than in PMOS.
We can see from Equations 2.7 and 2.8 that basically four possibilities to reduce I_{leak} exist. In order to reduce I_{sub} one can either reduce the supply voltage, the threshold voltage or the channel width. This, as we have seen before, results in a usually not acceptable performance loss. In order to reduce I_{ox} one can reduce the dielectric thickness. However, in order keep the influence of short-channel MOSFET effects low, the oxide thickness has to be scaled proportionally to the channel length. New techniques like high-k insulators are improving the situation due to their superior insulation capabilities. However, in general one can see that if MOSFETs shall be fast, the leakage current is inevitably increasing. The same can be seen for the switching current. A reduction of the power consumption results in a decreasing performance either resulting in a decreased operating frequency or in timing violations.

Therefore, the dilemma of integrated circuits built in their current technology is that they cannot be fast, reliable an energy efficient at the same time.

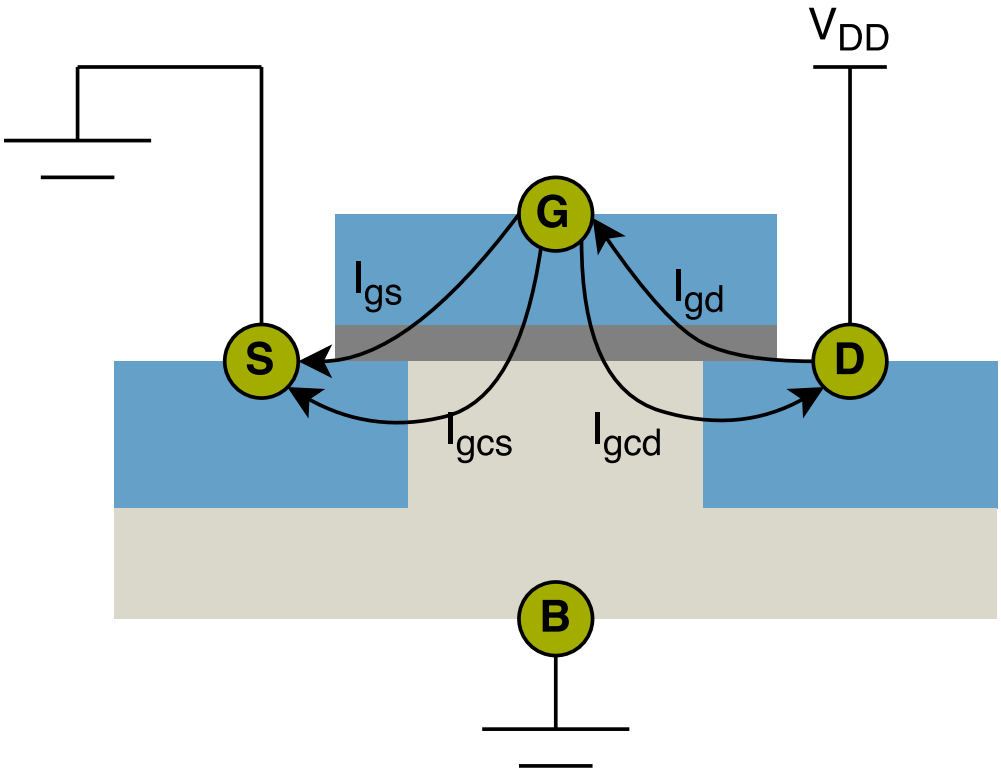


Figure 2.3: The various forms of gate-leakage in CMOS devices

2.2 Approximate Computing

Approximate Computing is a novel paradigm for energy-efficient digital systems that has considerably attracted interest over the last decade [5, 8, 9]. The core principle of Approximate Computing is to trade off power with precision, as visualized in Figure 2.4. It hereby relies on the ability of many applications and the end-user itself to tolerate a loss of quality or imprecise computational results. Clearly, the idea of computing results that are good enough for a certain application, instead of computing with unnecessary high precision is not new. For instance, any lossy compression system for video or audio data is applying the same principle, even though the motivation is slightly different. However, in the last years an increasing number of application fields, at various points in the design flow, have discovered the paradigm for themselves. A variety of techniques have been started capturing under the hood of Approximate Computing. Especially on the hardware level, tolerated non-determinism is a novelty.

There are two other noticeable research fields that are related to Approximate Computing. One is “Probabilistic Computing” and the other is “Near-Threshold Computing”, which will be shortly explained in the following.

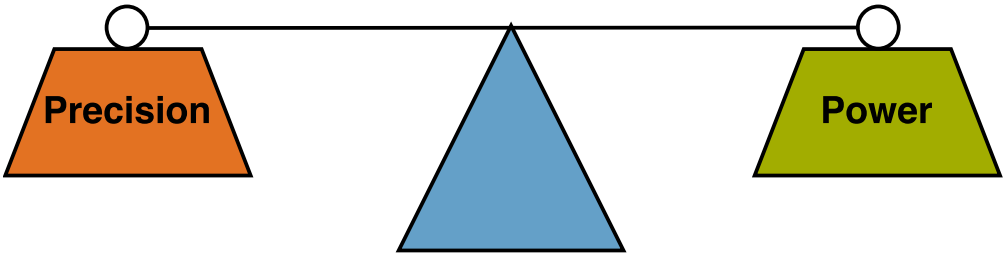


Figure 2.4: Approximate Computing: trading-off power consumption with precision

Near-Threshold Computing Near-threshold computing proposes to apply voltage-scaling in order to reduce the power consumption. Compared to regular voltage-scaling techniques it is trying to aggressively scale the voltage in the range of the CMOS threshold voltage. The concomitant performance loss is tried to be overcome by massive parallelism and modern design techniques like 3D integration. Near-threshold computing is also aware of functional failures. However, compared to Approximate Computing, it tries to completely avoid them by design changes or redundancy mechanisms. A comprehensive overview can be found in [10] and [11].

Probabilistic Computing Probabilistic computing proposes to exploit the probabilistic nature of circuit elements. Palem et al. developed the notion of probabilistic binary switches under the influence of thermal noise, i.e. operated at low-voltage [12]. Based on these switches probabilistic CMOS circuits (PCMOS) have been proposed, developed and the approach been evaluated [13, 14, 15, 16, 17, 18]. The principal idea of probabilistic computing is the same as of Approximate Computing, namely trading off power consumption with accuracy. The main difference however is, that Approximate Computing is usually using deterministic circuit elements to generate imprecise results. In contrast, probabilistic computing usually uses non-deterministic elements, and is hence considering another fault model. The methodology developed in this work is also not limited to deterministic CMOS elements. Due to the probability-awareness of the presented approach, hence assigning error probabilities to circuit elements, it can be applied as well in the domain of probabilistic computing. Indeed, techniques like circuit pruning, which can be applied for Approximate Computing, have their origin in probabilistic computing [19].

The consideration of imprecisions and non-determinism in digital circuits is going back to the origin of digital design using electromagnetic switches and vacuum tubes. Von Neumann himself, was searching for ways to build reliable circuits from unreliable elements [20]. With the rise of semiconductors and improving process technologies, the problems of unreliable building blocks seemed to disappear. For safety-critical or radiation-intense applications, reliability, of course, has always been an issues. However, means like error correction or redundancy could be exploited to guarantee a functional correctness, at least to a certain degree. However, now as it seems that the end of the classical Moore's

law has been reached [21], it looks like the problems are coming back. Increasing parameter variations, due to shrinking feature sizes, as well as ultra low-power operation tend to make the building blocks again unreliable. At least if the, so called, "happy scaling" continues. Approximate Computing is still assuming that the building block are working correctly. It is trying to directly tackle the power density problems of today's circuits. With the end of classical Dennard scaling [22] about 10 years ago, new technology generations do not lead to a constant frequency increase or a decrease of the power consumption, respectively. Even the shift to the multi-core era is not solving the problem of the high power density, which has led to dark silicon on the chip. In contrast to dark silicon management approaches, Approximate Computing is trying to reduce the power consumption on a chip, while preserving the speed and the functional density. This is achieved by tolerating imprecisions and non-determinisms in the circuit, for instance due to aggressive voltage-scaling or by removing parts of a circuit that are not required for an approximate, yet sufficiently correct, result. The main problem to be solved can be regarded as to find a distribution of approximations made within the circuit as large as possible, while at the same time meeting the constraint on the quality at the output of the circuit.

"Voltage over-scaling" and "circuit pruning" are the main approximation techniques considered in this work. While the former is going in the direction of probabilistic computing, the latter is going in the direction of Approximate Computing. However, as already mentioned, most methods developed to realize such approaches can be shared between these two fields. In the following the types of approximations and sources of imprecisions, hence the fault model, will be described more in detail.

2.2.1 Approximation Types

As already mentioned, Approximate Computing can be applied on multiple levels of abstraction. Hence, the actual point where a circuit is approximated can differ. For instance, in case the approximation is done for an application on algorithm-level, for instance by applying fixed point arithmetic instead of floating-point operations, the approximation is solely based on an algorithmic modification. No changes are made on the hardware level. However, for instance, when removing the least significant bit calculations of a floating-point unit at register-transfer level, hardware is indeed modified, while the initial algorithm of the application remains untouched. This work is focusing on approximations at hardware level. As it has been shown in previous work [23], bit-flips can be used to model a variety of physical sources of faults. Figure 2.5 illustrates this relation. All of these faults in the lower half of the hourglass concentrate in a potential error as bit-flips. As we will see in the following chapters in detail, the abstraction to RT-level, at least for an initial analysis of a circuit, gives a balanced trade-off between imprecision due to abstraction and analysis speed-up. When approximating an application one is not only interested in where approximations can be applied but also to what extent. Therefore, in this work bit-flip probabilities at register-transfer level have been introduced. This allows to model the probability of occurrence of the individual approximations, for instance due to masking effects.

Hence, the fault model for the initial analysis and approximation of a circuit in the presented methodology are bit-flips at registers that appear with a certain probability. For an exact analysis the information gained at RT-level has to be transferred down to gate-level, depending on the applied approximation techniques, in order to model the corresponding effects. In the following, the approximation techniques considered in this work will be presented in detail.

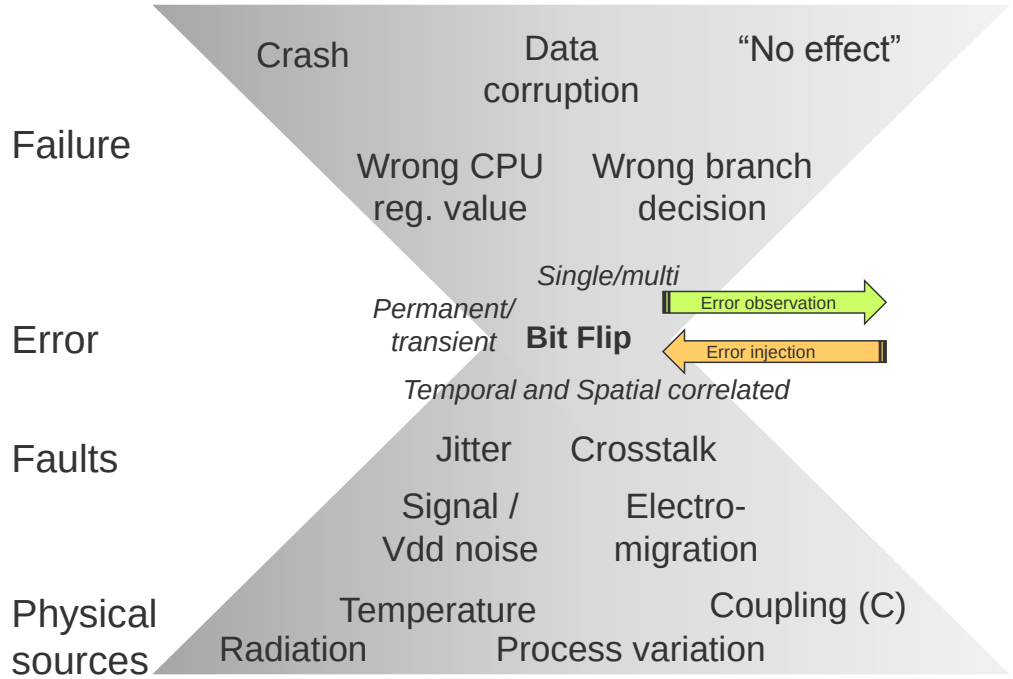


Figure 2.5: Bit-flip probability as the mean to model a variety of faults and approximation techniques, respectively [23]

Physically Unreliable Circuit Elements The first type of approximations considered in this work is the use of *Unreliable Circuit Elements*. Clearly, these are not approximations per se, but anyhow result in a non-deterministic behavior of a circuit. Unreliable circuit elements are the main source of imprecision in the domain of *probabilistic computing*. There are various reasons why circuit elements, usually logic gates, can be unreliable. The first category are unreliable circuit elements due to "noise-based" faults [24]. The ever increasing demand for fast and energy-saving, hence small, devices led to this problem. Due to the reduced supply voltage and the increasing integration density, noise immunity is becoming difficult to achieve. This results in very tight noise margins, leaving no room for further optimization. Thermal noise in the circuit or voltage regulators [25, 26] can therefore actually limit a further decrease of supply voltage. Similar behavior can be seen with parameter and process variations in the circuit [27]. For a further scale-down of the MOSFETs either the technology process has to be improved, or one has to deal with the resulting non-determinism.

Another source of unreliable circuit elements are *soft errors* due to radiation. Alpha-

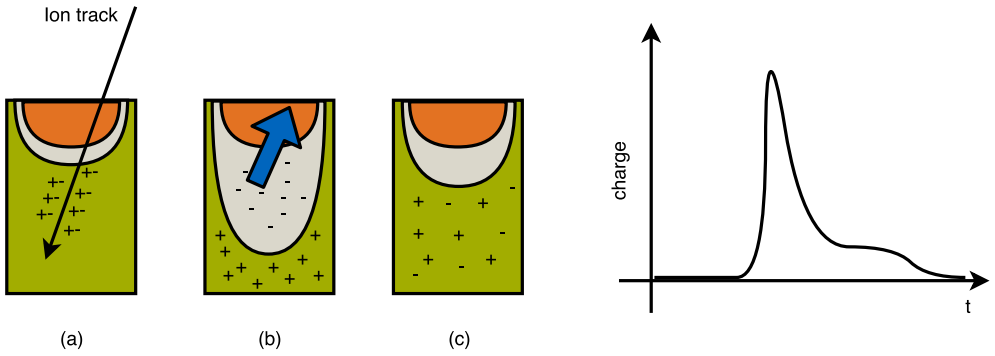


Figure 2.6: Charge generation and collection phases in a reverse-biased junction and the resultant current pulse caused by the passage of a high-energy ion. [28]

particles as well as high-energy and low-energy rays are the dominant sources of soft errors due to cosmic radiation [28]. These mechanisms are generating ions, either directly or indirectly, in the silicon, as shown in Figure 2.6. The ion's impact results in a track of electron hole pairs (a). Afterwards, the carriers get rapidly collected, resulting in a current peak (b). The impact ends with a diffusion of the electrons and holes (c). Depending on the amount of energy induced by the particle strike and the resulting current pulse, a logical value in the circuit can possibly be flipped. The amount of energy required to induce a soft-error in a MOSFET, usually denoted as Q_{crit}, the critical charge, is depending on technology and type of the gate, as well as the location of the strike. Additionally, the time instant when the particle is striking is highly influencing the impact of the strike. If the energy is high enough, the value of a net can change from "0" -> "1". This effect is called a "single-event transient" (SET). This SET is then possibly latched into a register, resulting eventually in a "soft-error", also called "single-event upset" (SEU). Clearly, the transition from SET to SEU is depending on the Boolean function of the combinational logic, the actual state of the logic and the time instant when the particle strikes. It is also possible that multiple particles strike the circuit at the same time. However, the occurrence of multi-bit upsets (MBU) is rare. On the other hand it not unlikely that a single-event upset results in multiple errors not only over time but also at several locations. Technically, it is also possible that the particle energy is large enough to permanently damage a circuit. This can "single-event latchup" (SEL) results in a permanent malfunction of the circuit. A SEU instead is reversible. Hence, in the next clock cycle the error is overwritten by a new value.

Compared to traditional Approximate Computing, in this case one is not only aiming for a reduction of the power consumption. Approximations with unreliable elements due to soft-errors can also be used to reduce the logic overhead that would be regularly introduced by redundancy or error correction mechanisms. Hence, by omitting them not only area on the chip can be saved but also the static and dynamic power consumption of these blocks.

The difficulty of exploiting this behavior as an approximation technique lies clearly in the estimation of the error rates. For noise-based faults, as well as for soft-errors it is nearly impossible to analytically predict the error rates, as the physical context is very complex. However, a lot of research has been made in the classical reliability domain over the last decades to estimate the error rates, or at least to give worst-case rates. These values can be used to describe the non-determinism of the elements and use them for Approximate Computing.

Functional Approximation Functional approximation is the most common approximation technique. It proposes the intentional modification of a circuit in order to save power or area, while making the operation imprecise. Various forms of functional approximation exist, there is no common approach. First of all the modifications made, can be either temporal or permanent. A permanent modification clearly results in the highest savings concerning power and area. This technique is usually referred as "Circuit Pruning", hence parts of the circuit get removed, in order to implement a less power consuming but less precise algorithm. A simple exemplary approximation of a 1-bit full adder is shown in Figure 2.7. By simply removing the marked AND gate, the calculation

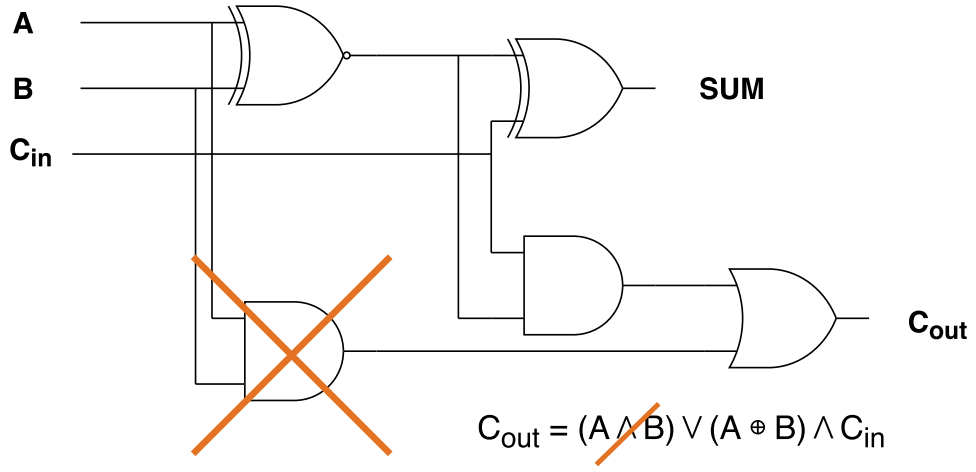


Figure 2.7: Simple functional approximation at gate level of a 1-bit full adder

of C_{out} approximates. A subsequent change in order to compute more precise results, e.g. due to changed precision requirements, is not possible. For ASIC technology, once the circuit has been approximated, it cannot be reverted. On re-programmable devices, like FPGAs, the situation is different. For changing demands on the precision, a different bitstream with differing precision settings could be loaded on demand. A dynamic adaption of precision using functional approximation on ASICs is possible as well. This procedure is usually referred to as "Dynamic Bit-With Reduction". The idea is to change the precision of a calculation based on the actual demand of the application, by changing the bit-with precision. A simple example for bit-width reduction is shown in Figure 2.8 for a 4-bit adder. By removing either the complete lowest 1-bit adder or parts of it,

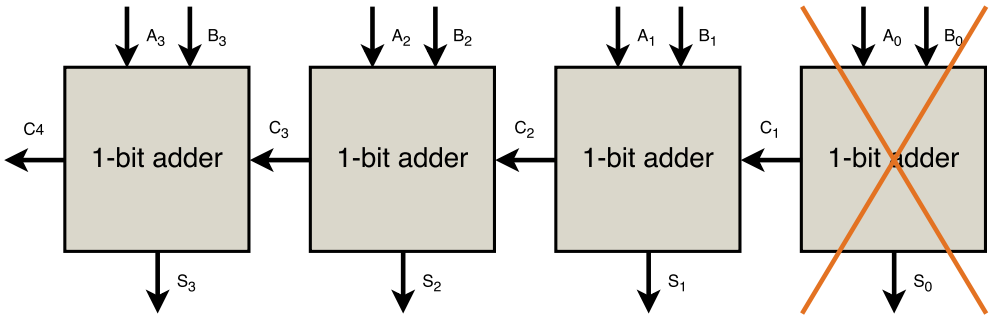


Figure 2.8: Bit-width reduction of a 4-bit adder

the resulting sum of A and B is approximated. Later in this work, when presenting the approximation methodology of this work in Chapter 4, one can see that circuit pruning is not limited to bit-width adaptation of mathematical calculations. The presented methodology considers each register as a potential candidate for dynamic approximation, in order not to impose unnecessary restrictions. The savings of a dynamic adaption are of course less than of static circuit pruning. Area cannot be saved at all, as no elements get removed. Power consumption can be saved either in form of dynamic power consumption, by gating the gates not needed or even as static power, by applying power gating to the unused gates. However, the main benefit of this approach is the adaptivity. Precision can be switched on and off based on the actual requirement of the application. If the application, for instance due to actual conditions, requires less precision, parts of a circuit can be switched off, and energy can be saved. If, however, the application at some point needs a highly precise result, the precision can simply turned on again, accepting a higher power consumption.
The difficulty, as with all approximation techniques, is to find out how pruning, or a bit-width adaption influences the overall behavior of a circuit. The analysis usually has to be fast and precise at the same time, and should be generally applicable to all kinds of circuits. Existing approaches are usually limited to the approximation of basic building blocks. In this work however, a methodology will be presented that allows to functional approximate complete circuits at once. The verification of these techniques is comparably simple, as fast functional simulations are sufficient. Similar, the power savings of static as well as dynamic modification can be easily estimated, by using power estimation tools. By generating simulation traces for realistic scenarios the switching factor of the individual nets can be calculated. By applying them to power estimation tools, a realistic estimation of the power consumption before and after approximation can be calculated.

Voltage & Frequency Scaling Voltage scaling is another main approximation technique considered in this work. As we have seen earlier, reducing the supply voltage of a circuit can result in large power savings due to its quadratic relation on the dynamic power consumption. However, as we have seen as well, the propagation delay of MOSFETs increases inverse proportional to the supply voltage. This relation is again visualized

in Figure 2.9. Depending on the supply voltage V_{dd} the operating point can be moved between power efficiency and performance. An optimal point can be found regarding these two criteria. The characteristic of a circuit are sometimes measured in form of the "Energy-Delay Product", which gives an indication about the power consumption and speed at certain supply voltage. When operating a sequential circuit with reduced supply

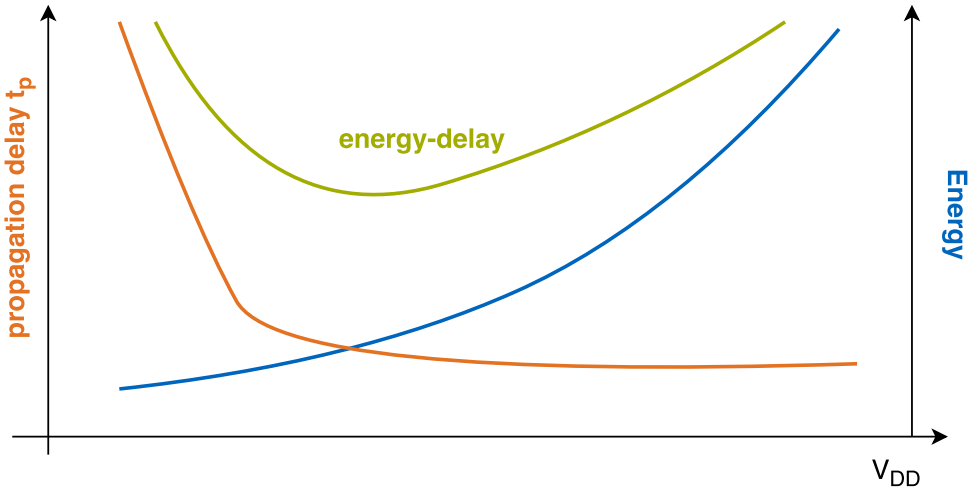


Figure 2.9: Visualization of the Energy-Delay product (EDP) of CMOS circuits

voltage and constant operating frequency, the propagation delay of the MOSFETs will be larger than at nominal voltage and the timing constraints might not be met anymore, i.e. setup times could be violated, as shown in Figure 2.10. While a transition on the longest path is arriving prior the setup time t_{setup} for voltage V_1 it is arriving within the setup time for a supply voltage V_2, and there is a probability that the transition is not sampled in the Flip-flop and hence can be seen as a bit-flip. The appearance of

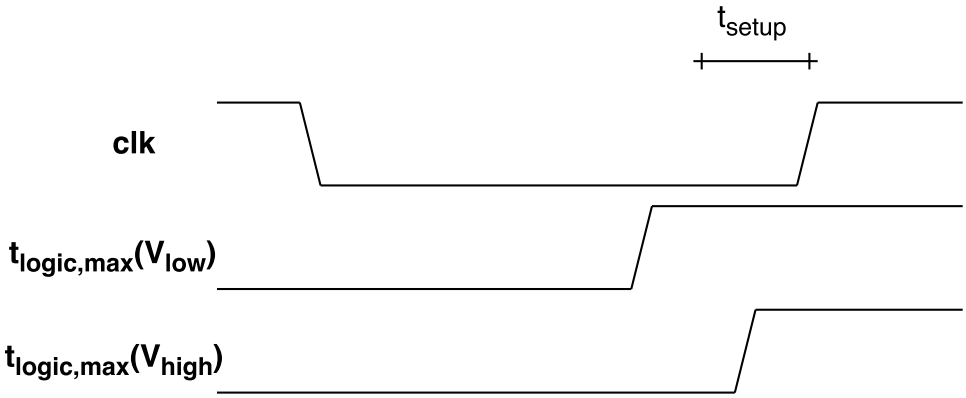


Figure 2.10: Timing violation in sequential circuits. The signal transition is not arriving in time at the flip-flop

timing errors depends on the circuit, its state and input transitions. Timing errors are comparable to soft errors, as they are of temporary nature and can be modeled with a bit-flip probability at a register. Like soft-errors, timing errors are usually critical. Their appearance can, dependent on location and time, result in a complete malfunction of the circuit. However, they don't have to necessarily result in a failure of the circuit as the value which is actually wrong can be masked, i.e. it is not read due to the Boolean function and state. The difficulty as we will see later is to determine if and when timing violations are critical and their influence on the overall functionality of the circuit. The goal of voltage over-scaling is to reduce the supply voltage of a circuit (or parts of it) in such a way that the power consumption is significantly reduced, but the influence of the resulting timing violations on the functionality of the application are minimal. Voltage over-scaling is closely related to frequency over-scaling. Operating a circuit at a higher frequency than the one determined in the timing analysis results in the same timing errors as when over-scaling the supply voltage. The optimization goal however in this case would be the circuit performance and not the power consumption. Nevertheless, the same methods developed throughout this work could be used to realize approximate frequency over-scaling. Similar to functional approximations, voltage over-scaling can be either applied statically or dynamically. In case of static voltage over-scaling the designer has to decide at design time the degree of approximation through voltage scaling. I.e. one has to decide how imprecise the final circuit should operate. Clearly, this can only be applied for circuits whose purpose is fixed, like hardware accelerators. For multi-purpose circuits like CPUs a static approximation is not suitable as the work is not known at design time, and the precision requirements are changing over time. The complexity of static voltage over-scaling however is much less than that of dynamic voltage over-scaling, as only one approximate operating point has to be determined at design time. The benefit of a dynamic approach is clearly that the circuit can adapt the current requirements on the precision and power consumption. The drawback however is not only the requirement for configurable power supplies. Considering that at design time a set of gates that can be operated at a reduced supply voltage V_1 and an approximate operating point $\mathrm{OP}_{\mathrm{approx}_1}$ (e.g. an error probability at an output pin) have been determined. For a second, more precise approximate operating point $\mathrm{OP}_{\mathrm{approx}_2}$, it might not be sufficient to simply increase the supply voltage of that voltage island uniformly. It could happen for instance that for that new operating point certain gates have to be operated at nominal voltage in order to meet the requirements on the precision, while other can still be operated at a scaled voltage. It would then be required, in order to have an optimal operating point, to reorder the voltage domains, which is of course very complex. Instead one could define at design time the voltage domains. These domains would then be fixed. The supply voltage of these domains however can change. The assigned supply voltage however has to be always the maximum one that has been determined for a set of gates and one operating point, even if some gates could be operated at lower frequency. This circumstance will be explained later in Chapter 5 in more detail. Another question that has to be answered at design time is how many voltage islands an approximate circuit can have. The more domains can be implemented, the more fine-granular the approximation can be made. More voltage domains however are complex to route and require external

Table 2.1: Truth table approximation of an adder [29]

Inputs			Accurate out		Approximate out	
A	B	C_{in}	Sum	C_{out}	Sum'	C_{out}'
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	1	0	1	0
0	1	1	0	1	1	0
1	0	0	1	0	0	1
1	0	1	0	1	0	1
1	1	0	0	1	1	1
1	1	1	1	1	1	1

power supply units. Therefore a trade-off has to be found between the efficiency of the approximation and the additional required area. Nevertheless, as we will see later, even a very small number of voltage domains can be sufficient to get a significant reduction of the power consumption.

Boolean Modification Similar to functional approximation, and in particular to the example shown in Figure 2.7, in this approach the existing circuit is optimized by using less components (e.g. gates or transistors). To judge the effect on the circuit objectively, again the primary outputs are observed. In order to do so, the differences between the modified design's truth table and the original design are compared. To keep the complexity of the optimization problem and the effects on the primary outputs low, the technique is normal applied to sub circuits that are supposed to bear a certain error tolerance. Ideally, these sub circuits are used frequently in the containing design. Note, that the approximate units not only have reduced number of transistors, but also the internal node capacitances are reduced [29]. This overall leads to a design with lower complexity and therefore lower static (leakage current) and dynamic (switched capacitance) power consumption. Another benefit of a circuit containing less transistors can be shorter critical paths that in turn allow reducing the supply voltage, without causing timing errors. A general tactic to find such a design can be to inspect the truth table for outputs that are correlated to a combination of a low number of other outputs or inputs. These correlations could be a hint for a simple logic combination of in- and/or outputs that can give the right result of the target output for most of the input combinations. An example can be seen in Table 2.1, where the approximation Sum' = B and C_{out}' = A have been made. Errors are marked in red. As with all approximations presented in this work, it is very important to respect their position and influence on the whole system under real operation circumstances [140]. Boolean modification is not further considered in this work as the methods developed in this work lead to the same benefits, while being at the same time more general and flexible.

2.2.2 Suitable Applications

As we have already seen in the motivation of this work, not all types of applications are suitable for approximation. This is one of the main limitation of Approximate Computing and it is very important to be aware of it and it must not be withheld. It has to be clear that an approximated circuit will always give a less precise result. This is a core principle of Approximate Computing. Hence applications that need a numerical correct result, will never be suitable for Approximate Computing. However it is important to keep in mind that the idea of Approximate Computing is to operate only less precise, not wrong. That means that ideally the user can still trust the result, at least up to a certain point, a confidence interval. This is also one important part of this work. The functionality of an application, hence its control flow, should never be affected. Nevertheless, control centric applications and of course safety critical applications cannot be used for Approximate Computing. Fortunately however, as we will see, the field of potential applications is still large. In general, applications suitable for Approximate Computing qualify themselves

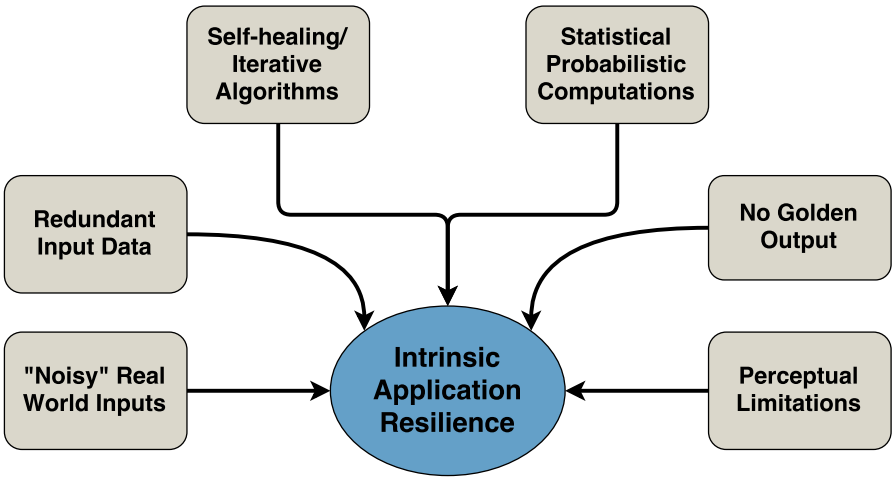


Figure 2.11: Intrinsic application resilience [8]

by having one or more of the following properties regarding the requirements on the result [5, 8, 30, 31]:

No golden result exists, a range of answers is acceptable This applies for instance for applications in the "search" & "recommendation" domain, where multiple answers are correct.

The golden result exists, but even a perfect circuit is unlikely to find it Applications like "Machine learning", "Pattern matching" do fall into this category.

The golden result is not necessary, a less-than-optimal result is sufficient A wide range of applications falls into this category, as on can see below. This category has been focused mainly in the scope of this work.

Usually applications are said to be suitable for Approximate Computing that have a *intrinsic imprecision tolerance* [32], which corresponds mainly to the last of the above mentioned category. A wide branch of applications falls into this category as shown in Figure 2.11.

Interaction with (limited) perceptual capability of humans This is the most common mentioned category of applications when talking about Approximate Computing. Due to the fact that the human perception and the sensory system is not perfect, minor differences and imperfection in application results that interfere with the human perception will not be noticed by the user. For instance, the physical reception of sound is limited to a range of frequencies. Humans normally hear sound in frequencies between approximately 20 Hz and 20,000 Hz while the upper limited decreases with the age [33], as shown in Figure 2.12. Hence, any audio information outside of these limits will not be noticed. But not only the resulting information at those frequencies is irrelevant, but also all calculations within the application that have been made in order to come to that result. This is an important fact to remember as it is responsible for the majority of potential power savings. The visual perception of humans is limited as well. First of all are the photoreceptor cells on the retina in the human eye noisy sensors. Hence, the image that gets forwarded to the brain always has a certain noise floor. As we all know this noise floor increases in low light situations. Furthermore, the human eye is only able to distinguish about 10 million colors [34], which is why usually 24bit are used as "true color" color depth. But also for the visual perception a certain contrast in the image is required in order to detect objects. Hence applications that interact with the human visual perception can be imprecise regarding noise up to a certain point without the user would notice any difference. But also certain frequencies of an image are not required as the human perception is not able to recognize them. The remaining human sensory systems, somatosensory system, olfaction and taste behave very similar, but at least today, potential applications are limited. Clearly these observations are not new, and have been exploited for a long time in lossy video and audio compression systems. Approximate Computing tries to extend these principles down to the hardware level. Also, the motivation is different. While these mechanisms have previously been applied in order to save memory or data-rate, Approximate Computing is using them to reduce the power consumption.

Noisy and redundant input data This category can be found especially in wireless communication systems. For instance, wireless radio receivers in this domain usually have to deal with noisy and imperfect input data due to the imperfect radio channel. The useful data has to be extracted from a noisy input stream. For the user however, it is not important whether the noise in the receiver chain is coming from the wireless radio channel or from noise within the circuit, like the FFT or the Viterbi decoder as depicted in Figure 2.13. Channel noise as well as circuit approximations can result in bit-flips, which is why circuit approximations in this example behave similar as channel noise. Important for the system and the end user is, that the resulting bit-error rate at the output of the receiver is staying below a tolera-

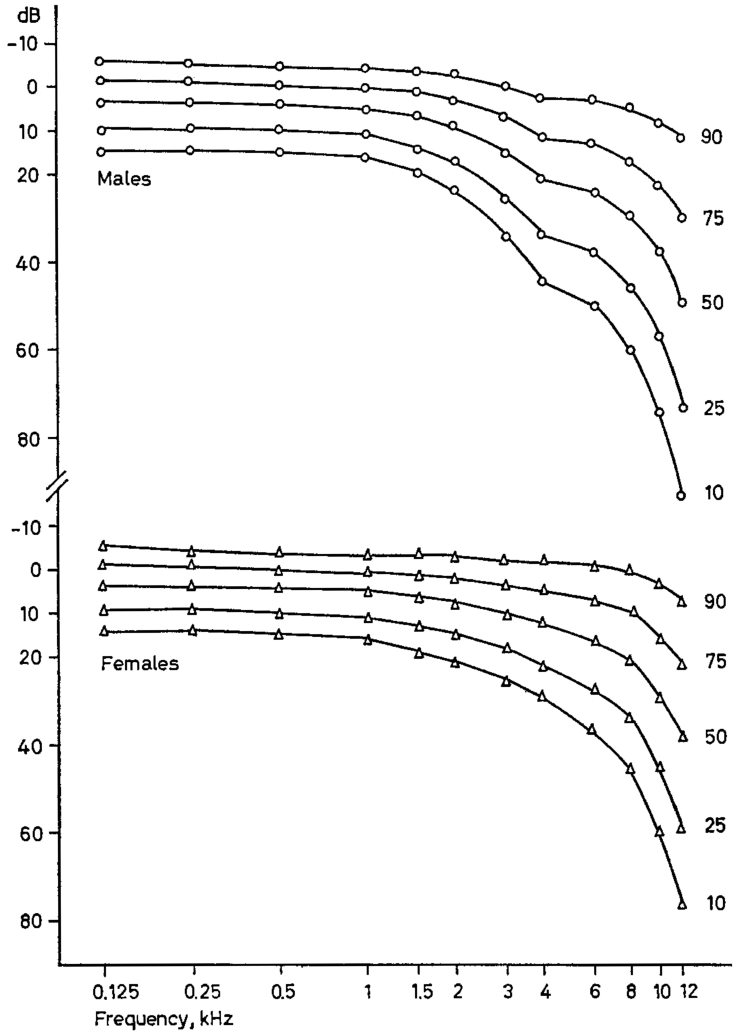


Figure 2.12: Change in hearing threshold level between age 18 and 55, for males and females, as a function of frequency, showing calculated medians, quartiles and deciles [33].

ble limit. If for instance the bit error-rate due to the channel noise decreases, the approximations within the circuit could be increased, maintaining a constant bit-error rate at the output of the receiver. Applications that have to deal with noisy input data usually have mechanisms built in to correct errors, like channel coding in wireless communication systems. Error protection always relies on redundancy. Hence, in systems that have to deal with noisy data, usually redundancy in the input data is included as it is the case in most wireless communication systems. One idea trying to exploit Approximate Computing in these applications is depicted in Figure 2.14. In this example the number of mantissa bits has been scaled in the decoder chain of a DAB receiver. If the receiver is close to the transmit antenna it

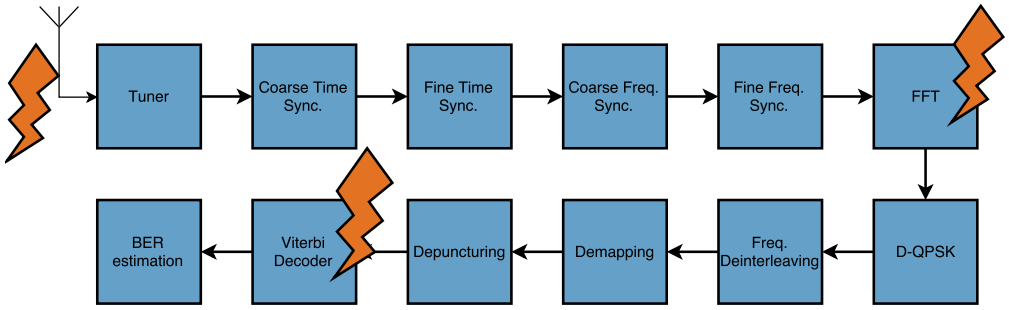


Figure 2.13: Decoder chain of a "Digital Audio Broadcast" ETSI EN 300 401 receiver

is assumed in this simplified example that the signal quality is good and hence the SNR is high. If the receiver is moving away from the transmitter, the free-space path loss is increasing, i.e. the signal power at the receiver is decreasing and the SNR as well. Such systems are designed to be able to operate in good as well as in bad radio channel situations. Error correction mechanisms are taking care of a correct data reception even in low quality channel environments. However, if the radio channel is good, e.g. if the receiver is close to the transmitter, the redundancy and the error correction at the receiver is actually not necessary at least not in the same form as when the channel would be very bad. The proposed

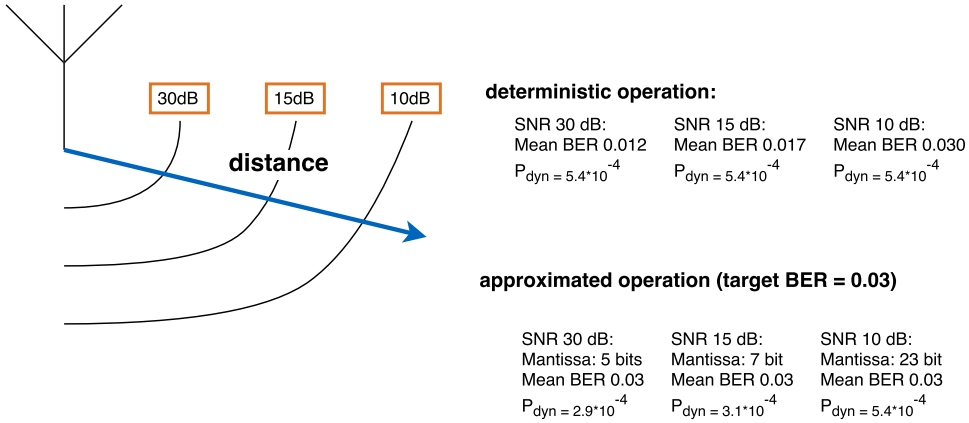


Figure 2.14: Adaptive approximation of a DAB receiver based on signal-to-noise ratio of the received signal

idea is to operate the receiver and its error correction blocks in a low power but less precise state, whenever the full precision is not needed as the signal quality is sufficient. If however the channel quality decreases and the full error correction capabilities of the receiver are needed, precision could be enabled in the receiver, the power consumption would increase, but the overall error rate at the output of the receiver remains constant. This way, such a system could be always operated in a sufficiently precise state, consuming not more energy than needed.

Self-healing and iterative algorithms (e.g. Turbo decoder) Another domain of applications suitable for Approximate Computing can be found as well in the area of wireless communication. Iterative algorithms like iterative decoder could be also potential candidates for approximations [35]. In this case precision could be traded off with iterations. In the scope of this thesis, these types of applications are not further considered.

Statistical & probabilistic computations (Neural networks) The last category of applications worth mentioning here, are applications that are itself based on stochastic processes. Applications like "Neural Networks" or "Genetic Algorithms" are largely based on stochastic processes. The randomness of circuits introduced by approximation could be exploited for these applications as proposed by Chakrapani et al. [36]. As it has been shown earlier in this work, probabilistic computing, a research field closely related to Approximate Computing is making use of this. For the scope of this thesis these types of applications are not further considered.

The nature of workloads has changed in the last years. While *Big Data* tasks like managing and searching data are computed in the cloud on computational powerful clusters, mobile devices have a different workload. Today, mobile devices are working on all types of media, not only playback and encoding, but also recognition and virtual reality. Natural interaction between the user and the environment has become a very important feature [8]. One can already see that most of these tasks do not rely on absolutely precise results. Also, in the world of the "Internet-of-things", applications come with an intrinsic error-resilience property as these devices are processing noisy, sometimes redundant input data from sensors that give inexact results [30]. Recent studies [37] have shown that an average of 83% of the time spent for exemplary calculations can tolerate imprecise results, as can be seen in Table 2.2.

2.2.3 Levels of Approximation

The idea of Approximate Computing is to be applied on all levels of abstraction. This would result in the largest savings while accepting a minimal loss of quality, as on one each level the optimal approximation could be applied. At some levels, especially at high abstraction levels, even today approximation is a well-established principle [8]. Algorithms that trade off the quality of a result with the complexity of calculating it, can be found in many applications [38, 39]. Especially lossy compression algorithms for audio and video applications exploit the limitation of the human perception in order to save data-rate since many years. Systems like the "MPEG 1 Layer 3" system [40] actually contributed a lot to the modern digital mobile world. But even network protocols, especially connection less ones like IP or UDP, utilize some kind of an approximation [41]. By accepting a potential packet loss, a reduced quality of service, routing schemes like "best-effort", have a reduced complexity on the application side but also on the network side. One can see that existing applications that exploit "approximations" usually try to decrease complexity, latency and data-rate. Reducing the power consumption is new goal that came into play with the increasing demand for low-power mobile applications. The

Table 2.2: Average time spent computing in resilient kernels for exemplary benchmark applications [37]

Application	Algorithm	% Runtime in resilient kernels	dominant kernel (Contribution to runtime)
Document Search	Semantic Search Index	90	Dot Product Computation (86)
Image Search	Feature Extraction	78	Dot Product Computation (71)
Hand Written Digit Classification	Support Vector Machines (SVM): Testing	94	Dot Product Computation (89)
Hand Written Digit Model Generation	Support Vector Machines (SVM): Training	97	Dot Product Computation (93)
Eye Detection	Generalized Learning Vector Quantization (GLVQ): Testing	89	Distance Computation (83)
Eye Model Generation	Generalized Learning Vector Quantization (GLVQ): Training	96	Distance Computation (92)
Image Segmentation	K-means Clustering	74	Distance Computation(66)
Census Data Modeling	Neural Networks: Multi Layer Back Propagation	62	Matrix Vector Multiplication (42)
Census Data Classification	Neural Networks: Forward Propagation	79	Matrix Vector Multiplication (64)
Nutrition and Health Information Analysis	Logistic Regression	65	Dot Product Computation (48)
Digit Recognition	K-Nearest Neighbors	96	Distance Computation (92)
Online Data Clustering	Stream Cluster	77	Distance Computation (68)

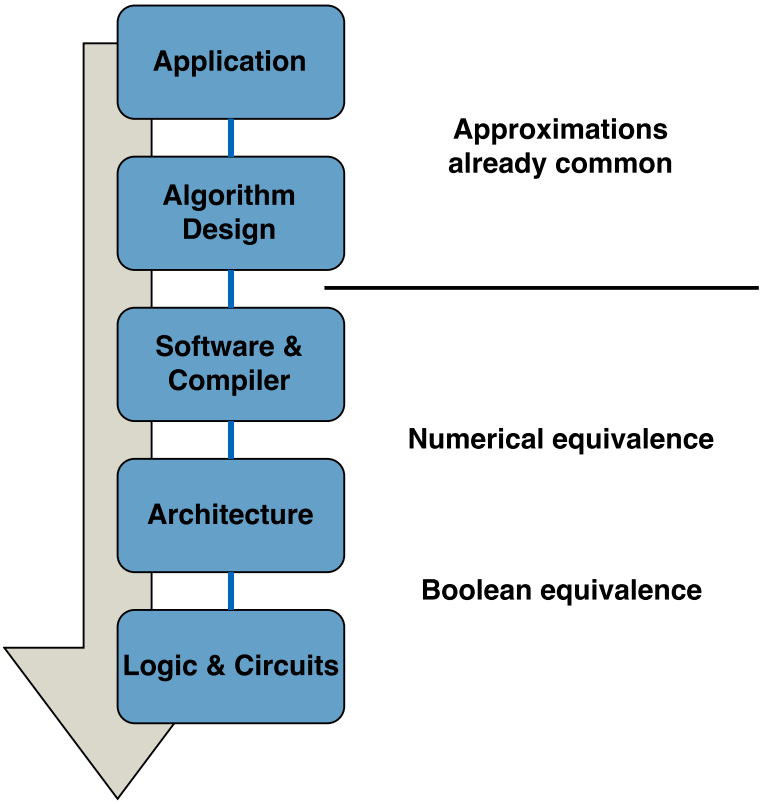


Figure 2.15: Abstraction levels already used for approximations in today's applications

above mentioned examples however are only applied at the higher levels of abstractions. Languages, compilers, operating systems, architectures and circuits are still designed to operate deterministic. Between software and architecture a strict numerical equivalence and between architecture and circuit a strict Boolean equivalence is expected, as shown in Figure 2.15. This is where Approximate Computing is focusing on in particular. Established techniques like bit width adaption and fixed point arithmetic are some of the few existing exceptions.

Algorithm/Software Level The highest abstraction level of Approximate Computing is the algorithm level. Approximations made at this level are no novelty and cannot be claimed by approximated computing alone. One technique worth mentioning is *incremental refinement*. Iterative algorithms can be approximated at this level. For instance iterative detection algorithms, like [42], are applying this technique. By terminating an iterative algorithm earlier than intended, one can save power while accepting a certain quality degradation. Iterative algorithms can be also found in today's mobile network systems, like LTE (Turbo decoding, LDPC decoding). Most likely in next generation technologies these algorithms will be still used. However, "approximate signal processing" like incremental refinement is not part of the standards, defining these systems. It

is up to the companies implementing this paradigm into their products and offer the end user an extra value. Using fixed-point arithmetic instead of floating point calculations for instance in signal processing applications [43] would be another technique to trade-off precision with power. By using the regular ALU instead of the complex floating point unit, the performance could be increased and power can be saved. Selectively skipping computations [44] whose influence on the overall result is less important could be also one approach to save power. One can see that already several ideas exist on the highest levels of abstraction. All these ideas however have to be manually implemented by the developer. The developer has to define which parts of an algorithm have to be approximated. There are already frameworks available that try to automatically identify blocks that can be approximated [37]. But the research in this field has just begun. All previously mentioned techniques assume that the underlying hardware is deterministic. Executing code on non-deterministic hardware can also be defined on a high level of abstraction. For instance, the definition how precise certain calculations have to be performed could be done already in the source code by the programmer who might have an insight on the effects of approximations at this point. Projects like the EnerJ framework [45] provide means to define these properties in the source code. Adaptivity, one key feature of Approximate Computing, is very important to consider at a high level of abstraction. Today's communication systems already use adaptivity for instance by changing the transmit power of the mobile device based on the channel quality. This is of course a very efficient way to save power, where approximations are not required at all. Another adaptive method already exists in digital video broadcast, although rarely used. The DVB standard for instance allows to simultaneously transmit a low quality copy of the original video signal. Due to the lower quality and hence the lower data rate, this stream can be transmitted using more robust, lower order modulation schemes. The receiver can decide based on the received signal quality which stream to decode in order to give the best end-user experience. However, these are just two of the very few examples that are used today. The majority of applications is designed not only to operate statically at one operating point, but usually also completely deterministic. Methods and tools are missing that simplify the adoption of Approximate Computing. This is not only true for the software level, but for all layers of abstraction.

Architecture Level Approximate Computing at architecture level tries to utilize approximate circuit elements in order to create an architecture suitable for Approximate Computing. The design space ranges from approximated hardware accelerators via approximate general purpose CPUs to memories. Hardware accelerators are the most commonly used approximated hardware blocks at architecture level. In this thesis the focus is put as well on approximating hardware accelerators. It is important to notice, when dealing with approximations at architecture level, that the accelerators provide means to adjust the degree of approximations. It has to be clear and easy for the architecture designer how to use the approximation features of a hardware accelerator. This also means that approximations made at the hardware level have to be flexible. Multiple operating points have to be defined and accessible. This is one of the main contributions of this

thesis. In Section 2.3 many examples will be shown for approximated hardware accelerators. Approximate general purpose CPUs are another way of providing approximation support at the architecture level. Usually the idea is to extend the existing ISA with approximate instructions like an approximate ALU [46]. It is again very important that these features are usable easily by the programmer. Programming language extensions like the EnerJ framework are therefore inevitable [45]. Depending on how excessively approximate instructions are used by an application, a decent amount of energy can be saved. The utilization of approximate memories is a third option to use at the architecture level [47, 48, 49]. Of course all techniques can be combined. The idea is usually to provide means to differentiate between important data and data that can be lost. Energy can be saved for instance by reducing the supply voltage or reducing refresh intervals in DRAM. However, approximate memories cannot be used without having a knowledge about the importance of data at software or compiler level. Hence approximate memories go hand in hand with programming language, compiler and micro-architectural support.

