N° Ordre...../Institut/UMBB/2012

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA - Boumerdes -

Institute of Electrical and Electronic Engineering

Department of Electronics

Report Presented in Partial Fulfilment of the Requirement of the Degree of

'MAGISTER'

In **Electronics**

Option: **Communication**

Title:

Implementation of IP Cores Dedicated To Cryptography

Presented By:

- ALOUANI Brahim

President:	M[r.] REFOUFI Larbi	Professor	UMBB, Boumerdes
Members:	M[r.] BENTERZI Hamid	MC (A)	UMBB, Boumerdes
	M[r.] KARA Redouane	MC (A)	UMMTO, T.OUZOU
Supervisor:	M[r.] Djamel Benazzouz	Professor	UMBB, Boumerdes

January 2012

Contents

Abstract

This work consists to implement IP cores dedicated to cryptography, using an FPGA as a hardware working platform and VHDL as a hardware description language.

The cryptography can be classified into two main parts. Secret key and public key encryption. The last one is more complicated because it uses hard arithmetic; the RSA (Rivest, Shamir, and Adleman), DH (Diffie-Hellman Key Agreement) and the elliptic curve algorithm (ECC) are the most used public key protocols in cryptography.

The modular exponentiation is the core operation in these protocols. The security of these systems depends directly on the key bit lenght, in practice (between 160 and 1024 bits or more in some applications). A high bit length and complicated operations as the modular arithmetic make the hardware solution better in different applications.

The modular exponentiation can be done by successive operations of modular multiplication, that is why we chose modular multiplication as a problematic in this report.

Montgomery and the interleaved are the most modular product algorithms appropriate to hardware design. In our case; we chose an optimized version of Montgomery.

Key words: cryptography, public key cryptography, FPGA, VHDL, modular multiplication, modular multiplication, Montgomery multiplication.

Résumé

Ce travail consiste à faire une implantation matérielle des cors IP dédiée pour la cryptographie, utilisant une Platform configurable (FPGA), et le langage de description matérielle (VHDL).

La cryptographie peut être classée en deux axes principaux, cryptographie à clé secret et cryptographie à clé publique. La dernière est plus difficile à cause des arithmétiques complexes utilisées à ses opérations. Le RSA (Rivest, Shamir, et Adleman), DH (Diffie-Hellman pour l'agreement de la clé) et ECC (cryptographie par les courbes elliptiques) algorithmes sont les plus utilisées.

L'exponentiation modulaire est l'opération principale dans ces différents protocoles, la sécurité dans ces systèmes dépend directement de la taille de la clé utilisée, en pratique (entre 160 et 1024 bits et plus pour quelques application).Donc la grande taille des clés et la complexité des opérations modulaires demandent un temps de calcule élevé, c'est pour cette raison que la solution matériel est très approprié.

L'exponentiation modulaire peut être faite par des opérations successives de la multiplication modulaire.

A notre cas nous avons opté comme problématique L'implantation de la multiplication modulaire sur FPGA.

Montgomery et blackley (en anglais connus par «interleaved ») sont deux techniques utilisées pour la multiplication modulaire sans division, et sont aussi appropriée pour l'implantation matérielle. Dans notre travail on a implanté ces deux techniques.

Mots clés : Cryptographie, Cryptographie à clés publique, FPGA, VHDL, Exponentiation modulaire, Multiplication modulaire, Multiplication de Montgomery.

ملخص

يتمثل هذا العمل في تثبيت مجسم حسابي خاص بمجال التشفير, حيث يتم استعمال الدارة المبرمجة FPGA و لغة الوصف VHDL

عملية التشفير يمكن تقسيمها إلى قسمين أساسيين, تشفير بمفتاح سري, و تشفير بمفتاح عمومي و معروف, هذا الأخير يعتبر أكثر صعوبة بسبب العمليات الحسابية المعقدة التي يستعملها.

خوارزميات التشفيرRSA و DH وECC تعتبر الأكثر شهرة و استعمالا, نسبة الأمان في هذه الخوارزميات تتعلق مباشرة بطول المفتاح, تطبيقيا يتم استعمال مفتاح (من 160 إلى 1024 بيت أو أكثر), إذن فطول المفتاح و صعوبة العمليات يجعل اختيار التثبيت على الهاردوور مبرر لرفع سرعة انجاز الحسابات.

عملية الأس التردد هي العملية الأساسية في هذه الخوارزميات و يمكن تثبيتها بواسطة عمليات جذاء ترددي متتالية و لذلك فان اهتمامنا سينصب على عملية الجذاء الترددي

مونتفومري. و بلاكلي هما الطريقتان الأنسب للتثبيت على الهاردوور و في حالتنا قمنا باستعمال طريقة مطورة عن مونتفومري.

كلمات مفاتيح: التشفير, التشفير بمفتاح عمومي, الدارات المبرمجة, لغة الوصف الهاردوور, عملية الجذاء الترددي, عملية الأس الترددي, عملية جذاء مونتفومري.

Dedication

This modest work is dedicated

To my beloved mother, who has been a source of inspiration and encouragement to me throughout my life.

To my beloved father who gave everything they ever had, so i could pursue my dream;

To my sisters and brothers

To all my family;

To all my friends without exception

Acknowledgement

First of all, I thank Allah for giving me strength and ability to complete this study. Furthermore, I would like to express my deepest gratitude to my advisors *Pr. Djamel Benazzouz* and *Mr. Gougam Abd el Hamid* for their powerful guidance, motivation, valuable suggestions, support and attention throughout this research. I thank all my friends for sharing their experiences and knowledge. Special thank to *Mr. Khaled Belhoute* for his useful help during my research. Finally, I am deeply grateful to my parents for their trust everlasting love, care, and indefectible support morally and materially during all my years of studying.

LIST OF TABLES

LIST OF FIGURES

Introduction

The rising growth of data communication and electronic transactions over the internet has made security to become the most important issue over the network. To provide modern security features, One of the main objectives of cryptography consists of providing *confidentiality*, which is a service used to keep secret publicly available information from all but those authorized to access it. There exists many ways to provide secrecy. They range from physical protection to mathematical solutions, which render the data unintelligible. The latter uses *encryption/decryption* methods, public-key cryptosystems are widely used. The widely used algorithms for public-key cryptosystems are RSA, Diffie-Hellman key agreement (DH), the digital signature algorithm (DSA) and systems based on elliptic curve cryptography (ECC). All these algorithms have one thing in common: they operate on great numbers (e.g. 160 to 2048 bits). Long word lengths are necessary to provide a sufficient amount of security, but also account for the computational cost of these algorithms.

The modular multiplication is used to perform modular exponentiations, which, in their turn, are used by several public-key cryptosystems. The performance of public-key cryptosystems is primarily determined by the implementation's efficiency of the modular exponentiation. So, consequently, modular multiplication is an important factor in these systems.

The Montgomery multiplication algorithm is considered to be the fastest algorithm to compute X**Y mod M* in computers when the values of *X*, *Y* and *M* are large. Another efficient algorithm for modular multiplication is the interleaved modular multiplication algorithm
This thesis was organized as follows:

In chapter one, we presented the working platform (FPGA and its hardware description language VHDL) and appear their characteristics.

Chapter two deals with cryptography; we presented its meaning, goals, classes and techniques. After that we concentrated about public key encryption because it is well known and used more, due to its characteristics.

Third chapter presents briefly the most important public key cryptographic algorithms as the RSA (Rivest, Shamir, and Adleman), DH (Diffie-Hellman Key Agreement) and the elliptic curve algorithm (ECC), and shows that the modular exponentiation is the core operation in these different protocols. Each hardware design and optimization of modular product will improve the encryption /decryption process.

A study about the well known and optimized algorithms and architectures, as Montgomery and interleaved algorithms are presented in chapter four. For each algorithms we presented the standard algorithms and its optimized versions, it exists many other techniques but we selected these two algorithms because they are suitable for hardware implementation.

After that in chapter five we described the Montgomery technique in VHDL language, we simulated and implement our architectures using simulation tool (modelsim) and implementation tool (ISE of XILINX), after that we presented the characteristics and the results of our design.

Finally, we concluded of our work and gave perspectives.

1.1 Introduction

The historical roots of FPGAs are in complex programmable logic devices (CPLDs) of the early to mid 1980s. **Ross Freeman**, Xilinx co-founder, invented the field programmable gate array in 1984. CPLDs and FPGAs include a relatively large number of programmable logic elements. CPLD logic gate densities range from the equivalent of several thousand to tens of thousands of logic gates, while FPGAs typically range from tens of thousands to several million.

The primary differences between CPLDs and FPGAs are architectural. A CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers. The result of this is less flexibility, with the advantage of more predictable timing delays and a higher logic-to-interconnect ratio. The FPGA architectures, on the other hand, are dominated by interconnect. This makes them far more flexible (in terms of the range of designs that are practical for implementation within them) but also far more complex to design for.

Another notable difference between CPLDs and FPGAs is the presence in most FPGAs of higher-level embedded functions (such as adders and multipliers) and embedded memories. A related, important difference is that many modern FPGAs support full or partial in-system reconfiguration, allowing their designs to be changed "on the fly" either for system upgrades or for dynamic reconfiguration as a normal part of system operation. Some FPGAs have the capability of partial re-configuration that lets one portion of the device be re-programmed while other portions continue running.

1.2 Programmable logic devices (PLDs)

Programmable logic devices (PLDs) were introduced in the mid 1970s .the idea was to construct combinational logic circuits that were programmable .however, contrary to microprocessors, which can run a program but posses a fixed hardware, the programmability of PLDs was intended at the hardware level .in other words, a PLD is a general purpose chip whose hardware can be reconfigured to meat particular specification.

The first PLD were called PAL (programmable array logic) or PLA (programmable logic array), depending on the programming scheme .they used only logic gates (no flip-flop),

thus allowing only the implementation of combinational circuits .to circumvent this problem registered PLDs were launched soon after, which included one flip-flop at each output of the circuit .with them, simple sequential functions could then be implemented as well.

In the beginning of the 1980s,additional logic circuit was added to each PLD output .the new output cell, called macro cell ,contained(besides the flip-flop) logic gates and multiplexers .moreover, the cell itself was programmable, allowing several modes of operations .Additionally, it provide a return (feedback) signal from the output of the circuit to the programmable array, which gave the PLD greater flexibility .this new PLD structure was called generic PAL(GAL).a similar architecture was known as PALCE(PAL CMOS Electrically erasable/programmable) device.

All these chips (PAL, PLA, registered PLD and GAL/PALCE) are now collectively referred to as SPLDs (Simple PLD).

Later, several GAL devices was fabricated on the same chip, using a more sophisticated routing scheme, more advanced silicon technology, and several additional features (like JTAG support and interface to several logic standards).this approach became known as CPLD (complex PLD).CPLD are currently very popular due to their high density, high performance and low cost .

Finally in the mid 1980s ,FPGA(Field Programmable Gate Array) were introduced .FPGA differ from CPLD in architecture ,technology, built in features ,and cost .they are aimed mainly at the implementation of large size ,high performance circuits.[1]

A summery of the evolution of PLDs is presented in the table below.

<table>
<tr><td rowspan="3">PLDs</td><td rowspan="2">Simple PLD (SPLDs)</td><td>PAL
PLA
Registered PAL/PLA
GAL</td></tr>
<tr><td colspan="2">Complex PLD (CPLD)</td></tr>
<tr><td colspan="2">FPGA</td></tr>
</table>

Table 1.1: PLDs classification

A final remark : all CPLDs simple or complex are non volatile .they can be OTP (One Time Programmable),in which case fuses or antifuses are used ,or can be reprogrammable ,with EEPROM or Flash memory(flash is the technology of choice in most new devices)

FPGAs in the other hand are mostly volatile ,for they make use of SRAM to store the connections, in which case a configuration ROM is necessary to load the interconnects at power up, there are .however non volatile options like the use of antifuse .

1.2.1 SPLDs (Simple PLD)

As mentioned above PLA, PAL and GAL DEVICES are collectively called simple PLDs .a description of each of these architecture flows.

PAL Devices

PAL (programmable array logic) chips were introduced by monolithic memories in the mid 1970s its basic architecture is illustrated symbolically in figure (1.1) where the little circles represent programmable connections .as can be seen the circuit is composed of a programmable array of AND gates followed by a fixed array of OR gates.

The implementation of figure(1.1) was based on the fact that any combinational function can be represented by a sum of products (SOP);that is if $a_1, a_2, ..., a_N$ are the logic inputs, then any combinational output x can be computed as

$$x = m_1 + m_2 + ... + m_M$$

Where $m_i = f(a_1, a_2, ..., a_N)$ are the minterms of the function x for example

$$x = a_1 \bar{a}_2 + a_2 a_3 \bar{a}_4 + \bar{a}_1 \bar{a}_2 a_3 \bar{a}_5$$

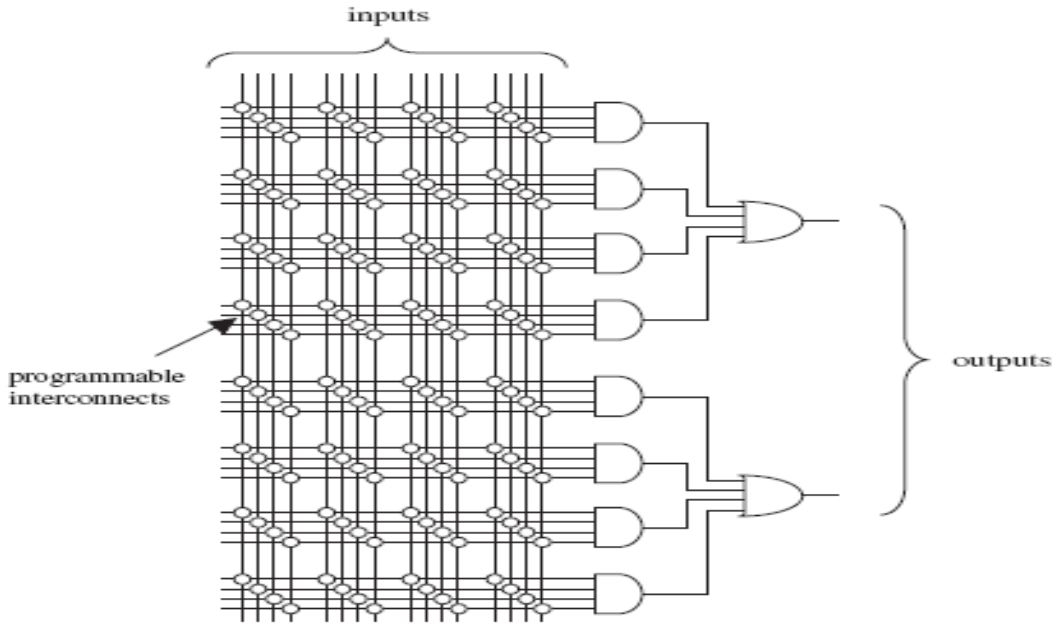

Figure 1.1: *Illustration of PAL architecture*

Hence, the products (minterms) can be obtained by means of AND gates, whose outputs are then connected to an OR gate to compute their sum, thus implementing the SOP equation described above.

The main limitation of this approach was the fact that is allowed only the implementation of combinational functions. to circumvent this problem, registered PALs. Were launched toward the end of the 1970s.these included a flip-flop at each output (after OR gates in figure 1.1), thus allowing the implementation of sequential functions as well.

An example of a than popular PAL chips is the PAL 16L18 device ,which contained 16 inputs and 8 outputs (through only 18I/O pins were indeed available ,because it was a 20pin DIP package ;there were ten IN pins, two OUTpins,and six IN/OUT pins(bidirectional),plus VCC and GND).its registered counterpart was the 16R8 chip(where R stands for Registered).

The early technology employed in the fabrication of PAL devices was bipolar, with 5V supply and current consumption (with open output) around 200mA.the maximum frequency was of the order of 100MHz,and the programmable cells were of PROM(fuse links) or EPROM type.

PLA devices

PLA (Programmable Logic Array) chips were also introduced in the mid 1970s .the basic architecture of PLA comparing it with the PAL. The only fundamental difference between them is that while a PAL has programmable AND connections and fixed OR connections .both are programmable in the PLA .the obvious advantage was greater flexibility .however, higher time constants at the internal nodes lowered the circuit speed.

An example of a then popular PLA chip is the signetics PLS161 device. It contained 12 inputs and 8 outputs, being the AND inputs and the OR inputs all programmable. at the outputs, additional programmable XOR gates were also available.

The technology then employed in the fabrication of the PLAs was the same as those of PALs through PLAs are also absolute now, they reappeared recently as a building block in the first family of low power CPLDs. [2]

GAL Devices

The GAL devices (Generic PAL) architecture was introduced by lattice in the beginning of the 1980s .it contained several important improvements over the first PAL devices :

first, a more sophisticated output cell (macrocell) was constructed ,which included beside flip-flop ,several gates and multiplexers ;second, the macrocell itself was programmable, allowing several modes of operations; third, a ‘return’ signal from the output of the macrocell to the

programmable array was also included ,conferring the circuit moreversatility;fourth, EEPROM was employed instead of PROM or EPROM. An electronic signature for identification was also included.

As mentioned earlier, GAL is the only SPLD (simple PLD) still manufactured in a standalone package. Additionally, it also serves as a basic building block in the construction of most CPLDs [1]

1.2.2 CPLD (complex PLD)

the basic approach in the construction of a CPLD is illustrated in figure 1.2 .as shown ,it consists of several PLDs (in general of GAL type) fabricated on a single chip, with a programmable switch matrix used to connect them together and to the I/O pins .Moreover ,CPLDs normally contain a few additional features ,like JTAG support and interface to other logic standards (1.8 V,2.5 V,5 V,etc..). [1]

Regarding figure 1.2,as an example we can mention the Xilinx XC9500 CPLD .it consists of n PLDs,each resembling a 36V18 GAL device, where n=2,4,6,8,12or 16 .

Several companies manufacture CPLDs, like Altera, Xilinx, Lattice, Atmel, Cypress, etc.

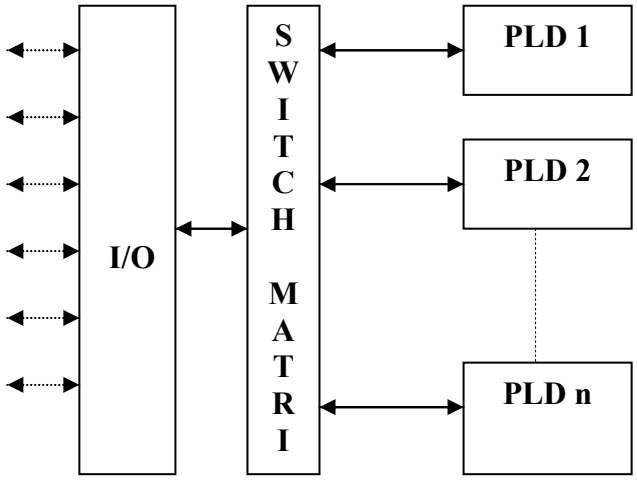

Figure 1.2: *CPLD Architecture*

1.3 FPGA (Field Programmable Gate Array)

Before the advent of programmable logic, custom logic circuits were built at the board level using standard components, or at the gate level in expensive application-specific (custom) integrated circuits. The FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are

interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnects form a fabric of basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to create the desired circuit. [3]

Or by other definition Field Programmable Gate Array (FPGAs) are digital integrated circuits (ICs) that contain configurable (programmable) logic blocks (CLB) along with configurable interconnects between these blocks .depending on the way in which they are implemented, some FPGAs may only be programmed a single time, while others may be programmed over and over again .not surprising a device that can be programmed only one time is referred to as "one-time-programmable», the 'Field Programmable 'portion of the FPGA's name refers to the fact that its programming take place "in a field" .As opposed to devices whose internal functionality is hard-wired by the manufacturer. If a device is capable of being programmed while remaining resident in a higher level system, it is referred to as being In-system programmable (SP)

The main components of FPGAs are presented in figure 1.3.

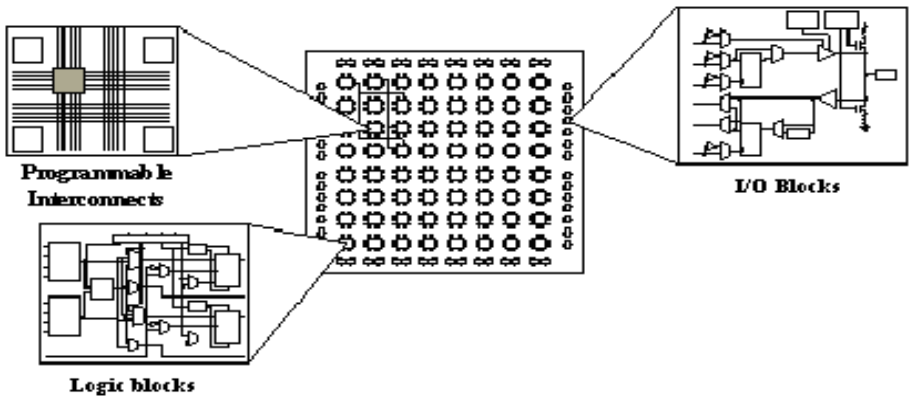

Figure 1.3: *internal architecture of FPGA*

The figure 1.4 presents the four class architecture of FPGAs in the industrial site of microproducts.

1. Symmetrical Array
2. Row based
3. Sea of gates
4. Hierarchical PLD

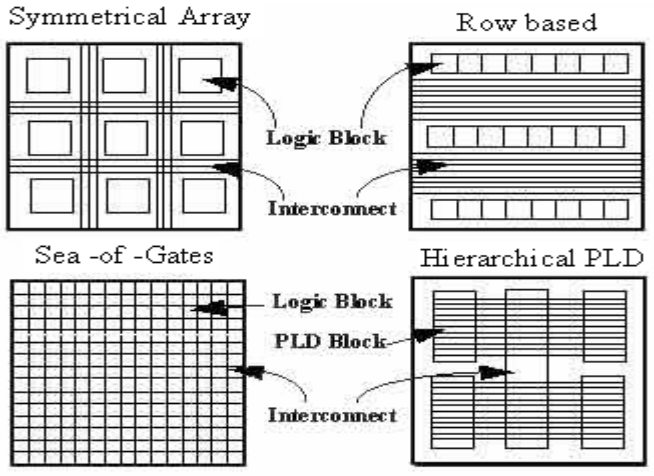

Figure 1.4 *classes of FPGA architecture*

What does 'Field Programmable' mean?

Field Programmable means that the FPGA's function is defined by a user's program rather than by the manufacturer of the device. A typical integrated circuit performs a particular function defined at the time of manufacture. In contrast, the FPGA's function is defined by a program written by someone other than the device manufacturer. Depending on the particular device, the program is either 'burned' in permanently or semi-permanently as part of a board assembly process, or is loaded from an external memory each time the device is powered up. This user programmability gives the user access to complex integrated designs without the high engineering costs associated with application specific integrated circuits.

What does a logic cell (logic block CLB) do?

The logic cell architecture varies between different device families. Generally speaking, each logic cell combines a few binary inputs (typically between 3 and 10) to one or two outputs according to a Boolean logic function specified in the user program. In most families, the user also has the option of registering the combinatorial output of the cell, so that clocked logic can be easily implemented. The cell's combinatorial logic may be physically implemented as a small look-up table memory (LUT) or as a set of multiplexers and gates. LUT devices tend to be a bit more flexible and provide more inputs per cell than multiplexer cells at the expense of propagation delay. Logic block in an FPGA can be implemented in

ways that differ in number of inputs and outputs, amount of area consumed, complexity of logic functions that it can implement total number of transistors that it consumes.

The figure 1.5 presents CLB architecture of some Xilinx FPGA and gives us an idea about the main body of some Xilinx logic blocks [4]

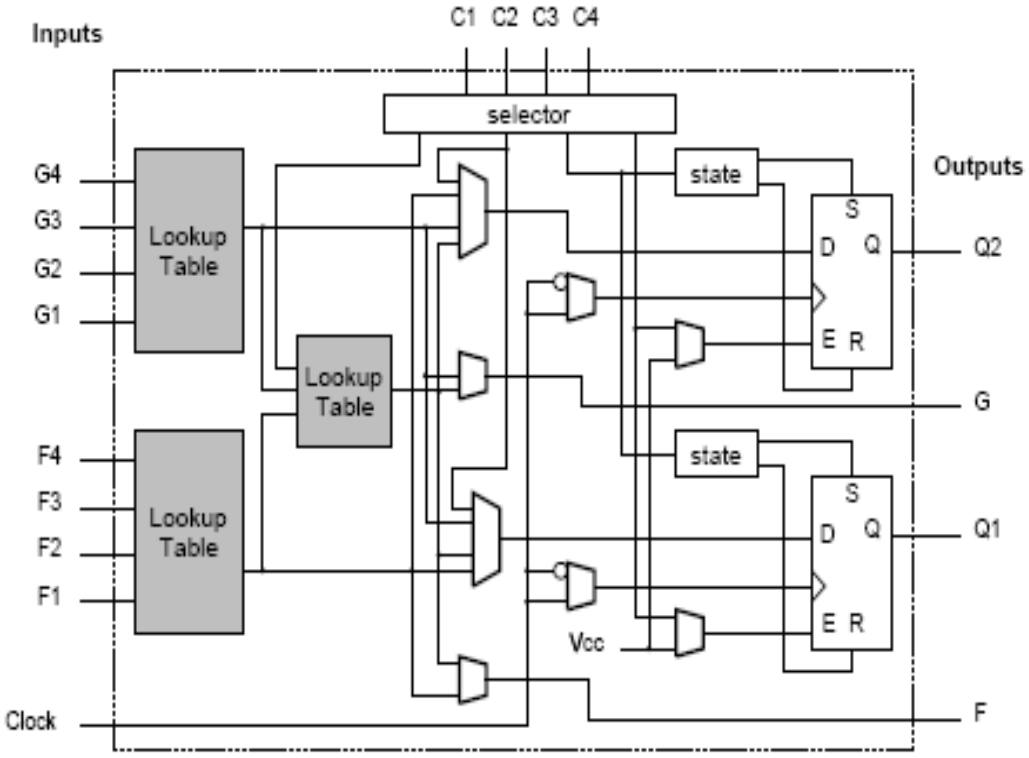

Figure 1.5: *Xilinx XC4000 Configurable Logic Block (CLB).*

1.4 FPGA Technology:

One of the most common field-programmable elements is programmable logic devices (PLDs). PLDs concentrate primarily on two-level, sum-of-products implementations of logic functions. They have simple routing structures with predictable delays. Since they are completely prefabricated, they are ready to use in seconds, avoiding long delays for chip fabrication. Field-Programmable Gate Arrays (FPGAs) are also completely prefabricated, but instead of two-level logic they are optimized for multi-level circuits. This allows them to handle much more complex circuits on a single chip, but it often sacrifices the predictable delays of PLDs. Note that FPGAs are sometimes considered another form of PLD, often under the heading Complex Programmable Logic Device (CPLD).

