Technical University of Crete

Hardware and Software Development for the Efficient
Mapping of Mathematical
Problems on Field Programmable Gate Array Systems

A Dissertation Submitted
in Partial Fullfillment of the Requirements

for the degree

Doctor of Philosophy
In the Field of Electronics & Computer Engineering
By

Dimitrios Meintanis

Chania 2012

ΑΝΑΠΤΥΞΗ ΥΛΙΚΟΥ ΚΑΙ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΑΠΟΔΟΤΙΚΗ ΑΠΕΙΚΟΝΙΣΗ ΜΑΘΗΜΑΤΙΚΩΝ ΠΡΟΒΛΗΜΑΤΩΝ ΣΕ ΑΝΑΔΙΑΤΑΣΣΟΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Περιληψη

Στην αγορά της βιομηχανίας, οι σχεδιαστές έχουν σοβαρά κίνητρα για να βγάλουν τα προϊόντα τους γρήγορα στην αγορά. Αυτά είναι η αύξηση των κερδών και η διάρκεια του χρόνου διάθεσης του προϊόντος μέσα στην αγορά. Κάθε βδομάδα που ένα προϊόν δεν πωλείται, αυτό αντιπροσωπεύει χαμένα κέρδη και αυξάνει το ρίσκο αποτυχίας του προϊόντος στην αγορά. Τα επαναπορργραμματιζόμενα ολοκληρωμένα κυκλώματα (FPGAs) προσφέρουν μία λύση χαμηλού ρίσκου με γρήγορη είσοδο στην αγορά, όπου οι σχεδιαστές μπορούν εύκολα να μεταβάλλουν το προϊόν όταν χρειάζεται να κάνουν αλλαγές, να διορθώνουν λάθη ή ακόμη και να δημιουργήσουν παράγωγα προϊόντων κάποια στιγμή στο μέλλον.

Παρόλ'αυτά, η αυξημένη κατανάλωση ενέργειας είναι ένα απο τα κύρια μειονεκτήματα των ολοκληρωμένων FPGAs. Η ευελιξία που προσφέρουν οφείλεται σε έναν μεγάλο αριθμό διαδρόμων δρομολόγησης και λογικών πυλών που καταναλώνουν πολύ μεγάλο ποσό ενέργειας. Αυτό το πρόβλημα γίνεται ακόμη πιο κρίσιμο όταν οι FPGAs χρησιμοποιούνται σε εφαρμογές κρυπτογραφίας, μιας και η μεγάλη κατανάλωση ενέργειας κάνει τις εφαρμογές αυτές πιο ευάλωτες σε επιθέσεις ανάλυσης της ενέργειας (Power Analysis Attacks).

Για να αντιμετωπίσουν αυτή την πρόκληση οι σχεδιαστές έιναι αναγκασμένοι, μερικές φορές, να κάνουν ριζοσπαστικές αλλαγές στον σχεδιασμό του

προϊοντος. Αυτές οι αλλαγές μπορούν να περιλαμβάνουν τον διαχωρισμό του αλγορίθμου σε υποκομμάτια υλικού και λογισμικού (Hardware / Software Partitioning), ανακατανομή της λογικής ή ακόμη και επανασχεδίαση όλου του συστήματος. Μιας και αυτές οι αλλαγές έχουν μεγάλο κόστος τόσο σε χρήματα όσο και σε χρόνο, η εκτίμηση της κατανάλωσης της ενέργειας στις FPGAs στα αρχικά στάδια του σχεδιασμού γίνεται όλο και πιο σημαντική.

Σε αυτή την διατριβή ερευνούμε το πρόβλημα της κατανάλωσης ενέργειας στις FPGAs και αξιολογούμε το λογισμικό της εκτίμησης της ενέργειας που παρέχεται απο τους μεγαλύτερους κατασκευαστές αυτών των ολοκληρωμένων. Αξιολογούμε την διαφορά ανάμεσα σε υλικό και λογισμικό για πολλούς κρυπτογραφικούς αλγορίθμους σε σχέση με την κατανάλωση ενέργειας και την ταχύτητα εκτέλεσης τους. Ένα άλλο κομμάτι αυτής της διατριβής διερευνά τη μείωση της κατανάλωσης ενέργειας που μπορούμε να επιτύχουμε είτε με την χρήση της τεχνικής του Clock-Gating είτε της τεχνικής του επαναπρογραμματισμού σε πραγματικό χρόνο.

Συνεχίζουμε με την βαθύτερη διερεύνηση της κρυπτογραφίας και των αλγορίθμων ασφαλείας με την πρόταση και την υλοποίηση γρηγορότερων αλγορίθμων για την παραγοντοποίηση μεγάλων αριθμών. Διαχωρίζουμε τους αλγορίθμους σε λογισμικό και υλικό με τέτοιο τρόπο ώστε να έχουμε το βέλτιστο δυνατό αποτέλεσμα. Τέλος, χρησιμοποιούμε την δυνατότητα επαναπρογραμματισμού σε πραγματικό χρόνο των ολοκληρωμένων κυκλωμάτων FPGA ώστε να επιτύχουμε ακόμη πιο μεγάλες επιταχύνσεις.

ABSTRACT

In the industrial market, designers have a significant incentive to get their products to market quickly: to maximize revenue and time-in-market. Every week that a product is not being sold represents lost revenue, increases the product's market risk and lowers the chance of success. Field Programmable Gate Arrays (FPGAs) offer a low-risk, quick time-to-market solution that industrial designers can easily modify when they need to make changes, fix bugs or create product derivatives at some point in the future.

However, power consumption is one of the main disadvantages of FPGAs. The post-fabrication flexibility provided by these devices is implemented using a large number of pre-fabricated routing tracks and programmable switches that consume a significant amount of power. This problem becomes even more critical when FPGAs are used for applications related to cryptography, since this high power consumption makes potential security applications more vulnerable to power analysis attacks.

Facing this challenge the designers are sometimes forced to make radical changes to the design. These changes could include the partitioning of the algorithm into different hardware and software blocks, re-allocation of the FPGA logic or even re-designing the whole system from scratch. Since these options have a huge impact on time and money, early phase power estimations on FPGA devices are becoming more and more important.

In this dissertation we investigate the power consumption problem and we evaluate the power estimation software that is being provided by the major FPGA vendors. We evaluate the difference between software and hardware cryptography blocks related to their power consumption and processing speed. Another part of this thesis also investigates the power consumption reduction

that we can achieve by either using clock gating or Real-Time reconfiguration.

We continue by exploring deeper the cryptography and the security algorithms by proposing and implementing faster large number factorization algorithms. We partition the algorithms in both software and hardware in such a way that we can achieve the best possible performance. Finally, we use the Real-Time reconfiguration capability of the state of the art FPGA devices in order to achieve even higher speed-ups.

Acknowledgements

This work is part of the 03ED851 research project, implemented within the framework of the "Reinforcement Programme of Human Research Manpower" (P.E.N.E.D) and co-financed by National and Community Funds (25% from the Greek Ministry of Development-General Secretariat of Research and Technology and 75% from E.U.-European Social Fund)

Στους γονείς μου,
για όλες τις θυσίες
που κάνανε ολα
αυτά τα χρόνια
για να με στηρίξουν
στις σπουδές μου.

To my parents
for all the sucrifices
they did in order
to support me
all these years
with my studies.

CONTENTS

1 Introduction **19**

1.1 Motivation 19

1.2 Contribution 21

1.3 Structure 22

2 Security Algorithms and Power Consuption **25**

2.1 Power Consumption 25

2.2 The need for Power Estimation 26

2.3 Cryptography 29

2.3.1 Type of Cryptographic Algorithms 30

2.4 Power Consumption and Cryptography 35

2.4.1 Power Analysis Attacks 36

2.4.2 Preventing Power Analysis Attacks 37

3 Related Work **39**

3.1 Power Measurements on FPGAs 39

3.2 Utilizing Field Programmable Gate Arrays 40

3.3 Algorithms for a Specific Factor Type 41

3.4 General Purpose Algorithms 42

4 Power Estimation and Measurements **43**

4.1 Security Algorithms . . . 43
4.2 Experimental Framework . . . 45
4.3 Hardware and Software Power Measurements . . . 47
4.3.1 Hardware . . . 47
4.3.2 Software . . . 49
4.4 Power Estimation . . . 51
4.5 Power vs Utilization . . . 53
4.6 Clock Gating vs Reconfiguration . . . 56
4.7 Results and Evaluation . . . 58

5 Cryptography and Large Number factorization 67
5.1 Background . . . 67
5.2 Algorithms for a Specific Factor Type . . . 69
5.2.1 Pollard (rho - 1) Algorithm . . . 69
5.2.2 Architecture . . . 70
5.2.3 Hardware . . . 70
5.2.4 Software . . . 75
5.2.5 Experimental Framework . . . 75
5.2.6 Measuring H/W Time . . . 75
5.2.7 Measuring Software Time . . . 77
5.2.8 Results and Evaluation . . . 77
5.3 General Purpose Algorithms . . . 79
5.3.1 Number Field Shieve . . . 79
5.3.2 Background . . . 79
5.3.3 Main Algorithm . . . 80
5.3.4 Sub-Matrix Computation . . . 81
5.3.5 Design . . . 82
5.3.6 Implementation and Methodology . . . 88

5.3.7 Results and Evaluation 91

6 Evaluation and Conclusions **95**

6.1 Evaluation 95

6.2 Future Work 97

6.3 Conclusions 98

6.3.1 Acknowledgements 99

7 Appendix **107**

7.1 List of Publications 107

7.1.1 Within the context of the Ph.D research 107

7.1.2 Other Publications 107

LIST OF FIGURES

4.1 Digilent Spartan-3E and XUP Board 45

4.2 Altera DE2 Board 46

4.3 Measurement Configuration 46

4.4 Operational Amplifier Circuit 47

4.5 Integral Formula 50

4.6 Oscilloscope Waveform for AES 51

4.7 Edited Waveform for AES 52

4.8 Power Estimation Flow 52

4.9 Watts per Utilization 55

4.10 Consumption per 1K in Spartan-3 Devices 56

4.11 Consumption per 1K 57

4.12 Watts per internal cores 58

4.13 Percentance of power reduction 59

4.14 SW execution time 60

4.15 SW Power Consume per 32bit Data Enc 61

4.16 HW Energy Consume per 32bit Data Enc 62

4.17 SW/HW Energy Consume per 32bit Data Enc 63

4.18 SW/HW Time consume per 32bit Data Enc 63

4.19 Estimation & Measurement Results 64

4.20 Difference in mW 64

4.21 Error Percentage 65

5.1 Great Common Divisor 71

5.2 Left 2 Right Binary Exponantiation 73

5.3 Pollard Algorithm 73

5.4 Scheduler Block Diagram 74

5.5 Running Time 78

5.6 Circuit of matrix $A_{6\times6}$ 81

5.7 Virtex-5 Development Board 83

5.8 Design Architecture 84

5.9 Detailed Design Example 86

5.10 Bus Macro Example 90

5.11 Bus Macro Direction 91

5.12 Planahead Floorplaning on a V4 92

5.13 Days per Number of Chips used 94

LIST OF TABLES

4.1 Measured Values in mW 48

4.2 True Measured Values 50

4.3 Estimated Values in mW 53

4.4 Virtex-II Pro Measured Values 54

4.5 Virtex-4 Measured Values 54

4.6 Spartan-3 Measured Values 54

4.7 Time for a 32bit encode 58

5.1 Number of parallel cores used 76

5.2 Real and Cropped Frequency Numbers (MHz) 76

5.3 Time in Miliseconds for S/W and H/W 78

5.4 Design Variables 92

5.5 Speedup Compared to Mesh Design 93

CHAPTER

ONE

INTRODUCTION

1.1 Motivation

In the industrial market, designers have a significant incentive to get their products to market quickly: to maximize revenue and time-in-market. Every week that a product is not being sold represents lost revenue, increases the product's market risk and lowers the chance of success.

Using Field Programmable Gate Arrays (FPGAs), designers can develop a custom solution without the nonrecurring engineering (NRE) charges or the fabrication and assembly time delays typically associated with Application Specific Integrated Circuits (ASICs). Designers do not have to deal with the unwanted functionality provided by application specific standard products or the inevitable silicon spins associated with ASICs.

FPGAs offer a low-risk, quick time-to-market solution that industrial designers can easily modify when they need to make changes, fix bugs or create product derivatives at some point in the future. However, power consumption is one of the main disadvantages of FPGAs. The post-fabrication flexibility provided by these devices is implemented using a large number of pre-fabricated routing tracks and programmable switches. These tracks can be long, and can consume a significant amount of power every time they switch. In

addition, the programmable switches add capacitance to each track; this further increases the power dissipation of the FPGAs.

Recent advances in semiconductor process technology have led to rapid scaling of transistor dimensions, allowing a large number of them to be packed on the same chip. Field Programmable Devices (FPDs), which consist of a higher number of transistors compared to their alternative ASICs, have also enjoyed a rapid growth due to these technology advancements. High density of transistors on the same chip has made power consumption one of the major challenges of deep sub micron IC design [1].

On the other hand, low power devices, need a quick, yet powerful, encryption/decryption mechanism. But those encryption/authentication algorithms need high processing power and/or have large execution times and thus they consume significant amounts of energy. For battery-powered embedded systems, perhaps one of the foremost challenges is the mismatch between the (energy and performance) requirements of security processing and the available battery and processor capabilities. A common solution for time critical or time consuming software applications, such as in security applications, is a hardware co-processor; through the last ten years, more and more industries have developed custom ICs to maximize speed for various end-user products.

By doing so, a new problem arise for the designer: By adding this extra hardware to a device, the total power consumption of the product is increased. Many techniques have been introduced through the years, with the most common solution being to power down the co-processor IC when not in use.

Power consumption off FPGAs is not as straight-forward as on the co-processor technique, since both dedicated hardware modules and processing cores co-exist on the same die. The theory that software is, in general, slower but it consumes less power does not apply in the case of the FPGAs. Another very importand factor is that high power consumption makes security applications more vulnerable to power analysis attacks. Power analysis is an important and efficient mean for side channel attacks on security applications. By monitoring devices' power consumption during various operations it is possible

to collect security information (e.g. the encryption key).

If the information leakage can be directly observed it is called an SPA attack (Simple Power Analysis). On the other hand, if it is necessary to use statistical methods to analyze the information leakage, it is called a DPA attack (Differential Power analysis) [2],[3]. In fact, the DPA needs to compute one mean on a large number of power consumption samples to establish correlation between the data being manipulated (and depending on few key bits) and the information leakage. This family of attacks supposes that there is an observable difference in the power consumption when a bit is set or clear.

Since FPGAs consume more power than custom ICs, they are even more vulnurable to these power analysis techniques. So, it is crucial for the designer not only to be able to partition correctly the application to software and hardware parts for minimizing power but also to be able to estimate with high accuracy the power consumption of the FPGA chip.

1.2 Contribution

The main contributions of this thesis can be summarized as follows.

- This is the only work, to the best of my knowledge, that compares the actual power consumption between the software and hardware implementations of security algorithms that are all executed on the same state-of-the-art FPGA fabric, which already uses advanced process technology with reduced power supply.

- I analyze the accuracy of the two most-widely used power estimation tools, in the FPGA domain, when applied to security modules. In particular, I demonstrate that even those tools cannot estimate the power consumption of the state-of-the-art reconfigurable devices accurately; they are always pessimistic and our results demonstrate that they overestimate the power consumption by even more than 200%.

- I explore deeper into cryptography and it's aspects by designing and presenting a new scalable FPGA implementation of Pollard's $(\rho - 1)$ special factorization algorithm. The design is faster than software implementations and results in a speedup ranging from 20 up to 231.

- I present another important advantage of run-time reconfiguration : reduction in overall power consumption. For example ,a high-throughput, fast and power consuming FPGA die could be reprogrammed with a low logic-cell count design, if the demands on high throughput are low on a certain time period. This would leave most of the FPGA fabric unprogrammed, thus reducing static and dynamic power consumption. Another and easier way of achieving low power consumption on this scenario could be via clock-gating. So, a rational question could be: What is better? Clock gating or run-time reconfiguration? Do we consume less power by using clock gating on a design or by reconfiguring the device with a slower and smaller design?

- Finally, I propose and implement a new architecture for a time efficient matrix-by-vector multiplication used on the Algebra step of the Number Field Shieve factorization algorithm. This architecture is based on two main characteristics of the linear algebra matrix. I have also explained and analyzed why my system can achieve a $19,4\times$ to $57,4\times$ speedup, compared to other proposed architectures, depending on the parameters of the block Wiedemann algorithm and on the FPGA device used.

1.3 Structure

The dissertation is structured as follows:

Chapter 2 is devoted into the introduction of the challenges of power consumption. Why is there a need of power estimation on the early design process of an electronic product? We also include an introduction to the security and the cryptographic

algorithms that will be used further in this dissertation. Finally, we introduce some of the attacks to the cryptographic modules related to the power consumption.

Chapter 3 summarizes the related work that has been done on all the sectors of this dissertation. We describe the different kind of research that has been done on real time reconfiguration and large number factorization. Finally we present with a time line the work that has been done on power estimation and Field Programmable Gate Arrays.

Chapter 4 is devoted on our research on the comparison of the power estimation and the actual power consumption on FPGA devices. We explore the properties of cryptographic software that is executed on embedded processors on the FPGA devices and we compare our findings. We also look deeper on the power consumtion on an FPGA device in comparison to clock gating and real time reconfigurations. Finally we present our results and we evaluate them.

Chapter 5 In this chapter we enter the world of cryptography and large number factorization. We explore two different problems. One problem is the efficient implementation of an Specific Factor Type algorithm on reconfigurable logic and its comparison with software blocks. The second problem is the implementation of an efficient matrix multiplication for the Number Field Sieve (General Purpose Algorithm) with the use of real time reconfiguration. We present our implementation and we evaluate our work and the measured data.

Chapter 6 In this chapter we summarize our evaluation findings giving the reader a more structured visualization of our work. Finaly we present our conclusions and we propose future work that can be done within the borders of this disseration.

CHAPTER

TWO

SECURITY ALGORITHMS AND POWER CONSUPTION

2.1 Power Consumption

More than a quarter century ago, Gordon Moore forecast the rapid pace of technology innovation. His prediction, popularly known as «Moore's Law», states that transistor density on integrated circuits doubles every two years. Today, semiconductor companies continue to apply the principles of «Moore's Law», achieving higher levels of integration and producing a steady stream of smaller, faster, cheaper chips, bringing exponential growth in computing and communications technology to consumers and businesses worldwide.

As computing and communications converge and pervade nearly every aspect of life, the demand increases for devices with more functionality, faster operation, and lower costs. Keeping pace with Moore's Law is essential to help computing and communications industries deliver chips that meet these needs.

However, as more transistors are integrated on a chip to enable more functions, and higher frequency is used to obtain increased performance, the total power consumption increases and this results in more heat. As more transistors are packed into a smaller area, the power density also increases. Consumers desire for mobility and multifunctional

small form factor devices places additional challenges like efficient battery operation.

Efficient power and thermal management are vital as systems become smaller or more capable with every generation of «Moore's Law». Power must be delivered and used efficiently by the chips, wiring, and display, while effectively dissipating the heat from the system – economically, of course.

Addressing the power challenge allows a continuation of the trend toward smaller, faster, cost effective chips and devices. As a result, futuristic applications that require more powerful processors may be realized. Increasing capability and density of computing and communications chips may be sustained. Comprehensive power and thermal management techniques are a fundamental part of continuing to receive the benefits of Moore's Law.

In fewer words, today's semiconductors are smaller, run faster, process more information and, as a result, generate more heat. Removing this heat, generated by current flowing in transistors and the connections between them, is a key challenge in today's electronics products and systems. Achieving low thermal resistance from the silicon chip to the external environment ensures a minimum increase in the temperature of the chips, thereby maximizing product reliability and lifetime. As a result, the ongoing performance improvements in semiconductor devices require the continuing development of improved thermal management solutions.

2.2 The need for Power Estimation

IC designers today are facing continuous challenges in balancing design performance and power consumption. This task is becoming more critical as designs grow larger and more complex and process geometries shrink to 90-nm and below. Designers must meet these challenges by finding the right formula for optimizing power without sacrificing performance. FPGAs provide the performance and features designers need, but suffer due to higher power consumption.

Designing for low power in programmable logic is important to the overall design equation for different end markets. The communications industry is a more mature FPGA market in this respect, with programmable logic used heavily in routers, switches, base stations and storage servers. These newer generation products demand larger and higher-performing FPGAs. Engineers are continuing to push the performance envelope by migrating more functionality into FPGAs, replacing ASICs and ASSPs (Application Specific Standard Products), while demanding higher system performance. Although the newest 28-nm-generation FPGAs consume less power than previous 90-nm products, FPGAs still consume a large portion of total system power. Engineers today design into tighter and more enclosed spaces, making it difficult to improve airflow and install proper size heat sinks and indeed, despite lowering total power consumption, power density increases. Thus thermal management, and consequently power management, continues to be an important topic in high-end FPGA design.

Designing for the low-power marketplace is not a trivial task. Engineers use numerous techniques to reduce power in FPGA designs. Various types of FPGAs, different design methodologies, numerous intellectual property (IP) cores, assorted system design methods, diverse software algorithms and power tools all contribute to power used in a design.

The two primary sources of power consumption in FPGAs are:

- Static power dissipation due to leakage currents during device standby. Using finer semiconductor process geometries (specifically the 28-nm geometry) has caused an increase in static power consumption in FPGAs. As transistor size shrinks and lower voltages are utilized, a greater sub-threshold leakage current occurs in the transistor channel when the transistor is in the off state. Consequently, static power consumption rises when using the 28-nm process.

