REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA



Faculté de Technologie

Mémoire de fin d'études de Master

Filière : Télécommunications.

Spécialité : Systèmes des télécommunications.

Study and implementation of a crypto system on an FPGA platform

---

Préparé et présenté par :

-BELGACEMI Mohamed Amine Hichem

-BENSABER Sabrina

Encadreur : Dr. BENGHERBIA Billel

Co-encadreur : Dr. BOUHEDDA Mounir

Année universitaire 2020-2021

THANKS AND DEDICATIONS

Throughout the writing of this dissertation we have received a great deal of support and assistance.

We would first like to thank our supervisor, dr.Bengharbia whose expertise was invaluable in formulating the research questions and methodology.

Your insightful feedback helped us to sharpen our thinking and brought our work to a higher level.

In addition, the support, encouragements and comfort of our dear parents are forever appreciated beside their wise counsel and sympathetic ear. You are always there for us.

Finally, we could not have completed this dissertation without the support of our friends, who provided stimulating discussions as well as happy distractions to rest our mind outside of our research.

We dedicate this modest work to our dear parents, for their

moral and financial support and for being the light of our life,

who have always encouraged us to finish our studies, hoping

them a long life and may God keep them

- To our promoter **Mr. 'BENGHARBIA.B'**

- To our brothers and our sisters

- To all our dear friends

- To all teachers in the Faculty of Science and Technology

MOHAMED AMINE HICHEM AND SABRINA

ملخص

الهدف من هذا المشروع هو تنفيذ خوارزمية تشفير / فك تشفير جديدة تعتمد على الحمض النووي باستخدام HLS لتجميع الوظائف وإنشاء نوىIP.

لهذا الغرض, تم تطوير خوارزمية على المستوى C حيث تم تحويل بعض الوظائف إلى نوى IP على HLS وتنفيذها على لوحة "FPGA ZYBO"xc7z01clg1-400.

تم التحقق من صحة الحل المقترح على نموذج أولي تم تصنيعه في Vivado 2018.2.

الكلمات المفتاحية: الحمض النووي, النوى IP,C, FPGA ZYBO , Vivado 2018.2.

Abstract

The aim of this project is to implement a new DNA-based encryption/decryption algorithm using HLS to synthesize functions and creating IP cores.

For this purpose, an algorithm is developed on C-Level, where some functions have been transferred to IP cores on HLS and implemented on an FPGA ZYBO board "xc7z010clg400-1". The proposed solution is validated on a prototype made on Vivado 2018.2.

Key words: DNA, HLS, IP cores, C-Level, FPGA ZYBO, Vivado 2018.2.

Résumé

Le but de ce projet est d'implémenter un nouvel algorithme de cryptage/décryptage basé sur l'ADN en utilisant HLS pour synthétiser des fonctions et créer des cœurs IP.

Pour cela, un algorithme est développé au niveau C où certaines fonctions ont été transférées à des cœurs IP sur HLS et implémentées sur une carte FPGA ZYBO "xc7z010clg400-1". La solution proposée est validée sur un prototype réalisé sur Vivado 2018.2.

Mots clé : ADN, HLS, coeurs IP, C, FPGA ZYBO, Vivado 2018.2.