Just as in PLDs, FPGAs are completely prefabricated, and contain special features for customization. These configuration points are normally SRAM cells, EPROM, EEPROM, or antifuses. Antifuses are one-time programmable devices (Figure 1.6), which when "blown" create a connection, while when "unblown" no current can flow between their terminals (thus, it is an "anti"-fuse, since its behaviour is opposite to a standard fuse).

Because the configuration of an antifuse is permanent, antifuse-based FPGAs are one-time programmable, while SRAM-based FPGAs are reprogrammable, even in the target system. Since SRAMs are volatile, an SRAM-based FPGA must be reprogrammed every time the system is powered up, usually from a ROM included in the circuit to hold configuration files. Note that FPGAs often have on-chip control circuitry to automatically load this configuration data. EEPROM/EPROM (Figure 1.7) is devices somewhere between SRAM and antifuse in their features. The programming of an EEPROM/EPROM is retained even when the power is turned off, avoiding the need to reprogram the chip at power-up, while their configuration can be changed electrically. However, the high voltages required to program the device often means that they are not reprogrammed in the target system. [4]

SRAM cells are larger than antifuses and EEPROM/EPROM, meaning that SRAM-based FPGAs will have fewer configuration points than FPGAs using other programming technologies. However, SRAM-based FPGAs have numerous advantages. Since they are easily reprogrammable, their configurations can be changed for bug fixes or upgrades. Thus they provide an ideal prototyping medium. Also, these devices can be used in situations where they can expect to have numerous different configurations, such as multi-mode systems and reconfigurable computing machines. Because antifuse-based FPGAs are only one-time programmable, they are generally not used in reprogrammable systems. EEPROM/EPROM devices could potentially be reprogrammed in system, although in general this feature is not widely used.

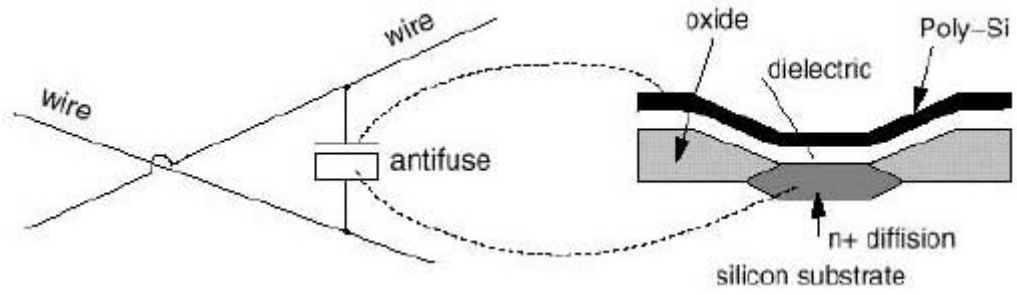

Figure 1.6: *Antifuse technology*

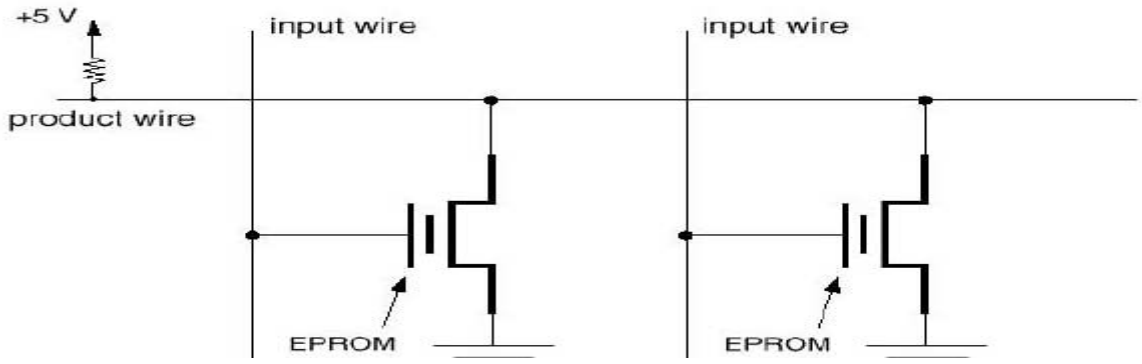

Figure1.7*: EPROM Technology*

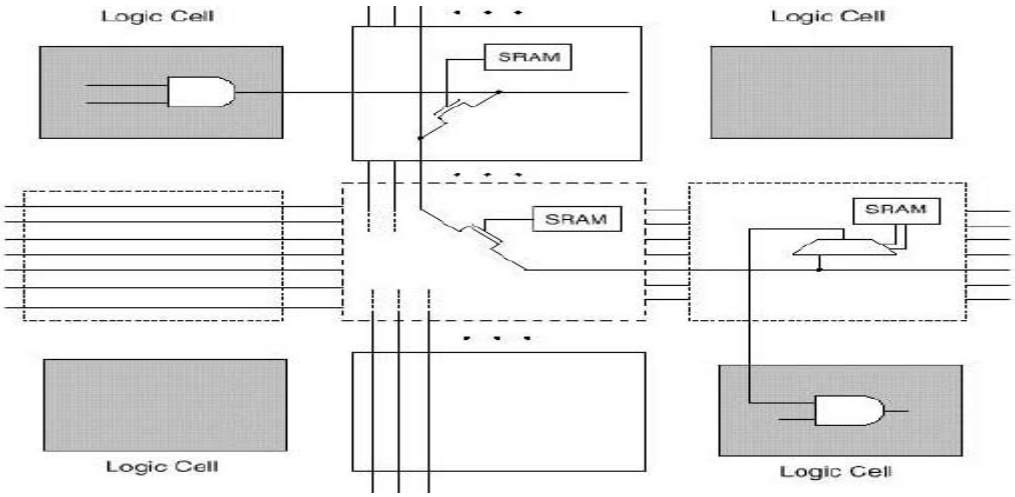

Figure 1.8: *SRAM Technology*

The following table shows the different technologies and their characteristics

Name	Re-programmable	Volatile	technology
Fuse	**No**	**No**	**Bipolar**
EPROM	**Yes** **Out of circuit**	**No**	**UVCMOS**
EEPROM	**Yes** **In circuit**	**No**	**EECMOS**
SRAM	**Yes** **In circuit**	**Yes**	**CMOS**
Antifuse	**No**	**No**	**CMOS+**

Table 1.2: Classification of FPGAs technologies

1.5 VHDL

1.5.1 Definition

VHDL stands for **V**HSIC (Very High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage. VHDL development was initiated originally from the American Department of Defense (DoD), In the mid-1980's, with the goal to develop very high-speed integrated circuit. They requested a language for describing a hardware, which had to be readable for machines and humans at the same time and strictly forces the developer to write structured and comprehensible code, so that the source code itself can serve as a kind of specification document. Most important was the concept of concurrency to cope with the parallelism of digital hardware. Sequential statements to model very complex functions in a compact form were also allowed. In 1987, VHDL was standardized by the American Institute of Electrical and Electronics Engineers (IEEE) for the first time with the first official update in 1993. VHDL has become now one of industry's standard languages used to describe digital systems. The other widely used hardware description language is Verilog. Both are powerful languages that allow us to describe and simulate complex digital systems. A third HDL language is ABEL (Advanced Boolean Equation Language) which was specifically designed for Programmable Logic Devices (PLD). ABEL is less powerful than the other two languages and is less popular in industry. Although these languages look similar as conventional programming languages, there are some important differences. A hardware description language is inherently parallel, i.e. commands, which correspond to logic gates, are executed (computed) in parallel, as soon as a new input arrives. A HDL program mimics the behavior of a physical, usually digital, system. It also allows incorporation of timing specifications (gate delays) as well as to describe a system as an interconnection of different components [5].

1.5.2 VHDL Structural Elements

VHDL is a very strict language in which hardly a cryptic programming style is possible (as it is the case with the programming language C). Every signal, for example, has to possess a certain data type, it has to be declared at a certain position, and it only accepts assignments from the same data type.

The main units in VHDL are entities, architectures, configurations and packages (together with package bodies).

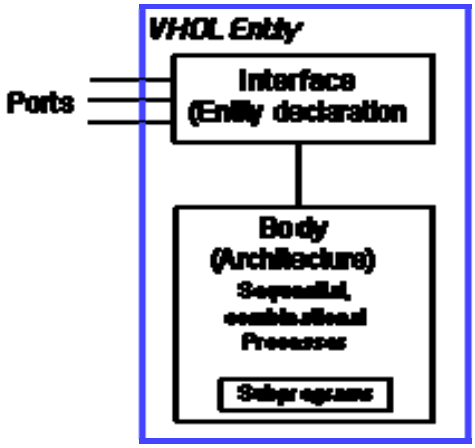

Figure 1.9: A VHDL entity consisting of an interface (entity declaration) and a body (architecture description).

While an entity describes an interface consisting of the port list most of the time, architecture contains the description of the function of the corresponding module. In general, a configuration is used for simulation purposes, only. In fact, the configuration is the only simulatable object in VHDL as it explicitly selects the entity/architecture pairs to build the complete model. Packages hold the definition of commonly used data types, constants and subprograms. By referencing a package, its content can be accessed and used. Another important construct is the process. While statements in VHDL are generally concurrent in nature, this construct allows for a sequential execution of the assignments. The process itself, when viewed as a whole object, is concurrent. In reality, the process code is not always executed. Instead, it waits for certain events to occur and is suspended most of the time.

A library in VHDL is the logical name of a collection of compiled VHDL units (object code). This logical name has to be mapped by the corresponding simulation or synthesis tool to a physical path on the file system of the computer.

To make a functional test of a VHDL model, a testbench can be written also in VHDL, which delivers the verification environment for the model. In it, stimuli are described as input signals for the model, and furthermore the expected model responses can be checked. The testbench appears as the top hierarchy level for simulation, and therefore has neither input- nor output ports [5].

VHDL uses reserved keywords that cannot be used as signal names or identifiers. Keywords and user-defined identifiers are case insensitive. Lines with comments start with two adjacent hyphens (--) and will be ignored by the compiler. VHDL also ignores line breaks and extra spaces. VHDL is a strongly typed language which implies that one has always to declare the type

1.6 Design flow

The major utility of VHDL is to allow the synthesis of a circuit in a programmable device as FPGA or PLD. The steps followed during such a project are summarized in figure 1.10 bellow.

The design starts by writing the VHDL code, which is saved in a file with the extension (.vhd) and it has the same name as its ENTITY‘s name. The first step in the synthesis process is compilation which is the conversion of the high-level VHDL code describing the circuit at the register Ternsfer Level (RTL), into a netlist at the gate level. The second step is the optimization, which is performed on the gate-level netlist for speed or for area. At this stage, the design can be simulated. Finally, a place and route software will generate the physical layout for an FPGA/PLD chip or will generate the masks for an ASIC [5].

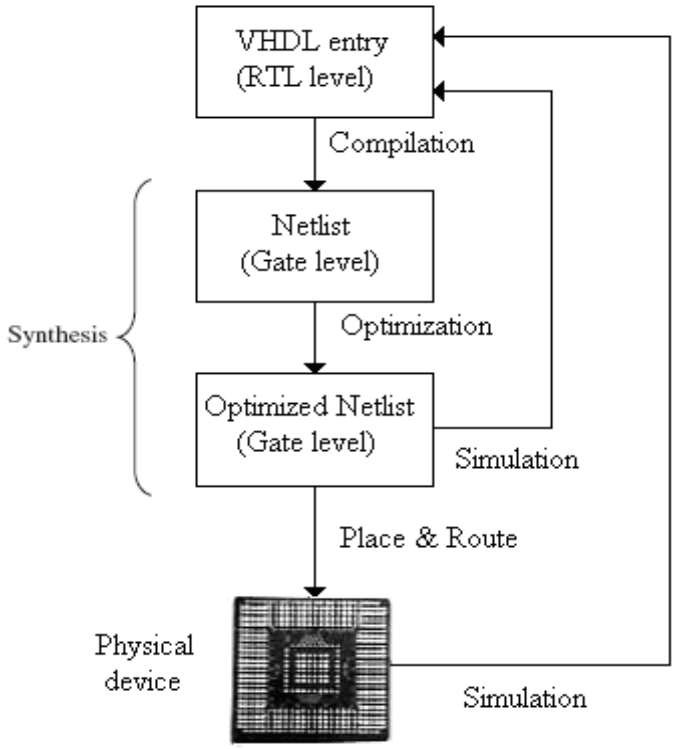

Figure 1.10: VHDL design flow

1.7 Synthesis- placement and simulation software

The following table contains some synthesis, placement and simulation tools

Tools	**Applications**
ISE +ModelSim	**Xilinx CPLDs and FPGAs**
MaxPlus+advanced syhthesis software	**Altera CPLDs and FPGAs**
Quartus	**Altera CPLDs and FPGAs**

Table 1.3: FPGAs & CPLDs softwar tools

Xilinx ISE is a comprehensive synthesis and implementation environment for Xilinx programmable devices. ModelSim (from Model technology) is also provided as a part of the package .the former is employed for circuit synthesis and design implementation, while the latter is used for simulation.

This is very brief tutorial, which is divided into five parts

1- Entering VHDL Code
2- Synthesis and implementation
3- Creating test benches
4- Simulation(with ModelSim)
5- Physical realization

ISE controls all aspects of the design flow. Through the Project Navigator interface, you can access all of the various design entry and design implementation tools. You can also access the files and documents associated with your project. Project Navigator maintains a flat directory structure; therefore, the user must maintain revision control through the use of snapshots [1].

1.8 Conclusion:

The FPGAs are really very interested technology because it is a VLSI chip that can be programmed over and over again, and then this new design methodology of FPGA offers more performance than the software implementation, an easiness of work, shortest possible time and a low cost, moreover FPGA make any process very fast, this speed due to the application specific design these can be implemented by FPGA and parallelism of operations FPGA solve the most problems and risks of implementation because we have the chance of simulating the design before.

One of the excellent characteristics of the FPGAs that when we want to improve or modify a design we can just update the similar previous designs.

From the economic point of view FPGAs are fast in implementation and with a low cost.

2.1 Introduction

Cryptography has had an interesting history and has undergone many changes through the centuries. It seems that keeping secrets has been important throughout the ages of civilization for one reason or another. Keeping secrets gives individuals or groups the ability to hide true intentions, gain a competitive edge, and reduce vulnerability.

The changes that cryptograph has undergone throughout history closely follow the advances in technology. Cryptography methods began with a person carving messages into wood or stone, which were then passed to the intended individual who had the necessary means to decipher the messages. This is a long way from how cryptography is being used today. Cryptography that used to be carved into materials is now being inserted into streams of binary code that passes over network wires, Internet communication paths, and airwaves.

In the past, messengers were used as the transmission mechanism, and encryption helped protect the message in case the messenger was captured. Today, the transmission mechanism has changed from human beings to packets carrying 0's and 1's passing through network cables or open airwaves. The messages are still encrypted in case an intruder captures the transmission mechanism (the packets) as they travel along their paths.

The growth of the Internet as a vehicle for secure communication and electronic commerce has brought cryptographic processing performance to the forefront of high throughput system design. Cryptography provides the mechanisms necessary to implement accountability, accuracy, and confidentiality in communication. This trend will be further underscored with the widespread adoption of secure protocols such as secure IP (IPSEC) and virtual private networks (VPNs). Efficient cryptographic processing, therefore, will become increasingly vital to good system performance.

2.2 History of Cryptography

Cryptography has roots that began around 2000 B.C. in Egypt when hieroglyphics were used to decorate tombs to tell the story of the life of the deceased. The practice was not as much to hide the messages themselves, but to make them seem more noble, ceremonial, and majestic. Encryption methods evolved from being mainly for show into practical applications used to hide information from others.

A Hebrew cryptographic method required the alphabet to be flipped so that each letter in the original alphabet is mapped to a different letter in the flipped alphabet. The encryption

method was called *atbash*. An example of an encryption key used in the atbash encryption scheme is shown in following:

```
ABCDEFGHI JK LMNOPQ R STU VW XYZ
ZYXWVUTSR QP ONMLKJ I HGF ED CBA
```

For example, the word "security" is encrypted into "hvxfirgb." What does "xrhhk" come out to be? This is a ***substitution cipher***, because one character is replaced with another character. This type of substitution cipher is referred to as a ***monoalphabetic substitution*** because it uses only one alphabet, compared to other ciphers that use multiple alphabets at a time.

This simplistic encryption method worked for its time and for particular cultures, but eventually more complex mechanisms were required.

In another time and place in history, Julius Caesar developed a simple method of shifting letters of the alphabet, similar to the atbash scheme. Today this technique seems too simplistic to be effective, but in that day not many people could read in the first place, so it provided a high level of protection. The evolution of cryptography continued as Europe refined its practices using new methods, tools, and practices throughout the Middle Ages, and by the late 1800s, cryptography was commonly used in the methods of communication between military factions.

During World War II, simplistic encryption devices were used for tactical communication, which drastically improved with the mechanical and electromechanical technology that provided the world with telegraphic and radio communication. The rotor cipher machine, which is a device that substitutes letters using different rotors within the machine, was a huge breakthrough in military cryptography that provided complexity that proved difficult to break. This work gave way to the most famous cipher machine in history to date: Germany's ***Enigma*** machine. The Enigma machine had three rotors, a plug board, and a reflecting rotor.

As computers came to be, the possibilities for encryption methods and devices advanced, and cryptography efforts expanded exponentially. This era brought unprecedented opportunity for cryptographic designers and encryption techniques. The most well-known and successful project was ***Lucifer***, which was developed at IBM. Lucifer introduced complex mathematical equations and functions that were later adopted and modified by the U.S. National Security Agency (NSA) to come up with the U.S. Data Encryption Standard (DES). DES has been adopted as a federal government standard, is used worldwide for financial transactions, and is imbedded into numerous commercial applications. DES has had a rich history in computer-oriented encryption and has been in use for over 20 years. [6]

2.3 Basic terminology and concepts

The Term cryptography is derived from the Greek word Kryptos. Kryptos is used to describe anything that is hidden, obscured, veiled, secret or mysterious. Cryptography, over the ages, has been an art practiced by many who have devised ad-hoc techniques to meet various information security requirements. However over the past twenty years, cryptography has moved from an art perspective to a science perspective. The definition of cryptography given by A. Menezes, P. van Oorschot, and S. Vanstone [6] is as follows:

Cryptography is the study of mathematical techniques related to aspects of information security such as privacy, data integrity, entity authentication, and data origin authentication.

The word *cryptography* refers to the science of keeping secrecy of messages exchanged between a sender and a receiver over an insecure channel. The objective is achieved by encoding data so that it can only be decoded by specific individuals. The original message M being wanted to be sent is called *plaintext* since it is clearly intelligible, whereas the term used to refer to the message C being transited over an insecure channel is *ciphertext*. The process E of transforming a plaintext into a ciphertext is called *encryption*, while the opposite procedure D that turns a ciphertext into a plaintext at the receiver's side is said *decryption*. In symbols

$$E(M) = C \qquad (II.1)$$
$$D(C) = M$$

Now, cryptography is widely used in computer security, hence it embraces many branches. The theme of computer security differs from the theme of computer safety though both are closely related. Computer safety is concerned with guarding against accidental damages while computer security is concerned with guarding against intentional damages. Thus, cryptography is an important part in computer security and safety but it does not solely guarantee it. Figure 2.1 below depicts cryptography as a branch in computer security. [7]

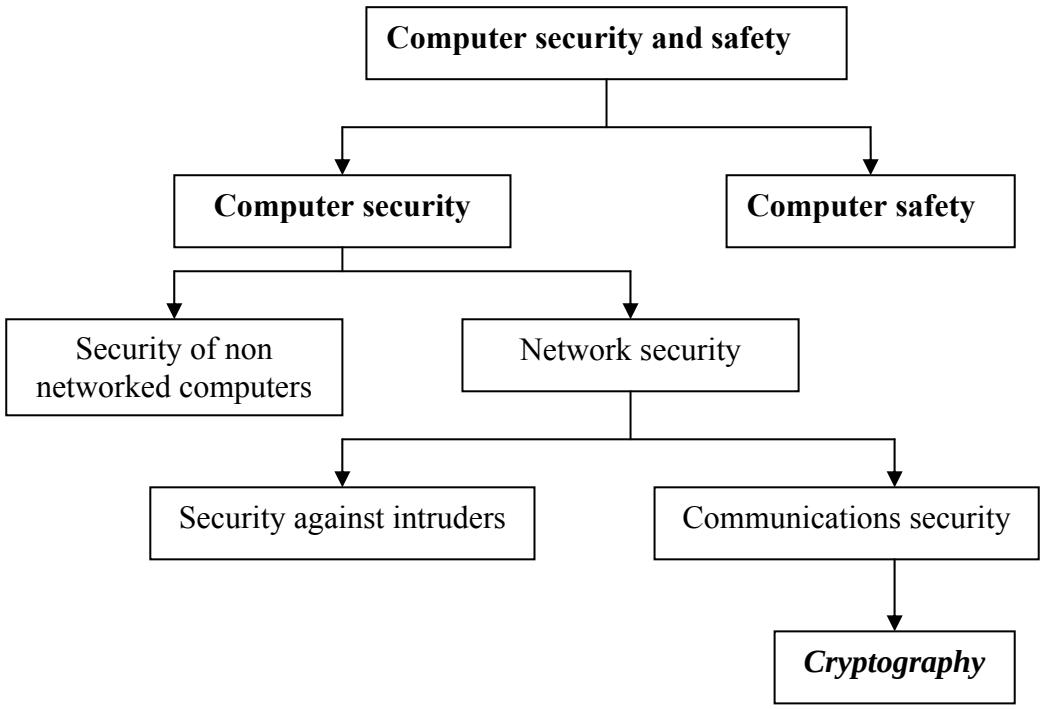

Figure 2.1: Cryptography in Computer Security and Safety

2.4 Cryptography goals

The fundamental goal of cryptography is to prevent and detect, cheating and other malicious activities. This goal is achieved by adequately addressing four frameworks in both theory and practice. These four frameworks that are commonly applied in network services are described as follows:

Confidentiality:

It is a service used to keep the content of information from all but those authorized to see/understand it. Secrecy is a term synonymous with confidentiality and privacy. There are numerous approaches to provide confidentiality, ranging from physical protection to mathematical algorithms which render data unintelligible.

Data integrity:

It is a service which addresses the unauthorized alteration of data. To assure data integrity, one must have the ability to detect data manipulation by unauthorized parties. Data manipulation includes such actions as insertion, deletion, substitution and multiplication.

Authentication:

It is a service related to identification. This service applies to both the sender and the receiver entity. To clarify, two parties initiated into a secure communication should first identify each other.

Non-repudiation:

It is a service which prevents an entity from denying previous commitments or actions. This service is desired in situations where, for example, one entity can authorize the purchase of property to another entity and later denies such authorization was granted. In practice the involvement of a trusted third party is necessary to resolve such disputes. The mathematical techniques, as the definition states, are known as cryptographic algorithms. These cryptography algorithms are the fundamental building blocks for the four frameworks that are described above. Each type of cryptographic algorithm can be classified according to their characteristic features. The next section will describes these classifications and describes the ciphering principles of each classification. [8]

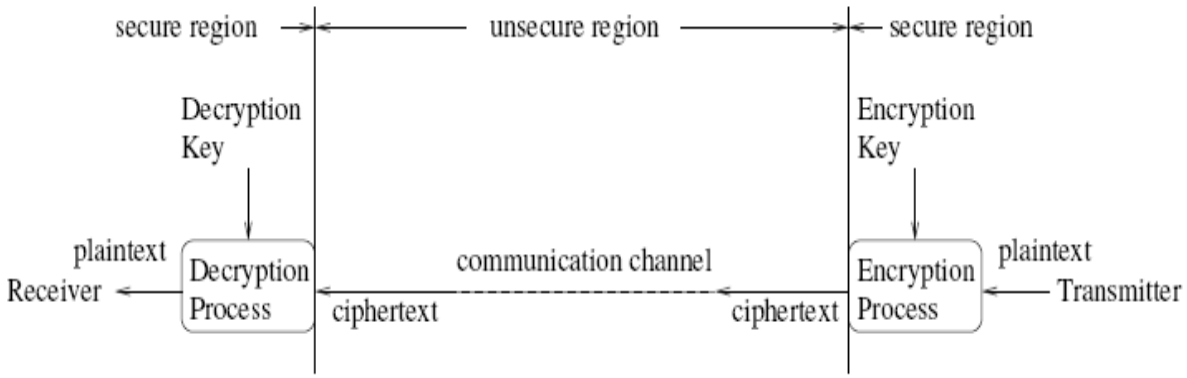

Figure 2.2: Encryption and decryption process

2.5 Ciphering Techniques

There are three groups of cryptographic algorithms which are: symmetric cryptographic algorithms, public-key algorithms and hash functions. Algorithms of the first two classifications are key based techniques in which plain-text is transformed into cipher-text or vice versa. Plain-text is a state of data in which information is easily accessible. While, cipher-text is a state of data in which information is hard to reveal. The process of transforming plain-text into cipher-text is called encryption, while the process of transforming cipher-text into plain-text is called decryption. Note that the cipher-text can be

transformed back into the plain-text only by using a valid key. The algorithms of the last classification, hash functions, are based on mapping data of arbitrary length to a certain value.

The characteristic feature of symmetric cryptographic algorithms, or also known as ciphers, is that both the encryption and decryption processes are accomplished by using the same key.

The characteristic feature of public-key algorithms is that the encryption and decryption processes are accomplished by using different keys. [9]

2.5.1 Symmetric Key Cryptography (called also secret key cryptography)

The first notion of symmetric key cryptography dates from thousands of years ago. Julius Caesar encrypted his secret documents by replacing each character by the character that is located three positions further in the alphabet. Although it is obvious that this encryption technique is not free of flaws, it can be used as an example to explain the basics of symmetric key cryptography. In Caesar's scheme, encrypting a message means shifting each character over a certain number of positions in the alphabet. The decryption operation shifts each character over the same number of positions back in the alphabet. The secret key in this scheme is the number of positions over which the characters are shifted. In symmetric key cryptography, we require that the encryption and decryption keys are equal or can be derived easily from each other. This is illustrated in Fig 2.3, where Alice encrypts a plaintext m using an encryption function E and a key k, resulting in a ciphertext $c = E_k(m)$. Bob uses the same key for decrypting the ciphertext in order to recover Alice's original message $m = D_k(c)$. An eavesdropper, called Eve as shown in Fig 2.3, cannot recover the plaintext from the ciphertext without knowing the secret key k. She is allowed, however, to have full knowledge of the encryption and decryption schemes E and D. This is known as Kerckhoffs' principle: "A cryptosystem should be secure, even if an adversary knows everything about the system, except for the key" [10].