Hardware/Circuit Level Approximations at hardware level are the most common approximations. This category can be even further divided into building block level, register level, gate level and even transistor level. Most approximations however reported in the literature are done at register-transfer level and gate level. The approximations are usually done by functionally approximating the circuit. That means by removing parts of the circuit or replacing with lower precision copies. The difficulty is again to identify which parts of the circuit can be modified and to which degree, while maintaining a quality constraint. Many related works, as we will see later in Section 2.3, try to identify commonly used kernels in circuits, building blocks, like adders and other arithmetic blocks. For these kernels less precise variants have been developed (optimized for instance at gate level) in advance that can now replace the original kernel. Other works, including the presented one, try to approximate circuits at a finer granularity, at the register level. This gives a trade off during analysis between the level of detail, hence quality of approximation, and speed. Compared to the approximation at building block level, one does not have to rely on a previously defined set of commonly used building blocks, approximations can be made much more flexible. The circuit analysis however is much more complicated as it has to be done for the whole circuit at once. Approximations at gate level are also a common practice. The idea to modify the Boolean function so that less gates or gates that consume less power are used. An approximation at gate-level for the whole circuit at once is impractical due to the complexity of the analysis. This is why this technique is usually applied for sub-circuits like arithmetic building blocks. These building blocks can then be independently analyzed and optimized. Going one step down to the transistor level increases the complexity even more. Modifications at this abstraction level could be for instance done at the transistor dimension, mainly the channel width and the supply voltage. A modification of these parameters in order to consume less power results in both cases in a larger propagation delay, i.e. slower circuits. Over voltage-scaling is a widely used technique in Approximate Computing. The idea of over voltage-scaling or aggressive voltage-scaling is to reduce the supply voltage so that

the power consumption of the circuit significantly reduces. The resulting performance degradation and timing violations possibly resulting in bit errors are accepted. However, one has to keep in mind that the idea of Approximate Computing is still to minimize the resulting effects of the approximations. Hence, for voltage over-scaling it is inevitable to perform a detailed analysis of the resulting effect of the timing violations. This topic will be discussed in detail in Sections 2.3 and 5. A summary of the hierarchies of Approximate

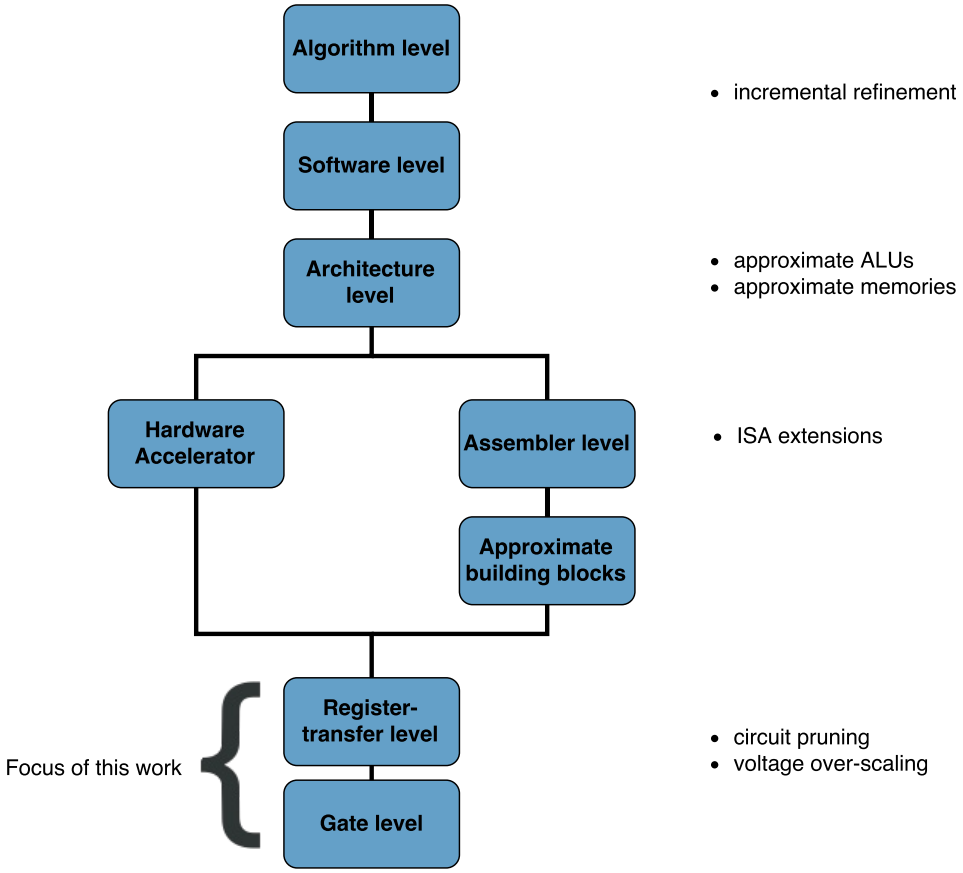


Figure 2.16: Approximation techniques at various abstraction levels

Computing can be seen again in Figure 2.16. In the figure one can see that the vertical path through the hierarchies is splitting up at a certain point. Depending on whether the approximations are done for a dedicated hardware accelerator or a CPU the path differs. In the latter case an additional Assembler/ISA level is introduced and hardware approximations can directly be made on a building block level, i.e. adders, FPUs etc. In the end, building blocks are just small hardware accelerators. The methods for analyzing and approximating a circuit therefore do not differ for a dedicated hardware accelerator or an approximated CPU.

2.2.4 Metrics of Approximation

There are various possibilities to quantify the effects of Approximate Computing. The most important metric is of course the power consumption, as this is the main goal Approximate Computing is trying to optimize. However, as we have already seen, it could be as well the chip area or the operating frequency of the circuit. These units are usually easy to measure and are well suited to quantify the quality of approximations. On the other side the situation is much more complicated. The degradation of the quality of the application is usually much more difficult to measure. Furthermore it is impossible to use the same metric for all applications. The quality of a search algorithms has to be quantified differently than that of an image processing algorithm. Application specific metrics are therefore inevitable. A variety of metrics are available and can be used to quantify the loss of quality of an application due to approximation. In image and video processing systems metrics like the peak signal-to-noise ratio (PSNR) and the structural similarity (SSIM) can be used. For signal processing algorithms the signal-to-noise ratio (SNR) can be a sufficient metric. For a channel decoder however, the bit error rate (BER) at its output could be a appropriate metric. Some works utilize error rate/frequency (ER) [50, 51], the ratio between correct input bits and faulty output bits, to quantify the quality of a circuit. For circuits that have multiple outputs that form a number, metrics like error significance (ES), error magnitude and hamming distance [52] are more suitable, especially if different bits have a different weight. Average error, relative error and error distribution are also sometimes used in order to quantify the quality of a circuit [53]. For approximate arithmetic blocks the (mean) error distance [53] is usually used. The product of power consumption and error distance can be used to quantify the trade-off of the approximation. One can see that a variety of metrics exist and most applications have the need for their own ones. Especially at the algorithm level it is impossible to define a metric that suits all applications. However, when going down in the hierarchy more general metrics can be applied. Actually one can see that at register-transfer level there actually exists one common metric, which is the bit-error rate, the error probability of a bit (see again Figure 2.5). That means, when approximating circuits at this level, usually the probability of a bit flip can be used to quantify the quality of the circuit. In order to approximate an application it is still inevitable to define the required quality constraints at the uppermost level, the level where the user is interacting. These quality constraints then have to be propagated down to the point where the designer wants to approximate the application. As already mentioned, the focus in this work is the approximation at register-transfer level. Hence, it is required to propagate the information about the required quality from the application level down to the register-transfer level in order to perform an adequate approximation that in the end meets the constraints defined at user level.

2.3 Prior Art

In the following related and prior art, that is directly related to the contribution of this work, will be presented.

2.3.1 Fault Analysis

Fault analysis is the first main contribution of this work. In order to make Approximate Computing usable and reliable, an accurate yet simple and fast analysis of the application is required. As this work is mainly focusing on the approximation of hardware circuits, methods to analyze digital circuits are presented in the following. Various possibilities in order to do so exist. The goal is always to determine how a circuit behaves, especially the outputs of a circuit, if the inputs or the elements of the circuit itself are unreliable. The metric in this case is usually error rate. The techniques that have been presented in earlier work can be grouped into analytic error rate prediction, software-based simulations and FPGA-based simulations [141].

Analytical Error Rate Prediction Analytical error rate prediction tries to calculate the error rate at the output of a circuit for a given set of gate-level error probabilities. Chakrapani et al. [54] have been defining a "Probabilistic Boolean Logic" in order to provide a modeling framework that allows in general to take the reliability of the single building blocks (gates) into account. Analytical error rate predictions are usually designed to operate at gate-level. However, for most approaches an adaption to register-transfer level would be easy. Often a matrix-based formalism, a set of linear equations is used in order to calculate the propagation of faults through the circuit [55, 56]. Nodes are either gates or registers. One limitation of this approach is that recursive feedback loops are reportedly difficult to model. Furthermore, the larger the circuit, the more complex it will be to solve the equations. Some works use probabilistic transfer matrices (PTM) to define the behavior of gates and calculate that of the overall circuit [57] or the reliability of individual signals [58]. A PTM is basically describing the probability of a gate that an erroneous input signal is visible at the output depending on the Boolean function of the gate. Based on the interconnection of gates, the individual PTMs are composed to a new PTM describing the whole circuit. For instance, a parallel composition of gates can be calculated with the tensor product of the gate PTMs, as shown in Figure 2.17. By defining the PTM of each gate, i.e. defining the relation between

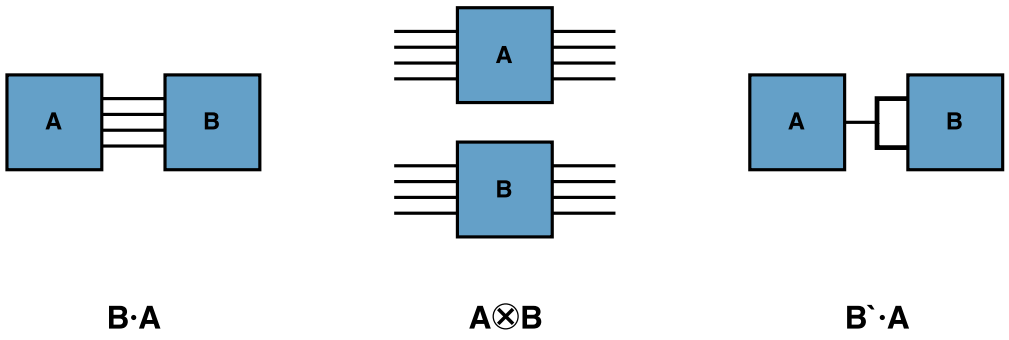


Figure 2.17: Matrix operations used to combine individual probabilistic transfer matrices (PTM) to describe whole combinational circuits [57]

input error probability and output error probability, successively the overall circuit PTM is calculated. With this circuit PTM, accurate information about the error rate can be gained. The main benefit of this method is clearly the exact calculation of probabilities and the perfect observability. However, as the memory required to process such circuit PTMs is very large, the method only scales well for smaller circuits or sub-blocks of large circuits. Furthermore, sequential circuits are hard to handle. In [59] an extension to this methodology is proposed. By compressing the PTMs with algebraic decision diagrams the required memory is reduced. The method is promising, but the compression factor is varying largely dependent on the circuit and the investigated circuit probability [141]. The idea of probabilistic transfer matrices can be even used to model multi-bit soft errors [60]. Similar to the PTM approach, Wang et al. [61] propose a probabilistic error masking approach. The prediction is performed by propagating erroneous tokens through abstract logic networks. Their approach is performed at a building block level for instance to predict the influence of errors in a CPU pipeline, but could likely be applied on gate-level as well. Error masking prediction is very important when performing voltage over-scaling, as we will see later. Rejimon et al. [62] propose a probabilistic error model based on Bayesian networks to estimate the expected output error probability, given dynamic error probabilities in each gate. This seems to be also a promising approach that, according to the authors, scales well with the circuit size. However, it is unclear how this approach can be applied at sequential register-transfer level, with recursive feedback loops. Another approach is presented by Mirkhani et al. in [63]. "RAVEN" is a statistical method to estimate the error rate of a circuit when injecting faults at register level. This method is performing a divide and conquer approach to tackle the complexity. By using fast local simulations for each error injection and calculating the probabilities for the whole circuit, RAVEN is a very interesting approach for vulnerability estimation. Finally another approach worth mentioning here is presented in [64]. "ADEPT" provides VHDL models as well as analytical descriptions for essential building blocks. These models can then be used, on a block level, to estimate the propagation of errors. Analytical error rate prediction captures some interesting approaches for the analysis of circuits. The limitations are clearly the complexity especially with respect to large and sequential circuits. However, provided that these problems are solved in the future, analytical error rate prediction is a technique that should be reconsidered when analyzing the behavior of integrated circuits with unreliable building blocks due to the aforementioned benefits.

Software-Based Simulation and Error Rate Prediction On the software side, related work can be divided into two main blocks, depending on which level of abstraction has been chosen [134]. One large block covers the development of accurate models that describe the behavior of essential building blocks when operated at low power. The non-determinism is originating in this case from noise based faults or timing violations. Hence this field is mainly about simulating the physical behavior of MOSFETs. Most common are SPICE simulations in order to determine the behavior of approximated building blocks. However, realistic noise models are required in order to generate useful results

[16]. George et al. [65] simulated probabilistic one-bit full adders and extrapolated the results to a multi-bit ripple carry adder. Later, it pointed out that simply extrapolating data from single elements overestimated the error of the overall circuit, as noise filtering effects were not taken into account. Several following works, like [66, 67, 68], faced this problem and developed more accurate extrapolation techniques to describe the probabilistic behavior of complex circuit structures by the composition of simple and repeating structures. Noise based faults and approximations are in particular important in the field of probabilistic computing. Analyzing the effects of timing violation due to voltage scaling is a different field. The related work of voltage over-scaling will be covered later in this section.

The other large area of software-based simulations is about modeling the approximations at gate or register-transfer level. This field is closely related to reliability estimation, e.g. when trying to efficiently harden circuits against soft-errors. The amount of related work is huge. An extensive overview can be found in [69, 70]. Basically two methods exist. Either the source HDL code is modified in order to allow fault injection or the simulation tool is used to force the injection of errors during simulation. Fault injection experiments in simulations came up in the early 90s and are since then an inherent part of reliability related research. A well-known work is the DEPEND system that allows modeling at functional level based on failure injection [71]. A system that is mostly referenced when talking about source code modification is the MEFISTO fault injection [72]. MEFISTO modifies VHDL descriptions to allow fault injections. The benefits of software-based simulations are the observability as well as the accuracy which depends on the abstraction level. The disadvantage of software-based fault injections is clearly the simulation time that grows exponential with the simulation granularity and the circuit size. Over the years the methods have been tried to be improved, mostly in terms of simulation speed.

Another branch of software-based error rate prediction is the sensitivity analysis. Sensitivity analysis tries to determine which parts of a circuit are sensitive at what point in time. The difference to fault injection experiments is that the information is not gained by Monte-Carlo experiments but with other methods. One method, among others, worth mentioning here is the backwards analysis [73]. By tracing the output of a circuit and a specific testbench backwards, from outputs to input, the backward analysis is able to perform a detailed sensitivity analysis at a reasonable simulation speed.

Even though there are many promising approaches, at different levels of abstraction, they all lack one property required for our approach. A fault at gate level, or a failure at block level does either have an influence on the behavior of the application or not. All the methods presented at software-level generate “binary” results. For Approximate Computing however it is more important to know the degree of the influence than the knowledge of an influence itself. This probability awareness, is not covered by any of the software-based fault injection or simulation methods.

FPGA-Based Emulation In order to circumvent long simulation times that are inevitable for software based simulations, hardware accelerated simulations can be per-

formed. One approach that is often used is to emulate the circuit-under-test on an FPGA and inject faults into the emulated circuit. As the hardware in this case is emulated and not simulated, the execution time is much faster. Again, many different variants of FPGA-based fault emulation exist. Mainly two techniques for injecting errors into the circuit-under-test exist. One approach is to reconfigure the FPGA (not dynamically) with modified bitstreams that contain bit-flips, e.g. as described by Cheng et al. [74] or Perreira et al.[75]. Partial reconfiguration could be used to increase the reconfiguration delay but generating the huge number of different bitstreams in advance is still a bottleneck in this approach. Furthermore, only permanent faults, i.e. "stuck-at-faults", can be emulated. Kafka et al. [76] use a dedicated CPU core to speed up the download of new modified bitstreams onto the FPGA. Antoni et al. [77] use a special feature of Xilinx Virtex FPGAs (JBits) which allows for a dynamic manipulation of bitstreams at run-time and therefore the modeling of transient faults. One of the main benefits of this approach is the low hardware overhead caused by the fault injection. As no additional logic is required, the area requirement is about the same as for the fault free variant. Mostly due to the increasing size of today's FPGAs, this static fault injection approach lost its practical relevance as the area benefit relativized. Another approach for FPGA-based fault injection is to inject faults dynamically into the circuit based on an injection plan. This requires some kind of circuit modification that allows for dynamically flipping bits. This method is called "circuit instrumentation". It describes the process of inserting instruments into the circuit that allow for injection or observation of faults, as depicted in Figure 2.18. The additional input, denoted as "E", allows to flip the value stored in the flip-flop.

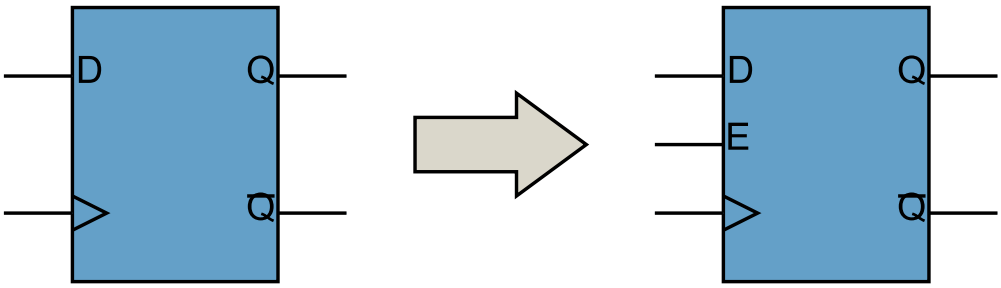


Figure 2.18: Circuit Instrumentation

The circuit modification is an automated process in most of the related work. Circuit instrumentation in general consists of two parts. First, it requires to replace primitives like flip-flops or logic gates with functional equivalents that additionally allow for a fault injection. And second, some kind of control infrastructure is required that triggers fault injections based on a plan defined in advance. Related work differs on the instruments and the control infrastructure. Hwang et al. [78] inject errors into the combinational part of a circuit with a scan-chain. Their instrumentation is limited to permanent faults. Most approaches however, like [79, 80, 81, 82], change the content of flip-flops, by replacing them with instrumented models. The difference in the approaches is usually how the fault injection plan is transferred onto and stored on the FPGA. Usually, a list is generated in

advance containing where and when a fault is going to be injected. Depending on whether this list is transferred on-the-fly, during run-time, to the FPGA or in advance, the speed of the emulation as well as the memory requirements differ a lot. Nevertheless, by accepting a certain logic-overhead for the injection chain and simulation control, circuit instrumentation is the fastest way of injecting and analyzing the behavior of errors in a circuit, as no reconfiguration is needed. The hardware overhead however is not negligible especially the routing overhead which results in a very congested design as we will see later.

One important consideration has to be made for FPGA-based fault injection when analyzing ASIC designs. Emulating ASIC designs on an FPGA requires an adequate translation of design primitives. This step is usually not mentioned in related work but is it is crucial in order to generate trustworthy results. The approach proposed in this work will be presented in Section 3.

Similar as for software-based fault injection, the probability-awareness which is inevitable for Approximate Computing is not considered in any of the related works. In general the FPGA-based emulator presented in this work is similar to the ones presented here. However the error model is different. Previously presented emulators try to observe if an error that has been injected into a circuit element is visible at the output of the circuit for a specific circuit state. Hence, whether it is masked or not. For Approximate Computing however, one is interested in the error rate at the outputs of the circuit for certain error rates at the circuit elements, without further specifying the circuit state. In order to gain this information it is required to inject faults into the circuit based on a defined error probability. In this work the circuit instrumentation approach is extended to support this mechanism.

2.3.2 Confidence Intervals for Probabilistic Experiments

The probability awareness proposed in this work does come with some obstacles that have to be cleared. The injection of faults based on probabilities results in the fact that the measured error probabilities at the outputs of the circuit are as well underlying statistical properties. The measured results vary depending on how long the circuit has been emulated, the set of input test pattern, the error probabilities and the state of the circuit [143]. Hence the measured error rates differ even for the same set of applied error probabilities to the flip-flops. One important obstacle to clear is to make the emulations reproducible and trustworthy. Hence, the variance of the fault injection experiment has to be small. In order to reduce the variance, hence to reduce the limits of the confidence interval, the simulation time, i.e. the number of emulated clock cycles, has to be increased. However, simulation time, even for FPAG-based emulation, is a critical factor. It is therefore necessary to keep the number of emulated clock cycles large enough so that the confidence interval satisfies the users' needs and small enough so that the simulation time remains short. This problem seems to find a surprisingly small interest in the research community. For "deterministic" fault injection experiments, hence when injecting faults at specific circuit states based on previously defined injection plans, approaches have been presented to quantify both the error on the results and the

confidence on the interval [83]. This is usually referred to as "statistical fault injection". In statistical fault injections, only a subset of the possible errors is injected. This subset is selected randomly with respect to the injection target and with respect to the injection cycle. Leveugle et al. presented a simple framework to estimate the number of faults to inject for a desired error margin based on statistical sampling theory [83]. Emulators exist that make use of these methods [84]. Statistical fault injection is a very interesting and useful approach. Reducing the search space is inevitable for probability-aware fault injection, as presented in this work, as well. However, the simple sampling theory cannot be applied here as each injected fault is a random process on its own, and all are active throughout the whole circuit at once. Additionally, the information gained in related work is, whether an error is visible at the output or not, no nuances in between. The emulation is usually stopped once an error has been detected at the output of the circuit. For instance, Lopez et al. [80] emulate the circuit until the fault is classified or until the testbench ends. Clearly, the problem in this work is more complicated as we are trying to tolerate faults and the resulting errors within the circuit. For probability-aware fault injection experiments new methods had to be developed, which will be presented in Section 4.

2.3.3 Circuit Approximation

In this section an overview about the various available techniques to practically approximate integrated digital circuits is given. The overview is by no means complete, but contains the related work which is related most to the approaches presented in this work.

Approximate Building Blocks The approximation of essential building blocks was the first type of Approximate Computing that could be found in the literature. As we have already seen before, this type of Approximate Computing tries to identify essential building blocks in existing implementations that can be replaced by approximated variants. Essential building blocks are usually arithmetic blocks. Optimizing arithmetic building blocks for DSP applications is also a large area of study outside of Approximate Computing in deterministic applications. Energy efficiency in these cases can be achieved for instance by applying a dynamic voltage-frequency scaling scheme to the operations. Hence, when the workload is high, the circuits are operated at higher supply voltage, allowing to increase the operating frequency. Instead, when the workload is low, the operating frequency can be reduced and so can be the supply voltage [85, 86]. Other previous works do not propose adaptive approaches but instead provide the circuit with different supply voltages at the same time. Critical paths are operated at a higher supply voltage than non-critical ones [87, 88]. Note that all of these approaches still guarantee a completely precise, i.e. fault free result. Techniques exist that tolerate timing violations in order to increase the power efficiency. For instance, the "Razor" approach [89] proposes to tune the operating voltage so that timing violations are unlikely but can happen. Razor (shadow) flip-flops are introduced in order to detect timing violations helping for instance to restart the pipeline. Other techniques propose not to correct the timing errors on the circuit level but instead on a higher level. For instance Hegde et

al. propose an approach where the degradation in performance of the DSP algorithms is restored via algorithmic noise tolerance, where signal statistics are exploited to develop low complexity error-control schemes [90]. Shim et al. [91] propose to add "reduced-precision replicas" into DSP applications whose less precise, but in general correct, result can be utilized in case the main computational block suffers from timing violations. This approach is already closely related to the idea of Approximate Computing, as it utilizes a less precise result for subsequent computations. The normal case however is that errors are detected and corrected, before continuing the operation. This area of research is sometimes referred to as "algorithmic noise-tolerance" [92].

In Approximate Computing building blocks are approximated as well by functional approximation or voltage over-scaling. Compared to algorithmic noise tolerance, errors are neither detected nor corrected. Adaptivity is introduced, hence no reduced-precision replica is needed. The goal is therefore to build functional approximate replacements that operate as precise as possible (or as needed) while consuming minimal power.

In 2006, George et al. [65] demonstrated how imprecise arithmetic blocks can be used to produce "good enough" results while saving a large amount of energy. Since then much research has been done on building optimal approximated building blocks. Chakrapani et al. [93] provide a mathematical model to describe the behavior of arithmetic blocks when applying aggressive voltage-scaling. Applied to a Ripple Carry Adder, the authors were able to approximate a DFT and quantify the resulting error and power savings. Huang et al. [94] classified and compared different imprecise adder designs in terms of their error characteristics and power-delay efficiency. A case study revealed that sometimes simple precision pruning, i.e. bit-width adaption, can lead to better results than approximated adders due to voltage-scaling. Gupta et al. [29] propose a set of functionally approximated adders, approximated by modifying their schematic, so that they requires less transistors. Applied to video and image compression algorithms, the authors were able to show a power saving of 60% and an area saving of up to 37%. Figure 2.19 is showing the difference in the image quality when applying their adders to a DCT-IDCT transformation. One can see that the quality of the approximated version is nearly as good as the fault-free variant. Bit-with truncation however in this case results in a noticeable quality degradation.

In [52], Kahng et al. propose an approximate adder whose degree of approximation can be adapted at run-time. This freedom however results in a certain area overhead. In [95] Liu et al. explicitly focus on the approximation of multipliers by using newly designed approximate adders. The proposed multiplier has a very high accuracy and a low power consumption compared to a Wallace multiplier.

One can see the list of related work in the domain of low-power arithmetic building blocks is huge. Most of them promise high energy savings. However, none of them presents an approach where the approximation is performed automatically. A manual design process has always been required for creating the approximated variants. Furthermore, a lot of potential power savings are lost by concentrating solely on the approximation of arithmetic blocks. Hence, in order to optimize the benefits of Approximate Computing methods are required that automatically approximate digital circuits.

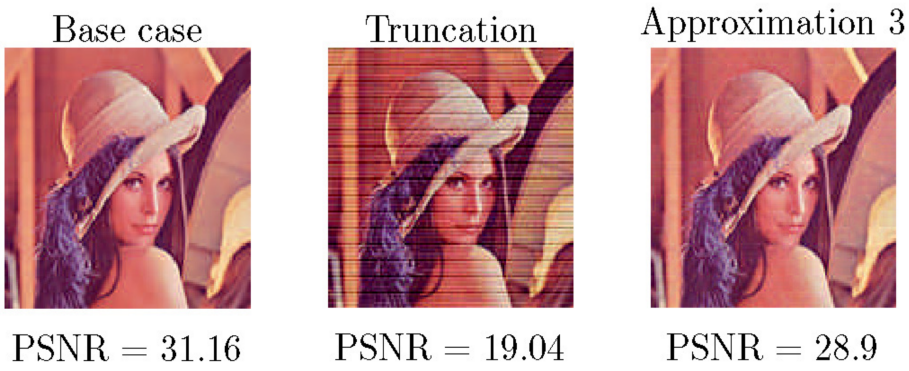


Figure 2.19: Quality degradation of test image due to an approximated DCT-IDCT transformation [29]

Automated Approximation Automated approximation of integrated circuits is an aspect of Approximate Computing where until now not much research has been reported. This is surprising as it seems to be very important for the acceptance of a novel paradigm to be easy to use for the average user. As it can be seen however, most research has been focused on the manual approximation of arithmetic blocks, namely adders. Different levels of abstraction can be covered of automated approximations. For instance, a circuit designer could define the maximum error rate for the outputs of an adder and an automated tool is analyzing which parts of the adder netlist can be approximated. Or, in case of voltage over-scaling, it could mean that a tool tries to identify the smallest value of the supply voltage possible so that the demands on the output error rate are still met. The first work in this direction is published by Lingamneni et al. [19]. The authors introduce "probabilistic pruning" of arithmetic units. Its approach focuses on architectural modifications to circuits based on pruning of the circuit elements. The idea is straight-forward. Successively the circuit elements get pruned that have a lower probability of being active during operation. Afterwards the circuit get simulated and the resulting error rate is compared with the constraint. If the error rate is smaller than the constraint the next element is pruned and otherwise, if it is too high, the last pruning operation is undone. Although being a very simple approach, it is yet very effective. One drawback is that even though an element or wire has a low activity, it does not mean that its significance on the output is small. However, the methodology presented in this work is in general similar, but the lengthy simulations can be avoided. Nepal et al.[96] proposed another approach that starts the automated approximation already at the behavioral RTL description. Based on the source files their approach builds an abstract syntax tree (AST). This AST is then modified, so that a syntactically correct but approximated circuit is resulting. The authors propose a set of transformations. The most important ones are "Data Type Simplifications", e.g. bit-width reduction and fixed point arithmetic, and "Operation Transformations", e.g. replacing an addition with a bit-wise OR. The problem with this approach is clearly the huge design space. For each

modification, the circuit has to be simulated and synthesized in order to check if the constraints are met. This can result in a very long design time. Nevertheless, this is a very interesting approach as it can combine several approximation techniques in one framework. Ranjan et al. [97] use formal verification techniques to identify the impact of approximated parts of a sequential circuit on the global output. This is the only prior work known that explicitly mentions the approximation of sequential circuits. Their approach promises to automatically build an approximated version of the circuit that consumes lower energy while meeting the specified quality bound. By constructing a "Sequential Quality Constraint Circuit" (SQCC) composed of the original and approximated versions of the sequential circuit, along with a "Quality Evaluation Circuit" (QEC) that codifies the user specified constraints on global output quality, the quality of an approximated circuit is evaluated [97]. The principle is shown in Figure 2.20. The original circuit (top) together with the approximated circuit (bottom) and the "Quality Evaluation Circuit" (right) form the "Sequential Quality Constraint Circuit". The valid output V is indicating that the approximate circuit output is ready to be evaluated. The output Q indicates if the quality constraints are met. By formulating the quality constraints as a sequential model checking problem, and applying formal verification techniques it is ensured that the quality measurements are correct. Approximations itself are again

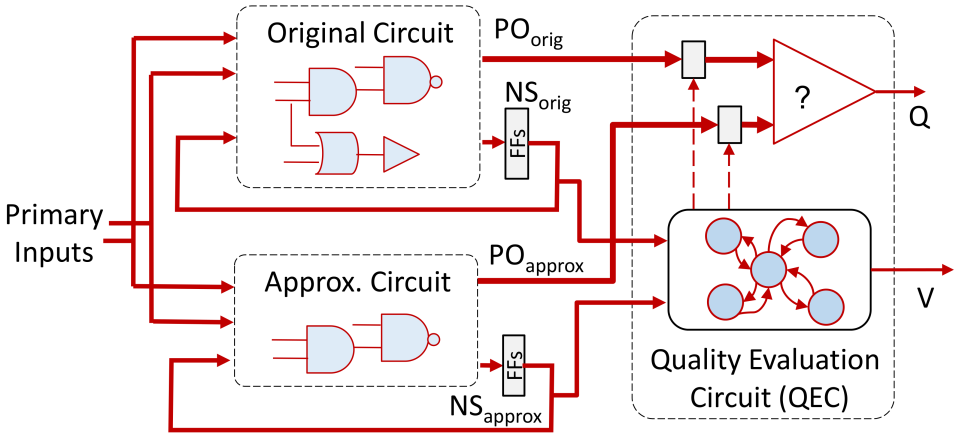


Figure 2.20: Sequential quality constraint circuit [97]

made by identifying essential building blocks. The tool identifies combinational blocks in the sequential circuit that are amenable to approximation, generates local quality-energy trade-off curves for them and uses a gradient-descent approach to iteratively approximate the entire sequential circuit [97]. The approach is again limited by focusing only on arithmetic units, as the complexity of a fine-grained approximation of large circuits would be impossible to handle.

Approximation at Synthesis Time Until now, most presented approximation techniques have been focused on modifying existing circuit implementations, usually netlists. A few approaches exist that propose an automated generation of approximated circuits at synthesis time. Choudhury et al. [98] propose a scalable algorithm for the synthesis of approximate circuits. An approximate logic circuit is constructed by selectively converting minterms from the off-set or on-set of its Boolean function into "don't cares". In order to select, which minterm is approximated an efficient algorithm that manipulates a multi-level network to synthesize approximate logic circuits is described. Hence, this approach can be seen as an automated *functional* approximation. The limitation however is that no automated estimations on the error rate can be made. In the work of Shin et al. [99], similarly, minterm complements are identified that produce an approximate circuit version that has the smallest number of literals for a given error rate threshold. The authors are proposing two methods for the identification. The first one is a simple search for the optimal one, similar to the approach presented in [98]. The second is a heuristic approach where the authors claim that it is practically usable even for large circuits and at the same time is near-optimal. An automated method for voltage over-scaling is presented for instance by Miao et al. [100]. The authors present an algorithm that synthesizes adder implementations for different application requirements and target technologies. For that, a formal model has been introduced to quantify the quality of approximated adders due to voltage over-scaling. One can see that already interesting work has been done in the field of approximated circuit synthesis. The future will show whether these techniques can be embedded into a complete approximation flow starting from the user level. If possible and if the performance, approximation-wise as well as regarding the synthesis, is comparable, approximate logic synthesis seems to be very promising approach.

Approximation of Sequential Circuits The approximation of sequential circuits is essential if the designer wants to approximate a circuit as a whole and not only essential building blocks. The related work that explicitly mentions the approximation of sequential circuits is rare. Analytical approaches are very difficult to realize due to feedback loops. The work presented by Nepal et al. [97] that has been presented earlier is the only work known explicitly mentioning sequential circuits. However, the work by Nepal et al. [96], also presented earlier, is likely to be adoptable to sequential circuits. Their proposed AST modification is applicable to combinational as well as sequential circuits. Nevertheless, the approximation of sequential circuits is a new field as most work has been focused on approximate arithmetic blocks. But for more complex circuits, like a floating point unit, that is usually implemented pipelined, analytical models cannot be applied anymore, and exhaustive software-based simulations become too slow. The presented works offer very promising approaches to verify the behavior of approximated circuits. They lack however again the probability-awareness, hence determining the error probability at output pins.

Approximation of Generic Boolean Circuits We have seen that many approaches to approximate arithmetic building blocks have been presented in the literature. The amount of works focusing on the approximation of generic circuits is limited. Especially when choosing voltage over-scaling as the primary approximation technique. One interesting work in this field has been presented by Venkatesan et al. in [32]. The authors propose a systematic methodology that allows to analyze how approximations effect the functionality of a circuit by calculating metrics like worst-case error and error probability. According to the authors, the methodology can be used for timing-induced approximations such as voltage over-scaling or over-clocking, as well as for functional approximations. By constructing an "equivalent untimed circuit" the approach converts timing-induced approximations into the functional domain. This is a very interesting idea. The methodology uses library information as well as desired frequency and supply voltage into account and generates a functional description of the circuit, describing how it would behave when applying the parameters. Figures 2.21 and 2.22 are showing how timing information is annotated to a combinational circuits and finally an "equivalent untimed circuit" is generated taking the timing violations into account. Using the equivalent untimed circuit

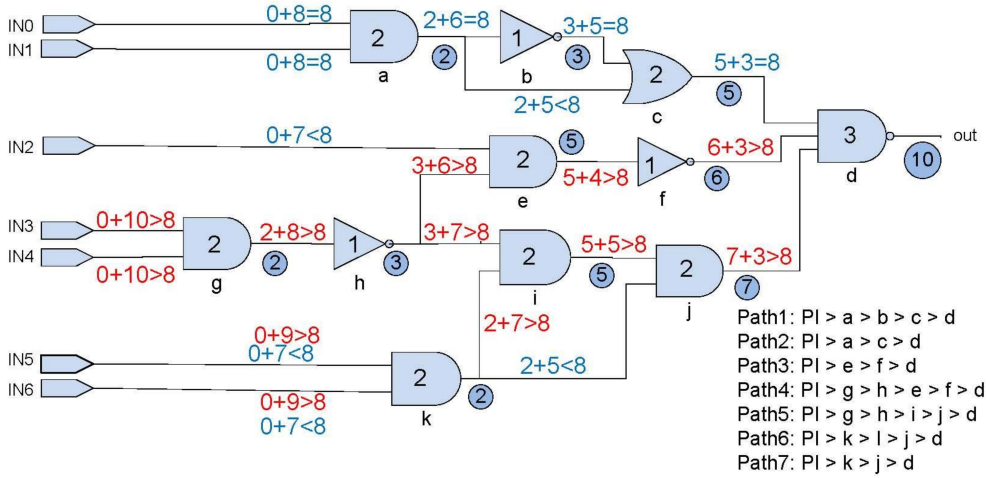


Figure 2.21: Annotated approximated circuit with timing information due to voltage over-scaling as proposed in [32]

and the golden circuit a "virtual error circuit" is generated comparing the outputs of the two circuit variants. The methodology then utilizes conventional verification tools like Boolean satisfiability (SAT) solvers, Binary Decision Diagrams (BDDs) as well as Monte-Carlo simulations to perform an error analysis. Especially the generation of an equivalent untimed circuit is very interesting and unique. Clearly, the approach is limited to combinational circuits and the error analysis is time consuming. Nevertheless, this is a very promising approach, especially for circuit approximation due to voltage over-scaling.

Voltage Over-scaling The last block of related work, presented in this section, is about voltage over-scaling of integrated circuits. As we have already seen before, voltage over-

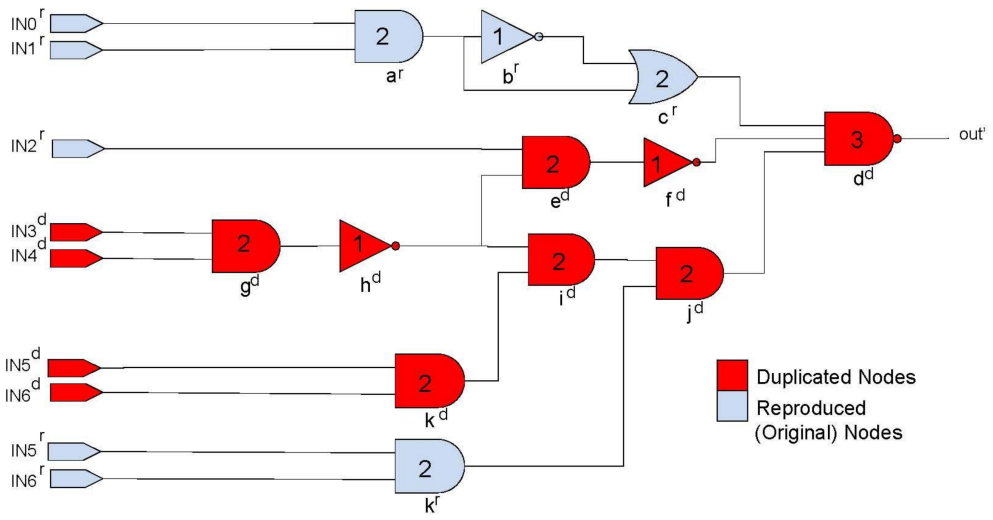


Figure 2.22: Equivalent untimed circuit as proposed in [32]

scaling is the process of operating a circuit at lower than nominal voltage, so that potential timing violations occur, due to the increasing propagation delay of the MOSFETs. Apart from the increasing propagation delay, the circuit does also become more susceptible to radiation induced soft-errors. Chandra et al. [101] show in their work that by scaling the voltage, independent of timing violations, the soft-error susceptibility is increasing dramatically as a much lower voltage peak is sufficient to flip a voltage level. In general, the analysis of Single Event Transient faults, hence radiation induced voltage peaks, have similar properties as errors due to timing violations, as their propagation is also depending on logical masking and the faults are not permanent. The various factors of an accurate SET analysis have been studied in [102]. Frequency over-scaling, a technique closely related to voltage over-scaling has been investigated for instance in [103]. In this work a predictive instruction-based dynamic clock adjustment technique is presented. I.e. based on the executed instruction, the clock is scaled in order to trim dynamic timing margins in pipelined microprocessors. Voltage over-scaling is a widely proposed technique in Approximate Computing, some of them already being presented earlier in this work. Hedge et al. [90] scaled the supply voltage of a digital filter in order to save power. However, instead of accepting the resulting errors, they introduced error correction mechanisms in order to avoid them. The error correction block is operated at nominal voltage. Shim et al. [91] focus their work also on scaling the voltage of digital filters. Instead of applying error correction schemes the authors introduce a low precision redundant filter with less critical timing constraints. This replica serves as less precise, but reliable result in low voltage conditions. This approach is clearly related to the idea of Approximate Computing. It requires however additional hardware to guarantee a less precise but functionally correct operation. In [104] Mohapatra et al. also focused on essential building blocks. The authors propose to identify computational kernels, that are common in applications with inherent resilience. These kernels have then been opti-

mized by applying "Dynamic-segmentation" and "Delay Budgeting", in order to increase the energy vs. quality tradeoff. Dynamic segmentation is a technique to create efficient approximated adders. Delay budgeting is a technique to chain, usually pipelined, arithmetic units within one clock cycle, as it has been shown that the delay is often much lower than the sum of the delays of the individual units [104]. In [105], He et al. also focus on scaling the supply voltage of arithmetic units. In order to guarantee that the error stays within quality bounds, the error is controlled by exploiting the knowledge of the operand statistics and by dynamic reordering of accumulations. These methods modify the behavior of the early and worst timing error offenders to allow for larger reduction of the supply voltage. Emre et al.[106] scaled the voltage of a JPEG codec. In order to minimize the errors induced by voltage scaling, a compensation technique has been used that exploits specific DCT characteristics. Again, this is a manual modification that ensures that less critical paths for the visual impression are failing first.

One can see that all of the presented approaches rely on manual interaction and a detailed knowledge of the circuit implementation. Furthermore, some modifications are very specific to the application. Real generic optimization techniques are rare. Kahng et al. [107] propose in their work "Slack redistribution". The authors propose to swap cells that are part of the critical paths with those having a larger drive strength. It is therefore possible to modify the slack distribution of the circuit to a more equally distributed one, as shown in Figure 2.23. Additionally in this work, a simple yet efficient analytic

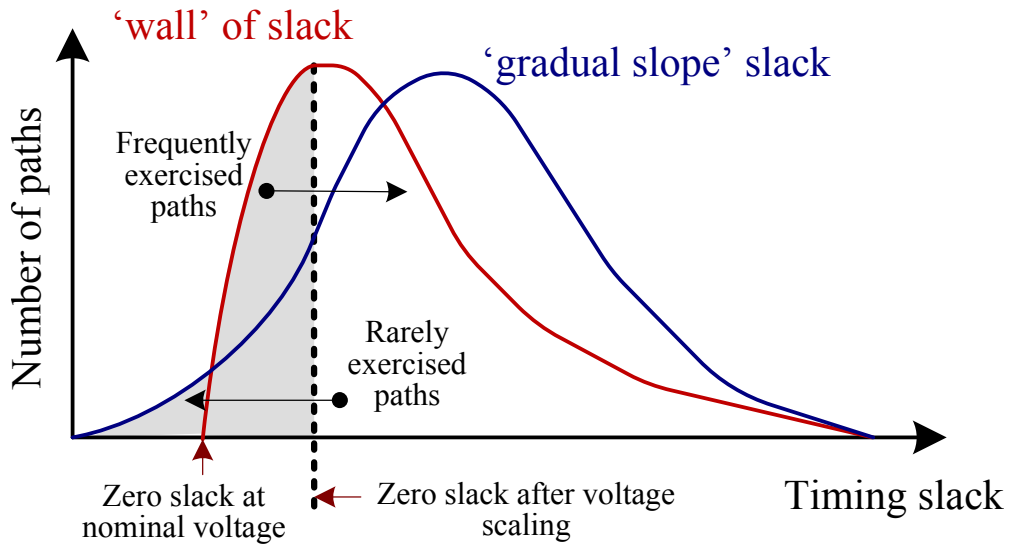


Figure 2.23: The slack redistribution technique in order to move the slack distribution from a critical "wall" into one with a more gradual failure characteristic [107]

estimation is presented allowing to estimate the error rate depending on the supply voltage. Unfortunately,the approach is reportedly sometimes inaccurate. However, in most cases, e.g. like [108], Monte-Carlo simulations are used to determine the average bit-error rate for over-scaled operating points which is much less efficient. Liu et al. [109]

presented in their work another analytic approach. In this work an algorithmic framework to estimate error magnitude of algorithmic functions is presented. Unfortunately, this approach is not applicable to generic circuits, only to adders. The methodology presented in Chapter 5 tries to offer a fast, more accurate estimation, to estimate the error rate of combinational circuits when applying voltage over-scaling.

The limitations of the methodologies presented in prior works are clear. Regarding the error analysis of circuits, the presented works lack the "probability-awareness". The definition of approximations and their resulting impacts on the circuit outputs based on error probabilities allows to model a variety of approximation techniques in a common way. The probability-awareness however requires sufficiently long simulations in order to generate confident results. Analysis speed is therefore of highest importance.

Almost all works proposing approximated circuits focus on the manual approximation of essential building blocks, mostly adders. Automated approaches and approaches that focus on generic circuits are very rare. The best approximation results can however be only achieved when approximating a circuit at once without any restriction on specific building blocks. This clearly requires automated approaches, that are furthermore able to approximate in a reasonable amount of time due to the complexity of generic circuits. In the following these problems will be tried to be solved.

3 Probabilistic Fault Emulation

In this section the proposed fault emulation system is presented. In Chapter 2 we have seen that many ways exist to analyze the behavior of circuits in case of faults or when approximations are tolerated. A reliable and accurate analysis of a circuit is inevitable, before it can be approximated. In order to approximate an integrated circuit it is very important to know exactly the effects of the approximation on the result of the application. As we have already seen, one key principle of Approximate Computing is that only the quality is affected by the modifications and not the functionality of the circuit. And furthermore, the degree of the quality reduction has to be defined at design time and must not be exceeded at run-time. Hence, the in-depth analysis pursues two goals. First, it is necessary to identify which parts of the circuit can be approximated at all. We will see later in this work that a general rule is that only the data-path of a circuit can be approximated. Approximations in the control-path would usually lead to unpredictable behavior which is not tolerable in Approximate Computing. And second, the analysis is necessary to identify the degree of approximation that is possible for these identified circuit parts so that the constraints on the resulting quality are still met. The degree of approximation can be a binary decision in case of circuit pruning (removing the element or not) or it could be for instance a supply voltage in case of voltage over-scaling as the chosen approximation technique.

One can imagine that with increasing circuit size this is becoming a complex problem. This work focuses on the approximation of sequential circuits mainly at register-transfer level. The decision to analyze at register-transfer level instead of gate-level decreases the complexity of the analysis a lot. At the same time it gives a decent level of detail, at least for an initial analysis. As we have seen already, analyzing the impact of errors at the register level allows to model a variety of low-level faults [23]. For functional approximations at register-transfer level, as it will be presented in Chapter 4, the level of detail is sufficient. For voltage over-scaling, as it will be presented in Chapter 5, additional analysis is required as we will see later. However, the initial fault emulation at register-transfer level already reduces the complexity dramatically, which enables us to perform a more detailed analysis at gate-level in a reasonable amount of time. But even at register-transfer level, the complexity is enormous. A sequential circuit with N registers has 2^N states. A full fault coverage would require that faults have to be injected into N different registers in 2^N states. In case of multi-bit errors, the search space would increase even more. Hence, when performing a simulative analysis it is essential in order to analyze realistic and large circuits that the simulation is fast. As we have already seen in Chapter 2, analytical estimations of the error rate are very difficult to realize for large circuits due to the memory constraints, and even more complex to realize for sequential circuits. This limitation and the slow simulation speed of software-based simulations,

lead to the decision to utilize hardware-accelerated FPGA-based fault emulations.

The idea of FPGA-based fault emulation is to run a circuit, even if it is intended to be fabricated as an ASIC, on a FPGA. Before emulating, the circuit is modified by some means to inject faults. In our case faults are injected into registers, i.e. flip-flops. By actually running the circuit on real hardware, the fault injection experiment can be performed much faster, up to in real-time.

A typical FPGA-based fault emulator consists of two main blocks, as shown in Figure 3.1. The first one being the control and evaluation software running on a host computer

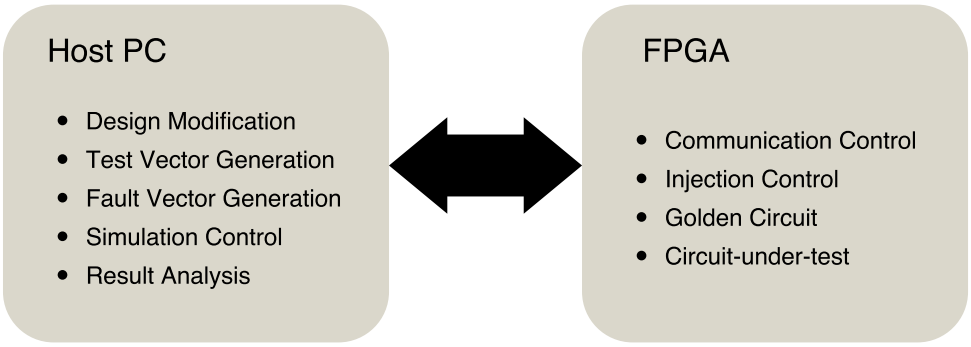


Figure 3.1: The two main blocks of FPGA-based fault emulation

and the second block being the hardware modules running on the FPGA. The host-based software itself has several functions. One important step prior the actual fault emulation is the instrumentation of the hardware description, i.e. the functionality to inject faults has to be added. This process is described in detail in Section 3.2.3. Once instrumented, the modified as well as the original, golden circuit can be synthesized for the FPGA. The golden circuit is necessary to compare the output of the faulty one for errors. Hence, both variants have to run simultaneously and synchronously, fed by the same test pattern on the FPGA. Running as well on the host PC is the control software, which is responsible for controlling the emulation. Depending on the actual implementation of the emulation system several tasks are performed. First of all, start and end time of the emulation are controlled. Hence, for instance the number of simulated clock cycles or the duration in seconds has to be controlled. Test vectors have to be generated or loaded and transferred to the FPGA. The generation is a complex task, as we will see later. Not only because a lot of actual data has to be generated but also because of the decision where to inject faults at which time. In order to control the system, an interface between host PC and FPGA is required. The details of the implementation presented in this work will be presented in 3.2. Once the simulation has finished the results have to be loaded and evaluated. On the hardware side, the main blocks are clearly the golden circuit and the instrumented circuit, the circuit-under-test. Based on the type of instrumentation, the instrumented variant can differ significantly in size, as we will see later. Apart from that, some control infrastructure is required that enables the communication with the host PC. Depending on the implementation, sometimes large memories are required to store the test vectors and injection vectors. Although the implementation presented in this

work is in general similar to related works, it differs significantly in the way how faults are generated. In related works, faults, i.e. the location and the time instant of a bit flip, are generated on the host PC based on some strategy. This can be done either prior the emulation or just in time. Benefits and drawbacks will be discussed later in Section 3.2.5. The implementation presented in this work generates the faults at run-time on the FPGA, independently for each register, based on previously assigned error probabilities. This actually makes it possible to use fault emulations for the analysis of circuits regarding their applicability for Approximate Computing, as it implements the required probability-awareness, that is necessary to model the effects of the approximations.