- Dynamic power dissipation during charging and discharging of internal capacitances in the logic array and interconnect networks of an active device. Dynamic power is

affected in two ways by process scaling. First, the use of smaller feature sizes and lower voltages significantly reduces dynamic power consumption. However, higher device operating frequencies are possible in 28-nm technology, and, since dynamic power increases with operating frequency, designs that make full use of the speed of 28-nm technology will see less reduction in dynamic power.

As a result of the expected blanket increase in power consumption for smaller process geometries, semiconductor manufacturers use various techniques to optimize power consumption, both in the dynamic and static domains.

Having accurate power estimation tools that are quick and easy to use allows designers to realistically hit power budget numbers and incrementally improve designs efficiently. Without realistic data, both in early power estimation tools and data sheets, the design phase will be seriously hindered. Obtaining early estimation tools in spreadsheet form allows designers to gather early estimates of power requirements before starting the design phase.

As the design progresses, designers can load placed-and-routed designs into power estimator programs and receive more accurate power consumption estimates. The best tools allow simulation files to be integrated seamlessly into the power tools, acquiring an accurate representation of switching power. If a simulation has not been done, the power analysis tools will intelligently estimate toggling within the design. Altera's Quartus II PowerPlay Power Analyzer tool as well as Xilinx's XPower estimator utilize design signal activity information and other important design factors affecting power consumption to accurately estimate design power.

These power estimator tools require the design to be fully compiled in order to extract the target device as well as place and route information of the device. When combined with signal activity information and operating conditions of the device comprehensive power consumption reports are produced. These reports facilitate both thermal and power supply planning requirements. Additionally, these reports also pinpoint which device structures and even design hierarchy blocks are dissipating the most thermal power,

thus enabling design decisions that reduce power consumption. This provides very high quality power estimates and benchmarking data to be obtained using these tools with an estimated accuracy within 20 percent of device measurements.

2.3 Cryptography

Cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the Internet.

Within the context of any application-to-application communication, there are some specific security requirements, including:

- Authentication: The process of proving one's identity. (The primary forms of host-to-host authentication on the Internet today are name-based or address-based, both of which are notoriously weak.)
- Privacy/confidentiality: Ensuring that no one can read the message except the intended receiver.
- Integrity: Assuring the receiver that the received message has not been altered in any way from the original.
- Non-repudiation: A mechanism to prove that the sender really sent this message.

Cryptography, then, not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes

typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions, each of which is described below. In all cases, the initial unencrypted data is referred to as plaintext. It is encrypted into ciphertext, which will in turn (usually) be decrypted into usable plaintext.

In many of the descriptions below, two communicating parties will be referred to as Alice and Bob; this is the common nomenclature in the crypto field and literature to make it easier to identify the communicating parties.

2.3.1 Type of Cryptographic Algorithms

There are several ways of classifying cryptographic algorithms. For purposes of this thesis, they will be categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms that will be discussed are:

- Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption
- Public Key Cryptography (PKC): Uses one key for encryption and another for decryption
- Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information

Secret Key Cryptography

With secret key cryptography, a single key is used for both encryption and decryption. The sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key (or ruleset) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption.

With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

Secret key cryptography schemes are generally categorized as being either stream ciphers or block ciphers. Stream ciphers operate on a single bit (byte or computer word) at a time and implement some form of feedback mechanism so that the key is constantly changing. A block cipher is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will always encrypt to the same ciphertext when using the same key in a block cipher whereas the same plaintext will encrypt to different ciphertext in a stream cipher.

Stream ciphers come in several flavors but two are worth mentioning here. Self-synchronizing stream ciphers calculate each bit in the keystream as a function of the previous n bits in the keystream. It is termed "self-synchronizing" because the decryption process can stay synchronized with the encryption process merely by knowing how far into the n-bit keystream it is. One problem is error propagation; a garbled bit in transmission will result in n garbled bits at the receiving side. Synchronous stream ciphers generate the keystream in a fashion independent of the message stream but by using the same keystream generation function at sender and receiver. While stream ciphers do not propagate transmission errors, they are, by their nature, periodic so that the keystream will eventually repeat.

Some known Secret Key algorithms are:

- AES: The Advanced Encryption Standard (AES) [4]
- The Data Encryption Standard (DES) [5], [6]
- Triple-DES (3DES) [7]

Public Key Cryptography

Public-key cryptography has been said to be the most significant new development in cryptography in the last 300-400 years. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

PKC depends upon the existence of so-called one-way functions, or mathematical functions that are easy to computer whereas their inverse function is relatively difficult to compute. We can see two simple examples, below:

1. Multiplication vs. factorization: Suppose I tell you that I have two numbers, 9 and 16, and that I want to calculate the product; it should take almost no time to calculate the product, 144. Suppose instead that I tell you that I have a number, 144, and I need you tell me which pair of integers I multiplied together to obtain that number. You will eventually come up with the solution but whereas calculating the product took milliseconds, factoring will take longer because you first need to find the 8 pairs of integer factors and then determine which one is the correct pair.

2. Exponentiation vs. logarithms: Suppose I tell you that I want to take the number 3 to the 6th power; again, it is easy to calculate $3^6 = 729$. But if I tell you that I have the number 729 and want you to tell me the two integers that I used, x and y so that $log_x(729) = y$, it will take you longer to find all possible solutions and select the pair that I used.

While the examples above are trivial, they do represent two of the functional pairs that are used with PKC; namely, the ease of multiplication and exponentiation versus the relative difficulty of factoring and calculating logarithms, respectively. The mathematical "trick" in PKC is to find a trap door in the one-way function so that the inverse calculation becomes easy given knowledge of some item of information. (The problem is further

exacerbated because the algorithms don't use just any old integers, but very large prime numbers.)

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work. Because a pair of keys are required, this approach is also called asymmetric cryptography.

In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straight forward to send messages under this scheme. Suppose Alice wants to send Bob a message. Alice encrypts some information using Bob's public key; Bob decrypts the ciphertext using his private key. This method could be also used to prove who sent a message; Alice, for example, could encrypt some plaintext with her private key; when Bob decrypts using Alice's public key, he knows that Alice sent the message and Alice cannot deny having sent the message (non-repudiation).

The most popular Public-key cryptography algorithm used today is the R.S.A [8], [9], [10]. This PKC implementation, named for the three MIT mathematicians who developed it — Ronald Rivest, Adi Shamir, and Leonard Adleman. RSA today is used in hundreds of software products and can be used for key exchange, digital signatures, or encryption of small blocks of data. RSA uses a variable size encryption block and a variable size key.

The key-pair is derived from a very large number, n, that is the product of two prime numbers chosen according to special rules; these primes may be 100 or more digits in length each, yielding an n with roughly twice as many digits as the prime factors. The public key information includes n and a derivative of one of the factors of n; an attacker cannot determine the prime factors of n (and, therefore, the private key) from this information alone and that is what makes the RSA algorithm so secure. (Some descriptions of PKC erroneously state that RSA's safety is due to the difficulty in factoring

large prime numbers. In fact, large prime numbers, like small prime numbers, only have two factors!) The ability for computers to factor large numbers, and therefore attack schemes such as RSA, is rapidly improving and systems today can find the prime factors of numbers with more than 200 digits. Nevertheless, if a large number is created from two prime factors that are roughly the same size, there is no known factorization algorithm that will solve the problem in a reasonable amount of time; a 2005 test to factor a 200-digit number took 1.5 years and over 50 years of compute time (see the Wikipedia article on integer factorization.) Regardless, one presumed protection of RSA is that users can easily increase the key size to always stay ahead of the computer processing curve. As an aside, the patent for RSA expired in September 2000 which does not appear to have affected RSA's popularity one way or the other.

Hash Functions

Hash functions, also called message digests and one-way encryption, are algorithms that, in some sense, use no key. Instead, a fixed-length hash value is computed based upon the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered. Hash algorithms are typically used to provide a digital fingerprint of a file's contents, often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating systems to encrypt passwords. Hash functions, then, provide a measure of the integrity of a file.

Hash algorithms that are in common use today include:

- Message Digest (MD) algorithms: A series of byte-oriented algorithms that produce a 128-bit hash value from an arbitrary-length message [11], [12].
- MD2 (RFC 1319): Designed for systems with limited memory, such as smart cards. (MD2 has been relegated to historical status, per RFC 6149.) [13]
- MD4 (RFC 1320): Developed by Rivest, similar to MD2 but designed specifically for fast processing in software. (MD4 has been relegated to historical status, per

RFC 6150.) [14], [15]

- MD5 (RFC 1321): Also developed by Rivest after potential weaknesses were reported in MD4; this scheme is similar to MD4 but is slower because more manipulation is made to the original data. MD5 has been implemented in a large number of products although several weaknesses in the algorithm were demonstrated by German cryptographer Hans Dobbertin in 1996 ("Cryptanalysis of MD5 Compress") [16], [17].

2.4 Power Consumption and Cryptography

Attacks that involve multiple parts of a security system are diffcult to predict and model. If cipher designers, software developers, and hardware engineers do not understand or review each other's work, security assumptions made at each level of a system's design may be incomplete or unrealistic. As a result, security faults often involve unanticipated interactions between components designed by different people.

Many techniques have been designed for testing cryptographic algorithms in isolation. For example, differential cryptanalysis [18] and linear cryptanalysis [19] can exploit extremely small statistical characteristics in a cipher's inputs and outputs. These methods have been well studied because they can be applied by analyzing only one part of a system's architecture - an algorithm's mathematical structure.

A correct implementation of a strong protocol is not necessarily secure. For example, failures can be caused by defective computations [20], [21] and information leaked during secret key operations. Attacks using timing information [22], [23] as well as data collected using invasive measuring techniques [24], [25] have been demonstrated. The U.S. government has invested considerable resources in the classified TEMPEST program to prevent sensitive information from leaking through electromagnetic emanations.

2.4.1 Power Analysis Attacks

Most modern cryptographic devices are implemented using semiconductor logic gates, which are constructed out of transistors. Electrons flow across the silicon substrate when charge is applied to (or removed from) a transistor's gate, consuming power and producing electromagnetic radiation. To measure a circuit's power consumption, a small (e.g., 50 ohm) resistor is inserted in series with the power or ground input. The voltage difference across the resistor divided by the resistance yields the current. Well-equipped electronics labs have equipment that can digitally sample voltage differences at extraordinarily high rates (over 20 GHz) with excellent accuracy (less than 1% error). Devices capable of sampling at 20MHz or faster and transferring the data to a PC can be bought for less than $400. [26]

Simple Power Analysis (SPA) is a technique that involves directly interpreting power consumption measurements collected during cryptographic operations. SPA can yield information about a device's operation as well as key material. Because SPA can reveal the sequence of instructions executed, it can be used to break cryptographic implementations in which the execution path depends on the data being processed. For example:

- DES key schedule: The DES key schedule computation involves rotating 28-bit key registers. A conditional branch is commonly used to check the bit shifted off the end so that "1" bits can be wrapped around. The resulting power consumption traces for a "1" bit and a "0" bit will contain different SPA features if the execution paths take different branches for each.

- DES permutations: DES implementations perform a variety of bit permutations. Conditional branching in software or microcode can cause signiffcant power consumption differences for "0" and "1" bits.

- Comparisons: String or memory comparison operations typically perform a conditional branch when a mismatch is found. This conditional branching causes large SPA (and sometimes timing) characteristics.

- Multipliers: Modular multiplication circuits tend to leak a great deal of information about the data they process. The leakage functions depend on the multiplier design, but are often strongly correlated to operand values and Hamming weights.

- Exponentiators: A simple modular exponentiation function scans across the exponent, performing a squaring operation in every iteration with an additional multiplication operation for each exponent bit that is equal to "1". The exponent can be compromised if squaring and multiplication operations have different power consumption characteristics, take different amounts of time, or are separated by different code. Modular exponentiation functions that operate on two or more exponent bits at a time may have more complex leakage functions.

2.4.2 Preventing Power Analysis Attacks

Power distribution as well as power consumption of an integrated circuit are really important factors, if we need to keep our data safe. One solution on preventing these power analysis attacks is the early knowledge of the power consumtion distribution of the circuit. Moreover, if low power is achieved on a cryptographic device, Power Analysis attacks will be difficult (if not impossible) to implement. Measuring power variation on a really low power device will be a challenging task, since measure data will be easily confused with noise comming out of the system.

FPGAs on the other hand, are by far a high power consuming device. This makes the low power senario not quite usefull for preventing these kind of attacks. So, an early estimation of the die's power consumption would be really important, since it would provide the engineers an insight of the distribution of the power consumption on the chip. Thus a posibility of re-routing and power re-optimization of the design would be one way of preventing this kind of cryptographic attacks.

CHAPTER

THREE

RELATED WORK

3.1 Power Measurements on FPGAs

Hardware power measurements of large FPGAs has received little attention compared to that of standard cell ASIC, which has been extensively studied in the literature. In particular, only the power consumption when certain matrix multiplication algorithms are implemented in hardware have been presented in [27] and of various digital signal processing modules in [28]. Moreover, Lysecky and Vahid [29] have studied the differences between the performance achieved when certain tasks are executed in the embedded processing cores of a state-of-the-art-FPGA and when their are executed by dedicated hardware modules.

In general, the difference between the energy estimated and the energy actually consumed in FPGAs, has not been studied yet. The only paper which is, in a small sense, related to the work presented here is one by Becker, Huebner and Ullmann [30] which discusses the exact power consumption trade-offs between the measured runtime consumption of a mapped application and the measured reconfiguration time consumption of different dynamically reconfigured applications. However, even this work has no comparisons between the measured and estimated power consumption. On the other hand,

there is considerable amount of work done in both industry and academia related to cryptographic algorithms and their performance evaluation. In the literature, there are several implementations of the three most widely used cryptographic algorithms (DES, AES and MD5) in either software, or hardware; the hardware approaches are tailored to both ASICs and FPGAs [31],[32], [33], [34], [35].

Extensive work has also been done on the investigation of various timing analysis algorithms [36], [37], [38], [39]. Research showed that countermeasures sould be considered for preventing these kind of attacks. Paul Kocher [2] proposed three techniques for preventing DPA and related attacks. The first approach was to reduce signal sizes and physically shield the device. The second approach involved introducing noise into power consumption measurements and the third and final approach involved designing cryptosystems with realistic assumptions about the underlying hardware. In most of the above cases the countermeasures for the DPA attacks need accurate and trustworthy power estimation tools. Power drop-off counter effects could be reduced or even eliminated, if accurate power estimations could be made on the early design cycles.

3.2 Utilizing Field Programmable Gate Arrays

There exist a number of implementations of factorization algorithms and the most efficient versions use parallel computing. To the best of the authors acknowledge, until 1999 there was not any hardware implementation of number factoring.

The first approach named TWINKLE was presented by A. Shamir [40]. Another hardware approach was presented by H.J.Kim and W.H.M.Smith. They implemented the MPQS algorithm on an FPGA [41]. X. Wang and S. Ziavras implemented a parallel LU factorization of Sparse Matrises on FPGA-based configurable computing engines [42]. J. Franke and T. Kleinjung proposed a hardware architecture for factorization with the Elliptic Curve Method [43].

The latest works were presented by T. Izu and J. Kogure who implemented a sieving

step with the lattice sieving [44], and G. Southern and C. Mason who implemented a high throughput circuit for trial division by small primes [45]. Both systems were implemented on FPGAs.

3.3 Algorithms for a Specific Factor Type

Power consumption measurements of FPGA devices were reported by Becker *et al* in [30]. In an earlier work, we have also presented measurements on power consumption of cryptographic designs on Virtex-II Pro devices.

A lot of work has also been done on the analysis and measurements of the dynamic power consumption of FPGA devices. Shang *et al* [46] presented an analysis of dynamic power dissipation in Virtex-II devices. They report that most of the dynamic power dissipation in an FPGA fabric is due to the programmable interconnects and clocking resources.

In [47] Wilton *et al.* demonstrate reductions in power consumption by increasing the number of pipeline stages in an FPGA design. However, none of these approaches use run-time reconfiguration in order to reduce the power consumption.

Noguera, in [48], proposes the use of adaptive partial reconfiguration for reducing the power consumption. However, the results demonstrated do not prove this claim, whereas (a) they do not compare their results with simple clock gating (b) they focus on a specific FPGA device and (c) they just use a certain networking module provided by the FPGA vendor. Another interesting work was presented by Boemo *et al.* [49] who used reconfiguration in order to monitor the temperature of the FPGA die.

In general, none of these research efforts analyzed the effect of the migration to new and advanced processing technologies by the semiconductor companies. Also, none has ever measured the difference of the power reduction than can be achieved by using clock gating on the design, versus the power reduction achieved when utilizing run-time reconfiguration.

3.4 General Purpose Algorithms

Much work has been done on the matrix step of the Number field sieve. In 2001, Daniel J. Bernstein observed that the NFS linear algebra step has a huge input that is accessed repeatedly. When implemented on traditional computers, very few processors are accessing that huge input, which is inherently inefficient. He proposed an implementation using mesh sorting, which reduces the asymptotic cost and the design was given on an abstract layer [50]. In 2002, Arjen K. Lenstra and Adi Shamir, proposed an alternative concrete high-level parallel design, wafer-scale electronic device consisting of an array of cells, executing mesh routing [51].

In 2003, Willi Geiselmann improved Lenstra's design by adding splitting into smaller and semi-independent chips. This variant was prototyped on FPGA. In 2004, Sashisu Bajracharya implemented the improved Lenstras design on an FPGA [52]. Final, in 2005 Willi Geiselmann proposed a highly-parallel electronic device, based on a pipelined architecture. More efficient than the above mesh-based devices, and more technologically feasible due to smaller chips and better scalability [53].

CHAPTER

FOUR

POWER ESTIMATION AND MEASUREMENTS

4.1 Security Algorithms

Our four chosen security algorithms, AES [54], DES, 3DES [55] and MD5 hash [56], comply with todays mobile needs. All algorithms are compact, powerful and widely used in todays mobile market.

- The Data Encryption Standard (DES) is a cipher selected as an official Federal Information Processing Standard (FIPS) for the United States in 1976, and which has subsequently enjoyed widespread use internationally. The algorithm was initially controversial, with classified design elements, a relatively short key length, and suspicions about a National Security Agency (NSA) back-door. DES consequently came under intense academic scrutiny, and motivated the modern understanding of block ciphers and their cryptanalysis.

  The most common SKC scheme used today, DES was designed by IBM in the 1970s and adopted by the National Bureau of Standards (NBS) [now the National Institute for Standards and Technology (NIST)] in 1977 for commercial and unclassified government applications. DES is a block-cipher employing a 56-bit key that operates on 64-bit blocks. DES has a complex set of rules and transformations that

were designed specifically to yield fast hardware implementations and slow software implementations, although this latter point is becoming less significant today since the speed of computer processors is several orders of magnitude faster today than twenty years ago. IBM also proposed a 112-bit key for DES, which was rejected at the time by the government; the use of 112-bit keys was considered in the 1990s, however, conversion was never seriously considered.

- In 1997, NIST initiated a very public, 4-1/2 year process to develop a new secure cryptosystem for U.S. government applications. The result, the Advanced Encryption Standard, became the official successor to DES in December 2001. AES uses an SKC scheme called Rijndael, a block cipher designed by Belgian cryptographers Joan Daemen and Vincent Rijmen. The algorithm can use a variable block length and key length; the latest specification allowed any combination of keys lengths of 128, 192, or 256 bits and blocks of length 128, 192, or 256 bits. NIST initially selected Rijndael in October 2000 and formal adoption as the AES standard came in December 2001. FIPS PUB 197 describes a 128-bit block cipher employing a 128-, 192-, or 256-bit key. As of 2006, AES is one of the most popular algorithms used in symmetric key cryptography.

- Triple DES is also known as TDES or, more standard, TDEA (Triple Data Encryption Alg), is a block cipher formed from the Data Encryption Standard (DES) cipher by using it three times. A variant of DES that employs up to three 56-bit keys and makes three encryption/decryption passes over the block; 3DES is also described in FIPS 46-3 and is the recommended replacement to DES.

- Finally, MD5 (Message-Digest algorithm 5) is a widely-used cryptographic hash function with a 128-bit hash value. As an Internet standard (RFC 1321), MD5 has been employed in a wide variety of security applications, and is also commonly used to check the integrity of files.

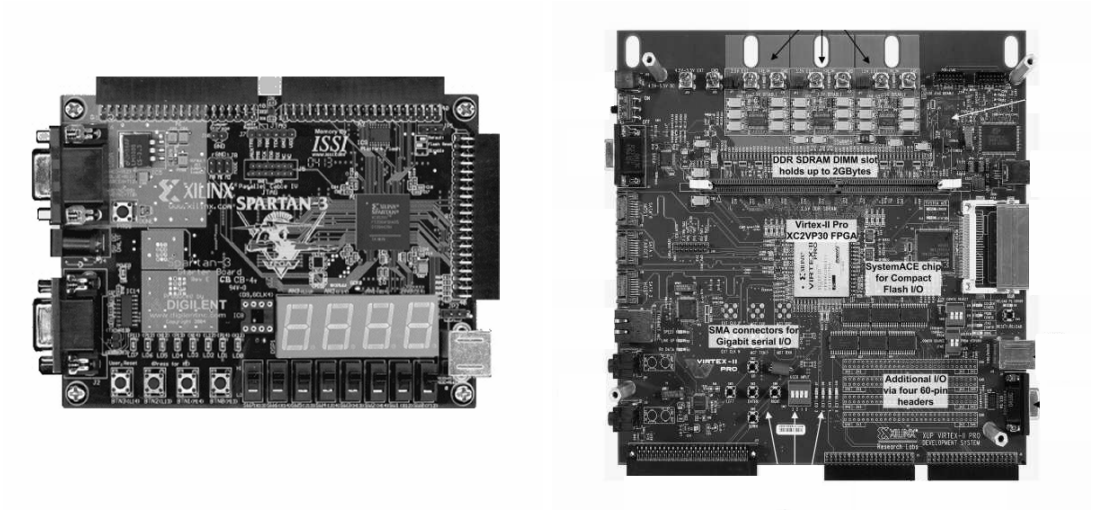

Figure 4.1: Digilent Spartan-3E and XUP Board

4.2 Experimental Framework

The equipment used for our experiments are: a Digilent XUP Development board [57] (figure 4.1) with a Xilinx Virtex-II Pro FPGA, a Digilent Spartan-3 Development board (figure 4.1) with a Spartan3 FPGA and an Altera DE2 Board [58] (figure 4.2) with a Cyclone-II FPGA.