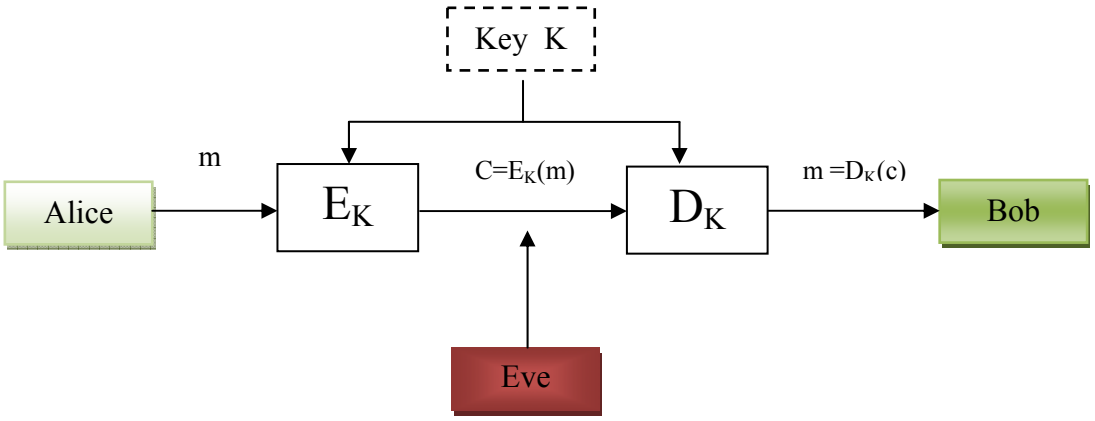

Fig 2.3: *General model for private key encryption.*

Whereas Caesar's cipher can easily be broken by a brute-force attack, i.e., trying all possible keys until a meaningful message is produced, some more secure and practical encryption schemes have been developed over the past decades. These schemes can be divided into block ciphers and stream ciphers. Whereas block ciphers operate on a "block" of data, stream ciphers evaluate one bit or one byte at a time. Stream ciphers also have an internal state, which is stored in a piece of memory [11].

The difference between block ciphers and stream ciphers is shown in Fig 2.4

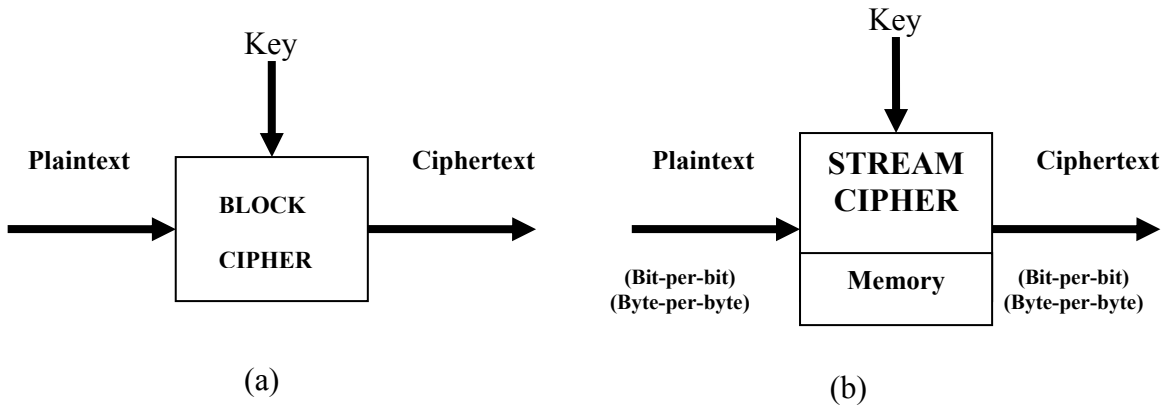

Figure 2.4: General model of a block cipher and a stream cipher

Block Ciphers:

Two important principles in the construction of block ciphers are substitution and transposition. Substitution replaces characters or symbols by other characters or symbols, while transposition permutes the characters or symbols in a block of data. Substitution induces confusion in a cipher, i.e., it makes it hard to find a relationship between the key and the ciphertext on the one hand and the key and the plaintext on the other hand. Transposition causes diffusion, which makes sure there is no local relationship between the statistics of the symbols in the plaintext and the ciphertext.

The two most widely used block ciphers are DES and AES. The Data Encryption Standard (DES) was standardized in 1977. However, because of its 56-bit key, DES is considered to be insecure for practical applications. A 3-times cascaded version of DES, called Triple DES or TDEA, is believed to be practically secure. Because the block length and the performance of Triple DES did not fulfil the requirements of future applications, an open competition for a new block cipher standard was launched by the National Institute of Standards and Technology (NIST). As a result, the Advanced Encryption Standard (AES) was announced in 2001. The AES cipher exists with a 128-, 192- and 256-bit key length. In order for block ciphers to handle plaintexts that contain more bits than the block width of operation can be implemented. The most straightforward mode is the Electronic CodeBook (ECB) mode, in which the plaintext is divided into parts, several modes of which the number of bits is equal to the block width. Each block is fed through the block cipher using the same key. This mode of operation has several security flaws, because it does not hide data patterns.

Stream Ciphers:

Stream ciphers are used for applications where small area and/or high speed are important requirements. Examples of standardized stream ciphers are RC4, designed by Ron Rivest in 1987, which provide security for the Internet and wireless networks, GSM communication and the Bluetooth protocol, respectively. However, most standardized stream ciphers have been proven to be insecure over the past years. [10]

Key management through symmetric-key technique

One solution which employs symmetric-key techniques involves an entity in the network which is trusted by all other entities. this entity is referred to as a *trusted third party* (TTP). Each entity A_i shares a distinct symmetric key k_i with the TTP. These keys are assumed to have been distributed over a secured channel. If two entities subsequently wish to

communicate, the TTP generates a key k (sometimes called a *session key*) and sends it encrypted under each of the fixed keys as depicted in Figure 2.5 for entities A1 and A5. [11]

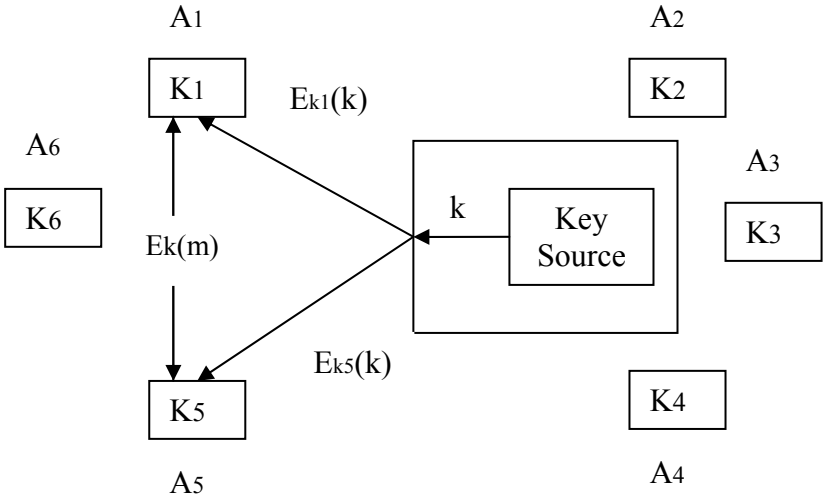

Figure 2.5: Key management using a trusted third party (TTP).

Advantages of this approach include:

1. It is easy to add and remove entities from the network.
2. Each entity needs to store only one long-term secret key.

Disadvantages include:

1. All communications require initial interaction with the TTP.
2. The TTP must store n long-term secret keys.
3. The TTP has the ability to read all messages.
4. If the TTP is compromised, all communications are insecure.

2.5.2 Asymmetric Cryptography (Public Key Cryptography)

Some things you can tell the public, but some things you just want to keep private. In symmetric key cryptography, a single secret key is used between entities, whereas in public key systems, each entity has different keys, or ***asymmetric keys***. The two different asymmetric keys are mathematically related. If a message is encrypted by one key, the other key is required to decrypt the message. In a public key system, the pair of keys is made up of one public key and one private key. The ***public key*** can be known to everyone, and the ***private key*** must only be known to the owner. Many times, public keys are listed in directories and databases of e-mail addresses so they are available to anyone who wants to

use these keys to encrypt or decrypt data when communicating with a particular person. Figure 2.6 illustrates an asymmetric cryptosystem.

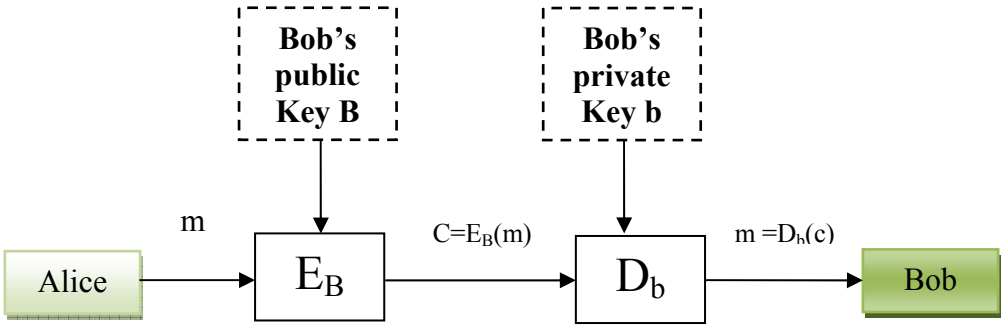

Fig 2.6: General model for public key encryption.

The public and private keys are mathematically related, but cannot be derived from each other. This means that if an evildoer gets a copy of Bob's public key, it does not mean he can now use some mathematical magic and find out Bob's private key. If Bob encrypts a message with his private key, the receiver must have a copy of Bob's public key to decrypt it. The receiver can decrypt Bob's message and decide to reply back to Bob in an encrypted form. All she needs to do is encrypt her reply with Bob's public key, and then Bob can decrypt the message with his private key. It is not possible to encrypt and decrypt using the exact same key when using an asymmetric key encryption technology.

Bob can encrypt a message with his private key and the receiver can then decrypt it with Bob's public key. By decrypting the message with Bob's public key, the receiver can be sure that the message really came from Bob. A message can only be decrypted with a public key if the message was encrypted with the corresponding private key. This provides authentication, because Bob is the only one who is supposed to have his private key. When the receiver wants to make sure Bob is the only one that can read her reply, she will encrypt the response with his public key. Only Bob will be able to decrypt the message because he is the only one who has the necessary private key.

Now the receiver can also encrypt her response with her private key instead of using Bob's public key. Why would she do that? She wants Bob to know that the message came from her and no one else. If she encrypted the response with Bob's public key, it does not provide authenticity because anyone can get a hold of Bob's public key. If she uses her private key to encrypt the message, then Bob can be sure that the message came from her and no one else. Symmetric keys do not provide authenticity because the same key is used on both ends.

Using one of the secret keys does not ensure that the message originated from a specific entity.

If confidentiality is the most important security service to a sender, she would encrypt the file with the receiver's public key. This is called a ***secure message format*** because it can only be decrypted by the person who has the corresponding private key. If authentication is the most important security service to the sender, then she would encrypt the message with her private key. This provides assurance to the receiver that the only person who could have encrypted the message is the individual who has possession of that private key. If the sender encrypted the message with the receiver's public key, authentication is not provided because this public key is available to anyone. Encrypting a message with the sender's private key is called an ***open message format*** because anyone with a copy of the corresponding public key can decrypt the message; thus, confidentiality is not ensured.

For a message to be in a ***secure and signed format***, the sender would encrypt the message with her private key and then encrypt it again with the receiver's public key. The receiver would then need to decrypt the message with his own private key and then decrypt it again with the sender's public key. This provides confidentiality and authentication for that delivered message. The different encryption methods are shown in Figure 2.7.

Each key type can be used to encrypt and decrypt, so do not get confused and think the public key is only for encryption and the private key is only for decryption. They both have the capability to encrypt and decrypt data. However, if data is encrypted with a private key, it cannot be decrypted with a private key. If data is encrypted with a private key, it must be decrypted with the corresponding public key. If data is encrypted with a public key, it must be decrypted with the corresponding private key. Figure 2.8 further explains the steps of a signed and secure message. [9]

An asymmetric cryptosystem works much slower than symmetric systems, but can provide confidentiality, authentication, and nonrepudiation depending on its configuration and use. Asymmetric systems also provide for easier and more manageable key distribution than symmetric systems and do not have the scalability issues of symmetric systems. The "Public Key Cryptography" section will show how these two systems can be used together to get the best of both worlds.

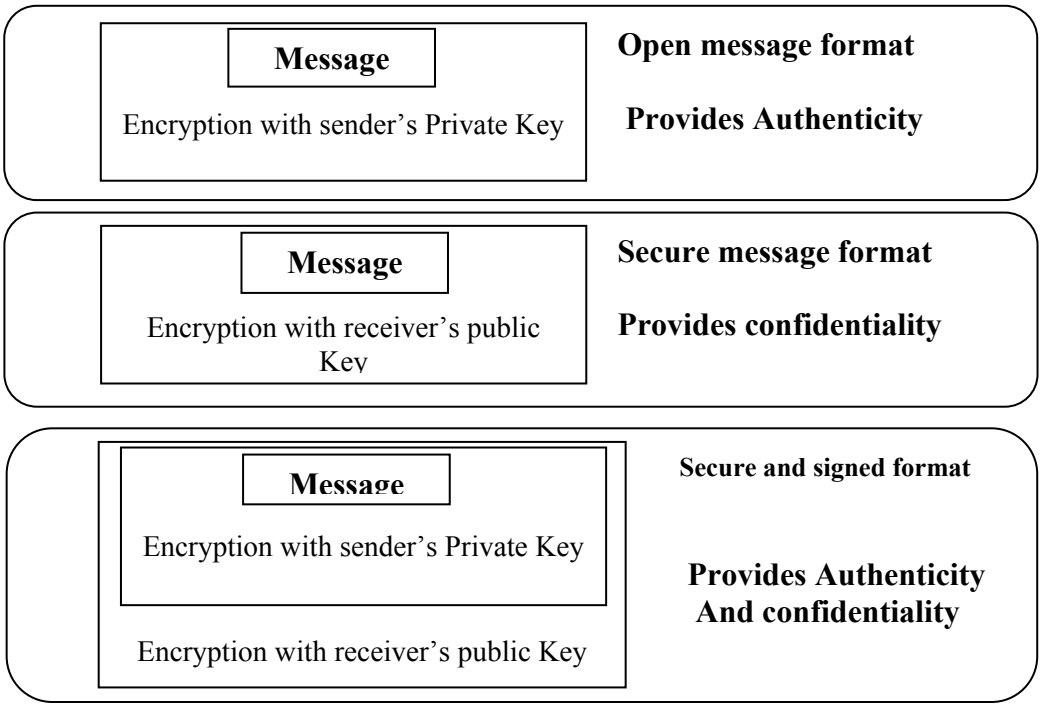

Fig 2.7 the way that the sender encrypts the message dictates the type of security service that will be provided

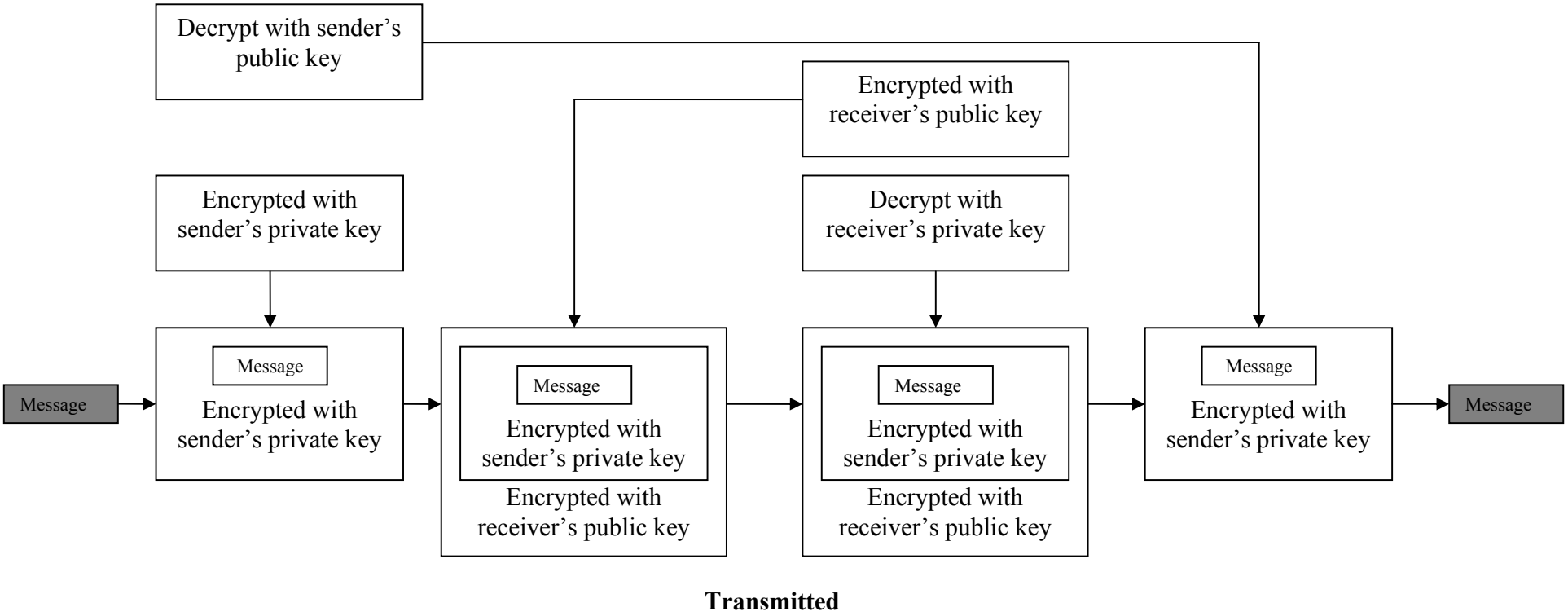

Figure 2.8: A secured and signed message is encrypted twice with the sender and receiver's keys

The following outlines the strengths and weaknesses of asymmetric key systems:

Strengths

- Better key distribution than symmetric systems
- Better scalability than symmetric systems
- Can provide confidentiality, authentication, and nonrepudiation

Weaknesses

- Works much slower than symmetric systems

The following are examples of asymmetric key algorithms:

- RSA
- Elliptic Curve Cryptosystem (ECC)
- Diffie-Hellman
- El Gamal
- Digital Signature Standard (DSS)

Tab 2.1: Different characteristics between symmetric and asymmetric systems

Attributes	Symmetric	Asymmetric
Keys	One key is shared between two Or more entities	One entity has the public key and the other entity has a private key
Key exchange	Out-of-band	Symmetric key is encrypted and sent with message; thus the key is distributed by inbound means.
speed	Algorithm is less complex and faster	Algorithm is more complex and slower
Key length	Fixed-key length	variable-key length
use	Bulk encryption, which means encrypting files and communication paths.	Key encryption and distributing keys
Security service provided	Confidentiality and integrity	Confidentiality, integrity, authentication and nonrepudiation

2.5.3 Hybrid Encryption Methods

Comparisons were made between symmetric and asymmetric algorithms earlier. So because symmetric has some downfalls, surely asymmetric will be our saving grace, right? Not so fast.

Asymmetric does provide more security services than the symmetric methods. It provides confidentiality by encryption. It also provides nonrepudiation when a sender encrypts a message using his private key, it provides integrity because if the message was tampered with it could not be properly decrypted, and it provides access control because only the people with the private key and corresponding public key can access the encoded data. However, asymmetric algorithms are unacceptably slow. Their algorithms are so complex and intensive that they require more system resources and take too long to encrypt and decrypt messages. We just can't seem to win. So we turn to a hybrid system that uses symmetric and asymmetric encryption methods together and call it hybrid key cryptography. [12]

2.6 Hash functions

One of the fundamental primitives in modern cryptography is the cryptographic hash function, often informally called a one-way hash function.

A *hash function* is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called *hash-values*. For a hash function which outputs n-bit hash-values (e.g., n = 128 or 160) and has desirable properties, the probability that a randomly chosen string gets mapped to a particular n-bit hash-value (image) is 2^{-n}. The basic idea is that a hash-value serves as a compact representative of an input string. To be of cryptographic use, a hash function h is typically chosen such that it is computationally infeasible to find two distinct inputs which hash to a common value (i.e., two *colliding* inputs x and y such that h(x) = h(y)), and that given a specific hash-value y, it is computationally infeasible to find an input (pre-image) x such that h(x) = y. [12]

The Different Hashing Algorithms Available are:

- Message Digest 2 (MD2) algorithm
- Message Digest 4 (MD4) algorithm
- Message Digest 5 (MD5) algorithm
- SHA Secured Hash algorithm and its Updated version SHA-1

2.7 Digital signatures

A digital signature is the way to check data integrity with asymmetric cryptography. It works in a similar way as *message authentication codes*, MAC. The signing process uses a hash function producing a fixed sized hash-digest. The signing function encrypts the hash-digest using the private key into a signature and the signature can be verified by anyone with the sender's public key. When *Alice* signs a message for *Bob*, she uses an appropriate hash function to generate the hash-digest, $h_d = HASH(m)$. She then signs h_d using her private key, S_{Alice}, giving

$s = S(S_{Alice}, h_d)$ and transmits the signature, s, together with the message.

Bob can then verify the signature by computing his own hash-digest of the received message, $h_{d,comp} = HASH(m_{rcv})$. The result should be the same as returned from the signature verification algorithm, $h_{d,rcv} = V(P_{Alice}, s_{rcv})$. If $h_{d,comp}$ equals $h_{d,rcv}$ the message integrity is verified, otherwise it has been tampered with. Where S and V are the asymmetric encryption, decryption used functions.

Since the message is signed with a private key and a message is encrypted with a public key the same key-pair should never be used for both applications. Each user has to have at least two separate key-pairs, one for encrypting messages and one for signing them. [12]

2.8 Cryptanalysis

Cryptanalysis is the study of retrieving the plain-text without knowledge of the valid key.

A cipher is said to be breakable if a third party, without prior knowledge of the key, can systematically recover plain-text from the corresponding cipher-text. This all, within a time frame shorter than using the exhaustive search method. With the exhaustive search method, also known as brute force attack, all possible keys are tried in order to reveal the plain-text.

There are three types of cryptanalysis techniques, these are: linear, differential and side-channel techniques. Linear cryptanalysis takes advantage of eventual input-output correlations over a few rounds of the cipher. This technique uses a linear approximation to describe the behavior of the block cipher. Given sufficient pairs of known plain-text and corresponding cipher-text, bits of information about the key can be obtained and increased amounts of data will usually give a higher probability of success. Differential cryptanalysis, developed by Eli Biham and Adi Shamir, is a type of cryptanalytic technique that appears to

be most effective on block ciphers. This technique is based on the evolution of the differences made in two related plain-texts encrypted with the same key.

The side-channel cryptanalysis techniques are based on timing, fault and power analysis of systems. For example, the power consumption of the electrical components is logged to deduce secret information like the encrypting key. In practice, sometimes devices are tampered in order to have it perform some erroneous operations. All these techniques are used within the framework of revealing the secret key or the secret information. [13]

2.9 Security of Cryptosystems

Whereas the science of cryptography aims at the construction of new ciphers, cryptanalysis is the study of techniques to break these ciphers. These two research areas stimulate each other by surpassing each other step by step: once a new cipher is designed, cryptanalysts try to break it; once it is broken, cryptographers try to redesign it in order to overcome the flaws; etc.

Classical cryptanalysis focuses on weaknesses in the algorithm. The most straightforward weakness is a badly chosen key length. If the size of the key space is too small, the cipher can be broken by a brute-force attack. The two most frequently studied cryptanalytic techniques for symmetric key cryptography are linear cryptanalysis, which tries to find a linear approximation of the behavior of an algorithm, and differential cryptanalysis, which exploits the relationship between differences in the input and subsequent differences in the output of a cipher. For the cryptanalysis of public key cryptography, there exist several algorithms based on number theory. More recently, a new class of cryptanalytic attacks has been introduced, called implementation attacks. In this case, the attacker does not focus on flaws in the algorithm, but tries to break the system by exploiting weaknesses in the implementation of the algorithm. Implementation attacks can be performed in an invasive or a non-invasive way. In the former case, the attacker has unlimited access to the cryptographic device. In the latter case, the attacker retrieves information without interfering with the normal functioning of the device.

Important classes of attacks that can be categorized as non-invasive are side channel attacks. Side channel attacks impose a new model on cryptosystems. An attacker is no longer limited to using plaintext and/or ciphertext information. [13]

Side channels such as power consumption, timing information, electromagnetic emanation, etc. can be used to extract sensitive information. This is illustrated in Fig 2.9.

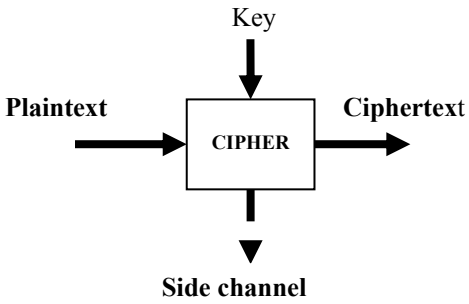

Figure 2.9: general model of a cryptosystem in a presence of a side channel

The first official information on side channel attacks dates from 1956. Peter Wright describes how he helped the British secret services to break a rotor machine by listening to the clicking sound with a microphone. In the mid 1980s there was a lot of commotion about the electromagnetic emanation of video screens. In 1996, Paul Kocher described how timing information can be exploited as a side channel. He also introduced the first attacks based on the power consumption of a cryptosystem. In 2001, the first results on the analysis of the electromagnetic radiation of modern cryptographic devices were reported. However, measurements of electromagnetic fields have been performed since the 1950s for military purposes. This research has led to a never published set of standards for reducing the electromagnetic radiation of electronic devices. TEMPEST is the codeword that the American government used for these standards. [9]

There are two main flows in recent research on side channel attacks. On the one hand, advanced analysis and processing techniques are developed to enhance side channel analysis attacks and in particular power analysis attacks. On the other hand, new countermeasures are implemented at all levels of design abstraction. Here, the trade-off between performance and side channel resistance is the key issue. The levels of design abstraction are depicted in Fig 2.10. Practical examples show that the lower the level on which the countermeasure is implemented, the more effective it is. However, the degradation in area and speed also increases when we descend in the levels of design abstraction.

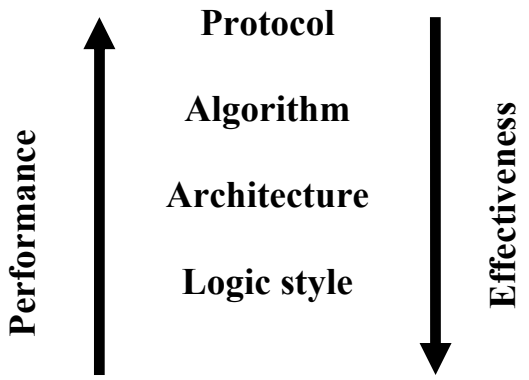

Figure 2.10: Behaviour of effectiveness of a countermeasure and the performance of the system for the adoption of countermeasures at different levels of the design.