Summary

Summary

THANKS AND DEDICATIONS……I

Abstract……III

List of the tables……VI

List of figures ……VIII

List of abbreviations……XI

General introduction……...1

CHAPTER I: DNA-based cryptography

I.1 Introduction……2

I.2 History of cryptography……2

I.3 Cryptography……5

I.3.1 Definition……5

I.3.2 How the encryption works……..5

I.3.3 Defining Concepts:……6

I.3.4 Principle of cryptography……...6

I.3.5 Types of Cryptography……...7

I.3.6 Strength of crypto-systems……8

I.3.7 Modern cryptography……9

I.4 DNA-based encryption……12

Summary

I.4.1 Biological Study……..13

I.4.2 DNA computing……...19

I.4.3 DNA cryptography…….....20

I.4.4 Molecular storage and calculations……..…21

I.4.5 Digital DNA in cryptography……...…21

I.5 Proposed method for DNA-based cryptography …….…...22

I.5.1 An illustration of the method……26

I.6 Conclusion……...……28

CHAPTER II: FPGA and HLS

II.1 Introduction……29

II.2 FPGA……..……29

II.2.1 History…….……29

II.2.2 Definition…….…...30

II.2.3 FPGA Architecture…….……31

II.2.4 Internal structure……..32

II.2.5 FPGA Applications…….38

II.2.6 FPGA Development tools……...39

II.3 Presentation of the zynq 7000 card……40

II.3.1 Zynq-7000 Family Description……...40

II.3.2 Zynq-7000……...40

Summary

II.4 High-Level Synthesis…….41

II.4.1 Definition……41

II.4.2 High-Level Synthesis for FPGAs……42

II.4.3 HLS Benefits……...42

II.5 Conclusion…….....43

Chapter III: Experiments and results

III.1 Introduction: ……....44

III.2 Presentation of the development tools: ……44

III.2.1 Vivado…….....44

III.2.2 Vivado HLS……...44

III.2.3 C Language……....44

III.2.4 ZYBO Board……..45

III.3 Vivado HLS design flow…….45

III.3.1 Design step…….....46

III.3.2 Develop C Algorithm..……...46

III.3.3 Developing custom IP core using HLS……..46

III.3.4 Verify C Algorithm……50

III.4 Using developed IP core in vivado design suite……...54

III.4.1 Create a New Project with Included Developed IP Core..…….……54

III.4.2 Block diagram with added necessary IPs and generate Bitstream…….……55

Summary

III.4.3 Result obtained……...56

III.5 Conclusion…….....60

General conclusion……....61

Annex …….....64

List of the tables

Table I. 1 Binary coded DNA…….. 22

Table I. 2 Translation table of alphabets, numbers and punctuation marks in DNA bases…..24

Table I. 3 the process of coding two nitrogenous bases into English letters……25

Table I. 4 Conversion from alphabet to binary…….26

Table I.5 Encoding of binary numbers in nitrogenous bases……...26

Table I. 6 the process of coding nitrogenous bases into English lettres (for the example taken)…….. ...28

List of figures

Figure I. 1 A scytale..... 3

Figure I. 2 Machine Enigma.....4

Figure I. 3 Symmetric Cryptography. 10

Figure I. 4 Asymmetric Cryptography..... 12

Figure I. 5 Double helical structure of DNA..... 14

Figure I. 6 Nucleotide structure.....15

Figure I. 7 DNA strands formed by nucleotide and hydrogen bonds. 15

Figure I. 8 DNA structure..... 15

Figure I. 9 Chromosome 17

Figure I. 10 pyrimidines and purines.....18

Figure I. 11 Structure of Nucleotide. 19

Figure I. 12 nucleoside and nucleotide 19

Figure I.13 genetic code.....23

Figure II.1 A PLA schematic paradigm [1]..... .31

Figure II. 2 Xilinx FPGA board.....32

Figure II. 3 Internal architecture of the Xilinx manufactured FPGA. 33

Figure II. 4 A logic cell is the foundation of an FPGA[12]. 34

Figure II. 5 Generic input / output block [12] .....,36

List of figure

Figure II. 6 Clock distribution networks. Left: H-structure; right: backbone and rib structure38

Figure II. 7 CMOS inverter.....39

Figure II. 8 Architectural Overview..... 42

Figure III. 1 Vivado HLS design flow 46

Figure III. 2 Design Steps 47

Figure III. 3 Create new vivado project dialog box 48

Figure III. 4 Add/remove design file dialog box 49

Figure III. 5 Add/remove c testbench file 49

Figure III. 6 the Solution Configuration window 50

Figure III. 7 Select board dialog box 51

Figure III. 8 Simulation configuration dialog box.....51

Figure III. 9 the Run C Simulation button 52

Figure III. 10 the csim folder..... 53

Figure III. 11 the Run C synthesis button 53

Figure III. 12 choose a source file 54

Figure III. 13 the decryption directive for the function convert_binaire_caractere 55

Figure III. 14 the export RTL button 55

Figure III. 15 Choosing the ZYBO board 56

Figure III. 16 Repository add to the project 56

Figure III. 17 add the IP and the ZYNQ7000 Processing System..... 57

Figure III. 18 architecture the design on vivado suite 57

Figure III. 19 Encryption_Decryption function drivers. 58

Figure III. 20 tab to program FPGA board..... 59

Figure III. 21 Encryption results..... 60

Figure III. 22 decryption results 61

List of abbreviations

ADP: Adenosine DiPhosphate.