Measuring the FPGA's internal core power consumption, can be a challenging procedure, especially if the board in which the experiments are carried out on, is not specifically designed for such experiments. In order to measure accurately the power consumption we had to isolate the board's internal logic voltage, (in most cases 1.2 Volts), and drive it by an external regulated DC power supply. An all boards, except the Digilent XUP, we had to make hardware changes so that we could be able to do this.

Since internal core logic current consumption was quite low, we could not use a plain amperometer. Moreover, current consumption was not always stable throughout the experiment. So, we had to insert a 0.5Ohm (1%, 5Watt) resistor on the returning path of the internal core logic power network. By calculating the voltage difference between the two poles of the resistor we could measure the exact current consumption of the FPGA chip. The exact experiment setup, can be seen in figure 4.3.

It sould be noted, that on all three boards, the internal logic core voltage network,

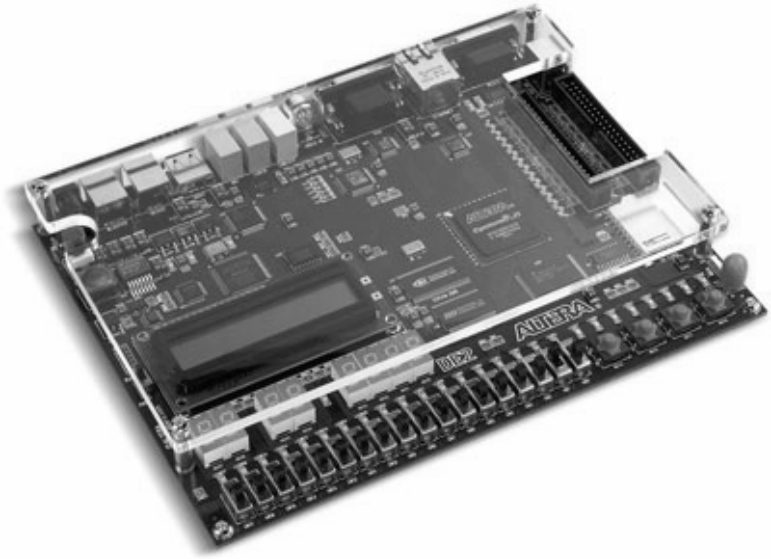

Figure 4.2: Altera DE2 Board

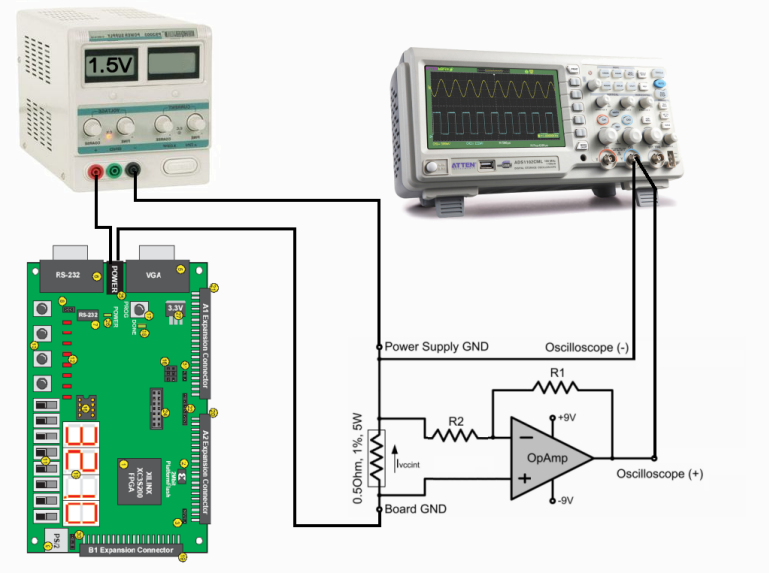

Figure 4.3: Measurement Configuration

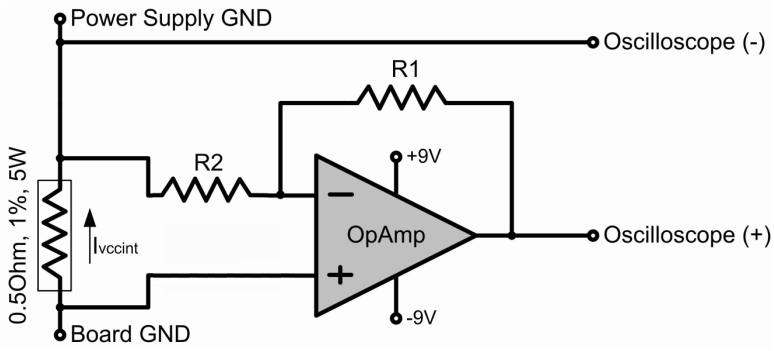

Figure 4.4: Operational Amplifier Circuit

was only connected with the FPGA chip and not with any other external sources. This quaranties that the measured current is exactly the current consumed by the FPGA's internal logic core.

Since the voltage difference on the resistor's pins was quite small, we had to amplify the measured voltage difference by implementing an amplification circuit as demonstrated in figure 4.4. This circuit amplifies by a factor of 23 the measured ΔV value. All resistors, had a small tolerance of 1%. Finaly, a highly advanced mixed signal oscilloscope MSO6052A, of Agilent Technologies [59], was used for sampling the operational amplifier output.

4.3 Hardware and Software Power Measurements

4.3.1 Hardware

The security cores used, were not unified. Each module had a different bus architecture and a different input and output logic. To be able to get accurate results, a top HDL file was inserted on all four algorithms. The inputs to the HDL file, were only the clock and the reset signal, and an internal FSM controlled all the security core signals. An endless loop of reading and writing data to the core was implemented, providing a high predictability rate for all the core signals and constant power consumption. It should be noted that although we could put a large number of hardware modules working in parallel,

	DES	AES	$3DES$	$MD5$
$Virtex-IIPro$	254.89	915.21	300.32	571.12
$Spartan-3$	38.19	135.23	43.04	52.69
$Cyclone-II$	131.68	193.04	135.65	144.83

Table 4.1: Measured Values in mW

we would prefer not to imply such parallelism so as to be more fair when comparing the performance and the power consumption between the hardware implementations and the software ones (i.e. there is no parallelism in the software case).

Also, the execution time of the each crypto module may be different. The crypto module will return a signal to main controller, when the en/decrypt operations are finished. The top module FSM, will send data to the crypto module and will wait until the operation is finished. Then, data is read, new values are inserted on the input register, and the proper sequence for starting the en/decrypt operation is made. The bus architecture for each core is different, so different top modules are used in each algorithm. For example, AES and MD5 cores use a 128bit bus for data switching and DES/3DES use a 64bit bus.

Due to the fact that the cores were consuming the same amount of power throughout the experiment, the measurement was really straight forward and extremely accurate. All measurements were repeated three times for maximum accuracy. Mean values of the measurements are displayed on table 4.1, while the variance in the power consumption numbers, in the different experiments was insignificant (up to 2%).

By starting the measurement, all security cores, are imported on Xilinx ISE tool for placement and routing, until we get the programming bit-stream for the Virtex-II Pro FPGA. Then, the programming software configures the FPGA with the selected security core, over the USB cable. A manual reset is initiated on the development board and the core enters an endless loop.

The current information shown by the oscilloscope, remains stable, making it possible for error free current measurement, simply by reading the voltage value of the oscilloscope trace and by diving it by 23.

Then, the Modelsim SE by Mentor Graphics [60] was used to measure the number of hardware cycles needed for each 32bit data encryption. So, by using the formula

$$E = V \times I \times t$$

we could calculate the true Energy that was used for an 32bit encryption for each algorithm.

Finally, all measurements, which include the power consumption of the total hardware resources used (e.g. including the top-level control FSM) were saved for latter comparison.

4.3.2 Software

Firstly, it should be stated that the embedded CPU in our experimental set-up is a PowerPC which is well know for its high-performance and relatively low power consumption. In particular is has been claimed that «The PowerPC sets the new standard in terms of performance, low power, security and multi-platform support, making it ideal for next generation 801.11n and WiMAX applications» [61]

Measuring the Software Energy Consumption, was a more complicated procedure since we didn't have the ability to measure the clock cycles needed for each 32bit data encrypted as we can do with the hardware modules.

One initial problem was that no matter what software code was executed in the embedded CPU, the power consumption stayed the same. Although we inserted an infinite loop with a NOOP command in our source code, the measured power consumption that was associated with the 1.5V supply (i.e. that is the supply connected only to the embedded CPU) was still the same. Thus we could not calculate the actual real processing time of the various software schemes. In order to overcome the above, we have used the power down command of the PowerPC processor; at the end of our program, we altered a special control register, so that the processor would come to its power down state, thus

	DES	AES	$3DES$	$MD5$
$S/WTime(Seconds)$	$0,0000347$	$0,0000123$	$0,0000872$	$0,000008$
$H/WTime(Seconds)$	$0,00000036$	$0,00000048$	$0,00000038$	$0,00000032$
$S/WEnergy(Joules)$	$0,000958883$	$0,000208086$	$0,000435148$	$0,000160518$
$H/WEnergy(Joules)$	$0,918 \times 10^{-07}$	$4,392 \times 10^{-07}$	$0,114 \times 10^{-07}$	$1,824 \times 10^{-07}$

Table 4.2: True Measured Values

minimizing its power.

$$E = \int_{t_{reset}}^{t_{powerdown}} P(t) \cdot dt$$

Figure 4.5: Integral Formula

Every time we press the reset button, the processor exits the power-down state and starts executing the software loaded in its memory until it enters the power-down state again, at the end of the execution. By measuring this power consumption variation, we were able to calculate the exact power consumed by our device when only the embedded hard-core CPU (i.e. implemented just if it was a stand-alone CPU) was executing the corresponding software as described in the next paragraph.

In particular, we extracted the values from the voltage graph that was captured by our accurate oscilloscope, and inserted them into MatLAB [62]. We divided each measured voltage by 23, which is the amplification factor implied by our amplifier, and then we calculated the integral of the voltage for the time interval between reset and power down, as seen in figure 4.5.

Finally, we divided the total Energy consumed with the total number of 32bit data items that were encrypted. In this manner, we calculated the exact energy that each program consumed for the encryption of a 32bit data stream. It should be noted that in the numbers shown we also include the power consumed in the wake-up process of the CPU. However, we have run each encryption algorithm for 1000 times, without powering off the process, and divided the overall power consumption by 1000 and we realised that the overall wake-up power consumption is less than 8% of the power consumption of each

application.

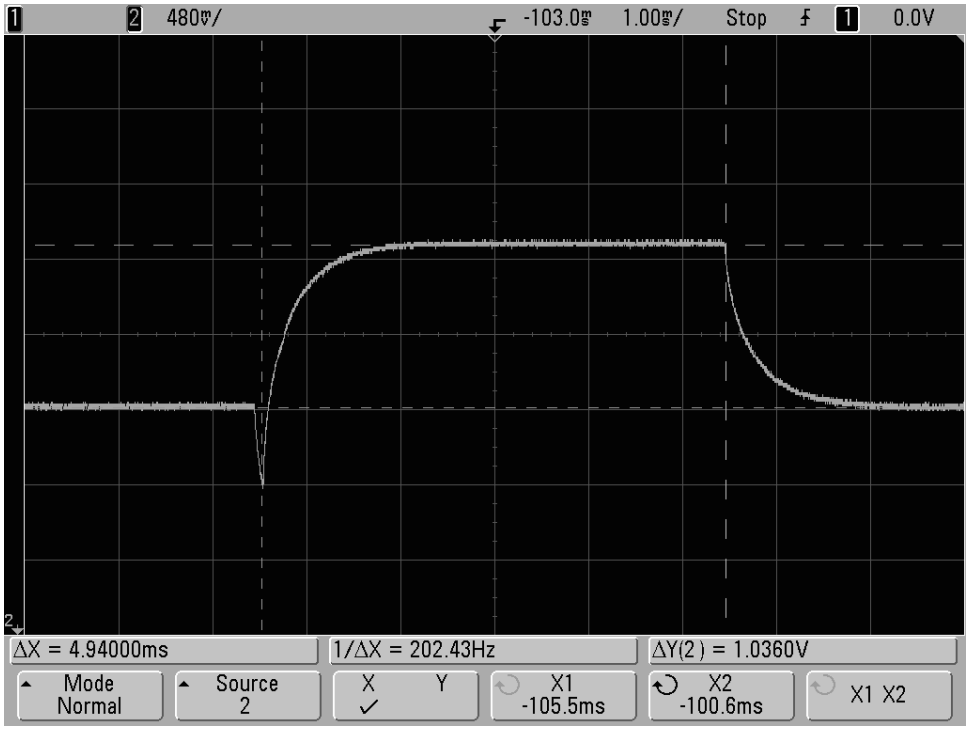

Figure 4.6: Oscilloscope Waveform for AES

This procedure was repeated three times, for all four algorithms, to ensure that the current displayed was the same for all repetitions. One representative waveform that was acquired by the Oscilloscope is shown in figure 4.6.

Data were extracted from the oscilloscope and after importing them to Matlab, we croped everything before the start and after the stop points, so as to get the exact time period that each security application was running. A representative processed waveform is shown in figure 4.7. The integral of this waveform, represents the actual power that our AES program consumed for encoding 12800 bytes of data.

4.4 Power Estimation

The power estimation numbers for each core and for each FPGA, were also reported by each vendors power estimation tool. We used Xilinx's Xpower tool [63] for Virtex and Spartan FPGAs, and Altera's PowerPlay [64] for the Cyclone-II device. These tools

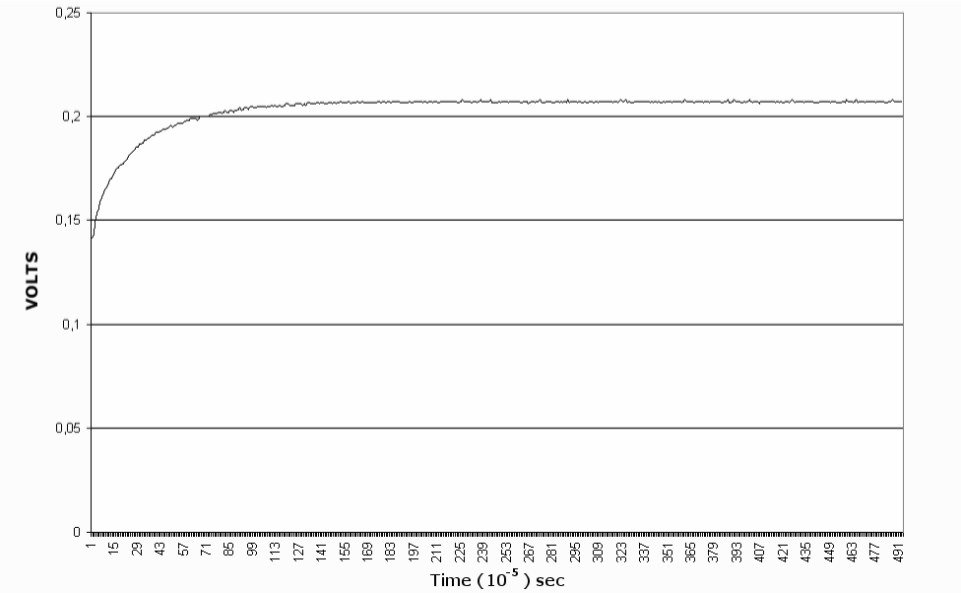

Figure 4.7: Edited Waveform for AES

calculate the consumed energy based on the fact that the dynamic power consumption inside an FPGA circuit is mainly caused by its switching activity. For every element e.g. LUT, RAM, I/O, wiring there exists a corresponding capacitance model. These tools can calculate the total energy by summing up the consumed energy on all references elements in the design; the energy consumption is calculated based on the specific capacitance models of the elements and on the modules' switching activity.

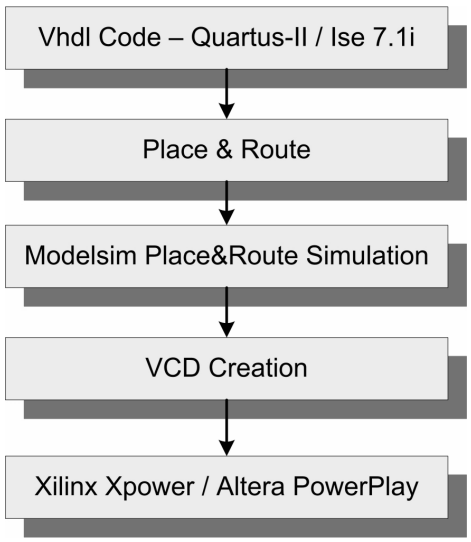

Figure 4.8: Power Estimation Flow

	DES	AES	$3DES$	$MD5$
$Virtex-IIPro$	$349,5$	1185	$370,5$	$481,5$
$Spartan-3$	118	340	127	127
$Cyclone-II$	$139,57$	$130,96$	$143,31$	$126,98$

Table 4.3: Estimated Values in mW

The information about every element's activity rate was taken from the VCD-file which was generated, during timing simulation, using Modelsim SE by Mentor Graphics [60]. The flow chart of estimating the power consumption of an FPGA-based system, is shown in figure 4.8.

When performing the timing simulation for the creation of the VCD file, a continuous 1ms simulation run was executed. There is no need for larger simulation times, since all signal rates remain the same throughout time. So this 1ms window is practically more than enough for XPower and PowerPlay to calculate the exact activity rate for each signal. An extended simulation of 1 second was also performed for each algorithm, which showed that the overall results differed no more than 1 to 2 mW. The estimation values computed by both power tools, when executing the exact same test bench as the one applied in our experiments accordingly, are demonstrated on table 4.3. All values are in mW.

All experiments were conducted in room temperature of about 21 to 23 degrees of Celsious. Power consumption estimated values were computed by using the default 25 degrees option on both estimator tools. Moreover, Xilinx Xpower tool didn't support the alternation of the input thermal data.

4.5 Power vs Utilization

First we measured how the consumption of the FPGAs internal core changed in relation to the silicon utilization. So, we implemented many instances of the AES core so as to cover 25%, 50%, 75% and 100% of the FPGA logic resources.

The selected Virtex-II Pro has 30K gates, Virtex-4 has 25K gates while Spartan-3 has

Table 4.4: Virtex-II Pro Measured Values

Utilization	Current	Power	Core Voltage
26%	0,706A	1,059W	1.5V
53%	1,27A	1,905W	1.5V
86%	1,75A	2,625W	1.5V
97%	2A	3W	1.5V

1000K gates. For example, for utilizing 99% of the Virtex-4 device, we needed 7 cores, while utilizing the large Spartan-3 FPGA, we needed 19 cores.

The actual measured data for all of the FPGA devices are demonstrated in the following tables: 4.4,4.5 , 4.6. Those present device utilization, FPGA internal core current consumption, internal core voltage and power consumption.

Table 4.5: Virtex-4 Measured Values

Utilization	Current	Power	Core Voltage
22%	0,237A	0,285W	1.2V
51%	0,396A	0,475W	1.2V
85%	0,604A	0,725W	1.2V
99%	0,510A	0,613W	1.2V

Table 4.6: Spartan-3 Measured Values

Utilization	Current	Power	Core Voltage
25%	0,049A	0,059W	1.5V
50%	0,060A	0,072W	1.5V
75%	0,076A	0,091W	1.5V
95%	0,082A	0,099W	1.5V

In figure 4.9, we can see the plot of the total power consumption of each FPGA's internal core, in relation to device utilization. As we can see in the plot, Virtex-4 full utilization power consumption (99%) is less than the power consumption when only 85% of the device is utilized. The explanation to this odd power measurement comes from the different placing and routing of the two designs. Apparently the placing and routing of the large design was much more efficient than the smaller one. This states that power the consumption is extremely dependant on the placement and routing of the design.