Security in embedded systems requires the choice of a suitable implementation platform. For some systems a general purpose microprocessor suffices the requirements, but when high performance is the main criteria, cryptographic coprocessors in hardware are indispensable. Examples of high performance applications are ATMs and trusted computing platforms. When very high performance is required or when a high volume of coprocessors is needed, ASICs are chosen as implementation platforms. In this case, the reconfigurability of FPGAs is only used for prototyping. However, thanks to the efforts of FPGA manufacturing companies, the performance gap between ASICs and FPGAs becomes smaller and smaller. The progress in the performance and capabilities of FPGAs is demonstrated, which lines up the evolution of available resources in Xilinx products. it can be derived that FPGAs have become heterogeneous systems with a variety of dedicated resources. This explains the trend that FPGAs are more and more used in end products such as routers and banking applications. Following this trend, the need for specific FPGA architectures can be justified. [9]

2.10 Conclusion

Cryptology research is a very large domain. This chapter provided the background information for this thesis. It was described that cryptographic algorithms can be classified in three groups: symmetric algorithms, public-key algorithms and hash functions.

After showing the classes of encryption algorithms we found that public key ciphering techniques are the most important ones due to their security services.

As explained, symmetric algorithms are based on one key, public-key algorithms are based on several keys and hash functions maps data of an arbitrary length to a certain value.

In the next chapter we will do a fast overview of arithmetic operations over finite field, as an initiation to another study which concerns important operations used in principal PKC algorithms as the RSA and the elliptic curve cryptography.

Many cryptographic algorithms are based on specialized arithmetic computations such as finite field arithmetic. For clients that only perform cryptographic computations occasionally, the central processing unit (CPU) in a PC is sufficient. However the work load on a server that will handle thousands of requests per second may be unacceptably large. In addition, clients which have very limited computing resources, such as smartcards, mobile phones and handheld computers may not have sufficient computing power. Special hardware cryptosystems can offer higher performance than conventional CPUs. In addition, cryptosystems implemented in software may have lower security than tamper proof hardware devices

3.1 Introduction

This chapter will be specified more and more for some well known public key cryptosystems ,as the RSA (Rivest, Shamir, and Adleman),DH(Diffie-Hellman Key Agreement) and the elliptic curve algorithm(ECC), We choose public key because in this part of cryptography we deal with arithmetic operation these are very computationally intensive, operating on very large integers, and because our goal is to implement IP cores dedicated to cryptography ,we have to know the computationally operations that need really hardware implementation

3.2 The RSA algorithm

The RSA algorithm was inverted by Rivest, Shamir, and Adleman in 1977 and published in 1978 . It is one of the simplest and most widely used public-key cryptosystems. Figure 1 summarizes the RSA algorithm. [14]

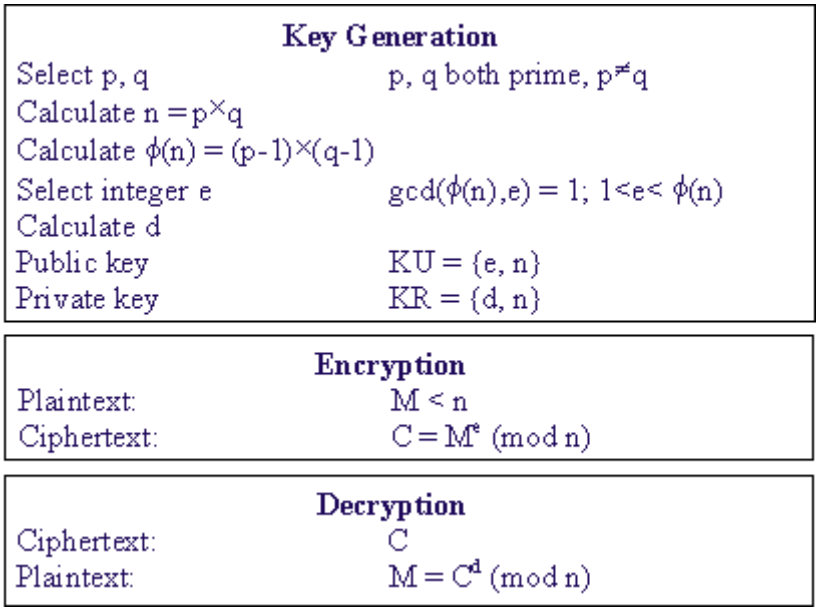

Fig 3.1 The RSA algorithm

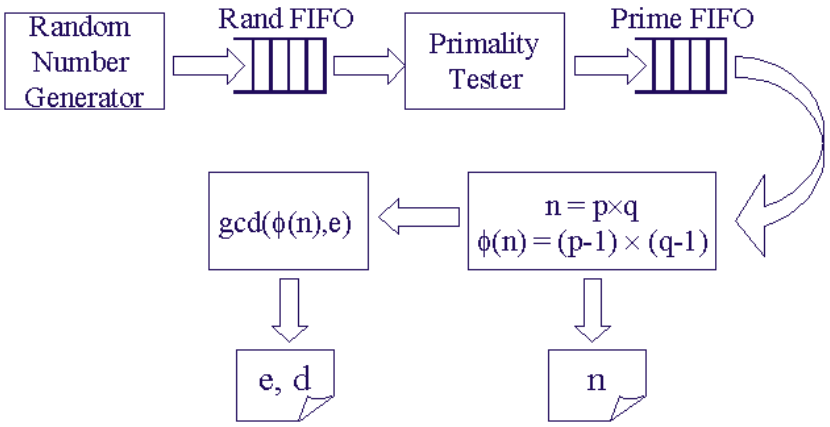

Fig 3.2 The system architecture for RSA key generation

The system architecture for key generation is shown in Figure 2. A random number generator generates pseudo random numbers and stores them in the rand FIFO. Once the FIFO is full, the random number generator stops working until a random number is pulled out by the primality tester. The primality tester takes a random number as input and tests if it is a prime. Confirmed primes are put in the prime FIFO. Similarly to the random number generator, primality tester starts new test only when prime FIFO is not full. A lot of power is saved by using the two FIFOs because computation is performed only when needed. When new key pair is required, the down stream component pulls out two primes from the prime FIFO, and calculates n and φ(n). N is stored in a register. φ(n) is sent to the Greatest Common Divider (GCD), where public exponent e is selected such that gcd(φ(n), e) = 1, and private exponent d is obtained by inverting e modulo φ(n). E and d are also stored in registers.

Once n, d, and e are generated, RSA encryption/decryption is simply a modular exponentiation operation. Figure 3 shows the RSA encryption/decryption structure in hardware implementation.

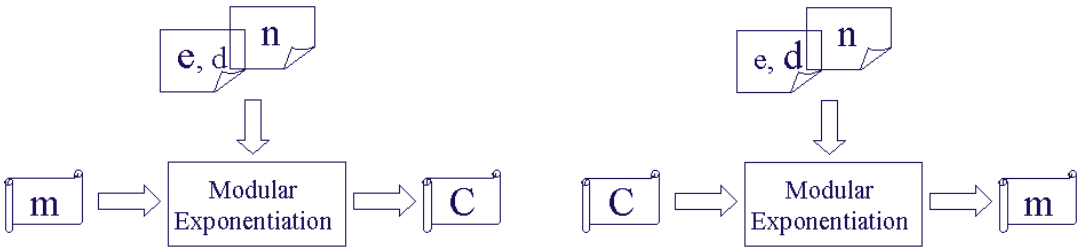

Fig 3.3 The RSA encryption/decryption structure

The core of the RSA implementation is how efficient the modular arithmetic operations are, which include modular addition, modular subtraction, modular multiplication and modular exponentiation. The RSA also involves some regular arithmetic operations, such as regular

addition, subtraction and multiplication used to calculate n and φ(n), and regular division used in GCD operation. [14]

3.3 DH - Diffie-Hellman Key Agreement

Diffie-Hellman is a key agreement algorithm that helps two devices to agree on a shared secret between them with out the need to exchange any secret/private information. An overview of the algorithm is given below. [15]

3.3.1 Key agreement Algorithm

For establishing shared secret between two device A and B, both device agrees on public constants *p* and *g*. Where p is a prime number and *g* is the generator less than *p*.

- D1. Let *a* and *b* be the private keys of the devices A and B respectively, Private keys are random number less than p.
- D2. Let $g^a mod\ p$ and $g^b mod\ p$ be the public keys of devices A and B respectively
- D3. A and B exchanged their public keys.
- D4. The end A computes $(g^b mod\ p)^a\ mod\ p$ that is equal to $g^{ba} mod\ p$.
- D5. The end B computes $(g^a mod\ p)^b\ mod\ p$ that is equal to $g^{ab} mod\ p$.
- D6. Since $K = g^{ba} mod\ p = g^{ab} mod\ p$, shared secret = *K*.

3.3.2 DH - Mathematical Explanation

From the properties of modular arithmetic

$a\ mod\ n * b\ mod\ n \equiv a * b\ (mod\ n)$

Which can be written as

$(a_1\ mod\ n)*(a_2\ mod\ n)*\ldots *(a_k\ mod\ n) \equiv a_1 * a_2 * \ldots * a_k\ (mod\ n),$

If $a_i=a$, where $i = 1, 2, 3 \ldots k$

$(a\ mod\ n)^k \equiv a^k mod\ n$

$(g^a mod\ p)^b mod\ p = g^{ab} mod\ p$ and

$(g^b mod\ p)^a mod\ p = g^{ba} mod\ p$

For all integers $g^{ab}=g^{ba}$,

Therefore shared secret $K=g^{ab} mod\ p=g^{ba} mod\ p$

Since it is practically impossible to find the private key *a* or *b* from the public key $g^a mod\ p$ or $g^b mod\ p$, it is not possible to obtain the shared secret K for a third party.

3.4 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is relatively new technology compared to other public key cryptography such as RSA. Elliptic key operates on smaller key size. A 160-bit key in ECC is considered to be as secured as a 1024 bit key in RSA. ECC operates on the points in the elliptic curve $\mathbf{y^2=x^3+ax+b}$, where $\mathbf{4a^3+27b^2\neq 0}$.

The above equation of elliptic curve is in real coordinate. To make elliptic curve operation efficient and accurate the elliptic curve can be defined in finite fields. Elliptic curve in two finite fields, prime field and binary field, are defined by standard. In prime field operation the elliptic curve equation is modified as $\mathbf{y^2 mod\ p=x^3+ax+b\ mod\ p}$, where $\mathbf{4a^3+27b^2\ mod\ p\neq 0}$.

There are certain public constants that are shared between parties involved in secured and trusted ECC communication. This includes curve parameter **a**, **b**, a generator point **G** in the chosen curve, the modulus **p**, order of the curve **n** and the cofactor **h**. [16]

3.5 Conclusion

After this short review on the most important public key cryptography algorithms, we noticed that the core operation in most of them is modular arithmetic (addition, multiplication and exponentiation), for example the core operation in RSA is the modular exponentiation, and because this one can be calculated by successive modular multiplication operations ,we decide to specify our studies more and more on this operation, the next chapter contains the newest and the most popular algorithms dedicated to modular product computation

4.1 Introduction

The rising growth of data communication and electronic transactions over the internet has made security to become the most important issue over the network. To provide modern security features, public-key cryptosystems are used. The widely used algorithms for public-key cryptosystems are RSA, Diffie-Hellman key agreement (DH), the digital signature algorithm (DSA) and systems based on elliptic curve cryptography (ECC). All these algorithms have one thing in common: they operate on very huge numbers (e.g. 160 to 2048 bits). Long word lengths are necessary to provide a sufficient amount of security, but also account for the computational cost of these algorithms.

In the recent years, we have witnessed to an increasing deployment of hardware devices for providing security functions, such as confidentiality, authentication, integrity and non-repudiation. Among the existing techniques the Rivest-Shamir-Adleman (RSA) algorithm is by far the most widely adopted public-key cryptography algorithm. The RSA algorithm has a number of applications, such as encryption and digital signature. The basic operation of this algorithm is modular exponentiation on large integers, i.e. $\boldsymbol{Y = XE \bmod N}$, which is used for decryption/ signature and encryption/verification. The security level of an RSA cryptosystem is tied to the length of the modulus $\boldsymbol{N}$. A modulus of at least 768 bits is recommended, but one had best use 1024-bit moduli at least for long-term security. All operands involved in the computation of modular exponentiation have normally the same size as the modulus. All existing techniques for computing $\boldsymbol{XE \bmod N}$ reduce modular exponentiation to a sequence of modular multiplications.

Several sub-optimal algorithms have been presented in the literature to compute the sequence of multiplications leading to the $\boldsymbol{E}$th power of X, such as binary methods (RL algorithm and LR-algorithm), M-ary methods, and more. We adopted the method known as Binary Right-to-Left Algorithm which consists of repeated squaring and multiplication operations. This choice was motivated by its simple and efficient hardware implementation. Since modular multiplication is the core computation of all modular exponentiation algorithms, the efficiency of its execution is crucial for any implementation of the RSA or other public key algorithms. Unfortunately, modular multiplication is a complex arithmetic operation because of the inherent cost of multiplication and modulo operations. Several techniques have been proposed in the last years for achieving efficient implementations of modular multiplication. In particular, Blakley's method (or interleaved) and Montgomery's method are the most studied ones. Indeed, they are the only algorithms suitable for practical

hardware implementation. Both Blakley's method and Montgomery's method perform the modular reduction during the multiplication process. No division operation is needed at any point in the process. However, Blakley's method needs a comparison between two large integers at each step of the modular multiplication process, while the Montgomery's method does not. This is achieved by resorting to a representation of the operands as a residue class modulo ***N***. Furthermore, the Montgomery's technique requires some preprocessing and post processing steps, which are needed to convert the numbers to and from the residue based representation. However, the cost of these steps is negligible when many consecutive modular multiplications are to be executed, as in the case of RSA. This is the reason why the Montgomery's method is considered the most efficient algorithm for implementing RSA operations. There exist several versions of the Montgomery's algorithm, depending on the number *r* used as the radix for the representation of numbers. In hardware implementations *r* is always a power of 2.

4.2 Modular Exponentiation Operation

The modular exponentiation operation is simply an exponentiation operation where multiplication and squaring operations are modular operations. The exponentiation heuristics developed for computing M^e are applicable for computing M^e (mod n). it exists several techniques for the exponentiation operation and several algorithms and architectures for hardware implementation . [17]

The binary method for computing M^e (mod n) given the integers M , e ,and n has two variation depending on the direction by which the bits of e are scanned :
Left-to-Right (LR) and Right-to-Left (RL). The LR binary method is more widely known :

4.2.1 LR Binary Method

Input : M, e, n
Output : $C := M^e \bmod n$
1. *if* $e_{h-1} = 1$ *then* $C := M$ *else* $C := 1$
2. *For* $i = h - 2$ *downto* 0
2.*a* $C := C \cdot C \pmod{n}$
2*b* *if* $e_i = 1$ *then* $C := C \cdot M \pmod{n}$
3. *return* C

The bits of e are scanned from the most significant to the least significant, and a modular squaring is performed for each bit. A modular multiplication operation is performed only if the bit is 1. An example of LR binary method is illustrated below for h=6 and e= 55 = (110111). Since $e_5 = 1$, the LR algorithm starts with C: = M, and proceeds as

i	e_i	*Step 2a (C)*	*Step 2b (C)*
4	*1*	$(M)^2 = M^2$	$M^2 * M = M^3$
3	*0*	$(M^3)^2 = M^6$	M^6
2	*1*	$(M^6)^2 = M^{12}$	$M^{12} * M = M^{13}$
1	*1*	$M^{13})^2 = M^{26}$	$M^{26} * M = M^{27}$
0	*1*	$(M^{27})^2 = M^{54}$	$M^{54} * M = M^{55}$

Table 4.1: example of LR binary method

The RL binary algorithm, on the other hand, scans the bits of e from the least significant to the most significant, and uses an auxiliary variable P to keep the powers M.

4.2.2 RL Binary Method

Input: M, e, n
Output: $C := M^e \bmod n$
1. $C := 1$; $P := M$
2. *for* $i = 0$ *to* $h - 2$
2a *if* $e_i = 1$ *then* $C := C \cdot P \pmod{n}$
2b $P := P \cdot P \pmod{n}$
3. *if* $e_{h-1} = 1$ *then* $C := C \cdot P \pmod{n}$
4. *return* C

The RL algorithm starts with C: = 1 and P: = M, proceeds to compute M55 as follows:

i	e_i	Step 2a (C)	Step 2b (P)
0	1	$1 * M = M$	$(M)^2 = M^2$
1	1	$M * M^2 = M^3$	$((M)^2)^2 = M^4$
2	1	$M^3 * M^4 = M^7$	$(M^4)^2 = M^8$
3	0	M^7	$(M^8)^2 = M^{16}$
4	1	$M^7 * M^{16} = M^{23}$	$(M^{16})^2 = M^{32}$
Step 3 : $e_5 = 1$, thus C:= $M^{23} * M^{32} = M^{55}$			

Table 4.2: example of RL binary method

We compare the LR and the RL algorithm in term of time and space requirements below:

- Both methods require *h-1* squarings and an average of *½ (h-1)* multiplications.
- The LR binary method requires two registers: *M* and *C*
- The RL binary method requires three registers: *M, C* and *P*. However, we note P can used in place of M, if the value of M is mot needed there after.
- The multiplication (step 2a) and squaring (step 2b) operations in the RL binary method are independent of one another, and thus these steps can be parallelized. Provide that we have two multipliers (one multiplier and one squarer) available, the running time of the RL binary method is bounded by the total time required for computing *h-1* squaring operations on k-bit integers. [17]

4.3 Modular multiplication

The modular multiplication is used to perform modular exponentiations, which, in their turn, are used by several public-key cryptosystems. The performance of public-key cryptosystems is primarily determined by the implementation's efficiency of the modular exponentiation. So, consequently, modular multiplication is an important factor in these systems.

The modular multiplication problem is defined as the computation of P = AB (mod n) given the integers A, B, and n. It is usually assumed that A and B are positive integers with $0 \leq A, B < n$, i.e. they are the least positive residues. There are basically four approaches for computing the product P

- Multiply and then divide
- The steps of multiplication and reduction are interleaved
- Brickell's method.
- Montgomery's method.

4.3.1 Multiply and then divide

The multiply and then divide method first multiplies A and B to obtain the $2k$-bit number

$$\boxed{P' := A \cdot B}.$$

Then, the result P' is divided (reduced) by n to obtain the K-bit number

$$\boxed{P := P' \,\%_0\, n}.$$

The Add-and-Shift Algorithm

The add-and-shift algorithm of multiplication is performed using only add and shift operations. The partial product starts in zero and then each multiplier's bit is processed at a time. The multiplicand is added to partial product if that bit is set and, at the ending of this process, the partial product is right-shifted. Here, the reduction step is implemented with successive subtractions until the result is less than the modulus.

The Booth's Algorithm

The Booth's algorithm is also performed with partial products, but it uses several partial product generators together with several adders that operate in parallel. Each partial product obtained is shifted left or right depending on whether the starting bit was the less or the most significant and added up. The number of partial products generated is bound above by the size of the multiplier operand. So, once the sum of the partial products is obtained, the rest of this sum is finally the result of the multiplication. Likewise the previous algorithm, the reduction step will be implemented with successive subtractions.

We will not study the multiply and divide method in details since the interleaving and Montgomery methods more suitable and more efficient for our problem that is in hardware domain. The multiply and then divide method is useful only when one needs the product P'

4.3.2 Montgomery Multiplication

Montgomery multiplication is an efficient method to perform modular multiplication. It was introduced by Montgomery in 1985. The Montgomery algorithm computes $(X \cdot Y \cdot R-1)$ mod M, where X and Y are the operands, M is the modulus and R is a power of two. Whereas computing $(X \cdot Y)$ mod M requires a trial division, Montgomery multiplication only needs a division by a power of two, which comes for free in hardware. Before performing a modular multiplication using the Montgomery algorithm, the operands need to be transformed into Montgomery representation. The Montgomery representation of an integer X, denoted by X_{Mont}, can be computed by performing a Montgomery multiplication on X and R^2, denoted by $Mont_M(X, R^2)$, resulting in $X_{Mont} = MontM(X, R^2) = (X \cdot R^2 \cdot R^{-1})$ mod M $= (X \cdot R)$ mod M. After computing the Montgomery multiplication of two operands in Montgomery representation, the result is also in Montgomery representation and can be converted back by multiplication with R^{-1}, This can be illustrated as follows: [9]

- Conversion of X into Montgomery representation :

$$\begin{aligned} X_{Mont} &= Mont_M \quad (X, R^2) \\ &= (X.\ R^2.\ R^{-1}) \mod M \\ &= (X.\ R) \mod M \end{aligned}$$

- Conversion of Y into Montgomery representation:

$$\begin{aligned} Y_{Mont} &= Mont_M \quad (Y, R^2) \\ &= (Y.\ R^2.\ R^{-1}) \mod M \\ &= (Y.\ R) \mod M \end{aligned}$$

- Computation of the result in Montgomery representation:

$$\begin{aligned} T_{Mont} &= Mont_M \quad (X_{Mont} \quad , Y_{Mont}) \\ &= (X_{Mont} . \ Y_{Mont} . \ R^{-1}) \mod M \\ &= (X \ . \ R . \ Y . \ R . \ R^{-1}) \mod M \\ &= (X . \ Y . \ R) \mod M \end{aligned}$$

- Conversion of T_{Mont} into T:

$$\begin{aligned} T &= Mont_M \quad (T_{Mont} \quad , 1) \\ &= (X \ . \ Y . \ R . \ 1 . \ R^{-1}) \mod M \\ &= (X . \ Y) \mod M \end{aligned}$$

This means that four Montgomery multiplications are needed for one modular multiplication. That is why the use of Montgomery multiplication is only interesting when many consecutive modular multiplications need to be performed. In this case, the Montgomery representation can be maintained for intermediate results. Conversion is only needed before the first and after the last modular multiplication. Note that a single modular multiplication can also be performed with only two Montgomery multiplications, which makes the Montgomery algorithm even more interesting for practical implementations. In this case, only one operand is converted into Montgomery representation and the result does not need to be converted back into normal representation [9]

- Conversion of X into Montgomery representation:

$$\begin{aligned} X_{Mont} &= Mont_M \quad (X, R^2) \\ &= (X . \ R^2 . \ R^{-1}) \mod M \\ &= (X . \ R) \mod M \end{aligned}$$

- Computation of the result:

$$
\begin{aligned}
T = Mont_M \quad & (X_{Mont} \quad , Y) \\
& = (X_{Mont} \,.\, Y \,.\, R^{-1}) \mod M \\
& = (X \,.\, R \,.\, Y \,.\, R^{-1}) \mod M \\
& = (X \,.\, Y) \mod M
\end{aligned}
$$

4.3.3 Montgomery Multiplication Algorithms

The Montgomery algorithm, [Algorithm 1] computes ***P = (X*Y* (2n)-1)*** mod *M*. The idea of Montgomery is to keep the lengths of the intermediate results smaller than *n*+ 1 bit. This is achieved by interleaving the computations and additions of new partial products with divisions by 2; each of them reduces the bitlength of the intermediate result by one.

The key concepts of the Montgomery algorithm are the following:

- Adding a multiple of *M* to the intermediate result does not change the value of the final result; because the result is computed modulo *M*. *M* is an odd number.
- After each addition in the inner loop the least significant bit (LSB) of the intermediate result is inspected. If it is 1, i.e., the intermediate result is odd, we add *M* to make it even. This even number can be divided by 2 without remainder. This division by 2 reduces the intermediate result to *n*+1 bits again.
- After *n* steps these divisions add up to one division by 2^n.

The Montgomery algorithm is very easy to implement since it operates least significant bit first and does not require any comparisons. A modification of Algorithm 1 with carry save adders is given in [Algorithm2] [18]

Algorithm 1: Montgomery multiplication

$Inputs: X, Y, M \quad with \quad 0 \le X, Y < M$
$output: P = (X * Y(2^n)^{-1}) \mod M$
$n: number\ of\ bits\ in\ X;$
$x_i: i^{th} bit\ of\ X;$
$p_0: LSB\ of\ P;$
(1) $P := 0;$
(2) $for \quad (i = 0; i < n; i++)\{$
(3) $P := P + x_i * Y;$
(4) $P := P + p_0 * M;$
(5) $P := P\ div\ 2\ ;\}$
(6) if (P ≥ M) $then\ P := P - M;$

Algorithm 2: Fast Montgomery multiplication

$Inputs: X, Y, M \quad with \quad 0 \le X, Y < M$
$Output: P = (X * Y(2^n)^{-1}) \mod M$
$n: number\ of\ bits\ in\ X;$
$x_i: i^{th}\ bit\ of\ X;$
$s_0: LSB\ of\ S;$
(1) $S := 0; \quad C := 0;$
(2) $for(i = 0\ ; i < n\ ;\ i++)$
(3) $S, C := S + C + x_i * Y;$
(4) $S, C := S + C + s_0 * M;$
(5) $S := S\ div\ 2; \quad C := C\ div\ 2;$
(6) $P := S + C;$
(7) $if\ (P \ge M)\ then\ P := P - M;$

In this algorithm the delay of one pass through the loop is reduced from $O(n)$ to $O(1)$. This remarkable improvement of the propagation delay inside the loop of Algorithm 2 is due to the use of carry save adders to implement step (3) and (4).

Step (3) and (4) in Algorithm 2 represent carry save adders. *S* and *C* denote the sum and carry of the three input operands respectively. Of course, the additions in step (6) and (7) are conventional additions. But since they are performed only once while the additions in the loop are performed *n* times this is subdominant with respect to the time complexity.