3.1 Probability-awareness

Circuit approximations at hardware-level result in wrong bit values at the output of the circuit. Even for the same approximations, e.g. when removing one gate at one defined position, the output pattern differs over the time, due to the changing input pattern. Hence, the impact of an approximation can be described as the probability of a wrong bit for each output pin for one test case. For the practical approximation of applications it makes sense to define an upper limit of this error probability, based on quality constraints, for instance a minimal required PSNR for a video application. Usually this minimum quality is defined on higher layers, at the application level. As we will see later, this information has to be transferred down to the actual circuit level in order to approximate it. We can see that for the outputs of a circuit it makes sense to measure error rates.

This work focuses mainly on two types of approximations. The first one is the functional approximation at register level. Functional approximation means that registers are removed (and hence all fanin that only has a connection to that register), if their influence on the circuit outputs is below a defined constraint, as shown in Figure 3.2. Assuming that the activity factor of the signal at the register input $\alpha = 0.5$, one can model the removal of that register by flipping the value of the register with a probability of 0.5. For more a more accurate model the activity factors, respectively the 1/0 probability has to be estimated or measured and the error probability has to be changed accordingly. The second technique is voltage over-scaling. Voltage over-scaling means that one path, defined by a start register and an end register, is operated at a lower voltage, so that the clock period is shorter than the propagation delay. Whether this timing violation results in an erroneous value at the register input (i.e. a new value does not arrive in time) depends on the logic values at all gate inputs involved in this path and the Boolean masking of the gates involved. Hence the error at the register input can be expressed as an error probability that is depending on the circuit inputs, i.e. the test vectors, as shown in Figure 3.3. We can see that both approximation techniques can be modeled as bit-flip probabilities at registers. The list of effects that can be modeled at register level is not limited to the two presented ones. Herkersdorf et al.[23, 110] identified in their work the register level as the common point to model a variety of low level effects that result in faults (see again Figure 2.5). The implementation of that probability-awareness

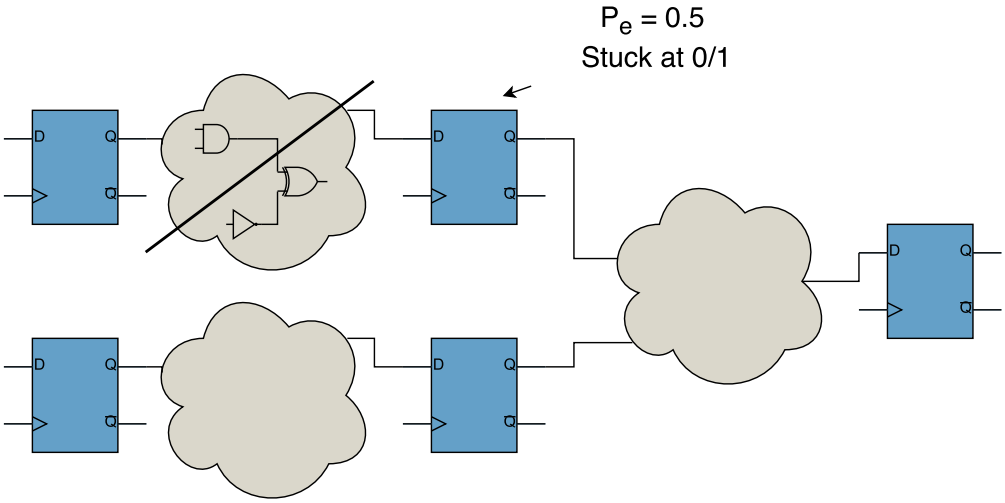


Figure 3.2: Functional approximation of circuits with "circuit pruning" modeled as an error rate at the input of the following flip-flop

clearly differentiates this work from others. Not only by modeling approximations as bit-flips with an error probability at register-level, but also by performing fault injection experiments based on error probabilities [141]. Existing fault injection systems, hardware or software based, inject faults at specified locations (usually only one at a time) and time instants, based on defined injection plans. These systems try to observe whether the fault is propagated to the output or not. The emulator presented in this work however, assigns error probabilities to each register in the circuit, where all can be different, and tries to measure the resulting error rate at the output pins.

3.2 FPGA-based probability-aware Fault Emulation - faultify

In this section the proposed FPGA-based fault emulation system, called "faultify", will be presented in detail. Means on how to efficiently control the emulator, hence develop injection strategies, in order to approximate a circuit, will be explained in the next Chapters 4 and 5. The implementation has actually evolved and improved over the time of this work. The particular steps of improvement will be explained as well, as they demonstrate the problems that had to be overcome. The general structure of an FPGA-based emulation system has already be explained before. At first the software-side emulation API will be explained as it demonstrates the desired functionality of the hardware side.

3.2.1 Software-side Emulation API

The basic API on the software side is very simple. Complex analysis algorithms that are required for the actual approximation are built on top of it. This abstraction into

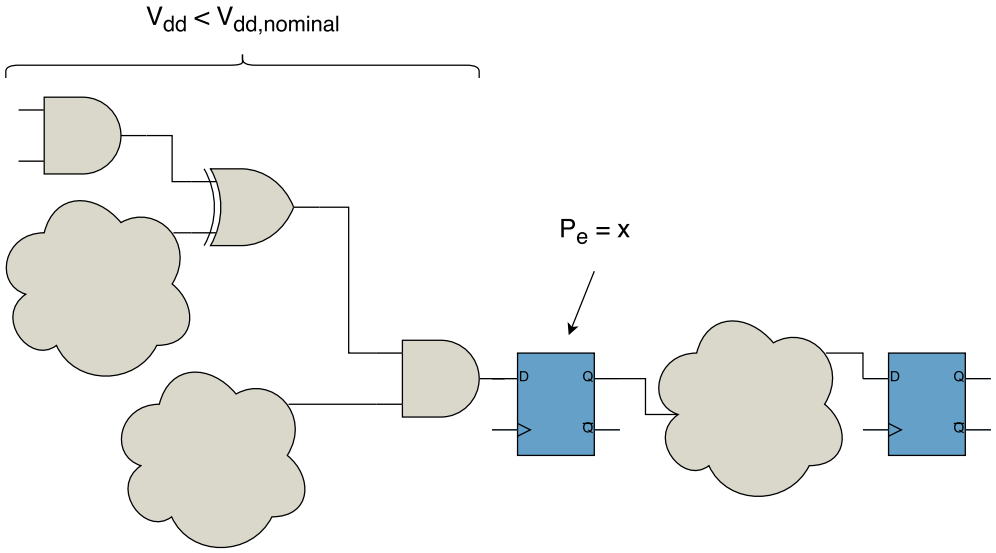


Figure 3.3: Voltage over-scaling as a further approximation technique that can be modeled as error rates at the register inputs

layers allows a simple implementation of new algorithms. Furthermore, it makes the analysis algorithms independent from the emulation hardware. As it will be shown later in this section, the emulator has been implemented on different hardware platforms. By using different communication libraries on the software side, the algorithms developed on higher levels are compatible with any hardware. The low-level API is developed in C and presents four simple functions. Two functions are used to open and close the connection to the emulator, as shown in Listing 3.1 and 3.2.

```
1 int faultify_open(struct faultify_handle ctx);
```

Listing 3.1: API call to open a connection to the emulator

```
1 int faultify_close(struct faultify_handle ctx);
```

Listing 3.2: API call to close a connection to the emulator

The configuration, hence the assignment of error probabilities to registers is shown in Figure 3.3. It is performed by simply providing an array of error probabilities, one for each register. Hence, it is very important to keep the order that is defined during instrumentation. We will see later how the order originates during instrumentation.

```
1 int faultify_inject(struct faultify_handle ctx,
2                      double * pe);
```

Listing 3.3: API call to configure the fault injection

The last API call required to perform a fault emulation is shown in Listing 3.4. The command starts the emulation and runs it for exactly *num_cycles* clock cycles. Once finished, an array is returned, containing the number of faults observed at each circuit output during the run.

```
1  int faultify_run(struct faultify_handle ctx,
2                   uint32_t num_cycles,
3                   uint32_t * result);
```

Listing 3.4: API call to run the fault injection

The four presented API function offer very limited but fundamental functionality to control the emulator. Most other functionality and algorithms can be built on top of these. In the following the required hardware implementation in order to realize these API commands will be presented.

3.2.2 Hardware Implementation

As already mentioned, the hardware implementation has evolved over the time, improvements have been added and it has been made more generic and independent from specific hardware.

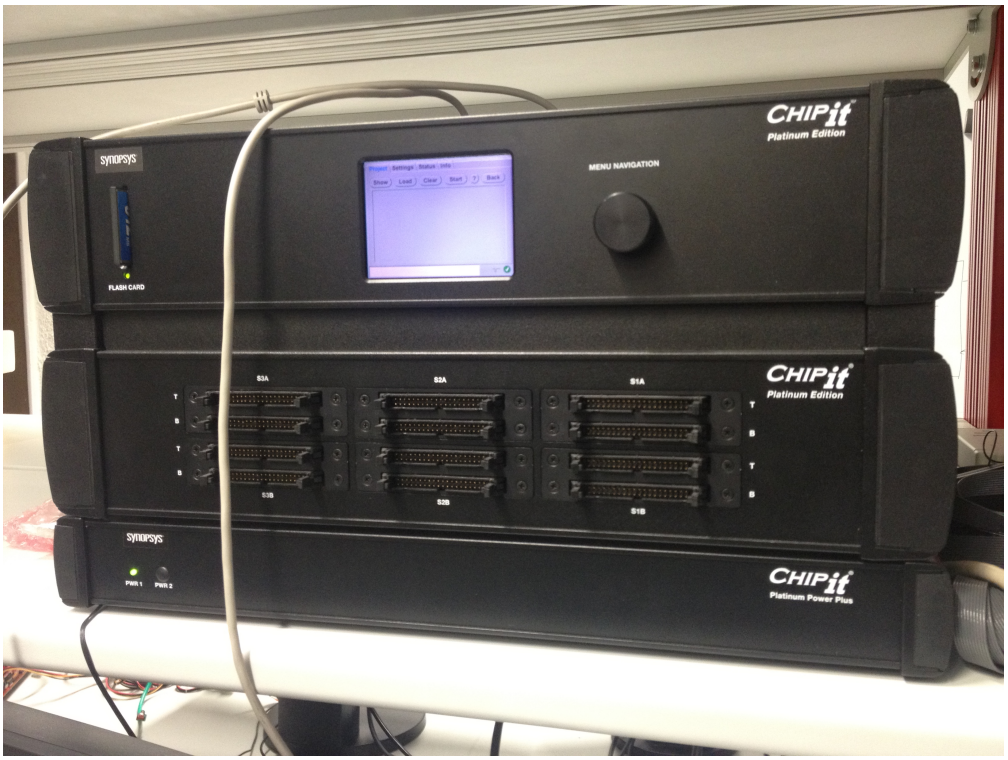


Figure 3.4: The Synopsys "CHIP*it*" simulation system

Synopsys CHIP*it* Specific Implementation The first version of the emulator has been developed for the Synopsys CHIP*it* Platinum emulation system, as shown in Figure 3.4. The system consists of six Xilinx Virtex-5 XC5VLX330 FPGAs that are connected together, appearing as one single system, hence offering enough programmable logic to emulate even large circuits. Additional key features are the following:

- $6 \cdot$ Virtex-5 LX330
  - $6 \cdot 4 \cdot 51840$ LUT6s
  - $6 \cdot 4 \cdot 51840$ FDs
- $6 \cdot 8$ MByte SSRAM
- Various extension boards
- Universal-Multi-Resource (UMR) bus
  - Used for download of bitstreams
  - Clock/Reset configuration
  - User-defined applications
  - 55 MB/s net-rate

The CHIP*it* system has a dedicated interface to the host PC, the Universal Multi-Resource Bus (UMR-BUS). This interface is on the one hand used to download bitstreams onto the FPGAs, configuring clock and reset generators, and for debugging purposes. On the other hand it can be used to send and receive any kind of user data. The net data-rate is roughly 55 MByte/s which makes it suitable for the simulator as it poses no potential communication bottleneck. This interface is used to transmit the configuration, i.e. the error probabilities to the simulator and to control the simulation, i.e. configuring the number of simulation runs and transmitting back the results of the golden circuit and the circuit-under-test. On the host PC side, the CHIP*it* system is connected via a PCIe card. Interfacing the UMR-BUS is done with the help of a shared library provided by the vendor. The API mentioned above, is built on top of the provided library. A block diagram of the hardware running on the CHIP*it* system is shown in Figure 3.5. Apart from the golden circuit and the circuit-under-test two other modules can be seen. The simulation controller is responsible for interpreting commands sent from the host PC, loading probabilities into the corresponding bit-error generators, controlling the simulation, sending back the results and generating the test vectors. The UMR controller is the interface controller for the connection between host PC and FPGA development system. A more detailed structural diagram can be seen in Figure 3.6, giving a more detailed insight into the implementation. In order to allow an automation of the generation of the HDL files the circuits have to be encapsulated into a wrapper so that they provide a general interface to the outside world. In this implementation the golden reference circuit is defined by the following interface ports:

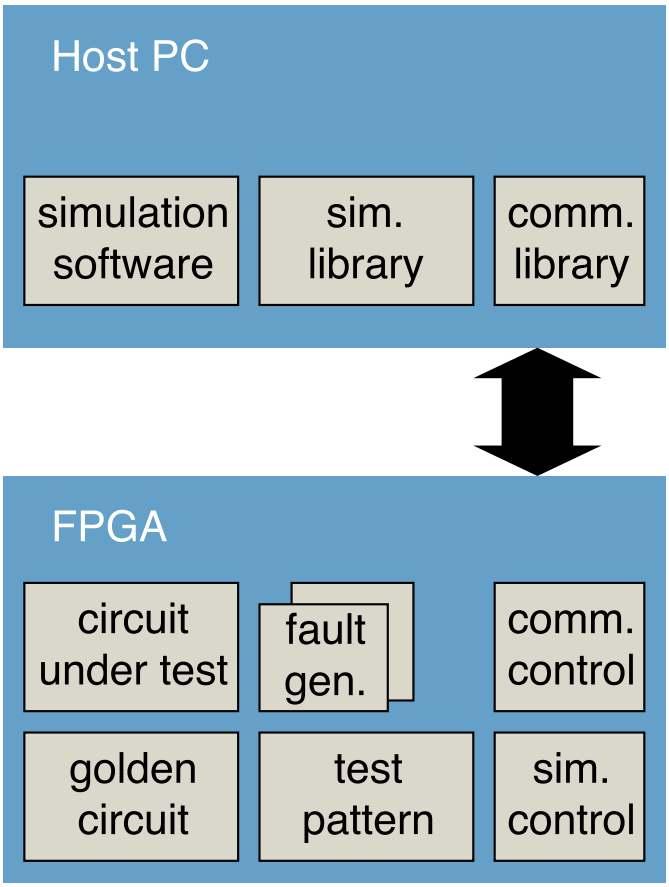


Figure 3.5: Simple block diagram of the fault emulator running on the CHIP*it* system

```
1  clk : in std_logic;
2  rst : in std_logic;
3  testvector : in std_logic_vector(numIn-1 downto 0);
4  resultvector : out std_logic_vector(numOut-1 downto 0);
```

Listing 3.5: VHDL interface ports of the golden circuit wrapper

The parameter "numIn" corresponds to the number of input pins and the "numOut" to the number of output pins. As the emulator focuses on the analysis of sequential circuits, the clock input "clk" provides a mean to supply a clock to the circuit. In this first version of the emulator, the whole systems is clocked with a single clock frequency, determined by the frequency of the UMR bus interface, running at 70 MHz. The reset input "rst" is optional. Circuits that provide means to reset the flip-flops should however use the input, as it speeds up the reset phase between fault injection experiments. Once faults are injected into a circuit, it is sometimes difficult to restore a circuit to its original state if flip-flops without a reset are used and without re-flashing the bitstream. The generation of the VHDL wrapper for the circuit under test is part of the circuit instrumentation, being described in Section 3.2.3. All inputs of the circuit are concatenated and connected

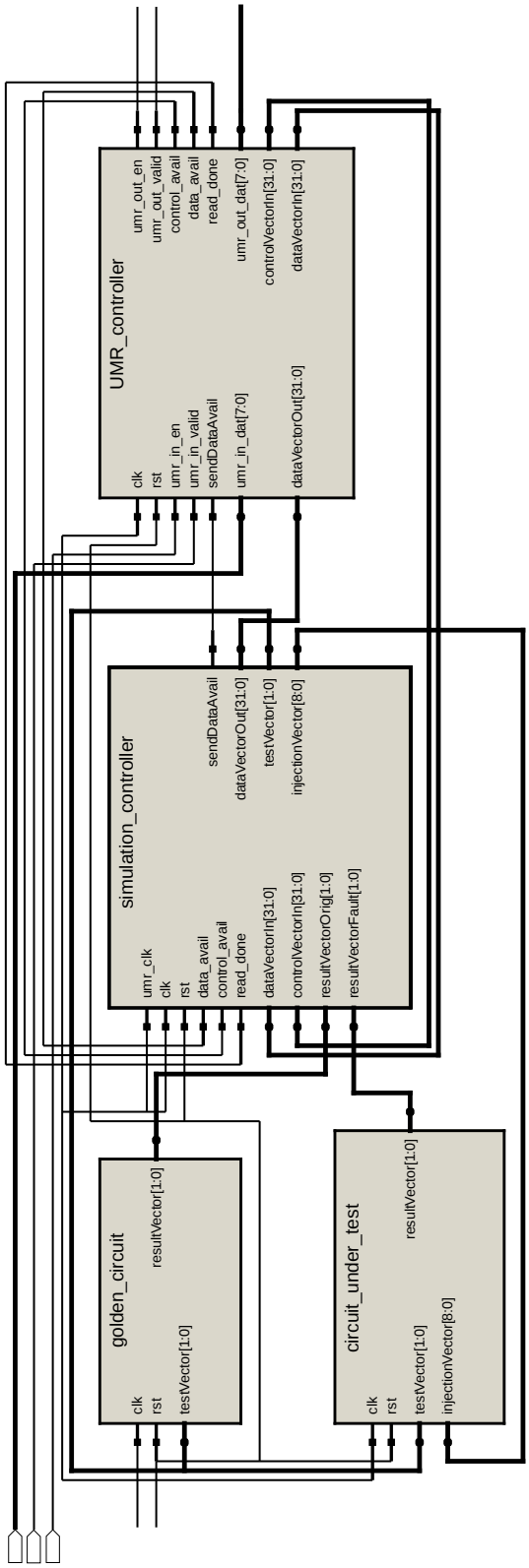


Figure 3.6: Detailed block diagram of the fault emulator running on the CHIP*it* system

to the input vector "inputvector". In the simplest case, this input vector is connected to random bit generator, providing random stimuli. For some circuits this is would be a valid input pattern. But even for other circuits this can be a valid method. However, for most application, more complex input patterns are applied, for instance if a protocol has to be fulfilled. This can be done for instance by connecting a state-machine that is still providing random data but is complying to a required protocol. It is important to note that in this case each circuit need its own test-pattern implementation. An automated flow is only possible when assigning completely random data to the input pins. All output pins are connected to a vector called "outputvector". This vector can then be compared in each clock cycle with the output of the faulty circuit, the one where faults are injected. This "circuit_under_test" has the same input and output ports. Additionally it contains another input vector, the "injectionvector":

```
1  injectionvector : out std_logic_vector(numInj-1 downto 0);
```

Listing 3.6: Additional VHDL interface port of the circuit_under_test wrapper

This parallel vector is "numInj" bits wide, where "numInj" corresponds to the number of fault injection positions. In our case usually each register can be flipped, hence one bit is provided for each register in the circuit. The idea is that if a "1" is assigned, the bit of the register is flipped, otherwise not. The mechanism, how the bit is actually flipped, will be presented in Section 3.2.4.

PCIe-based Interface Additional to the previously presented UMR interface, a PCI express based interfaces has been developed to be independent from the proprietary UMR bus interface [137], as shown in Figure 3.7. On the software, as well as on the hardware interface level, the interface has been adapted to behave the same way as the previously presented one. The interface is based on "RIFFA" an open PCI express interface developed for Xilinx as well as Altera FPGAs [111]. The implementation offers a very fast, and more important, easy to use FIFO based interface on top of the PCI express bus. Hence the backend could be transparently implemented into the existing environment. Additionally, the interface offers the possibility to use multiple simultaneous independent channels. As we have already seen, the generation of realistic test vectors is crucial for a realistic emulation of a circuit. The additional channels of the interface can be used for instance to transfer the test vectors to the circuit from the PC. The PCI express interface offers enough throughput to transfer large amounts of data from the host PC to the FPGA in a short period of time. Similar the output of the circuit-under-test can be transferred back to the host PC, and e.g. its quality can be evaluated there. In the scope of this thesis a demonstration application has been developed, where the additional independent channels have been used exactly for this purpose. In this demo a h.264 video decoder has been approximated. Encoded h.264 video data is transferred at real-time to the FPGA. The decoder is decoding the data, and the decoded video data is transferred back to the host PC in real-time. This allows on the one hand to provide the circuit with accurate test data to create a realistic test environment, and on the other hand to analyze the effects of the fault injection on the host PC, which is in most cases much

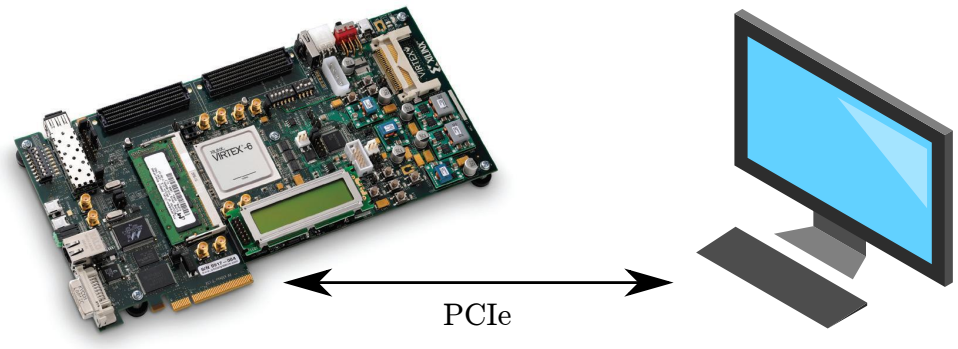


Figure 3.7: Xilinx ML605 evaluation board connected via PCIe to a host PC

more flexible. In case of the h.264 video decoder it allows to calculate the PSNR on the host PC.

Generic Ethernet-based Implementation Besides the PCI express based variant an even more generic implementation has been developed. This variant not only introduces yet another, even more generic interface, but also offers a more generic way to interface the circuit-under-test. This makes the emulator comparably easy to adapt to new circuits-under-test, and makes it easy to provide test data and read back results. Although in general the components of the emulator remain the same as above, they are now arranged around a Xilinx "microblaze" based microprocessor system. The microblaze processors fulfills several tasks. Most important, it is running the TCP/IP stack required for the interface. The introduction of Ethernet as the communication interface offers great flexibility and compatibility. For instance it allows to be connected simply to a network, at another location as the engineer is located. Additionally, running on the CPU is the control software, which hence replaces the communication controller, previously implemented in hardware. The task is the same. It interprets commands arriving from the interface, and configures the emulator accordingly. The interface on the host PC remains the same as previously introduced. However, due to the generic microprocessor additional, more complex tasks can be performed, even independent of the host PC. Furthermore, the microprocessor system can be used to provide realistic test data, either generated on the system itself or provided from the host system, where the microblaze serves as a bridge. The output of the circuit-under test can be evaluated directly on the microprocessor or sent back to the host PC. Similar to the PCI express based interface, the Ethernet interface by nature provides concurrent parallel communication channels. Based on the port number the channels can be separated. The TCP/IP stack running on the microblaze CPU is the "lightweight IP" (lwip) implementation [112]. The MAC layer is provided by Xilinx. This implementation is overcoming many limitations of the other presented ones and it is as generic as possible. However, even though it offers a very simple interface on the control side and a generic interface, the AXI bus, on the

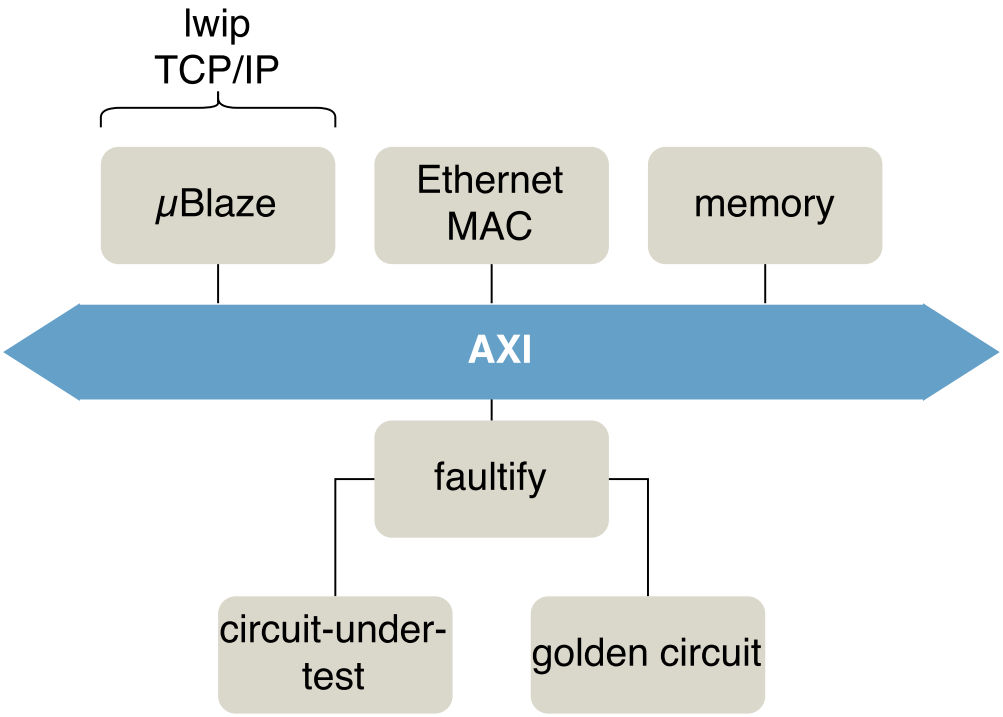


Figure 3.8: A block diagram of the AXI-based emulator system with an Ethernet control interface

hardware side, the emulator still requires a lot of manual engineering. Each circuit-under-test requires its own implementation. Each circuit requires its own test patterns and its own quality metric. This is a general limitation not in particular of FPGA-based fault emulation systems. However, the system presented here, offers an easy-to-use structure with minimal implementation effort.

3.2.3 Circuit Instrumentation

As already mentioned above, circuit instrumentation is the process that replaces circuit elements (flip-flops in our case) by functional equivalents, that additionally allow to flip the value of the bit. In order to efficiently perform the instrumentation it has to be automated. A manual modification of the source code or the circuit netlist would not be suitable for large circuits. The circuit instrumentation presented in this work is performed on synthesized netlists of the circuit-under-test. Hence, the first step is the synthesis of the circuit, if it available as a HDL description, as depicted in Figure 3.9. As already mentioned, FPGA-based emulation can be used to emulate FPGA targeted designs, as well as ASIC designs. The common case however is the emulation of ASIC circuits. Furthermore, not all approximation techniques are available on FPGAs, like voltage over-scaling. Nevertheless, functional approximation techniques like circuit pruning can be applied on FPGA designs as well as on ASIC designs. Depending on the target

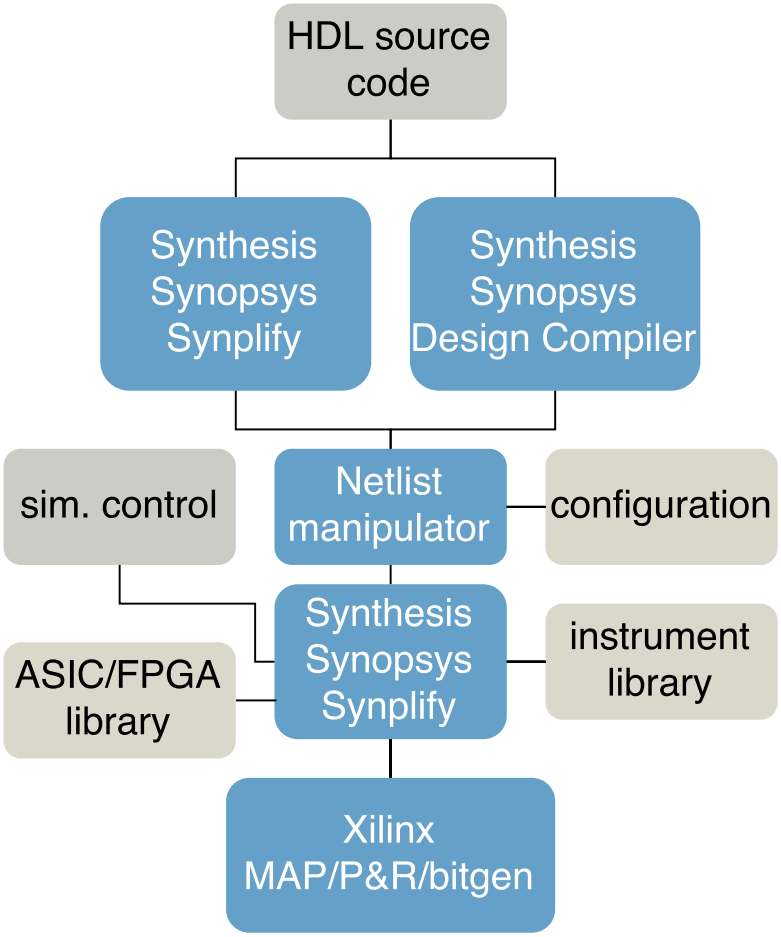


Figure 3.9: Instrumentation flow of the proposed emulator

architecture either Synopsys "Synplify" or "Design Compiler" is used to synthesize the designs. In general, any other synthesis tool can be used. Important for the next step is that the synthesized netlist is in the Verilog netlist format. The netlist in Verilog format consists of instances of library primitives. In case of Xilinx FPGAs these cells could be for example "FD" flip-flops or "LUT4" look-up tables. In case of netlists synthesized with Synopsys "Design Compiler" these cells could be for example "DFFX1" flip-flops or "GTECH AND2" 2-Input AND gates from a generic library. The task of the "Netlist Manipulator" is to replace the instances by those replacements that allow for an error injection. Unfortunately it is not sufficient to simply replace the string of the component instantiated. Additionally, the control wire has to be routed from the instantiated element all the way up to the toplevel. Hence, a simple string replacement is not sufficient, instead the complete circuit has to be parsed and a syntax tree has to be build. Fortunately, not the complete Verilog syntax has to be supported. Only the subset required for Verilog netlists has to be supported. The tool is written in C++. Its parser is developed using the "flex" and "GNU bison" infrastructure [113]. The syntax of how to run the manipulator

is shown in Listing 3.7.

```
1 faultify_instrumentation <netlist> <toplevel>
2 <clock_port> <reset_port> [ <original_instance> <replacement_instance> ...]
```

Listing 3.7: Netlist manipulator syntax

As an input from the user it requires the path to the netlist, the toplevel name, the name of the system clock, and the system reset. Clock and reset signals are then automatically connected to the simulator clock and reset generators. Furthermore, all cell types that should be replaced by the software and the corresponding instrument have to be specified. An exemplary replacement for a Xilinx flip-flop cell can be seen in Figure 3.10. One can see that by simply adding an XOR gate, the value stored in the flip-flop can be flipped. For ASIC technologies the procedure would be the same. When setting the second input

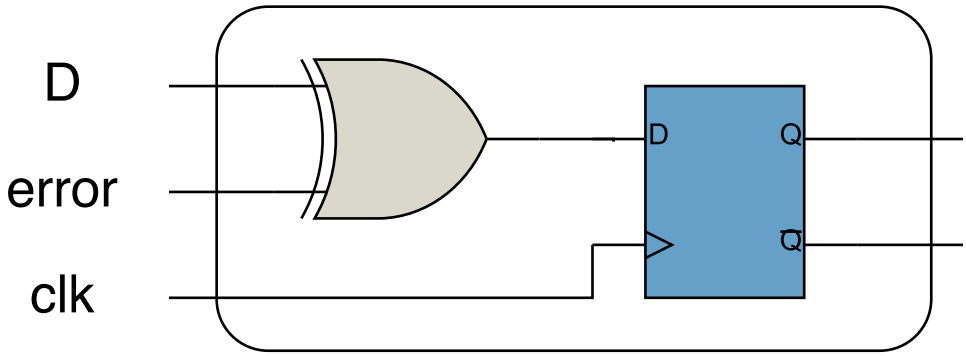


Figure 3.10: Xilinx FD flip-flop instrument enabling fault injection

of the XOR gate to "1" the value flips, otherwise not. This wire has to be routed from deep within the circuit to the toplevel. There it gets connected to the vector "injectionvector" with the help of a wrapper, mentioned above. The modified syntax tree is then written out as the instrumented Verilog netlist. This modified netlist can then be synthesized again, together with the control and interface logic, mentioned above. When emulating a circuit designated to be implemented on an ASIC, the synthesis of the emulator is not so straight-forward. The circuit actually has to be translated to run on an FPGA. In order to do so, a library implementing technology specific primitives on FPGA primitives has been developed. This library can then be used in conjunction with the instrumented netlist and synthesized for the FPGA. In this work the ASIC technology has been limited to Synopsys generic 90nm technology. An exemplary model of a "DFFX1" flip-flop of the Synopsys technology library is shown in Listing 3.8

```
1 module DFFX1 (D,CLK,Q,QN);
2
3 output Q,QN;
4 input D,CLK;
5 wire Qtemp;
6
```

```
 7  assign QN = ~Qtemp;
 8  assign Q = Qtemp;
 9  FD XIL_FD (.D(D),.C(CLK),.Q(Qtemp));
10
11  endmodule // DFFX1
```

Listing 3.8: FPGA model of a "DFFX1" flip-flop of the Synopsys generic 90nm library

For more complex primitives, the translation is more difficult. The FPGA implementation of a "FADDX1" full-adder is shown in Listing 3.9.

```
 1  module FADDX1 (A,B,CI,CO,S);
 2   input A;
 3   input B;
 4   input CI;
 5   output CO;
 6   output S;
 7
 8   reg CO,S;
 9
10   always @ (A or B or CI)
11   begin
12   {CO,S} <= A + B + CI;
13   end
14
15  endmodule
```

Listing 3.9: FPGA model of a "FADDX1" full-adder of the Synopsys generic 90nm library

As the injection if performed at register-transfer level, i.e. into flip-flops, a functional modeling of the primitives is sufficient, not gate equivalent replacement is required.

3.2.4 Parallel Bit-error Generation

The distinct feature of the emulator presented in this work is the ability to inject faults based on probabilities, where the error probability for each register can be different. As it will be shown in Chapter 3.2.5, it is not advisable to generate bit-errors, i.e. the time instant when a fault shall occur, in software running on the host PC. For thousands of registers, the complexity is simply too high. Hence, in order to allow for a very fast emulation speed the faults have to be generated in hardware on the FPGA. As it will be shown later, this clearly results in a not negligible hardware overhead. An exemplary bit-error generator is shown in Figure 3.11. A probabilistic bit-error consists of a parallel random number generator, i.e. all bits of the random number can be read in parallel in each clock cycle. This randomly generated number is compared with a number stored in a register. If the random number is larger than the number in the register, a "1" is generated, otherwise not. This process is called a Bernoulli trial, named after Jakob I. Bernoulli [114]. A Bernoulli random number is "1" with a probability of $P(X = 1) = \pi$ and "0" with a probability of $P(X = 0) = 1-\pi$. In the example in Figure 3.11 the random numbers generated are 24 bits wide. Hence, the probability can be set with a resolution of $6*10^{-8}$. For example, to generate a bit with a probability $p = 0.5$, the threshold input of

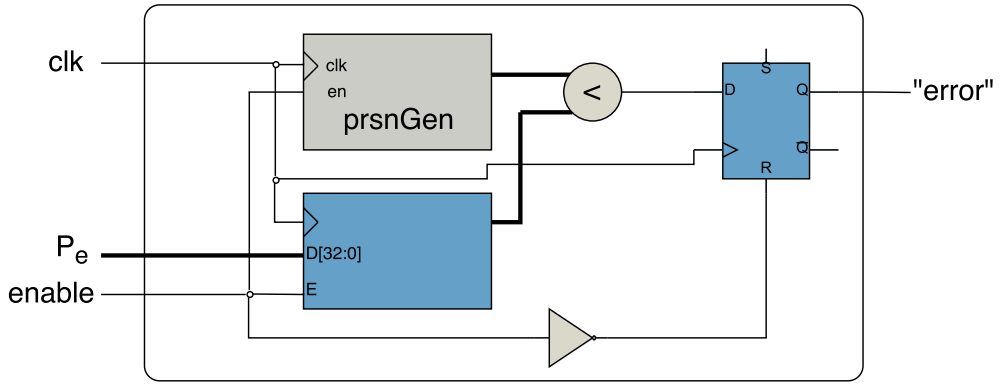


Figure 3.11: Parallel 24-bit bit error generation

the generator needs to be set to 8388608 (2^{23}). The random numbers itself are generated by a linear feedback shift register with maximum length, as proposed in [115]. A random number generator based on shift-registers is actually a pseudo random number generator, as it has a predetermined sequence. Clearly, the statistical properties of a LFSR as a random number generator are not optimal. However, as the numbers are only used to control a Monte Carlo experiment and not used for cryptography applications, the limitations are acceptable. The exemplary implementation of the bit-error generators is 24-bit wide. The implementation of the corresponding LFSR is shown in Figure 3.12.

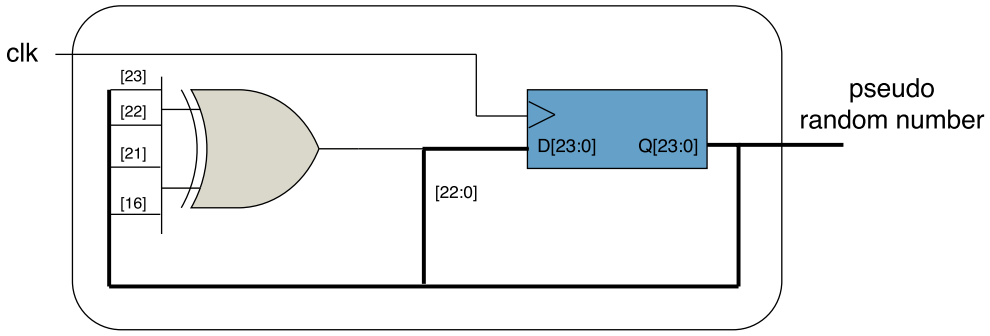


Figure 3.12: 24-bit LFSR with maximum length

While the resolution offered by a 24-bit wide bit-error generator seems sufficient, one has to keep in mind that the random numbers repeat each 2^{24} clock cycles. Hence, when running the emulator at 1 MHz, the sequence would repeat every $\sim$ 16 seconds. For most fault injection experiments this periodicity is not sufficient. In order to make the bit-error generation at each register independent of each other the start value, the seed, has to be different. This is done at synthesis time. The reference value with which the random number is compared is stored in a register. The communication controller receives the value from the host PC. A direct, parallel assignment, of the x-bit wide value would result in a very congested design making routing very difficult. This is why the value in this design is loaded through a scan-chain into the register. This reduces the

number of required routes a lot.

For a first evaluation of a probability-aware fault emulation system, the usage of these parallel bit-error generators was sufficient. This implementation poses however one main limitations, which is that the area requirement is very large. The requirements for the 24-bit wide example, mentioned above can be seen in Table 3.1. When extending the

Module	LUTs	FDs
probability number generator	26	49
injection scan chain	0	24

Table 3.1: The resource utilization of the 24-bit wide pseudo random number generator and the injection scan chain

periodicity of the random number generators, for instance by extending the shift register length to 64 bit, the hardware requirement would increase accordingly. Note, that one probabilistic bit-error generator is required for each flip-flop. The benefit of this implementation is however, that one bit-error can be generated in each clock cycle. This makes this implementation the fastest one possible. The performance will be evaluated later in Section 3.2.9.

3.2.5 Host vs. FPGA Generated Bit-Errors

Instead of generating the faults directly on the FPGA, it would be also possible to generate them on the host PC. The benefit of doing this would be that no additional hardware would be required to generate the faults. As it will be shown later in this section, the hardware overhead due to generating faults just-in-time on the hardware is non-trivial. A modified version of the emulator has been developed in order to quantify the benefit and reveal potential problems [133]. Other works, like López-Ongil et al. [80], proposed a similar approach by storing the fault lists in the memory on the FPGA. Even though their approach is not aware of probabilities, the general idea is the same. On the host PC the location and the time instant of the faults is generated. This information is then transferred onto the FPGA and stored in some kind of memory. An extended block diagram of the UMR-based emulator is shown in Figure 3.13. The type of memory that is used depends on the implementation. In this work the test implementation uses block-ram on the FPGA. A Xilinx Virtex-5 LX330 FPGA used here, offers 288 RAM entities each providing 36 kBit of memory. Each of these blocks is divided again into 18 kBit blocks. Each block has a depth of 512 and is 32 bits wide (unused parity bits not counted). Each flip-flop requires one bit per clock cycle to store whether a fault is injected or not. Hence, on this type of FPGA, faults can be injected in up to $288 * 2 * 32 = 18432$ flip-flops. The organization is shown in Figure 3.14. Due to the given depth of the block ram, up to 512 clock cycles can be stored simultaneously. Emulating a maximum of 512 clock cycles is not sufficient for most test-cases. This is why the memory in this implementation has the function of a buffer. In this implementation the fault table is transferred and updated on-the-fly at run-time of the emulation. Even though the

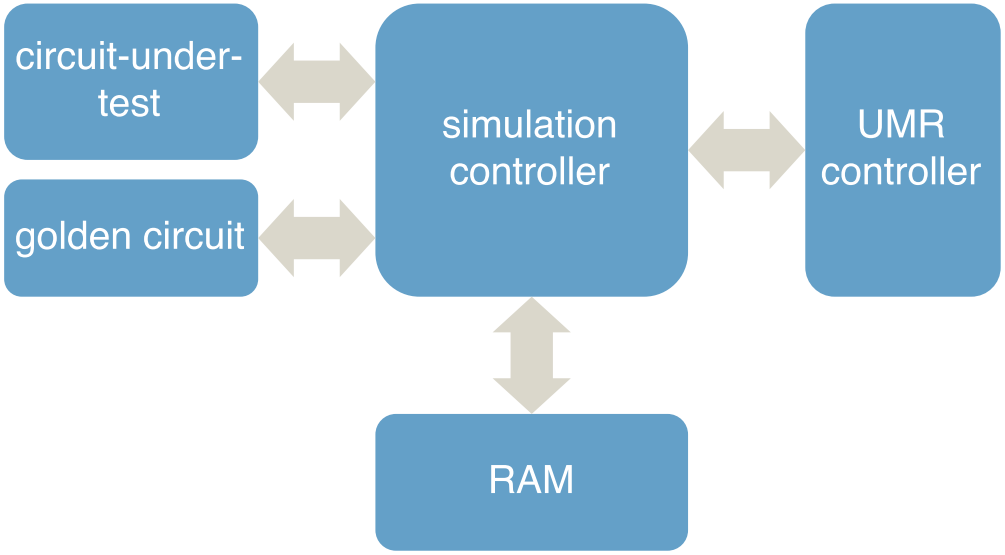


Figure 3.13: Block diagram of the emulator, supporting on-the-fly injection of bit-errors generated on the host PC

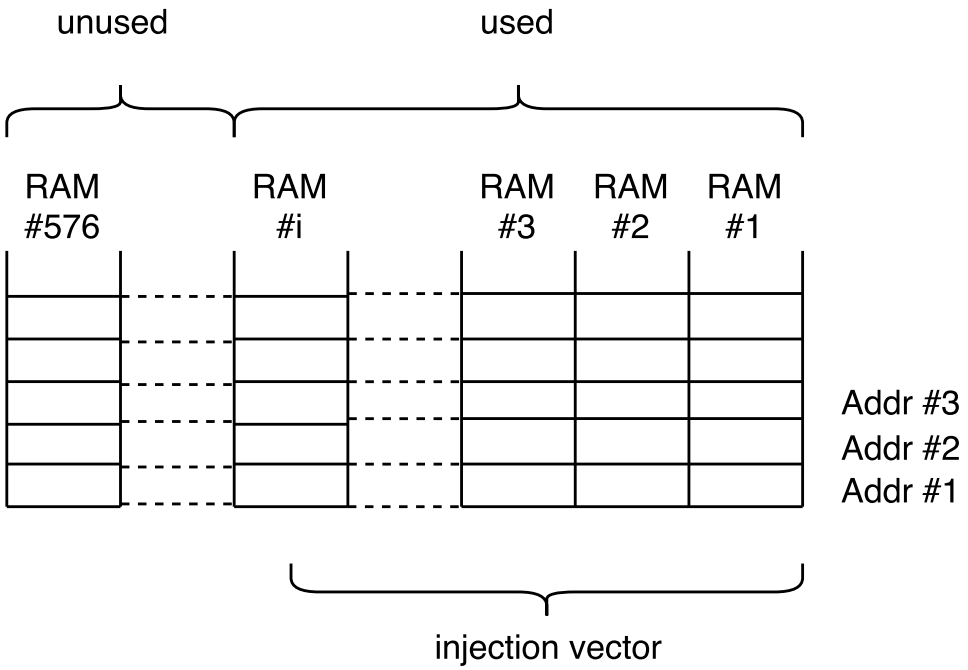


Figure 3.14: Memory organization, storing injection location and time on Xilinx FPGAs when generated on host PC

interfaces used in this work offer a high bandwidth, they are usually not sufficiently fast to guarantee an interruption-free execution. Instead, every time the buffer is getting

empty the circuit emulation has to be stopped until new data has arrive from the host PC. One can imagine that this results in a slowdown of the emulation. The speed of the emulation is now also becoming dependent of the number of flip-flops, being part of the fault injection. The more flip-flops are part of the fault injection, the more data has to be transferred between host PC and FPGA. The speed of this variant will be evaluated later in 3.2.9. Clearly there are several ways to implement an emulator that receives the injection table on-the-fly from the host PC. Especially when analyzing the effects of soft-errors where the probabilities are very small, another approach would be more beneficial. Instead of transmitting the information whether an fault is injected or not for each clock cycle, one could simply transfer the time-stamp and the location of a fault. Especially for small error probabilities, this would result in much less data to be transferred. This has been chosen as the implementation serving as a reference for the comparison. However, as it pointed out, the transfer of fault tables is not the main bottleneck. When performing probability-aware fault emulations for Approximate Computing, the goal is usually to find out which parts can be approximated for a given quality constraint, and which not. Hence, usually thousands of emulations will be executed each with different settings, i.e. fault injection tables, in order to gain this information. It pointed out that the generation of these tables is actually as time consuming as transferring it, as we will see later in Section 3.2.9. The reason for this is that due to the probability-awareness, each injection point requires a separate and independent random number generator. Generating several million random numbers (one for each clock cycle) even for one injection point is very time consuming. Repeating this for each injection point increases the time accordingly. Not only is the generation complex but also the amount of memory required on the host PC is increasing dramatically. These are the main reasons why the host-based generation has not been pursued any further. Nevertheless, this is a very interesting approach, as the hardware overhead on the FPGA is very low. It is left open for future work to develop a more efficient generation of faults on the host PC to overcome the presented limitations.

3.2.6 Improved Bit-error Generation

As it has been shown above, the methods presented to generate faults in the circuit, until now, are either very fast but have a very large are consumption or vice versa. Actually both variants presented are not practically applicable due to their limitations. A compromise between hardware overhead and performance had to be found to efficiently emulate even large circuits on a FPGA, which has been presented in [142].

The most complex block of the fault generation, as we have already seen, is the generation of random numbers. This applies for the host-based as well as FPGA-based generation. Even though the statistical properties are weak, "Linear Feedback Shift Registers" (LFSR) are practically the only possibility due to their small area footprint [115]. Their randomness should be enough for such an application, especially as we are able to change the seed of each generator at any time. As it has been already explained above, the length of the LFSR has to be sufficiently large so that the random numbers do not repeat within an experiment. For the improved bit-error generation a length of at least 64-bit has been desired. A 64-bit shift register with maximum length has a period of

$2^{64}-1$ which corresponds to about 5850 years when running at 100 MHz. This should be sufficient for most emulations, even when using multiple instances with different seeds, and hence sharing the random numbers [116]. Based on the random number and the specified probability a bit-error is generated - or not. This process can be seen a Bernoulli experiment, as already explained above. The parallel approach presented earlier is able to generate one bit-error per clock cycle, by generating on random number each clock cycle and compare it within the same cycle with the desired probability value. If the random number is smaller than the probability value an error is generated, otherwise not. However, the key for reducing the required resources is to generate bit-errors in a serial manner as shown in Figure 3.15.