In figure 4.9 we can also see the derivative of the total power dissipation of an FPGA device with respect to the percentance utilization of the core. This is represented by the

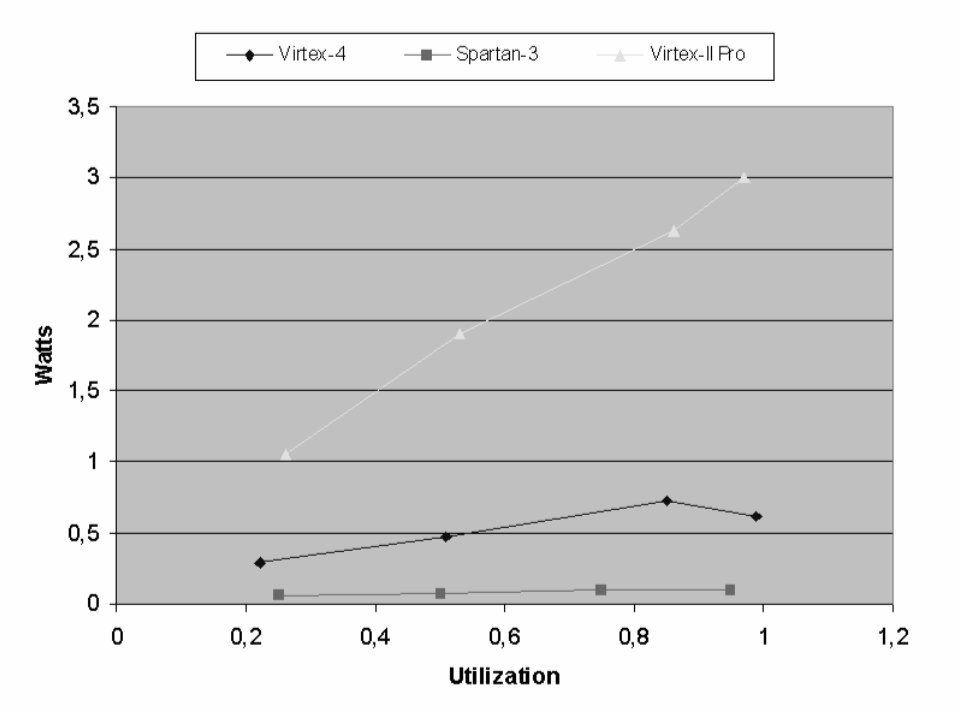

Figure 4.9: Watts per Utilization

slope of each graph. If we concentrate on the Virtex-II Pro and Virtex-4 devices (since both of these devices have embedded hard core peripheral logic) we realize that Virtex-4's graph slope is less that Virtex-II Pro's slope. This indicates that since the static current leakage is getting bigger and bigger as we pass to new processing technologies, and more hard-cores are built in the FPGA fabric, the reconfigurable devices tend to consume the same amount of energy regardless of their utilization. However, we are not there yet (and we will not probably be there for the next decade or so) since the power consumption is still heavily affected by the resources' utilization as demonstrated in the next paragraphs.

Another important remark is that Spartan-3 and Virtex-II Pro devices have a significant power consumption difference, although both devices use the same 90nm technology. Spartan-3 has about 3 times more gates than the Virtex-II Pro device and 4 times more than the Virtex-4 device.

In figure 4.10, we demonstrate in detail the total power consumption per 1000 gates in each device. This makes it clear that although Virtex-4 device has an advanced triple-oxide technology, Spartan-3 is by far a less power consuming FPGA. One possible expla-

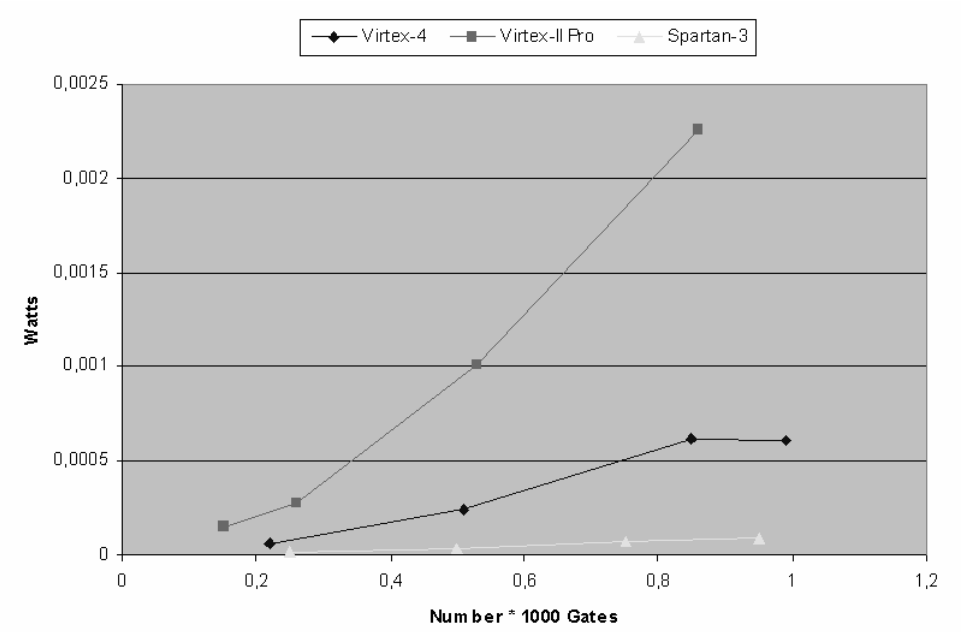

Figure 4.10: Consumption per 1K in Spartan-3 Devices

nation of this, is that the Virtex 4 device has embedded hard cores, such as numerous DSPs and much more, that consume power even when not programmed.

4.6 Clock Gating vs Reconfiguration

Another task of this thesis was to compare the actual power saved when clock gating as well as run-time reconfiguration are utilized. In graph 4.12 we can see the two numbers for a Virtex-II Pro FPGA. The higher graph, represents the power consumption of the clock gated design. First we apply the internal clock to only one core, then to two cores and so on. The lower graph represents the power consumption when real-time reconfiguration is utilized (i.e. we load only one module, then two and so on).

In all 7 measurements, a new design was compiled, placed and routed. Each of these designs was more power efficient than the related clock gated one. Furthermore, one additional advantage of the run-time reconfigurable approach against the clock gated one, is that we can have a large design mapped to a small FPGA device.

Finally, in figure 4.13, we demonstrate the overall percentance of power reduction

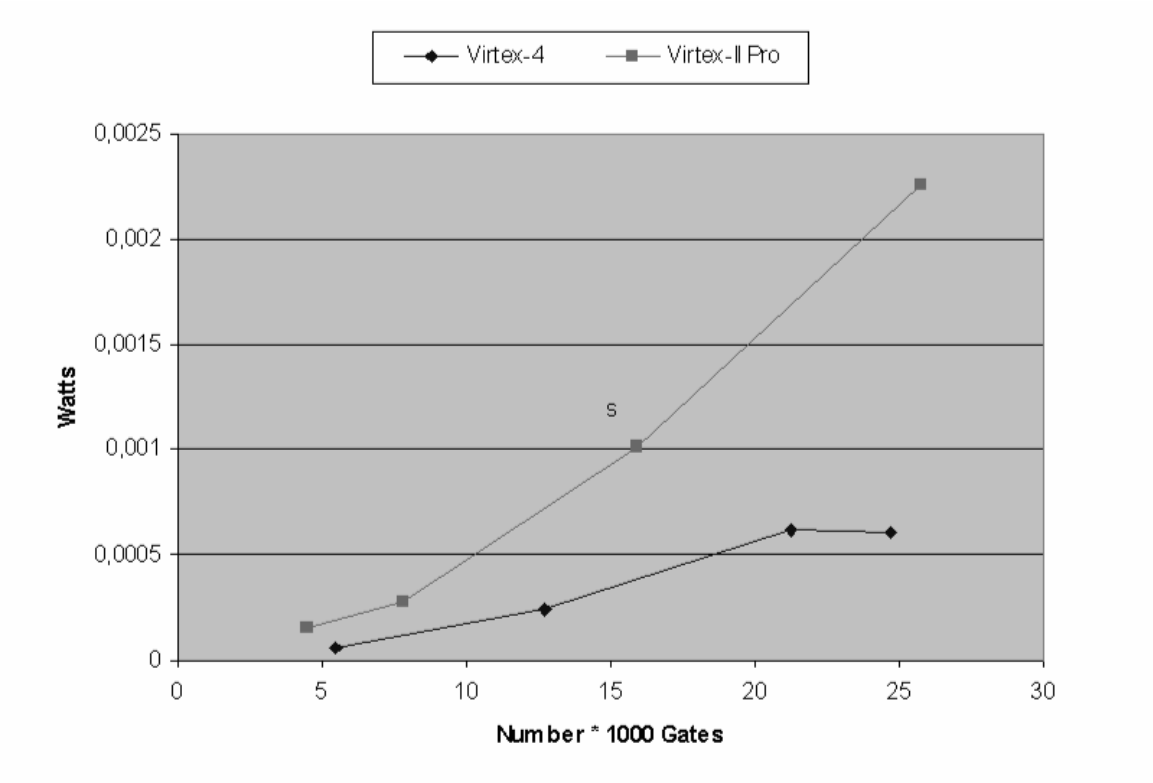

Figure 4.11: Consumption per 1K

when using run-time reconfiguration when compared with clock gating. Based on those measurements by using real-time reconfiguration we can from 7% to 27% of the power depending on the utilization of the device (i.e. the higher the utilization after reconfiguration the smaller the savings). Although this percent is not very large it can be very important for power critical applications.

Obviously, in order to be able to fully estimate the power reduction achieved when real-time reconfiguration is utilized, we also have to measure the time as well as the the power consumption of the reconfiguration process. Since reconfiguration is a time and energy consuming task, designs that have low reconfiguration frequencies would benefit from the presented approach (i.e. the energy saved over large periods of time would be more than that consumed by the reconfiguration process), whereas when frequent reconfiguration is needed the presented approach would not probably be beneficial.

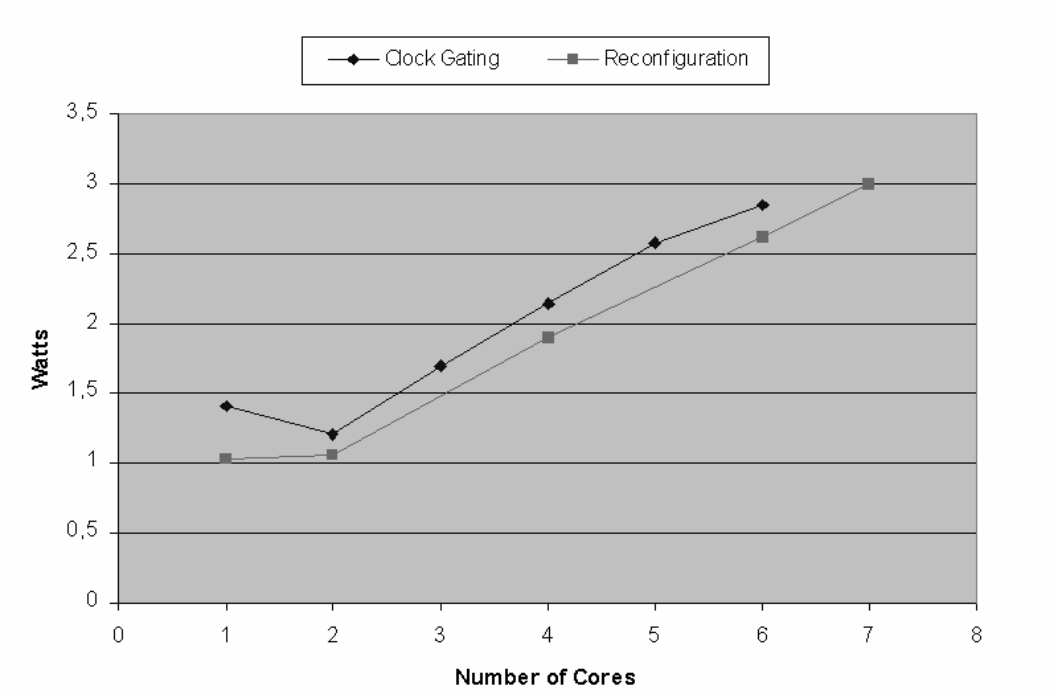

Figure 4.12: Watts per internal cores

	DES	AES	$3DES$	$MD5$
$ClockCycles$	36	48	38	32

Table 4.7: Time for a 32bit encode

4.7 Results and Evaluation

First we had to calculate the total execution time for each hardware algorithm. Although in software it is quite straight forward (we just measure the time from the waveform), in hardware it needed more complicated calculations. Since hardware run in an endless loop, we had to calculate the clock cycles needed for each algorithm to complete a 32bit encode.

By using ModelSim, we were able to count exactly the number of cycles needed by each algorithm. In these cycles, we added 24 cycles. These are the number of cycles needed for the Virtex-II PowerPC to write and read to the OPB bus (via the PLB-to-OPB bridge). This number is not generic, but it was used as a ”worst case” cycle time for each transaction. The total cycles needed by each security core is seen in table 4.7.

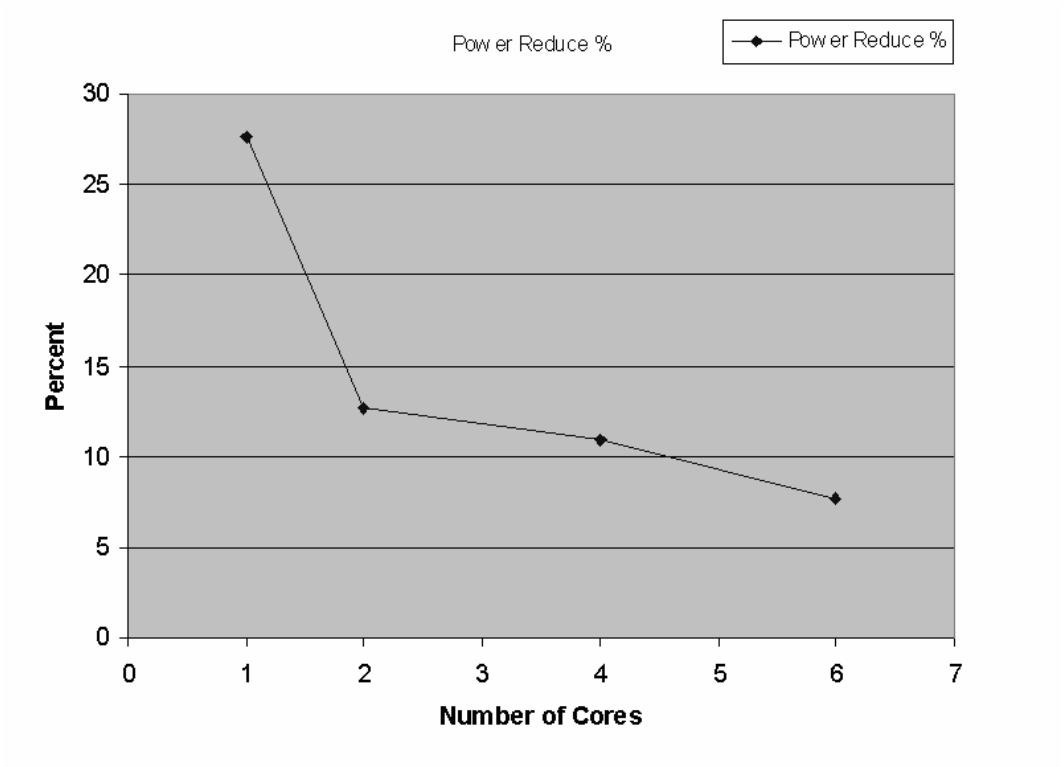

Figure 4.13: Percentance of power reduction

The Hardware modules and embedded CPU both run at 100Mhz, so we were able to calculate the total time needed for each algorithm to finish the encoding for a 32bit data chunk. In figure 4.14, we can see the total time needed for the SW to finish encoding. Also in figure 4.15, we can see the total Software Power consumption for each 32bit encoding.

In figure 4.16, we can see the total Hardware Energy consumption for a 32bit encoding. These values do not include the power consumption by the OPB bus and the OPB-to-PLB bridge.

As those results clearly demonstrate, although software, in general, is believed in certain cases in embedded systems, to be more power efficient, this is not true in real FPGA-based systems consisting of both reconfigurable resources and hard-core embedded CPUs. As we may observe in figure 4.17, software consumes almost 1000 times more energy than hardware. It should be noted that these values are the real measured data in the Virtex-II Pro device.

Hardware seems to win over software in time as well. In figure 4.18 the comparison

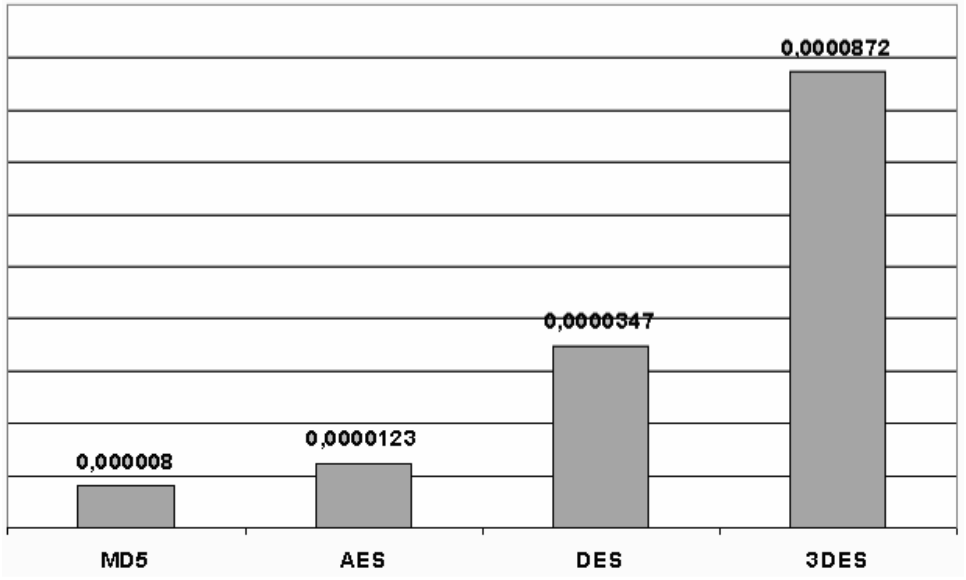

Figure 4.14: SW execution time

between software and hardware time needed for a 32bit data encode can be seen. Software needs over one thousand times more time than hardware for encoding the same amount of data. Real measured data are displayed in table 4.2.

Our next goal, was to compare the real power consumption data with the estimated ones. In figure 4.19, we demonstrate the overall measured and estimated results. All the left columns are the estimated values and all the right ones are the real measured data.

Although the FPGA families that are used are not identical, and thus not comparable with regards to their power consumption, we can clearly distinguish the Spartan-3 family as the most power efficient and the Virtex-II Pro family as the most power consuming one.

As it is clearly demonstrated there is a significant difference between the measured and the estimated values for the Spartan-3 device. In figure 4.20, we present the exact difference in milli-watts. If we divide this difference with the actual measured value, we will get the percentage of the error between the measured and the estimated value of each algorithm on each FPGA. Those numbers are presented in detail in figure 4.21. The variations for the Cyclone-II device are between 5,65% to 32,16%, for the Virtex-II Pro between 15,69% to 37,12% and for the Spartan-3 between 151,42% to 208,98%.

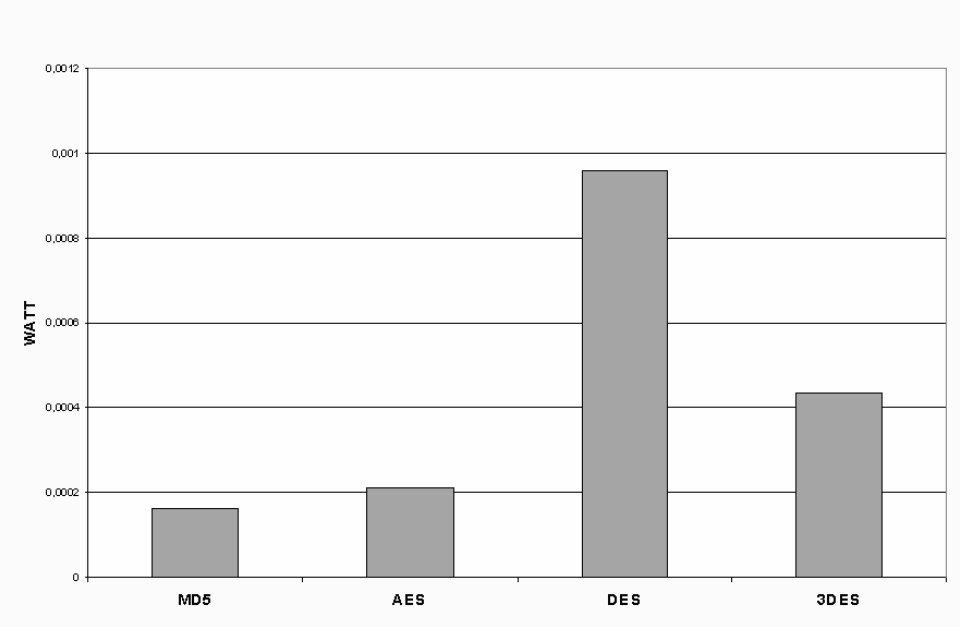

Figure 4.15: SW Power Consume per 32bit Data Enc

Although there are big differences between the measured and the estimated numbers, especially on the Spartan-3 device, we should mark that for all those measurements, that the XPower made a pessimistic estimation. This makes this tools quite reliable on estimating design's worst case power consumption which is an important factor when designing a real world application. On the other hand the variation in measured and estimated values cannot satisfy the needs of the designers of the various countermeasures for the power analysis attacks.

In the Altera case, although Powerplay is more accurate on its power estimations, the variations were between 5,65% to 32,16%, which still cannot address the needs of the designers.