Figure 1 shows the architecture for the implementation of the loop of Algorithm 2. The layout comprises of two carry save adders (CSA) and registers for storing the intermediate results of the sum and carry. The carry save adders are the dominant occupiers of area in hardware especially for very large values of *n* (e.g. $n \geq 1024$). [19]

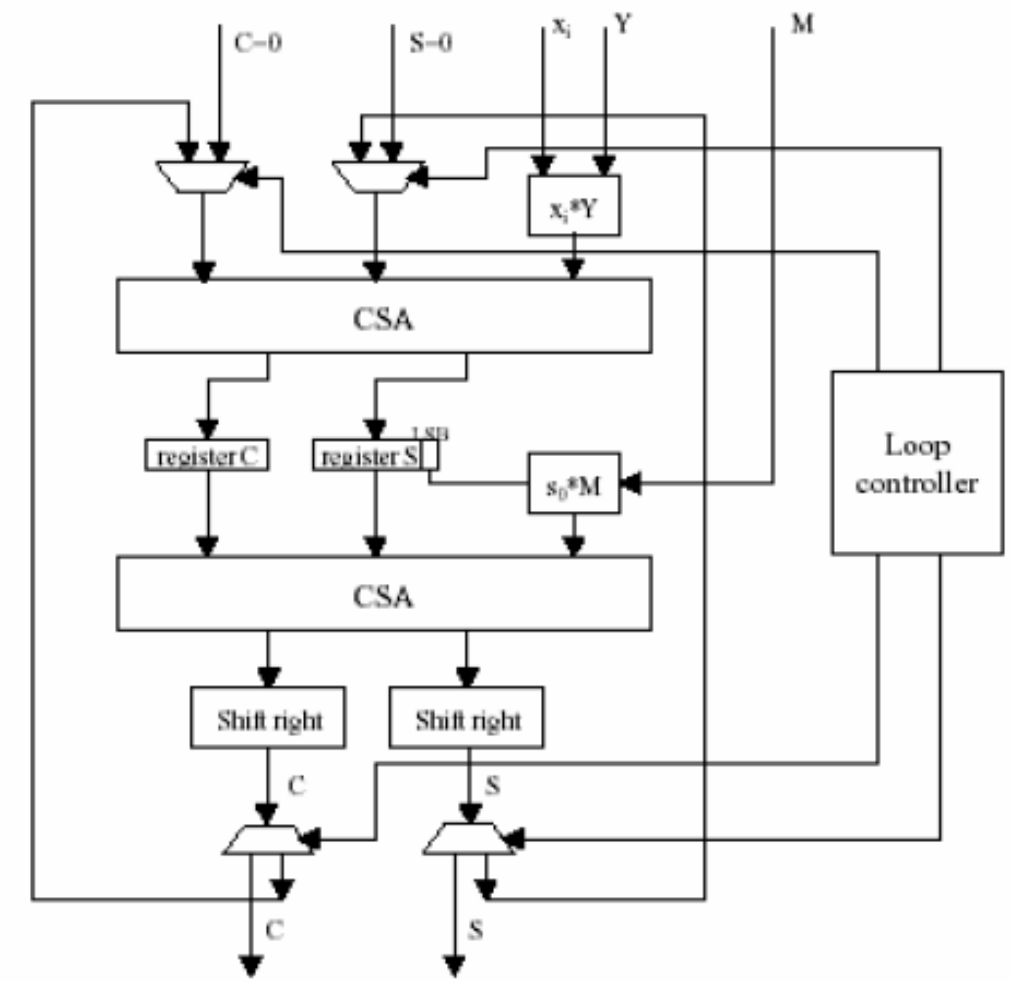

Figure 4.1: Architecture of the loop of algorithm 2

4.3.4 Faster Montgomery Algorithm

In Figure 1, the layout for the implementation of the loop of Algorithm 2 consists of two carry save adders. For large wordsizes (e.g. n = 1024 or higher), this would require considerable hardware resources to implement the architecture of Algorithm 2. The motivation behind this optimized algorithm is that of reducing the chip area for practical hardware implementation of Algorithm 2. This is possible if we can precompute the four possible values to be added to the intermediate result within the loop of Algorithm 2, thereby reducing the number of carry save adders from 2 to 1. There are four possible scenarios:

- If the sum of the old values of *S* and *C* is an even number, and if the actual bit *xi* of X is 0, then we add 0 before we perform the reduction of *S* and *C* by division by 2.
- If the sum of the old values of *S* and *C* is an odd number, and if the actual bit *xi* of X is 0, then we must add *M* to make the intermediate result even. Afterwards, we divide *S* and *C* by 2.
- If the sum of the old values of *S* and *C* is an even number, and if the actual bit *xi* of X is 1, but the increment *xi *Y* is even, too, then we do not need to add *M* to make the intermediate result even. Thus, in the loop we add *Y* before we perform the reduction of *S* and *C* by division by 2. The same action is necessary if the sum of *S* and *C* is odd, and if the actual bit *xi* of X is 1 and *Y* is odd as well. In this case, *S*+*C*+*Y* is an even number, too. [19]
- If the sum of the old values of *S* and *C* is odd, the actual bit *xi* of X is 1, but the increment *xi *Y* is even, then we must add *Y* and *M* to make the intermediate result even. Thus, in the loop we add *Y*+*M* before we perform the reduction of *S* and *C* by division by 2. The same action is necessary if the sum of *S* and *C* is even, and the actual bit *xi* of X is 1, and *Y* is odd. In this case, *S*+*C*+*Y*+*M* is an even number, too. The computation of *Y*+*M* can be done prior to the loop. This saves one of the two additions which are replaced by the choice of the right operand to be added to the old values of *S* and *C*. Algorithm 3 is a modification of Montgomery's method which takes advantage of this idea. The advantage of Algorithm 3 in comparison to Algorithm 1 can be seen in the implementation of the loop of Algorithm 3 in Figure 2. The possible values of *I* are stored in a lookup-table, which is addressed by the actual values of *xi*, *y0*, *s0* and *c0*. The operations in the loop are now reduced to one table lookup and one carry save addition. Both these activities can be performed

concurrently. Note that the shift right operations that implement the division by 2 can be done by routing.

***Algorithm 3*:** Faster Montgomery multiplication

$Inputs : X, Y, M \quad with \quad 0 \le X, Y < M$
$Outputs : P = (X * Y(2^n)^{-1}) \mod M$
$n : number \quad of \quad bits \quad in \quad X;$
$x_i : i^{th} \quad bit \quad of \quad X$
$s_0 : LSB \quad of \quad S \;, c_0 : LSB \quad of \quad C \;, y_0 : LSB \quad of \quad Y;$
$R : precompute \quad d \quad value \quad of \quad Y + M;$

(1) $S := 0; \quad C := 0;$
(2) $for \;\; (i = 0; i < n \;\; ; \;\; i++)\{$
(3) $if \;\; ((s_0 = c_0) \;\; and \;\; not \;\; x_i) \;\; then \;\; I := 0;$
(4) $if \;\; ((s_0 \neq c_0) \;\; and \;\; not \;\; x_i) \;\; then \;\; I := M \;;$
(5) $if \;\; (not \;\; (s_0 \oplus c_0 \oplus y_0) \;\; and \;\; x_i) \;\; then \;\; I := Y;$
(6) $if \;\; ((s_0 \oplus c_0 \oplus y_0) \;\; and \;\; x_i) \;\; then \;\; I := R;$
(7) $S, C := S + C + I,$
(8) $S := S \;\; div \;\; 2; \quad C := C \;\; div \;\; 2; \;\; \}$
(9) $P := S + C;$
(10) $if \;\; (P \ge M) \;\; then \;\; P := P - M;$

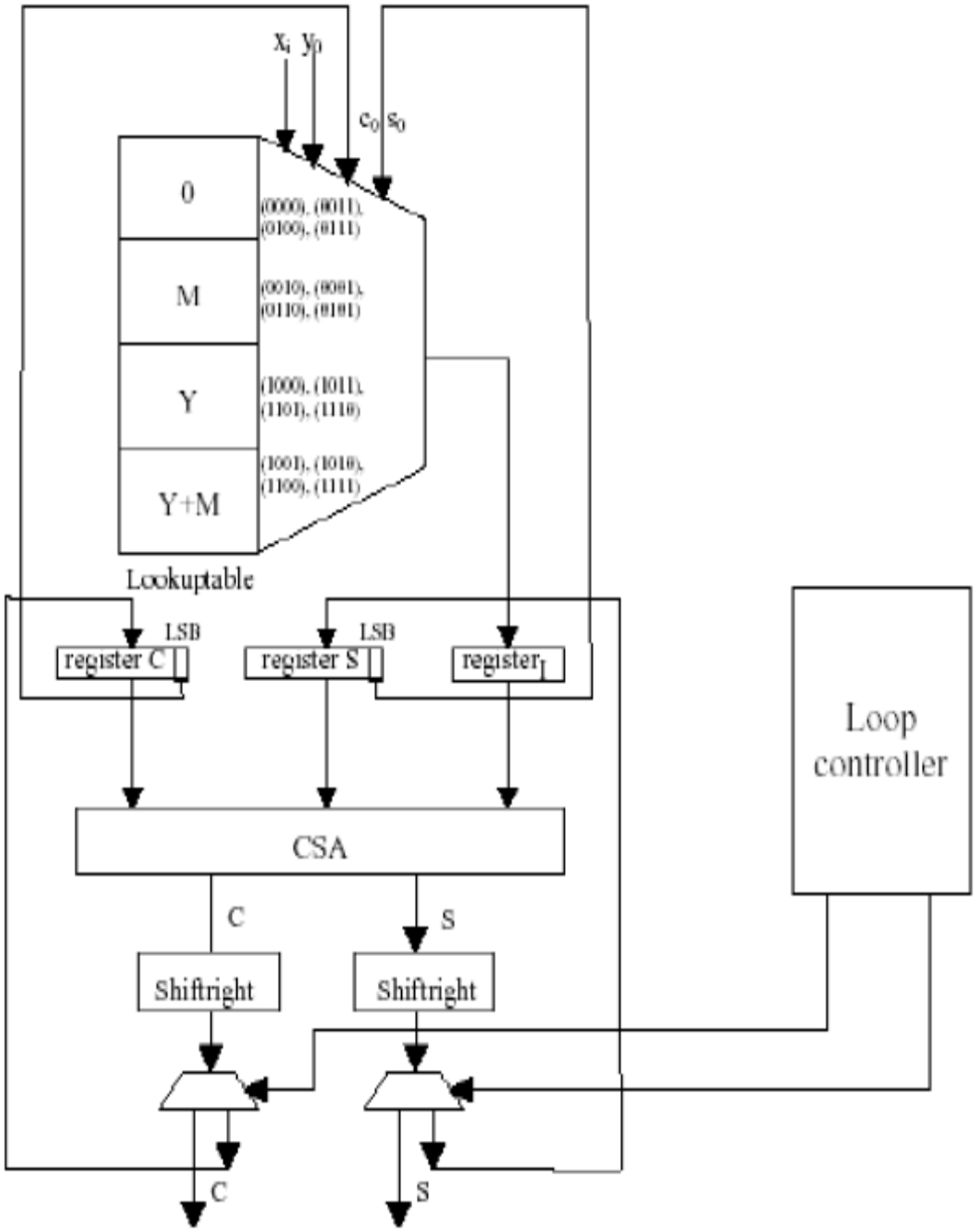

Figure 4.2: Architecture of Algorithm 3

4.4 The wide used adders used for modular multiplication [17]

4.4.1 Full Adder and half Adder cells:

The building blocks of different types of adders are the full adders (FA) and half adders (HA) cells.thus,we briefly introduce them here .Afull adder is a combinational circuit with 3inputs and 2 utputs. The inputs A_i,B_i,C_i and the outputs S_i and C_{i+1} are Boolean variables. It is assumed that A_i and B_i are the i^{th} bits of the integers A and B, respectively, and C_i is the carry bit received by the i^{th} position. The FA cell computes the sum bit S_i and the carry out bit C_{i+1} wich is is to be received by the next cell. The truth table of the FA cell is as follows:

A_i	B_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The Boolean function of the output values are as

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i,$$
$$S_i = A_i \oplus B_i \oplus C_i.$$

Similarly, a half adder is a combinational circuit with 2 inputs and 2 outputs. The inputs A_i and B_i and the outputs S_i and C_{i+1} are Boolean variables. It is assumed that A_i and B_i are the i^{th} bits of the integers A and B, respectively. The HA cell computes the sum bit S_i and the carry out bit C_{i+1}. Thus, a half adder is easily obtained by setting the third input C_i to zero. The truth table of the HA cell is as follows: [17]

A_i	B_i	C_{i+1}	S_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The Boolean functions of the output values are as $C_{i+1} = A_i B_i$ and $S_i = A_i \oplus B_i$, which can be obtained by setting the carry bit input C_i of the FA cell to zero. The following figures illustrates the FA and HA cells.

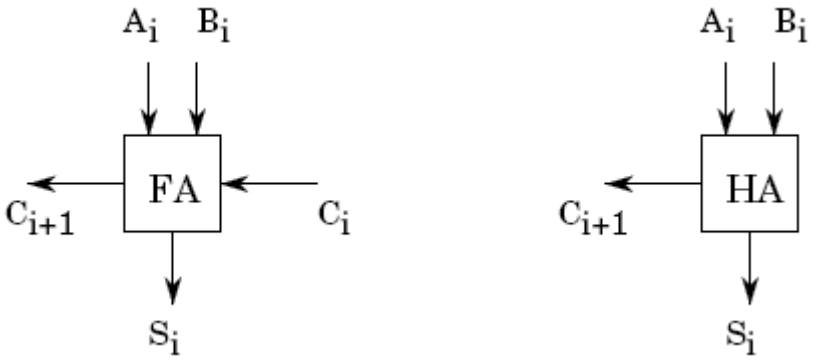

Figure 4.3: Full Adder and Half Adder cells

4.4.2 Carry save adder: *(for Montgomery and interleaved multiplication)*

The core operation of most algorithms for modular multiplication is addition. There are several different methods for addition in hardware: carry ripple addition, carry select addition, carry look ahead addition and others. The disadvantage of these methods is the carry propagation, which is directly proportional to the length of the operands. This is not a big problem for operands of size 32 or 64 bits but the typical operand size in cryptographic applications range from 160 to 2048 bits. The resulting delay has a significant influence on the time complexity of these adders. The carry save adder seems to be the most useful adder for modular multiplication architectures .it is simply a parallel ensemble of K full-adders without any horizontal connection. Its main function is to add three K-bit integers A,B, and C to produce two integers C' and S such that [17]

$$C' + S = A + B + C$$

As an example, let A=40, B=25 and C=20, we compute S and C' as shown below:

$$\begin{array}{ll} A = 40 = & 101000 \\ B = 25 = & 011001 \\ C = 20 = & 010100 \\ \hline S = 37 = & 100101 \\ C' = 48 = 0 & 11000 \\ \hline \end{array}$$

The i^{th} bit of the sum S_i and the (i+1)st bit of the carry C'_{i+1} is calculating using the equations

$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i'$$

In other words a carry save adder cell is just a full-adder cell.a carry save adder ,sometimes named a one-level CSA,is illustrated below for K=6.

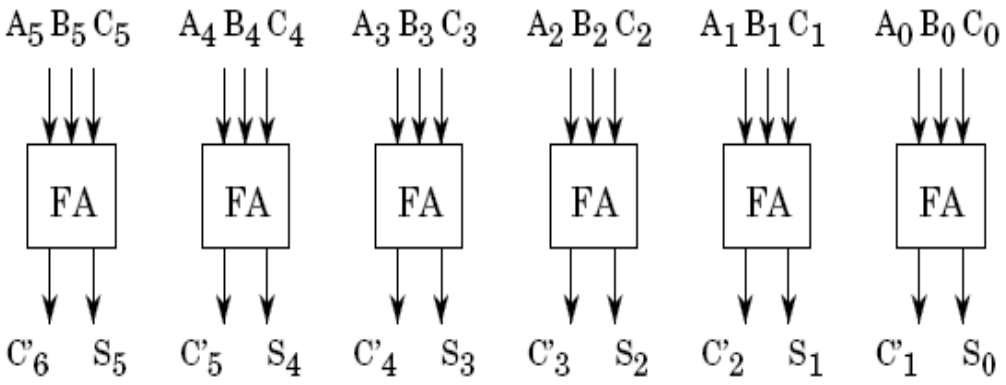

Figure 4.4: one-level CSA for K=6.

Since the input vectors A, B, and C are applied in parallel, the total delay of a carry save adder is equal to the total delay of a single FA cell.thus, the addition of three integers to compute two integers requires a single FA delay. Furthermore, the CSA requires only K times the areas of FA cell, and scales up very easily by adding more parallel cells. The subtraction operation can also be performed by using 2's complement encoding. There are basically two disadvantages of the carry save adders: [17]

- It does not really solve the problem of adding two integers and producing a single output.instead,it adds three integers and produces two such that sum of

these two is equal to the sum of three inputs .this method may not be suitable for application wich only needs the regular addition.

- The sign detection is hard: when a number is represented as a carry save pair (C,S) such that its actual value is C+S , we may not know the exact sign of total sum C+S. unless the addition is performed in full length ,the correct sign may never be determined.

4.4.3 Carry Delayed Adder :*(for Brickell's algorithm)*

The carry delayed adder is a two level carry save adder, a certain property of the carry delayed adder can be used to reduce the multiplication complexity. The carry delayed adder produced a pair of integers (*D, T*), called a carry delayed number, using the following set of equations:

$$S_i = A_i \oplus B_i \oplus C_i,$$
$$C_{i+1} = A_iB_i + A_iC_i + B_iC_i \ ,$$
$$T_i = S_i \oplus C_i \ ,$$
$$D_{i+1} = S_iC_i \ ,$$

Where $D_0 = 0$. notice that C_{i+1} and S_i are the outputs of a full adder cell which inputs A_i , B_i and C_i , while the values D_{i+1} and T_i are the outputs of an half adder cell.

An important property of the carry delayed adder is that $D_{i+1}T_i=0$ for all i=0, 1,…, K-1. This is easily verified as:

$$D_{i+1}T_i = S_iC_i(S_i \oplus C_i) = S_iC_i(\overline{S_i}C_i + S_i\overline{C_i}) = 0$$

As an example, let *A*=40, *B*=25, and *C*=20. In the first level, we compute the carry save pair(C, S) using the carry save equations. In the second level, we compute the carry delayed pair (*D, T*) using the definitions $D_{i+1} = S_iC_i$ and $T_i = S_i \oplus C_i$ as: [17]

A	= 40 =		101000
B	= 25 =		011001
C	= 20 =		010100
S	= 37 =		100101
C	= 48 =	0	110000
T	= 21 =		010101
D	= 64 =	1	000000

Thus, the carry delayed pair (64, 21) represents the total of $A+B+C=85$.the property of the carry delayed pair that $T_iD_{i+1}=0$ for all $i=0, 1... K-1$ also holds.

T	= 21 =		010101
D	= 64 =	1	000000
$T_i D_{i+1}$	=		000000

We will explore this property in the brickell's method for computing the modular multiplication. The following figure illustrates the carry delayed adder for $K=6$.

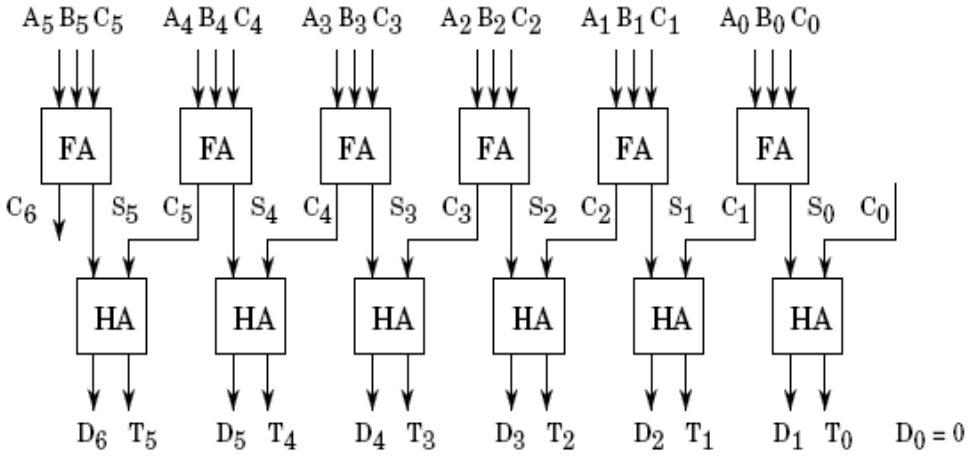

Figure 4.5: the carry delayed adder for K=6

4.5 Interleaved Modular Multiplication

Another well known algorithm for modular multiplication is the interleaved modular multiplication. The details of the method are sketched in [18]. The idea is to interleave multiplication and reduction such that the intermediate results are kept as short as possible.

As shown in [Algorithm 4], the computation of P requires n steps and at each step we perform the following operations: [20]

- A left shift : $2*P$
- A partial product computation : x_i*Y
- An addition : $2*P + x_i*Y$
- At most two subtractions :

  If $(P \geq M)$ then $P: = P-M$

  If $(P \geq M)$ then $P: = P-M$

4.5.1 Algorithm 4: The interleaved modular multiplication algorithm

$$
\begin{aligned}
&Inputs: X, Y, M \ with \ 0 \leq X, Y < M \\
&Output: P = X * Y \bmod M \\
&n: number of \ bits \ \ in \ X; \\
&x_i : i^{th} \ bit \ of \ X; \\
&(1) \quad p := 0; \\
&(2) \quad for \ (i = n-1; i \geq 0 \ ; i--)\{ \\
&(3) \quad\quad P := 2 * P; \\
&(4) \quad\quad I := x_i * Y; \\
&(5) \quad\quad P := P + I; \\
&(6) \quad if \ (P \geq M) \ then \ P := P - M; \\
&(7) \quad if \ (P \geq M) \ then \ P := P - M; \ \}
\end{aligned}
$$

The main advantages of the interleaved Algorithm compared to the separated multiplication and division are the following:

- Only one loop is required for the whole operation.
- The intermediate results are never any longer than n+2 bits (thus reducing the area for registers and full adders).

But there are some disadvantages as well:

- The algorithm requires three additions with carry propagation in steps (5), (6) and (7).
- In order to perform the comparisons in steps (4) and (5), the preceding additions have to be completed. This is important for the latency because the operands are large and, therefore, the carry propagation has a significant influence on the latency.
- The comparison in step (6) and (7) also requires the inspection of the full bit lengths of the operands in the worst case. In contrast to addition, the comparison is performed MSB first. Therefore, these two operations cannot be pipelined without delay. [19]

4.5.2 Optimized Interleaved Algorithm

The Algorithm 5 is an optimization of the interleaved modular multiplication. There are four details of the standard interleaved Algorithm were modified:

- The intermediate results are no longer compared to M (as in steps (6) and (7) of the standard Algorithm. Rather, a comparison to $k*2n$ (k=0... 6) is performed which can be done in constant time. This comparison is done implicitly in the mod-operation in step (11) of Algorithm 5.
- Subtractions in steps (6), (7) of the standard interleaved Algorithm are replaced by one subtraction of $k*2n$ which can be done in constant time by bit masking.
- Next, the value of $k*2^n$ mod M is added in order to generate the correct intermediate result (step (11) of Algorithm 5).
- Finally, carry save adders are used to perform the additions inside the loop, thereby reducing the latency to a constant. The intermediate results are in redundant form, coded in two words S and C instead of generated one word P.

These changes made by the authors in [20] led to Algorithm 5, which looks more complicated than the standard Algorithm. Its main advantage is the fact that all the computations in the loop can be performed in constant time. Hence, the time complexity of the whole algorithm is reduced to O (n), provided the values of $k*2^n$ mod M are pre computed before execution of the loop. [18]

Algorithm 5: Modular multiplication using carry save addition

$Inputs: X, Y, M \ with \ 0 \le X, Y < M$

$Outputs: P = X * Y \ \text{mod} \ M$

$n: number \ of \ bits \ in \ X;$

$x_i: i^{th} bit \ of \ X;$

(1) $S := 0 \ ; C := 0 \ ; A := 0 \ ;$

(2) $for \ (i = n-1 \ ; i \ge 0 \ ; i--)\{$

(3) $S := S \ \text{mod} \ 2^n;$

(4) $C := C \ \text{mod} \ 2^n;$

(5) $S := 2 * S;$

(6) $C := 2 * C;$

(7) $A := 2 * A;$

(8) $I := x_i * Y;$

(9) $(S, C) := CSA \ (S, C, I);$

(10) $(S, C) := CSA \ (S, C, A);$

(11) $A := (2 * S_{n+1} + S_n + 2 * C_{n+1} + C_n) * 2^n \ \text{mod} \ M \ \}$

(12) $P := (S + C) \ \text{mod} \ M;$

The architecture for the implementation of the loop of Algorithm 5 can be seen in the hardware layout in Figure 6.