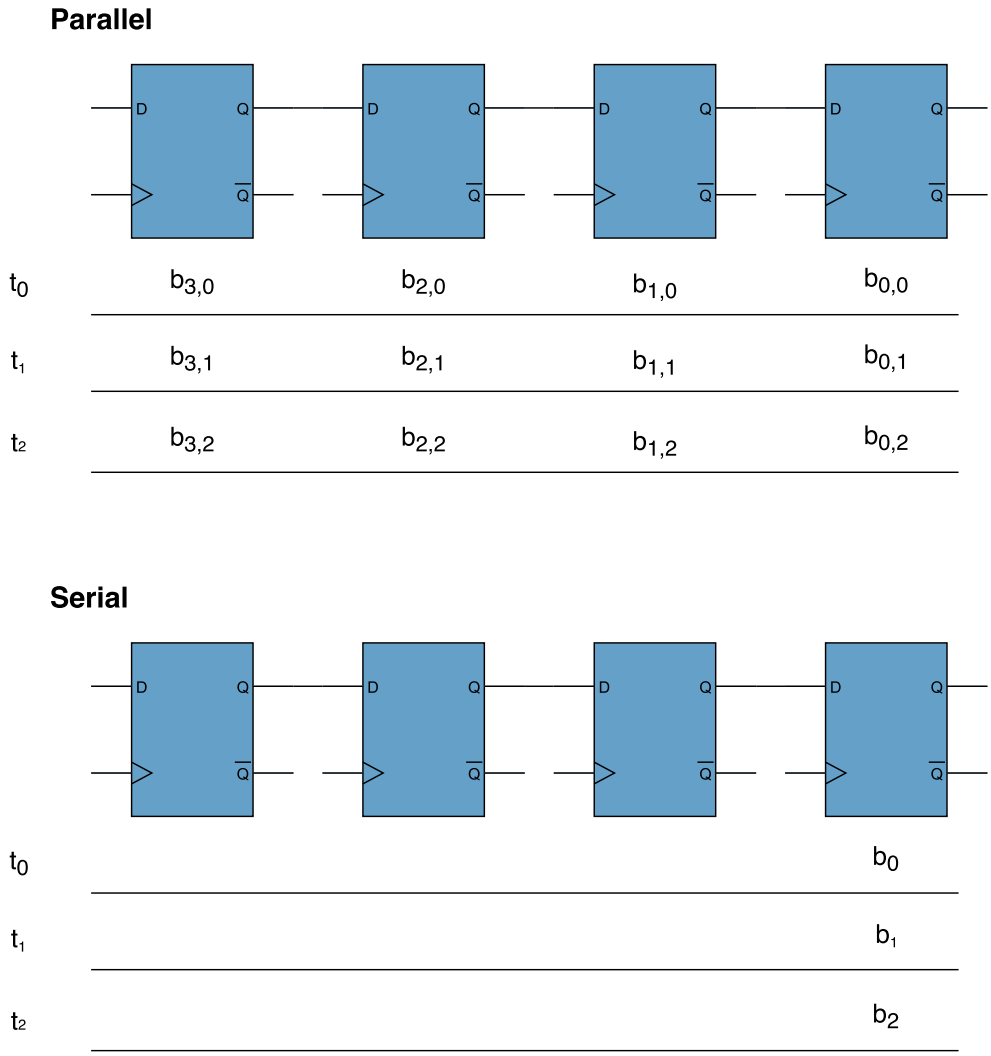


Figure 3.15: Serial versus parallel random number generation

When generating random numbers in a parallel manner, each bit has to be directly

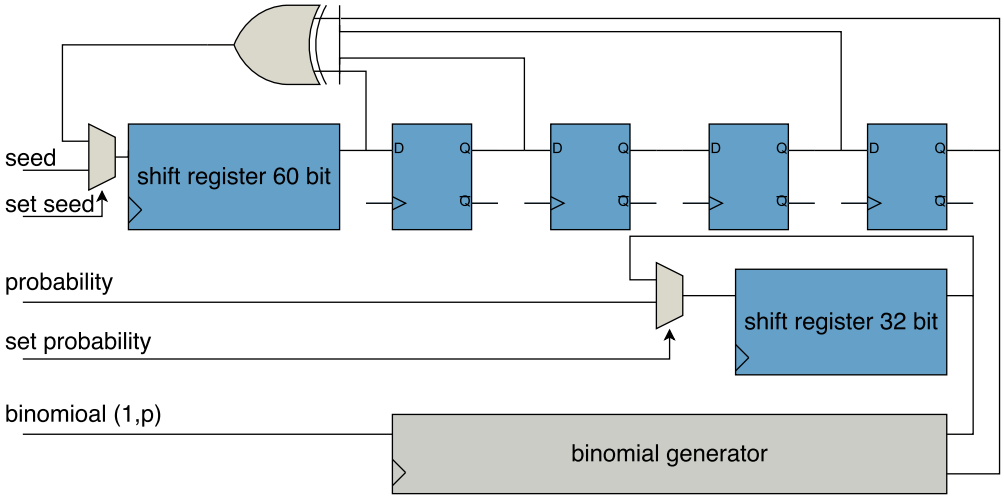


Figure 3.16: Improved serial bit-error generation

accessible. A bit is stored in a flip-flop on the FPGA. When generating random numbers in a serial fashion, it takes x cycles to generate a x-bit wide number. Accordingly, the individual bits of the shift register do not have to be accessible. Only at the last bit the value has to be read and at those bits, where a feedback is taken from. This fact can be exploited on Xilinx FPGAs, because most of the chain can be implemented using "SRL" primitives. Only the feedback positions have to be implemented using regular flip-flop primitives. On a Virtex-5 FPGA each "Configurable Logic Block" (CLB) contains one 128-bit wide shift register and 8 Flip-flops. For instance, generating a 64-bit wide random numbers in a serial manner requires one CLB, while it would take about 8 CLBs when generating in a parallel manner. This reduces the area overhead dramatically. Very similar the resources required to compare the random number and the desired probability, hence generating the actual bit, can be reduced. By performing the generation in a serial compared to in a parallel manner reduces the required resources a lot. When generating in parallel, a comparator is required deciding whether the stored value is larger than the random number, having the width of the resolution of the probability. The width of the resolution in the improved bit-error generation is chosen to 32 bits, which gives a resolution of $1.16 * 10^{-10}$, hence the desired error probability can be set very fine granular. With the probabilistic bit-error generators presented in this work a serial binomial generator is used as presented by Devroye et al. [117]. A 32-bit wide comparator using this algorithm can be implemented with a five bit counter and some additional logic. The implementation of the improved serial bit-error generators is shown in Figure 3.16. The price that has to be paid for this very low hardware requirement is that it now takes 32 clock cycles to generate one bit-flip. In other words, the bit error generators have to run 32 times faster than the circuit-under-test and the golden circuit. This trade-off clearly reduces the speed of the emulation. However, at the same time it reduces the required hardware resources a lot. Ultimately, it allows to emulate large circuits at remarkable speed, not only on large prototyping platforms but also on widely

available development platforms. The impact of the serial bit-error generators on speed and area overhead will be discussed in detail in Chapters 3.2.8 and 3.2.9.

3.2.7 Island Concept

There is another way of reducing the area overhead introduced by the random number generators. One feature of the implementations presented up to now is that the error probabilities assigned to the flip-flops can take any value from 0.0 to 0.5, with a resolution defined by the random number generators. Additionally, each flip-flop can be assigned with another error probability. This gives clearly the largest freedom when analyzing the susceptibility of integrated circuits to faults, as all possible error combinations can be theoretically emulated. However, for most emulations this freedom might not be necessary. For instance, one could limit the set of possible error probabilities. In turn this would mean that the assigned error probabilities, modeling circuit approximations, have to be quantized to the next smaller value. This could result in less than optimal approximation results as the optimal approximation might not be determined due to the quantization. For instance, assuming an error probability of $4 * 10^{-5}$ can be tolerated at a register for a given quality constraint. The emulator however tested the circuit with $1*10^{-5}$ and $5*10^{-5}$, as these are the only possible steps in this range. The emulation will now determine that the maximum tolerable error probability at this register is $1*10^{-5}$, as the next step $5*10^{-5}$ is already too high. It might therefore be necessary to find a trade-off between resolution and area. Additionally, one could give up the flexibility that every flip-flop has its own random number stream. By feeding each flip-flop that has been assigned with the same error probability from the same random number generator one could potentially save a lot of resources. These domains, all assigning the same error probability to the registers, will be called "probability islands" in the following. These two ideas have been implemented and their applicability has been evaluated [130, 131]. The most important step is to define which elements of a circuit correspond to which island. A naïve approach could be for instance to test different error probabilities at the individual blocks of a system-on-chip, and hope that for instance a hardware accelerator can tolerate more imprecision, and hence e.g. a lower supply voltage, than a CPU. Clearly, the situation is much more complicated. In an approximated circuit, even individual functional blocks will have several levels of approximations, as all gates of a circuit have a different influence on the functionality. Hence, during a circuit analysis the members of a probability island likely change, as different combinations will be tested in each run. It is therefore necessary to allow a flexible assignment of flip-flops to a probability island. As it will be shown later, this is actually the main limitation of this approach. For the time being it will be assumed that the members of an island are fixed. In order to benefit from the limitation of having a limited set of error probabilities it is required to feed a whole island from a single bit-error generator. A simplified block diagram is shown in Figure 3.17. In each clock cycle a new potential fault with a probability of $p_{e,\text{set}}$ is generated. This fault is then randomly assigned to one of the members of the island.

Clearly the resulting error probability at the flip-flops is not anymore $p_{e,\text{set}}$ but $\frac{p_{e,\text{set}}}{\text{numMem}_i}$, where numMem_i corresponds to the number of members of the island i. In order to ob-

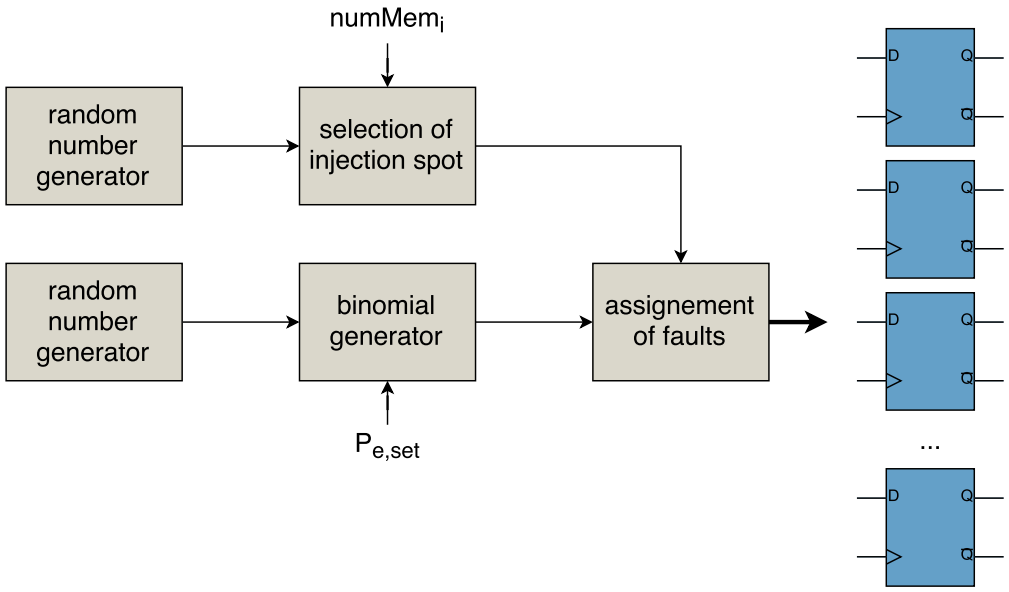


Figure 3.17: Feeding a whole "probability island" with probabilistic bit-errors from a single generator source

serve an error probability of $p_{e,\text{desired}}$ at the flip-flops, the error probability of the bit-error generator has to be set to $p_{e,\text{set}} = p_{e,\text{desired}} \times \text{numMem}_i$. One can see that for larger error probabilities or more members of an island one fault generator is not sufficient as $p_{e,\text{set}}$ cannot be larger than 0.5. The number of required fault generators per voltage island can hence be calculated as:

$$N_{gen} = \lceil \frac{p_{max} \times \text{numMem}_i}{0.5} \rceil \tag{3.1}$$

In the worst case, e.g. when emulating the effects of circuit pruning, i.e. when the error probability is very high, one bit-error generator per flip-flop is required, as it has been previously the case. Similar as before one could think of generating the faults in a serial manner. Instead of serving all flips-flops within one clock cycle, faults could be assigned serially served by one single bit-error generator. This would however slow down the emulation again by a factor corresponding to the number of served flip-flops. If speed is not the most important factor this approach would be a very interesting alternative. However, the main limitation is neither the speed nor the required area of the bit-error generators. The main limitation originates from the required flexibility. As already mentioned, the configuration of the voltage islands is usually changing many times after synthesis. Hence, not only the error probabilities of the islands is changing, but also the composition of an island, its members. In fact, every flip-flop has to be connectable to every probability-island. Hence, a fully configurable switch matrix is required to route the probability islands, as shown in Figure 3.18. It could be shown that when having more than 8 probability islands, the area overhead due to the switch matrix is superimposing the overhead of the bit-error generators. Unfortunately, this puts the saved area overhead

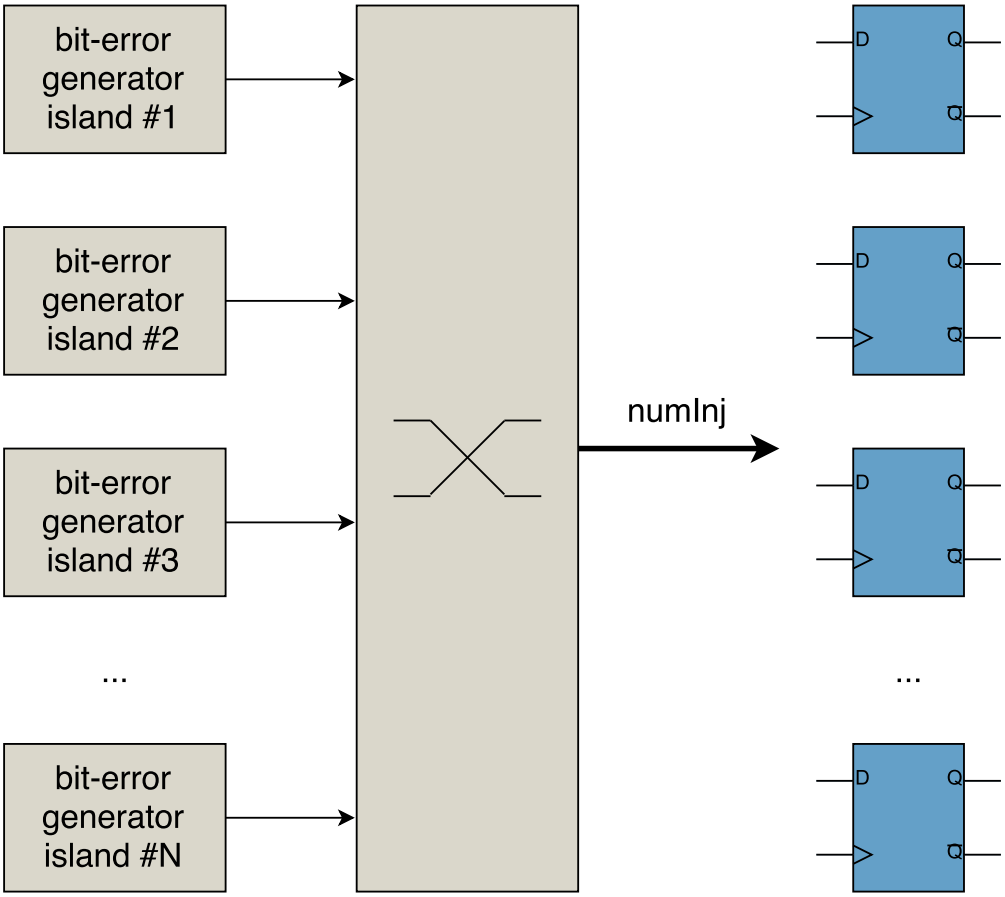


Figure 3.18: Switch matrix connecting each register in the circuit with each "probability island"

due to the massive reduction of random number generators into perspective. Dynamic partial reconfiguration of the FPGA could be used to dynamically reconfigure the switch matrix which would dramatically reduce the size of the switch that is mainly coming from the flexibility of reconfiguration at runtime. This approach has not been pursued any further as it would exceed the scope of this work. For this work, where flexibility and speed are most important, this approach does not seem beneficial. However, in case a circuit is emulated only with very small error probabilities, and very few probability islands, this approach offers a very efficient way, both in terms of speed and area overhead.

3.2.8 Hardware Overhead

Parallel HW-based fault generation The area overhead of the proposed emulation system can be divided into a fixed and a variable part. The fixed one contains those parts of the design that are independent of the circuit-under-test, like simulation controller and interface specific components. The variable component instead grows with the size of the circuit-under-test, respectively with the number of input ports, output ports and

most important, the number of flip-flops. The original variant of the emulator requires 299 look-up-tables and 494 flip-flops for the modules with fixed size. This size mainly depends on the interface used. The variant based on the PCI express and Ethernet interface require more hardware resources, but offer more flexibility than the UMR bus based variant. The variable hardware cost are mainly determined by the random number generators and the injection-chain as it has been already shown in Table 3.1. Note that one probabilistic bit-error generator is required per injection point. Compared to this hardware overhead, the overhead due to the circuit instrumentation, i.e. the insertion of XOR gates, is negligible. Often the additional XOR gate can be even combined in a not fully used FPGA look-up-table. The overhead of flip-flops and look-up-tables depending on the number of probabilistic elements (PEs), i.e. injection points is shown in Figure 3.23. The overall resource utilization of the simulator for two benchmark circuits is shown in Table 3.2. One can see that the area overhead induced by the original, i.e.

Circuit	LUTs	FDs	Overhead LUTs	Overhead FFs
b14	12476	19055	1010 %	8561 %
s1196	1245	2061	808 %	10205 %

Table 3.2: The overall resource utilization of the simulator for exemplary benchmark circuits

parallel, fault generation clearly is very large. This becomes also clear, when comparing the overhead with other simulators [141]. A comparison with absolute numbers of FFs and LUTs is not possible as these simulator are synthesized for different architectures. Hwang et al. [78] report an overhead of 813-817%, dependent on the injection strategy, for benchmark circuit *s1196* in their implementation. The emulator presented by Lopez et al. [80] has an overhead of 142-400%, dependent on the injection strategy, for benchmark circuit *b14*. The simulator presented by Civera et al. [79] has an overhead of only 42.8%, for benchmark circuit *b14*. However, these simulators cannot be directly compared with the one presented in this work as they do not support the injection of errors based on probabilities. However the area overhead is a clear drawback. The reason for the large area overhead of the simulator presented in this work are clearly the random-number generators and the injection chain, both 24-bit wide in this example. An area overhead of 73 FFs and 26 LUTs is induced for each injection point. We have already seen that for longer emulation runs the length of the shift-registers has to be increased in order to generate sufficiently random numbers.

Host-based fault generation One potential solution to overcome this problem has already been presented. The generation of the pseudo-random numbers on the host PC and not on the FPGA would remove the generators and comparators from the emulator and reduce the injection scan chain width from 24 registers to one register per injection point. A graph comparing the area-overhead of flip-flops, of FPGA-based versus host-based generation is shown in Figure 3.19. One can see that the overhead dramatically decreases. However, as it has been shown already above, the generation of random num-

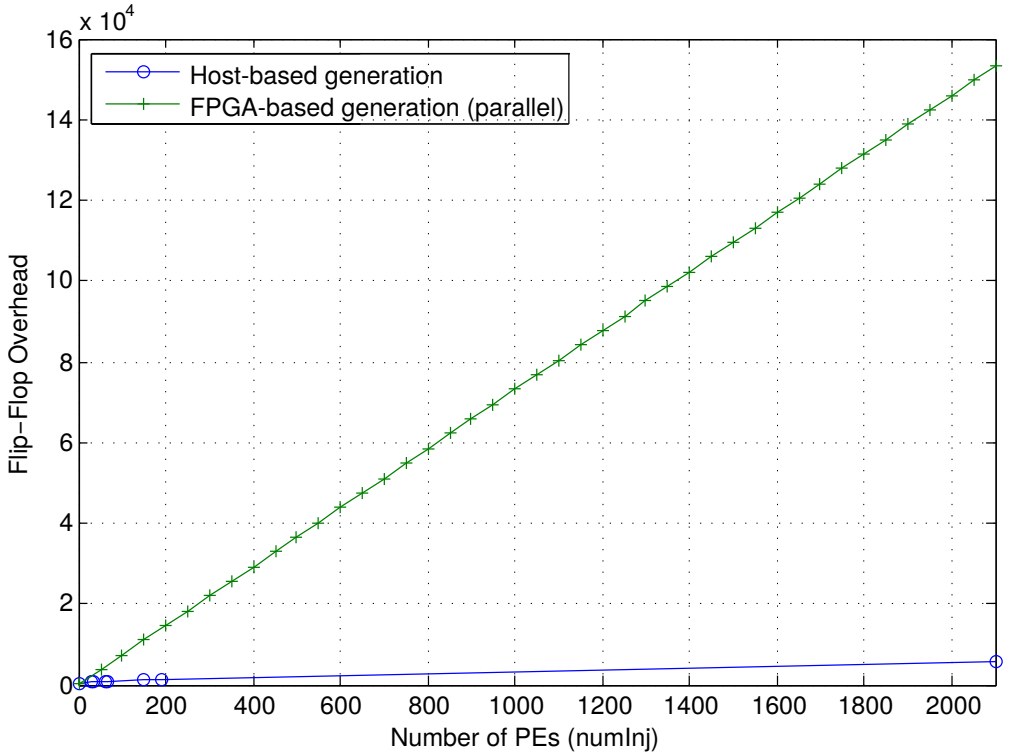


Figure 3.19: Area overhead when generating faults on the FPGA compared to when generating on the host PC

bers on the host PC and the transmission of fault tables imposes yet another bottleneck, which makes the emulator practically unusable for a complex circuit analysis. Hence, the only useful way to reduce the area overhead is to do it by improving the implementation of the bit-error generators as it has been shown above.

Serial HW-based fault generation As expected, the hardware overhead reduces a lot when using serial fault generators instead of parallel ones. The probabilistic bit-error generators, presented in Section 3.2.6, theoretically consume 15 FDs, 15 LUTs and 3 SRL16s on a Virtex-5 device. However, the real resource utilization differs due to optimization of the synthesis tools, as sometimes the LUTs can be combined. Furthermore, the proposed simulator consists of much more than just the error generators. Especially, counting the errors at the output of the circuit is a resource intensive task that is mainly accomplished by the usage of DSP slices. The absolute resource utilization for some exemplary circuits is shown in Table 3.3. One has to keep in mind that these numbers include the whole simulator, i.e. two instances of the circuit and all necessary control and communication logic. The actual overhead compared to the circuit under test itself is shown in Table 3.4. One can see that the overhead of flip-flops is much higher for small circuits, like benchmark circuit *s1196*, than for large circuits, like the Xilinx FFT IP. The reason for this is that the ratio of the fixed-size part of the simulator, like communication

Table 3.3: Total area requirement for the emulation of exemplary circuits (pre P&R)

Circuit	N_{PE}	FDs	LUTs	DSP48E
ISCAS'89 s1196	18	2433	1473	15
ITC'99 b14	216	8314	6975	55
ITC'99 b15	448	13093	10284	64
Xilinx xfft	2985	52656	48026	74

Table 3.4: Area overhead of complete simulator compared to circuit under test (pre P&R)

Circuit	FDs	LUTs	DSP48E
ISCAS'89 s1196	12065 %	975 %	-
ITC'99 b14	3679 %	520 %	-
ITC'99 b15	2922 %	870 %	-
Xilinx xfft	1707 %	1817 %	493 %

and control logic compared to the circuit itself is much higher. However, the overhead of required LUTs is not decreasing with the circuit size, as it is highly dependent on the circuit logic how LUTs can be combined and optimized by the synthesis tool. Hence, a general statement concerning the hardware overhead is difficult to make. When comparing the hardware overhead with the one presented in Table 3.2, it seems that the situation became even worse when using serial generators instead of parallel ones. However, one has to keep in mind that the length of the shift registers has increased a lot. A direct comparison is given in Table 3.5. A comparison of the hardware overhead of serial,

Approach	FDs	LUTs	SRLs
HW-based (serial, 64-bit LFSR)	15	15	3
HW-based (parallel, 24-bit LFSR)	26	49	0

Table 3.5: Comparison of the HW overhead for probabilistic bit-error generation on the FPGA

parallel and host-based fault generation is shown in Figure 3.20 and in Table 3.6. The presented simulator positions itself in the middle of host-generated bit-errors and purely parallel generated bit-errors concerning area overhead as well as performance. It can be seen that by making the compromise with the simulator performance the overhead of the required flip-flops could be reduced by 57 % and by 33 % for the required LUTs. This decrease of the overhead for flip-flops as well as look-up-tables in conjunction with the increase of the LFSR length to 64 bit make the emulator now practically applicable even for large circuits on mid-range FPGAs. The impact on the performance of the emulator for the three presented implementations will be discussed in the following Section.

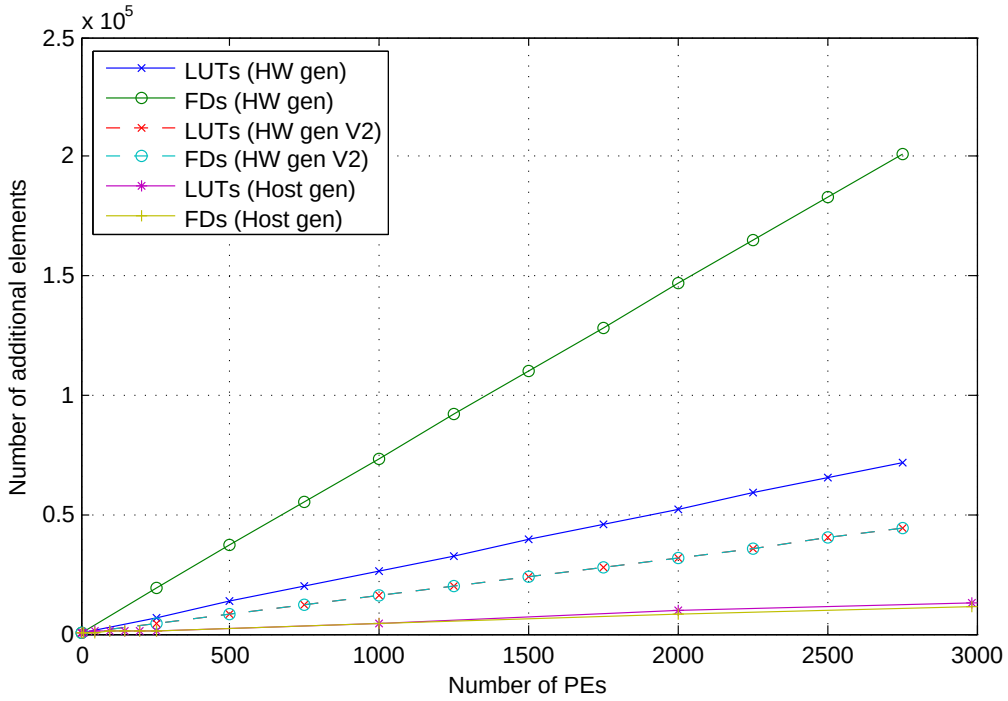


Figure 3.20: Comparison of the area overhead of the probabilistic bit-error generators, for serial, parallel and host-based generation.

3.2.9 Performance Analysis

Parallel HW-based fault generation In this section the performance of the presented implementations will be analyzed. First, the time it takes to configure the emulator will be analyzed. An analytic analysis is only possible for the UMR-bus based implementation. Only there the access times are more or less deterministic. For the PCIe-based, as well as the Ethernet based implementation, the performance is very similar. However, no measurements have been made, as the configuration is non-critical in the presented implementation, as will be shown later. Nevertheless, for the sake of completeness, the configuration time will be explained in the following:

$$t_{config} = t_{\text{scan}} + t_{\text{transmission}} + t_{\text{control}} \tag{3.2}$$

$$= \frac{1}{f_{clk}} \cdot \text{numInj} + t_{umr} \cdot \text{numInj} + 2 \cdot t_{umr} \tag{3.3}$$

where *numInj* corresponds to the number of transmitted error probabilities to the circuit, t_{umr} to the average time it takes to transmit one command between host and simulator and f_{clk} to the system clock frequency, in our case 70 MHz. t_{scan} describes the actual shifting of the probabilities through the scan chain. $t_{transmission}$ describes the time it takes to transmit the values from host PC to simulator. The value of t_{umr} has been determined to be around 0.0085 ms on a standard PC, which is actually very high when considering the net data-rate of the UMR-Bus to be around 55 MByte/s. The reason for

Table 3.6: Comparison of the overall simulator area overhead for ITC'99 b14 (post P&R)

Approach	FDs	LUTs
Host-based	5182 (2255 %)	2975 (165 %)
HW-based (serial)	8179 (3617 %)	8329 (641 %)
HW-based (parallel)	19055 (8561 %)	12476 (1010 %)

this is that single-transfers are used to transfer data on the UMR-Bus. Experiments using burst transfers show a huge speed-up. However, as already mentioned the configuration time poses no bottleneck. $t_{control}$ describes two extra commands necessary to control the injection. A plot showing the injection time with respect to *numInj* is shown in Figure

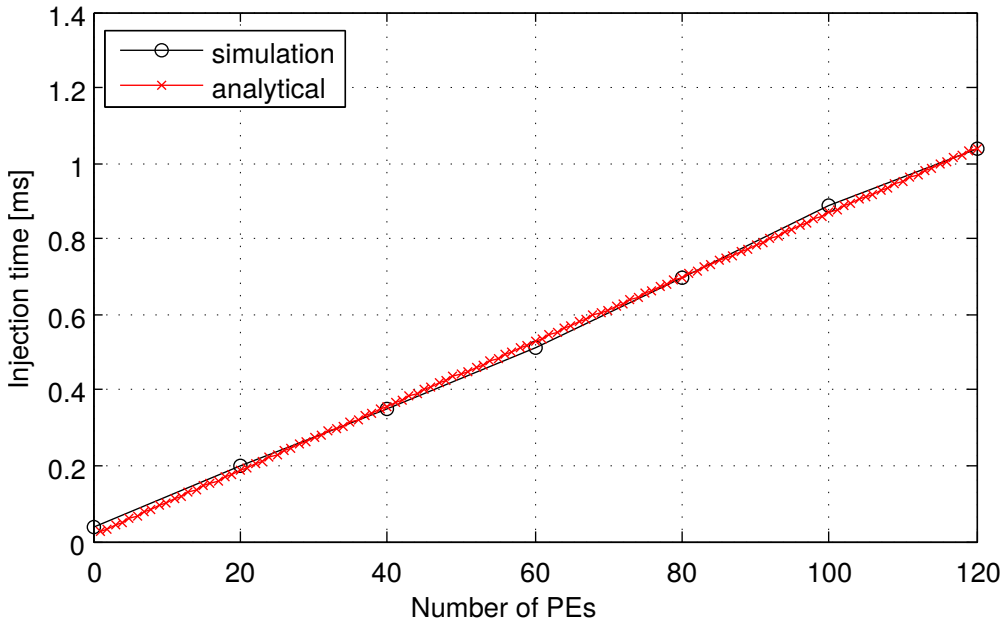


Figure 3.21: Configuration time required to transfer desired error probabilities to the emulator depending on the number of probabilistic elements (measured and analytical)

3.21. One can clearly see how the analytical estimation matches the measurements. The simulation time, i.e. the time it takes to run a fault injection experiment can be calculated as follows:

$$t_{sim} = t_{\text{sim\_duration}} + t_{\text{control}} + t_{\text{result\_transm}} \quad (3.4)$$

$$= \frac{1}{f_{clk}} \cdot \text{numSim} + 2 \cdot t_{umr} + \text{numOut} \cdot t_{umr} \quad (3.5)$$

where *numSim* corresponds to the number of simulated clock cycles and *numOut* to the number of circuit outputs. $t_{sim\_duration}$ describes the time the simulation runs. $t_{control}$,

two extra commands necessary to control the simulation, and t_{result_transm} the time it takes to transmit the number of measured errors back to the host PC. A plot showing the simulation time with respect to the number of simulation runs can be seen in Figure 3.22. Again, it can be seen that the estimation matches closely the measurements.

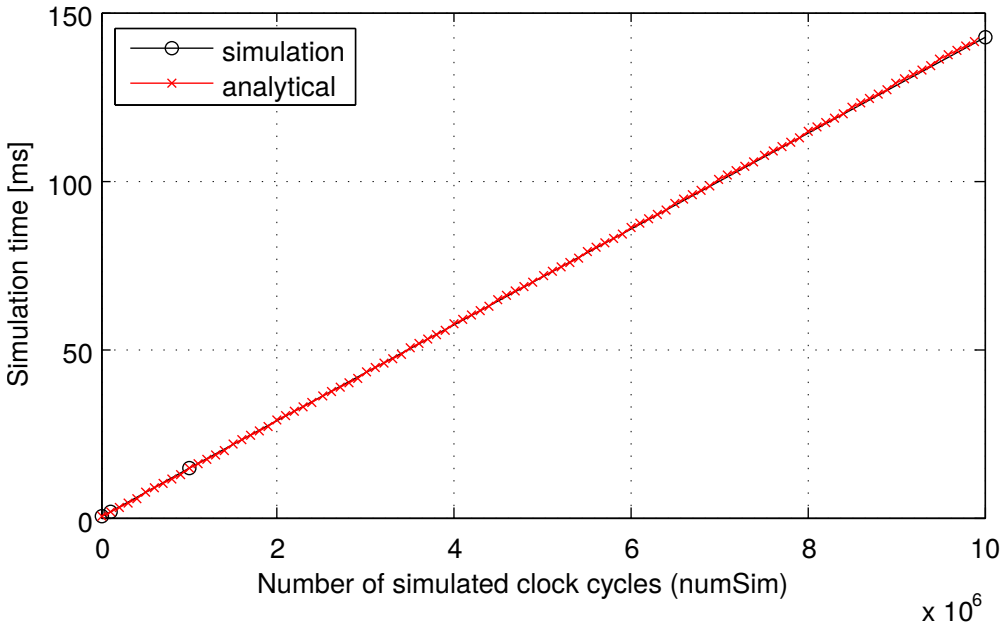


Figure 3.22: Simulation time required to run an emulation depending on the number of emulated clock cycles (measured and analytical)

The simulation run-time is determined by the number of emulated clock cycles. The communication interface poses no bottleneck, as the amount of data that has to be transferred between FPGA and host PC is small. This is only the case because the generation of faults and stimuli has been moved from the host PC to the FPGA in this implementation. This makes the emulator as fast as possible. The relocation of the fault generation onto the FPGA reduces the necessary amount of communication vastly, which is especially useful in our simulation approach, where we need thousands of simulation runs to minimize the variance of the output probabilities. The time it takes to perform one injection experiment, can be calculated as:

$$t_{\mathrm{tcase_{in}}} = t_{inj} + t_{sim}, \tag{3.6}$$

where t_{inj} corresponds to the injection time, and t_{sim} to the simulation time, introduced in Equations 3.3 and 3.5. By moving the generators from the FPGA back to the PC, the equation could be expressed very simplified as:

$$\begin{aligned} t_{\mathrm{tcase_{out}}} = \mathrm{numSim} \cdot (&\frac{1}{f_{clk}} \cdot (\mathrm{numInj} + 1) \\ &+ t_{umr} \cdot (\mathrm{numInj} + 4 + \mathrm{numOut})) \end{aligned} \tag{3.7}$$

However, as it will be shown later in this section, the generation of faults on the host-PC is limited not only by transferring the fault lists. This analytically calculated speed-up $\frac{t_{tcase_{in}}}{t_{tcase_{out}}}$ compared to the area overhead, introduced by generating the random numbers on the FPGA, is shown in Figure 3.23. One can see that the speed-up grows mainly with the number of emulated clock cycles. The speed-up dependent on the number of PEs is almost constant. By contrast, the area overhead grows proportional with the number of PEs. As the required number of emulated clock cycles usually grows with the size of the circuit, in order to have a sufficiently large sample space, one can see that likely a FPGA-based generation of faults is preferable compared to a host-based generation.

Host-based fault generation As we have just seen, the host-based generation of faults, solves the issue of the required hardware overhead, but introduces a huge performance bottleneck. The simulation time when generating faults on the host PC for the implementation presented above in Chapter 3.2.5, is shown in Figure 3.24 for some exemplary circuit sizes. The main difference to hardware-generated faults is that the simulation time now depends much stronger on the number of emulated cycles. While in the hardware-based case the simulation time more or less equals to the number of emulated clock cycles multiplied with the clock period, the limiting factor now became the transfer of the fault lists. Additionally, as we have also seen, the generation of random numbers limits the simulation speed as well. In Figure 3.25 the time it takes to generate random numbers on a host PC depending on the circuit size and the number of emulated clock cycles is shown. The measurements have been performed on a regular PC with MATLAB. Clearly a parallelization would decrease the required time accordingly. Nevertheless, even with a division by four or even by 8, the required time would be significant. Note, that in order to analyze a circuit for Approximate Computing thousands of different fault injections have to be performed, each one requiring a new set of fault tables. Furthermore, the amount of memory required on the host PC is non trivial. For instance a fault table for 4000 injection points covering 1 million cycles, already requires about 500 Mb of memory. Unfortunately, these limitations make the host-based generation of faults practically unusable, at least when performing probability-aware experiments.

Serial HW-based fault generation As it could be seen in the previous section, the hardware overhead of the serial HW-based fault generators is in between the one of the parallel implementation and the host-based one. The performance change of the serial implementation ideally behaves similar. Due to the serial generation of faults, the emulator is running in two clock domains. The maximum speed is hence limited by the maximum speed of the faster clock, the one of the serial generators. In other words one can say that the implementation based on serial fault generation is running 32 times slower than the parallel implementation. However, one has to keep in mind that the length of the LFSR has increased to 64 bit, while at the same time dramatically decreasing the required hardware. Analytically, the execution time of one raw simulation run can be determined as:

$$t_{sim} = t_{clk\_x32} \cdot N_{\text{cycles}} \cdot 32, \tag{3.8}$$

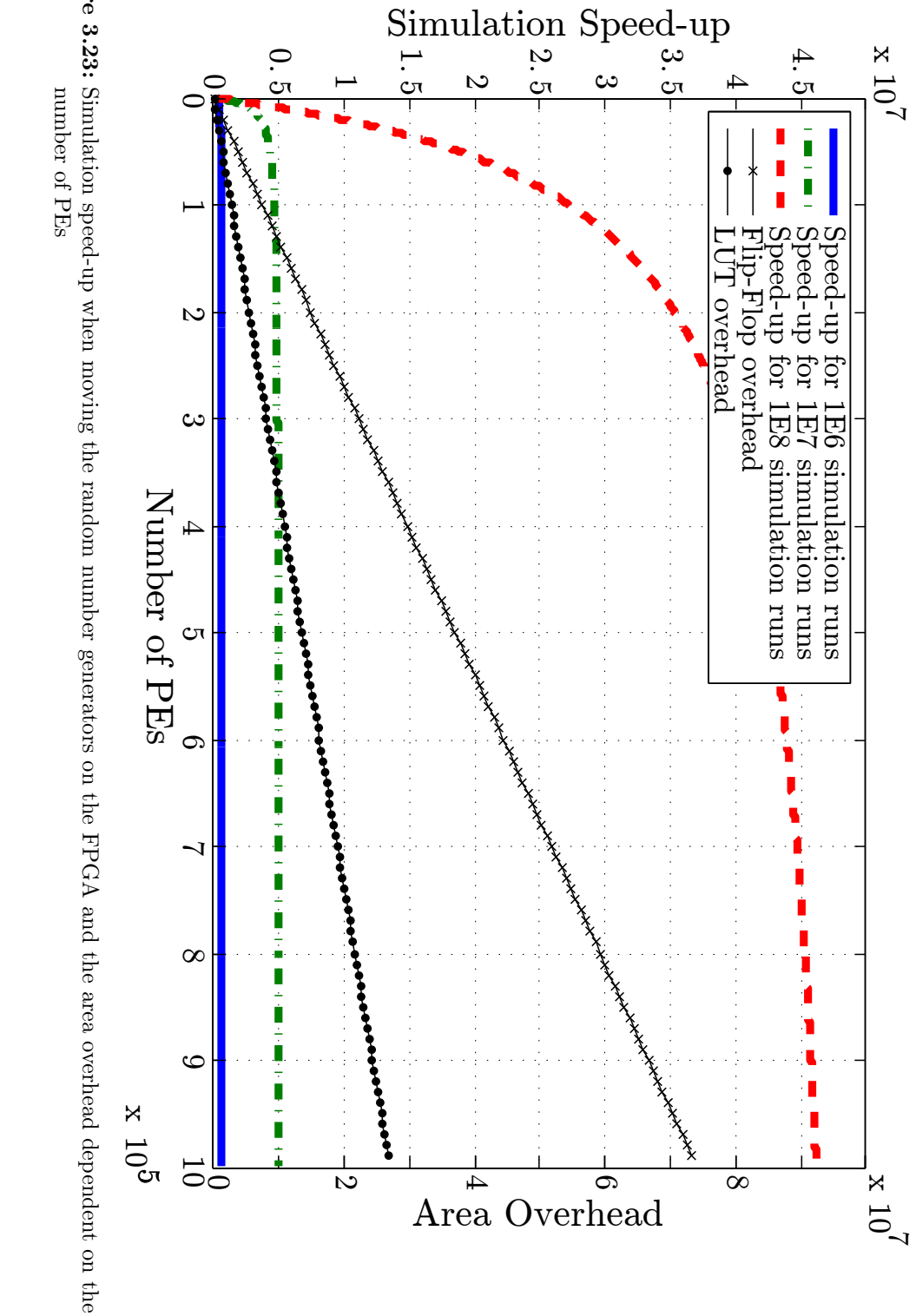


Figure 3.23: Simulation speed-up when moving the random number generators on the FPGA and the area overhead dependent on the number of PEs

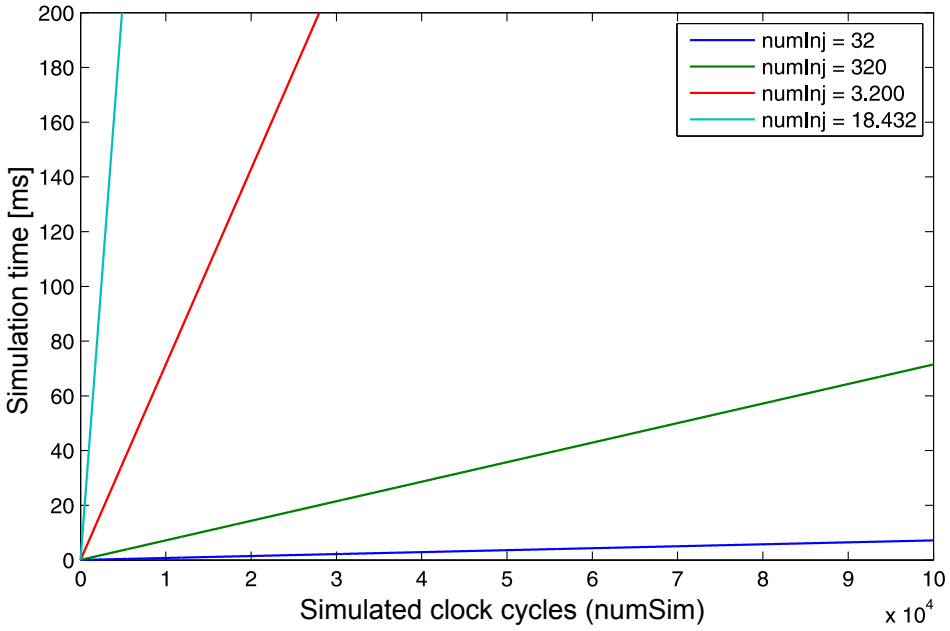


Figure 3.24: Simulation time when generating faults on the host PC for some exemplary circuit sizes [133]

where t_{clk_x32} the clock period of the probabilistic bit-error generators and N_{cycles} the number of clock cycles to be simulated. The required time for a complete simulation campaign, i.e. storing seeds and probabilities, running the simulation and reading back the results, can be determined as:

$$t_{\text{campaign}} = t_{\text{comm}} \cdot N_{PE} \cdot 6 + t_{\text{sim}} + t_{\text{comm}} \cdot N_{PE} \cdot 2, \tag{3.9}$$

where t_{comm} is the time it takes to transfer one 32-bit word between the host PC and the FPGA and N_{PE} the amount of probabilistic elements, i.e. the registers where errors might be injected, in the circuit. A final comparison of the performance of the three presented variants is shown in Figure 3.26. The execution time has been measured emulating 10^7 clock cycles. The clock frequency of the circuit-under-test is 70 MHz. One can see that, when generating faults on the host-PC, the execution time is increasing with the error probabilities applied to the flip-flops. For very small error probabilities, the execution time settles. The speed in this case is limited by the random number generators on the PC. Note the logarithmic scale and that the benchmark circuits are very small. For larger circuits, like the Xilinx FFT circuit, the settle point is already at $1.5 * 10^3$ s. The random number generation for these measurements has been parallelized on 16 CPU cores. As expected, the fasted fault generation can be achieved by using parallel generators running on the FPGA. The execution time is independent of the error probability and, more importantly, of the number injection points, as no communication between host-PC and FPGA is required and each injection point has its own generator. The serial implementation is 32 times slower than the parallel one. For very small circuits

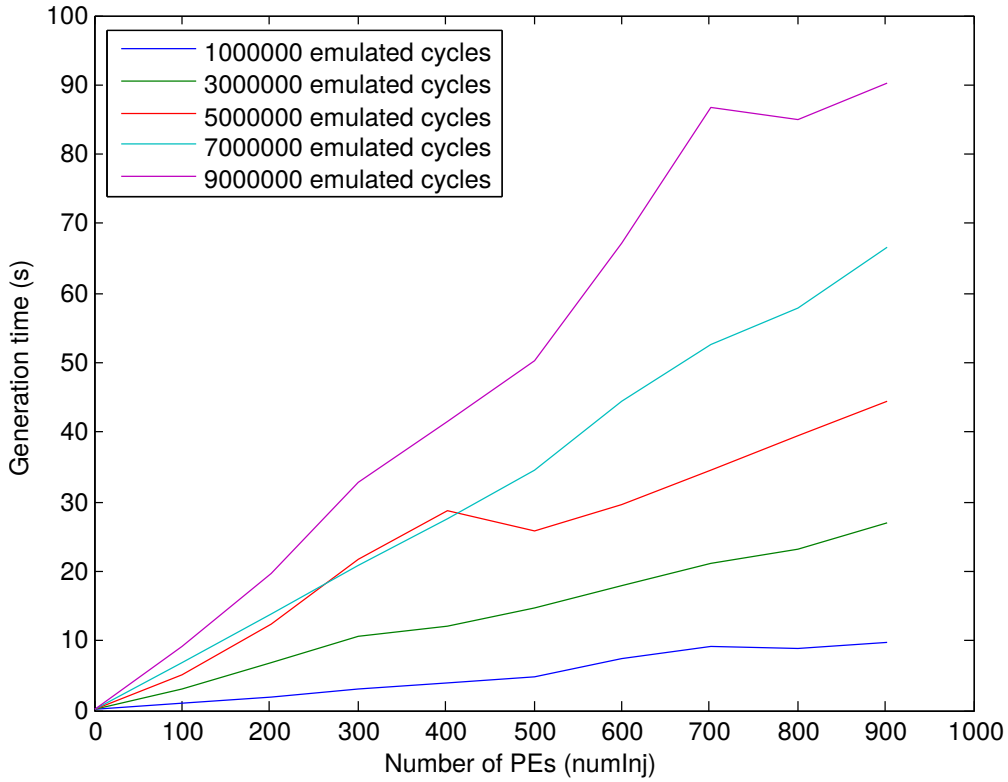


Figure 3.25: Time required to generate random numbers on a regular desktop PC depending on the circuit size

and small error probabilities it is slower than the software based version, but for larger circuits it is still much faster despite the serial bit-error generation. The execution time of simulating 10^7 clock cycles of benchmark circuit "b14" reduces to 26 % when generating bit-errors in a serial and to 0.8 % when generating in a parallel manner if errors appear with a probability of 10^{-5}. If the error probability is set to 10^{-2} the execution time reduces to 1 %, respectively 0.03 %.

3.3 Evaluation

In order to verify the general functionality of the emulator, several circuits have been emulated and tested. Note, that the emulations presented here did not perform any further analysis concerning their applicability of Approximate Computing mechanisms. In the previous section, evaluating the performance and the overhead of the different emulator implementations, some benchmark circuits have already been introduced. Especially the sequential benchmark circuits ISCAS'89 [118] and ITC'99 [119] have been excessively used to verify the functionality of the emulator. However, their use is limited. In case of the ISCAS'89 circuits, the functionality of the circuits is usually unknown and they are provided only in a netlist format. This makes it impossible to provide accurate test

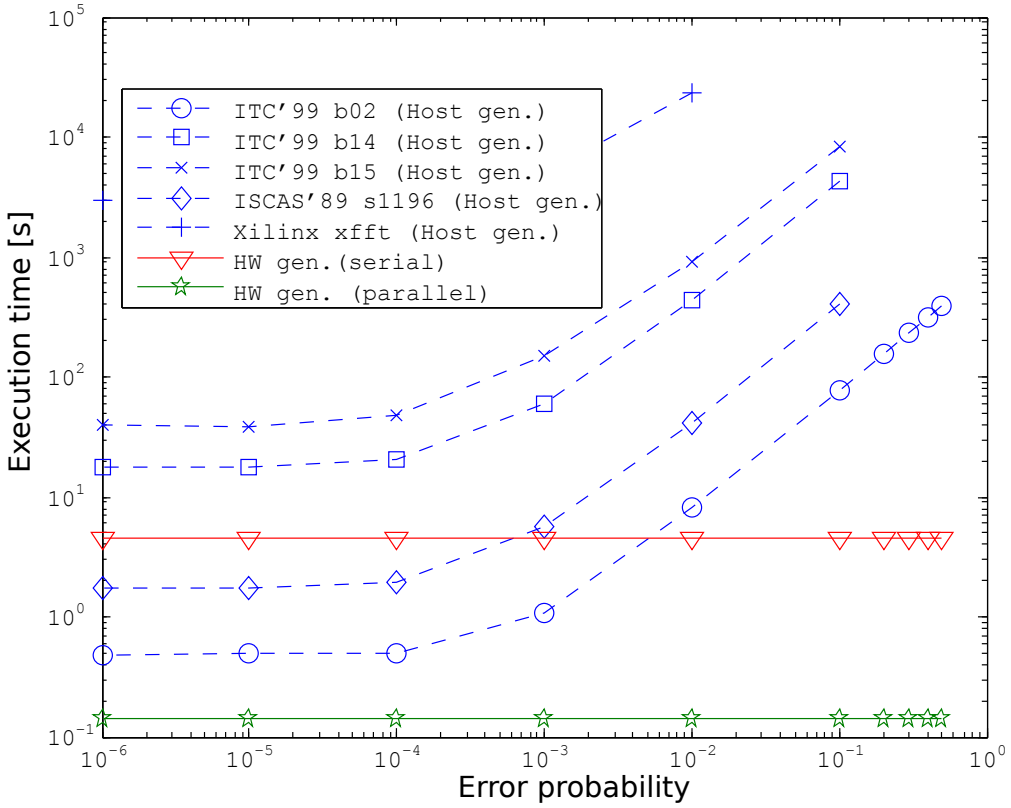


Figure 3.26: Performance comparison of host-based, parallel and serial fault generation

patterns. Even though the ITC'99 benchmarks are available as functional source code, the same applies, as the detailed functionality is unknown. Hence, the general functionality and applicability of the emulator has been tested with circuits, whose functionality is known and where appropriate test patterns could be applied. These will be presented here for the sake of completeness. The application of the fault emulation system with respect to approximation techniques will be demonstrated using various examples in the next chapters.

The first realistic and complex application that has been analyzed using the FPGA-based fault emulation system has been a hardware-based DAB radio receiver chain [120, 136]. The goal was to find out if and how the emulator performs at injecting faults with an error probability and measure the impact on the error probability at the output pins. Prerecorded baseband samples have been used to provide realistic test-pattern to the emulator. Using a Matlab/Simulink model of the hardware implementation [120], at first its general susceptibility to faults has been analyzed. DAB is an OFDM-based system. It pointed out that the FFT could be a potential candidate for approximation. In case of approximations within the FFT, faults have a manageable impact on the behavior of the receiver. Faults within other blocks like frequency and time synchronization have a much higher impact on the resulting output bit error rate. The analyzed decoder chain

consists of the blocks: “frequency correction”, “AGC”, “time synchronization”, “FFT” and the “demodulator”. Two experiments have been performed. The radio data, real and imaginary part, is transferred in the “8.7” fixed-point format between these blocks. In the first experiment the error probability of all bits has been equally increased. In the second experiment the error rate of the two most significant bits (bit 5 and 6) has been fixed to zero and only those of the less significant bits has been increased. Less surprising the output error rate reduces when protecting more significant bits. The location where in the chain the faults are injected does not seem to make a large difference. An exception could be observed when injecting faults before the demodulator, i.e. emulating approximations in the FFT. A fault injection after the FFT has much less impact on the resulting bit error rate than at any position earlier in the chain. Note, that a fault injection after a certain block means that one is analyzing the effects of faults within that block, as these result in faulty values at the output of the block. Due to this behavior of the FFT the exemplary FPGA-based emulations have been carried out by analyzing the FFT block. Additionally, the time synchronization block has been analyzed in order to see how approximations in blocks that do not seem to be directly suitable for Approximate Computing could be performed. The hardware implementation of the time synchronization consists of 87 flip-flops and the Xilinx FFT, configured for the DAB system, consists of 326 flip-flops. Hence, 87, respectively 326 probabilistic faults have to be injected per clock cycles. The time synchronization block outputs 8 bits for the real part of the samples and 8 bits for the imaginary part. Additionally, 14 bits are used as control signals. Note that for these simply tests no sophisticated algorithms have been applied when performing the emulation. The algorithms will be presented in the next two chapters. The fault emulation campaigns used here are just to verify the functionality and usefulness of FPGA-based probability-aware fault emulation. In case of the time synchronization block the following arbitrary constraint has been defined for the output pins. The, in total 16, data bits must not exceed an error probability of 10^{-4}. The control pins must not exceed any error. In the following it has been tried to find a distribution of error probabilities (as high as possible) at the flip-flops while still meeting this constraint on the outputs. Without going further into detail on the algorithm, one resulting distribution is show in Figure 3.27. One can see that some flip-flops tolerate a larger error probability than others. Some flip-flops do not tolerate any error probability at all. Algorithms on how to generate this information will be presented in the next chapters. Similar results have been created by emulating the Xilinx FFT block. DAB is a (D)QPSK based system. Hence, the information is actually encoded in the sign of the real and imaginary data. In this case it is therefore important to tolerate faults only in those parts of the FFT that do not influence the sign of the real and the imaginary output value. Figure 3.28 is showing the average possible error probability of all flip-flops in the circuit for a varying tolerated error probability of the sign bit of the real part output.