As we may deduce from our measurements, in all those real-world security application, the hardware cores are almost 1000 times faster, while the hardware cores consume only the one thousandth of the energy that software implementation consumes. Therefore, this work clearly demonstrates that the optimal choice for the implementation of the security applications is the state-of-the-art reconfigurable hardware which consumes extremely less power and is 3 orders of magnitude faster than the software executed on a state-of-the-art

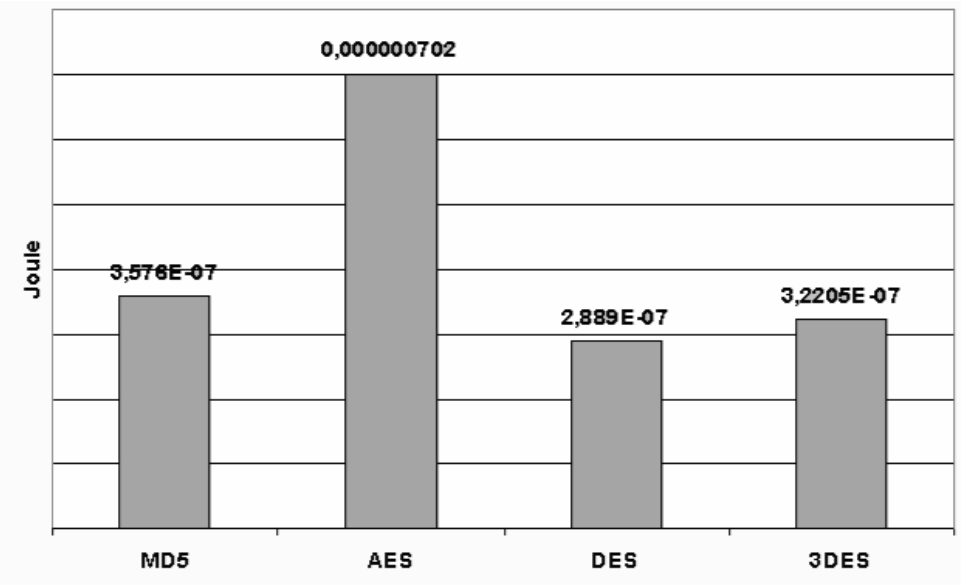

Figure 4.16: HW Energy Consume per 32bit Data Enc

low power RISC CPU, while it also offers much higher flexibility than an ASIC.

Based on those results we believe that the reconfigurable hardware can be used in both high-end devices supporting high levels of security such as multi-gigabit network cards and gateways, video processing devices as well as in low-end embedded systems such as battery operated sensor network nodes, wearable devices etc.

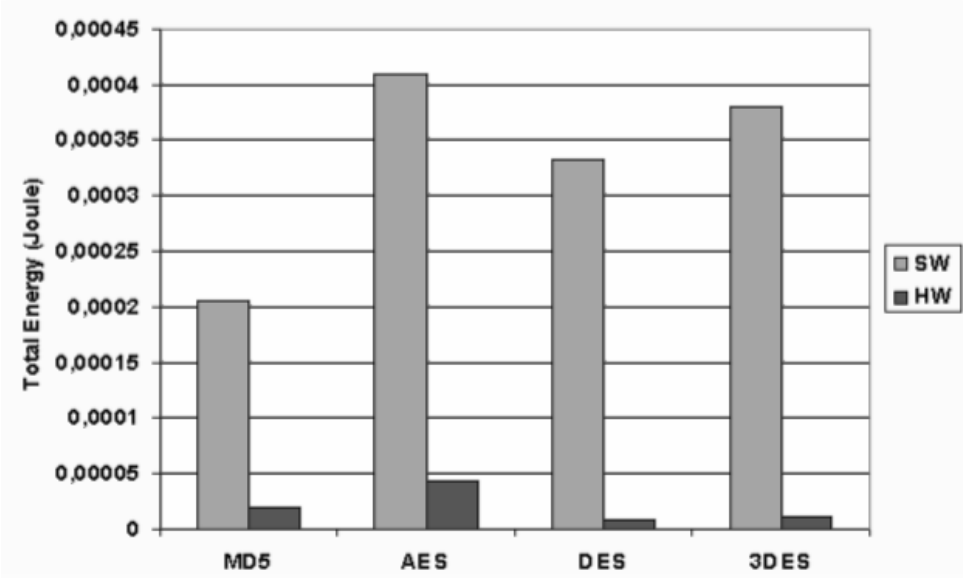

Figure 4.17: SW/HW Energy Consume per 32bit Data Enc

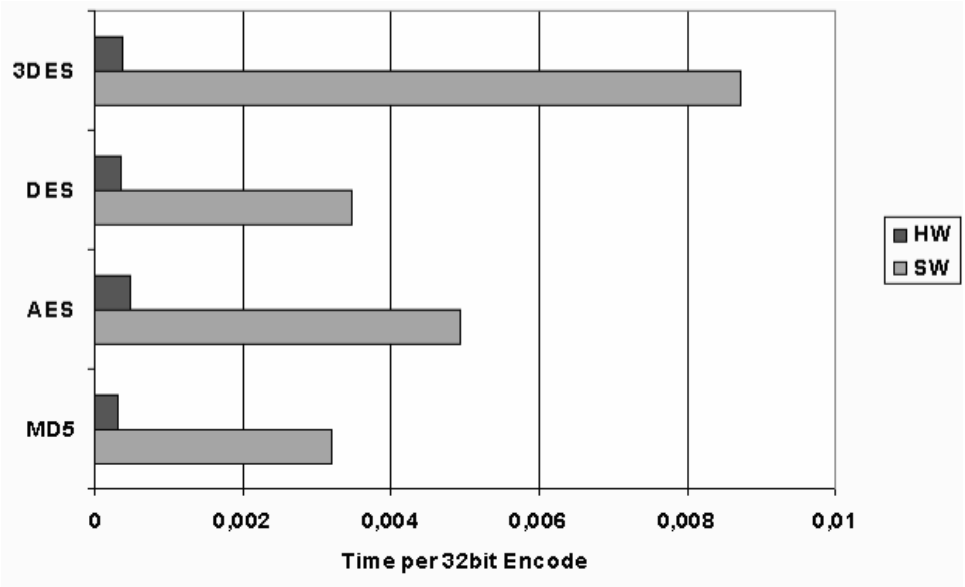

Figure 4.18: SW/HW Time consume per 32bit Data Enc

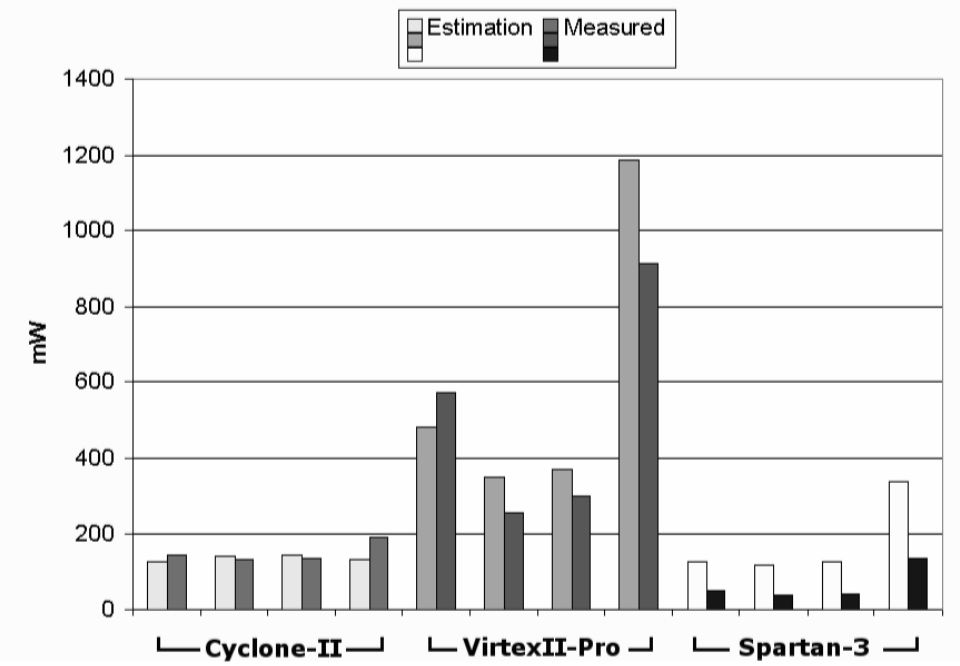

Figure 4.19: Estimation & Measurement Results

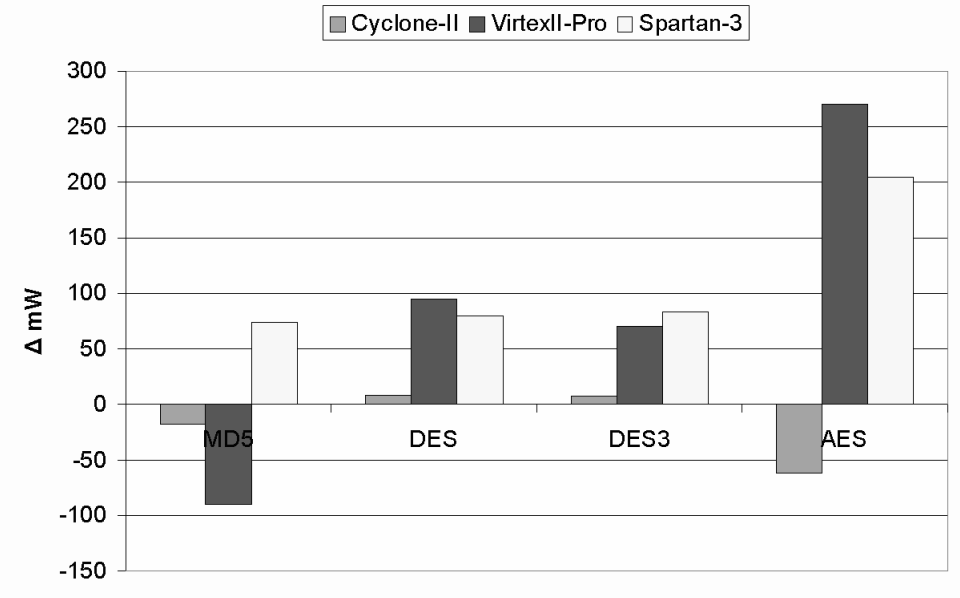

Figure 4.20: Difference in mW

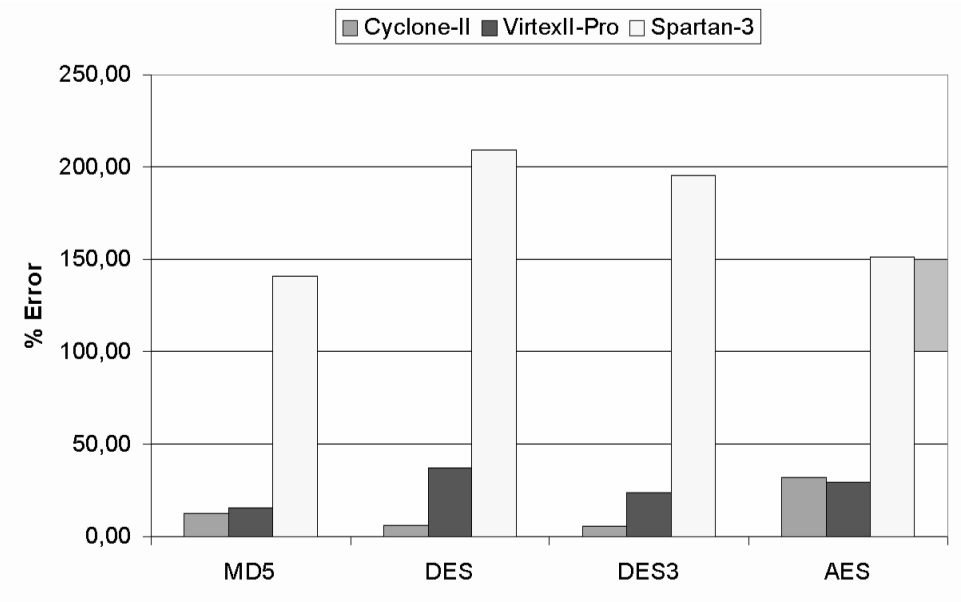

Figure 4.21: Error Percentage

CHAPTER

FIVE

CRYPTOGRAPHY AND LARGE NUMBER FACTORIZATION

5.1 Background

Factoring a positive integer n means finding positive integers u and v such that the product of u and v equals n, and such that both u and v are greater than 1. Such u and v are called factors (or divisors) of n, and $n = u \times v$ is called a factorization of n. Positive integers that can be factored are called composites. Positive integers greater than 1 that cannot be factored are called primes. For example, $n = 15$ can be factored as the product of the primes $u = 3$ and $v = 5$, and $n = 105$ can be factored as the product of the prime $u = 7$ and the composite $v = 15$. A factorization of a composite number is not necessarily unique: $n = 105$ can also be factored as the product of the prime $u = 5$ and the composite $v = 21$. But the prime factorization of a number - writing it as a product of prime numbers - is unique, up to the order of the factors: $n = 3 \times 5 \times 7$ is the prime factorization of $n = 105$, and $n = 5$ is the prime factorization of $n = 5$.

In this thesis we concentrate on finding just a factorization. The prime factorization can be obtained by further factoring the factors that happen to be composite: both factorizations $n = 7 \times 15$ and $n = 5 \times 21$ of $n = 105$ can be further refined to the prime

factorization $n = 3 \times 5 \times 7$ of $n = 105$, the first by further factoring 15, the second by factoring 21. There are efficient methods to distinguish primes from composites that do not require factoring the composites (cf. [29], [50] - Integer factoring p.31). These methods can be used to establish beyond doubt that a certain number is composite without, however, giving any information about its factors.

Factoring a composite integer is believed to be a hard problem. This is, of course, not the case for all composites -composites with small factors are easy to factor- but, in general, the problem seems to be difficult. As yet there is no firm mathematical ground on which this assumption can be based. The only evidence that factoring is hard consists of our failure so far to find a fast and practical factoring algorithm. Interestingly, and to an outsider maybe surprisingly, an entire industry is based on this belief that factoring is hard: the security, i.e., the unbreakability, of one of the most popular public key cryptosystems relies on the supposed difficulty of factoring.

This relation between factoring and cryptography is one of the main reasons why people are interested in evaluating the practical difficulty of the integer factorization problem. Currently the limits of our factoring capabilities lie around 130 decimal digits. Factoring hard integers in that range requires enormous amounts of computing power. A cheap and convenient way to get the computing power needed is to distribute the computation over the Internet. This approach was first used in 1988 to factor a 100-digit integer [32], since then to factor many integers in the 100 to 120 digit range, and in 1994 to factor the famous 129-digit RSA-challenge number (cf. [4]).1 In 1996 a 130-digit number was factored, partially using a World Wide Web interface [13].

In 2009 a six-institution research team led by T. Kleinjung has successfully factored the RSA-768 challenge number. While the RSA Factoring Challenge is no longer active, the factoring of RSA-768 represents a major milestone for the community. The effort took almost 2000 2.2GHz-Opteron-CPU years according to the submitters, just short of 3 years of calendar time.

We distinguish two main types of factoring methods: those that work quickly if one

is lucky, and those that are almost guaranteed towork no matter how unlucky one is. The latter are referred to as general-purpose algorithms and have an expected run time that depends solely on the size of the number n being factored. The former are called special-purpose algorithms; they have an expected run time that also depends on the properties of the - unknown - factors of n. When evaluating the security of factoring-based cryptosystems, people employ general-purpose factoring algorithms.

5.2 Algorithms for a Specific Factor Type

5.2.1 Pollard (rho - 1) Algorithm

Pollard's $\rho - 1$ algorithm is a number theoretic integer factorization algorithm, invented by John Pollard in 1974 [65]. This method is based on a combination of two ideas that are also useful for various other factoring methods. The first idea is the well known birthday paradox: a group of at least 23 (randomly selected) people contains two persons with the same birthday in more than 50% of the cases. More generally: if numbers are picked at random from a set containing p numbers, the probability of picking the same number twice exceeds 50% after $1.177\sqrt{p}$ numbers have been picked. The first duplicate can be expected after $c \cdot \sqrt{p}$ numbers have been selected, for some small constant c. The second idea is the following: if p is some unknown divisor of n and x and y are two integers that are suspected to be identical modulo p, i.e., $x = (y) mod(p)$, then this can be checked by computing $gcd(x - y, n)$ (Greatest Common Divider). More importantly, this computation may reveal a factorization of n, unless x and y are also identical modulo n.

This is a special-purpose algorithm, meaning that it is only suitable for integers with specific types of factors.

This algorithm relies on the hypothesis that N has a prime divisor p such that $p-1$ is a product of small primes. In this case, taking k to be a product of lots of small primes, we will eventually get $p-1|k$, whence by the Lagrange Theorem, we have $p|\alpha^k - 1$. Thus

taking $gcd(\alpha^k - 1, N)$, we detect p.

In more detail, Pollard's algorithm works in the abelian group $(Z/NZ)^*$. Using the Theorem of Lagrange, we know that for every $\alpha \in Z$ with $gcd(\alpha, p) = 1$.

$$\alpha^{(p-1)} = 0 \ mod \ p$$

And even for exponents $e = k \cdot (p-1)$ for some $k \in N$, we have

$$\alpha^e - 1 = 0 \ mod \ p$$

On the other hand, if $gcd(\alpha, N) = 1$, and if the order of a modulo N is not a divisor of e, then

$$\alpha^e - 1 \neq 0 \ mod \ N$$

which means that $d = gcd(\alpha^e - 1, N)$ is a proper divisor of N. Of course, we do not know either p or $p-1$. But if $p-1$ factors into a power product of prime numbers smaller or equal to some bound B_1 (i.e. $p-1$ is B_1-smooth), a multiple of $p-1$ can be computed as the product of all prime numbers bounded by B_1 "with small exponents".

Let's for example choose $N = 437$. We randomly choose the base $\alpha = 2$ and we use the exponent $e = 2^4 \cdot 3^4$. Then

$$\alpha^e = \alpha^{1296} = 305 \ mod \ N \text{ and } gcd(305 - 1, N) = 19$$

So 19 is a factor of N ($N = 19 \cdot 23$).

5.2.2 Architecture

5.2.3 Hardware

The simplified version of the Pollard $(\rho - 1)$ algorithm, is displayed in the following lines:

```
1: For (j=1 to k) do
2:   Compute b = ( b^e(j) ) mod ( N )
3: od
```

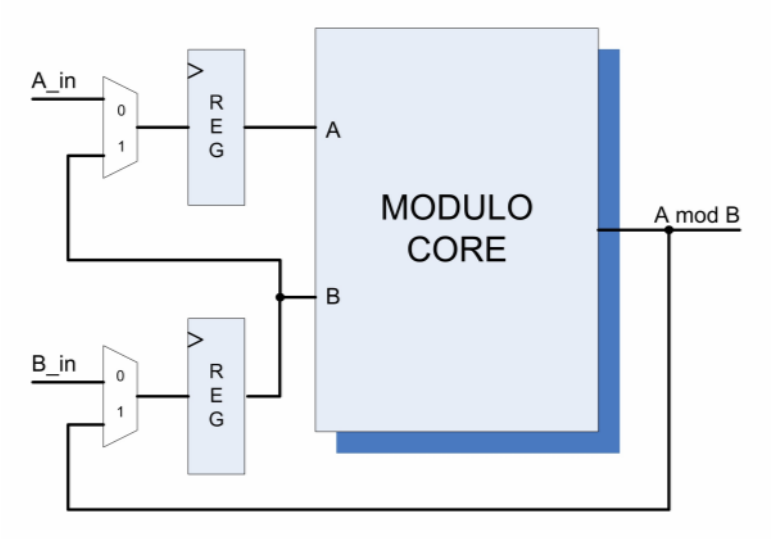

Figure 5.1: Great Common Divisor

```
4: if ( gcd(b - 1, N) > 1) then
5:    return(gcd(b - 1, N))
6: fi
7: return ("no factor found")
```

Where k is the number of multiplicities used, and e_j is the j-th multiplicity. Step 2 of the algorithm, uses the Left-to-Right Binary Exponantiation algorithm, described later on this section. Initial value of variable b is a random number less than N and greater than 1. All random and multiplicities numbers are stored inside the FPGA to Read only memories.

Hardware is designed using hierchical structure. The bottom core modules are the modulo and the multiplier unit. These two blocks are sequential. This means that the number of clock cycles needed for a computation is equal to, and some times less than, the number of bits of the integer number.

Second from the bottom in hierarchy is the Greatest Common Divisor module, which uses one modulo unit, two registers and two multiplexers, as seen in figure 5.1. This design uses the Euclides algorith of finding the greatest common divisor of two numbers. Let's assume that we need to find the greatest common divider of 27 and 15 - gcd(27,15).

So we have the following relations.

$$gcd(27, 15) \equiv gcd(15, 27 modulo 15) \equiv gcd(15, 12) \Rightarrow$$

$$gcd(27, 15) \equiv gcd(12, 15 modulo 12) \equiv gcd(12, 3) \Rightarrow$$

$$gcd(27, 15) \equiv gcd(3, 12 modulo 3) \equiv gcd(3, 0) \Rightarrow$$

$$gcd(27, 15) \equiv 3$$

The number of the iterations is not known and is different for each pair of numbers.

The algorithm used for Right to Left Binary Exponentiation, is displayed on the following lines:

```
1: Initialize b = 1, c = a modulo N.
2: while(d >= 1) do
3:   if (d is odd) then
4:      b = b * c modulo N, and d = d - 1.
5:   fi
6:   d = d / 2, and c = c * c modulo N.
7: od
8: return (b)
```

As we can see from the code, in each step, we have to calculate one multiplication and one modulo. These two functions are indepentend of each other. So, in hardware, we use two multipliers and two modulo units. First we calculate the multiplication, and then we calculate the modulo of each multiplication. So we need two multiplication units and two modulo units, all working in parallel. The main block diagram of the Left to Right algorithm can be seen on figure 5.2.

Higher in hierarchy we have the Pollard algorithm block. This block uses a Left2Binary block, a GCD block and a small memory block for reading the multiplicities. As input, we have the number N we want to factor and a random number less than N. A more detailed diagram can be seen in figure 5.3.