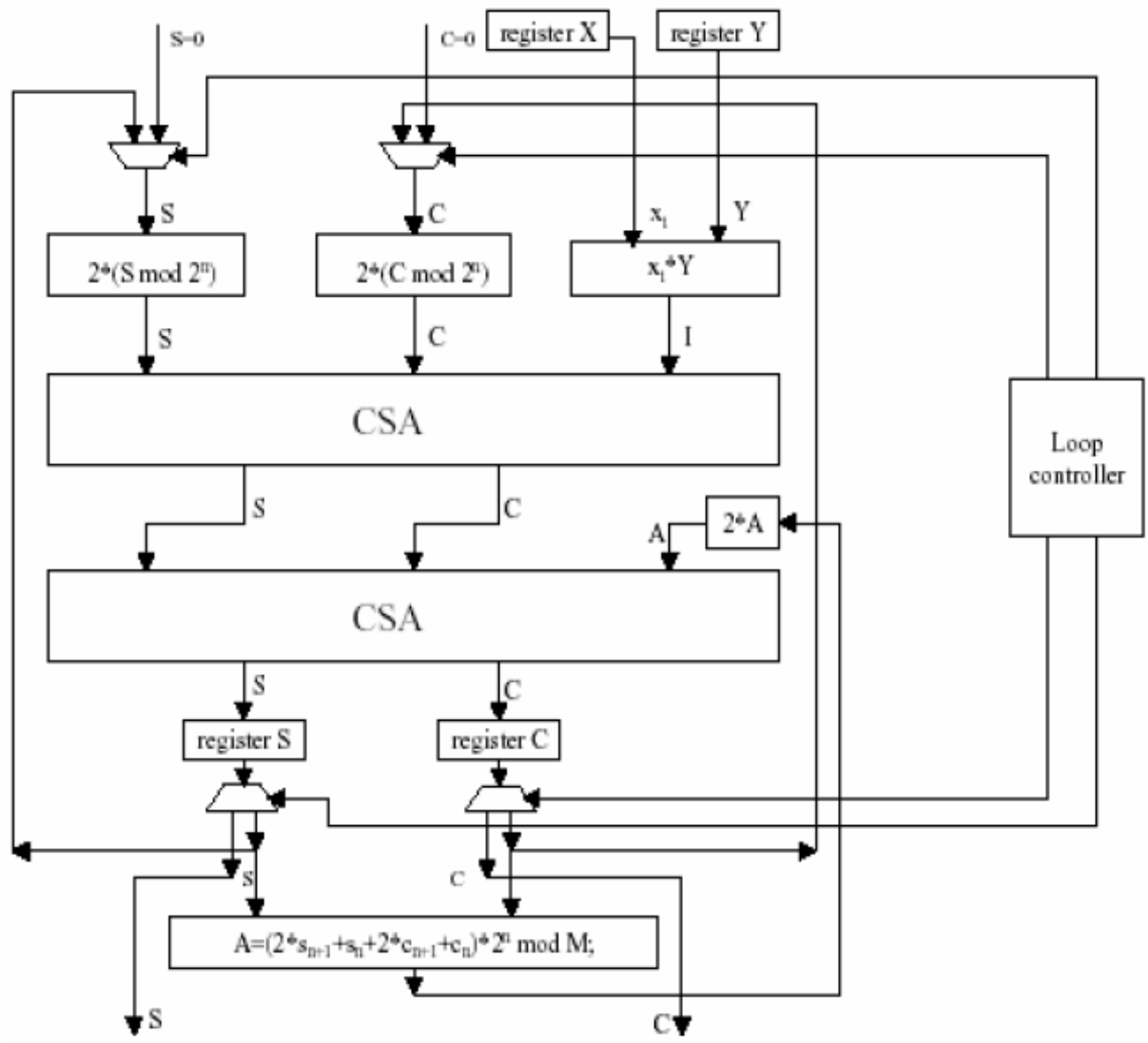

Figure 4.6: Inner loop of modular multiplication using carry save addition

In [20], the authors showed how to reduce both area and time by further exploiting precalculation of values in a lookup-table and thus saving one carry save adder. The basic idea is:

The increment *I* within the loop can have only two possible values: i.e., 0 or *Y*. If *xi* is 0 then *I* is 0 as well. If *xi* is 1 then *I* is *Y*. taking this into account we can reduce the number of operands to be added in the loop from four to three. In a precomputing phase each possible value for $2A+I$ is determined and stored in the lookup-table. During execution of the loop the lookup-table is addressed by the sum of the two leading digits of $S+C$ and by *xi*. By these constructions, the number of carry save adders is reduced to 1 in Algorithm 6 [19]

***Algorithm 6*: Optimized version of the algorithm 5**

$Inputs: X, Y, M \ with \ 0 \le X, Y < M$

$Output: P = X * Y \ \text{mod} \ M$

$n: number \ of \ bits \ in X;$

$x_i: i^{th} bit \ of \ X;$

$\Pr ecomputing: LookUp\ (7) - - - LookUp\ (0);$

(1) $S := 0;$

(2) $C := 0;$

(3) $A := LookUp\ (x_{n-1})$

(4) $for\ (i = n-1\ ; i \ge 0\ ; i--)\{$

(5) $S := S \ \text{mod} \ 2^n;$

(6) $C := C \ \text{mod} \ 2^n;$

(7) $S := 2 * S;$

(8) $C := 2 * C;$

(9) $(S.C) := CSA\ (S, C, A);$

(10) $A := LookUp(2 * (s_n + 2 * c_{n+1} + c_n) + x_{i-1});\ \}$

(11) $P := (S + C) \ \text{mod} \ M;$

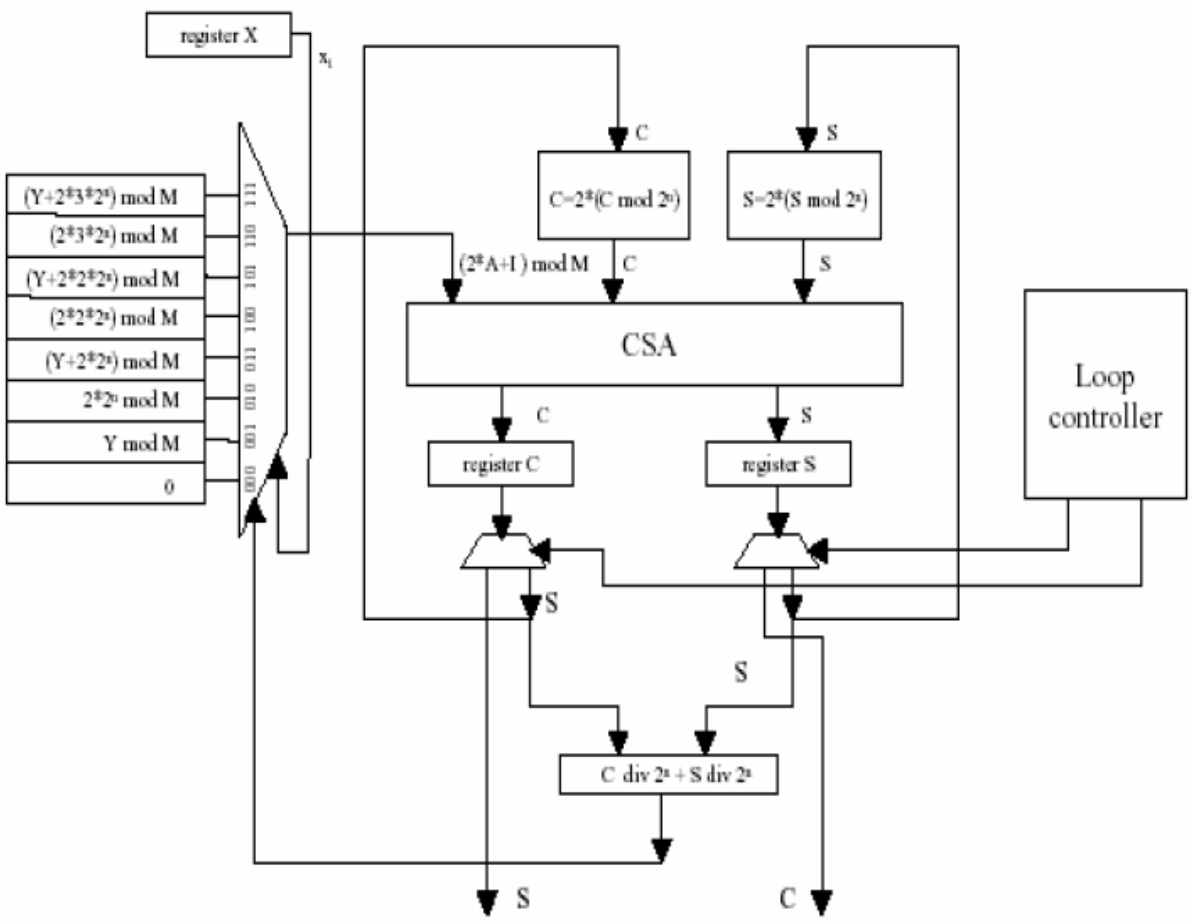

Figure 4.7: Modular multiplications with one carry save adder

4.6 Brickell's Method

This method is based on the use of carry delayed integer introduced in the carry delayed adder section. Let A be a carry delayed integer, then, it can be written as:

$$A = \sum_{i=0}^{K-1} (T_i + D_i) \cdot 2^i.$$

The product P=AB can be computed by summing the terms:

$$
\begin{aligned}
&(T_0 \cdot B + D_0 \cdot B) \cdot 2^0 + \\
&(T_1 \cdot B + D_1 \cdot B) \cdot 2^1 + \\
&(T_2 \cdot B + D_2 \cdot B) \cdot 2^2 + \\
&\quad . \\
&\quad . \\
&(T_{K-1} \cdot B + D_{K-1} \cdot B) \cdot 2^{K-1}
\end{aligned}
$$

Since D_0=0, we rearrange to obtain

$$
\begin{aligned}
&2^0 \cdot T_0 \cdot B + 2^1 \cdot D_1 \cdot B + \\
&2^1 \cdot T_1 \cdot B + 2^2 \cdot D_2 \cdot B + \\
&2^2 \cdot T_2 \cdot B + 2^3 \cdot D_3 \cdot B + \\
&\quad . \\
&\quad . \\
&2^{K-2} \cdot T_{K-2} \cdot B + 2^{K-1} \cdot D_{K-1} \cdot B + \\
&\quad 2^{K-1} \cdot T_{K-1} \cdot B
\end{aligned}
$$

Also recall that either T_i or D_{i+1} is zero due to the property of the carry delayed adder. Thus, each step requires a shift of B and addition of at most 2 carry delayed integers:

- Either: $(P_d, P_t) := (P_d, P_t) + 2^i \cdot T_i \cdot B$
- Or: $(P_d, P_t) := (P_d, P_t) + 2^{i+1} \cdot D_{i+1} \cdot B$

After K steps P = (P_d, P_t) is obtained. In order to compute P (mod n) , we perform reduction:

$$
\begin{aligned}
&If \quad P \geq 2^{K-1} \cdot n \quad then \quad P := P - 2^{K-1} \cdot n \\
&If \quad P \geq 2^{K-2} \cdot n \quad then \quad P := P - 2^{K-2} \cdot n \\
&If \quad P \geq 2^{K-3} \cdot n \quad then \quad P := P - 2^{K-3} \cdot n \\
&\quad . \\
&\quad . \\
&If \quad P \geq n \quad then \quad P := P - n
\end{aligned}
$$

We can also reverse these steps to obtain:

$$P := T_{K-1} \cdot B \cdot 2^{K-1}$$
$$P := P + T_{K-2} \cdot B \cdot 2^{K-2} + D_{K-1} \cdot B \cdot 2^{K-1}$$
$$P := P + T_{K-3} \cdot B \cdot 2^{K-3} + D_{K-2} \cdot B \cdot 2^{K-2}$$
$$\vdots$$
$$P := P + T_1 \cdot B \cdot 2^1 + D_2 \cdot B \cdot 2^2$$
$$P := P + T_0 \cdot B \cdot 2^0 + D_1 \cdot B \cdot 2^1$$

Also, the multiplication steps can be interleaved with reduction steps. To perform the reduction, the sign of $P - 2^i \cdot n$ needs to be determined (estimated). [17]

4.7 Conclusion

In this chapter we presented the different algorithms for modular multiplication and the characteristics of each one. Montgomery and the interleaved algorithms represented in the explained architectures are the most popular techniques for hardware computation of modular product. We can also find different presentations of this algorithms depending to the kind of addition operation as carry save and carry delayed or other adders used.

As a note there is an other kind of techniques based on systolic array architectures, represented in an arrangement of processing elements in three dimensions. But this technique is not in our field of interest.

In the next chapter we will see the hardware design of modular multiplication using some techniques studied in this chapter by using different bit lengths as 1024 dedicated to RSA cryptosystems and others in the range of hundreds dedicated to ECC cryptosystems. Also we will see the characteristics of each design.

5.1 Introduction

After the presentation of different algorithms and architectures for modular product, we will translate these algorithms to a hardware description code using VHDL. this chapter contain the many implemented architectures, with their different simulation (behavioral, post translate, post map and post route) using MODELSIM software and ISE of XILINX, we will present the characteristics of the core of each architecture (the clock frequency and used hardware resources) for different bit lengths of operators .

The second step is called in circuit verification, we will use The Virtex-II V2MB1000 Development Kit that contain Virtex-II device (XC2V1000-4FG456C), and the Chip Scope Pro Analyzer tool to perform in circuit verification (also known as on-chip debugging)

Finally we will give a comparison between these architectures in the spent time and the occupied area.

5.2 ISE design flow

The Integrated Software Environment (ISE™) is the Xilinx® design software suite that allows taking design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes the design through the following steps in the ISE design flow.

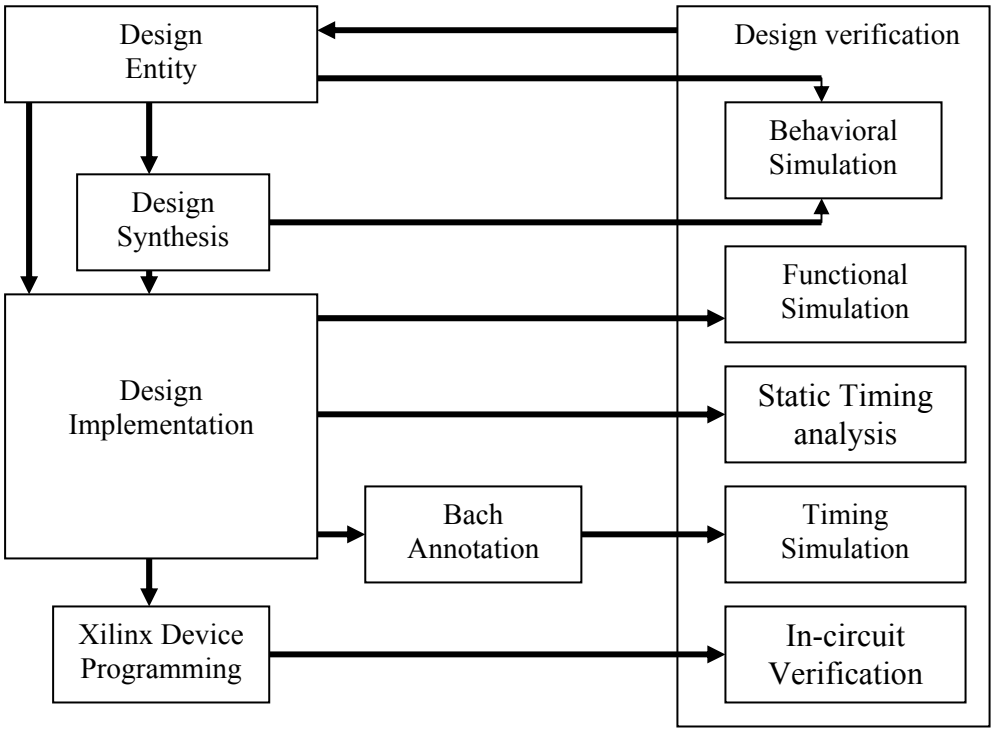

FIG 5.1: ISE design flow.

Design Entry

Design entry is the first step in the ISE design flow. During design entry, we create our source files based on the design objectives. We can create our top-level design file using a Hardware Description Language (HDL), such as VHDL, Verilog, or ABEL, or using a schematic. We can use multiple formats for the lower-level source files in the design.

Synthesis

After design entry and optional simulation, we run synthesis. During this step, VHDL, Verilog, or mixed language designs become netlist files that are accepted as input to the implementation step.

Implementation

After synthesis, we run design implementation, which converts the logical design into a physical file format that can be downloaded to the selected target device. From Project Navigator, we can run the implementation process in one step, or we can run each of the Pyou are targeting a Field Programmable Gate Array (FPGA) or a Complex Programmable Logic Device (CPLD).

Implementation includes many phases:

- ✓ *Translate*: Merge multiple design files into a single netlist
- ✓ *Map*: Group logical symbols from the netlist (gates) into physical components (slices and IOBs).
- ✓ *Place & Route*: Place components onto the chip, connect the components, and extract timing data into reports

Verification

We can verify the functionality of our design at several points in the design flow. We can use simulator software to verify the functionality and timing of our design or a portion of the design. The simulator interprets VHDL or Verilog code into circuit functionality and displays logical results of the described HDL to determine correct circuit operation. Simulation allows creating and verifying complex functions in a relatively small amount of time. We can also run in-circuit verification after programming the device.

Device Configuration

After generating a programming file, we configure our device. During configuration, we generate configuration files and download the programming files from a host computer to a Xilinx device.

In-circuit verification

After configuring the device, we can debug our FPGA design using ChipScope Pro™ software.

To use the ChipScope Pro software to perform in-circuit verification, we must do the following:

- Insert ChipScope Pro cores in our design using the ChipScope Pro Core Manager in either Core Generator or Core Inserter mode.
- Implement our design in Project Navigator and configure our device.
- Analyze our design using the ChipScope Pro Analyzer.

5.3 Implementation of the standard Montgomery multiplier:

The following block diagram shows the structure of the implemented architecture for standard Montgomery multiplication algorithm

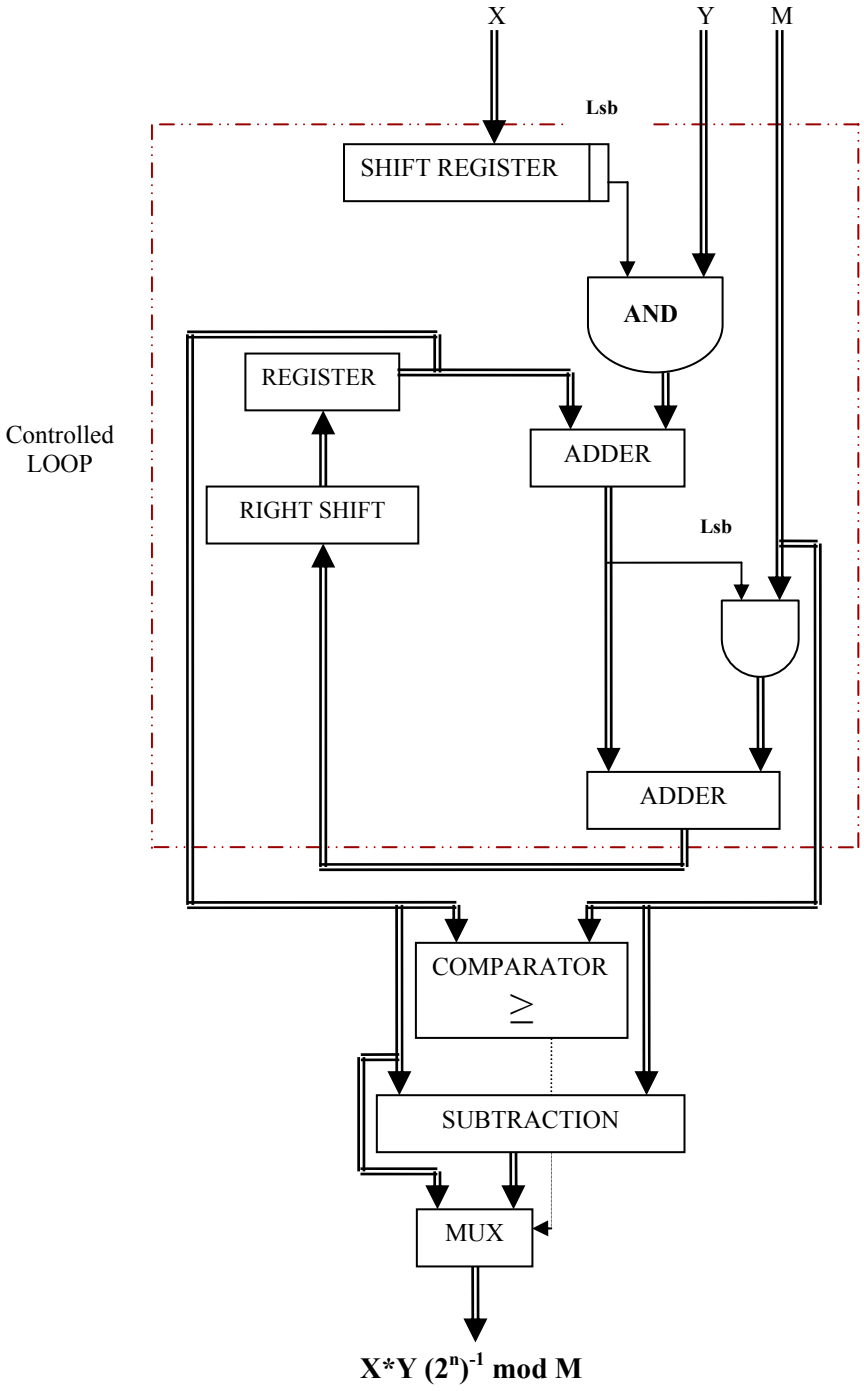

FIG 5.2: Architecture of the standard Montgomery multiplier

The following figure represents the behavioral simulation of this architecture by using MODELSIM SE 6.4b, from the figure we show that.

X = 234 ;

Y = 167 ;

M = 293 ;

Before entering data to Montgomery multiplier we have to convert at lest one operand to Montgomery domain

We have

$$X_{Mont} = Mont_{M}(X, R^2)$$
$$= (X . R^2 . R^{-1}) \mod M$$
$$= (X . R) \mod M$$

For our example we found :

R= 2^n ;

$R \bmod M = 2^n \bmod M$

$= 2^{16} \bmod 293$

$= 65536 \bmod 293$

$= 197$

Then $X_{Mont} = X * (R \bmod M) = 234*197 = 46098$

and we have also

$$P = Mont_{M}(X_{Mont}, Y)$$
$$= (X_{Mont} . Y . R^{-1}) \mod M$$
$$= (X . R . Y . R^{-1}) \mod M$$
$$= (X . Y) \mod M$$

Then the output of our simulation must be

P = Mont(*46098,* 167)

= (234*167) mod 293

= 109 ;

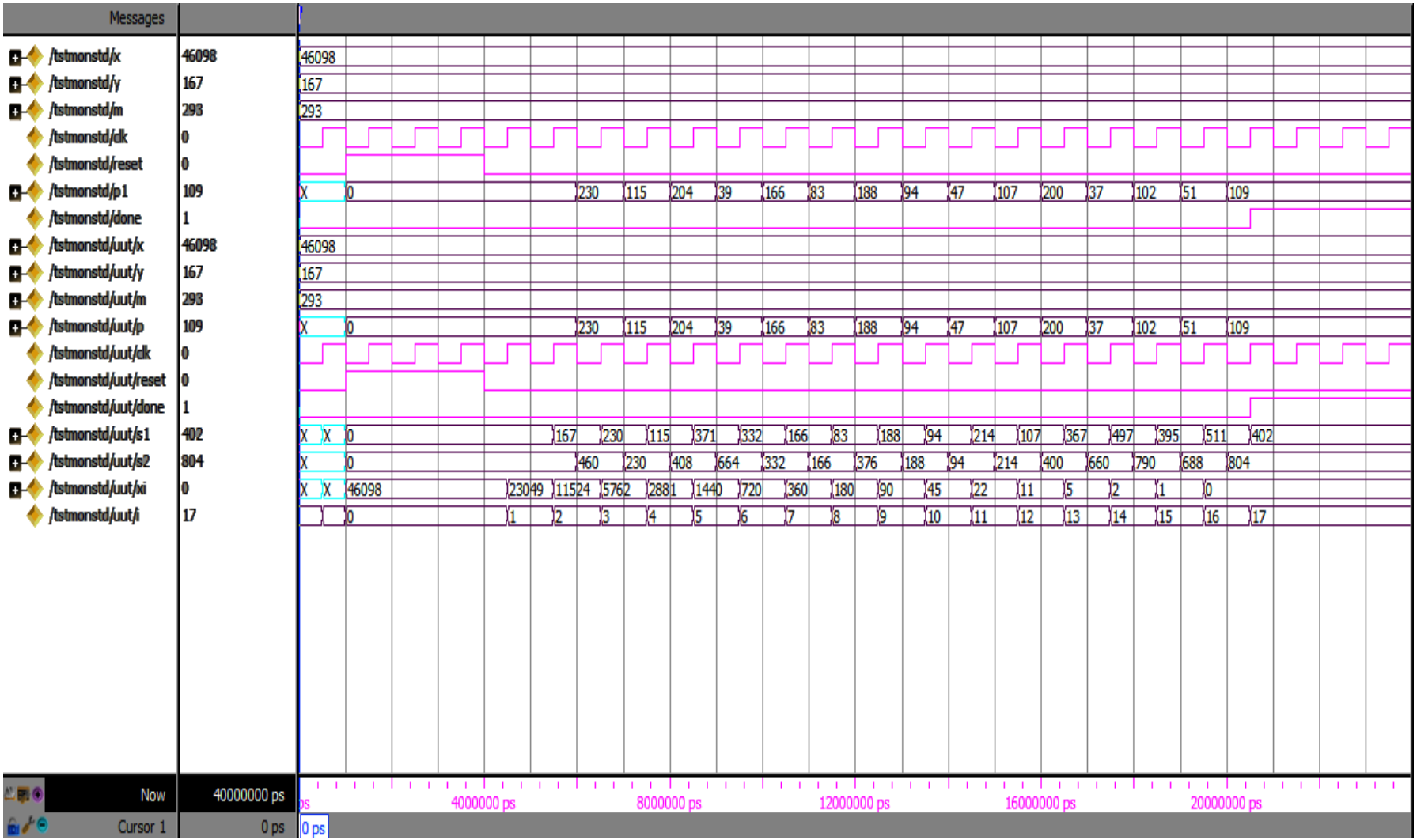

FIG 5.3: Behavioral Simulation of 16 bits standard Montgomery multiplier

5.4 Implementation of the Faster Montgomery multiplier:

The following block diagram shows the structure of the implemented architecture for faster Montgomery multiplication algorithm

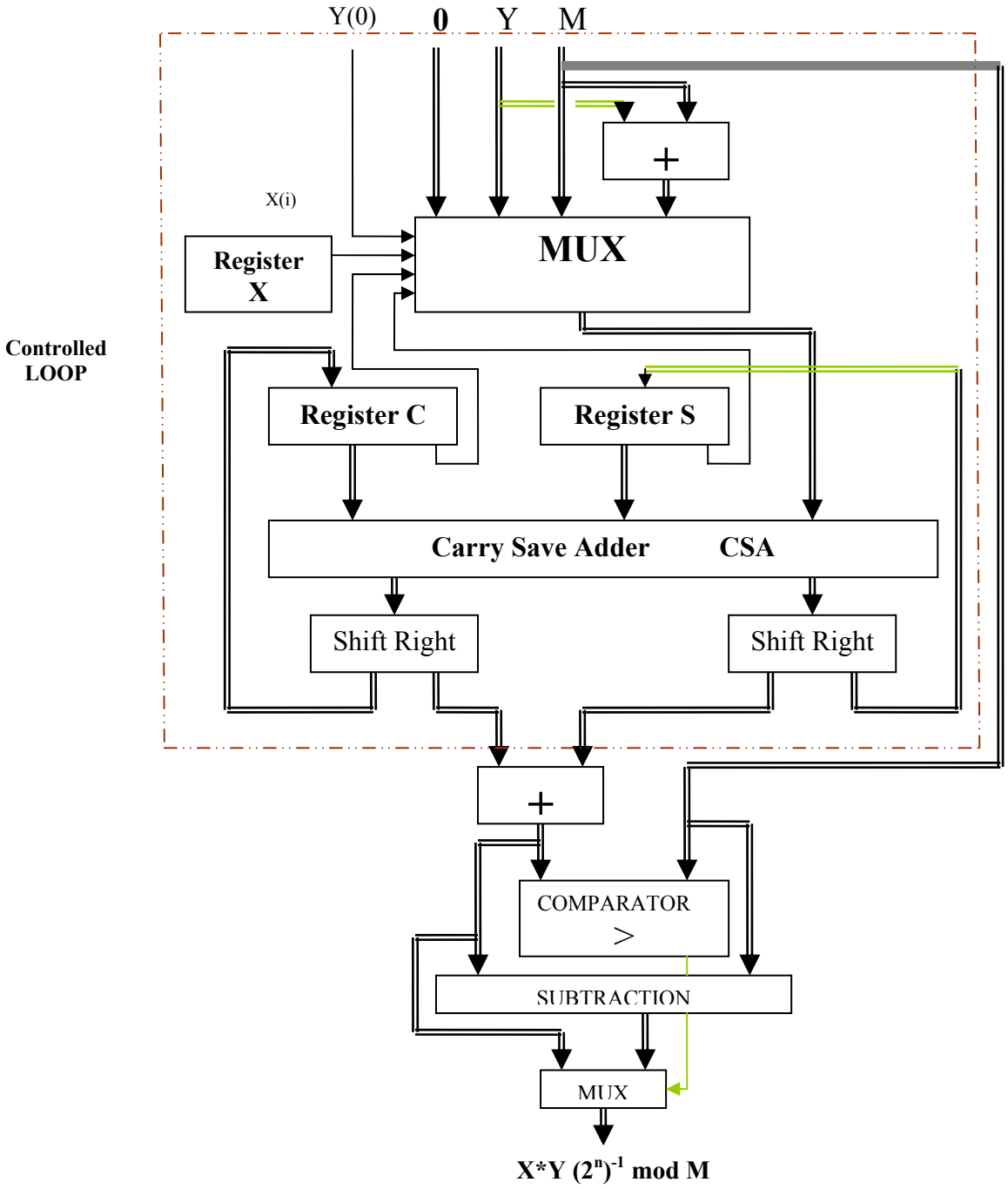

FIG 5.4: Architecture of the Implemented Faster Montgomery multiplier

The next figure represents the behavioral simulation of this architecture for the same previous example (X = 234 ;Y = 167 ;M = 293 ;)