AES: Advanced Encryption Standard.

ARM: Advanced RISC Machines.

ASIC: Application Specific Integrated Circuit.

ASSPs: Application Specific Standard Products.

ASCII: American Standard Code For Information Interchange.

ATP: Adénosine TriPhosphate.

CBC: Cipher Block Chaining CFB: Cipher FeedBack.

CMOS: Metal-Oxide Semiconductor.

DNA: Deoxyribose Nucleic Acid.

DES: Data Encryption Standard.

DSP: digital signal processing.

ECB: Electronic Code Book.

ECC: Error Correcting Code.

ELF: Extended Language Facility.

EEPROM: electrically erasable programmable read-only memory.

FPGAs: Field Programmable Gate Arrays.

GSM: Global System for Mobile.

HLL: high level language.

HSDPA: High Speed Downlink Packet Access.

List of abbreviations

IC: Integrated Circuit.

IDE: Integrated Development Environment.

IEC: International Electrotechnical Commission.

IP**:** Internet Protocol.

ISO: Organisation internationale de normalisation.

I/O: input and output.

JTAG: Joint Test Action Group.

LUT: lookup table.

MAC: Medium access control.

MD5: Message Digest 5.

MOC: Metal-Oxide Semiconductor.

NAS: Network Attached Storage.

NIST: National Institute of Standards and Technology.

NRE: Non-Recurring Engineering.

OFB: Output FeedBack.

OTP: One Time Pad.

PS: Processing System.

PL: Programable Logic.

PLAs: Programmable Logic Arrays.

PLD: Programmable Logic Devices.

PROM: programmable read-only memory.

PCs: personal computers.

PGP: Pretty Good Privacy.

List of abbreviations

PROM: programmable read-only memory.

PCR: polymerase chain reaction.

PCI: Payment Card Industry.

Pmod: Peripheral Module.

RAM: Random Access Memory.

RSA: Rivest Shamir Adelman.

RNA: ribonucleic acid.

RTL: register transfer level.

SAN:Storage Area Network.

SRAM: Static Random Access Memory.

SoC: System-On-Chip.

SDK: Software Development Kit.

UART: Universal Asynchronous Receiver Transmitter.

WiMAX: World Interoperability for Microwave Access.

WCDMA: Wideband Code Division Multiple Acces.

General introduction

Since life started, peoples used to talk to each other for clarified ideas and share what's in mind and communicate together, communication is the way to exchange knowledge and information, but it was only for real-life, but scientists and researchers made life smaller and found a way to communicate from a far distance using voice, texts and even a video call, technology have made all that possible, but also become more dangerous for the government, military or any other important information to share, because of the hackers or enemies if they become able to obtain that information, share them or edit the real meaning from the transmitter to the recipient that's why another field has been opened which is the Information Security to hide the information for encrypt it and be able to decrypted and save it from attacks.

Encryption is the method by which information is converted into secret code that hides the information's true meaning. And there are many ways to encrypt information. In this dissertation we are going to introduce DNA encryption which is one of the most complicated methods to save and convert information because of its proprieties.

It will be realized on HLS High-Level Synthesis which enables the user to transform (synthesize) C/C++/SystemC code into VHDL that's means enable us to program FPGA in high-level languages Focusing on C and run on FPGA ZYBO board.

Chapter I: DNA-based cryptography

I.1 Introduction

Information has become a very important strategic resource and it has become imperative to protect this information with the continuous development of protection methods, given the increasing pace of Internet technology. For this reason, several cryptographic systems have been developed to meet these needs.

Cryptography, the art and science of preparing coded or protected communications intended to be intelligible only to the person possessing a key, and includes two branches: cryptography and cryptanalysis.