In the top level of the hierarchy we have a round-robin scheduler. The scheduler reads the random numbers stored in the read only memory, and distributes them to the

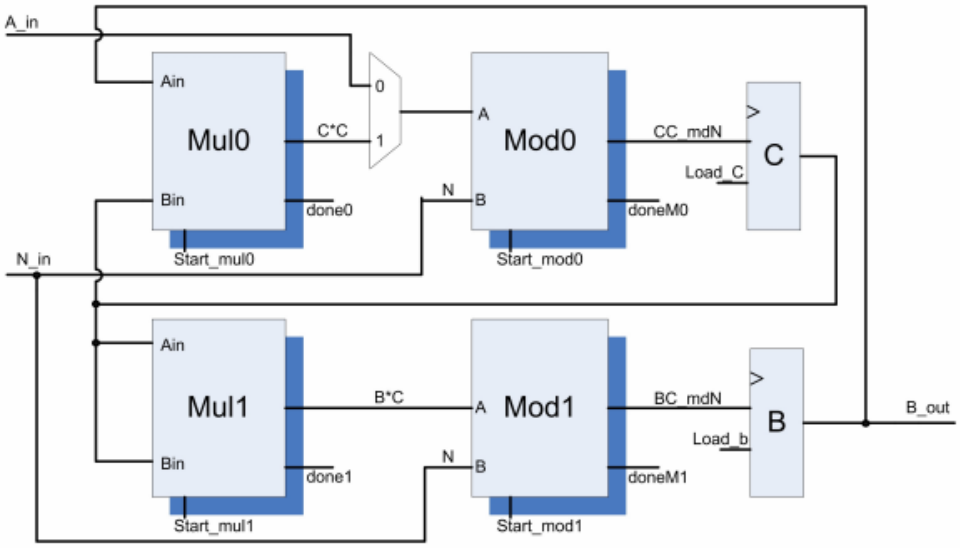

Figure 5.2: Left 2 Right Binary Exponantiation

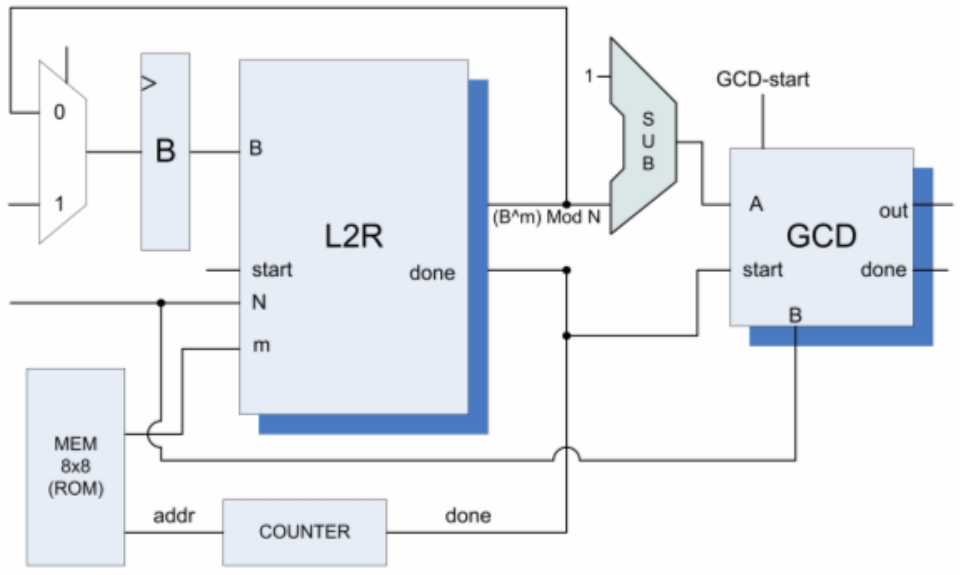

Figure 5.3: Pollard Algorithm

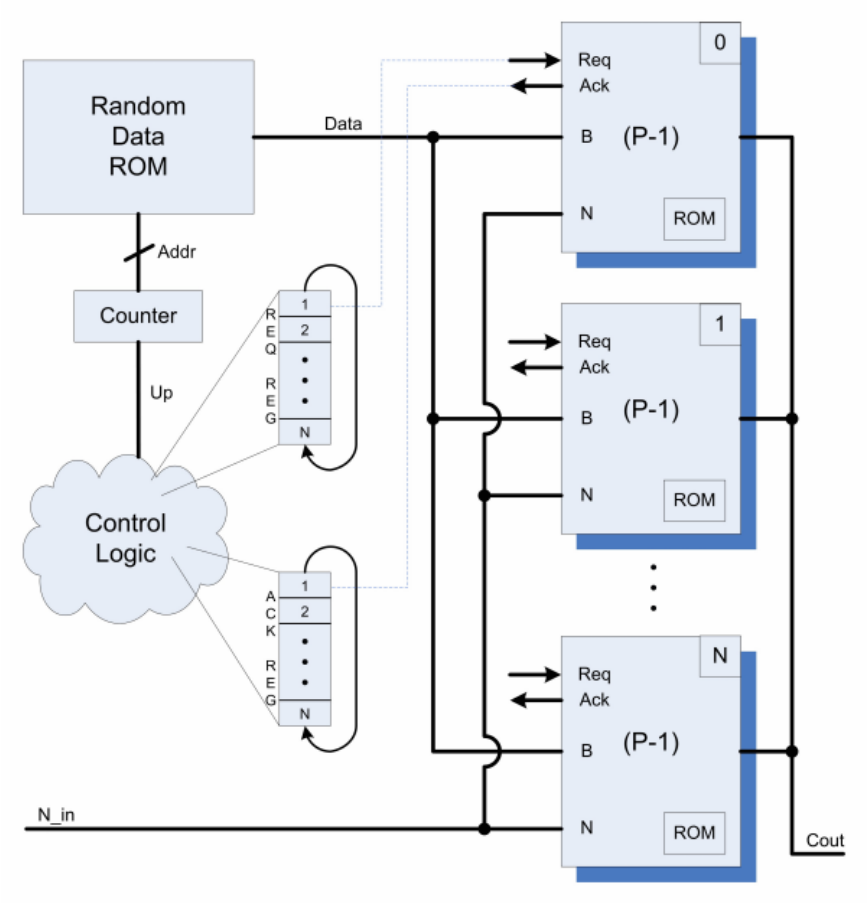

Figure 5.4: Scheduler Block Diagram

various pollard algorithm blocks. A simple request-acknowlegment protocol is used for the communication between the scheduler and the pollard blocks. Special parameters on the VHDL code, are used to define the length of the number N we want to factor and the number of the parallel Pollard algorithm cores used in the design. An abstract of the scheduler block is seen on figure 5.4.

5.2.4 Software

Software design was made using the Multiprecision Integer and Rational Arithmetic C Library (miracl) [66], which is optimized for large number arithmetic. Core fuctions are assembly optimized for each general purpose processor. The internal loops and the architecture of the program, is based exacly on the same algorithm that hardware uses.

Both multiplicities and random data are stored in static tables and are directly accessed by the implemented routines. No random number generation takes process on run time and all the stored values are identical to the hardware unit.

In a brief, first we get the random data from the table. Second, we calculate the binary exponentiation, and after subtracting one from the result, we calculate the greater common divisor. When this flow is finished, we get the next random number and we repeat the process. Both hardware and software cores find the factorization using the last (16-th) random number, stored in memory.

5.2.5 Experimental Framework

In this section we will try to demonstrate each procedure used to measure the exact execution time for software and for hardware.

5.2.6 Measuring H/W Time

For the Hardware part, we used the Xilinx ISE 9.1i HDL compiler [67] and the Mentor Graphics Modelsim SE simulator [60] programs. We run our tests for a VirtexII-Pro

Table 5.1: Number of parallel cores used

Bits	VirtexII-Pro	Virtex-4
34	7	35
91	4	13
112	3	10
250	2	8

Table 5.2: Real and Cropped Frequency Numbers (MHz)

Bits	VirtexII-Pro		Virtex-4	
34	105	100	193,798	190
91	94,879	90	118,517	115
112	86,809	85	102,613	100
250	81,397	80	91,62	90

device, with 30.000 logical gates and for a Virtex4 device, with 100.000 logical gates.

First, we had to determine the number of the parallel pollard algorithm blocks, that could fit to each device, depending on the width of the number to be factored. We used four different number widths. A 34bit, a 91bit, a 112bit and a 250bit number. So, we run the HDL compiler, to discover the total of the parallel blocks that can be used in each case. In table 5.1, we can see the exact number of cores used in each case. It is obvious, that when our architecture increases its width, we can fit fewer cores into the device.

After determining the number of the cores in each case, we used the HDL tool, to synthesis and generate Post-Place and Route static timing information. So in each case, we determined the maximum frequency at which our design is able to run. We cropped each frequency to the lower multiple of 5, and we continued our measurements. In table 5.2 we can see the exact frequencies for our designed, plus the cropped ones, used for our final numbers. All numbers are in MHz.

Next step was to determine the exact running cycles for each design. So, by using modelsim SE, we were able to determine the exact cycles needed by each algorithm to factor the input integer. A 64-bit counter was used to count the total cycles needed. The start signal would mark the begining of the factorization and the done signal would mark the end of the process.

The total cycles measured, divided by the core running frequency, give us the total running time of the factorization process.

5.2.7 Measuring Software Time

In the software part, the time measurement was quite straight forward. We used for our measurements a Pentium-IV, HT processor, running at 3.0GHz frequency. All programs were compiled using Microsoft Visual Studio 6. The time was measured using Intels Vtune Analyser 9.0 for Windows XP [68].

To achieve maximum accuracy, we run each factorization program 1000 times in a row, and then we divided the measured time by 1000. We used this procedure, because the accuracy of the Vtune analyzer is not so accurate in small programs.

Random and multipliciates of data are all stored static on tables inside the program. The flow of the program uses exactly the same flow as the hardware does, except for the parallelization. Each measurement was repeated for 5 times, and we used the mean time from all the repeatitions. All measured data is represented on table 5.3.

5.2.8 Results and Evaluation

In figure 5.5, we display the total running time in milliseconds for both hardware and software implementations. The light gray bar, representing the software execution time, is by far higher in all four situations. Also, the dark gray, representing the Virtex-4 device family, is always faster, in all four cases.

In table 5.3, we display the exact times measured, according to the bits of the factoring number. Software execution time is increasing linear, but hardware is not. Although we would expect to have a linear speedup, this is not true for our FPGA approach. This is due to the following reasons.

First, when the factoring number is increasing in width, our critical path is getting bigger and thus the hardware frequency is decreasing. Second, multipliers and dividers

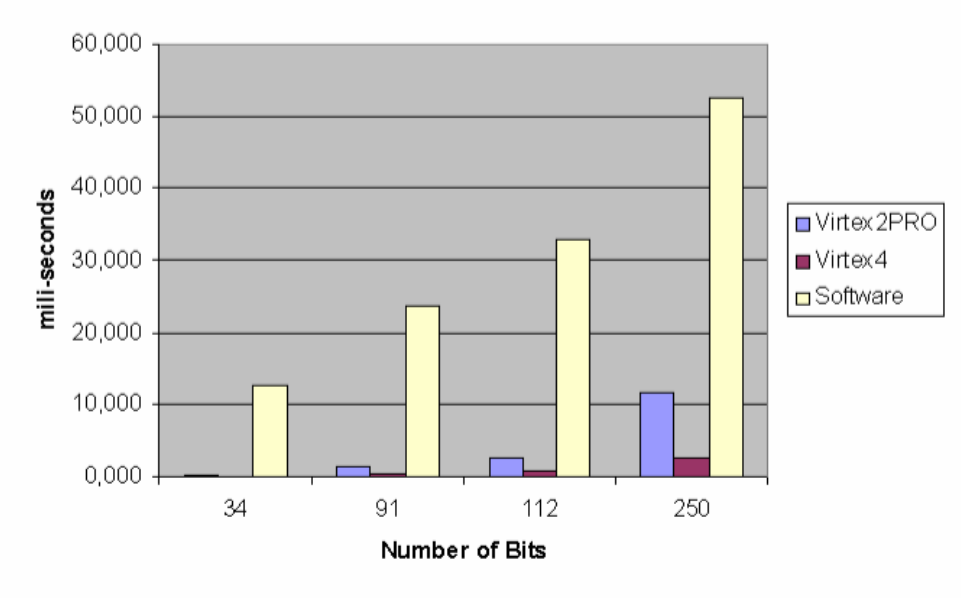

Figure 5.5: Running Time

Table 5.3: Time in Miliseconds for S/W and H/W

Bits	VirtexII-Pro	Virtex-4	Software
34	0,297	0,054	12,558
91	1,441	0,572	23,502
112	2,823	0,810	32,832
250	11,660	2,690	52,608

need, in general, the same number of cycles as the width of the factoring number. So a multiplication of a 34 bit number needs 34 cycles and a multiplication of a 250 bit needs 250 cycles. Finally, as the Pollard core block is increasing in gates, fewer parallel computing cores fit in the FPGA, as seen in table 5.1.

In general, hardware speedup is from 42 to 4,5 times faster than software on a VirtexII-Pro device implemantation, and from 231 to 19,56 times faster when the implementation is made on the Virtex-4 device family.

5.3 General Purpose Algorithms

5.3.1 Number Field Shieve

5.3.2 Background

The best known algorithm for factoring large numbers is currently the Number Field Sieve (NFS) [69], proposed by J.M Pollard in 1988. This algorithm tries to factor a 'hard' composite number by exploiting factorizations of smooth numbers in a well-chosen algebraic number field. It has four main steps: Polynomial Selection, Sieving, Linear Algebra and Square Root. The most time consuming are the Sieving and the Linear Algebra steps. The Linear algebra step, which is the one we deal with, tries to find a linear dependency among vectors over the GF(2) over a large and sparse system of linear equations. There are four main algorithms used for finding these dependencies. The Structured Gaussian Elimination, the Lanczos Algorithm [70], the Conjugate gradient and the Block Wiedemann Algorithm [71] [72]. Structured Gaussian ellimination was designed to reduce the problem to a much smaller one that could be solved by other methods. The Wiedemann algorithm, is currently the most attractive one, since it can work with high propability of success, while the others algorithms are only heuristic.

This Block Wiedemman algorithm needs to compute sequences of the form

$$A \cdot u_i, A^2 \cdot u_i, ..., A^k \cdot u_i$$

where A is the $(m \times m)$ sparse linear equation matrix, u_i is a -not necessarily sparse- binary vector, with $u \in GF(2)^m$ and $k \approx 2 * m/K$ (K is the blocking factor with typical values $K = 1$ or $K \geq 12$). Typical values of m for factoring a 512-bit integer is $6,7 \cdot 10^6$ and for i from 50 to 300.

The linear algebra step is a time and space consuming task. Although we store only the non-zero entries of matrix A, we still need about 2.3Gbytes of memory for a $(6,7 \cdot 10^6) \times (6,7 \cdot 10^6)$ matrix with a mean of 63 non zero entries per column. Also, for

computing $A^k \cdot u$ we need $k \approx 2 \cdot 6,7 \cdot 10^6/K$ matrix-by-vector multiplications. Whole procedure sould be repeated for i =50 to 300 times. This makes necessery the use of special-purpose hardware.

5.3.3 Main Algorithm

Let's assume that we have a sparce matrix $A_{6\times6}$ with mean of one non zero entry per column.

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

If we multiplicate this matrix A by a binary vector U we will get the following result.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{pmatrix} = \begin{pmatrix} u_5 \\ 0 \\ u_3 \\ u_1 + u_5 \\ u_4 \\ u_2 \end{pmatrix}$$

The equivelant circuit for this multiplication is shown in figure 5.6. As we can see, a 2-input xor gate is used only for the line that has two non zero entries. All other lines have either a ground signal (0) or an input signal directly connected to the output. With this circuit we have the advantage to make one multiplication by vector every clock cycle. Further more, the clock cycle width depends entirely on the delay of the 2-input xor gate. This help us build high frequency multiplication circuits.

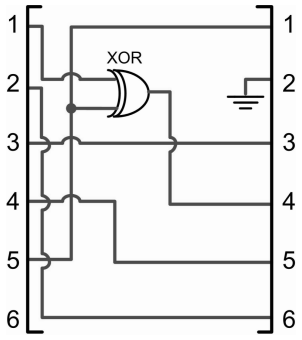

Figure 5.6: Circuit of matrix $A_{6\times 6}$

The main matrix A of the 1024-bit Linear Algebra step has about 100 non zero entries for each $4 \cdot 10^7$ wide column. If we take a random 256×256 sub matrix B of A, then we have maximum only one non zero binary entry. Matrix A does't necesery have uniform distributed non zero entries. But with an initial pre-processing of matrix A, we can have the desired distribution of the entries.

5.3.4 Sub-Matrix Computation

Since the size of matrix A is huge, it is a good idea to distribute the matrix-by-vector multiplications to a amount of parallel working machines. This is easy to be done, since we can use the attributes of the matrix by vector multiplication. We split matrix A, into $t \times s$ sub-matrices $A_{i,j}$. We also split vector U into t sub-vectors.

$$A \cdot U = \begin{pmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,t} \\ A_{2,1} & A_{2,2} & \dots & A_{2,t} \\ \vdots & \vdots & \ddots & \vdots \\ A_{s,1} & A_{s,2} & \dots & A_{s,t} \end{pmatrix} \cdot \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_l \end{pmatrix} \Rightarrow$$

The desired $A \cdot U$ result is equal to:

$$A \cdot U = \begin{pmatrix} \sum_{k=1}^{k=t} A_{1,k} \cdot U_k \\ \sum_{k=1}^{k=t} A_{2,k} \cdot U_k \\ \vdots \\ \sum_{k=1}^{k=t} A_{s,k} \cdot U_k \end{pmatrix}$$

As we can see, each sum of matrix-by-vector multiplication is independent to each other, making it possible to distribute this multiplication to s parallel machines. The only disanvantage is that each machine needs to know the entire vector U. But this surplus of data is only a small fraction of the complete data and it can be easily duplicated to each machine without much overhead.

5.3.5 Design

Input/Output Control

One of the limitations of the linear algebra step, is the need for high input/output bandwidth. Not only must the custom design calculate as fast as possible the matrix-by-vector multiplication, but it must also do it in high rate. Althouth this is a crucial factor for the overall speed of the circuit, I/O has not been taken into serious account.

Sashisu Bajracharya [52] for example, doesn't give detailed information on the architecture of the I/O. After some calculations on Sashisu FPGA design, we found out that the I/O meets the following formula:

$$Circuit\_Speed(Mhz) \times IO\_Pads\_Number = 151347$$

This reveals a major flaw of the design. If we replace the frequency with 80Mhz (where the circuit run), we will find out that we need about 1892 I/O Pads on the FPGA. But even if the frequency was as high as 200Mhz, we would still need 757 I/O Pads. In both cases, the pad number is unrealistic.

To overcome this problem, we use the high speed, serial trasceivers (MGTs) of the Xilinx Virtex devices. Each one of the trasceivers, can transmit and receive data at a rate of 6.5Gbps. For the serialization/deserialization of the data, as well as for the control of the I/O data, we use the Aurora Protocol by Xilinx [73]. Aurora protocol uses a 8b/10b serial encoder/decoder that downgrades the overall capacity of the transceiver to 5.2Gbps. This is translated to 32bit parallel input or output data every clock cycle, at a frequency of 162.5Mhz for each of the MGTs. The development system that we used in order to implement our design, was at the time a state-of-the-art Virtex-5 FPGA Development board. This development board, as seen in figure 5.7, has an XC5VLX330T FPGA device with 24x RocketIO digital tranceivers. This provides us a high I/O interface that we can utilize in order to sufficiently recieve and transmit all the matrix data.

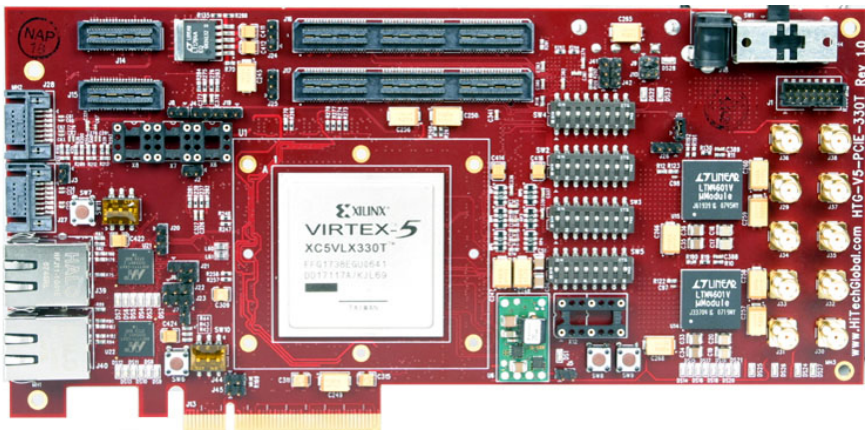

Figure 5.7: Virtex-5 Development Board

The same development board was used by Ioannis Kartsonakis in order to build a Linux Device driver and a PCI-E hardware block. The intention of this work was to provide an easy access port for the Real-Time reconfiguration of the FPGA device.

Architecture

Although our fist intention was to implement our design also on a Virtex-II Pro FPGA, this was not feasible due to the physical limitations of the specific device. Virtex-II Pro transceivers are located at the top and the bottom of the device. Meanwhile, run time

reconfiguration uses only full vertical block of slices on the specific device. This makes it impossible for us to maintain a static block around the MGTs of the Virtex-II Pro FPGA. On the other hand, Virtex4 and Virtex5 devices have the MGTs located on the right and left side of the chip, making them ideal for our design.