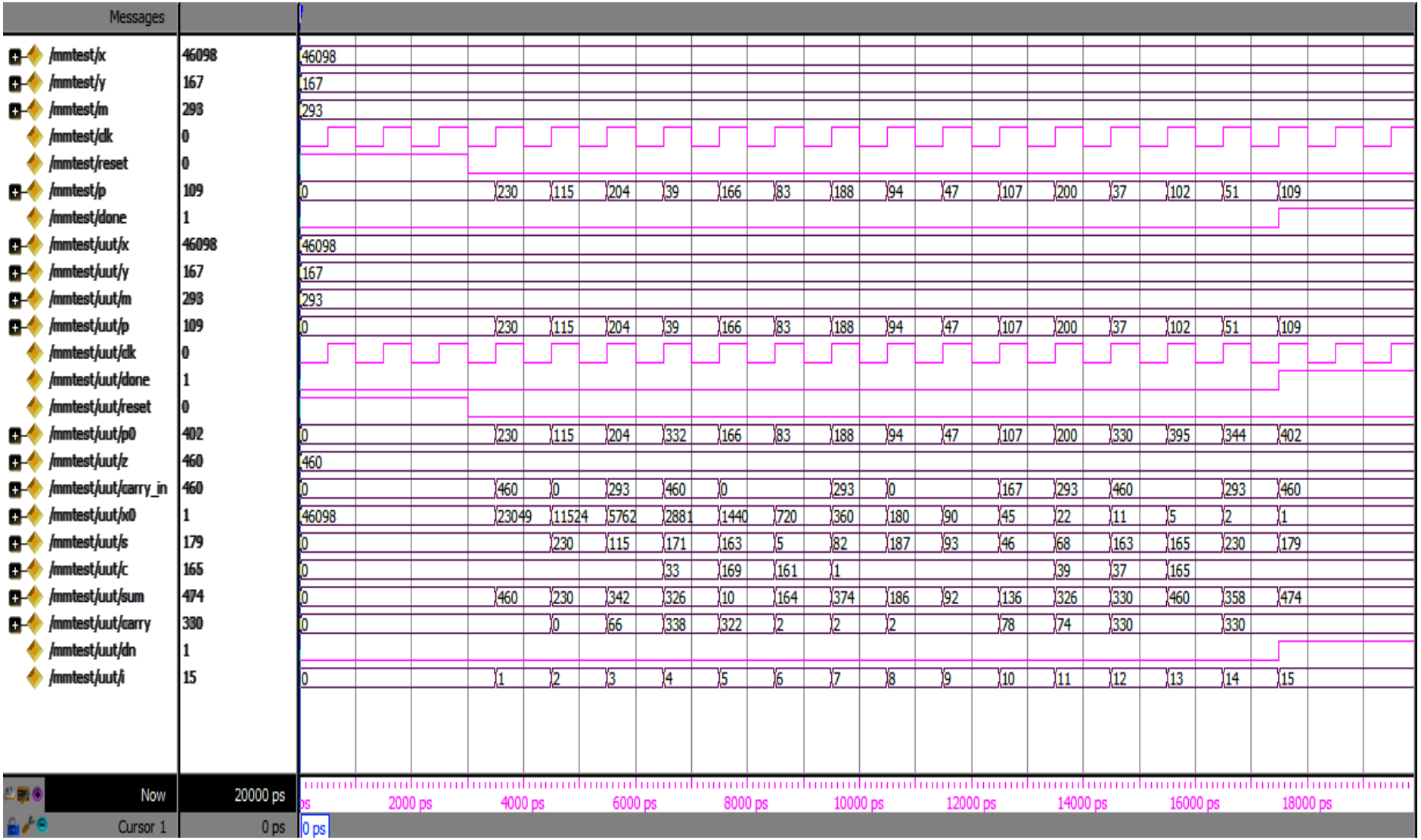

FIG 5.5: Behavioral Simulation of 16 bits Faster Montgomery multiplier

5.5 The Interleaved modular multiplication

The following block diagram shows the structure and the implemented architecture for the interleaved multiplication algorithm

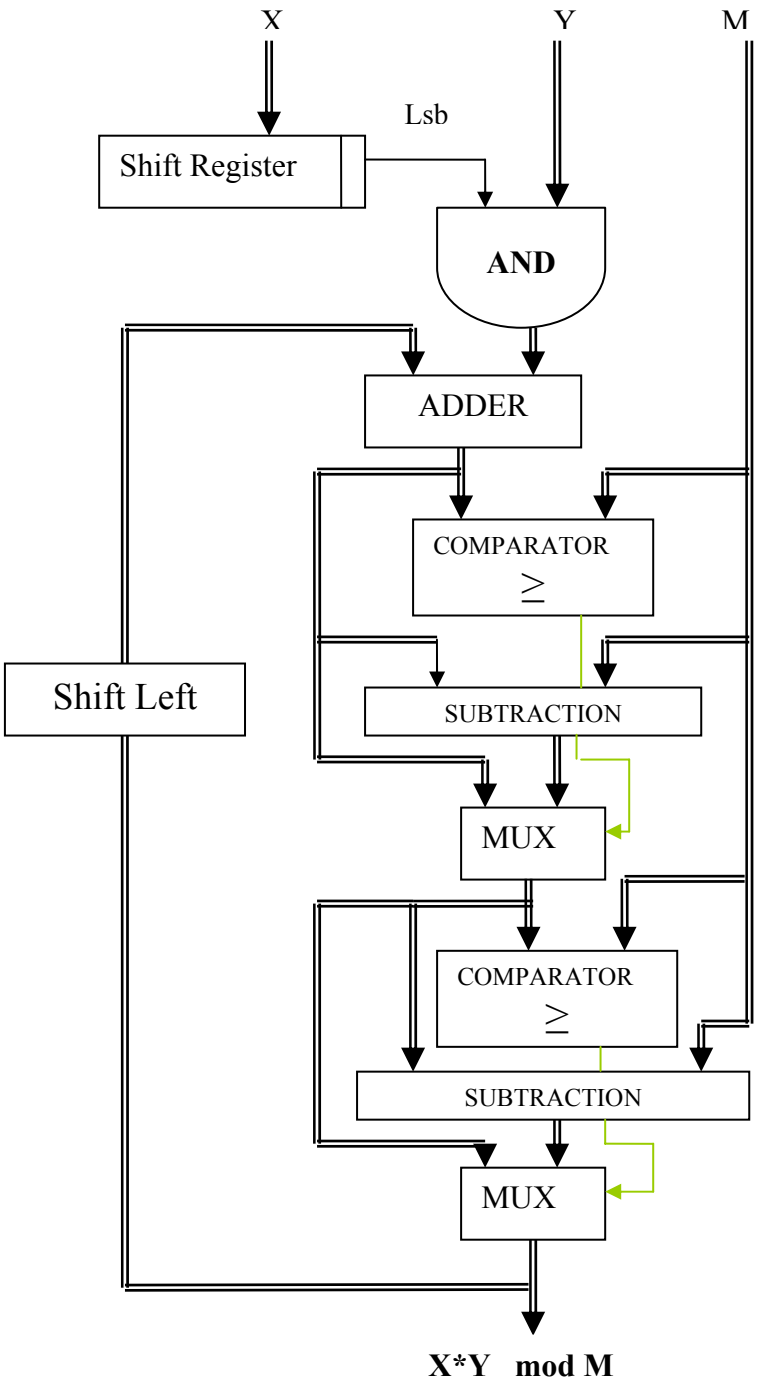

FIG 5.6: Architecture of the Implemented Interleaved multiplier

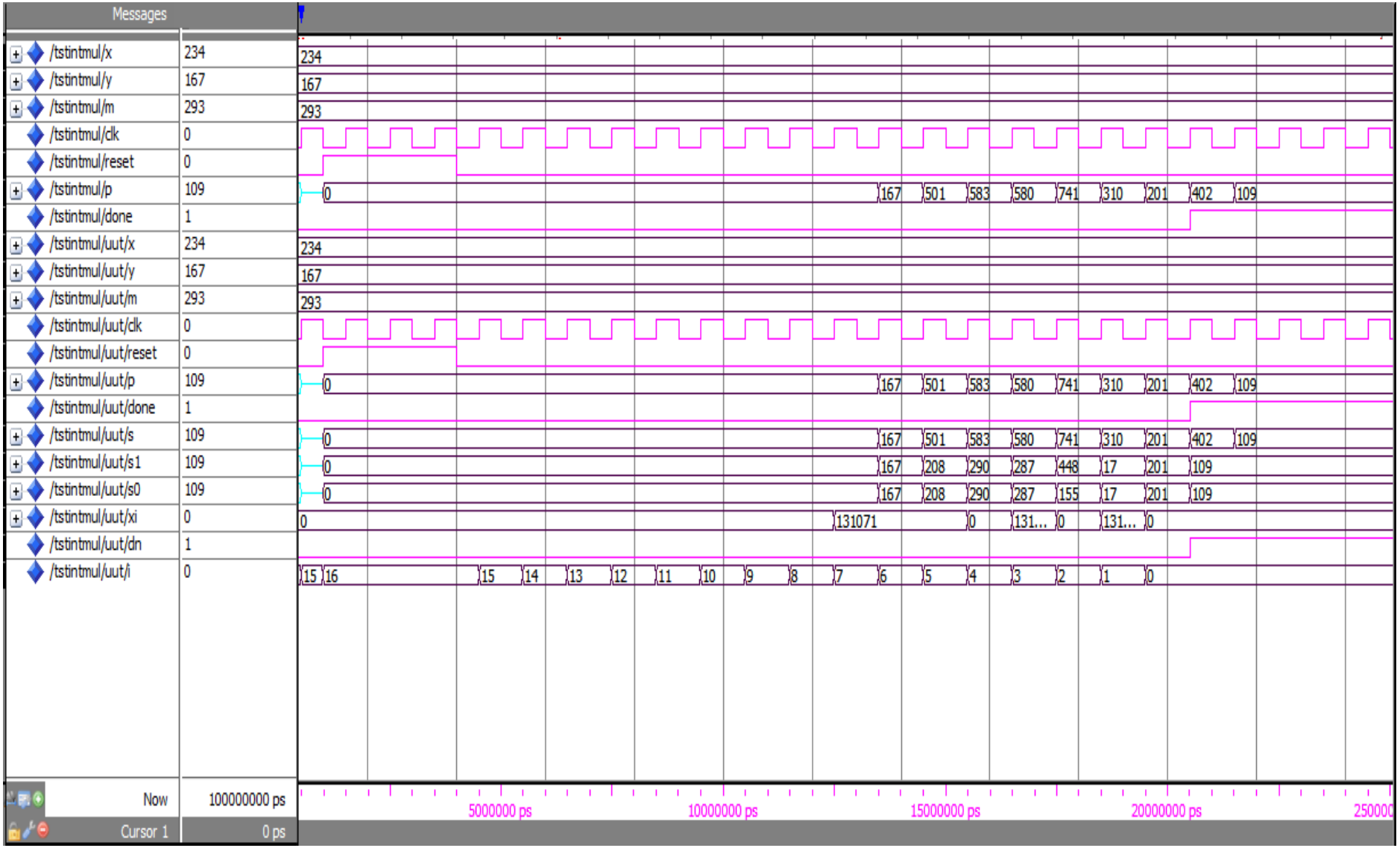

FIG 5.7: Behavioral Simulation of 16 bits Interleaved multiplier

5.6 Synthesis results

The design capture of the modular multiplication architectures using VHDL was done for 64 to 2048 bit. The basic units of the architectures which comprise carry save adders, comparators, full adders, multiplexers, shift registers (Left and Wright) and registers were modeled as components in the principal architecture.

The following figure represents the RTL schematic of the implemented multipliers.

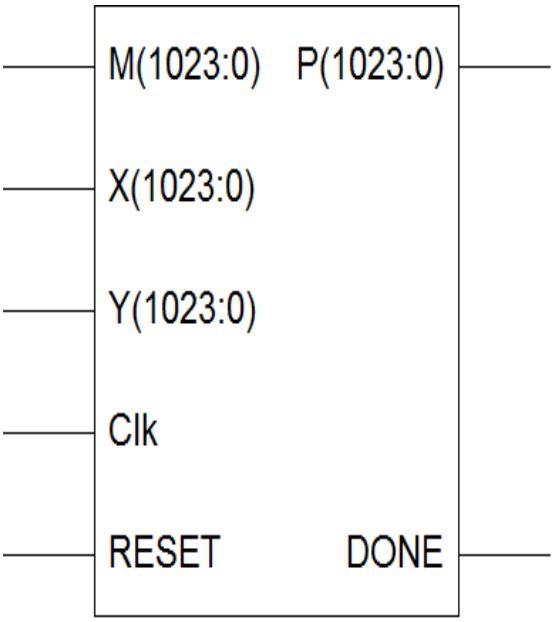

FIG 5.8: The RTL Schematic of different multipliers (example of 1024 bits)

The following tables show the used resources of FPGA Virtex 5 (xc5vlx50), in different word bit length from 64 to 2048 for the three architectures.

The tables contain the following primitives of FPGA:

- ✓ Slice Registers
- ✓ Slice LUTs : Slice Lookup Table
- ✓ LUT-FF pairs : Lookup Table/Flip-flop pairs
- ✓ BUFG/BUFGCTRLs : Global Clock Buffers / Global Clock Mux Buffer

Bit length	64	128	160	256	512	1024	2048
Number of Slice Registers	103 (0%)	165 (0%)	197 (0%)	293 (1%)	549 (1%)	1061 (3%)	2085 (7%)
Number of Slice LUTs	504 (1%)	952 3%)	1175 (4%)	1847 (6%)	3639 (12%)	7224 (25%)	14392 (49%)
Number of fully used LUT-FF pairs	93 (18%)	156 (16%)	187 (15%)	283 (15%)	538 (14%)	1049 (14%)	2072 (14%)
Number of BUFG/BUFGCTRLs	1 (3%)	1 (3%)	1 (3%)	1 (3%)	1 (3%)	1 (3%)	1 (3%)

Table 5.1: Device Utilization Summary of the standard interleaved multiplier

Bit length	64	128	160	256	512	1024	2048
Number of Slice Registers	229 (0%)	422 (1%)	518 (1%)	806 (2%)	1572 (5%)	3108 (10%)	6180 (21%)
Number of Slice LUTs	478 (1%)	894 (3%)	1102 (3%)	1726 (5%)	3390 (11%)	6718 (23%)	13374 (46%)
Number of fully used LUT-FF pairs	187 (35%)	348 (35%)	428 (35%)	669 (35%)	1308 (35%)	2587 (35%)	5147 (35%)
Number of BUFG/BUFGCTRLs	1 (3%)	1 (3%)	1 (3%)	1 (3%)	1 (3%)	1 (3%)	2 (6%)

Table 5.2: Device Utilization Summary of the standard Montgomery multiplier

Bit length	64	128	160	256	512	1024	2048
Number of Slice Registers	201 (0%)	394 (1%)	491 (1%)	779 (2%)	1548 (5%)	3085 (10%)	6158 (21%)
Number of Slice LUTs	369 (1%)	852 (2%)	1060 (3%)	1684 (5%)	3349 (11%)	6678 (23%)	13335 (46%)
Number of fully used LUT-FF pairs	105 (22%)	267 (27%)	331 (27%)	523 (26%)	1036 (26%)	2061 (26%)	4111 (26%)
Number of BUFG/BUFGCTRLs	1 (3%)	1 (3%)	1 (3%)	1 (3%)	1 (3%)	1 (3%)	2 (6%)

Table 5.3: Device Utilization Summary of the Faster Montgomery multiplier

The occupied area of the implemented multipliers is previously optimized due to the iterative form that appears in the controlled loop.

From the synthesis results we found that:

The FPGA occupied area increased with the operands bit length because the size of multiplexers, adders and substructures need more number of **fpga** resources (slice registers, slice LUTs and LUT-FF pairs) when the operands bit length increase.

The standard Montgomery multiplier occupied more resources than the faster multiplier, exactly in the number of fully used LUT-FF pairs.

The standard interleaved architecture needs fewer resources than Montgomery multipliers.

The following table shows the maximum clock frequency of every multiplier for different word bit length

Bit length	64	128	160	256	512	1024	2048
Standard Montgomery	142.025 MHz	101.012 MHz	88.061 MHz	63.598 MHz	36.436 MHz	19.708 MHz	10.274 MHz
Standard Interleaved	110.578 MHz	87.616 MHz	78.177 MHz	58.227 MHz	34.695 MHz	19.187 MHz	10.130 MHz
Faster Montgomery	534.623MHz	509.568 MHz	480.634 MHz	480.215 MHz	466.331 MHz	434.518 MHz	419.569 MHz

Table 5.4: The multipliers maximum clock frequency

In our implementation we gave importance to the speed (represented in clock frequency); by pushing the synthesis tool to optimize speed only.

From the *table 5.4* we found that:

The maximum clock frequency decrease in function of the operands bit length, because the delay in combinatory logic become important and the routing path became longer when the occupied FPGA area rise.

By comparing the clock frequency of the standard Montgomery architecture with the standard interleaved one we Remarque that the first is faster. But in the same time we have to take in consideration that one interleaved multiplication is equivalent to two Montgomery multiplications, and then we conclude that the interleaved architecture is better to do modular multiplication.

By comparing the faster Montgomery architecture with the standard one, we found that there is an important difference in clock frequency due to the optimizations done on the first multiplier, and this is the advantage of using carry save adder that can accelerate addition by reducing the carry propagation in each loop iteration to only one carry propagation in the last iteration.

By comparing the faster Montgomery architecture with the standard interleaved. We found that one modular multiplication by using the first multiplier is at least faster than two or three multiplications of the second architecture, then the faster Montgomery multiplier is more efficient than the standard interleaved to implement cryptographic IP core for RSA, DH or ECC protocols.

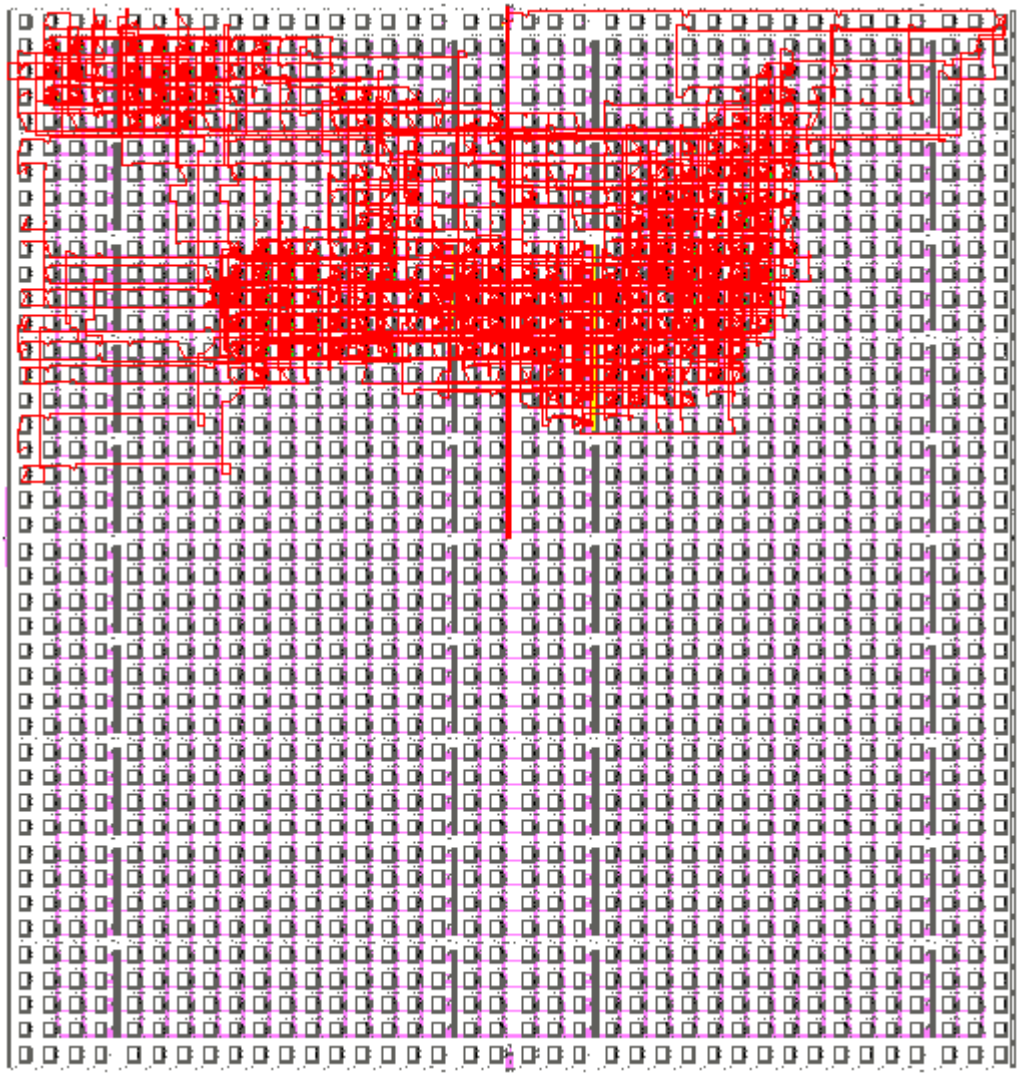

FIG5.9: Routing scheme for 16 bit interleaved multiplier created by FPGA Editor

5.7 Device configuration:

After generating a programming file, we configure our device by connection the Virtex-II V2MB1000 Development Kit by JTAG cable and load the VHDL code represented in the programming file (.bit).

5.8 In Circuit verification results:

The circuit debugging was done in different steps:

1- Use the Chip Scope Pro core Inserter to insert the following cores:

- ✓ ICON (Integrated Controller) to manage the communication between Chip Scope pro software and chip scope integrated cores.
- ✓ ILA core (Integrated Logic Analyzer): for data acquisition from the implemented design to Chip Scope pro software.

2- Connect the ILA with signals that we want to verify.
3- Rerun the implementation steps and configure the device again.
4- Launch the Chip Scope pro analyzer software, initialize the JTAG chain and initialize the trigger to the event that start data acquisition (for example we can use the rising edge of Reset signal)
5- Launch the acquisition and play the event that satisfy the trigger condition.

The following figure represents the Chip Scope Pro chronogram debugging of the implemented architectures of modular multiplication.

(a) Is the Chip Scope Pro chronogram debugging of faster Montgomery multiplier
(b) Is the Chip Scope Pro chronogram debugging of standard Montgomery multiplier
(c) Is the Chip Scope Pro chronogram debugging of standard interleaved multiplier

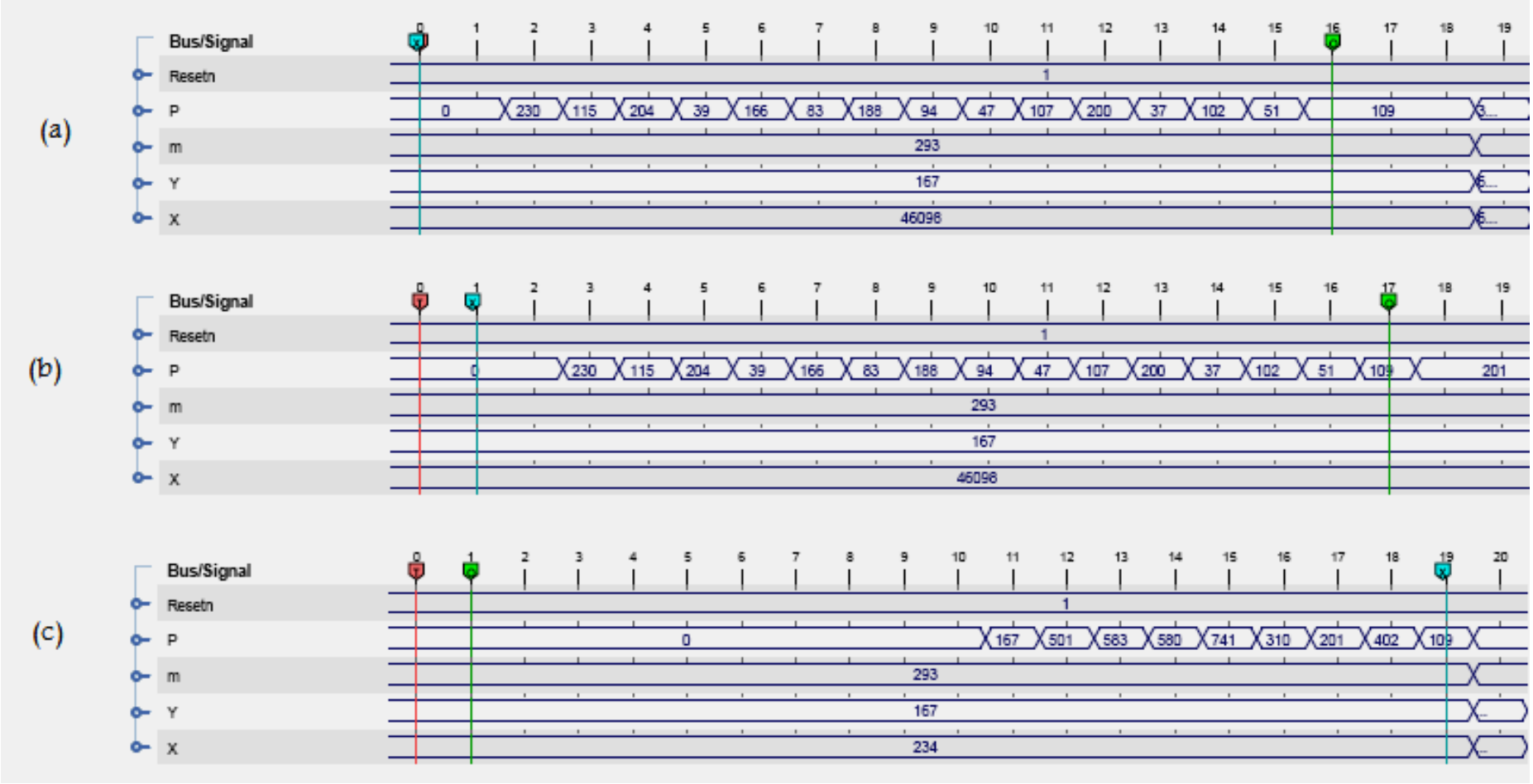

Fig5.10: Chip Scope Pro chronogram debugging of the implemented architectures

5.9 Conclusion

After the implementation results of three architectures we found that the faster Montgomery multiplier represents the best solution for modular multiplication IP core, due to the operating clock frequency,

The core operation in modular multiplication is addition; Addition in high bit length operands became very slow due to the long path of carry propagation , each optimization on the used adder will increase the speed of multiplier, and we showed the difference between the full adders and the carry save adders in the standard and the faster Montgomery multipliers.