Cryptography refers both to the process or skill of communicating or deciphering secret writings (codes or ciphers) and to the use of codes to convert computerized data so that only a recipient will be able to read it using a key.

Currently, two types of encryption are used together. Asymmetric algorithms for the transfer of cryptographic keys and symmetric algorithms for encrypting the data to be protected.

This work first talks about the history of cryptography. Then, it presents the basic concepts related to cryptography it describes several symmetric and asymmetric algorithms and their implementation.

Recent studies have focused on DNA, and with the increased development of bio-inspiration, there is a promising emerging encryption technique in the field of cryptography called DNA cryptography. The main objective of this method is to encrypt the plaintext and hide it in digital DNA form by exploiting the biological characteristics of DNA especially the complex and long structure of DNA.

I.2 History of cryptography

Human beings have always had the need to hide information. Whether it is a secret that should not be disclosed in his entourage that would compromise individuals or tactical information during the various battles and wars that have marked history.

Concealing information has always been a necessity. Here is a non-exhaustive list of different techniques used over the centuries that mark the evolution of cryptography through the ages.

The first traces of cryptography date back to antiquity, more precisely around the 16th century BC. A potter in Iraq had engraved on a clay table his recipe by deleting consonants and changing the spelling of words. Later, between the tenth and seventh centuries BC, the Greeks used scytales (Figure I.1), a kind of wooden stick. When the transmitter wanted to communicate, he rolled up a leather strip on the scytale and wrote the message. Once the tape was unrolled, the letters were no longer in order and therefore no longer made sense. The only way to understand the message was to roll up the tape on a scythe of the same diameter so that the letters could be aligned correctly.



Figure I.1 A scytale

In the first century BC, Caesar's encryption was introduced. This cipher was used by Julius Caesar to communicate secretly. It is one of the first substitution ciphers. Its principle is simple, it is enough to substitute each character of the original message by another in the alphabet, which is always at a fixed distance. As time goes by, as the methods used previously become easier and easier to understand and to divert, other measures had to be taken in order to be able to prevent the decryption of messages. In the 16th century, the Vigenere cipher made its appearance. It is also a substitution cipher, but it is more advanced than Caesar's cipher. Rather than using a fixed offset, the Vigenere cipher relies on a key that will determine the offset for each character.

Since the last century, cryptography has played a key role in various wars, and the technology began to be used with, for example, radio messages or telegrams. Information could be

intercepted much more easily than in the past. This was the case with the German Foreign Minister Arthur Zimmermann on January 16, 1917. He sent a telegram to the German ambassador in Mexico proposing a plot to keep the United States out of the war. However, the message was intercepted by the British because the Germans used the "diplomatic code 0075" which was a substitution code with 1000 possible substitutions.

After the First World War, the Enigma machine (Figure I.2) was created and the Germans, having understood the importance that sensitive information can have, invested in a more complex military version of this machine. However, although the operation of the machine is complex, Polish researchers have studied the operation of the machine to try to decrypt the messages. Subsequently, the famous British mathematician, cryptologist and computer scientist Alan Turing collaborated in decrypting the messages coded by the machine. Thanks to this, the British were able to decipher messages, which was a serious advantage that allowed them to win the war. It was even estimated that Turing's discovery shortened the Second World War by two years.



Figure I.2 Machine Enigma

In 1977, the DES1 encryption standard was proposed as a standard by the NIST2. It is a 56-bit text block cipher (actually 64, but one bit is used to control parity). It remained as a standard until the end of the 1990s. Indeed, at the end of the 70's, computers were not powerful enough to decipher a message coded with DES, but with the multiplication of their power, the deciphering became possible in a reasonable time. Encrypted information in DES can be decrypted in 30 hours with 1000 PCs in parallel clocked at 1GHz. That's why, at the end of the 90's, the NIST launched a contest to replace DES and define a new encryption standard that would become AES3. The winners were the two Belgian researchers Joan Daemen and Vincent Rijmen with their Rijndael encryption.