The second example application that has been analyzed is a purely hardware-based h.264 decoder. The purpose of the h.264 decoder is to decode an encoded video stream and give out the decoded video on a display. This allows to visually score the video quality. Later in this work it will be shown how this process can be automated. Three major blocks have been analyzed using the presented fault emulation system, namely the

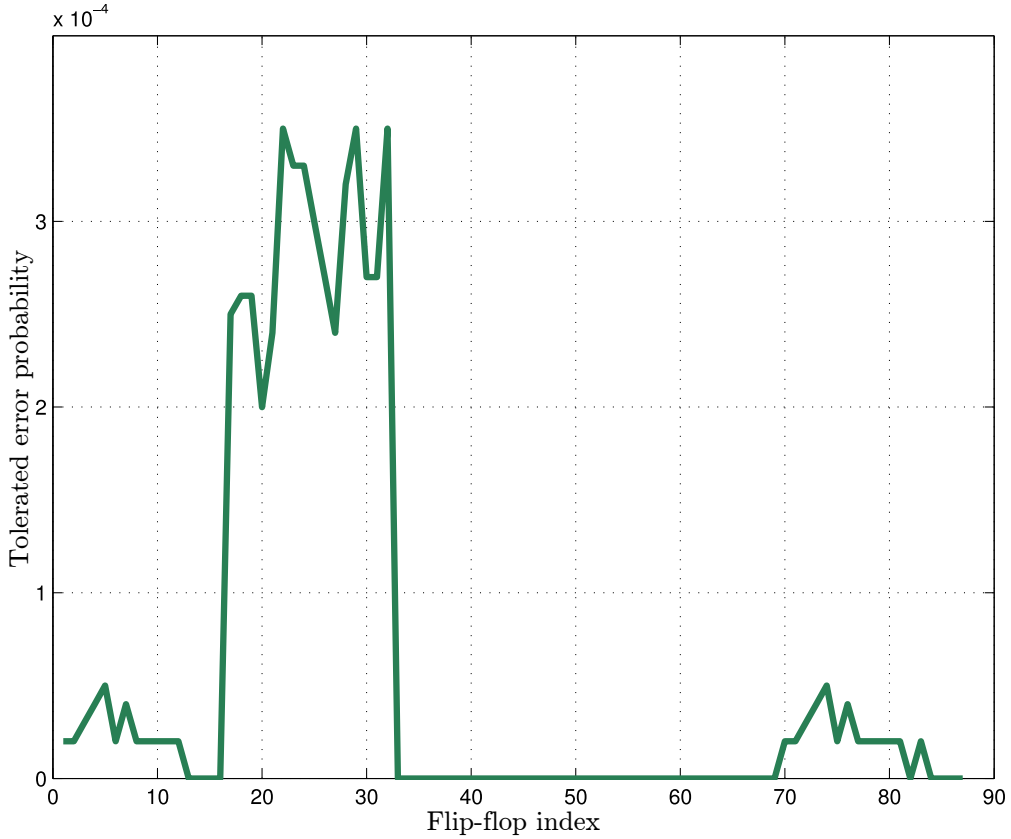


Figure 3.27: Possible error distribution at the flip-flops of the time synchronization block of a DAB receiver, still meeting an arbitrary set of constraints on the output pins [136]

intraprediction decoding, the interprediction decoding and the de-blocking filter. A naïve injection of faults into all flip-flops with equal error probability revealed the differences of the impact on the visual quality of the three blocks, as shown in Figure 3.29. One can see that depending on where faults are injected, hence tolerated, the impact on the visual quality differs dramatically. Additionally, what cannot be seen in the figures is that reproducible after some time, sometimes sooner, sometimes later, the decoding completely stops. Hence, not only the visual quality is affected, but also the functionality. Clearly, this is something which cannot be tolerated even with Approximate Computing. Some flips-flops do not only influence the quality but also the functionality. This functional difference can simplified be described as control and data-path. When assigning a large error probability one by one to each flip-flop one can determine the influence of each flip-flop to the outputs of a circuit. When doing this a graph showing the connection for each register and each output pin can be generated. For the intraprediction block, this graph is shown in Figure 3.30. One can see that most of the flip-flops (probabilistic elements) do only influence a certain index range of output pins. Only a few have significant influence on the first part of the output pins. Actually, the decoded video is given out at exactly this last block of output pins. The first half is used to interface

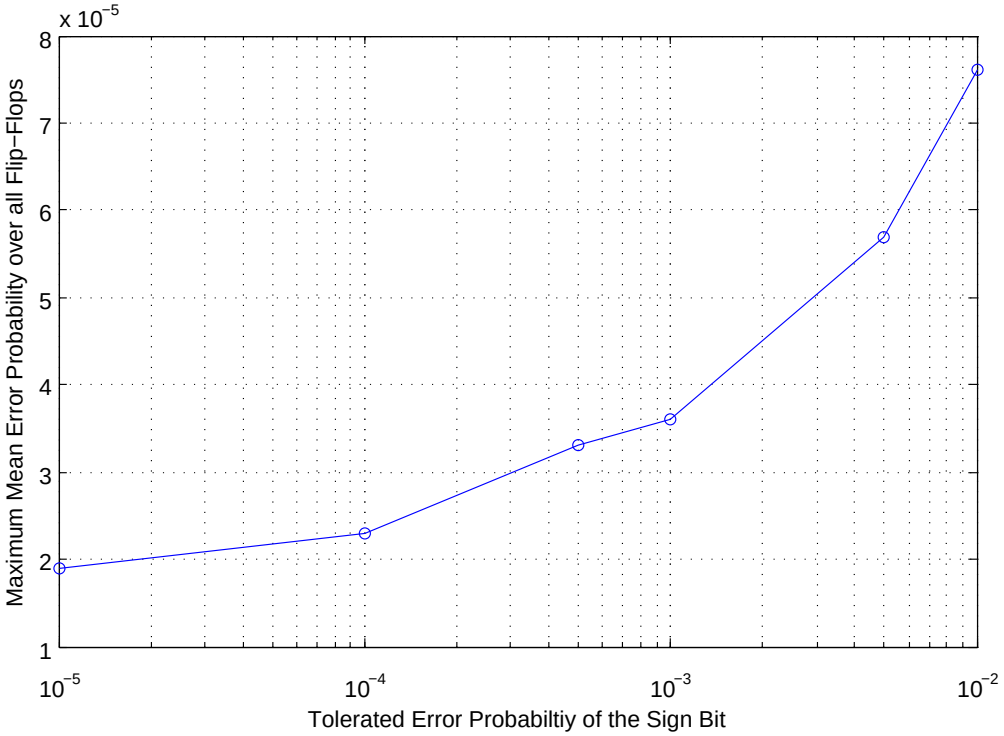


Figure 3.28: Possible maximum mean error-probability over all flip-flops of the FFT block of a DAB receiver, still meeting an arbitrary set of constraints on the output pins [136]

external memory. Hence, the approximations made within the circuit should likely affect only that last block of pins, so that only the video quality is affected. This fundamental relation will be elaborated in detail in the next chapter.

The fault injection experiments performed on the DAB receiver chain as well as the h.264 decoder helped to gain an initial idea on what steps are necessary in order to approximate a circuit. Furthermore, more or less obvious problems and limitations could be revealed and confirmed. On the one hand these initial tests showed that probability-aware fault emulations can be used to analyze the behavior of circuits to approximations within the circuit. Varying error rates at the circuit elements affect the applications differently. Additionally, it could be shown that it is indeed necessary to assign different error probabilities to different circuit elements, as not all elements can be approximated in the same way, as their influence on the application can vary dramatically. It could be shown that it is fundamental to transfer the information about the quality metric down from higher levels to the circuit level. For instance in case of the FFT benchmark, this means that the sign of a number is more important than the amplitude. Or in case of the h.264, it is very important to know that the last block of output bits actually contains the image data. It could be seen that it is fundamental for the approximation of a circuit that before actually starting with the approximation those flip-flops have to

(a) Interprediction decoding

(b) Intraprediction decoding

(c) Deblocking filter

Figure 3.29: Impact on fault injection into different functional parts of the h.264 video decoder [135]

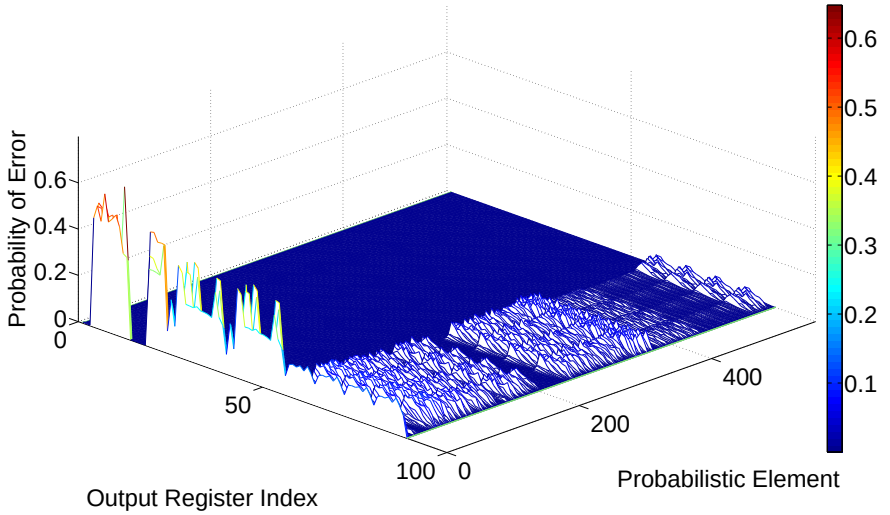


Figure 3.30: Relation of errors ($p_e = 0.5$) injected at registers and resulting error rates at output pins of the intraprediction block of a h.264 video decoder [135]

be identified that are generally qualified for an approximations. Flip-flops being part of the control flow of an application, likely cannot be approximated as it would result in a unpredictable behavior of the application. Furthermore, one could see that it is required to emulate a circuit for a sufficiently long time in order to gain a trustworthy result. Some faults, especially if their probability of appearance is chosen small, take a long time to be visible at the output of a circuit. But even these very unlikely errors can result in a highly unwanted behavior of the application. Hence, it is necessary to run the emulation long enough so that even these unlikely faults have appeared and that their impacts are visible. And finally, these initial experiments reveal that the approximation of integrated circuits at register-transfer level can be seen as an optimization problem. Assuming that the information about minimal quality has been transferred down to the output pins of circuit, a maximum tolerable error probability of each output pin is known. Hence, the approximations made within the circuit must not result in error rates larger than these. At the same time as much approximations as possible are desired in the circuit. The more approximations tolerated, likely the more power can be saved, as more elements can be pruned or as the supply voltage could be reduced more. Hence, the optimization problem is to find a distribution of error probabilities assigned to the flip-flops of the circuit as large as possible, while at the same time meeting the constraint on the error probability at the output pins. In the remainder of this work methods and algorithms will be presented that try to solve exactly the problems just described.

4 Automated Functional Approximation of Sequential Circuits

In the previous chapter a tool has been presented that allows to analyze the behavior of integrated circuits to faults within the circuit. An FPGA-based fault emulation system has been proposed for this purpose. Different alternatives have been presented of implementing such an emulator, each having certain benefits and drawbacks. For the remainder of this work the implementation has been chosen that offers a fair trade-off between area overhead and performance, using serial fault generation. In this section a methodology will be presented that allows to use this tool in order to efficiently apply "Approximate Computing" to integrated circuits. When testing the emulator with real world applications, several fundamental basics in order put Approximate Computing into practice have already been discovered. In order to quantify the effects of approximations, i.e. faults, within a circuit one has to understand how faults at hardware level affect the overall application that uses a particular hardware block. It is therefore necessary to transfer the information about "good enough" quality from the user level, down to the circuit level. "Application-reasoned Approximation" will be covered in Section 4.3. However, before starting with the actual approximation it has to be analyzed which circuit elements qualify themselves for getting approximated. As we have already seen in the previous chapter, faults in some flip-flops do not only affect the quality of the result but also the functionality. Hence, a careful pre-selection has to be performed in advance. This very fundamental prerequisite will be covered in detail in Section 4.1. In order to make the emulations meaningful, so that one can trust the result, not only realistic test pattern have to be provided, but it is also required that the emulation is running for a sufficiently long time. This is in particular necessary as faults are injected based on a probability in the presented emulator. If the probability is small it can take very long until a single fault appears. As we will see in Section 4.2 several metrics can be used to determine if a measured error probability at the output of a circuit is statistically stable. If these fundamental problems are solved, one can finally start with the actual approximation. Approximation means that we want to find out which circuit elements, i.e. flip-flops, can experience faults without violating a constraint on the output quality. Additionally to finding out which element may experience faults one is also interested in the rate at which faults may appear. In general it is desired to tolerate as much faults as possible when approximating a circuit, as that means eventually that more power can be saved. Hence the initial approximation can be seen as an optimization problem. In Section 4.4 and 4.5 a methodology will be presented to solve this problem with reasonable complexity. An overview of the methodology is shown in Figure 4.1. Once the problem is solved, i.e. if it is known which flip-flops may experience which error probability, one

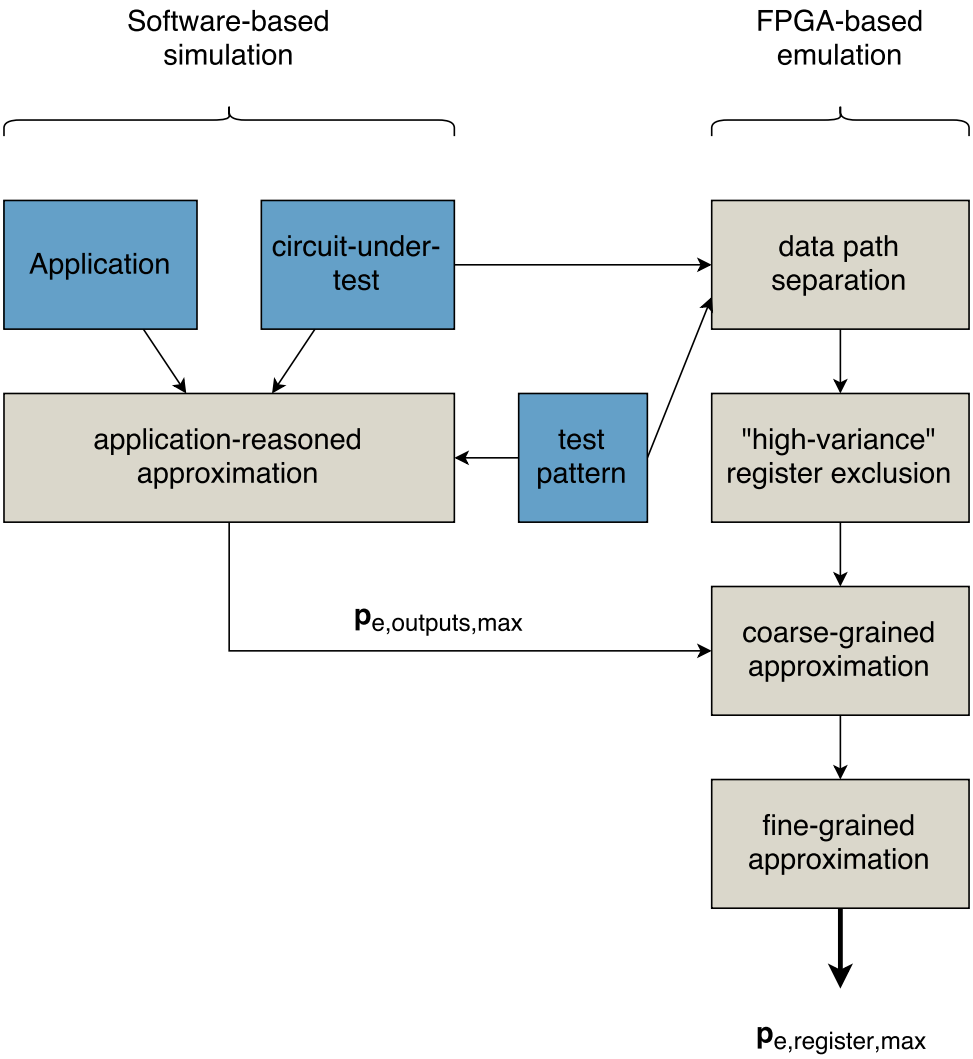


Figure 4.1: An overview of the approximation tool-flow presented in this work

can directly approximate the circuit using the "circuit pruning" technique, a functional approximation. The evaluation of exemplary circuits approximated with the mentioned methodology using circuit pruning will be presented in Section 4.6. The application of voltage over-scaling as the approximation technique requires more analysis steps. These steps will be presented and evaluated in Chapter 5.

4.1 Datapath Separation

The first important prerequisite before starting the approximation in the presented methodology is the identification of elements that are qualified for approximation. It

has already been explained in the previous chapter how imprecisions in some elements do only affect the quality of a circuit result while others also affect the control flow, the function of a circuit. Hence these faults can render the operation completely useless. This does not only result in very high error-rates at the output pins, which is usually unwanted even in Approximate Computing. What is even worse for Approximate Computing is that the behavior is usually not reproducible. The measured error-rates differ from run to run, even if the same set of test pattern is used and the same error probabilities are assigned to the flip-flops. The reason for this is that the propagation of faults in these registers is largely depending of the state and the transition of the state of the circuit. Clearly, the registers that behave like this are part of the control path of a circuit. In most cases they are part of the state machines controlling the procedure of a circuit. This is why in the following this initial approximation prerequisite will be referred to as a separation of data and control path. In most cases approximations can only be tolerated in those elements of a circuit that do only affect the data path. Approximations, hence faults, in the control path of the circuit usually result in very high error rates as the control flow is disturbed, which is why they are usually not qualified for an approximation. Additionally, as the propagation of faults in the control path of a circuit depends in the actual state of the circuit it sometimes takes more time to observe the faults and sometimes less. This would make it very difficult during the analysis of the circuit to generate meaningful results and have a reproducible fault behavior. Figure 4.2 visualizes this problem. In this example faults have been injected into a hardware-based "spacewire"

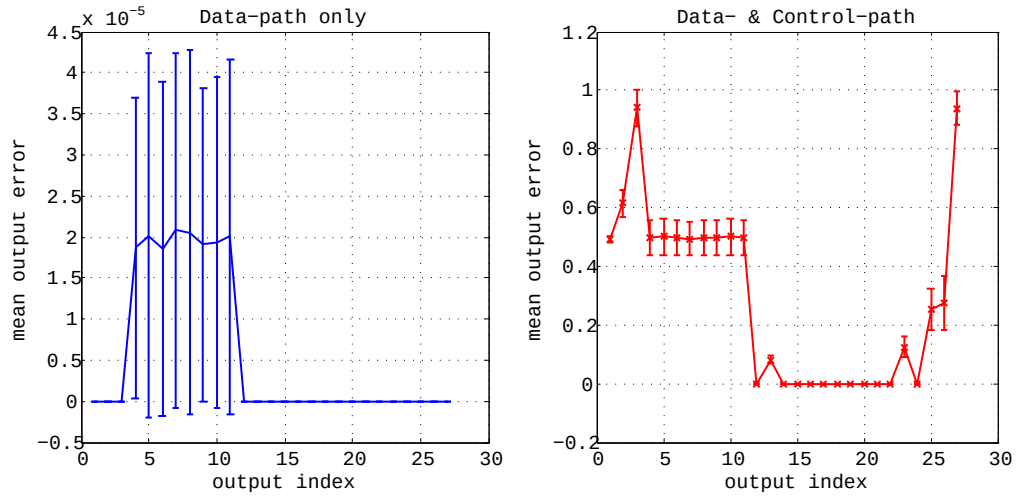


Figure 4.2: Output error probability and variance of a "spacewire" implementation when injecting equally errors with a probability 0.0001 in both data- and control path [143]

implementation. Spacewire is a communications standard to interchange data in space applications [121]. Without having to know the details of the application, it is clear that this application consists obviously of a control path, that is for instance responsible for framing of data. The data path instead likely consists only of buffers, temporary storing

the data received and to be transmitted. In the experiment, errors have been injected equally into each flip-flop with an error probability of 0.0001 [137]. The emulation length has been set to $1,000,000$ clock cycles. This trial has been repeated 50 times. The figures are showing the mean of the measured error probability at each output pin and their variance. The right figure is showing the measurements that have been made when injecting faults into each register, data and control path. The left figure is showing the results when injecting the faults only into the data path. First of all, one can see that the error probabilities are much higher when injecting faults also into the control path. Note, that the scale of the y-axis differs. Furthermore, one can see that when injecting fault into the data path only, faults are only visible at the output pins with index 4-11. The error probability at the other pins is 0.0. Note that the error probability in the right figure for pins 12-24 is not zero but $\sim 10^{-5}$. The 8 output pins that experience an error probability are the actual data pins, outputting the received payload. Furthermore, one can see that the variance of the determined error rate is much less, by a factor about $2 * 10^4$. It looks like allowing approximations only in the data path of a circuit is the key to create a trustworthy and direct relationship between the degree of approximation and the effects on the outputs. When moving the focus from Approximate Computing to the classical reliability analysis one can infer that the control path of a circuit requires special protection, in order to be robust to soft errors due to radiation. The difficulty however is to identify which parts of a circuit correspond to data and which to the control path. If the circuit is small and if the functionality is clear the identification could be done manually. However, for most applications this identification has to be automated. Two methods have been developed and evaluated that allow an automated identification of elements corresponding to data, respectively to the control path of a circuit.

For both methods it is required that the circuit has input and output pins that handle data and pins that are controlling the circuit. A generic diagram showing the separation of a circuit into control and data path is shown in Figure 4.3. In order to identify the members of the data, respectively the control path, the corresponding input and output pins are identified. Simplified one can define that those pins where faults can be tolerated correspond to the data path. These pins are usually containing data. Even for Approximate Computing it is unlikely that faults can be tolerated in control pins. For example, the h.264 video decoder that has already been introduced in the previous chapter, outputs the decoded video data into an external memory. The interface therefore consists of data pins containing the video data and several control pins like address pins and "read/write enable" pins used for controlling the data output. In this example it is clear that the video output pins would belong to the data path and control pins to the control path. Clearly, this approach is only working for circuits that not only output data, but also have a control interface. For those circuits where this separation is not very clear, i.e. where the control path cannot be identified, another step is mandatory prior approximation that will be presented in Section 4.1.3.

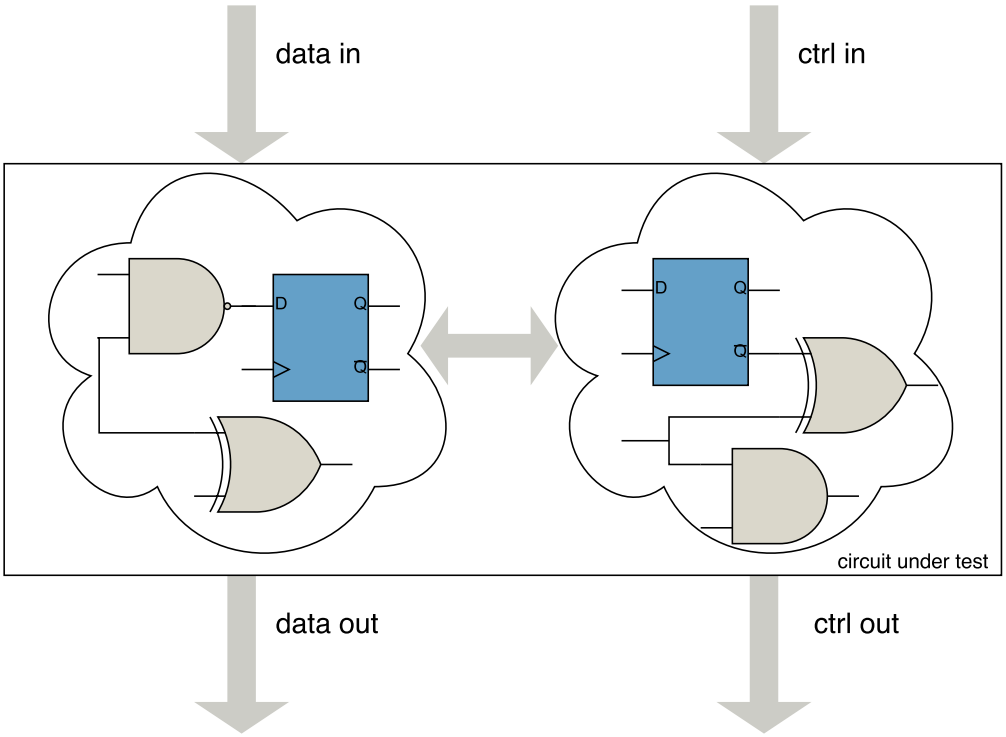


Figure 4.3: Visualization of control and data flow of a generic circuit

4.1.1 Netlist-based Separation Approach

The first approach presented is based on analyzing the netlist of a circuit. Once the input and output pins of the circuit are categorized the idea is to follow the path of these pins through the circuit. By doing so, one can identify which elements have a connection to which output pins. This can be done by building an abstract syntax tree of the circuit, as it has been already done for the circuit instrumentation. Based on that tree one can then identify all nodes that do influence only data outputs and those that do also influence control outputs. This approach is visualized in Figure 4.4. One can already see that this separation technique is very strict. One single connection to a control output is sufficient to be categorized as control path element. It could happen however, that the influence of such an element to the control flow of the circuit is minimal even though it is technically part of it. The separation based on the second technique presented in the next section will therefore be less strict. However, this method based on the netlist clearly gives the most precise and sharp separation of control and data path. The netlist-based separation can also be performed without having access to a netlist parser. By exploiting a simple trick the synthesis and optimization capabilities of a regular synthesis tool can be used to separate the control from the data path. By leaving the input and output pins of the data path unconnected, the synthesis tool is usually removing all related components due to the circuit optimization. The resulting netlist is then comparing only control path related elements. By comparing the optimized netlist with the unoptimized one, the data path

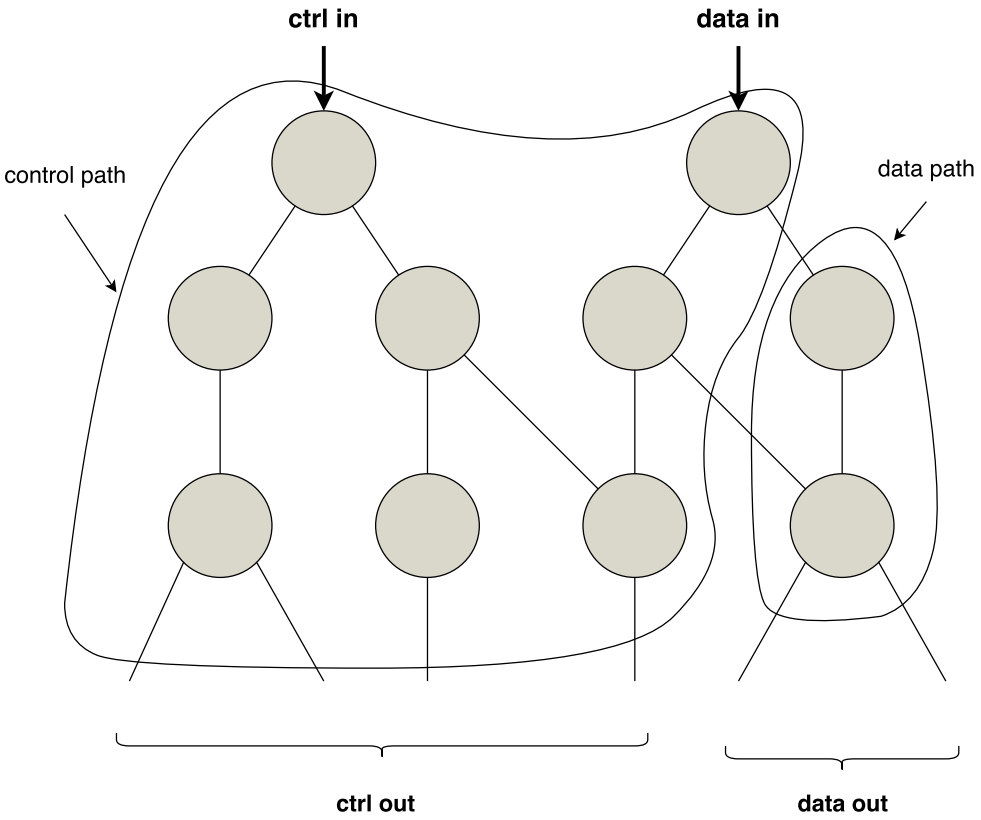


Figure 4.4: Data and control path separation based on analyzing the netlist of a circuit

can be extracted. Clearly, this method is not very reliable as no details are known about the optimization mechanisms of commercial synthesis tools. However, it pointed out that this in most cases it is working well. Independent on how the netlist gets separated, this method has a superior benefit over the one presented in the next section. The netlist-based approach will by design not produce any false-positives. False-positive in this case means that registers are detected to be part of the data-path, even though they are control-path related registers. A false-positive could mean that it will be approximated, hence faults will be tolerated, even though the faults can be propagated to the control structures of the circuit. However, it is also possible that the influence of such a register on the control flow is only minimal and hence could be tolerated. The potential savings of approximating this register will then be lost. However, if an element has any connection to a control related output pin it will not be detected as control path element with the netlist-based method. It is possible that registers get classified as data path, even though they are technically part of the control path. This is only possible if that register has no further connection to the control path, which in turn means that faults in general can be tolerated. Similarly, if the control path is data dependent, as depicted in Figure 4.4, any register that has at least one connection to the control path will be classified as control

path register.

The netlist-based approach hence is a very conservative separation, as every element that has at least one connection to a control path related output pin will be classified as a control path element, even though the influence is very small or not even possible due to masking effects. The netlist-based approach therefore serves as a baseline for the next emulation-based approach which is less strict and tries to allow approximations even in the control path if their effects are negligible.

4.1.2 Emulative Separation Approach

The emulative approach to detect the data and the control path related elements in a circuit uses the FPGA-based emulator itself. The initial step is the same as for the netlist-based approach. The output pins get classified into those where faults can be tolerated and those where no faults can be tolerated. Note, that at this early stage of approximation it is not required to distinguish in the rate of errors that can be tolerated. At this stage of the approximation it is only interesting to know which elements can in general tolerate faults.

In the next step, the error probability of each register is consecutively set to 0.5, but only one at a time. Then, for each register the circuit is emulated for a certain number of clock cycles and the resulting error probability at the output pins is measured. By doing this for each register one can analyze which register in the circuit has an influence on which output pin. This result can be used to build a matrix with the dimensions $numInj \times numOut$, showing the relation of approximations at the registers to the effects on the output pins. The value of $numInj$ corresponds to the number of injection spots, i.e. usually the number of registers, and $numOut$ the number of outputs. This matrix will be called in the following "Probability-Relation-Matrix" PRM. Each row is showing the error probability that could be observed at the outputs when the error probability of that specific element has been set to 0.5, and all others to 0.0. It is important to keep in mind that the matrix is only showing the relation when one single register has been approximated, not multiple at the same time. The procedure is again shown in Algorithm 1. This matrix can be graphically visualized making an interpretation more

Algorithm 1 Determination of the "Probability-Relation-Matrix"

```
for i = 0 to numInj − 1 do
    for j = 0 to numInj − 1 do
        p_e[j] = 0.0
    end for
    p_e[i] = 0.5
    output_errors = emulate_circuit_for_x_cycles(p_e)
    for k = 0 to numOut − 1 do
        PRM[i][k] = output_errors[k]
    end for
end for
```

easy. Figure 4.5 is showing a plot of the PRM of an exemplary benchmark circuit, a

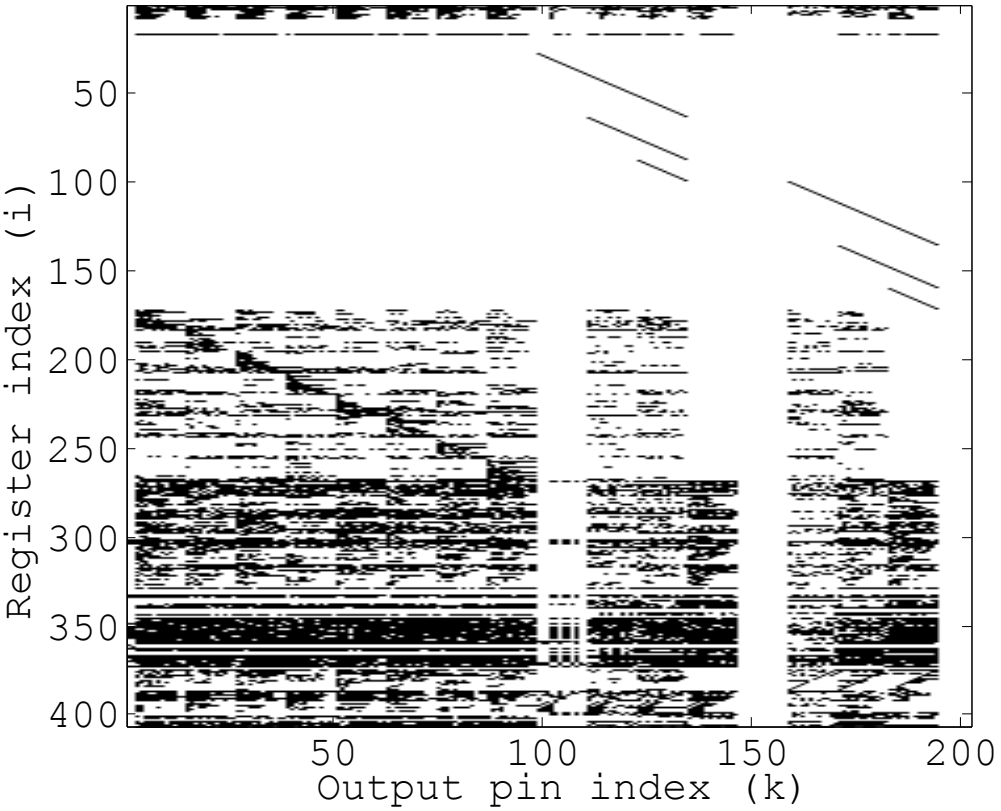


Figure 4.5: Visual representation of the "Probability-Relation-Matrix" of benchmark circuit "QR Decomposition" [122]

QR decomposition, where each error probability $p_e > 0.0$ is marked as a black dot. The visualization makes it easy to analyze the relation between registers in the circuit and the output pins. Assuming that the information of the PRM is correct, it is now easy to separate the data from the control path elements, by applying the following simple rule:

Rule 1 *If errors in row i do only appear at outputs k that are related to the data-path, the corresponding register with index i is a data-path register. If, however, errors are visible in at least one control-path output, the register is classified as a control-path register.*

This rule is actually very similar to the netlist-based approach. The difference however is that it relies on the correctness of the generated PRM. Due to the probabilistic nature of the generation of the PRM, the emulation-based separation is not as exact and strict as the netlist-based approach. That means that in the worst case false-positive classification can be made. In our case a false-positive classification would be a register that is technically part of the control path, but has been detected as a data-path register. This can happen, when faults that have been injected into such a register, could not be observed

at control related outputs. The reason for this is that the emulation is not able to cover the whole state space. Recalling, that a circuit has 2^{numFF} states, one can see that it is difficult to cover even a fraction of the state space. Hence, as the system is only able to emulate a fraction of the state-space, false-positive classifications cannot be avoided. However, the proposed algorithms try to minimize them. Nevertheless, it is also possible that the influence of a control path register on the control flow exists but is negligible, and hence a approximation would be possible. In this case false-positives would be desired. Less critical are false-negatives. That means, a register is detected as a control path register even though it is a data path register. The appearance of false-negatives is nearly impossible. If a register influences at least one control output it is classified as a control path register, otherwise not. However, it is possible that the fault injection triggers no faults at all at the output pins, neither data nor control outputs. This can happen for instance if the test patterns are not triggering all states of the circuit. In this case, as one cannot be sure to which path the register belongs, it is safely classified as control path register. Most important for the generation of the PRM is clearly to provide accurate test patterns to the application. Not only that it triggers all functionality, all states of the circuit, but more important, that it triggers the functionality that is later needed when running the application in the field.

The only parameter of an emulation that can be modified in order to minimize false-positive, is the number of emulated clock cycles. Clearly, the longer the circuit is emulated the more faults will be triggered and the more states will be emulated. Furthermore, the appearance of faults is clearly depending on the probability of faults assigned to the flip-flops. This is why faults are injected with an error probability of 0.5. Hence, the value of a flip-flop is completely random. Unfortunately, it is not possible to infinitely increase the number of emulated cycles to emulate as much circuits states as possible. A trade-off has to be found between accuracy and speed. However, it pointed out that instead of generating the PRM once with a large number of emulated cycles, the false-positive/negative rate can be improved by a simple trick. Repeatedly generating the PRM, even with less emulated clock cycles, improves the accuracy. This effect is shown for exemplary benchmark circuit in Table 4.1. The reference for the detection of false-positives and false-negatives is the netlist-based separation as presented in the previous section. The column with the label "single-trial" is showing the number of false-positives and false-negatives when separating the netlist applying Algorithm 1 and Rule 1, for 50×10^5 emulated cycles. One can see that the number of false-positives is already quite small. Not shown here in this table is that the number increases when emulating fewer clock cycles as shown in [143]. The number of false-negatives, i.e. no faults could be observed at the output pins, is fairly high, especially for benchmark circuit "QR". This is unwanted as it means that potential candidates for the approximation are lost and hence less power could be saved. The column labeled as "multi-trial - not combined" contains the number of false classifications when performing the separation based on Algorithm 1 several times ($N_{rep} = 50$). The single runs are now performed each with less emulated clock cycles 1×10^5 but in sum again 50×10^5 clock cycles are emulated. *After* applying Rule 1 the results of the separation are combined by building the average. One can see that this time the number of false-negatives has decreased but the number of

Table 4.1: Evaluation of simulation-based data-path separation, by false-positive and false-negative detected registers (each in total 50×10^5 emulated cycles)

circuit		single-trial	multi-trial not comb.	multi-trial comb.
b13	false positive	0	3	0
	false negative	3	0	0
b14	false positive	0	3	0
	false negative	0	0	0
spacewire	false positive	3	4	1
	false negative	0	0	0
QR dec.	false positive	2	4	3
	false negative	45	27	8

false-positives has increased. The combination only helps to detect previously undetected relations between registers and data outputs as a *broader* state space is covered. This in turn reduces the number of false-negatives. Due to the reduction of emulated clock cycles not all connections between registers and control outputs can be detected. The number of false positives increases. It looks like that a combination of the first and the second method is desired to minimize false-positives as well as false-negatives. Again the "PRM" is generated several times N_{rep}, 50 in this example. But instead of applying Rule 1 immediately to each "PRM", the "PRMs" are summed up. Once the 50 are generated and the sum of them is calculated, Rule 1 is applied. One can see in Table 4.1 that this time the number of false-positives as well as the number of false-negatives reduces. The false-positive rate increases as all detected faults are considered for the separation. The false-negative rate improves as a broader state space is captured. In case of the "single-trial" method, x consecutive cycles have been emulated, starting from a single circuit state. In case of the "multi-trial" methods, all random number generators, not only from the fault generators, but also from the test pattern generators, receive a new seed in each run. This obviously helps to capture as many connections between registers and outputs as possible. Clearly, the numbers could be improved by emulating more clock cycles and by increasing N_{rep}. But even for 50×10^5 emulated clock cycles, the results are quite good for this purpose. In order to support the separation on the user side, the following API function has been introduced.

```
1 int faultify_dpcp_separation(struct faultify_handle ctx,
2                               uint32_t num_dp_registers,
3                               uint32_t * dp);
```

Listing 4.1: API call to perform the separation of data and control path

When calling the function, automatically the data-path is detected, returning an array of the indices of the data path register.

Both separation methods presented in this work can equally be used to separate the data from the control path. The netlist-based method is very strict and makes a clear separation. Due to this exactness sometimes potential approximation candidates get lost. The emulation-based method instead offers a less strict, but also less accurate method. If not carefully performed it is possible that registers are classified wrongly, and faults will hence be tolerated in control critical parts of a circuit. For circuits where a strict separation between control and date path is not possible an additional intermediate step is required before actually approximating the circuit, in order to make the emulations reproducible and trustworthy.

4.1.3 High-Variance Register Exclusion

In order to identify registers that are qualified for an approximation, sometimes the data-path separation is not sufficient. There also exist circuits where a clear separation between data and control path is not possible. For some circuit the separation based on the output pins is not possible as the output pins cannot be categorized. Nevertheless, it is absolutely necessary that an approximated circuit has an reproducible behavior. The results of the emulations have to be trustworthy. For these circuits a simple algorithm is applied to detect if a register has a "linear" connection to the output pins or if the influence is varying a lot for different runs. For this, the PRM presented in the previous section is simply being build n times and the variance of the values is calculated. If the variance of a connection between one register and one output pin exceeds a threshold, this register is not suitable for approximation. The threshold has to be carefully chosen. A very small value is proposed in the range of $10^{-10} - 10^{-12}$. However, for very tight constraints on the stability of the resulting error probabilities of the output pins, an even smaller value has to be chosen. The API extensions used to detect registers that result in a unacceptable high variance of the error probability at the output pins is shown in Listing 4.2. The number of repetitive generated PRMs "n" as well as the variance threshold is a parameter of the function.

```
1 int faultify_high_v_detection(struct faultify_handle ctx,
2                   double threshold,
3                   uint32_t n,
4                   uint32_t num_high_v_register,
5                   uint32_t * high_v);
```

Listing 4.2: API call to detect high variance registers

4.1.4 Evaluation

It could be shown why the separation of data and control path is mandatory for the presented methodology to approach the approximation of integrated circuits. The classification of the locations of faults is also presented in other research, and it is in general similar. For instance Daveau et al. [84], perform a classification of faults in their fault

emulator as well. For instance, if the effect of a fault on the output pins results in "unexpected behavior [...] and possibly incorrect control flow" it is classified as a *failure*, whereas "incorrect numerical results" will be classified as *errors*. In order to evaluate the separation techniques the quality has been tested with several benchmark circuits. The netlist-based approach serves as a reference, as its separation is, by definition, the sharpest one. Figure 4.6 is showing the impact on the circuit outputs, when injecting faults in the data-path only, based on the previously presented methods. Faults have been injected with a probability of $p_e = 0.0001$. The result of an ideal separation would be that faults are only observable at the data-path related output pins. The gray boxes in the background of the figure are showing where faults can be tolerated, i.e. the data-path related pins of the circuits. Hence, ideally at those pins where the background is white, the control-path related pins, the measured mean error probability is $p_e = 0.0$. For instance, when looking at the results of benchmark circuit "b13" one can see that the only data-path related pin is at pin index 10. Ideally, one should observe faults only at this pin. One can see that when injecting faults into all registers (*no separation*) faults are not only observable at pin 10 but also at all other pins. This is not surprising as faults have obviously been injected also into the control-path. However, one can also see that the same behavior can be observed when injecting faults into the data-path, as detected by the *multi-trial* (not combined 50E3) method. The separation based on this method did not work well, parts of the results were false positives. The results of the injection into the data-path, detected with the netlist-based approach (*ideal separation*), are optimal as they show that faults were only observed at pin 10, which is not further surprising. However, the results of the separation based on the *multi-trial comb.*, as well as the *single-trial* method also meet the constraint. This corresponds to the findings presented in Table 4.1. The simulation of benchmark circuit "QR" instead behaves differently compared to the findings in Table 4.1. While the analysis of the separation, as shown in the table, showed that false-positives have been created with all methods, these false-positives are not visible in the fault injection experiment. Faults are only observable at control-related outputs when injecting equally into all registers, data and control-path (*no separation*). All separation methods work fine. This benchmark circuit gives an example where the netlist-based separation by definition gives the correct separation result, but not the optimal one. The separation of the netlist-based method is sometimes too strict. The emulative approaches usually detects more registers that could tolerate faults which in the end increases the power savings of the approximation. The evaluation has been performed for benchmark circuit "spacewire" which is a mainly control centric circuit. The results are very similar to benchmark circuit "b13". The best results can be seen for the separations based on the netlist (*ideal separation*) and the improved emulative method (*multi-trial comb.*).

We have seen that the methods developed in order to automatically separate the data from the control-path have a different quality. While the netlist-based separation never fails, as it always gives the ideal result by definition, sometimes the separation is too conservative. That means that registers that could experience some degree of approximation, are not categorized as those. The emulative methods instead tends to create false-positives. Registers that are related to the control-path of the circuit are catego-

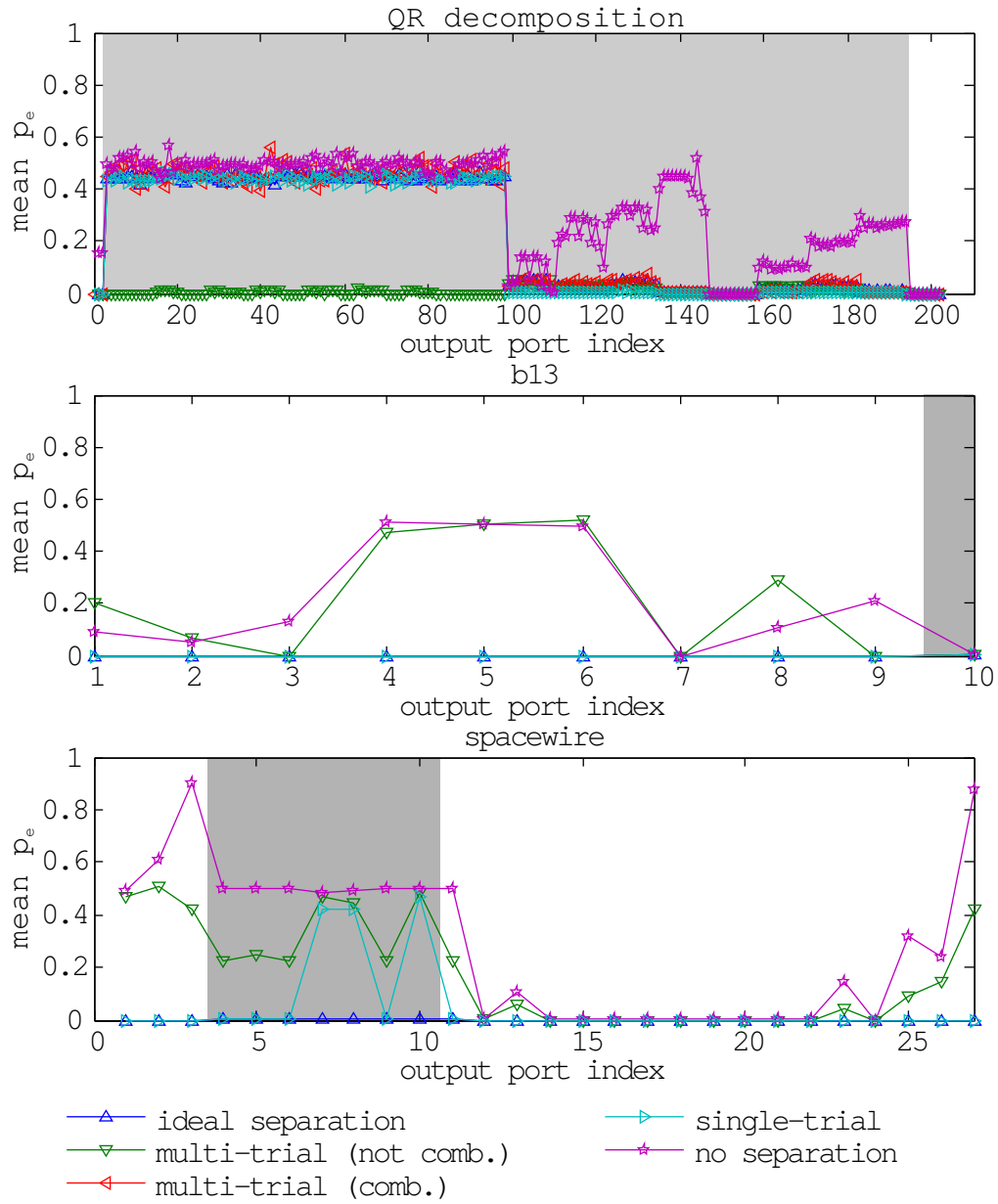


Figure 4.6: Emulation results of different evaluation circuits when injecting errors with a probability of $p_e = 0.0001$ into registers based on different separation techniques as presented in Section 4.1, each emulating 50×10^3 clock cycles [143]

rized as data-path registers. This might not always result in problems, but if it does, the effects can be tremendous. On the other hand, the emulative approach embeds nicely into the framework and it is easy to perform. The extended "combined multi-trial" algorithm seems to offer a good trade-off between usability, false-positives and strictness of the separation. This is why in the experiments presented in this work usually this methods has been chosen. The netlist-based methods however, offers a very good alternative whenever absolute correctness is required.

4.2 Result Significance

We have seen that the proposed probability awareness allows to model a variety of approximation techniques and apply various fault models. The assignment of error probabilities to the circuit elements also means that the faults that are observed at the circuit outputs have to be measured in terms of their rate of appearance. By loosening the restriction that faults do either appear or not, one will eventually be able to tolerate faults as one can quantify the rate and the resulting impact on the application. Nevertheless, analyzing the rate of faults introduces problems that have to be tackled. Even if faults are injected, hence tolerated, only in the data path of the circuit, as it has been proposed above, the behavior still remains probabilistic. That means that the rate is always an average over a chosen sample time. That in turn means that depending on the value of the assigned error probabilities, the circuit itself and of course the chance, it varies how long it takes until the measured error probability at the output becomes stable. Unfortunately, regular sampling strategies, helping to define a confidence interval, are difficult to apply here. Clearly, the smaller the error probabilities are, the more unlikely it is to observe a fault. But the appearance of faults at the output pins is also depending on the structure of the circuit and the applied test pattern. Hence, no general formula can be developed. Other methods had to be developed, estimating the required number of emulated clock cycles, in order get a stable and trustworthy estimation of the resulting error probability.