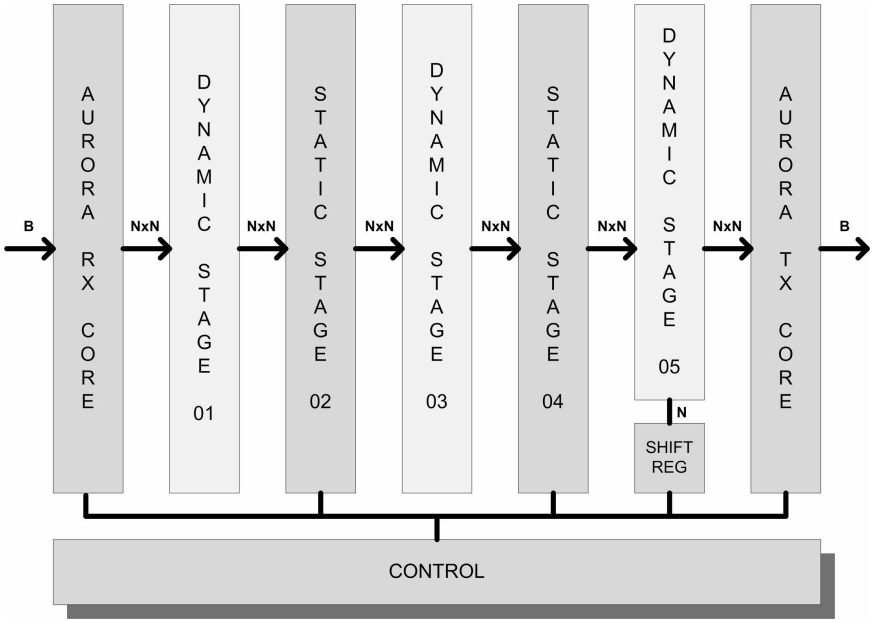

Figure 5.8: Design Architecture

The main concept of our design is to use a run time reconfigurable block for describing the sub-matrix A, half of MGT transceivers for receiving the sub-vector U and the other half transceivers for transmiting the matrix-by-vector $A \cdot U$ multiplication.

If we have $T = 2 \cdot d$ tranceivers on the device then d of them are used in parallel for receiving the data and d of them for transmiting the data. This means that we can manipulate $B = 32 \cdot d$ bits every clock cycle.

Let's now pick A_i, a random $((32 \cdot d) \cdot N) \times ((32 \cdot d) \cdot N)$ sub matrix of A. This matrix, can be splitted into N^2, $(32 \cdot d) \times (32 \cdot d)$ sub matrixes S_j. Each of these S_j matrixes,

will have only one non zero entry (matrix A is very sparse), when d is relatively small.

$$A_i = \begin{pmatrix} S_1 & S_2 & \dots & S_N \\ S_{N+1} & S_{N+2} & \dots & S_{2 \cdot N} \\ \vdots & \vdots & \ddots & \vdots \\ S_{N \cdot (N-1)+1} & S_{N \cdot (N-1)+2} & \dots & S_{N^2} \end{pmatrix}$$

If U_i is the multiplication vector of A_i, we will need N clock cycles for loading it into memory before multiplication. This would require N, $(32 \cdot d)$-bit registers. But since column k

$$Column(A_i, k) = \begin{pmatrix} S_k \\ S_{N+k} \\ \vdots \\ S_{N \cdot (N-1)+k} \end{pmatrix}$$

of matrix A_i has max N non zero elements, we only need N N-bit registers.

In figure 5.8, we can see the generic block diagram of our design. Left hand tranceivers are used for data input and right hand for data output. Our design uses three dynamic reconfigurable blocks (01,03 and 05) and six static blocks (tx, rx, 02, 04, control and shift register).

Our design is fully syncronized at 162.5Mhz. In this rate, we can load and transmit $32 \cdot d$ bit of data every clock cycle. Since we use a $((32 \cdot d) \cdot N) \times ((32 \cdot d) \cdot N)$ wide matrix, we need N clock cycles for loading each new $32 \cdot d \cdot N$ wide vector.

The needed registers for storing this vector are located on static block 02. On dynamic block 03, we have an interconnect that multiplies the input vector with the matrix A_i, with the same concept as seen in figure 5.6. On the N-th cycle, the vector will be multiplied and saved on the registers of static block 04. At the next N clock cycles the result will be transmited via the Aurora-TX block and at the same time a new vector will be loaded on static block 02.

Our design is capable of multiplying a $((32 \cdot d) \cdot N) \times ((32 \cdot d) \cdot N)$ wide matrix with

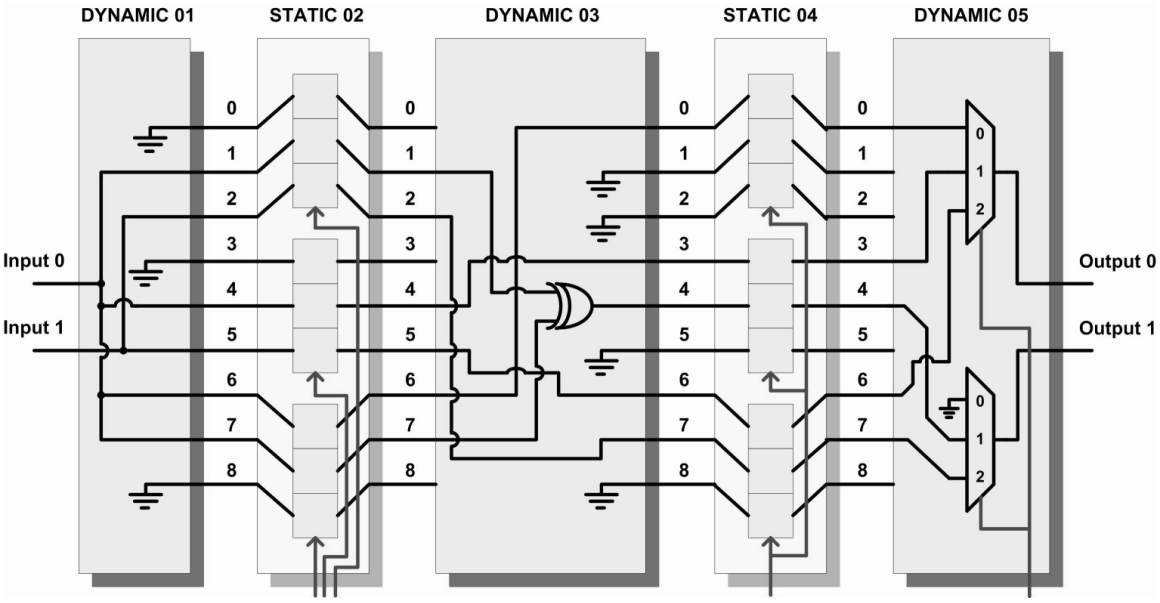

Figure 5.9: Detailed Design Example

a $32 \cdot d \cdot N$ wide vector, every N clock cycles, at a speed of 162.5Mhz.

In figure 5.9, we illustrate the datapath for the $A_{6\times6}$ matrix that was seen in section 3. We assume that instead of $32 \cdot d$ bits per clock cycle, we only have 2-bits per clock cycle. Also, we assume that $N = 3$, where each 2×2 submatrix of $A_{6\times6}$ has only one non-zero element.

At the first clock cycle, input data are stored at the top 3-bit register of static module 02. At the second clock cycle, the middle 3-bit register is used and at the third cycle, the bottom one is used. At the fourth clock cycle, data are loaded in parallel to all three, 3-bit registers of static block 04. At the same time, the first 2 bits of the result are transmitted on the output ports. Finaly, in cycles 5 and 6, all data have been transmited on the output ports and the new vector has been loaded in the stage 02 registers.

All signals that control the stage 02 and 04 registers as well as the stage 05 multiplexers, are generated by the control static block of our system.

Routing algorithm

Although the routing algorithm seems rather complicated, it is quite simple. On a matrix-by-vector multiplication, each element of the vector u_i will only be multiplied by each

element of the i-th column of matrix $A_{6\times6}$. So the first 2-bit input of stage 01 (vector bits u_1, u_2), will only be multiplied by columns 1 and 2 of matrix $A_{6\times6}$. The first 3-bit output of this stage, corresponds to the three submatrices $S_1, S_4, S7$ of matrix $A_{6\times6}$.

$$A_{6\times6} = \begin{pmatrix} S_1 & S_2 & S_3 \\ S_4 & S_5 & S_6 \\ S_7 & S_8 & S_9 \end{pmatrix}$$

If submatrix S_1 has a non zero element, this means that one of the vector elements u_1, u_2 sould be stored. In our example, submatrix S_1 has only zero elements, so, output 0 is grounded. Submatrix S_4 has a non zero element on column 1, which means that vector element u_1 sould be stored. The same comes with submatrix S_7 where vector element u_2 sould be stored.

After the first 3 clock cycles, all needed vector elements u_1, u_2, u_3, u_4, u_5 are stored on the registers of static stage 02. On the next clock cycle, the system is ready for the matrix-by-vector multiplication through dynamic stage 03.

On static stage 04, we have, once again, three 3-bit registers. In case all of the non-zero elements of the submatrixes S_1, S_2, S_3 are on the same row of matrix $A_{6\times6}$, we would need only one 1-bit register for storing the partial multiplication. But since in most of the cases, the non zero elements of matrixes $S_{j\cdot N+1}, S_{j\cdot N+2}, \ldots, S_{j\cdot N+N}$, are on different rows, we need in general one N-bit register for each row of the S arrays.

Dynamic stage 03, has as input a 9-bit vector, that corresponds to the vector S_{in},

where

$$S_{in} = Input[0:8] = \begin{pmatrix} s_1 \\ s_4 \\ s_7 \\ s_2 \\ s_5 \\ s_8 \\ s_3 \\ s_6 \\ s_9 \end{pmatrix} = \begin{pmatrix} 0 \\ u_1 \\ u_2 \\ 0 \\ u_3 \\ u_4 \\ u_5 \\ u_5 \\ 0 \end{pmatrix}$$

The routing algorithm checks each line of submatrices $S_{j \cdot N+1}, S_{j \cdot N+2}, \ldots, S_{j \cdot N+N}$, and routes the output to the j-th N-bit register of stage 04. In our specific example, submatrix S_1 and S_2 are both zero. Therefore, on the first bit of the first 3-bit register of stage 04, we store the $Input[6]$ of dynamic stage 03, which equals to the non zero element of submatrix S_3. Another example is the second bit, of the second 3-bit register of static stage 04, which is connected to the output of an xor gate. This gate is connected to $Input[1]$ and $Input[7]$ of dynamic stage 03 and outputs the xor function of the elements s_4 and s_6.

Finaly, dynamic stage 05 uses the necessary number of N-to-1 multiplexers, to route the multiplication result to the aurora tx module. In our specific result, on clock cycle number 4, $output[0]$ is the first bit of the first 3-bit multiplexer of static stage 04, and $output[1]$ is zero. On the fourth clock cycle, we will have as an output the 2-bit vector $u_5, 0$, which is the first 2 bits of the matrix-by-vector multiplication result.

5.3.6 Implementation and Methodology

Our design implementation is partitioned into two parts. The software and the hardware part.

Software

It is necessary for us to be able to create as fast as possible the three dynamic modules. To do this, we had to implement a C code program. As input to the program, we have the addresses of the non zero elements of the A_i sub matrix of A, as well as the size of the A_i matrix and the size of the sub matrix S. As output of the program, we have the VHDL sources of the three dynamic reconfigurable modules.

Also, we had to implement a C code program that would generate random A_i matrices, as well as worst case matrices, needed for testing purposes.

Hardware

Creating a partial Run-time reconfigurable hardware is a challenging procedure. Although it is quite straight forward, there are many critical factors that need to be taken into account, in order to have a full working system.

The first step towards implementing a partial run-time reconfigurable hardware is to separate the static and the dynamic parts. All static logic must be concentrated on one VHDL module. This includes static block 02, static block 04, system control and aurora tx and rx modules. All these modules are port mapped on one static module called "static.vhd". On the other hand, we have three dynamic modules, "dynamic01.vhd", "dynamic03.vhd" and "dynamic05.vhd". Each set of these three dynamic modules, represent a new A_i matrix.

All these modules, dynamic and static, need to be mapped to a top level module, called "top.vhd". All signals of the dynamic modules that need to be port mapped to the top level entity must not be directly connected to the other blocks.

To facilitate communication across reconfigurable module boundaries, yet still conform to the Partial Reconfiguration requirement that routing resources across such boundaries be completely fixed and static, the use of a special bus macro is required. Partial Reconfiguration requires the signals used as communication paths between or through reconfigurable modules must use fixed routing resources. That is, the routing resources

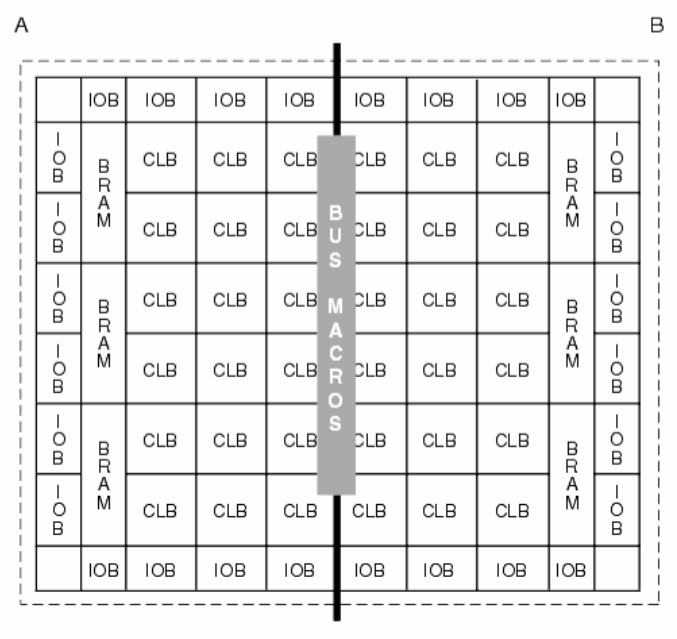

Figure 5.10: Bus Macro Example

used for such intermodule signals must not change when a module is reconfigured [74]. As shown in figure 5.10, the bus macro is a fixed routing bridge between the two sides, facilitating reliable communication. It is a pre-routed hard macro used to specify the exact routing channels and will not change from compilation to compilation.

The bus macro used in our design, have one 8-bit input and one 8-bit output port, with specific direction. As seen in figure 5.11, we have four different bus macros. The correct choise of the bus macro, depents on the direction of the data (input or output), as well as, on the specific side of the module (left or right) that will be placed.

After connecting the dynamic and static modules to the top level, we synthesized each module separately, using the Xilinx ISE 9.1i synthesizer [67]. Netlist files (ngc) were then extrated from each ISE project. These files, along with the ucf constrain file were used with Xilinx PlanAhead 10.1 [75] to create the floorplan constrains for our design. In all the ISE projects, we inserted manualy, where needed, all the necessery I/O, signal and clock buffers.

Dynamic reconfiguration areas as well as bus macros assignment, was done by hand, depending on the utilization needs of each seperate module. Planahead 10.1 is a quite

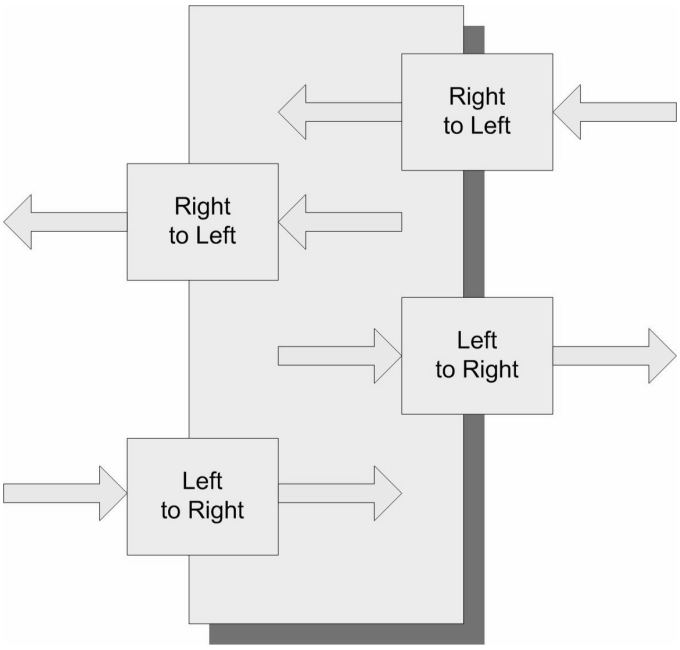

Figure 5.11: Bus Macro Direction

mature program, giving us a rough estimation of the slices that each module would need.

In figure 5.12, we have highlighted the three dynamic blocks (01,03 and 05) on a Virtex-4 floorplan. From left to right, we have the dynamic 01 block, 03 block and 05 block. Static block does not need to be specified, since it will consume all the unassigned floorplan area. We should also specify that the input MGT transceivers are on the left side of the FPGA and the output transceivers on the right side. This way, we can achieve a floorplan design as close as possible to the block diagram of our design that was seen in figure 5.8.

5.3.7 Results and Evaluation

Current bus macros utilize two horizontal slices on the device. Only one of the two slices should be inside the dynamic reconfiguration area. The location is also limited to the direction of the bus macro. For example, a left-to-right bus macro, on an output port of a dynamic module, can only be placed on the right border of the module.

The limiting factor of our design is the width of the interconnection between the dynamic and static modules. Since the input or output bus macro can only be placed

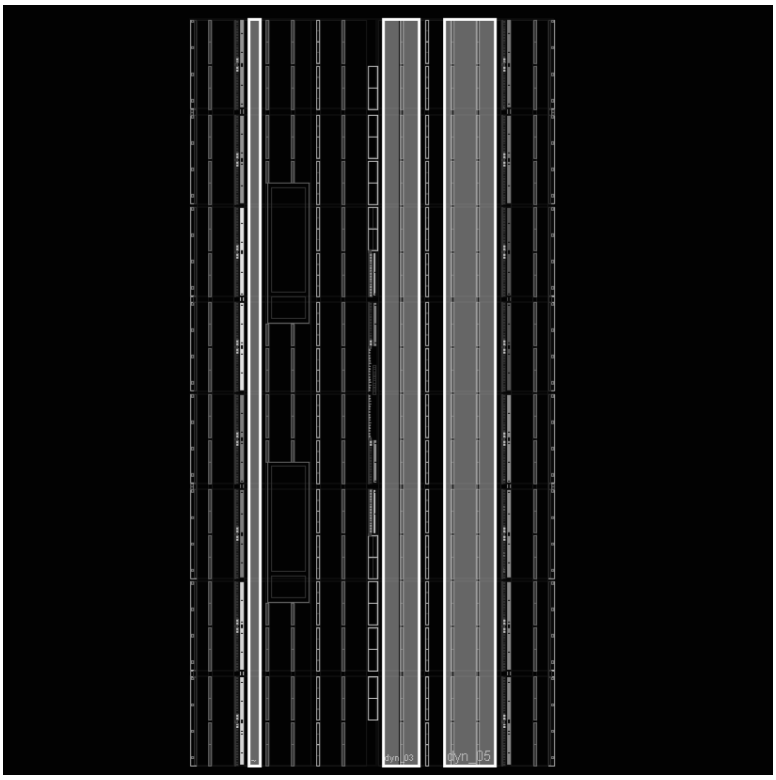

Figure 5.12: Planahead Floorplaning on a V4

on the one side of the module, we are limited to the number of vertical slices of each FPGA device. As we can see in table 5.4, on a Virtex-4 device we can achieve a 1024-bit interconnect ($N = \sqrt{1024} = 32$) and on a Virtex-5 a 1764-bit interconnect ($N = \sqrt{1764} = 42$). Also, on the Virtex-4 device, we use 16 MGTs, evenly distributed for receiving and transmiting data (d = 8) and on the Virtex-5 device we have 24 MGTs, resulting to a factor $d = 12$.

Table 5.4: Design Variables

Xilinx Device	MGTs	N	d	Frequency
Virtex-4	16	32	8	162.5 Mhz
Virtex-5	24	42	12	162.5 Mhz

Both of the designs are fully syncronized at Aurora's 162.5Mhz clock frequency. This simplifies our design, since we do not need to have two different clock domains in our system. This clock frequency can be easily achieved, since our design's critical path is limited to a max of an N-bit xor gate, depending on the matrix used.

In more detail, if matrix A_i has all non zero elements on the same row, this will result to a N-bit xor on dynamic module 03. Although this has a realy small possibility to occure, we sould take it in serious account. In Virtex-4, maximum combinational path delay for a 32-bit wide xor gate is 2.009ns and in Virtex-5, maximum combinational path delay for a 42-bit wide xor gate is 2.622ns. In both cases, our design is capable of achieving a 162.5Mhz clock frequency with not much effort.

Another posible limitation of our design, could be the number of N-to-1 multiplexers needed on dynamic stage 05. The worst case scenario for our design, although with extremely small possibility, would be if all submatrices S_i had a non zero element with non of them on the same row of the matrix A_i. This would result to a D number of N-to-1 multiplexers in dynamic module 05. But since our design doesn't utilize all of the FPGA fabric, we have a big number of CLBs to use for the dynamic module 05. Thus, it is relatively easy to overcome this limitation by simply scaling the FPGA area of the dynamic module 05. This can also be observed in figure 5.12, since the right dynamic module is larger than the other two.

Table 5.5: Speedup Compared to Mesh Design

Device	D	T_{clock}	Days	Speedup
Virtex-4	$4 \cdot 10^7$	6,15 ns	6520,62	19,4
Virtex-5	$4 \cdot 10^7$	6,15 ns	2208,09	57,4

The matrix A from the sieving step, for a 1024-bit factored number, has the size of $D \times D$, where $D = 4 \cdot 10^7$. Our system is capable of one $(32 \cdot d \cdot N) \times (32 \cdot d \cdot N)$ matrix to vector multiplication every N cycles. Thus, the total number of sub-matrix computations required to perform a sigle matrix-by-vector multiplication equals to

$$n = \frac{D^2 \cdot N \cdot T_{clock}}{(32 \cdot d \cdot N)^2} = \frac{D^2 \cdot T_{clock}}{(32 \cdot d)^2 \cdot N}$$

The matrix step needs about $3 \cdot D/K$ multiplications (in our case $K = 1$) for the block

Wiedemann algorithm [51]. Thus the total time for the matrix step equals to

$$n = \frac{3 \cdot D^3 \cdot T_{clock}}{(32 \cdot d)^2 \cdot N}$$

In table 5.5 we can see the results of our computations. These include the total days needed for factoring the 1024-bit integer with one FPGA device, as well ass the total speedup achieved in comparison to the latest FPGA mesh design [52].