At the same time with the DES, more precisely in 1976, the two cryptologists Whitfield Diffie and Martin Hellman published an article "New Directions in Cryptography" which proposes a new way to encrypt data. It is from their article that asymmetric encryption was born with the Diffie-Hellman cipher. Until then, all cryptosystems were symmetric. Although this type of encryption works well, keeping the key secret or simply communicating it can become restrictive. With their system, the secret key problem is no longer a problem because the operation is based on a key that can be known by everyone[2].

I.3 Cryptography

I.3.1 Definition

Is the process of transcribing intelligible information into unintelligible information by the application of secret conventions whose effect is reversible, i.e. transforming, thanks to an encryption key, a plain text into a ciphertext, in such a way that the reverse transformation is only possible with the knowledge of the decryption key. In other words, cryptography consists in the elaboration of encryption/decryption schemes or crypto-systems[3].

I.3.2 How the encryption works

Encryption uses a systematic or step-by-step procedure called an algorithm to convert data or the text of an original message, known as plaintext, into ciphertext.

Cryptographic algorithms normally require a string of characters called a key to encrypt or decrypt data. Those who possess the key and the algorithm can encrypt the plaintext into ciphertext and then decrypt the ciphertext back into plaintext [3].

Cryptologists engage in an unending competition to create stronger cryptographic techniques and to break them. Many recent cryptography techniques are nearly unbreakable even with the most powerful computers. These systems produce ciphertext that appears to be random characters. These systems resist most existing methods for deciphering back into plaintext. The many different types of new cryptosystems use highly complex mathematical language and resist breaking even though cryptologists may know the techniques used in creating them. Pretty Good Privacy (PGP) helps users of e-mail keep their communications private. In this two-key system, also known as the public key system, the computer sending an encrypted message uses a chosen private key that is never shared and so is known only to the sender. All computers authorized to receive and decrypt the message are given the matching public key. This method also establishes who sent the message. If a sending computer first encrypts the

message with the intended receiver's public key and again with the sender's secret, private key, then the receiving computer may decrypt the message, first using its secret key and then the sender's public key. Using this public-key cryptographic method, the sender and receiver are able to authenticate one another as well as protect the secrecy of the message. The use of the keys is done[4].

I.3.3 Defining Concepts:

- Encryption**:** operation which consists in transforming a message by means of an encryption key in order to hide its meaning from all those who are not authorized to know it[5].
- Decryption**:** reverse operation of encryption. Its purpose is to recover the hidden information by means of a decryption key[5].

  On the other hand, is the operation that allows to find the clear message corresponding to an encrypted message without having the decryption key.
- Cryptogram**:** Encrypted message; the legitimate recipient must be able to decrypt the cipher and obtain the clear text.
- Cryptanalysis**:** Cryptanalysis is the art of analyzing cipher text to extract the plain text or key. In other words, cryptanalysis is the opposite of cryptography. Cryptology: It is a mathematical science with two branches: includes both cryptography and cryptanalysis.
- Encoding a message in such a way as to make it secret is called encryption. The reverse method, consisting of finding the original message, is called decryption[6].
- Cryptosystem**:** A set of cryptographic algorithms and all the clear text, encrypted text and possible keys.
- The code: a set of processes and a set of symbols (letters, numbers, signs, etc.) used to replace the words of the message to be encoded[6].
- Key: This is the parameter that is involved and allows encryption and/or decryption operations. In the case of a symmetric algorithm, the key is identical during both operations. In the case of asymmetric algorithms, it differs for both operations.
- Public key**:** part of the certificate to be issued. It is intended for signature verification and message encryption. This is the part that is transmitted to the interlocutors or placed on key servers. The public key is used to encrypt messages to be sent and decrypt the signature of messages received.