The interleaved multiplier is more efficient than the standard Montgomery multiplier, because one operation of the first is equivalent to two operations of the second.

The faster Montgomery multiplier is more efficient than the interleaved multiplier due to the big difference in clock frequency.

The in circuit verification was done by ChipScope Pro and we showed the circuit debugging. By using Virtex-II V2MB1000 Development Kit.

1. Conclusions:

The implemented IP cores for modular multiplication are dedicated to cryptography applications that use the modular exponentiation as a core operation of data encryption as the RSA, DH and ECC (Elliptic Curve Cryptography). The modular exponentiation can be done easily by successive arithmetic operations of modular multiplication.

The hardware solution is mandatory in real time cryptographic applications; we use it as an accelerating core or cryptographic Co Processor; especially when dealing with big numbers (1024 or 2048 bit).

- ✓ In our cores of modular multiplication we proved the efficiency of hardware solution represented in configurable platforms as the FPGAs
- ✓ All our cores represent techniques to compute the modular multiplication without division, and this is the first advantage because the division of big numbers represents a big problem in hardware because it uses a big area.
- ✓ In the implemented cores we used iterative architectures that lead to a minimization of the occupied area of FPGA.
- ✓ In the case of the interleaved multiplier the design output gives us directly the final result; it means that we need one operation only.
- ✓ In the case of the standard and faster Montgomery multipliers for each operation we need two modular multiplications, the first to transform one of the input operands to the Montgomery domain, and the second to obtain the final result.
- ✓ In the implementation we didn't make importance to the area and we push the synthesis tool to optimize the speed, because the area of architectures is optimized previously in the level of algorithms.
- ✓ The synthesis results showed that we can operate at high clock frequency, especially for the faster Montgomery architecture (as 419.569 MHz for a word of 2048 bit)
- ✓ The carry save adder (CSA) prove its importance especially in high word bit length (512, 1024 and 2048) by reducing the carry propagation at each iteration, to only one propagation after the last iteration.
- ✓ The in circuit verification (in circuit debugging) prove that the multipliers work well in the hard level but in the same time we were limited by the resources of VIRTEX II.

2.Perspectives:

The perspective of this work is to use these cores to implement modular exponentiation cores to use them as a core accelerator module in cryptographic systems. It can be used in RSA, DH or ECC algorithms. Then we use the modular exponentiation core to implement RSA coprocessor for encryption/decryption by using keys of 1024 bit and 2048 bit.

References

[1]: Veloni A, Pedroni « Circuit design with VHDL ».

[2]: J pierre, Gery J, Gustavo D « synthesis of arithmetic circuit FPGA, ASICs and Embedded Systems »

[3]: Pong p. Chu «RTL Hardwar design VHDL».

[4]: www.xilinx.com

[5]: F. Mayer-Lindenberg « Methods in Hardware/Software System Design ».

[6]: Harris.X « Cryptography book ».

[7]: Raoel Ashruf « The AES targeted on the MOLEN processor »

[8]: BAHAGIAN B « A Hardware Implementation of Rivest –Shamir - Adleman Co-Processor for Resource Constrained Embedded Systems » Universiti Teknologi Malaysia may 2006

[9]: N MENTENS « Secure and Efficient Coprocessor Design for Cryptographic Applications on FPGAs. » June 2007

[10]: NIST. NIST Special Publication 800-67: « recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, » 2004.

[11]:A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. « Handbook of Applied Cryptography ». CRC Press, 1997.

[12]: Van Oorschot A.J. Menezes and S. A. Vanstone, «Handbook of applied cryptography», CRC Press, Waterloo, Ontario, Canada, 2001.

[13]: William Stallings. « Cryptography and Network Security Principles and Practices, Fourth Edition » 2005

[14]: Jing Lu,Wan Qian « Implementing a 1024- bit RSA on FPGA» 6/6/2010

[15]: Shafi G, Mihir B « Lecture Notes on Cryptography».

[16]: Kiomichi.A,Takazu .S,Shinji.Miura « Overview of Elliptic curve Cryptography».

[17]: Cetin Kaya .C «RSA Hardware Implementation».

[18]: David Narh.A « Efficient Hardware Architectures for Modular Multiplication ».

[19]: David Narh.A, Viktor.B, Manfred.S «Efficient Hardware Architectures for Modular Multiplication on FPGAs ».

[20]: V. Bunimov, M. Schimmler, «Area-Time Optimal Modular Multiplication »,
Embedded Cryptographic Hardware: Methodologie and Architectures, 2004,
ISBN: 1 – 59454 – 012 - 8.

Appendix

Virtex-II™ V2MB1000

Development Board

Overview

The Virtex-II V2MB1000 Development Kit provides a complete solution for developing designs and applications based on the Xilinx Virtex-II FPGA family. The kit bundles an expandable Virtex- II based system board with a power supply, user guide and reference designs. Also available from Memec Design, optional P160 expansion modules enable further application specific prototyping and testing. Xilinx ISE software and a JTAG cable are available as kit options.

The Virtex-II system board utilizes the 1-million gate Xilinx Virtex-II device (XC2V1000-4FG456C) in the 456 fine-pitch ball grid array package. The high gate density and large number of user I/Os allows complete system solutions to be implemented in the advanced platform FPGA. The system board includes a 16M x 16 DDR memory, two clock sources, RS-232 port, and additional support circuits. An LVDS interface is provided with a 16-bit transmit and 16-bit receive port plus clock, status, and control signals for each. The board also supports the Memec Design P160 expansion module standard, allowing application specific expansion modules to be easily added.

The Virtex-II FPGA family has the advanced features needed to fit demanding, high-performance applications. The Virtex-II Development Kit provides an excellent platform to explore these features so that you can quickly and effectively meet your time-to-market requirements.

The Virtex-II System Board

The Memec Design Virtex-II System Board provides the FPGA, support circuits and the P160 expansion slot for application specific add-on cards. Figure 1 shows a picture of the board and its features.

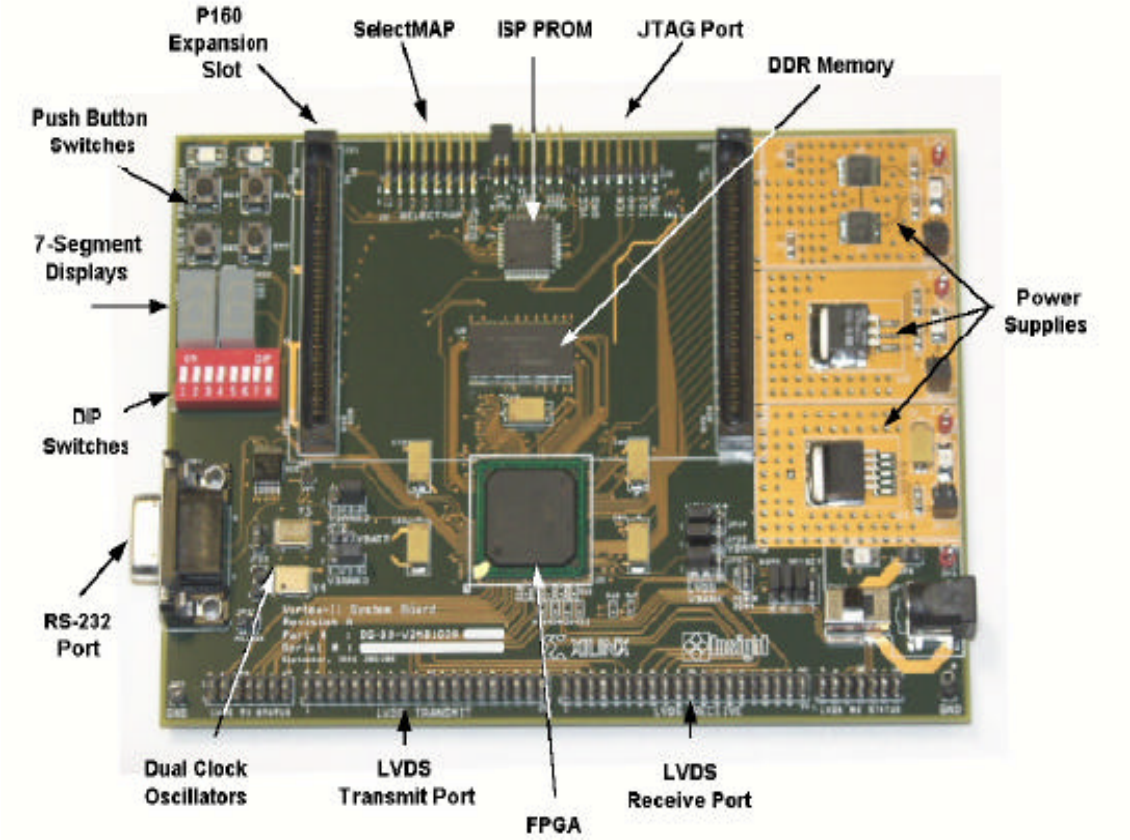

Figure 1: Virtex-II System Board

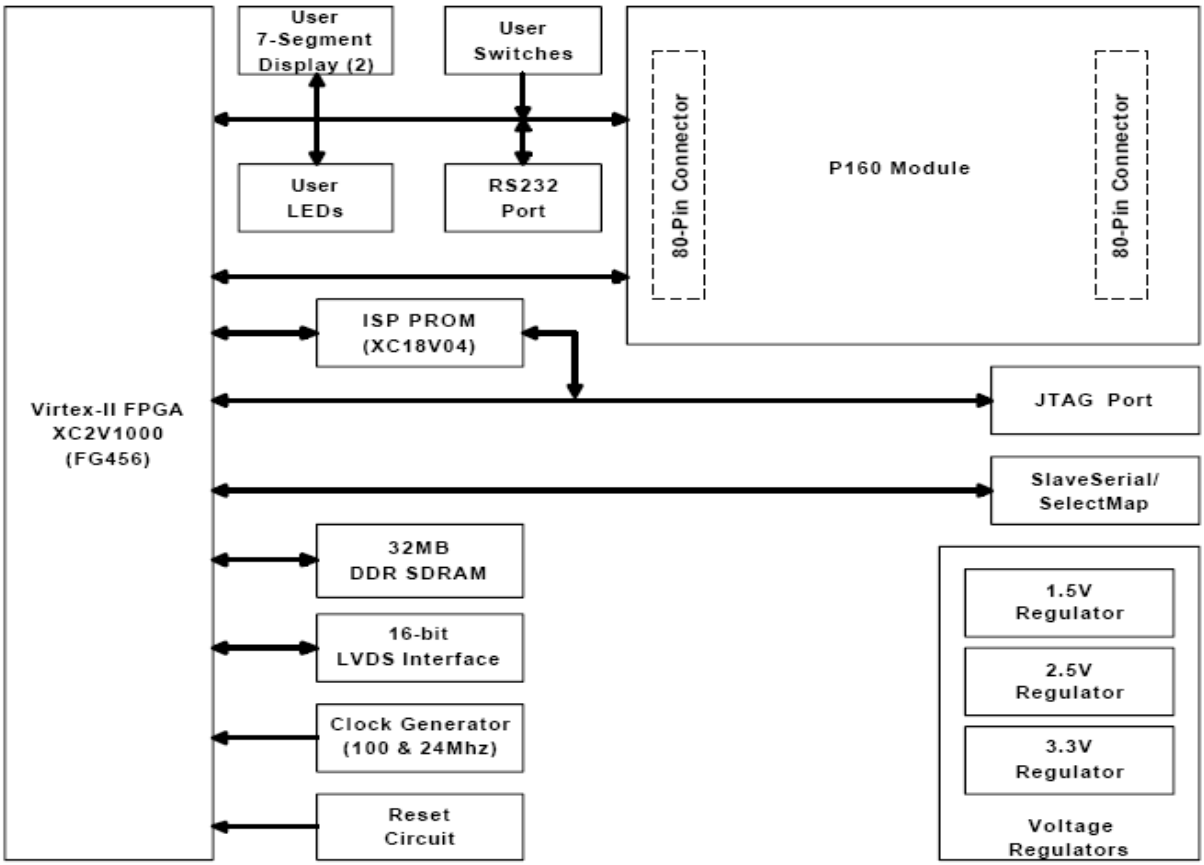

Figure 2: Virtex-II System Board Block Diagram

Virtex-II Device

The Virtex-II board utilizes the Xilinx Virtex-II XC2V1000-4FG456C. The Virtex -II family is a platform FPGA developed for high performance, low to high-density designs utilizing IP cores and customized modules. The Virtex-II family delivers complete solutions for telecommunication, wireless, networking, video, and DSP applications. The performance and density of the Virtex-II family along with its supported I/O standards such as LVDS, PCI, and DDR enables FPGA designers to meet the design requirements of the next generation Networking and telecommunication applications. The Xilinx Virtex-II FPGA along with its supporting I/O devices on this development board, will assist FPGA designers to prototype high-performance memory and I/O interfaces such as complete high-performance Packet Over SONET Level 4 (PL4) over a 16-bit LVDS bus, high speed DDR memory interface, and a variety of other I/O interfaces via the on-board I/O module.

User 7-Segment Display

The Virtex-II system board utilizes two common-cathode 7-segment LED displays that can be used during the test and debugging phase of a design. The user can turn a given segment on by driving the associated signal high. The following figure shows the user 7-segment display interface to the Virtex-II FPGA.

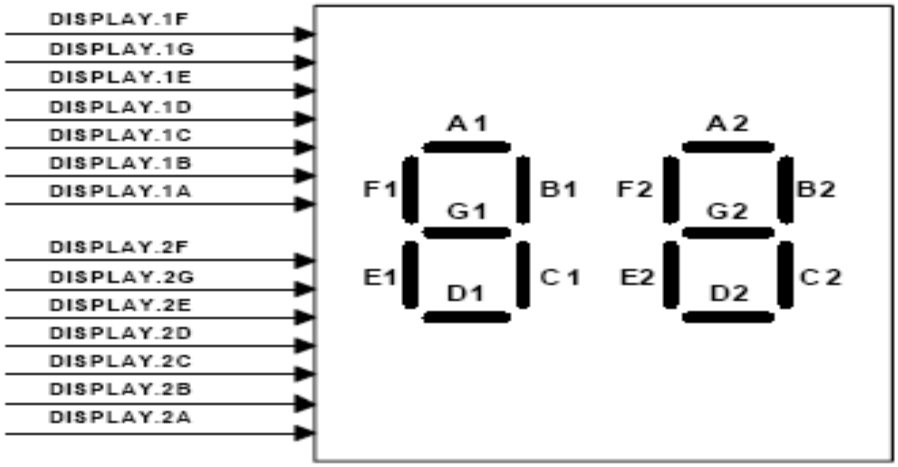

Figure 3: 7-Segment LED Display Interface

LVDS Port Signal Descriptions

The following table shows the LVDS port signal descriptions and the port signal assignments to the Virtex-II FPGA

Signal Name	Virtex-II Pin #	J4 Pin #	Description
LVDSOUT1N	H2	1	Negative Data Transmit Bit 1
LVDSOUT1P	H1	2	Positive Data Transmit Bit 1
LVDSOUT2N	J2	3	Negative Data Transmit Bit 2
LVDSOUT2P	J1	4	Positive Data Transmit Bit 2
LVDSOUT3N	K2	5	Negative Data Transmit Bit 3
LVDSOUT3P	K1	6	Positive Data Transmit Bit 3
LVDSOUT4N	E4	7	Negative Data Transmit Bit 4
LVDSOUT4P	E3	8	Positive Data Transmit Bit 4
GND	NA	9	Ground
GND	NA	10	Ground
LVDSOUT5N	F4	11	Negative Data Transmit Bit 5
LVDSOUT5P	F3	12	Positive Data Transmit Bit 5
LVDSOUT6N	G4	13	Negative Data Transmit Bit 6
LVDSOUT6P	G3	14	Positive Data Transmit Bit 6
LVDSOUT7N	H4	15	Negative Data Transmit Bit 7
LVDSOUT7P	H3	16	Positive Data Transmit Bit 7
LVDSOUT8N	J4	17	Negative Data Transmit Bit 8
LVDSOUT8P	J3	18	Positive Data Transmit Bit 8
GND	NA	19	Ground
GND	NA	20	Ground
LVDSOUT9N	K4	21	Negative Data Transmit Bit 9
LVDSOUT9P	K3	22	Positive Data Transmit Bit 9
LVDSOUT10N	L3	23	Negative Data Transmit Bit 10
LVDSOUT10P	L2	24	Positive Data Transmit Bit 10
LVDSOUT11N	L5	25	Negative Data Transmit Bit 11
LVDSOUT11P	L4	26	Positive Data Transmit Bit 11
LVDSOUT12N	E6	27	Negative Data Transmit Bit 12
LVDSOUT12P	E5	28	Positive Data Transmit Bit 12
GND	NA	29	Ground
GND	NA	30	Ground
LVDSOUT13N	F5	31	Negative Data Transmit Bit 13
LVDSOUT13P	G5	32	Positive Data Transmit Bit 13
LVDSOUT14N	H5	33	Negative Data Transmit Bit 14
LVDSOUT14P	J6	34	Positive Data Transmit Bit 14
LVDSOUT15N	J5	35	Negative Data Transmit Bit 15
LVDSOUT15P	K5	36	Positive Data Transmit Bit 15
LVDSOUT16N	K6	37	Negative Data Transmit Bit 16
LVDSOUT16P	L6	38	Positive Data Transmit Bit 16
GND	NA	39	Ground
GND	NA	40	Ground

Table 1: LVDS Transmit Port Signal Descriptions

Signal Name	Virtex-II Pin #	J6 Pin #	Description
GND	NA	1	Ground
GND	NA	2	Ground
LVDSIN1P	M2	3	Positive Data Receive Bit 1
LVDSIN1N	M1	4	Negative Data Receive Bit 1
LVDSIN2P	N2	5	Positive Data Receive Bit 2
LVDSIN2N	N1	6	Negative Data Receive Bit 2
LVDSIN3P	P2	7	Positive Data Receive Bit 3
LVDSIN3N	P1	8	Negative Data Receive Bit 3
LVDSIN4P	R2	9	Positive Data Receive Bit 4
LVDSIN4N	R1	10	Negative Data Receive Bit 4
GND	NA	11	Ground
GND	NA	12	Ground
LVDSIN5P	T2	13	Positive Data Receive Bit 5
LVDSIN5N	T1	14	Negative Data Receive Bit 5
LVDSIN6P	U2	15	Positive Data Receive Bit 6
LVDSIN6N	U1	16	Negative Data Receive Bit 6
LVDSIN7P	V2	17	Positive Data Receive Bit 7
LVDSIN7N	V1	18	Negative Data Receive Bit 7
LVDSIN8P	W2	19	Positive Data Receive Bit 8
LVDSIN8N	W1	20	Negative Data Receive Bit 8
GND	NA	21	Ground
GND	NA	22	Ground
LVDSIN9P	Y2	23	Positive Data Receive Bit 9
LVDSIN9N	Y1	24	Negative Data Receive Bit 9
LVDSIN10P	M6	25	Positive Data Receive Bit 10
LVDSIN10N	M5	26	Negative Data Receive Bit 10
LVDSIN11P	M4	27	Positive Data Receive Bit 11
LVDSIN11N	M3	28	Negative Data Receive Bit 11
LVDSIN12P	N4	29	Positive Data Receive Bit 12
LVDSIN12N	N3	30	Negative Data Receive Bit 12
GND	NA	31	Ground
GND	NA	32	Ground
LVDSIN13P	P4	33	Positive Data Receive Bit 13
LVDSIN13N	P3	34	Negative Data Receive Bit 13
LVDSIN14P	R4	35	Positive Data Receive Bit 14
LVDSIN14N	R3	36	Negative Data Receive Bit 14
LVDSIN15P	T4	37	Positive Data Receive Bit 15
LVDSIN15N	T3	38	Negative Data Receive Bit 15
LVDSIN16P	U4	39	Positive Data Receive Bit 16
LVDSIN16N	U3	40	Negative Data Receive Bit 16

Table 2: LVDS Receive Port Signal Descriptions

Signal Name	Virtex-II Pin #	J7 Pin #	Description
LVDSOUTCLKP	C1	1	Positive Transmit Clock
LVDSOUTCLKN	C2	2	Negative Transmit Clock
GND	NA	3	Ground
GND	NA	4	Ground
LVDSOUTSTATCLKP	D1	5	Positive Transmit Status Clock
LVDSOUTSTATCLKN	D2	6	Negative Transmit Status Clock
GND	NA	7	Ground
GND	NA	8	Ground
LVDSOUTSTAT1P	E1	9	Positive Transmit Status1
LVDSOUTSTAT1N	E2	10	Negative Transmit Status1
LVDSOUTSTAT2P	F1	11	Positive Transmit Status2
LVDSOUTSTAT2N	F2	12	Negative Transmit Status2
LVDSOUTCTRLP	G1	13	Positive Transmit Control
LVDSOUCTRLN	G2	14	Negative Transmit Control

Table 3: LVDS Transmit Control Port Signal Descriptions

Signal Name	Virtex-II Pin #	J8 Pin #	Description
LVDSINCTRLN	V3	1	Negative Receive Control
LVDSINCTRLP	V4	2	Positive Receive Control
LVDSINSTAT2N	N5	3	Negative Receive Status2
LVDSINSTAT2P	N6	4	Positive Receive Status2
LVDSINSTAT1N	P5	5	Negative Receive Status1
LVDSINSTAT1P	P6	6	Positive Receive Status1
GND	NA	7	Ground
GND	NA	8	Ground
LVDSINSTATCLKN	W11	9	Negative Receive Status Clock
LVDSINSTATCLKP	V11	10	Positive Receive Status Clock
GND	NA	11	Ground
GND	NA	12	Ground
LVDSINCLKN	AA11	13	Negative Receive Clock
LVDSINCLKP	Y11	14	Positive Receive Clock

Table 4: LVDS Receive Control Port Signal Descriptions

Design Download

The Virtex-II development board supports multiple methods of configuring the Virtex-II FPGA. The JTAG port on the Virtex-II development board can be used to directly configure the Virtex-II FPGA, or to program the on-board XC18V04 ISP PROM. Once the ISP PROM is programmed, it can be used to configure the Virtex -II FPGA. The Select Map/Slave Serial port on this development board can also be used to configure the Virtex-II FPGA. The following figure shows the setup for all Virtex-II FPGA configuration modes that are supported on the Virtex-II development board.

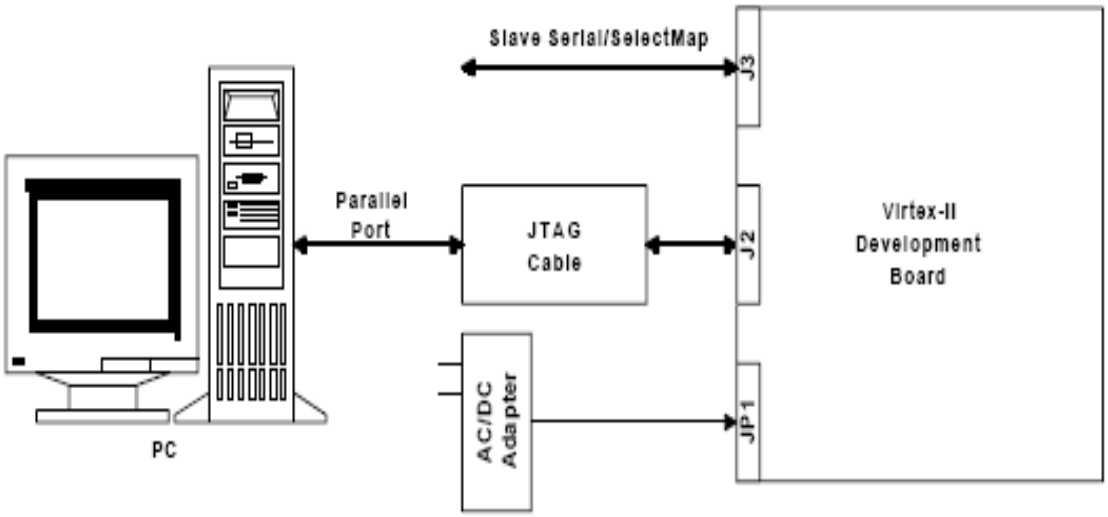

Figure 4 : Download Setup

Virtex-II Configuration Mode Select

The following table shows the Virtex-II Configuration Mode Select jumper settings. The jumper position 7-8 (M3) is connected to the HSWAP_EN pin of the Virtex-II FPGA. When this jumper is closed, the Virtex-II internal I/O pull-ups are enabled during the configuration.

Mode	PC Pull-up	J1			
		1-2 (M0)	3-4 (M1)	5-6 (M2)	7-8 (M3)
Master Serial	No	Closed	Closed	Closed	Closed
Master Serial	Yes	Closed	Closed	Closed	Open
Slave Serial	No	Open	Open	Open	Closed
Slave Serial	Yes	Open	Open	Open	Open
Master SelectMap	No	Closed	Open	Open	Closed
Master SelectMap	Yes	Closed	Open	Open	Open
Slave SelectMap	No	Open	Open	Closed	Closed
Slave SelectMap	Yes	Open	Open	Closed	Open
JTAG	No	Open	Closed	Open	Closed
JTAG	Yes	Open	Closed	Open	Open

Table 5 : Virtex-II Configuration Mode Select

JTAG Interface

The J2 JTAG connector on the Virtex-II development board can be used to configure the Virtex –II or to program the on-board XC18V04 ISP PROM. The Memec Design JTAG cable is connected to the Virtex-II development board via J2 at one end and to the PC parallel port at the other end.

Configuring the Virtex-II FPGA

When the JTAG port is used to configure the Virtex-II FPGA, the following steps must be taken:

- Using Table 5 set the Configuration Mode of the Virtex-II FPGA to JTAG Mode.
- Use the Xilinx JTAG programmer utility (iMPACT) to load the design bit file into the Virtex-II FPGA. You will need to associate the ISP PROM with either a dummy .mcs file, or a .bsd file to allow the JTAG programming software to pass data through the ISP PROM.

Programming the XC18V04 ISP PROM

When the JTAG port is used to program the ISP PROM, the following steps must be taken:

- Using Table 5 set the Configuration Mode of the Virtex-II FPGA to Master Serial or Master SelectMap Mode.
- Use the Xilinx JTAG programmer utility (iMPACT) to load the design mcs file into the ISPPROM. You will need to associate the FPGA with either a dummy .bit file or a .bsd file to allow the JTAG programming software to pass data through the FPGA.
- Upon programming of the 18V04 ISP PROM, the on-board PROGn push button switch (SW2) is used to initiate the Virtex-II FPGA configuration.