4.2.1 Variance-based Approach

In order to quantify the stability of a probabilistic experiment, the variance can be chosen as a metric. In the presented case one could repeat an experiment with identical settings and identical test pattern several times and calculate the variance of the measured output error probabilities. The results will vary because the random number generators will be supplied with a different seed and hence the random numbers will differ. Therefore, the faults will appear at other points in time. The variance is the expectation of the squared deviation of the mean error probability. The smaller the variance is, the more does the measured mean error probability correspond to the real one. If the calculated variance is above a chosen threshold, the duration of the emulation would have to be increased, in order to measure a mean error probability that is more close to the real one. If the variance is below the threshold, the mean value can be accepted. The calculated mean and the variance for a different number of emulated cycles is exemplary shown for

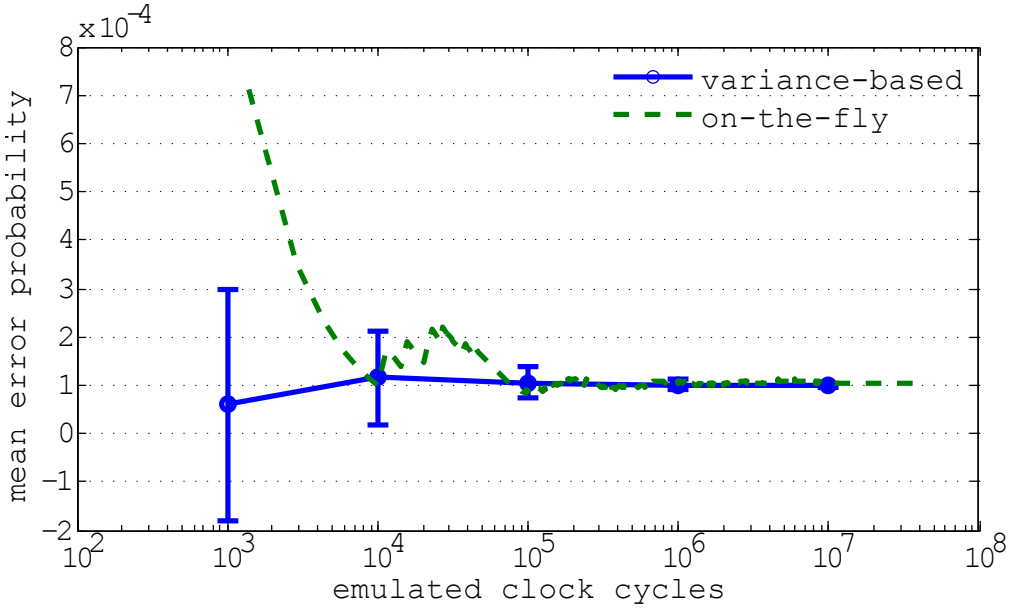


Figure 4.7: Error probability mean and variance over 50 trials for variance-based and "on-the-fly" approach at the data output pin of benchmark circuit b13. Equal injection of errors with $p_e = 0.0001$ into the data path. [143]

benchmark circuit b13 in Figure 4.7. The variance and the mean are calculated over 50 trials. One can see that the calculated variance is decreasing when increasing the number of emulated clock cycles and that the mean value is approaching to a steady point. The drawback of this method is that one usually does not want to emulate a circuit with a specific injection configuration with an unnecessary large number of clock cycles. The emulation speed, as it has been explained already several times, is critical as millions of different injection configurations have to be tested. On the other hand, one does not want to have to repeat a trial if the variance is above a threshold, as this would also cost precious time. Hence, the applicability of a method calculating the variance after the experiment is limited.

4.2.2 On-the-fly Approach

In the following a different approach will be presented, trying to estimate the stability at run-time, on-the-fly. This allows to overcome the limitation of the simple variance-based approach, as the simulation can be executed exactly as long as it is required. Also, there is no need any more to repeat an experiment with identical settings several times in case the number of emulated clock cycles has been chosen too small. The fundamental difference of the on-the-fly approach compared to the variance-based approach is that the results of the emulation, the measured error probabilities, are not determined at the end of the emulation but already while the emulation is running. By reading back the results of the emulation periodically in defined intervals, on-the-fly at run-time, one can

estimate the stability of the result and run the emulation exactly as long as it is required. Figure 4.7 is showing besides the variance-based approach also the sample mean error probability read back at run-time from the emulator. The benchmark circuit is the same as before, namely "b13". One can see that the measured error probability is approaching a steady value with an increasing number of emulated clock cycles. The *weak law of large numbers* [123] generally states that for a sufficiently large number of trials, the standard error will stay below a limit, as shown in Equation 4.1.

$$\lim_{n\to\infty} \mathrm{P}\left(\left|\overline{X}_n\right| > \varepsilon\right) = 0\,, \tag{4.1}$$

where $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n}(X_i - E(X_i))$ for any positive value of ε. Trials are in our case the number of emulated clock cycles. In order to estimate when the result is stable enough, the following equation has been developed [143]:

$$s_{oi}{}^2 = \frac{1}{N-1}\sum_{k=i-N}^{i}\left(\frac{\mathrm{soe}_{ok}}{t_k} - \frac{1}{N}\sum_{l=i-N}^{i}\frac{\mathrm{soe}_{ol}}{t_l}\right)^2 \leq \tau \tag{4.2}$$

, where soe_{ok} corresponds to the sum-of-errors at output-pin o at time instant t_k. In other words, the variance of the last N sample mean error probabilities is calculated periodically. If the variance ${s_{oi}}^2$ is below a certain threshold τ for all outputs o it is assumed that the measured error probability is stable. However, in order to make the algorithm more robust, the rule has been extended, so that the calculated variance has to stay below the threshold for M samples. Note, that M, N and τ have to be manually chosen based on the users demand, no general recommendation can be given. A pseudo-algorithm describing the on-the-fly approach is shown in Algorithm 2. Applying this

Algorithm 2 On-the-fly variance estimation

```
while done!=1 do
    i++;
    [time, soe] = ReadSumOfError()
    current_mean = soe/time;
    mean_array[i] = current_mean;
    current_variance = variance(mean_array[i-N:i]);
    std_array[i] = current_variance;
    if (std_array[i-M:i] < stability_threshold) then
        done=1;
    end if
end while
```

simple rule to the emulation makes it now easy to trust the measured results. The emulation API has been extended accordingly, as shown in Listing 4.3.

```
1 int faultify_run_auto_stop(faultify_handle ctx,
2                            uint32_t M,
3                            uint32_t N
4                            double tau,
5                            uint32_t * result);
```

Listing 4.3: API call to run the emulation long enough so that the results are stable based on the on-the-fly estimation of the variance

Executing this API call is running the emulation exactly long enough so that the measured resulting error probability at all output pins is becoming stable, as defined by the parameters N, M and τ.

4.2.3 Evaluation

In this section it will be evaluated whether the algorithms presented in the previous section allow to reliably estimate the number of required clock cycles in order to get a stable result. We have seen that a stable and reproducible emulation result means that the variance between different runs is small. It also means that the measured mean error probabilities at the output pins should approach a stable value, the more clock cycles are emulated. For three benchmark circuits the mean variance of the output pins is shown in Figure 4.8 on the left side in dependence of the number of emulated clock cycles. Faults are again injected with an error probability of $p_e = 0.0001$ into the data-path only. The data-path has been determined using the methods presented in the previous sections. One can see that in most cases for benchmark circuit "b14" and "spacewire" the variance is decreasing with the number of emulated clock cycles and also the better the separation has been performed. In case of benchmark circuit "QR" one can see that for the *multi-trial* separation method the variance is increasing with increasing emulated clock cycles. This is an indication that the separation contains false positives, affecting the control path. One can see that the quality of the separation is critical for the stability of the measured result. The right column of Figure 4.8 is showing the estimations of the required emulated clock cycles when using the on-the-fly method as presented before, depending on the desired threshold τ. Important similarities to the left column can be seen. Most importantly, it can be observed that the smaller the threshold τ has been chosen, the longer the circuit has to be emulated. This is what has been expected and what could now be verified with these experiments. For circuits, like the "QR" benchmark circuit, it is difficult to reach a small variance. When desiring a very small "stability threshold" τ the emulation time exponentially increases. For other circuits, a very stable result can be reached very soon. Furthermore one can see that the more accurate the separation has been performed, the faster a stable result can be reached. For instance, when injecting errors into registers based on the separation of the *not comb. multi-trial* method in benchmark circuit "b13" much more cycles are required to get the same "stability" of the result as compared to when using the *ideal* separation. In the left column, the same behavior can be seen. The relation between the variance of the mean error probability at the output pins and Equation 4.2 is clearly given. For most experiments in this work the on-the-fly method has been chosen to determine the stability of the results, due to

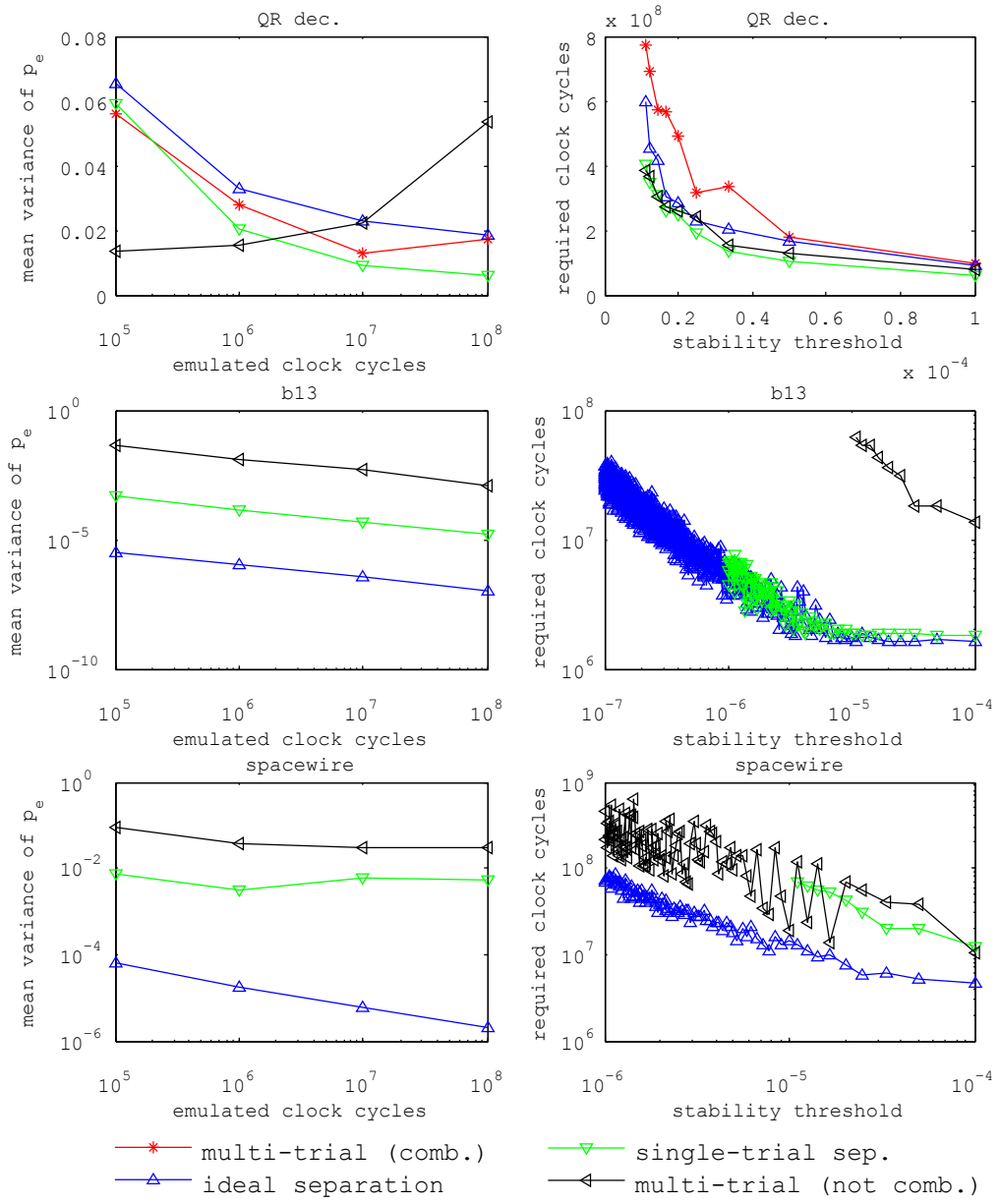


Figure 4.8: Figures in the left column are showing the mean variance of the measured output error probabilities for different benchmark circuits when injecting errors with a probability of $p_e = 0.0001$ into registers based on different separation techniques and for a different number of emulated clock cycles. The right column is showing the estimated required number of cycles for different stability thresholds using the method presented in Section 4.2.2 [143]

the fact that it is much faster than repetitively calculating the variance, as it has been explained before.

4.3 Application-reasoned Approximation

The methods presented in the previous sections were solving two important questions. Due to the separation of data and control path it is now clear which parts of the circuit do qualify for an approximation at all. And due to the real-time measurements of the result stability one can decide at run-time if a sufficiently large number of clock cycles has been emulated. One important step is missing before one can start with the actual approximation. In order to analyze which parts of a circuit can be approximated, one first has to know the worst case behavior of the circuit that the application, where the circuit is embedded in, can tolerate. Hence, the maximum tolerable error probabilities at the circuit output pins has to be determined. In order to gain this information the whole application has to be considered. By running the application in their natural environment, hence with realistic stimuli, the information about the required quality has to be brought down from the application level to the register-transfer level. In this work it is proposed to do this by injecting faults at a high abstraction level into the application, and observe the impact on the application quality. Figure 4.9 is visualizing the idea. The

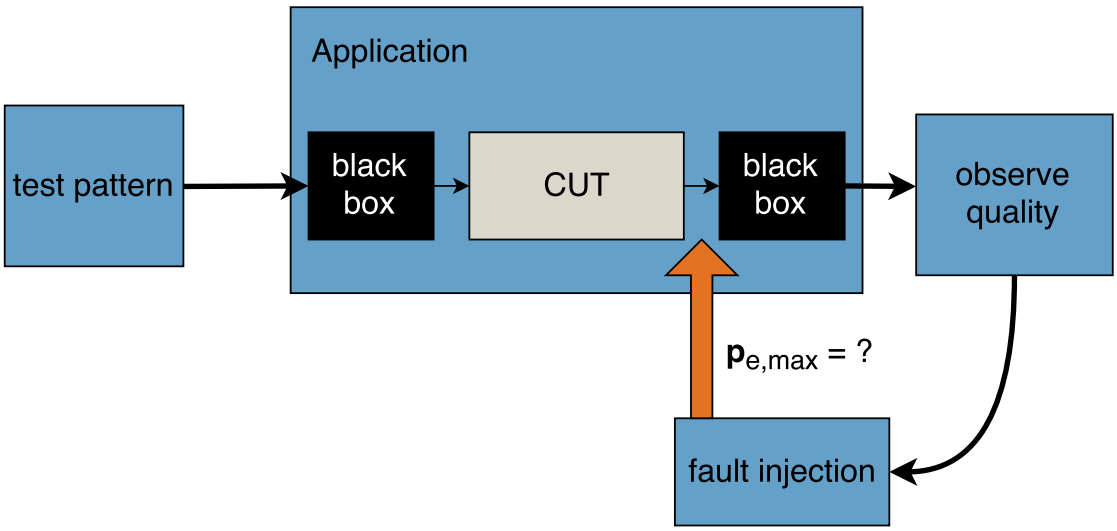


Figure 4.9: Software-based fault injection at application-level to determine the possible approximations at register-level for the circuit under test

circuit-under-test, hence the circuit that has to be approximated in the end, is embedded in an application. There might be other circuits preceding or following the circuit. It is assumed that there exist means to measure the quality or precision of the circuit. Now one can provide realistic stimuli to the circuit and run it. By injecting faults directly behind the circuit-under-test one can actually simulate an error probability at the output pins of the circuit caused by its approximation. By changing the error probability of the

injected faults and observing the impact on the application behavior one can determine how imprecise the circuit-under-test is allowed to be. It is important to keep in mind that this is not the actual approximation at hardware level. This simple fault injection is ideally performed at very abstract application models, e.g. developed in C, MATLAB or SystemC. It is important that the interfaces of the circuit-under-test correspond to the hardware implementation, however cycle accuracy is usually not required. Clearly, the analysis could be also performed with cycle accurate simulation models, but this would again blow up the complexity and simulation time. The divide and conquer approach pursued all over this work, is fundamental to approximate integrated circuits in a reasonable amount of time and with less effort. Additionally, it is very important that the test patterns supplied are very realistic, so that the behavior corresponds to the one it would have later in the field. The procedure will be explained more in detail with two exemplary test circuits. The first one is a generic *MIMO wireless system*. The application has been implemented as a MATLAB model. Random payload data is generated and transmitted over a fictional noisy wireless channel. The transmitter and the receiver use 8 antennas. The modulation has been set to 16 QAM. The receiver is trying to recover the received signal by using a "zero-forcing" equalizer, hence estimating the transfer function of the wireless channel. In order to do so the receiver has to calculate the inverse of the 8×8 channel matrix. This is a relatively complex task, which becomes more complex the more antennas are used. One efficient way of calculating the matrix inverse utilizes the QR decomposition, that has been already presented earlier in this work. Later in this work, the hardware implementation of the QR decomposition [122] will be be approximated. This is a realistic scenario, as on the one hand the implementation is complex, and therefore a reduction of the power consumption could be desired. And on the other hand this circuit is suitable for Approximate Computing as its required precision can change over time. With varying channel conditions the requirements on the precision of the QR decomposition could be dynamically adapted. For the scope of this work three arbitrary operating points of the MIMO receiver are defined. The target bit-error rate should not exceeded BER = 0.01. The channel quality is however varying between 30, 40 and 50 dB signal-to-noise ratio. The channel is assumed to be Gaussian. Hence for this application, one is looking for the maximum error probability at the output of the QR decomposition so that for these three channel qualities the overall bit-error rate of the system is not exceeding the threshold.

The second application that will be considered in detail in this and the following chapter is a simple *Sobel* image filter. A Sobel filter is a simple edge detection filter. The application is implemented in C. Compared to the previous example it differs in two major points. First of all, the hardware circuit that is going to be approximated is not a dedicated hardware accelerator, like the QR decomposition, but a floating-point unit [124] embedded in a CPU pipeline. And secondly the environment in this example is static. Instead the target quality of the filtered image could be changed according to the quality demand or the power budget. Three arbitrary target image qualities have been chosen, namely 30, 40 and 50 dB PSNR. Hence, the tolerable error probability at the output pins of the FPU is tried to be identified for the three operating points. The image quality is determined in relation to a fully precise, hence fault free, Sobel filter. Unfortunately,

bringing down the information about the required precision from application level to the hardware level, cannot be fully automated. It is inevitable that simulation models of the application exist in order to gain the information in a reasonable amount of time. Accurate test pattern are required in order to stimulate the same behavior of the circuit as it would be later in the field. Additionally, the simulation model has to be modified to provide means to inject faults at the position of the circuit-under-test outputs. However, as the simulation models are ideally on high abstraction level, the modification is usually easy. And the effort is worth it, as it simplifies the actual approximation, as one can limit the analysis to the circuit-under-test and does not have to consider the whole application.

In order to trace back the information about the application quality to the output pins of the circuit-under-test using simple fault injection at simulation level, a simple two-step algorithm has been developed. The algorithm is actually very similar to the one used for the approximation in the next section. Dividing the algorithm into two steps simplifies the problem and speeds it up a lot. In the first step the "integer" approximations are determined. With the integer approximations these output pins are identified, that can be actually ignored as their influence on the overall behavior of the application is minimal. These pins will likely later on be important for the approximation technique "circuit pruning" as all circuit elements that do only influence an output pin that is not used are actually not used as well. In terms of error probability, the maximum error probability at these pins corresponds to $p_e = 0.5$. This "integer" approximation at block level is usually a question of the required precision. The problem is also related to fixed-point arithmetic [125]. Assuming that the output interface of a circuit is a n-bit integer number, this step identifies how many of these n bits are actually required. If some of the output bits could be ignored it is likely that these are the least significant ones. The proposed algorithm makes use of this fact. Starting with the least significant bit of the outputs, the outputs pins are consecutively switched off ($p_e = 0.5$) and the effect on the application behavior is observed. If the quality of the application still meets the constraint one can assume that this bit can be ignored and one can continue with the next more significant bit. For most circuits, the problem space is quite narrow, and hence the complexity simple. For instance in case of the "QR decomposition", the precision of two (Q and R) 12 bit integer numbers has to be analyzed. In case of the FPU, three (fmul/fdiv/fsqrt) 32 bit IEEE 754 floating point numbers have to be optimized.

The second step proposed in this work is the fine-granular approximation. In this step those output pins are searched that cannot be ignored completely but still can be imprecise to a certain degree. These pins are becoming very important later on for more complex approximation techniques like voltage over-scaling, where elements are not switched off completely but operated imprecisely. In order to gain this information the error probability of the output pins is increased successively. Starting with the least significant bit that could not be pruned, the error probability is increased by a defined value. If the resulting quality degradation of the application is within the limits, one can continue increasing the error probability, and if not one has to stop and continue with the next more significant bit.

The results of this high-level application-reasoned approximation are shown in Figures 4.10 and 4.11. For each output bit the tolerable error probability is shown, for the

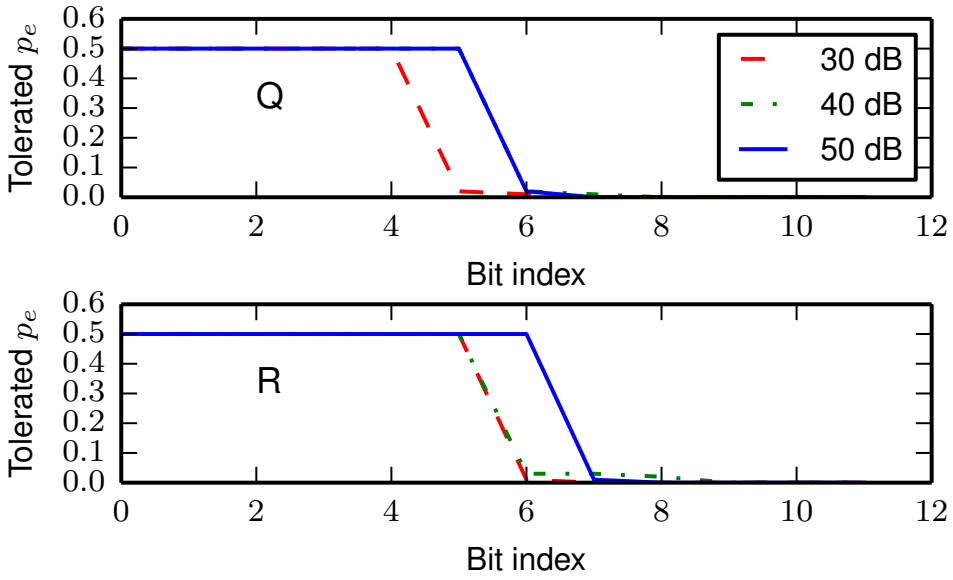


Figure 4.10: Tolerable imprecision of a QR decomposition, part of a 8x8 MIMO zero-forcing equalizer, for different signal qualities and a target BER=0.01 [144]

previously presented operating points. In case of the MIMO application, one can see that for the "Q" part up to 6 bits can be pruned (denoted as ($p_e = 0.5$), if the channel quality is 50 dB SNR. For the R part the first 7 bits of the 12 bit number can be pruned when the channel quality is at 50 dB SNR. If the channel quality is getting worse, less bits can ignored. It is clear that this large number of bits that can be ignored is specific to this application. For other applications the least significant bits might be more important. However, this shows why it is so important to reason the approximations of circuits based on the application that embeds it. One can see that, for instance for the "R" part, between 40 dB and 50 dB there is a difference of one bit in the required precision. This might seem small, but as we will see later, this result in a noticeable difference in the possible approximations within the circuit. The fine approximation is also visible. One can see for instance that for the "R" output, at 30 dB for bits 7 and 8 hardly no error probabilities can be tolerated. If the channel however improves to 40 dB bits 7 and 8 may experience a small, yet noticeable error probability. The results for the Sobel application are similar. One can see that depending on the desired target quality up to 20 bits of the Mantissa can be completely pruned from the division and the square root unit. The required precision of the square root unit is actually independent of the target quality. If the desired quality increases to 50 dB less bits can be pruned. The algorithm also detected that the sign of the division is irrelevant in this application, as later on the square is calculated. The fine approximation is clearly visible for instance for bit 19 of the division unit at 30 dB PSNR target quality. One can see that even though the algorithm is simple, it suits to detect the tolerable imprecision at the output of the circuit that

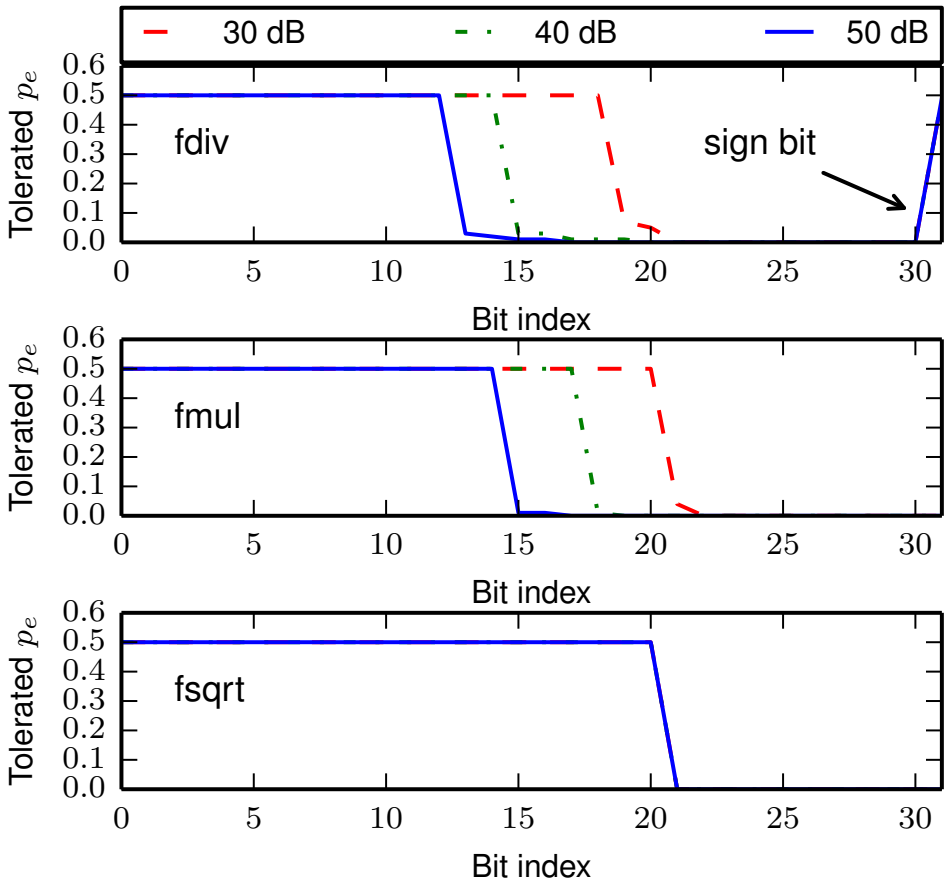


Figure 4.11: Tolerable imprecision of floating-point operations in a *sobel* filter for different target qualities (PSNR) [144]

has to be approximated. Due to tracing the information of the required quality from the application to circuit outputs, one can now analyze the circuits isolated from the application. This simplifies the approximation optimization problem tremendously and one can now finally start with the approximation of the circuit.

4.4 Coarse-grained Approximation

Due to the steps performed in preparation of the approximation, namely the data-path separation and the elimination of high-variance registers, the actual approximation has simplified a lot. Now that the maximum tolerable error probability at the output pins of the circuit-under-test is known, one can go one step down in the hierarchy for the approximation of the circuit. The goal of the approximation is to find out which elements in the circuit can be approximated and to what degree. Due to the reasons mentioned earlier in this work the focus at this point is put on the approximation of flip-flops. Hence, the

maximum tolerable error probability of each flip-flop is searched, so that the constraint on the precision of the output pins are still met. Later in this work when applying voltage over-scaling as the approximation technique, the focus will be shifted from flip-flops to logic gates, as the voltage has to be scaled individually for each gate. Nevertheless, at this point of the approximation it is sufficient to know the tolerated error probabilities of each flip-flop in the circuit. This value is actually modeling the approximation of all gates before that flip-flop influencing this one flip-flop, the so called "fanin". This abstraction of the approximation to the assignment of error probabilities to flip-flops simplifies the optimization problem a lot as the number of variables reduces dramatically. Later on, one can analyze each fanin independently and determine the possible approximations. But even finding the possible approximations at register-transfer level is a complex task. The problem is a typical global optimization problem as visualized in Figure 4.12. From a large set of possible combinations (error probabilities/approximation at the registers) one is looking for the one combination that on the one hand results in a circuit output error probability that meets the constraints previously determined. And on the other hand one is looking for the combination that gives most power savings. The most power savings can be usually achieved by tolerating most approximations, i.e. the highest error probability. As it has been presented in detail in the previous chapter, FPGA-based probability-aware fault emulation is used to gain this information. This approach allows to assign error probabilities to each register in the circuit, emulate it and observe the behavior of the circuit outputs. The utilization of evolutionary algorithms seems to be suitable to solve the optimization problem. However, it pointed out that the stochastic nature of the problem interferes with the optimization algorithm. The main problem is that the same set of injection parameters (error probabilities) leads to different results at the circuit output. The variance of the result has been already reduced with the means presented earlier, however it still tends to push the optimization algorithm in wrong directions. This results in excessive run times while generating bad results at the same time. Fortunately, it pointed out that by dividing the approximation into a "coarse" and "fine" parts reduces the complexity of the individual steps and allows to perform a simple search for an approximations in the circuit that fulfills the requirements.

In the first step, the coarse approximation, those registers in the circuit are identified that are not needed anymore and can be removed, or dynamically disabled from the circuit. If a register can be removed, consequently all previous elements having a connection only to that one register can be pruned as well. It is worth mentioning that circuit pruning not only can be used to statically remove circuit elements but can also be used to dynamically switch on and off elements either by power gating or clock gating. Clearly, statically removing and power gating the elements gives the largest energy savings. Removing the elements also reduced the footprint of the circuit. The idea of the coarse approximation is visualized in Figure 4.13. One can see that in this fictive example circuit the tolerable output error probability of $p_{e,bit3} = 0.0$, for $p_{e,bit2} = 0.01$, $p_{e,bit1} = 0.01$ and $p_{e,bit0} = 0.5$. Hence in this example the output bit 0 is actually ignored by the application, denoted by the error probability being 0.5, as the influence on the quality of the application is negligible. Now the idea of the coarse approximation is to find all gates and registers in the circuit that can be pruned as they only have

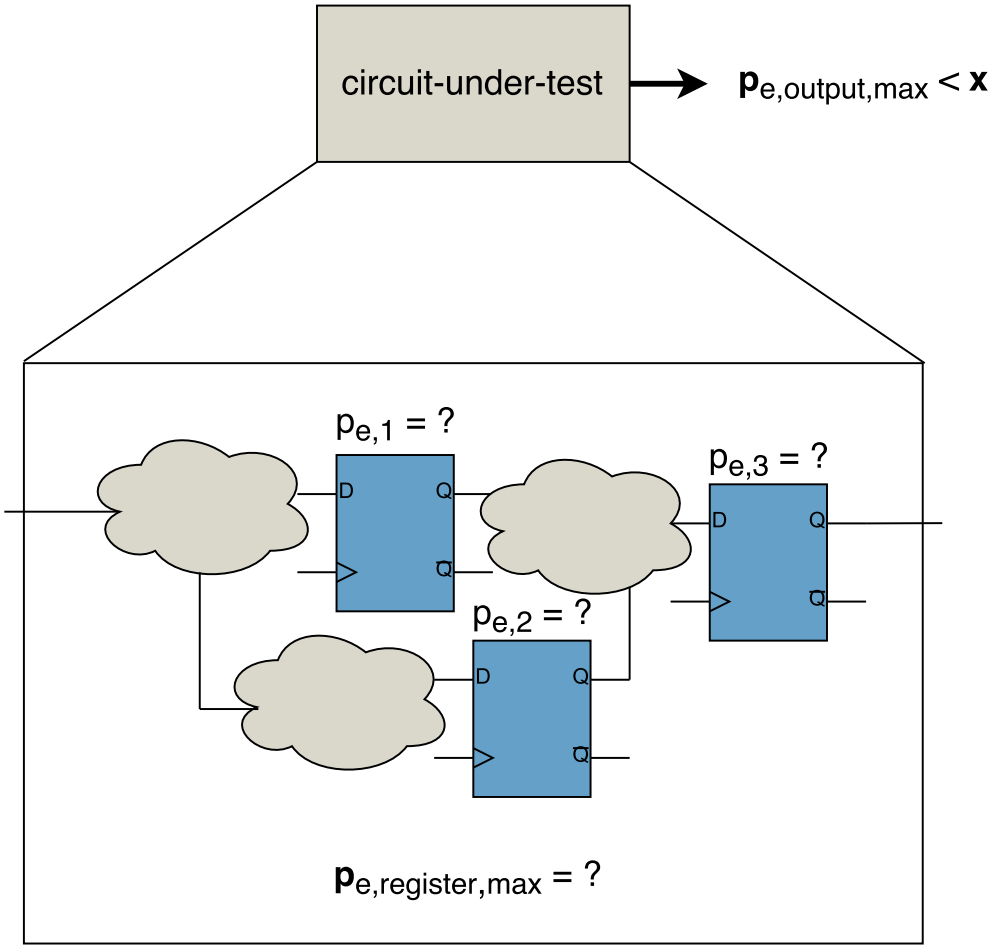


Figure 4.12: The optimization problem of Approximate Computing - finding the largest approximations for a given quality constraint

a connection to that one output. In order to find all these elements, the FPAG-based emulation is used. The emulator is used to successively switch off the approximation candidates in the circuit, one after another, and observe how this influences the error-rates at the output pins. The approximation candidates are those registers that have not been excluded in the previously presented preparation steps, i.e. control-path registers and "high-variance" registers. The subset of candidates is denoted as the vector $\mathbf{c} = \{c_1, c_2, \ldots, c_o\}$. Switching off the registers means, as it has been already explained previously, assigning an error probability of 0.5 to the register. Hence, the value of every second bit is flipped and the data contains no information anymore. After a register is switched off, the emulation is started and running as long as the results become stable (see Chapter 4.2). Once finished, one can compare the measured output pin error probability with the constraint. If the measured error probability at least at one output pin is larger than the constraint, this currently analyzed register cannot be pruned. If

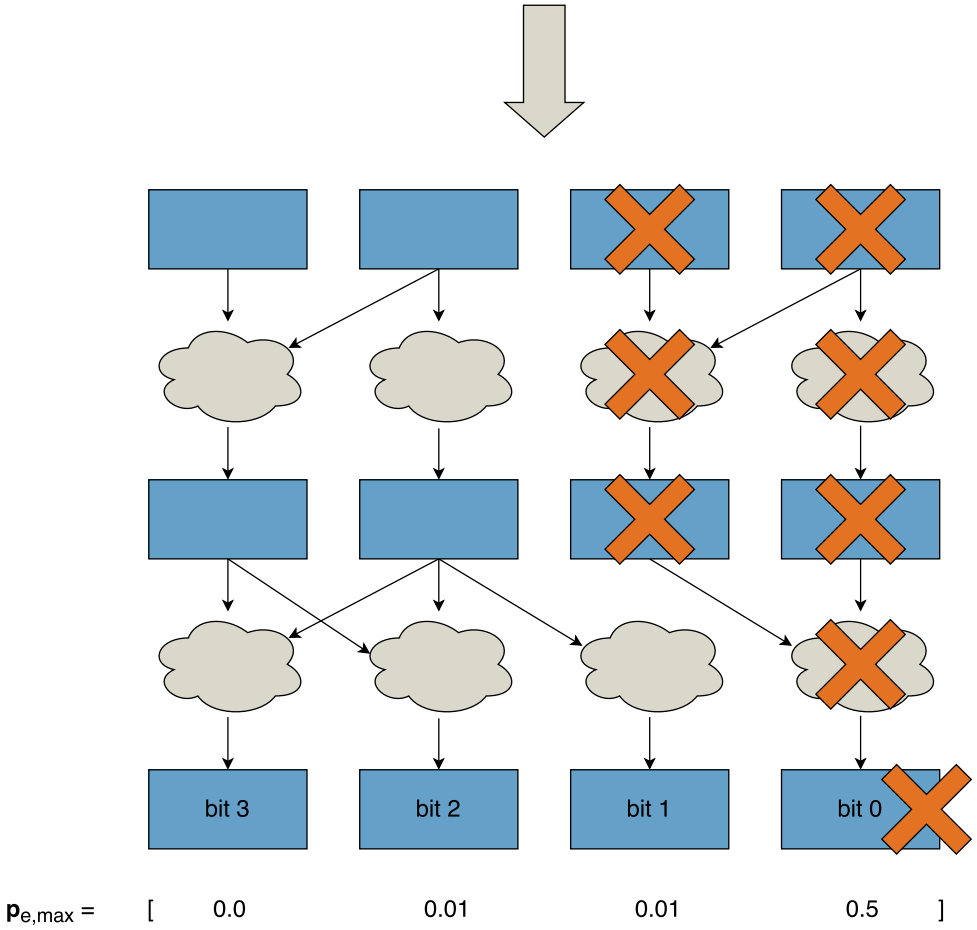


Figure 4.13: The coarse approximation is detecting all circuit elements that can be removed from the circuit as their influence on the quality of the application is negligible

however the output error-rate stays below the threshold, that register can be pruned. This step is performed for all approximation candidates o. For some circuits the order in which the registers are switched off might be important. For the test circuits used in this work however the order made no difference. This is not further surprising as if an element does really only have a connection to an unused output, it is not important if other elements have already been switched off or not. For the fine-approximation presented in the next section, the situation could be different. Due to the assignment of high error probabilities, the results are becoming stable very soon, and the execution of the coarse-approximation is comparably fast. By performing this step independently from the following fine-approximation, the set of fine approximation candidates is further reduced, reducing the complexity of the fine-approximation. The results of this analysis, can be used to apply the functional approximation method "circuit pruning", in which parts of the circuit are either removed at synthesis time or dynamically switched off at

Algorithm 3 Automated coarse approximation

```
1:  procedure CoarseApproximation(c, p_e,max)
2:      for i ← 1, o do
3:          p_e(i) ← 0.0
4:      end for
5:      for i ← 1, o do
6:          p_e(c_i) ← 0.5
7:          p_e,outputs = injectFaults(p_e)
8:          if p_e,outputs > p_e,max then
9:              p_e(c_i) ← 0.0
10:         else
11:             AppendToArray(c_coarse, i)
12:         end if
13:     end for
14:     return c_coarse
15: end procedure
```

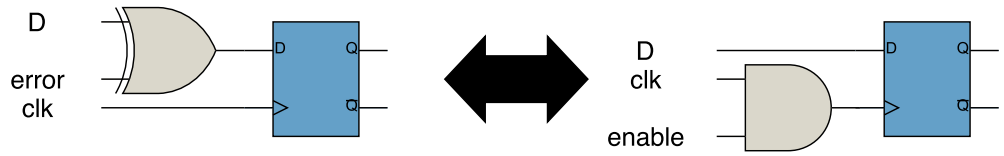


Figure 4.14: Clock gating of coarse approximated circuit elements realized by replacing fault injection instruments by flip-flops with clock enable input

run-time, either due to power-gating or clock gating. However, until now only the flip-flops that can be removed have been identified. The Boolean logic between the registers that can be pruned as well has to be identified separately. In order to do so, the optimization capabilities of the synthesis tool are again exploited. By removing the registers that can be pruned from the circuit netlist, the synthesis tool automatically removes all logic gates, that are not needed anymore. As it has been already explained in context of the data-path separation, this method is a very reliable and easy method. The methods is only required for a static pruning of the circuit. A dynamic approximation using clock gating can also be implemented easily using the netlist modified for the fault injection, by simply replacing the "fault-injection" registers with "clock-gating" registers, as depicted in Figure 4.14. Applying dynamic power gating as the approximation technique is more complex and not covered in this work. In order to support the coarse approximation on the software side, the API has been extended accordingly.

```
int faultify_coarse_approximation(struct faultify_handle ctx,
                  double * output_constraint,
                  uint32_t * coarse_approx_registers,
                  uint32_t num_coarse_approx_registers);
```

Listing 4.4: API call to perform the coarse-grained approximation of a circuit

4.5 Fine-grained Approximation

Once the identification of candidates for functional approximation has been completed, the remaining candidates are tested for their suitability for fine-grained approximation. Fine-grained approximation is the identification of registers, that cannot be removed completely, but still be operated imprecise. One does not only want to identify these registers , but also estimate the degree of imprecision that can be tolerated at these elements. In the presented approach that means that the maximum error probability that can be tolerated has to be determined. Again, the FPGA-based fault emulation is used to gain this information. The elements identified in this step can later on be approximated using approximation techniques like Boolean modification and voltage over-scaling, where the latter will be elaborated in detail in the next Chapter. In preparation of the fine-approximation, the elements already being identified as part of the coarse approximation have to be approximated, i.e. an error probability of 0.5 has to be applied to them. The idea of the fine-grained approximation is to raise the error probability of the remaining approximation candidates as long as the constraint on the output error probability is met, as shown in Figure 4.15. There are two different alternatives on how to approach this limit. One is to try to increase the tolerated error of the individual registers as much as possible. In other words, trying to approximate only a few elements but with a very high degree of approximation. The other alternative is to try to approximate as evenly as possible over the approximation candidates. Clearly, the goal is to choose that approach that offers the largest power savings. A combination of both, hence approximating many elements to a very high degree, is likely rarely possible, as the constraints on the output quality will soon be reached. For some cases it is beneficial to focus on very few, but large approximations. For instance, if there is a data-path that has a very high switching activity or if the fanin of a register is very large compared to that of other approximation candidates. In these cases it can make sense to focus particularly on the approximation of these registers as a larger approximation of these registers results in much larger energy savings. However, in most cases it is favorable to tolerate few approximations but at as many locations as possible. Previous work has shown that by tolerating a small error probability, large power savings can be made in logic gates, due to an exponential correlation [16], which is a clear argument for the second approach. Clearly, these factors could be taken into account when solving the optimization problem. A already mentioned earlier, in this work a simple “bruteforce” approach is chosen to find a valid and good distribution of fine approximations. For this work the focus has been put on validating the approach in total, and not on optimizing individual steps. However for future work,

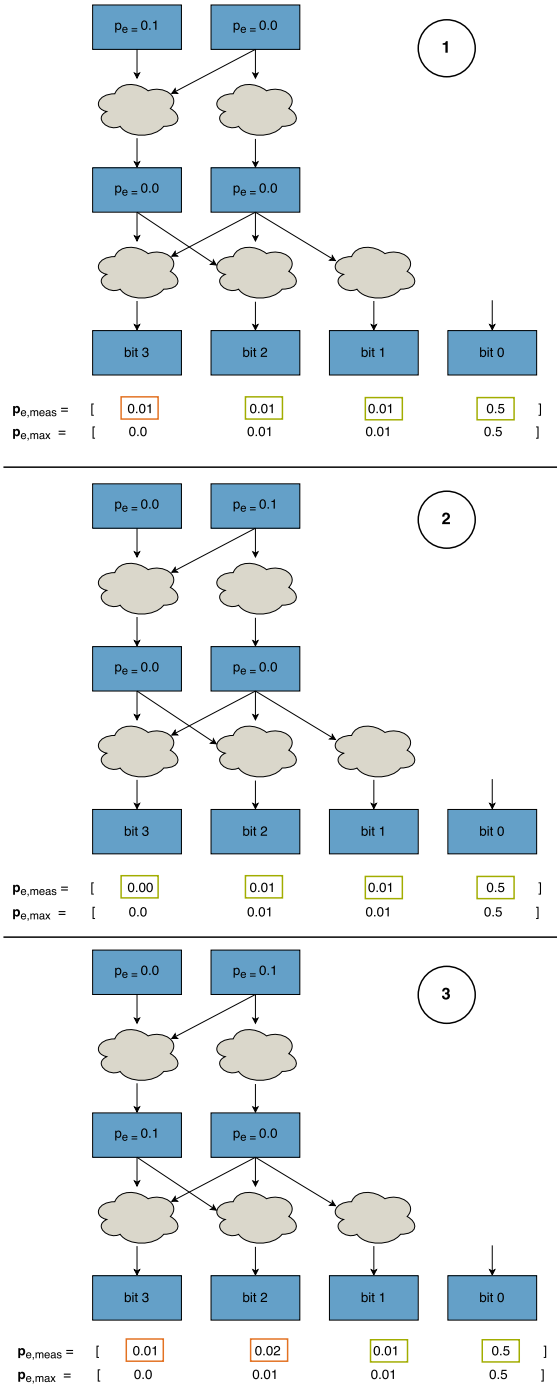


Figure 4.15: Fine approximation algorithm - step-wise increment of error probabilities at the registers

many different aspects could be taken into account in order to improve the solution of the optimization problem. Nevertheless, due to the various preparation steps preceding the approximation, the search-space is already highly sorted-out, which is why the fine-approximation can be performed with manageable complexity. In fact one can now iterate over the approximation candidates and successively increase their error rate as long as the constraints are met. The idea is shown in Algorithm 4 and visualized in Figure 4.15. Starting with one register, the injected error probability of the candidate gets increased by a factor of δ. If the increase led to a constraint violation, the error rate of this particular register is reduced again and the register is removed from the approximation candidates. Otherwise the error probability is kept. Then, the error probability of the next register is increased and the resulting error rate at the circuit outputs is checked again. This procedure is continued until the set of approximation candidates is empty. At the end of this procedure, it is known which registers can be approximated and to

Algorithm 4 Fine Approximation

procedure FINEAPPROXIMATION($\mathbf{c}, \mathbf{c_{coarse}}, \mathbf{p_{e,max}}, \delta$)
  $\mathbf{c_{fine}} = \mathbf{c} \setminus \mathbf{c_{coarse}}$ ▷ relative complement
  $\mathbf{p_e}(\mathbf{c}) \leftarrow 0.0$
  $\mathbf{p_e}(\mathbf{c_{coarse}}) \leftarrow 0.5$
  while (!isEmpty($\mathbf{c_{fine}}$)) **do**
    for $i \leftarrow 1, \text{numel}(\mathbf{c_{fine}})$ **do**
      $\mathbf{p_e}(c_{\text{fine},i}) \leftarrow \mathbf{p_e}(c_{\text{fine},i}) + \delta$
      $\mathbf{p_{e,outputs}} = \text{injectFaults}(\mathbf{p_e})$
      if $\mathbf{p_{e,outputs}} > \mathbf{p_{e,max}}$ **then**
        $\mathbf{p_e}(c_{\text{fine},i}) \leftarrow \mathbf{p_e}(c_{\text{fine},i}) - \delta$
        removeFromArray($\mathbf{c_{fine}}, i$)
      end if
    end for
  end while
  return $\mathbf{p_{e,approximation}} = \mathbf{p_e}$
end procedure

what extent. The API extension in order to perform the fine-grained approximation is shown in Listing 4.5.

```
int faultify_fine_approximation(struct faultify_handle ctx,
                  double * output_constraint,
                  uint32_t * coarse_approx_registers,
                  uint32_t num_coarse_approx_registers,
                  uint32_t * fine_approx_registers,
                  double * error_rate_fine_approx_registers,
                  uint32_t num_fine_approx_registers);
```

Listing 4.5: API call to perform the fine-grained approximation of a circuit

When combining the result of the coarse and the fine-grained approximation, the user now has a list denoting the maximum tolerable error probability at each register for one approximate operating point. A value of $p_e = 0.5$ denotes that the element (and all elements influencing only this element) can be pruned, or dynamically switched off. A value less than $p_e < 0.5$ denotes that the resulting error probability at the element input must not exceed this value. The circuit elements having an influence on the error rate at the input of this element (the "fanin") can however be operated imprecise, as long as the resulting error rate does not exceed the threshold. One method on how to operate the fanin imprecise is voltage over-scaling, which will be presented in the following chapter.