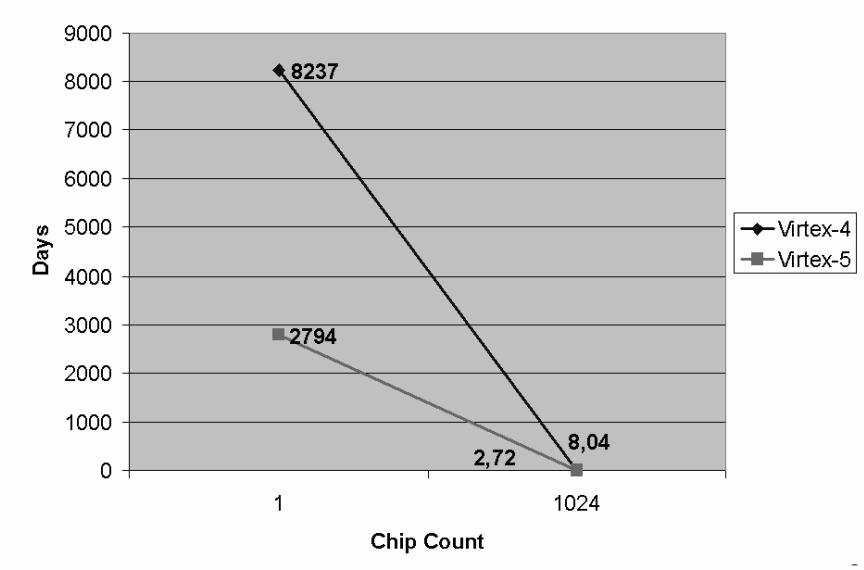

Figure 5.13: Days per Number of Chips used

Furthermore, in figure 5.13 we can see the total number of days needed for the Wiedemann algorithm, depending on the number of FPGAs used. Our system is linearly scalable with no additional add-up costs.

In our design, we do not take into account the time needed for the generation of the dynamic module bitstreams. This task can easily be merged to the sieving step of the algorithm, utilizing the long CPU idles between the sieve of each linear relation. Lastly, we do not take into account the time needed for the run time reconfiguration of the modules. In our future work, a pipeline technique will be used for transparent loading of each new submatrix A_i to the system.

CHAPTER

SIX

EVALUATION AND CONCLUSIONS

6.1 Evaluation

This Chapter summarizes the dissertation work and gives an overview of our results which were presented in detail in the previous chapters. This includes our work on power consumption, power estimation, cryptography as well as sparse matrix multiplication. All these areas that we investigated on this thesis are strongly related with each other giving an complete overview of the problems that arise from power consumption on the FPGA devices. Especialy for cryptographic applications.

- Our measurements on a state-of-the-art reconfigurable device show that, for the majority of the most widely used security algorithms in mobile applications, the FPGA-based hardware implementations are almost 1000 times faster than the corresponding software application. More importantly, with regards to the power consumption, the real-world measured values demonstrated that the reconfigurable hardware approach consumes three orders of magnitude less energy than the corresponding software implementations.

- We have also performed a comparison between the exact power consumption and the estimated power consumption extracted by the most widely used power estimating

tools in the FPGA world. The comparisons include both leading FPGA vendors and most of the widely used security applications. The differences, between measured and estimated power consumption, varied from 15.52% to 208% for the Xilinx devices and from 5.64% to 32.15% for the Altera device. These differences show that current power estimating tools are inadequate to produce the accurate results needed by the designers of the security hardware-based modules.

- We have evaluated the relation between the power dissipation and the processing technology for various FPGA devices. Our real-world experiments demonstrate that since static current leakage is growing, reconfigurable device tend to consume the same amount of energy regardless of the overall utilization of the device. However, we are not there yet (and we will not be there in the next few years) since the power consumption is still heavily affected by the resources utilization. We have also demonstrated that the power consumption of an FPGA device is heavily related to the overall placement and routing of the design. Designers need to be very careful with the routing and placement options since this can heavily affect the overall power consumed.

- More importantly, we have also presented the benefits of utilizing the real-time reconfiguration feature of current FPGAs for reducing the overall power consumption. As our results clearly demonstrate, the use of real-time reconfiguration, when not very frequent reconfigurations are required, can reduce the overall energy consumption of a design by up to 23% when compared with using clock-gating.

- In this dissertation, we have also demonstrated an FPGA-based design executing Pollard's $(\rho - 1)$ factorization algorithm. The proposed device is 20 to 231 times faster than the corresponding software application running on a 3GHz state-of-the-art CPU.

- Finaly we have introduced and implemented a new matrix-by-vector multiplication scheme using a module based run time reconfiguration design. Our design utilizes

the embedded, state-of-the-art, serial 6.5 Gbps transceivers of the Xilinx FPGAs, thus providing a feasible I/O mechanism. We have also explained and analyzed why our system can achieve a $19,4\times$ to $57,4\times$ speedup, compared to other proposed architectures, depending on the parameters of the block Wiedemann algorithm and on the FPGA device used.

It should also be stated that for the completion of this thesis a huge amount of energy and effort was consumed on obtaining the needed background theory and knowledge. This included the understanding and the deep investigation of Large Number Factoring algorithms and their relative mathematical background on Number Theory and Linear Algebra.

Another important part of this thesis was consummed in order to implement and construct linux device drivers, high level Object-Oriented Software and PCI-e hardware blocks that should later be used on our sparse matrix multiplication system. It should also be stated that for this task a Diploma Thesis was done by Ioannis Kartsonakis, whose work is greatly appreciated.

6.2 Future Work

Power consumption, FPGA devices and cryptographic algorithms are three major areas of interest that are closely related to each other. As the technology and our knowledge advances, new, faster and more practical solutions will be presented by the academic and industrial community.

A huge improvement potential on this thesis exists on the usage of Real-Time reconfiguration on cryptographic algorithms and especialy on sparse matrix multiplication. As the FPGA devices become larger and faster with even wider input/output interfaces, many problems that we faced on this dissertation can be overcommed.

Moreover, as power estimation software is becomming more and more sophisticated, a deeper analysis and evaluation can be performed. There is still a big gap between the

actual needs of the designer and the solution that FPGA ventors provide by their power estimation software. This provides us a huge area that can be still deeply explored.

The Linear Algerbra step of the Number Field Shieve is an area of interest that has huge potentials on deeper investigation and research. A task that emerges from this thesis is the implementation of a complete setup of a Number Field Shieve system. This would include our hardware implementation of the Linear Algebra step and Software implementation of the rest of the algorithm.

6.3 Conclusions

Guaranteeing security on mobile devices is challenging because of the specific problems this environment poses; the designer has to decide whether to implement the embedded security algorithms on software or to add special purpose hardware units to the system, executing those CPU intensive tasks. Moreover, as the security algorithms are becoming more and more vulnerable to the various differential power analysis attacks, it is crucial for the designers to be able to evaluate the power consumption, during the early development process, with high accuracy. But this task is quite difficult as the power estimation tools are not yet mature enough for the demanding mobile era.

We strongly believe that the results presented would lead numerous designers of mobile devices to consider to adopt FPGA-based hardware solutions for their implementations of security schemes, instead of the easily implemented but slow and power-hungry software implementations. We also believe that this work will guide many scientists to exploit the FPGA's real time reconfiguration capability within cryptography and Large Number Factorizaion. This is something that was first introduced on this thesis and has many new potential uses.

6.3.1 Acknowledgements

This work is part of the 03ED851 research project, implemented within the framework of the ”Reinforcement Programme of Human Research Manpower” (P.E.N.E.D) and co-financed by National and Community Funds (25% from the Greek Ministry of Development-General Secretariat of Research and Technology and 75% from E.U.-European Social Fund)

BIBLIOGRAPHY

[1] D. Sylvester, H. Kaul, "Future performance challenges in nanometer design," *Design Automation Conference*, pp. 3–8, June 2001.

[2] Paul Kocher. Joshua Jae and Benjamin Jun, "Differential power analysis," in *Advances in Cryptology: Proceedings of CRYPTO 99*, August 1999.

[3] T.S. Messerges, "Power analysis attacks and countermeasures for cryptographic algorithms," *Doctoral Thesis*, January 2000.

[4] Christof Paar, Jan Pelzl, "The advanced encryption standard," *Springer*, 2009.

[5] Biham, Eli and Shamir, Adi, "Differential cryptanalysis of des-like cryptosystems," *Journal of Cryptology 4*, 1991.

[6] Coppersmith, Don, "The data encryption standard (des) and its strength against attacks," *IBM Journal of Research and Development*, 1994.

[7] Ralph Merkle, Martin Hellman, "On the security of multiple encryption," *Communications of the ACM*, 1981.

[8] Rivest, R.; A. Shamir; L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, 1978.

[9] Don Coppersmith, "Small solutions to polynomial equations, and low exponent rsa vulnerabilities," *Journal of Cryptology*, 1997.

[10] Wiener, Michael J, "Cryptanalysis of short rsa secret exponents," *Information Theory, IEEE Transactions*, 1990.

[11] Antoine Joux, "Multicollisions in iterated hash functions," *Application to Cascaded Constructions*, 2004.

[12] Jonathan J. Hoch and Adi Shamir, "On the strength of the concatenated hash combiner when all the hash functions are weak," 2008.

[13] N. Rogier, Pascal Chauvaud, "The compression function of md2 is not collision free," *Selected Areas in Cryptography*, 1995.

[14] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, Xiuyuan Yu, "Cryptanalysis of the hash functions md4 and ripemd," *Eurocrypt*, 2005.

[15] Yu Sasaki, Lei Wang, Kazuo Ohta, Noboru Kunihiro, "New message difference for md4," *Fast Software Encryptio*, 2007.

[16] Berson, Thomas A, "Differential cryptanalysis mod 232 with applications to md5," *1992*, Eurocrypt.

[17] Bert den Boer; Antoon Bosselaers, "Collisions for the compression function of md5," *Springer*, 1993.

[18] E. Biham and A. Shamir, "Differential cryptanalysis of the data encryption standard," *Springer-Verlag*, 1993.

[19] M. Matsui, "The first experimental cryptanalysis of the data encryption standard," *Advances in Cryptology: Proceedings of CRYPTO '94, Springer-Verlag*.

[20] D. Boneh, R. DeMillo, and R. Lipton, "On the importance of checking cryptographic protocols for faults," *Advances in Cryptology: Proceedings of EUROCRYPT '97, Springer-Verlag*, pp. pp.37–51, 1997.

[21] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," *Advances in Cryptology: Proceedings of CRYPTO '97, Springer-Verlag*, pp. pp.513–525, 1997.

[22] P. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," *Advances in Cryptology: Proceedings of CRYPTO '96, Springer-Verlag*, pp. pp.104–113, 1996.

[23] J. Dhem, F. Koeune, P. Leroux, J. Quisquater, and J. Willems, "A practical implementation of the timing attack," *UCL Crypto Group Technical Report, Series: CG-1998/1*, 1998.

[24] R. Anderson, M. Kuhn, "Low cost attacks on tamper resistant devices," *Security Protocol Workshop*, 1997. [Online]. Available: http://www.cl.cam.ac.uk/ftp/users/rja14/tamper2.ps.gz

[25] R. Anderson and M. Kuhn, "Tamper resistance - a cautionary note," *The Second USENIX Workshop on Electronic Commerce Proceedings*, pp. pp. 1–11, 1996.

[26] Jameco Electronics, "Pc-multiscope (part 142834)," *Catalog, p.103*, 1999.

[27] Scrofanoss, R. Seonil Choi Prasanna, V.K., "Energy efficiency of fpgas and programmable processors for matrix multiplication," *In Proceeding of IEEE FPT, 2002*, 2002.

[28] Peter Waldeck and Neil Bergmann, "Evaluating software and hardware implementations of signal-processing tasks in an fpga," *In Proceeding of IEEE ICFPT 2004*, 2004.

[29] Roman Lysecky, Frank Vahid, "A study of the speedups and competitiveness of fpga soft processor cores using dynamic hardware/software partitioning," *In Proceeding of IEEE DATE 2005: 18-23.*, 2005.

[30] Juergen Becker, Michael Huebner, Michael Ullmann, "Power estimation and power measurement of xilinx virtex fpgas:trade-offs and limitations," in *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design*, 2003.

[31] C. Patterson, "High performance des encryption in virtex fpgas using jbits," *In Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'00)*, 2000.

[32] T. Schaffer, A. Glaser, S. Rao and P. Franzon, "A flip-chip implementation of the data encryption standard (des)," *In Proceedings of IEEE Multi-Chip Module Conference (MCMC 97)*, p. 13, 1997.

[33] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Efficient uses of fpgas for implementations of des and its experimental linear cryptanalysis," *IEEE Transactions on Computers*, p. 52(4):473.482, Apr 2003.

[34] A. Pfitzmann and R. Amann, "More efficient software implementations of (generalized) des," *Computers and Security*, p. 12(5):477.500, Aug 1993.

[35] Janaka Deepakumara, Howard M. Heys and R. Venkatesan, "Fpga impementation of md5 hash algorithm," *Electrical and Computer Engineering, 2001*, pp. 919–924, 2001.

[36] Siddika Berna and Frank Gurkaynak and Elisabeth Oswald and Bart Preneel, "Power-analysis attack on an asic aes implementation," *International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2*, p. p.546, 2004.

[37] P. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss and other systems," in *Advances in Cryptology: Proceedings of CRYPTO 96*, August 1996.

[38] J. D. Golic and C. Tymen, "Multiplicative masking and power anaylsis of aes," in *Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, August 2002.

[39] M.-L. Akkar and C. Giraud, "An implementation of des and aes, secure against some attacks," in *Proceedings of 3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, May 2001.

[40] C. Pomerance, "The quadratic sieve factoring algorithm," *Advances in Cryptology, EuroCrypt '84*, pp. 169–182, 1984.

[41] H. J. Kim, W. H. M. Smith, "Factoring large numbers with a programmable hardware implementation of sieving," *ACM*, 2000.

[42] X. Wang and S. Ziavras, "Parallel lu factorization of sparse matrices on fpga-based configurable computing engines," 2004. [Online]. Available: citeseer.ist.psu.edu/wang03parallel.html

[43] Martin Simka, "Fpga implementation of elliptic curve method for factorization," pp. 113–114, 2000.

[44] J. K. Tetsuya Izu and T. Shimoyama, "An fpga implementation of the sieving step with the lattice sieving," *SHARCS'07*, 2007.

[45] Gabriel Southern, Chris Mason, Lalitha Chikkam and Patrick Baier, "Fpga implementation of high throughput circuit for trial division by small primes," *SHARCS 07*, 2007.

[46] Shang, L., Kaviani, A. S., Bathala, K., "Dynamic power consumption in Virtex -II FPGA family," in *In Proceedings of International Symposium on FPGAs*, February 2002.

[47] Wilton, S. et al., "The Impact of Pipelining on Energy per Operation in FPGAs," in *In Proceedings of FPL 04*, 2004.

[48] Juanjo Noguera, Irwin O. Kennedy, "Power Reduction in Network Equipment through Adaptive Partial Reconfiguration," *FPL 2007*, 2007.

[49] Lopez-Buedo S., Garrido J., Boemo E., "Dynamically inserting, operating, and eliminating thermal sensors of FPGA-based systems," *IEEE Transactions on Components and Packaging Technologies, Vol. 25, Issue 4, Dec*, 2002.

[50] D. Bernstein, "Circuits for integer factorization: a proposal," 2001. [Online]. Available: citeseer.ist.psu.edu/bernstein01circuits.html

[51] A. Lenstra and A. Shamir and J. Tomlinson and E. Tromer, "Analysis of Bernstein 's factorization circuit," *Asiacrypt 2002*, pp. 1–26, 2002. [Online]. Available: citeseer.ist.psu.edu/lenstra02analysis.html

[52] S. Bajracharya, D. Misra, K. Gaj, T. El-Ghazawi, "Reconfigurable hardware implementation of mesh routing in number field sieve factorization," *FPT 2004*, 2004.

[53] Willi Geiselmann and Adi Shamir and Rainer Steinw and Eran Tromer, "Scalable hardware for sparse systems of linear equations, with applications to integer factorization," in *Cryptographic Hardware and Embedded Systems; CHES 2005 Proceedings, volume 3659 of Lecture Notes in Computer Science.* Springer, 2005, pp. 131–146.

[54] H. Satyanarayana, "Aes128 cryptographic core," December 2004. [Online]. Available: http://www.opencores.org/

[55] ASICS Inc, "Des/triple des ip cores," September 2001. [Online]. Available: http://www.opencores.org/projects.cgi/web/des/overview

[56] Universidad Rey Juan Carlos, "Systemc/verilog md5," August 2004. [Online]. Available: http://www.escet.urjc.es/ jmartine

[57] Digilent Inc, "Xup and spartan3 development boards, *www.digilentinc.com*," 2006. [Online]. Available: http://www.digilentinc.com/Products/

[58] Altera Inc, "Altera development and evaluation board, de2." [Online]. Available: http://www.altera.com/education/univ/materials/boards/unv-de2-board.html

[59] Agilent Technologies, "6000 series oscilloscopes with megazoom iii technology." [Online]. Available: http://www.agilent.com/

[60] Mentor Graphics, "Modelsim se simulator," 2006. [Online]. Available: http://www.mentor.com/

[61] AMCC, "Amccs powerpc 405EX embedded processor named product of the year by electronic products magazine," January 2008.

[62] M. The MathWorks, "http://www.mathworks.com."

[63] Xilinx Inc, "Xilinx xpower estimator," 2006. [Online]. Available: http://www.xilinx.com/

[64] Altera Inc, "Altera powerplay power estimation tool," 2006. [Online]. Available: http://www.altera.com/

[65] J. Pollard, "A monte carlo method for factorization," *BIT 15*, pp. 331–334, 1975.

[66] Shamus Software Ltd, "Miracl - multiprecision integer and rational arithmetic c library." [Online]. Available: http://www.shamus.ie/

[67] Xilinx Inc., "ISE Design Suite 10.1," 2008. [Online]. Available: http://www.xilinx.com/

[68] Intel Inc, "Vtune performance analyzer 9.0." [Online]. Available: http://www3.intel.com/cd/software/products/asmona/eng/vtune/vpa/219898.htm

[69] A.K. Lenstra, H.W. Lenstra Jr., M.S. Manasse, J.M. Pollard, "The number field sieve," *ACM Symp. on Theory of Computing*, pp. 564–572, 1990.

[70] P.L. Montgomery, "A block lanczos algorithm for ending dependencies over gf(2)," *Advances in Cryptology, Eurocrypt 95*, pp. 106–120, 1995.

[71] D H Wiedemann, "Solving sparse linear equations over finite fields," *IEEE Trans. Inf. Theory*, pp. 54–62, 1986. [Online]. Available: http://dx.doi.org/10.1109/TIT.1986.1057137

[72] Don Coppersmith, "Solving homogeneous linear equations over gf(2) via block wiedemann algorithm," *Mathematics of Computation*, vol. 62, no. 205, pp. 333–350, 1994.

[73] Xilinx Inc., "Aurora Link-layer Protocol," 2008. [Online]. Available: http://www.xilinx.com/

[74] Xilinx Inc, "Module-Based Partial Reconfiguration," 2008. [Online]. Available: http://www.xilinx.com/

[75] Xilinx Inc., "Planahead design analysis tool," 2008. [Online]. Available: http://www.xilinx.com/

CHAPTER

SEVEN

APPENDIX

7.1 List of Publications

7.1.1 Within the context of the Ph.D research

[**Meidanis, FPT07**] D. Meintanis and I.Papaefstathiou. An efficient FPGA-based implementation of Pollard's (rho - 1) factorization algorithm. IEEE International Conference on Field Programmable Technology, Japan, 2007.

[**Meidanis, Reconfig08**]] D. Meintanis and I.Papaefstathiou. Estimation vs Measurement for Power Consumption on Xilinx VirtexII Pro FPGAs. International Conference on ReConFigurable Computing and FPGAs, Mexico, 2008.

[**Meidanis, CCNC09**] D. Meintanis and I.Papaefstathiou. On the Power Consumption of security algorithms employed in wireless networks. IEEE Consumer Communications & Networking Conference, Las Vegas, 2009.

[**Meidanis, FPT09**] D. Meintanis and I.Papaefstathiou. A Modular Partial Reconfigurable System for Factorizing Large Numbers Over GF(2). IEEE International Conference on Field Programmable Technology, Australia, 2009.

[**Meidanis, To be submited**] D. Meintanis and I.Papaefstathiou. On the Power Consumption of Security Algorithms on Field-programmable Gate Arrays, ACM Transactions on Computational Logic.

[**Meidanis, To be submited**] D. Meintanis and I.Papaefstathiou. Utilizing runtime reconfiguration for power reduction.

7.1.2 Other Publications

[**Kornaros, ICCEDigest08**] G.Kornaros, D.Meintanis, I.Papaefstatiou, S.Chantzandroulis and S.Blionas. Architecture of a Consumer Lab-on-Chip for Pharmacogenomics. IEEE Consumer Electronics (ICCE), Las Vegas, 2008.

[**Platsis, RSP09**] M.Platsis, I.Papaefstation and D.Meintanis. Design and Implementation of an UWB Digital Transmitter. In IEEE International Symposium on Rapid System Prototyping, Paris, France, 2009.