4.6 Evaluation

Approximation Evaluation In order to evaluate the presented approximation methodology, the already introduced benchmark circuits are used. Additionally, another example circuit is introduced, a *Viterbi decoder*. A Viterbi decoder is used to decode convolutional codes in many wireless communication systems. The difference to the QR decomposition and the FPU benchmark circuits already presented is, that the output of the Viterbi decoder has to be de-facto fault free in most applications. For instance, in case of "Digital Audio Broadcast" (DAB), a frame would be discarded if the checksum of the viterbi output is wrong, resulting in an immediate sound drop. The goal is to find an approximated operating point for a set of different channel qualities, so that the resulting bit-error rate remains $BER \approx 0.0$. This is a good example for dynamic approximation, where the level of approximation is changed depending on the situation of the environment. The benchmark circuits used for the evaluation of the approximation algorithms are summarized in Table 4.2. Example circuit "fpu100" is synthesized for an ASIC in Synopsys

Table 4.2: The benchmark circuits used for the evaluation of the approximation methodology

Name	Description	Flip-flops	Technology
fpu100 [124]	32-bit floating point unit	2030	Synopsys 90nm
QR [122]	QR decomposition	414	Virtex 6
vitdec [126]	Viterbi decoder (131,81)	1297	Virtex 6

90nm technology. Example circuit "QR" and "vitdec" are synthesized for Xilinx Virtex 6 FPGA technology. This shows that the presented methodology is equally suitable for ASIC as well as FPGA targeted circuits. At first the presented approximation methodology is evaluated. The approximation algorithms are performing well if they are able to find a combination of approximations at the registers for all approximate operating points of the application. Not only is it required to find an approximation combination, but also should this combination offer the best power savings of all possible combinations. However, as already mentioned before, the focus has not been put on finding the optimal combination in this work. In this work the general applicability of the automated approximation of integrated circuits and the usefulness is verified. First of all, the approximation should be performed in a reasonable amount of time. A general state-

ment about the run-time of the coarse and the fine approximation cannot be given. The run-time is depending on many factors, mainly on the properties of the circuit itself, determining how fast the measured results become stable. The run-time in case of the used exemplary circuits ranges from about 1 hour in case of benchmark circuit "fpu100" up to several hours in case of benchmark circuits "vitdec". Clearly, the largest portion is coming from the fine approximation, the portion of the coarse approximation is usually negligible. The reason is that in case of the coarse approximation it has to be iterated only one time over the approximation candidates in the circuit and test their influence on the circuit output. In case of the fine approximation it has to be iterated several times over the approximation candidates. The more approximations the circuit tolerates the longer it takes to approximate the circuit as in this case the required iterations increase. However, as the approximation has to be performed only once per approximation point, one can likely live with high run-times, if the resulting approximation is accurate and resulting in significant power savings. The performance bottleneck is the speed of the FPGA, respectively its resources as mentioned earlier in this work. A reduction of the run-time could only be achieved by further parallelizing the emulations on the FPGA. The algorithms running on the host computer are very lightweight and could be even running on low-cost microprocessors. The results of the coarse and fine approximation of benchmark circuit "QR" are shown in Figure 4.16. The figure is showing the possible

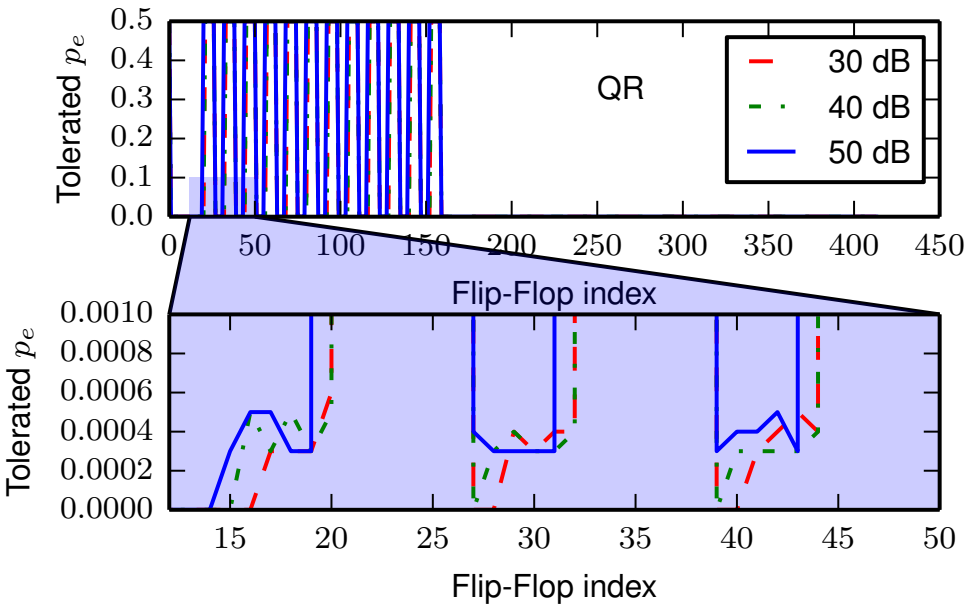


Figure 4.16: Approximation result for benchmark circuit *QR*, showing the maximum tolerable error probability at each register

approximation, hence error probabilities, at each register in the circuit for the three operating points. One can see, that most registers do not tolerate any approximation at all. An detailed analysis revealed that the registers related to the "Q" output could not

be approximated satisfactorily. While approximations in general were possible, all candidates have been removed when filtering for "high-variance" registers. For the "R" part related registers instead, many approximation were possible. Only for those registers the relation between applied approximations and the resulting output error probability has been direct and therefore predictable. This is an inevitable requirement for the approximation of integrated circuits as we have seen before. However, the approximations for the "R" related registers are not negligible. When looking back at Figure 4.10 we can see that when the signal quality increases from 40 dB to 50 dB, one additional bit of the "R" output can be ignored at application level. Now, one can see that this additional bit at the output spreads over multiple register within the circuit at register-transfer level. These bits, denoted by an error probability of $p_e = 0.5$ are those of the coarse approximation, now qualified for static or dynamic pruning. Also visible in the figure is the fine approximation, hence tolerated error probabilities less than 0.5. One can see, that if the channel quality increases more fine approximations can be tolerated, even if the coarse approximation does not change. The approximation results of benchmark circuit "fpu100" are shown in Figure 4.17. The Sobel application, embedding the FPU, is using the division, the multiplication and the square root unit of the FPU. These three blocks have been independently analyzed. Compared to the "QR" circuit, the "fpu100" is synthesized for an ASIC target using Synopsys 90 nm technology. The approximation algorithms presented were able to find a combination of possible error probabilities for each approximated operating point. The better the desired image quality has to be, the less approximations are possible. Table 4.3 is summarizing the two figures. In the table the sum of tolerated error probabilities over all registers is shown, serving as a reference for the tolerated approximations. It can be seen, that the difference between the operating points is much less than it seems to appear in the figures. This will result only in a small difference in the power savings possible between the operating points as we will see later. Nevertheless, even if the difference between the operating points is small, the initial approximations, possible for all three operating points are huge. Hence, the algorithms presented have also been able in this case to identify those elements in a circuit that are needed and those that are not needed for one particular application for specified quality constraints. Figure 4.18 is showing the results of a applying the ap-

Table 4.3: Possible Approximation for benchmark circuit *QR* and *fpu100* for different quality goals

QR		fpu100	
Signal Qual. [SNR]	$\sum p_{e,i,\max}$	Target Qual. [PSNR]	$\sum p_{e,i,\max}$
30 dB	36.5183	50 dB	214.5132
40 dB	36.5190	40 dB	207.0054
50 dB	42.5191	30 dB	187.0082

proximation algorithms to the benchmark circuit "vitdec". Due to the size of the circuit, the figure uses another format to visualize the results. Instead of showing the tolerable error rates of each register individually, in this figure, the sum of tolerated error prob-

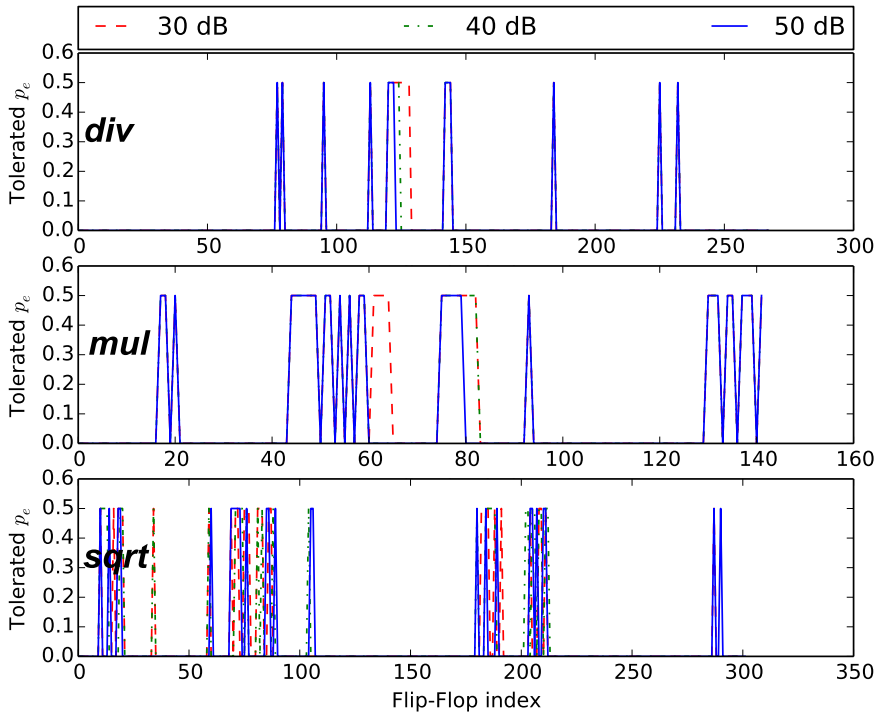


Figure 4.17: Approximation result for benchmark circuit "fpu100"

abilities is shown, depending on the signal-to-noise ration of the wireless channel. The coarse-grained approximation was not able to detect any registers that can be pruned. Hence, no circuit blocks can be switched off and on depending on the channel quality. The large base-level of $\sum p_e = 245$ is coming from the fact that the test application is not triggering all registers (traceback length, etc.), and therefore should not be taken into account. Nevertheless, in case of the fine-grained approximation, the algorithm was again able to find approximation combinations for the registers and for all channel signal-to-noise ratios. Not further surprisingly, the better the channel quality becomes, the more approximations can be tolerated inside the Viterbi decoder. The results of the three benchmark circuits showed that the algorithms presented in Chapter 4.4 and 4.5 perform well. The algorithms were able in any case to find a combination of approximations at the circuit registers, so that at the same time the constraints on the output error probability are met. Furthermore, the degree of approximation of an approximated working point is increasing always with decreasing requirements on the quality. Even though no statements can be made regarding the optimality of the approximations, this shows that the algorithms are working. The separation into coarse and fine grained approximation is clearly helping to speed up the approximation, as the pruning candidates can be sorted out early, reducing the search space for the fine approximation. The same applies for the preparation steps, namely data-path separation and high-variance register exclusion. If this reduction of the search space would not have happened in advance, the actual

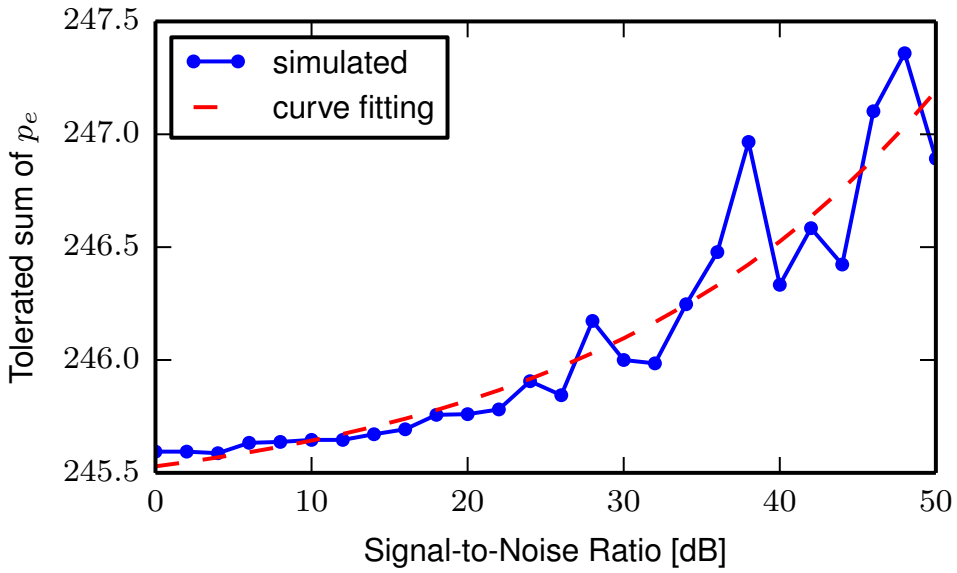


Figure 4.18: Possible approximations in terms of tolerated error probabilities, for benchmark circuit *viterbi* for varying signal qualities and a target BER=0.0

approximation steps would not only be very time consuming but also very unreliable. Time consuming, as the search space would correspond to the number of registers in the circuit, and unreliable, as the results would vary from emulation run to emulation run and no stable approximation configuration could be found. Nevertheless, there is still a high potential for optimizing the approximation, especially regarding the optimality of the results. Most important, by taking the knowledge of the resulting power savings into account, the approximation could be optimized regarding the resulting power consumption of the circuit.

Power Evaluation due to Circuit Pruning In the following the power savings due to "circuit pruning" are evaluated, hence the functional approximations based on the coarse approximation. In order to evaluate the power savings due the fine approximation one has to go down yet another step in the hierarchy. In this work the results of the fine approximation will be used to apply voltage over-scaling as the approximation technique. The necessary steps in order to perform this approximation technique will be presented in the next chapter. The power savings of approximating the "vitdec" benchmark circuit can therefore not be determined, as only fine approximations could be determined. Figure 4.19 is showing the estimated dynamic power consumption of benchmark circuit "QR" when applying the three different approximate operating points determined using the presented methodology. The circuit has been synthesized for Virtex 6 FPGA architecture. The power estimations have been made using Xilinx XPower Analyzer. In order to generate as accurate as possible estimations, realistic test pattern have been used to

generate the switching activity information. The clock frequency has been set to 100 MHz. One can see that, when applying no approximation at all, about 18.2 mW of

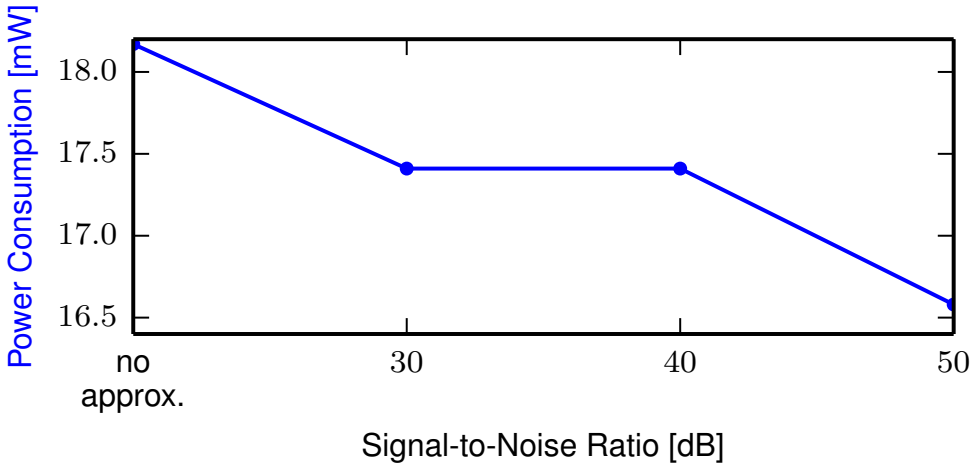


Figure 4.19: Dynamic power consumption of benchmark circuit *QR* for different approximations based on varying signal qualities, when performing 8x8 ZF Equalization [143]

power is consumed when running the MIMO application presented earlier in this work. When the channel signal-to-noise ratio is 30 dB, approximate pruning can be applied that reduces the power consumption to about 17.4 nW. The resulting bit-error rate is then guaranteed to be below BER ≤ 0.01. One can see that an increase of the channel SNR, does not lead to any significant power savings in the circuit. This is not further surprising, as we have seen in Figure 4.10 that an increase from 30 to 40 dB SNR does not lead to a decreasing precision requirement of the "R" output. A further noise reduction from 40 to 50 dB leads to another significant reduction of the power consumption. When the channel SNR is at 50 dB, the QR decomposition consumes only 16.6 mW. Hence, compared to no approximation 1.6 mW can be saved, which corresponds to an decrease of about 8.7%. Figure 4.20 is showing the power consumption when applying approximate circuit pruning to the benchmark circuit "fpu100" (multiplication) being part of a Sobel filter application. The circuit is synthesized for Synopsys 90nm ASIC technology, running at 100 MHz. The power estimation is performed using Synopsys "Design Compiler". In order to estimate the switching activity factor, the circuit is simulated in Synopsys "VCS" using realistic stimuli. One can see that depending on the desired quality of the filtered image and different approximate operating points, the power consumption is changing. The figure is not only showing the power consumption when approximating using static pruning but also using clock gating. One can see that in this example the difference between static pruning and clock gating is only barely visible. Nevertheless, as expected, the power saving of circuit pruning is larger compared to clock gating, due to the remaining static power. Compared to the deterministic variant of the circuit, up to 1.79 mW can be saved, which corresponds to an decrease of 43.3%. This is clearly

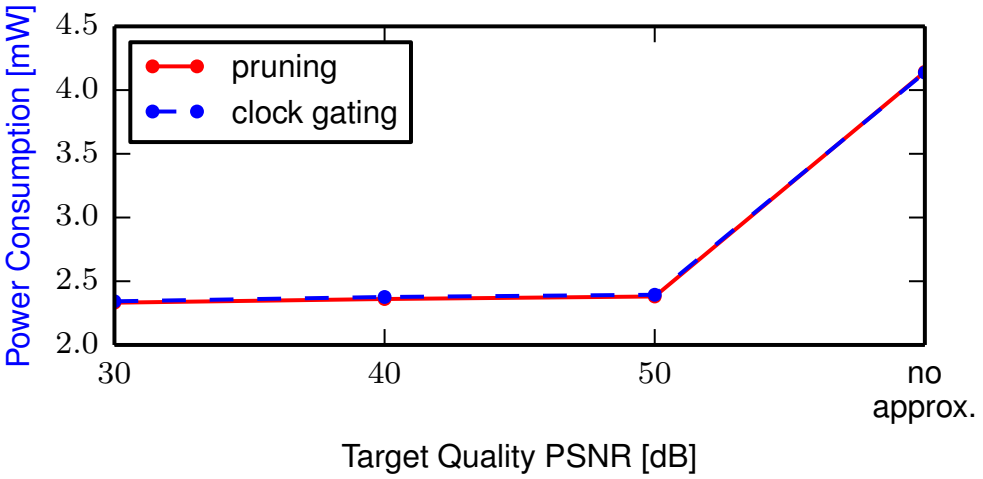


Figure 4.20: Power consumption of benchmark circuit *fpu100* (multiplication) for different approximations based on varying target qualities, when performing a sobel filter

an impressive value. When looking back again to Table 4.3 one can see that the sum of tolerated error probabilities is very high. The approximation algorithm where able to find many points suitable for approximation. This metric serves as a first indication of the potential power savings due to the approximation. When accepting a degradation of the image quality from 50 db to 30 dB, leads to a power saving of 0.05 mW (−2.1%). One can see that the power savings due to dynamically switching between approximate operating points in this example are small, compared to the ones due to the initial approximation. Nevertheless, the dynamic approximation is working in general, the possible savings are however depending on the circuit. In case of benchmark circuit "QR", significant energy can be saved when adapting dynamically to the environment.

4.7 Summary

In this chapter a methodology has been presented, allowing to approximate any integrated circuit. Compared to previous works, the approximation is performed at register-transfer level, and is not limited to using approximated algorithmic processing units. The methodology is automated wherever possible. However, manual interaction is not avoidable. The key to a trustworthy approximated circuit is to perform the approximation with realistic test-pattern. With "application-reasoned" approximation, an approach is introduced bringing down the information about quality from the application level to the actual place of approximation, the integrated circuit. In this work the approximation is performed at register-transfer level. In a later step this information can then be used to apply more fine-granular approximation methods to the circuit. The introduction of the notion of error probabilities as the notion of quality at the register-transfer level, allows to model a variety of approximation techniques and makes the approach as generic as possible. The FPGA-based fault emulation is used to analyze the effect of approximation

on the circuit outputs. These measured error probabilities can then be compared with the previously determined minimum quality. The proposed approximation of a circuit consists of two preparation steps and two approximation steps. The preparation steps are required to identify those registers in the circuit that are suitable for being approximated at all. Many registers in the circuit are responsible for the control flow of the application and usually cannot be approximated. The automated elimination of unsuited candidates is performed in the steps "data-path separation" and "high-variance register exclusion". Once the approximation candidates are identified, the actual approximation can be performed. The goal of the approximation is to find a distribution of error probabilities assigned to the registers of the circuit, as high as possible, while not violating the constraint on the maximum output error probability. The proposed method consists of two step, one being the coarse, and one being the fine-grained approximation. In the coarse grained approximation, all registers are identified that could be (either dynamically or statically) removed from the circuit, as their influence on the quality of the circuit output is negligible for the application. If "circuit pruning" is the only approximation technique to be applied, one can stop the analysis after this step. The information gained in this step can be directly used to remove unused circuit elements or insert clock-gating means into the circuit. More fine-granular approximation techniques, need one additional approximation step. The fine-grained approximation identifies, for those registers that cannot be completely removed, the degree of imprecision that may be tolerated there. For instance, when applying voltage over-scaling to a block of combinational logic, the bits at the register inputs of the next stage will be wrong with a certain error probability due to the timing violations, induced by over-scaling the supply voltage. The error-rate that could be tolerated will be determined in the fine-grained approximation. At the end of the approximation, for each register in the circuit the maximum tolerable error probability that may be observed at the register input is known. An error probability of $p_e = 0.5$ denotes that the register (and all fanin connected only to this wire) can be used for "circuit pruning". An error probability $p_e < 0.5$ instead denotes that this register input cannot be completely ignored but anyhow operated imprecise by some other means of approximation. Three exemplary benchmark circuits have been approximated using realistic application test-pattern. For each application a variety of approximate operating points have been defined, based on realistic application scenarios. For instance benchmark circuit "QR" being part of a MIMO receiver, is operated with varying precision based on the signal-to-noise ratio of the wireless channel. The presented approximation algorithms were able to find approximations at register-level for each of these operating points. The less precision the operating points requested, the more approximations were possible. It pointed out that the separation of the whole approximation into, in total, four step reduces the complexity of the final fine-grained approximation so that it can be performed in a reasonable amount of time. Applying "circuit pruning" to two of the benchmark circuit confirmed the functionality of the determined approximations. At any time in the simulation the error rates at the circuit outputs did not exceed the constraints. And at the same time, a significant amount of energy could be saved due to the approximations. The power savings due to dynamically switching between operating points for these benchmark circuits unfortunately pointed out to be small. The power savings com-

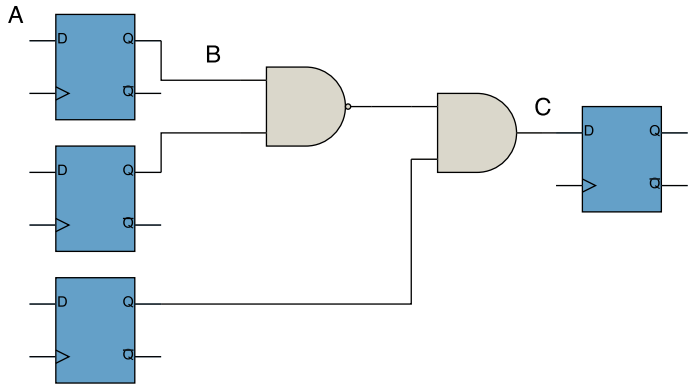
pared to the deterministic version however are significant. In case of benchmark circuit "fpu100" up to 43% of the power consumption could be saved. In case of benchmark circuit "QR" up to 8.7% could be saved. This shows that the proposed algorithms do work well. And there is even a potential for further improving the algorithms. By taking the information about power saving into account when approximating the circuits, the results could be optimized. However, for this work the algorithms satisfied to confirm the approximation methodology proposed. Modeling approximations as bit-flips at registers, hence giving them an error probability, can be used to practically approximate integrated circuits. The FPGA-based fault emulation system, used as a tool to speed up the analysis of the circuits did prove itself. Even though FPGA-based emulation requires more manual interaction, for instance compared to software-based solutions, it speeds-up the approximations a lot. In the next chapter it will be shown how the results of the fine-grained approximation can now be further used to apply gate-level approximation techniques like voltage-over scaling.

5 Approximation at Gate Level for Voltage Scaling

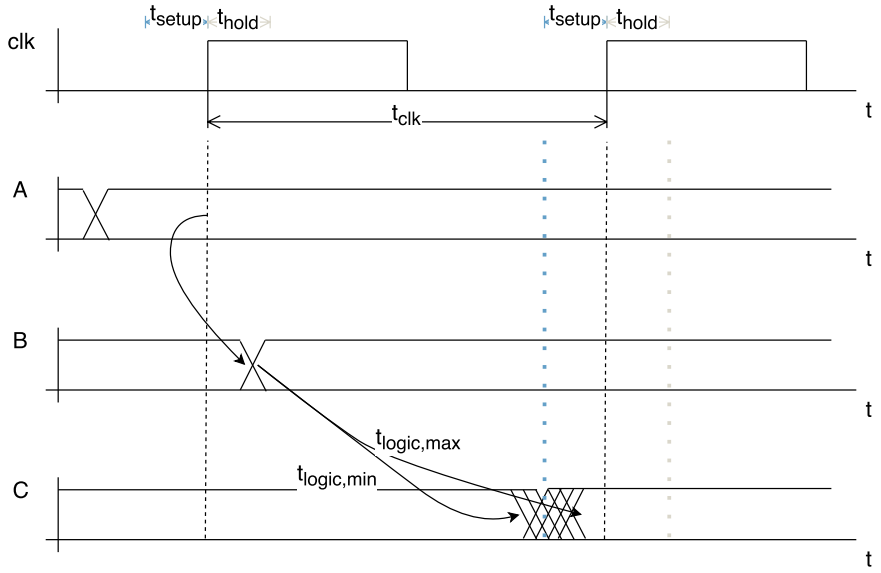
In the previous chapter, it has been shown how the maximum error probability at the register inputs of sequential circuits, that can be tolerated for an approximate operating point, can be determined. Furthermore, it has been shown how "functional approximation" can be applied in order to approximate a circuit by removing elements whose influence on the circuit quality is negligible. In this chapter it will be shown how the remaining parts of the circuit, those that cannot be completely removed, can be approximated in order to save additional power. There are several fine-granular approximation techniques known in the literature as it has been shown in Chapter 2. In this work the focus has been put on "voltage over-scaling", as it promises high power savings due to its quadratic relation to the power consumption, as it has been already shown in Equation 2.3. Voltage over-scaling promises a very precise tuning of approximations, power consumption and the resulting error probabilities, as supply voltages can be adjusted very fine-granular. Clearly, in practice it is not possible to assign different supply voltages to each gate. This limitation will be discussed in Section 5.4. Putting voltage over-scaling into practice requires a very detailed analysis of the circuit behavior as it will be shown. Especially as it is one of the main goals of this work to develop generic methods for generic circuits, existing approaches for voltage over-scaling cannot be used.

Unfortunately, applying voltage over-scaling does not only decrease the power consumption of a circuit. With decreasing supply voltage V_{dd} the propagation delay t_p increases, as it has been shown in Equation 2.4. That means, that the time it takes to propagate a signal edge from one input of a gate to the output increases. The circuit is becoming slower. That, in turn, means that during the design phase of a circuit, when performing the timing analysis, the determined maximum clock frequency of the sequential logic will decrease. The circuit has to be operated with a lower clock frequency, the lower the supply voltage has been chosen, so that no timing violations arise.

Simplified, a timing violation happens if the signal edge propagation does not arrive before the setup time t_{setup} of the timing endpoint, the following flip-flop. A simple example showing the problem of timing violations is shown in Figure 5.1a and 5.1b. Depending on which logic path is observed and the assigned supply voltage, the propagation delay of the path varies. In reality other factors do come into play, like clock jitter or temperature dependency. In other words, a timing violation means that the new value of a signal does not arrive in time at the endpoint and the old value of the signal remains active. If the old logic value however is the same as the new one, a timing violation does not result in an error at the register. Assuming, that for an application one approximate operating point has been determined, i.e. the maximum error probabilities that can be



(a) Simple exemplary sequential circuit



(b) Timing violations of simple exemplary sequential circuit

tolerated for each register are known. In order to apply voltage over-scaling, several questions have to be answered in advance. First of all, one has to determine which timing paths fail at which supply voltage V_{dd}. And second, if a timing path fails, one has to know the resulting impact on the timing endpoint, hence the resulting error probability.

For the former question it is necessary to be able to understand the relation between supply voltage and propagation delay. For this, the technology parameters have to be known of all elements compromising a timing path. Even though a circuit can consist of billions of timing paths, this seems to be a manageable task with today's CAD tools. In case of the latter question, the situation is not as easy as it will be shown later. Due to the fact that the theoretical approximations at register-transfer level have already been made, a divide and conquer approach can be used. As the tolerable error rates at the timing endpoints are already known, one can analyze the "fanin" of each endpoint separately. Hence, it is only necessary to scale the voltage of a small subset of the circuit

at once, which reduces the complexity a lot. Furthermore, this allows to parallelize the computations. In the end, however, the individual scaling results have to be carefully joined, which will be also explained later in this work.

In previous works, as it has been already summarized in Chapter 2, mainly manual methods were presented. Most works presenting practical examples for voltage over-scaling, performed the approximation by simply scaling the supply voltage as a whole, simulating the circuit with updated timing information and observing the effects. Based on inspection, some of the works then introduced manual optimization of the circuit in order to optimize the timing behavior. This is clearly a valid approach, however in this work the process is tried to be automated wherever possible. Approaches that propose to introduce multiple voltage domains, to approximate the circuit only where possible, are rare. Most approaches use Monte-Carlo simulations to find a distribution of supply voltages that fits the demands on the resulting error rate. This again requires lengthy simulations to determine the behavior of voltage scaled operating points. Only very few approaches exist that propose an analytic approach to determine the resulting error rates if the timing constraints are violated, but these are also limited to specific circuits. In this work an analytic, generic approach is presented in order to determine the minimal supply voltage for each gate in the circuit for an approximate operating point. An overview of the methodology is shown in Figure 5.2. One can see that the flow consists of basically two loops. The outer one is executed for each endpoint and the inner one is then scaling the voltage for each of these endpoints. The outer loop can easily be parallelized, as the dependencies between the endpoints do not have to be solved immediately. Requirements for the algorithm are apart from the circuit itself, the tolerable error rates at the circuit outputs as determined in the previous chapter. The result of the algorithm is a list, denoting the minimum supply voltage for each gate in the circuit.

5.1 Determination of Failing Timing Paths

As already mentioned, in order to divide the problem of determining the effects of voltage over-scaling into manageable peaces, each timing endpoint is analyzed separately. Hence, the group of elements whose behavior has to be analyzed at once is the “fanin” of the endpoint up to the next register stage. Figure 5.3 is showing the schematic of benchmark circuit “c17”. Marked in orange is the fanin of one timing endpoint. Each of these domains is analyzed separately, as the maximum error probability that can be accepted at the endpoint is already known. One can see that the fanin consists of several paths, each having an influence on the Boolean value at the flip-flop input. Hence, each of these paths, if failing, can falsify the correct value.

A timing path is defined as the connection between two flip-flops with only combinational logic in between. A timing path is furthermore defined by the type of transition at the input, a rising or falling edge. Additionally, the transitions at all inputs of all combinational gates compromising the path uniquely define the path. One can see that due to this variety of transitions the fanin of an endpoint consists of many timing paths. Even the connection between one startpoint and one endpoint defines multiple timing

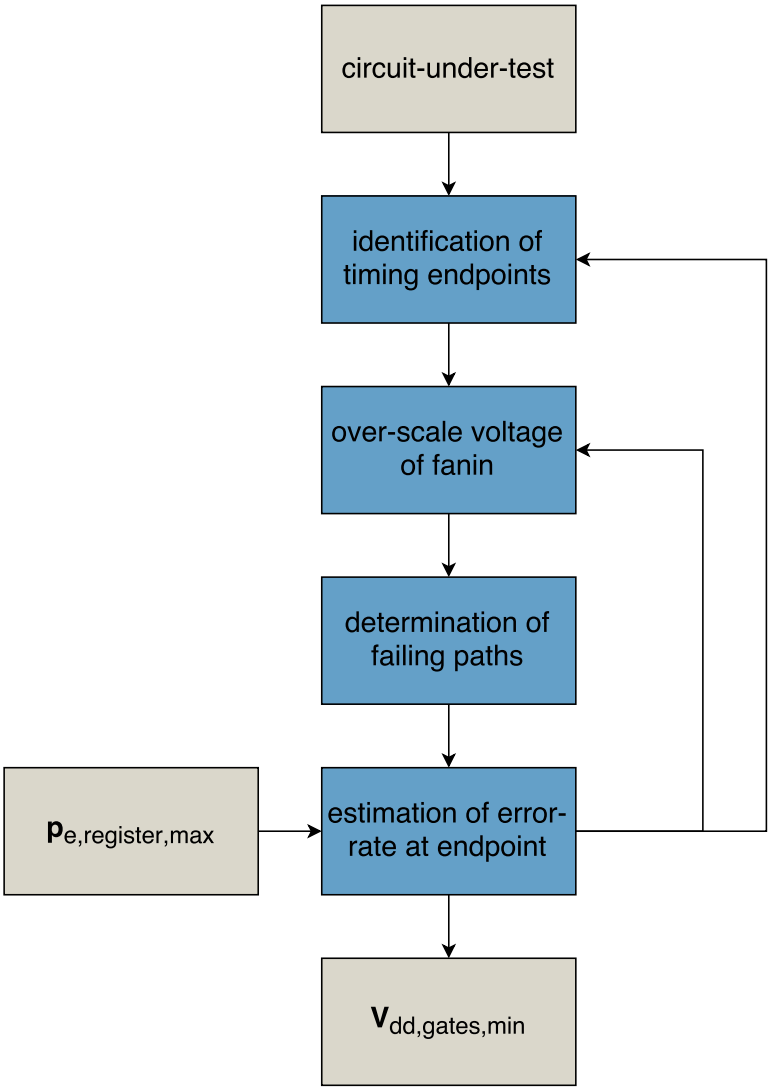


Figure 5.2: Overview of the voltage over-scaling methodology presented in this work

paths. The more gates are between a start and an endpoint, the more timing path exist, as the number of combinations grows rapidly. This has to be kept in mind when later calculating the resulting error rate due to timing violations. At first one has to find out, which timing paths fail at which over-scaled supply voltage. In order to do so, the knowledge about the propagation delay of the gates of the used technology is required. Usually the propagation delay is given only for the nominal voltage of the technology. Sometimes also the technology parameters of a low-voltage and a high-voltage operating point are given. Hence, the behavior of the paths for other supply voltages has to be determined by SPICE simulations at transistor level. This allows to simulate the propagation delay of the fanin for several supply voltages, even outside of the specifications. This approach

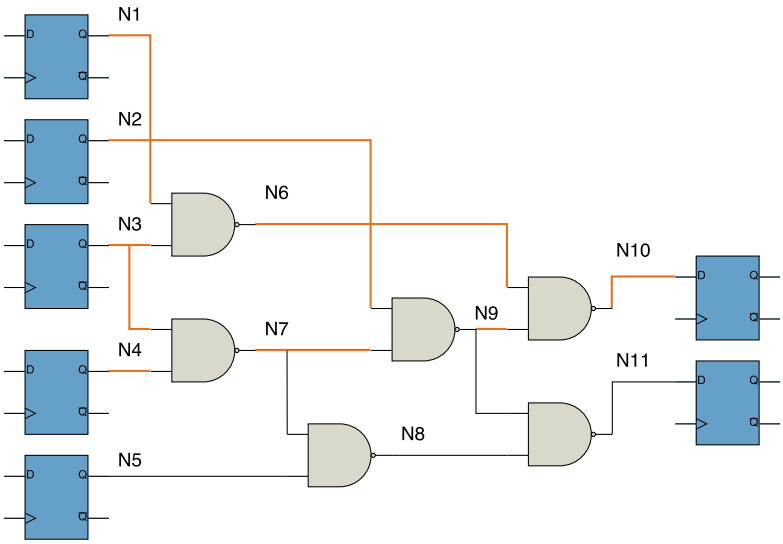


Figure 5.3: Schematic of benchmark circuit "c17". Marked in orange the fanin of the one output register

works well and delivers very accurate results [132, 139]. Care has to be taken that wire load models have to be taken into account. Furthermore, one must not forget to consider all load capacitance of the gates, when analyzing the fanin of an endpoint separately from the rest of the circuit. When performing this approach it is necessary to extract the fanin of an endpoint and convert it to a SPICE netlist. This can be achieved using the TCL interface of Synopsys' "Design Compiler" and some additional scripts. The main limitation of this approach is the speed of the SPICE simulations. One has to keep in mind that realistic circuits consist of billions of endpoints. Furthermore, the fanin of an usual endpoint is not as small as the one in the example circuit "c17". Simulating all these fanin groups for a variety of supply voltages requires a significant amount of time.

Fortunately, yet another approach can be used to determine the failing timing paths for an over-scaled supply voltage. The "Composite Current Source" model allows to interpolate between the operating points of technology libraries with very good accuracy [127]. Synopsys uses this technology for instance in their timing analysis tool "PrimeTime". PrimeTime can be used to assign supply voltages to each gate in the circuit and perform a timing analysis using the well-established models for noise and parameter variation. This simplifies the determination of failing timing paths a lot.

At first, all endpoints, i.e. all registers have to be identified in the circuits. This can easily be done using the TCL interface. The same scripts can be used as when using "Design Compiler". Then for each endpoint the fanin is identified. As already mentioned, only the fanin up to the next register stage has to be considered. If there is no previous register stage, the fanin ends at the circuit input pins. After this step all elements compromising the fanin of an endpoint are known and the supply voltage of these elements can be scaled. Scaling is simply done by assigning a new (scaled) supply-voltage to each gate of the fanin and perform the timing analysis. The timing analysis is not slower

than with nominal voltages. The timing analysis is now automatically done by interpolating the various parameters of the technology library for the assigned supply voltage. The timing results can then be analyzed, again using the TCL interface, in order to detect which paths fail at which supply voltages. By repeating this step for an interval of voltages, one can determine which timing paths fail at which supply voltage. This operation can now be performed for all timing endpoints and their fanin. The benefit of this approach compared to the SPICE based analysis is clearly the speed. The timing analyis in PrimeTime is several magnitudes faster than the SPICE simulation. Additionally, PrimeTime utilizes well-established models for estimating parameter variations, thermal noise models, temperature variations and wire load models. In order to generate trustworthy results in SPICE, these parameters would have to be modeled as well.

Figure 5.4 is showing a plot of the sum of failing timing paths for three benchmark circuits versus the supply voltage. The results have been determined using Synopsys PrimeTime. Benchmark circuit "simple circuit" is the circuit shown in Figure 5.1a, consisting of 3 inputs, one NAND2 gate connected to one AND2 gate. Benchmark circuit "c17" is shown in Figure 5.3. It has 4 inputs and 2 outputs connected by 6 NAND2 gates. Benchmark circuit "c432" has 36 inputs and 7 outputs connected by 160 logic gates. One

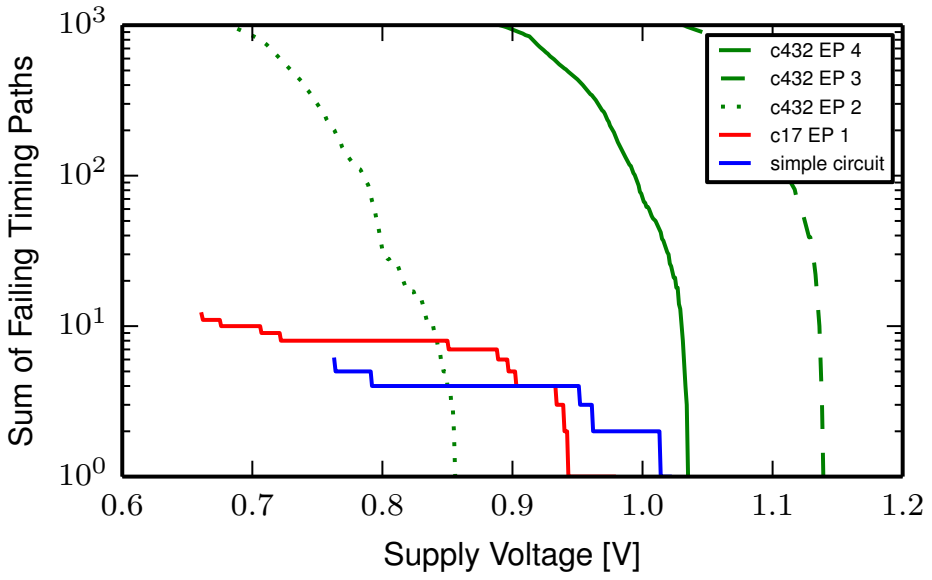


Figure 5.4: Sum of failing paths (selected endpoints EP) depending on supply voltage for benchmark circuits: "simple","c17" and "c432" [145]

can see that for large, realistic sized circuits, the number of failing paths is increasing very fast. As it has been already mentioned, the reason for this is the rapidly increasing number of timing paths per endpoint due to the increasing number of transition combinations. Even though an endpoint with a large fanin consists of many timing paths, it does not mean that the resulting error rate due to timing violations is increasing equally

fast. As we will see in the next section, the proportion of one path to the error rate at the endpoint is decreasing with the growing number of timing paths.

The process of determining which timing paths fail at which supply voltage, for each endpoint in a circuit has been completely automated using the TCL interface of Synopsys PrimeTime. When performing this step a list is generated containing all information required for the next step, namely estimating the resulting error probability at a timing endpoint, when timing paths fail.

5.2 Estimation of Error Probability at Timing Endpoint

The estimation of error rates due to timing violations is much more complex than the identification of failing timing paths. No commercial tools exist that can be utilized. The most straightforward method would be to use the timing information that has been interpolated using the CCS model, and simulate the circuit using regular simulation tools with the updated timing annotation. This approach is working well. It serves in this work as a reference for the analytic method developed, in order to compare the results. In order to perform the simulation two steps are required. First the timing information of the gates has to be updated. That means, that the propagation delay of the gates that the simulation tool (e.g. ModelSim) would usually use, have to be overwritten. This is done by loading a standard delay format (SDF), containing the updated timing information of the technology components at a scaled supply voltage. This SDF file can be generated using PrimeTime after assigning supply voltages to the gates and performing the timing analysis. And second, the simulation needs again accurate test pattern, in order to simulate the circuit in a realistic environment. This is actually the limitation of the approach. The simulation of large circuits can be very time consuming. The simulation would have to be executed several times for each endpoint. And in order to calculate reliable error-rates the circuit would have to be simulated for long time in order to simulate a decent amount of clock cycles. This would result in very long simulation times, making the approach de-facto unusable. Therefore, an analytic estimation of the resulting error-rates has been developed.

In order to develop an analytic framework to determine the resulting error rates caused by supply voltage scaling, one first has to understand the fundamentals of the relation between timing violations and resulting bit-errors. At first one timing path is analyzed isolated. The question to be answered is, when does a timing violation at one path results in a bit error at the timing endpoint. The relation has been identified to be the following:

> *A timing violation is observable at the input of a register i only if there was a transition of a path j, rising or falling, and if that transition has not been masked by any of the combinational gates k on the path* [145].

A timing violation does only result in a bit error, if the transition at the timing startpoint would have been propagated to the endpoint, but it is not arriving in time there. Hence, the resulting error probability at the endpoint i can be calculated as the product of probabilities that the initial transition (edge) has not been masked by any of the gates

k. The relation is expressed in the following Equation 5.1:

$$p_{e_{i,j}} = p_{\text{start}_{01}} \cdot ||\mathbf{p}_{\text{trans}_{i,0}} \otimes \mathbf{p}_{\text{trans}_{i,1}} \otimes \ldots \otimes \mathbf{p}_{\text{trans}_{i,N_j-1}}||_1, \tag{5.1}$$

where $p_{\text{start}_{01}}$ corresponds to the probability that a transition at the timing startpoint is happening and $\mathbf{p}_{\text{trans}_{i,k}}$ corresponds to the probabilities of the k-th gate in timing path i that a given transition at the input pin of a gate used by the timing path is not masked at the output of the gate. As there are usually several possibilities, this is a vector. The operator $||\cdot||_1$ is defined es $||x||_1 := \sum_{i=1}^{n} |x_i|$. N_j corresponds to the number of gates in the path. The Kronecker product $\otimes$ is required to cover all different possibilities. The more violated timing paths are "active", i.e. not masked by any logic gate, the more likely is an error at the timing endpoint. The error probability observed at the input of the timing endpoint for a certain supply voltage $p_{e_i}(V)$, hence can be expressed as:

$$p_{e_i}(V) = \sum_{j=0}^{j=M_{i,V}-1} p_{e_{i,j}}, \tag{5.2}$$

where $M_{i,V}$ corresponds to the number of timing paths that fail at a certain supply voltage V. This list of failing endpoints can be determined as described in the previous section.

The first problem to be solved is the determination of $\mathbf{p}_{\text{trans}_{i,k}}$, the "transition probabilities", i.e. the probabilities of a transition not being masked at a logic gate. This probability is clearly depending on the logic values of the nets at the inputs of the gate and the function of the gate itself. The various possibilities that an input transition is transferred to the output of gate depending on the logic values of all inputs are called "transition possibilities" in this work. For instance, assuming that one input of a NAND2 gate is falling (**f**), the transition is visible at the output of the gate in only two cases. It is visible if the second input is falling (**f**) as well or if the second input was a Boolean "1" and remains a "1" (**1**). Only in these two cases a transition is visible at the output of the gate, which is in this case rising (**r**). These transition possibilities have to be determined for all gates used in a circuit. They can however be determined in advanced and stored in a table. The transition possibilities of a NAND2 gate are shown in Table 5.1. Relevant are those entries where an edge is visible at the output of the gates. In case of a NAND2 gate these are six combinations. In case of a NAND4 gate these are already 110 combinations. An AO222 gate, a gate with six inputs, has 1998 transition possibilities. Note, that each of these combinations forms a separate timing path. With multiple gates in a path, the possible combinations grow exponentially. The probability of one gate not masking a transition at the input can now be expressed as the sum of the transition probabilities. Hence, for the NAND2 gate with a falling edge at one input, $\mathbf{p}_{\text{trans}_{i,k}}$ can be calculated as:

$$||\mathbf{p}_{\text{trans}_{i,k}}||_1 = p_{\text{IN2}_{\{x=1\}}} \cdot p_{\text{IN2}_{\{x=0\}}} + p_{\text{IN2}_{\{x=1\}}} \cdot p_{\text{IN2}_{\{x=1\}}} \tag{5.3}$$

,where $p_{\text{IN2}_{\{x=0\}}}$ corresponds to the probability that the logic value at input "IN2" is "0" and $p_{\text{IN2}_{\{x=1\}}}$ to the probability that it is "1". In a simple example where a timing paths

IN1	IN2	OUT
r	r	f
r	f	1
r	0	1
r	1	f
f	r	1
f	f	r
f	0	1
f	1	r
0	r	1
0	f	1
0	0	1
0	1	1
1	r	f
1	f	r
1	0	1
1	1	0

Table 5.1: Transition possibilities of a NAND2 gate. Highlighted are the combinations that transfer an edge

compromises only one NAND2 gate, the resulting error probability could be expressed as:

$$p_e = p_{\mathrm{IN1}_{\{x=1\}}} \cdot p_{\mathrm{IN1}_{\{x=0\}}} \cdot ||\mathbf{p}_{\mathrm{trans}_{i,k}}||_1 \tag{5.4}$$

One can already see that for timing paths consisting of multiple gates, the equation is becoming very complex.

In the next step, the probabilities of a signal being a logical "1" or "0" have to estimated. Simplified, the probability could be assumed to be 0.5. However, this is very inaccurate and leads to wrong probability estimations. One way to determine the "1/0" probabilities is to estimate them. Methods are proposed in the literature allowing an accurate estimation [128]. However, these methods are limited to combinational logic, which is why another approach is used in this work. As a simulation model is anyhow required, in particular accurate test pattern for the approximation, one can simply use the well established simulation tools in order to determine the "1/0" probability of each net in the circuit. Once again, the quality of the test pattern is very important. Only with realistic test pattern, i.e. as similar as possible to the infield environment, the estimated "1/0" probabilities will be close to the reality. The switching activity and the "1/0" activity of the nets of the circuit can be determined by running a normal simulation using the regular simulation tools like "Modelsim" or "VCS". The only difference is that resulting waveforms have to be written out by the simulation tool. These "vcd" or "saif" files can then be imported in Synopsys PrimeTime, which in turn calculates the required estimations for each of the nets in the circuit. Now that the "1/0" probabilities are know, the masking probabilities shown in Equation 5.3 can be calculated, and based on that

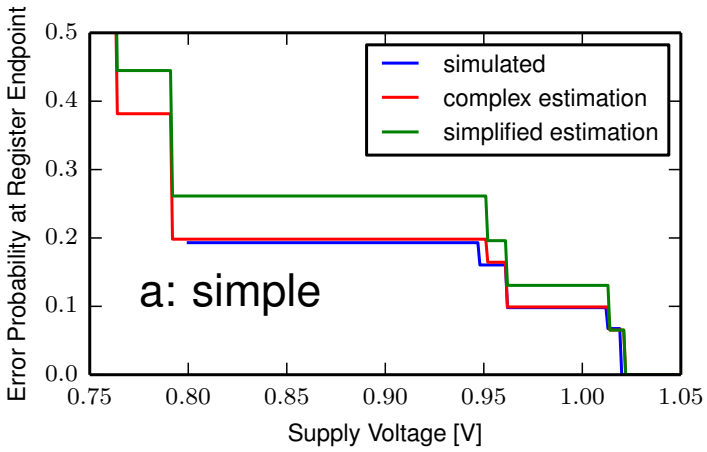
the resulting error probability at the timing endpoint shown in Equation 5.1.

Unfortunately, some pitfalls have to be considered. Foremost, it is important not to count the same timing path twice to the resulting error probability. This might sound unlikely at the first place, but can happen very easily. For instance when looking again at the schematic of benchmark circuit "c17" in Figure 5.3 the situation can be seen easily. One can see that the timing endpoint connected to net "N10" can be reached by several paths. It can be for instance reached from "N1" as well as from "N3". Note, that both of these paths compromise exactly the same components and nets. One possible transition possibility would be for instance: N1 **rising**, N3 **rising** and N9 **rising**. This would be a valid combination when starting at N1 as well as when starting at N3. In case of a timing violation, the impact on the resulting error probability must not be considered twice. This seems to be very simple in this simple example, but is becoming a very complex task for large fanins. In order not to count the same path twice, one has to memorize the "consumed" combinations. Storing all consumed combinations and search for duplicates in it is a time an memory consuming task, as the complexity is growing exponentially with the size of the fanin as we have seen already. Assuming a simple example circuit with one endpoint being influenced by 100 timing paths, and each of these paths consists in average of 20 NAND2 gates. As it has been shown in Table 5.1 for a NAND2 gate two possibilities exist for a transition at one pin, given a transition at the other pin. Hence for this small example 2^{20} masking combinations are possible. Hence for an endpoint consisting of 100 timing paths, roughly 100 million combinations have to be saved and searched for duplicates. This makes it practically impossible to perform this step with full accuracy. In future work one could try to store only hashes of the paths which would at least reduce the required memory. However, for this work, the removal of duplicate paths is not performed for large circuits due to the exploding computation time and memory requirements. Fortunately, the larger the fanin of an endpoint is, i.e. the more timing paths influence it, the less influence does a single path have. By not removing duplicate paths, the equation of calculating the resulting error probability also simplifies a lot. Instead of having to calculate all possible combinations using the Kronecker product as shown in Equation 5.1 one can now directly sum up the masking probabilities of the gates and then calculate the product over the whole path. The simplified equation is shown in Equation 5.5.

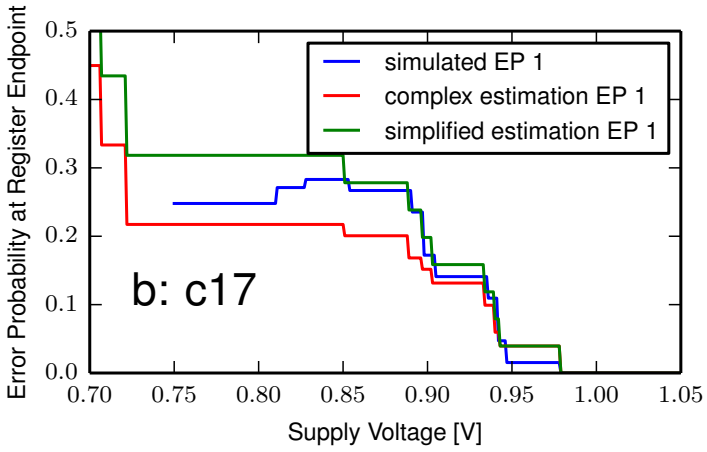
$$p_{e_{i,j}} = p_{\text{start}_{01}} \cdot \left(\prod_{k=0}^{k=N_j-1} ||\mathbf{p}_{\text{trans}_{i,k}}||_1 \right), \tag{5.5}$$

Figures 5.5a, 5.5b and 5.5c are showing the resulting error probabilities at the timing endpoints for the selected benchmark circuits, determined using the presented method. The "simulated" error probabilities are simulated using the simulation with annotated timing information as presented earlier. The "complex estimation" refers to the analytic estimation using Equation 5.1. The "simplified estimation" evaluates the simplified variant shown in Equation 5.5, where duplicated paths are not removed.

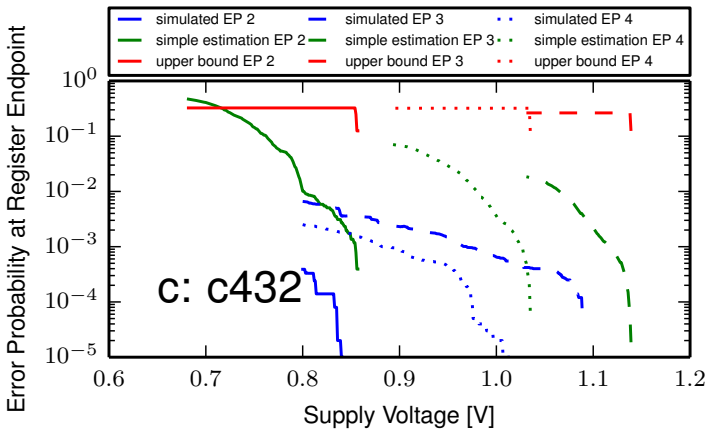
The "upper bound" shown in Figure 5.5c is showing the upper bound of the resulting error probability at the endpoint. The error probability can never be larger than any



(a) Estimated and simulated error probabilities at register endpoints due to voltage over-scaling for benchmark circuit: "simple" [145]



(b) Estimated and simulated error probabilities at register endpoints due to voltage over-scaling for benchmark circuit: "c17" [145]



(c) Estimated and simulated error probabilities at register endpoints due to voltage over-scaling for benchmark circuit: "c432" [145]

switching activity α at any of the startpoints of the failing paths. Furthermore, it cannot be larger than the measured switching activity at the input net at the timing endpoint. The "upper bound" therefore corresponds to $max(\alpha_{\text{startpoint}}, \alpha_{\text{endpoint}})$. The complex estimation could not be performed for circuit "c432" in Figure 5.5c due to the complexity of the algorithm. In case of the "simple circuit" in Figure 5.5a the complex estimation matches closely the simulated results. The simplified estimation lies a bit above the reference results due to duplicate paths not being filtered out.

In case of circuit "c17" shown in Figure 5.5b one can see that the complex estimation diverges from the simulated result with an increasing number of failing paths. Unfortunately, besides the problem of duplicate paths, there is another problem with the presented approach. It has been shown that the calculation of the error probability is mainly depending on the probability of the compromised signals being a logical "1" or a "0". The estimation of these probabilities is done by simulating the circuit and calculating the average value. The average value can sometimes be far off from the real one for a given combination of transition possibilities. When looking back at the schematic of benchmark circuit "c17" in Figure 5.3 the situation can be visualized. Assuming, that the current transition of N3 = **rising**, and for N4 = **falling** one can see that the signal probability of net N7 is changing. The only possible value for signal N7 is this situation is being a "1". Hence, when calculating the transition probability of timing path N5-N8-N11 the result can vary significantly. Another effect can be seen in Figure 5.5b between 0.72V and 0.85V for the simulated result. One can see that when the number of failing timing paths is changing (visible as new edges in the curve), the resulting error probability is actually decreasing. The estimation using PrimeTime however was not able to detect new failing timing paths in this interval, which is why the estimated error probability is not changing, and in particular not decreasing. The reason for this is likely that some paths are becoming so slow that the edges of the transitions do not reach the last stage of logic gates. This in turn results in a changing transition possibilities at the gates. Depending on load capacitance, wire lengths and these effects appear as new timing paths that would have to be considered. Surprisingly however, as it can be seen in the graph, these timing paths can actually reduce the resulting error probability at the timing endpoint. The operation at these very low supply voltages and high error rates is however very unlikely, even for Approximate Computing. This is why one can safely ignore these effects for now. Finally, in Figure 5.5c one can see the performance of the presented estimation for a regular size circuit. One can see that the simplified estimation gives a conservative estimation of the error rate. At any scaled supply voltage the estimations of the simplified estimation are above the simulated results. Due to its simplicity and the fact that it gives a pessimistic and safe estimation of the error rate, the simplified estimation has been selected as the method further pursued. However, there is definitely potential for a further improvement of the methodology in order to make the estimations more accurate. The results based on the simplified estimation are however accurate enough so that the potential power savings due to voltage over-scaling can be evaluated in the following for a benchmark circuit.

5.3 Shared Component Handling

When looking back again at the schematic of circuit "c17" in Figure 5.3, one can see that for instance the NAND2 gate driving net N9 is part two timing paths, one ending at net N10 and one ending at net N11. This gate is shared between timing paths. It is now likely that the scaling procedure of one endpoint results in another supply voltage than the one of another endpoint. In order to guarantee an operation within the approximation limits, it is therefore necessary to operate each gate with the worst-case supply voltage. Hence the largest supply voltage determined for the various endpoints has to be selected. For instance, if the previously mentioned NAND2 gate could be operated for endpoint 1 at 0.9V and for endpoint 2 at 1.0V, one would have to operate the gate with a supply voltage of at least 1.0V to be on the save side at any time.

5.4 Voltage Islands

Until now, no restrictions have been made regarding the number of different supply voltage domains within one circuit. In reality however, the number of different voltage domains has to be likely limited to very few, due to the complexity of implementing several supply voltage planes in the chip, the imposed routing problems and the need for various external voltage regulators. Similarly, no restrictions have been imposed on the capability of redefining voltage domains at run-time, which is clearly even more difficult than realizing several static voltage domains. Hence, dynamically switching between approximated operating points due to voltage over-scaling is likely very difficult. However, in the next section it will be shown that even when sticking to very few voltage domains, the savings due to voltage over-scaling can be significant. In future work, it has to be shown how floorplanning, and external voltage regulators affect the applicability of supply voltage over-scaling.

5.5 Applied voltage over-scaling

With the help of the methodology presented in the previous sections, it is now possible to perform the scaling of the supply voltage and estimate the resulting error rates. In the previous chapter it has been shown how the maximum error probability can be determined, that can be tolerated at each register input. This information is the prerequisite in this chapter in order to perform the approximation by voltage over-scaling. Hence, it is assumed that the maximum error rates that can be tolerated for an approximated operating point are known for each register in the circuit. Due to the analytic scaling approach presented, the scaling operation is straight-forward as the time it takes to estimate the error rates is not critical. Hence, the most simple strategy is to simply reduce the nominal voltage of the whole fanin of an endpoint by a predefined δ and estimate the resulting error rate. If the error rate is below the maximum value, one can repeat the step, if not, the voltage has to be increased by δ and the scaling has to be stopped. This scaling operation has to be performed for all timing endpoints in the circuit that

are voltage over-scaling candidates, i.e. that can tolerate an error probability larger than $p_e > 0.0$. Once done, it has to be taken care of the "shared components", as it has been describe in Section 5.3. Afterwards, the minimum voltage for each gate in the circuit is known. This information can then be used to define voltage islands, either statically defined at synthesis time, or dynamically at run-time. All these steps are comparably simple and in this work performed with the help of some TCL scripts.

The proposed methodology has been evaluated with the already introduced Sobel filter example application. The application has been approximated as it has been presented in the previous chapter. Three approximate operating points have been defined, resulting in an image quality (PSNR) of 30, 40 or 50 dB. There is practically no degradation of the visual quality visible between the three operating points, as it can be seen in Figure 5.6. The circuit that is to be approximated in this example is the multiplication unit of the

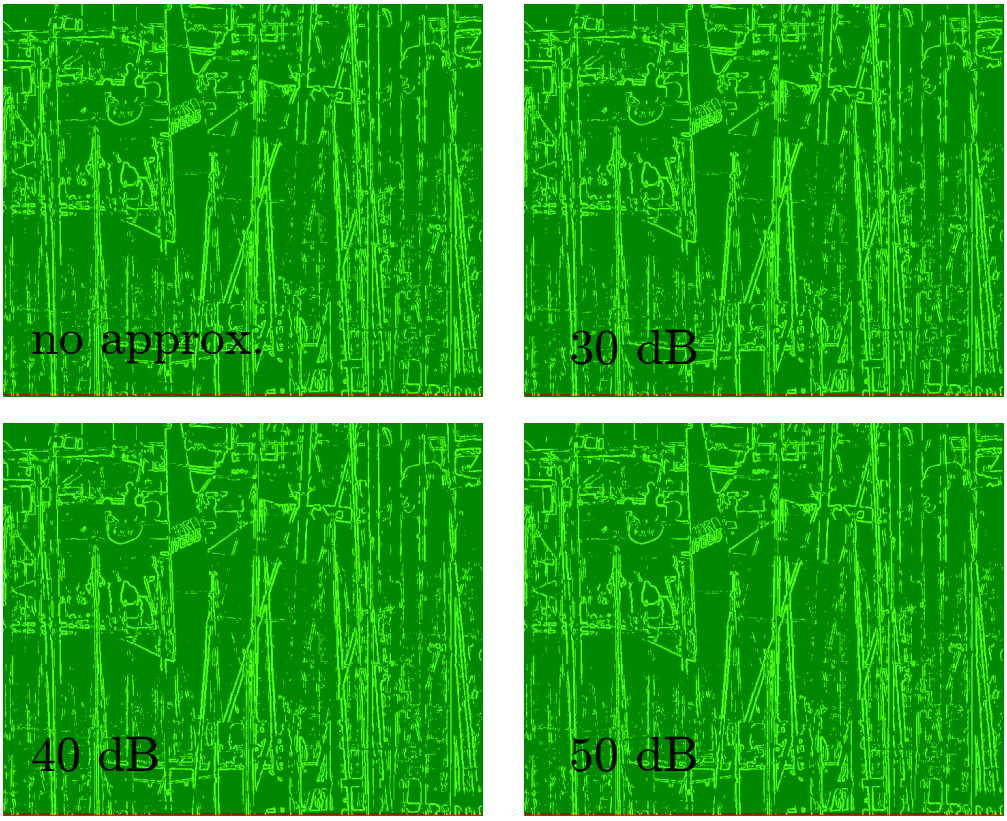


Figure 5.6: Difference of the visual quality of a Sobel filtered still when applying no approximation and an approximation for target qualities 30, 40 and 50 dB

floating-point unit that the Sobel filter utilizes. For each of the three operating points a list has been generated stating the maximum error probability that can be tolerated at each register of the FPU (fmul). The circuit consists of 2030 flip-flops, i.e. timing endpoints. Each of these endpoints has been scaled individually, using the "simplified" methodology presented in this chapter. Parallelizations have not been applied. The

scaling analysis took about one hour on a regular desktop PC. The functionality of the circuit for the three operating points has been verified using timing annotated simulations, as presented earlier. Figure 5.7 is showing the estimated power consumption of the FPU for the three operating points. The power consumption has been estimated using Synopsys PrimeTime with accurate test pattern. One can see that compared to applying

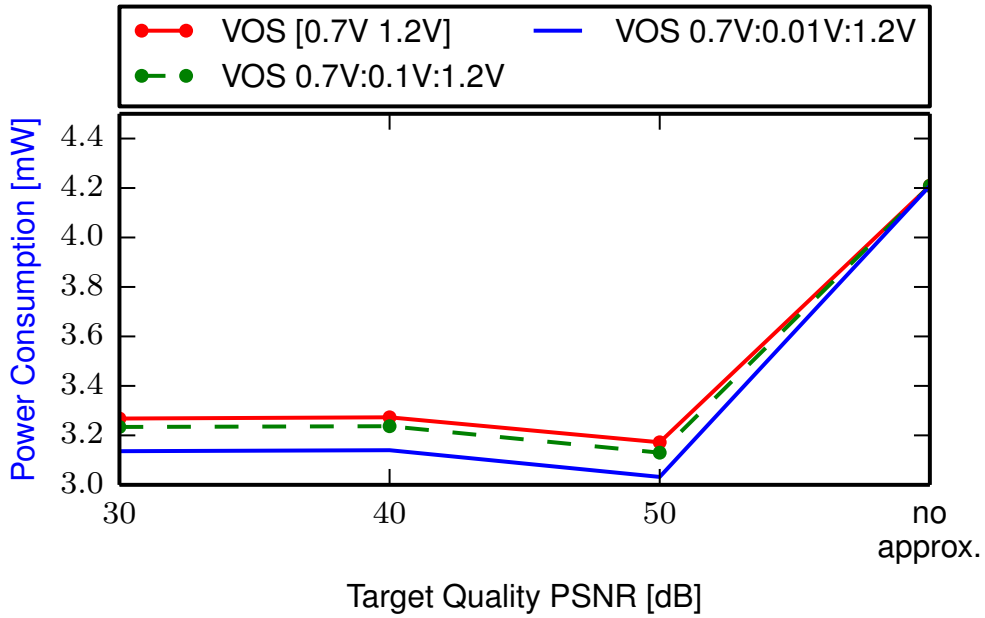


Figure 5.7: Estimated power consumption of a floating point unit for different approximated operating points of a Sobel filter when applying voltage over-scaling [145]

no approximation at all, up to 27.9 % of the power consumption could be saved, when running the used benchmark. This is clearly an impressive result, even though a bit less than when applying "circuit pruning". This is not surprising as circuit pruning tackles the static as well as the dynamic power consumption. Furthermore, $0.7V$ has been assumed to be the minimum voltage, as this is safely above the threshold voltage of the technology, even though often paths could have been operated even slower. Surprisingly, the lowest power consumption could be observed for the approximation point having the best quality. It could be seen already when approximating the circuit in the previous chapter that the operating points are very close together. The difference is not large enough to seen a significant difference in the power consumption between the operating points. The fact that the minimum power consumption is not at 30 dB but at 50 dB is likely caused by inaccuracies mainly during the coarse and the fine approximation. Nevertheless, the applicability of voltage over-scaling could be proven. The reduction of the power consumption by applying voltage over-scaling compared to applying no approximation at all is clearly visible. As it has been elaborated in Section 5.4 it is unlikely that many different voltage domains will be implemented on a chip due to the resulting overhead and complexity. The measurements have been made when allowing 2 supply voltages,

51 voltage domains and 501 voltage domains, as shown in Figure 5.7. For instance "VOS [0.7V 1.2V]" corresponds to voltage over scaling (VOS) with the two voltage domains 0.7V and 1.2V. Instead, "VOS 0.7V:0.1V:1.2V" corresponds to the voltage islands 0.7V, 0.8V, 0.9V, 1.0V, 1.1V and 1.2V, hence in steps of 0.1V. The minimum voltage has been set to 0.7 V and the nominal voltage of the technology is 1.2 V. One can see that the difference in the potential power savings is comparably small for different sizes of voltage domains. Clearly, the largest power savings are possible if almost no restrictions are made to the number of voltage domains. In this case almost no supply voltage has to be rounded up (unnecessarily) to the next supply voltage. However, even when operating only two supply voltages, the power savings are significant. The difference to operating 501 voltage domains is comparably small. This is a very promising result as it shows that even a very small number of voltage domains can be sufficient in order to successfully apply circuit approximation through voltage over-scaling. This even seems to make a dynamic assignment, at run-time, of gates to voltage domains, feasible as the routing overhead can be limited.

5.6 Summary

In this chapter a methodology has been presented that allows to analytically estimate the required supply voltages of each gate within a circuit in order to approach an approximated operating point. The methodology is generically applicable to any sequential ASIC circuit. It is based on the commercial timing analysis software PrimeTime by Synopsys. PrimeTime is used to estimate at which supply voltage, which timing path is failing. Using the "Composite Current Source" Model, PrimeTime is able to perform a timing analysis for any applied supply voltage, even outside of the specifications. Based on this information, for each timing endpoint the resulting error probabilities are estimated using an methodology developed in this work. The methodology basically estimates the likelihood of a signal edge not being propagated to an endpoint due to masking effects. Two variants have been presented, one presenting a more accurate estimation, and the other one presenting a rougher estimation offering a major simplification of the computations. Experimental results have shown that the estimation of error rates is working fine in general, but has some limitations, as not all effects are covered. However, the presented approach offers a simple and very fast way to estimate error rates due to voltage over-scaling. It has then been shown how his methodology could be used to practically apply voltage over-scaling to approximate an exemplary circuit. The methodology allowed to approach the desired approximate operating points very closely and match the desired error rates. It could be shown that even very few voltage domains can be sufficient to gain a significant reduction of the power consumption. The results of the presented methodology are very promising. However, the approach is still in a very early development stage. There are many points at which it could be improved. Nevertheless, in this work the first automated analytic approach that can be generically applied has been presented. It therefore proves the practical feasibility of voltage over-scaling as an approximation technique for a wide range of applications.

6 Conclusion and Outlook

Approximate Computing Approximate Computing is a relatively new paradigm, promising to substantially decrease the power consumption of digital integrated circuits. In the beginning of this work, a comprehensive introduction has been given into Approximate Computing itself and its classification as a low-power technique in general. The fundamental relations of the design parameters of an integrated circuit to the power consumption of integrated circuits have been explained, as well as its concomitant problems and limitations. It has been shown that a reduction of the power consumption is very hard to achieve without the introduction of new problems and limitations at other points within the design space. The four antagonists, "power", "performance", "area" and "reliability" have to be balanced according to the demands of the application and its field of application. It seems that the four opponents can never be optimized simultaneously, at least with today's technology. Every time one factor is optimized, another one is suffering. For instance, when trying to improve the power consumption of a circuit, the performance is suffering due to tightened timing constraints. If the clock speed is not scaled accordingly the circuit reliability is affected due to the introduction of timing faults.

It has been shown how "Approximate Computing" is tackling this conflict by intentionally loosening the requirements on the reliability of a circuit. By relaxing these demands, the power consumption can be reduced while maintaining the performance at the same time. It has been extensively elaborated which applications types can be used for applying such an approach. It could be shown that mainly applications from the signal, audio and video processing domains seem to be suitable for Approximate Computing. In general, most applications that interact with the human, i.e. imperfect, perception seem to be applicable.

A detailed overview has also been given about the various approximation techniques that have been reported and applied in the context of Approximate Computing in the literature. The two most common techniques, that also promise the highest power savings, have been selected to be considered throughout this work. With "functional approximation" a technique is proposed that tries to reduce the power consumption of a circuit by approximating its function. Simplified, functional approximation is the process of removing or approximating those parts of a circuit that are responsible only for functionality that is not absolutely needed or precision of a circuit that is not required. This technique, sometimes referred to as "circuit pruning", promises the largest power savings, as the dynamic and the static power consumption can be reduced. The difficulty of applying this technique is to find out which parts of a circuit can be approximated. Usually a minimum quality constraint has to be defined for the application. This constraint must not be violated when applying these approximations.

The second technique where focus has been put on in this work is "voltage over-scaling".

The idea of voltage over-scaling is to reduce the supply voltage such that the power consumption significantly reduces. Compared to regular voltage-scaling techniques, the tightening of timing constraints coming from the scaling is ignored, hence timing violations are accepted. Due to the quadratic relationship of the supply voltage of MOSFETs to its power consumption, the techniques suits well to save power. Usually it is not possible to scale the supply voltage of a circuit equally at once. The difficulty of applying this technique is again to find out where the approximations can be applied, i.e. where timing violations can be tolerated without violating a global quality constraint of the application. Varying (scaled) supply voltages results in a varying probabilities of timing violations, which in turn results in a varying global quality degradation. Hence, for this technique, it is additionally required to find out to which degree the elements of the circuit can be approximated.

Prior Art & Contributions of This Work Approximate Computing has already been widely applied in previous works. However, it has been shown in this work that existing works have some major drawbacks, that this work has been trying to solve. Many of the techniques and results presented in prior art in the context of Approximate Computing promise high power savings for a variety of applications. We have seen that in almost any case the approximations are applied more or less manually to the applications. I.e. usually the circuits are analyzed by inspection regarding their suitability for approximation techniques. This is one of the main limitations that has been tried to be solved in this work. In this work a methodology has been developed that allows to automatically, wherever possible, determine where approximations are possible and to what extent.

Another limitation of most of the presented prior art is that they solely focus on the approximation of algorithmic building blocks containing no sequential logic. Usually approximate variants of adders have developed, as they build the basis for a variety of circuits. This is clearly a valid approach and largely simplifies the design space. In this work however, the goal was not to impose any restrictions regarding the type of circuit. The methodology developed in this work is applicable to any kind and size of circuit. In particular, in contrast to almost any previous work, the approach presented in this work is applicable to sequential circuits, not only to combinational ones.

The presented approach can be used to apply a variety of different approximation techniques. Compared to related work, it proposes an intermediate abstraction layer to describe the resulting effects of the applied approximations, independent of the applied technique. Bit-flip probabilities at the register-transfer level are used as the common layer to describe the effects of approximations. This probability-awareness at register-transfer level, allows to analyze the circuits in the beginning at a slightly abstracted level. Later on, when analyzing the applicability of more fine-granular techniques at gate-level, one can still decrease the abstraction level of the analysis. But due to the fact that the global relations between approximation and resulting error rate are already analyzed at register-transfer level, the gate level approximation does not have to be analyzed globally but locally, i.e. from one start flip-flop to one end flip-flop. This “divide and conquer” based approach keeps the complexity manageable even for large circuits.

On the other hand, this probability-awareness however increases the complexity of the analysis itself, as faults have to be modeled by their probability of appearance. In order to generate significant results, a reasonable amount of clock cycles has to be analyzed. Approximation speed, i.e. the time that is required in order to automatically approximate a circuit, is therefore of highest importance. Compared to most existing generic approaches that are based on software-based simulations, the one elaborated in this work uses hardware accelerated emulations in order to analyze the behavior of approximated circuits. The emulation allows to analyze a circuit at run-time and directly observe the imprecise behavior of the circuit resulting from the applied approximations.

The FPGA-based emulation methodology is one of the main contributions presented in this work. It is the key to the automated approximation of generic sequential circuits without the need for lengthy software-based simulations. It has been shown in this work what a toolflow could look like, generating a FPGA design performing the emulation and fault injection of a circuit. Focus has been put on the ability to automate the generation wherever possible. The basic structure of the presented emulator stays the same for each circuit to be analyzed. It has been shown that compared to existing implementations, in this work faults are injected based on assigning error rates to the flip-flops in a circuit. This probability-awareness is the key differentiator to previous work. It has been shown that this flexibility does not come for free and results in an increasing demand for hardware resources. This increasing requirement for hardware resources on the FPGA concomitant with this flexibility has been a major difficulty that had to be overcome. Several different approaches in order to implement a probability-aware emulation system have been implemented and evaluated in this work, each having their benefits and drawbacks. One could see that the key for a good implementation of the emulator is to find an implementation that offers a compromise between speed and required hardware resources. At the same time the generation of faults has to be as random as possible in order to allow to model the approximations a realistic as possible. The implementation that had been selected to be used for the remainder of the work offered the most practical trade-off between speed, area overhead and random number quality. By exploiting special shift-register structures on the FPGAs, this implementation can keep the hardware overhead low even for random numbers with a large periodicity. For this implementation it takes 32 clock cycles to emulate one real clock cycle. This is several magnitudes faster than software-based simulations.

This emulation platform has been used to develop algorithms to automatically approximate circuits, by performing fault injection experiments. The methodology developed and evaluated in this work consists of several steps. The term “application-reasoned approximation” has been introduced, describing the need to propagate the requirements on the precision, down from the application to the circuit level. This is the first preparation step of the approximation. For the methodology introduced in this work it is therefore inevitable to provide realistic and accurate simulation models and testbenches of the application that embeds the circuit to be approximated. Using the models it is possible to accurately determine the precision required at the outputs of the hardware circuit, given a quality requirement of the embedding application. Not until the error rates that can be tolerated at each output pin of the circuit to be approximated are known, one

can start approximating it.

The goal of the first approximation step developed in this work is to find out which circuit elements are qualified for an approximation at all. It has been presented in detail, why it is usually not possible to approximate those parts of the circuit that are at any point involved in the control flow of the circuit. In order not to manually identify those circuit flip-flops, algorithms have been developed and evaluated that allow to automatically identify the flip-flops of the circuit that do not qualify for an approximation at all, as they are influencing the control flow of the circuits.

It could be observed that sometimes an additional step is required prior continuing with the actual approximation. For some circuits it could be observed that the approximation of some elements results in a very large variance of the resulting error rates measured at the output pins of the circuits. These elements are therefore also not qualified for an approximation, i.e. they have to be operated fully precise, as approximations always have to result in a predictable behavior. An approach has been presented to automatically exclude these circuit elements from the approximation candidates as well.

In the following, it had been shown how the remaining circuit elements can be approximated in order to approach the desired, imprecise, operating point, using the developed emulation system. The approximation has been split up into two parts in order to simplify the problem. The first part, the coarse approximation, identifies all circuit registers that can be completely removed or switched off in order to approach an approximated operating point. Approaches have been presented and evaluated, that allow to automatically perform this "circuit pruning" operation. The performance and the resulting power savings have been evaluated for exemplary benchmark circuits. It could be shown that for the presented benchmark circuits up to 43% of the power consumption could be saved.

The second approximation step developed in the scope of this work is required to identify the degree of approximation that is possible for those registers that do not qualify for a coarse approximation, but can still be operated imprecise. The algorithm developed in this work determines for each register in the circuit the imprecision that can be tolerated, in terms of error rate, without violating the overall quality constraint. The results of this analysis however cannot be directly used for the approximation of a circuit. The results serve instead as the basis for more fine-granular approximation techniques, like Boolean modification, frequency over-scaling or voltage over-scaling. The fine-grained approximation has therefore not been evaluated in terms of saved power, but in terms of the "sum of tolerated error probabilities".

The practicality of the results generated by the coarse and the fine-grained approximation has been evaluated by using them for "voltage over-scaling" as the approximation technique. In order to put voltage over-scaling into practice, more analysis steps have to be performed that have been developed in this work as well. Mainly two questions have been answered using the algorithms developed. At first, a methodology has been developed answering which timing paths get violated in a circuit at which supply voltage V_{dd}. And second, algorithms have been developed allowing to estimate the resulting error rate at the flip-flops, when the timing is violated due to scaled supply voltages. The accuracy of the estimation has been evaluated using several benchmark circuits. Finally,

voltage over-scaling has been applied to a benchmark circuit using the overall methodology presented in this work, from the beginning to the end, demonstrating the interplay of the various steps.

Limitations and Benefits Albeit the methodology presented in this work offers many benefits compared to other approaches, there are also some limitations and drawbacks. First of all, the usage of an FPGA-based fault emulation system in order to analyze the behavior of integrated circuits always requires extra effort compared to software-based simulations. Interfacing the external world does usually pose the biggest problems. Focus has been put on providing a very generic and easy to use interface with the emulation system. However, the required effort is still much higher than with purely software-based solutions. Furthermore, even though the hardware overhead has been reduced to a minimum, and today's FPGAs are growing rapidly, the hardware overhead due to the probability overhead is still a limiting factor. On the other hand however, the hardware-based emulation is enabling the approximation at such a detailed level for large sequential circuits in the first place. Software-based analysis would require several orders of magnitude more time to generate the same results. And the probability-awareness, resulting in the large area overhead, enables to analyze circuits for a variety of approximation techniques at once and globally for the whole circuit.

Another limitation is that all over the methodology accurate and realistic testbenches are required. Unfortunately, this cannot be avoided as only those allow to trigger a behavior as close to as it would be in a real world environment. Only then it is possible to identify which approximations have a critical effect and which can be tolerated.

The approximation at register-transfer level offers a good compromise between abstraction and speed. However for some approximation techniques the granularity has to be increased in a later step in order to be performed. The initial approximation analysis at register-transfer level, based on error rates, also offers a high flexibility regarding the approximation techniques to be applied. Additionally, as the approximation behavior is analyzed globally up to the register-level, a more fine-granular analysis can be performed locally, allowing to parallelize the analysis. The separation into two steps seems to be an efficient way to divide the approximation into manageable pieces. Regarding the fine approximation, the main limitation is that currently the power consumption is not directly taken into account as an optimization goal of this optimization problem and could be performed in a more optimal manner. The limitation of the presented methodology for automated voltage over-scaling is clearly the accuracy of the error rate estimation. While the detection of failing timing paths is a comparably simple problem, the analytical estimation of error rates is more complex. However, as the presented algorithms offer a conservative estimation, the inaccuracies only result in a under-estimation. Hence, the tolerable error rates are not exceeded, but the full potential of power savings cannot be exploited. Regardless of this drawback the presented approach offers a very fast alternative to software-based gate-level simulations or even SPICE simulations at transistor level. Due to the fact that usually thousands of timing endpoints and supply voltage combinations have to be tested, a simulative approach disqualifies due to the exorbitant

simulation times.

Future Work One limitation that should be analyzed and possibly solved in future work is that currently during the fine-grained approximation the power savings are not directly part of the optimization goals. Currently, the only optimization goal is to meet the reliability constraints. It might however be possible that the order of the approximation, with which flip-flop the algorithm starts, matters. Even though no relevant difference could be noticed for the used benchmark circuits, it is possible that potential power savings are lost at this point. For instance the estimated switching activity could be taken into account in order to estimate which gates are more active, and hence consume more power, than others.

The estimation of error rates due to failing timing paths could be improved in future work. We have seen that especially for a large reduction of the supply voltage the estimation becomes inaccurate. Even though it unlikely that circuits will be operated at such low supply voltages, potential power savings can be lost due to these inaccuracies.

One research field that has not been directly defined as a potential field of application but has been always kept in mind throughout this work, is the reliability analysis. The field is clearly closely related to Approximate Computing. The emulator and parts of the approximation algorithms can be directly, without any changes, used to analyze the susceptibility, respectively the resilience, of a circuit to soft-errors. Furthermore it can be used to identify those parts of the circuit that need further protection, e.g. by applying redundancy mechanisms. By defining a maximum tolerable error probability for the outputs of a circuit (usually much smaller than when applying Approximate Computing), the presented methodology can detect for every flip-flop in the circuit the maximum error rate that may appear due to soft-errors. The smaller the detected rate is, the more protection is needed and vice versa. However, comparable to voltage over-scaling, further fine-granular steps are required to translate this information into physical parameters like supply voltage and channel width. However, for flip-flops that have been identified to tolerate no error at all ($p_e = 0.0$) triple-modular redundancy could be directly applied for the whole fanin without any further analysis.

Assessment and Outlook We have seen that Approximate Computing is a promising field of research, that allows to save significant amounts of power. It's applicability however is very complex. The methodology presented in this work proposes one way to put it into practice. Throughout the whole work, focus has been put on automating the process wherever possible and to make it as generic and fast as possible. The presented approach is the first one to offer a full methodology from application level down to the gate-level, applying functional approximation at register-transfer level and voltage over-scaling at gate-level. Nevertheless, one can see that still a lot of manual interaction is required, although it is much less than for other approaches. Still, it seems that the practical applicability is one of the factors that has to be further improved in order to promote the approach more. One way to do this could be to develop approximate building libraries. At the level of arithmetic building blocks, this is already done today,

e.g. by Shafique et al. in [129]. However, this still requires the designer to have an in depth knowledge about the design and its requirements on reliability or precision. For a wider applicability it might be helpful to provide libraries containing macro-blocks instead of micro-blocks. The analysis of the macro-blocks could be performed using accurate and complex tools, like the one presented in this work. The designer would just have to configure them according to his needs and use them in his design. Thinking one step further, this approach would embed perfectly into today's high-level synthesis and block-level synthesis approaches, like Xilinx' "System Generator". The designer could implement its algorithms, as he is used to, in MATLAB or Simulink and define quality limits. The EDA tools could then perform a high level reliability, respectively precision, analysis like the one presented in this work in Section 4.3. Due to the high level of abstraction this is an easy and fast task. Later, when performing the high-level synthesis the "quality" of the macro-blocks would just have to be configured according to the previously gained results. This would ease the effort for the designer in order to apply Approximate Computing tremendously and allow to widely adopt it and benefit from it.

Bibliography

[1] The International Technology Roadmap for Semiconductors (ITRS), System Drivers, 2013, http://www.itrs.net/.

[2] M. Alam. Reliability-and process-variation aware design of integrated circuits. *Microelectronics Reliability*, 48(8):1114–1122, 2008.

[3] N. Weste, D. Harris, and A. Banerjee. Cmos vlsi design. *A circuits and systems perspective*, 11:739, 2005.

[4] M. Shafique, S. Garg, J. Henkel, and D. Marculescu. The eda challenges in the dark silicon era. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.

[5] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *Test Symposium (ETS), 2013 18th IEEE European*, pages 1–6. IEEE, 2013.

[6] A. P. Chandrakasan, W. J. Bowhill, and F. Fox. *Design of high-performance microprocessor circuits*. Wiley-IEEE press, 2000.

[7] K. Cao, W.-C. Lee, W. Liu, X. Jin, P. Su, S. Fung, J. An, B. Yu, and C. Hu. Bsim4 gate leakage model including source-drain partition. In *Electron Devices Meeting, 2000. IEDM'00. Technical Digest. International*, pages 815–818. IEEE, 2000.

[8] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan. Approximate computing and the quest for computing efficiency. In *Proceedings of the 52nd Annual Design Automation Conference*, page 120. ACM, 2015.

[9] G. Anthes. Inexact design: beyond fault-tolerance. *Communications of the ACM*, 56(4):18–20, 2013.

[10] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266, 2010.

[11] M. Hübner and C. Silvano. *Near Threshold Computing: Technology, Methods and Applications*. Springer, 2015.

[12] K. Palem. Energy aware computing through probabilistic switching: A study of limits. *Computers, IEEE Transactions on*, 54(9):1123–1137, 2005.

[13] S. Cheemalavagu, P. Korkmaz, K. Palem, B. Akgul, and L. Chakrapani. A probabilistic cmos switch and its realization by exploiting noise. In *IFIP International Conference on VLSI*, pages 535–541, 2005.

[14] B. Akgul, L. Chakrapani, P. Korkmaz, and K. Palem. Probabilistic cmos technology: A survey and future directions. In *Very Large Scale Integration, 2006 IFIP International Conference on*, pages 1–6. IEEE, 2006.

[15] L. N. Chakrapani, J. George, B. Marr, B. E. Akgul, and K. V. Palem. Probabilistic design: A survey of probabilistic cmos technology and future directions for terascale ic design. In *VLSI-SoC: Research Trends in VLSI and Systems on Chip*, pages 101–118. Springer, 2008.

[16] P. Korkmaz, B. E. Akgul, K. V. Palem, and L. N. Chakrapani. Advocating noise as an agent for ultra-low energy computing: probabilistic complementary metal–oxide–semiconductor devices and their characteristics. *Japanese journal of applied physics*, 45(4S):3307, 2006.

[17] K. Palem and A. Lingamneni. What to do about the end of moore's law, probably! In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 924–929. IEEE, 2012.

[18] K. Palem and A. Lingamneni. Ten years of building broken chips: the physics and engineering of inexact computing. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2s):87, 2013.

[19] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet. Energy parsimonious circuit design through probabilistic pruning. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[20] J. Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

[21] G. Moore. No exponential is forever–but we can delay forever. In *IBM Academy of Technology 2003 Annual Meeting, San Francisco, CA*, 2003.

[22] R. H. Dennard, V. Rideout, E. Bassous, and A. Leblanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, 1974.

[23] A. Herkersdorf, M. Engel, M. Glaß, J. Henkel, V. Kleeberger, M. Kochte, J. Kühn, S. Nassif, H. Rauchfuss, W. Rosenstiel, U. Schlichtmann, M. Shafique, M. Tahoori, J. Teich, N. Wehn, C. Weis, and H.-J. Wunderlich. Cross- layer dependability modeling and abstraction in systems on chip. In *Workshop on Silicon Errors in Logic System Effects (SELSE)*, 2013.

[24] K. Stein. Noise-induced error rate as limiting factory for energy per operation in digital ics. *Solid-State Circuits, IEEE Journal of*, 12(5):527–530, 1977.

[25] L. Kish. End of moore's law: thermal (noise) death of integration in micro and nano electronics. *Physics Letters A*, 305(3):144–149, 2002.

[26] P. Layman and S. Chamberlain. A compact thermal noise model for the investigation of soft error rates in mos vlsi digital circuits. *Solid-State Circuits, IEEE Journal of*, 24(1):79–89, 1989.

[27] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *Micro, IEEE*, 23(4):14–19, 2003.

[28] R. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *Device and Materials Reliability, IEEE Transactions on*, 5(3):305–316, 2005.

[29] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy. Impact: imprecise adders for low-power approximate computing. In *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, pages 409–414. IEEE Press, 2011.

[30] Q. Xu, T. Mytkowicz, and N. S. Kim. Approximate computing: A survey.

[31] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 113:1–113:9, New York, NY, USA, 2013. ACM. URL: `http://doi.acm.org/10.1145/2463209.2488873`, `doi:10.1145/2463209.2488873`.

[32] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *Proceedings of the International Conference on Computer-Aided Design*, pages 667–673. IEEE Press, 2011.

[33] D. Robinson and G. Sutton. Age effect in hearing-a comparative analysis of published threshold data. *Audiology*, 18(4):320–334, 1979.

[34] D. B. Judd and G. Wyszecki. Color in business, science and industry. 1963.

[35] C.-H. Huang, Y. Li, and L. Dolecek. Acoco: Adaptive coding for approximate computing on faulty memories. *IEEE Transactions on Communications*, 63(12):4615–4628, 2015.

[36] L. N. Chakrapani, B. E. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee. Ultra-efficient (embedded) soc architectures based on probabilistic cmos (pcmos) technology. In *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pages 1110–1115. European Design and Automation Association, 2006.

[37] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, page 113. ACM, 2013.

[38] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[39] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[40] C. ISO and I. DIS. 11172. *Coding of moving pictures and associated audio for digital storage media at up to about*, 1, 1992.

[41] L. L. Peterson and B. S. Davie. *Computer networks: a systems approach*. Elsevier, 2007.

[42] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig. Approximate signal processing. *Journal of VLSI signal processing systems for signal, image and video technology*, 15(1-2):177–200, 1997.

[43] Y. Wu and B. D. Woerner. The influence of quantization and fixed point arithmetic upon the ber performance of turbo codes. In *Vehicular Technology Conference, 1999 IEEE 49th*, volume 2, pages 1683–1687. IEEE, 1999.

[44] S. T. Chakradhar and A. Raghunathan. Best-effort computing: re-thinking parallel software and hardware. In *Proceedings of the 47th Design Automation Conference*, pages 865–870. ACM, 2010.

[45] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, volume 46, pages 164–174. ACM, 2011.

[46] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *ACM SIGPLAN Notices*, volume 47, pages 301–312. ACM, 2012.

[47] I. J. Chang, D. Mohapatra, and K. Roy. A priority-based 6t/8t hybrid sram architecture for aggressive voltage scaling in video applications. *Circuits and Systems for Video Technology, IEEE Transactions on*, 21(2):101–112, 2011.

[48] G. Karakonstantis, D. Mohapatra, and K. Roy. Logic and memory design based on unequal error protection for voltage-scalable, robust and adaptive dsp systems. *Journal of Signal Processing Systems*, 68(3):415–431, 2012.

[49] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan. Approximate storage for energy efficient spintronic memories. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[50] M. A. Breuer. Intelligible test techniques to support error-tolerance. In *Test Symposium, 2004. 13th Asian*, pages 386–393. IEEE, 2004.

[51] J. Miao, K. He, A. Gerstlauer, and M. Orshansky. Modeling and synthesis of quality-energy optimal approximate adders. In *Proceedings of the International Conference on Computer-Aided Design*, pages 728–735. ACM, 2012.

[52] A. B. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th Annual Design Automation Conference*, pages 820–825. ACM, 2012.

[53] J. Liang, J. Han, and F. Lombardi. New metrics for the reliability of approximate and probabilistic adders. *Computers, IEEE Transactions on*, 62(9):1760–1771, 2013.

[54] L. N. Chakrapani and K. V. Palem. A probabilistic boolean logic and its meaning. *Rice University, Department of Computer Science Technical Report*, 2008.

[55] V. Levin. Probability analysis of combination systems and their reliability. *Engin. Cybernetics*, (6):78–84, 1964.

[56] M. Baze, W. Bartholet, T. Dao, and S. Buchner. An seu analysis approach for error propagation in digital vlsi cmos asics. *IEEE Transactions on Nuclear Science*, 42(CONF-950716–), 1995.

[57] K. N. Patel, I. L. Markov, and J. P. Hayes. Evaluating circuit reliability under probabilistic gate-level fault models. In *Proceedings of the International Workshop on Logic and Synthesis*, pages 59–64, 2003.

[58] D. T. Franco, M. C. Vasconcelos, L. Naviner, and J. F. Naviner. Reliability analysis of logic circuits based on signal probability. In *Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on*, pages 670–673, Aug 2008. doi:10.1109/ICECS.2008.4674942.

[59] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes. Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 282–287. IEEE, 2005.

[60] D. T. Franco, M. C. Vasconcelos, L. Naviner, and J. F. Naviner. Reliability of logic circuits under multiple simultaneous faults. In *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium on*, pages 265–268, Aug 2008. doi:10.1109/MWSCAS.2008.4616787.

[61] Z. Wang, H. Xie, S. Chafekar, and A. Chattopadhyay. Architectural error prediction using probabilistic error masking matrices. In *Quality Electronic Design (ASQED), 2015 6th Asia Symposium on*, pages 31–36, Aug 2015. doi:10.1109/ACQED.2015.7274003.

[62] T. Rejimon, K. Lingasubramanian, and S. Bhanja. Probabilistic error modeling for nano-domain logic circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(1):55–65, 2009.

[63] S. Mirkhani, S. Mitra, C. Y. Cher, and J. Abraham. Efficient soft error vulnerability estimation of complex designs. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, pages 103–108, March 2015.

[64] S. Kumar, R. H. Klenke, J. H. Aylor, B. W. Johnson, R. D. Williams, and R. Waxman. Adept: A unified system level modeling design environment. In *Proceedings of the 1st Annual RASSP Conference, Arlington, VA*, pages 114–123, 1994.

[65] J. George, B. Marr, B. Akgul, and K. Palem. Probabilistic arithmetic and energy efficient embedded signal processing. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 158–168. ACM, 2006.

[66] A. Bhanu, M. S. Lau, K.-V. Ling, V. J. Mooney III, and A. Singh. A more precise model of noise based pcmos errors. In *Electronic Design, Test and Application, 2010. DELTA'10. Fifth IEEE International Symposium on*, pages 99–102. IEEE, 2010.

[67] A. Singh, A. Basu, K.-V. Ling, and V. J. Mooney III. Modeling multi-output filtering effects in pcmos. In *VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on*, pages 1–4. IEEE, 2011.

[68] M. S. Lau, K.-V. Ling, A. Bhanu, and V. J. Mooney III. Error rate prediction for probabilistic circuits with more general structures. *Proc. of the WS on Synthesis And System Integration of Mixed Information Technologies*, pages 220–225, 2010.

[69] J. Gracia, J. C. Baraza, D. Gil, and P. J. Gil. Comparison and application of different vhdl-based fault injection techniques. In *Defect and Fault Tolerance in VLSI Systems, 2001. Proceedings. 2001 IEEE International Symposium on*, pages 233–241. IEEE, 2001.

[70] N. Song, J. Qin, X. Pan, and Y. Deng. Fault injection methodology and tools. In *Electronics and Optoelectronics (ICEOE), 2011 International Conference on*, volume 1, pages V1–47. IEEE, 2011.

[71] K. K. Goswami and R. K. Iyer. Depend: A design environment for prediction and evaluation of system dependability. In *Digital Avionics Systems Conference, 1990. Proceedings., IEEE/AIAA/NASA 9th*, pages 87–92. IEEE, 1990.

[72] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson. Fault injection into vhdl models: the mefisto tool. In *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*, pages 66–75. IEEE, 1994.

[73] R. Hartl, A. J. Rohatschek, W. Stechele, and A. Herkersdorf. Architectural vulnerability factor estimation with backwards analysis. In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 605–612. IEEE, 2010.

[74] K.-T. Cheng, S.-Y. Huang, and W.-J. Dai. Fault emulation: A new methodology for fault grading. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(10):1487–1495, 1999.

[75] A. Parreira, J. Teixeira, and M. Santos. A novel approach to fpga-based hardware fault modeling and simulation. In *Design and Diagnostics of Electronic Circuits and Syst. Workshop*, pages 17–24, 2003.

[76] L. Kafka and O. Novák. Fpga-based fault simulator. In *Design and Diagnostics of Electronic Circuits and systems, 2006 IEEE*, pages 272–276. IEEE, 2006.

[77] L. Antoni, R. Leveugle, and B. Fehér. Using run-time reconfiguration for fault injection in hardware prototypes. In *Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings. 17th IEEE International Symposium on*, pages 245–253. IEEE, 2002.

[78] S.-A. Hwang, J.-H. Hong, and C.-W. Wu. Sequential circuit fault simulation using logic emulation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(8):724–736, 1998.

[79] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante. An fpga-based approach for speeding-up fault injection campaigns on safety-critical circuits. *Journal of Electronic Testing*, 18(3):261–271, 2002.

[80] C. López-Ongil, M. García-Valderas, M. Portela-García, and L. Entrena. Autonomous fault emulation: a new fpga-based acceleration system for hardness evaluation. *Nuclear Science, IEEE Transactions on*, 54(1):252–261, 2007.

[81] J.-M. Daveau, A. Blampey, G. Gasiot, J. Bulone, and P. Roche. An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip. In *Reliability Physics Symposium, 2009 IEEE International*, pages 212 –220, april 2009. doi:10.1109/IRPS.2009.5173253.

[82] A. Janning, J. Heyszl, F. Stumpf, and G. Sigl. A cost-effective fpga-based fault simulation environment. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*, pages 21–31. IEEE, 2011.

[83] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: quantified error and confidence. In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pages 502–506. IEEE, 2009.

[84] J.-M. Daveau, A. Blampey, G. Gasiot, J. Bulone, and P. Roche. An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip. In *Reliability Physics Symposium, 2009 IEEE International*, pages 212–220. IEEE, 2009.

[85] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the 1998 international symposium on Low power electronics and design*, pages 76–81. ACM, 1998.

[86] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 721–725. ACM, 2002.

[87] J.-M. Chang and M. Pedram. Energy minimization using multiple supply voltages. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 5(4):436–443, 1997.

[88] Y.-J. Yeh and S.-Y. Kuo. An optimization-based low-power voltage scaling technique using multiple supply voltages. In *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, volume 5, pages 535–538. IEEE, 2001.

[89] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7–18. IEEE, 2003.

[90] R. Hegde and N. R. Shanbhag. Soft digital signal processing. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 9(6):813–823, 2001.

[91] B. Shim, S. R. Sridhara, and N. R. Shanbhag. Reliable low-power digital signal processing via reduced precision redundancy. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(5):497–510, 2004.

[92] R. Hegde and N. R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proceedings of the 1999 international symposium on Low power electronics and design*, pages 30–35. ACM, 1999.

[93] L. N. Chakrapani, K. K. Muntimadugu, A. Lingamneni, J. George, and K. V. Palem. Highly energy and performance efficient embedded computing through approximately correct arithmetic: A mathematical foundation and preliminary experimental validation. In *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems*, pages 187–196. ACM, 2008.

[94] J. Huang and J. Lach. Exploring the fidelity-efficiency design space using imprecise arithmetic. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 579–584. IEEE Press, 2011.

[95] C. Liu, J. Han, and F. Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 95. European Design and Automation Association, 2014.

[96] K. Nepal, Y. Li, R. Bahar, and S. Reda. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *Proceedings of the con-*

ference on Design, Automation & Test in Europe, page 361. European Design and Automation Association, 2014.

[97] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan. Aslan: synthesis of approximate sequential circuits. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 364. European Design and Automation Association, 2014.

[98] M. R. Choudhury and K. Mohanram. Approximate logic circuits for low overhead, non-intrusive concurrent error detection. In *Design, Automation and Test in Europe, 2008. DATE'08*, pages 903–908. IEEE, 2008.

[99] D. Shin and S. K. Gupta. Approximate logic synthesis for error tolerant applications. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 957–960. European Design and Automation Association, 2010.

[100] J. Miao, A. Gerstlauer, and M. Orshansky. Approximate logic synthesis under general error magnitude and frequency constraints. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 779–786. IEEE, 2013.

[101] V. Chandra and R. Aitken. Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos. In *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*, pages 114–122. IEEE, 2008.

[102] E. Costenaro, D. Alexandrescu, K. Belhaddad, and M. Nicolaidis. A practical approach to single event transient analysis for highly complex design. *Journal of Electronic Testing*, 29(3):301–315, 2013. URL: `http://dx.doi.org/10.1007/s10836-013-5385-9`, `doi:10.1007/s10836-013-5385-9`.

[103] J. Constantin, L. Wang, G. Karakonstantis, A. Chattopadhyay, and A. Burg. Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 381–386. IEEE, 2015.

[104] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy. Design of voltage-scalable meta-functions for approximate computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[105] K. He, A. Gerstlauer, and M. Orshansky. Controlled timing-error acceptance for low energy idct design. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[106] Y. Emre and C. Chakrabarti. Quality-aware techniques for reducing power of jpeg codecs. *Journal of Signal Processing Systems*, 69(3):227–237, 2012.

[107] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. Slack redistribution for graceful degradation under voltage overscaling. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 825–831. IEEE, 2010.

[108] P. K. Krause and I. Polian. Adaptive voltage over-scaling for resilient applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[109] Y. Liu, T. Zhang, and K. K. Parhi. Computation error analysis in digital signal processing systems with overscaled supply voltage. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(4):517–526, 2010.

[110] A. Herkersdorf, H. Aliee, M. Engel, M. Glaß, C. Gimmler-Dumont, J. Henkel, V. B. Kleeberger, M. A. Kochte, J. M. Kühn, D. Mueller-Gritschneder, et al. Resilience articulation point (rap): Cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectronics Reliability*, 54(6):1066–1074, 2014.

[111] M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner. Riffa 2.1: A reusable integration framework for fpga accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 8(4):22, 2015.

[112] A. Dunkels. Design and implementation of the lwip tcp/ip stack. *Swedish Institute of Computer Science*, 2:77, 2001.

[113] C. Donnely and R. Stallman. Gnu bison–the yacc-compatible parser generator, 2015.

[114] J. Uspensky. *Introduction to mathematical probability*. McGraw-Hill book company, inc., 1937. URL: `https://books.google.de/books?id=aeRQAAAAMAAJ`.

[115] P. Alfke. Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators. Technical report, Xilinx, 1996. `www.xilinx.com/bvdocs/appnotes/xapp052.pdf`.

[116] D. R. Hill, C. Mazel, J. Passerat-Palmbach, and M. K. Traore. Distribution of random streams for simulation practitioners. *Concurrency and Computation: Practice and Experience*, 25(10):1427–1442, 2013.

[117] L. Devroye and L. Devroye. *Non-uniform random variate generation*, volume 4. Springer-Verlag New York, 1986.

[118] F. Brglez, D. Bryan, and K. Kozmiński. Combinational profiles of sequential benchmark circuits. In *Circuits and Systems, 1989., IEEE International Symposium on*, pages 1929–1934. IEEE, 1989.

[119] S. Davidson. Characteristics of the itc99 itcbenchmark circuits. In *IEEE International Test Synthesis Workshop (ITSW)*, 1999.

[120] M. Ihmig, N. Alt, C. Claus, and A. Herkersdorf. Resource-efficient sequential architecture for fpga-based dab receiver. In *5th Karlsruhe Workshop on Software Radios*, 2008.

[121] N. SpaceWire-Links. Routers and networks. *ECSS Standard ECSS-E-ST-50-12C*, 24, 2003.

[122] C. Gimmler-Dumont, P. Schlafer, and N. Wehn. Asic implementation of a modified qr decomposition for tree search based mimo detection. In *Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on*, pages 1–4. IEEE, 2013.

[123] M. Loeve. *Probability theory, vol. i*. Springer, 1978.

[124] J. Al-Eryani. fpu100 at opencores.org, 2007. URL: `http://opencores.org/project,fpu100`.

[125] R. Yates. Fixed-point arithmetic: An introduction. *Digital Signal Labs*, 81(83):198, 2009.

[126] M. Fehrenz and M. Alles. Viterbi decoder at opencores.org, 2014. URL: `http://opencores.org/project,viterbi_decoder_axi4s`.

[127] G. Mekhtarian. Composite current source (ccs) modeling technology backgrounder. *Synopsys, Inc., November*, 2005.

[128] J. C. Costa, J. C. Monteiro, and S. Devadas. Switching activity estimation using limited depth reconvergent path analysis. In *Proceedings of the 1997 international symposium on Low power electronics and design*, pages 184–189. ACM, 1997.

[129] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel. Cross-layer approximate computing: From logic to architectures. In *ACM/IEEE Design Automation Conference (DAC)*, 2016.

Supervised Student Research

[130] F. Haeusler. Entwicklung eines Simulators zur Analyse transienter Fehler in integrierten Schaltungen. 2014.

[131] F. Haeusler. Dokumentation: LIS Werkstudententätigkeit. 2014.

[132] F. Haeusler. Aggressive voltage scaling for approximate computing. 2015.

[133] P. Huber. Development of a resource-aware probabilistic fault simulator. 2012.

[134] M. Huber. Analyse der Empfindlichkeit von digitalen Schaltungen gegen Soft Error. 2012.

[135] A. Khattab. Analysis and hardening of a hardware-based video decoder against soft errros. 2013.

[136] A. Krutwig. On-chip cmos bit error analysis of a digital audio broadcast receiver. 2012.

[137] A. Krutwig. Final report of the research internship. 2014.

[138] T. Raimar. Portierung eines hardwarebasierten Bewegungserkennungs-Systems auf moderne FPGA Architekturen, 2014.

[139] A. Grabas. Automated voltage over scaling for generic integrated circuits. 2015.

[140] K. Mueller. Gate-level approximation in digital circuits, 2016.

Publications

[141] D. May and W. Stechele. An fpga-based probability-aware fault simulator. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages 302–309. IEEE, 2012.

[142] D. May and W. Stechele. A resource-efficient probabilistic fault simulator. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–4. IEEE, 2013.

[143] D. May and W. Stechele. Improving the significance of probabilistic circuit fault emulations. In *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, pages 128–133. IEEE, 2014.

[144] D. May and W. Stechele. Design of fine-grained sequential approximate circuits using probability-aware fault emulation. In *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pages 73–78. IEEE, 2015.

[145] D. May and W. Stechele. Voltage over-scaling in sequential circuits for approximate computing. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–6. IEEE, 2016.