I.3.4 Principle of cryptography

I.3.4.1 Kerckhoff's principle

A fundamental principle of cryptography was stated by Kerckhoffs at the end of the nineteenth century[7]. It states that the encryption method used must be "capable of falling into the hands of the enemy without harm". In other words, the security of encryption should not be based on its confidentiality but only on the protection of the key. This principle has several justifications:

- The confidentiality of a secret algorithm is difficult to guarantee. It is usually known to several people and is often distributed in software or hardware devices to users not entitled to secrecy. The confidentiality of the algorithm can succumb to corruption.
- The security of a secret algorithm is difficult to evaluate (many originally secret algorithms have been found to be extremely weak). It is generally accepted that the best guarantee of security of an algorithm is provided by a long period of evaluation by the world cryptographic community.
- A secret algorithm can hide undesirable properties for the end user (e.g. existence of weak keys). It is therefore not suitable if trust in the designer is not established.
- Finally, for the theorist, it is a working hypothesis without which it is impossible to obtain rigorous safety results.

I.3.4.2 Shannon's principle

The principle of Claude Shannon[8] can be considered as a reformulation of Kirchhoff's principle. Thus, Shannon stated that in order to eliminate redundancies in plain text, two techniques were necessary: confusion and diffusion.

- ❖ Confusion: it erases the relationship between plaintext and ciphertext. It avoids the analysis of the cipher text by searching for redundancies and statistical patterns. The simplest way to do this is through substitution such as the Julius Caesar cipher.
- ❖ Diffusion: ideally, the cipher text should look like a random string of letters typed by a chimpanzee. The cryptographer's goal is to eliminate any clues in the ciphertext that would help the cryptanalyst find the plaintext. This is done by eliminating the statistical relationships between the ciphertext and the corresponding plaintext. Diffusion combines transposition and substitution and diffuses the statistical structure of the plaintext among the ciphertext**[5].**

I.3.5 Types of Cryptography

In general there are three types of cryptography[1]:

I.3.5.1 Symmetric Key Cryptography:

It is an encryption system where the sender and receiver of message use a single common key to encrypt and decrypt messages. Symmetric Key Systems are faster and simpler but the problem is that sender and receiver must somehow exchange key in a secure manner. The most popular symmetric-key cryptography system is Data Encryption System (DES).

I.3.5.2 Hash functions

There is no usage of any key in this algorithm. A hash value with fixed length is calculated as per the plain text which makes it impossible for contents of plain text to be recovered. Many operating systems use hash functions to encrypt passwords.

I.3.5.3 Asummetric key cryptography

Under this system a pair of keys is used to encrypt and decrypt information. A public key is used for encryption and a private key is used for decryption. Public key and Private Key are different. Even if the public key is known by everyone the intended receiver can only decode it because he alone knows the private key.

I.3.6 Strength of crypto-systems

The strength of the encryption method comes from the algorithm, the secrecy of the key, the length of the key, and the initialization vectors.

The strength is correlated to the amount of processing required, the power and the time it takes to break the key or determine its value. Breaking a key can be accomplished by a brute force attack, which attempts by any means possible to the value of the key until the resulting plaintext is recovered. Depending on the algorithm and the length of the key, this can be a very easy task or a practically impossible one.

An algorithm contains a keyspace, which is a range of values that can be used to construct a key. The key is composed of random values in the key space. The larger the key space, the more values that are available that can be used to represent different keys, and the more random the keys are, the harder it is for intruders to figure them out. The encryption algorithm must use a keyspace and choose the values that the keys take on, very randomly. If a smaller key space was used, it would increase the chance of an attacker finding the keyvalue, and thus decrypting the protected information.

Another name for the strength of cryptography is the 'work factor', which is an estimate of the effort it would take for an attacker to penetrate an encryption method. This effort is evaluated by cryptanalysis.

Cryptanalysis is the study of the probabilities of success of possible attacks on cryptosystems in order to detect their possible weaknesses[9]. One of the main goals of cryptanalysis is to test whether an adversary can decrypt the plaintext or recover the secret key. To do this, the cryptanalyst puts himself in the adversary's shoes. Cryptography and cryptanalysis are two fields of study evolving constantly and in parallel. Indeed, new crypto-systems, always more complex, are developed to replace those which were broken by cryptanalysis and new cryptanalysis techniques are invented to test these new crypto-systems. Since then, many possible attacks against cryptosystems have been identified, so that it is difficult to establish an exhaustive list. However, two classes of attacks can be distinguished: active attacks and passive attacks.

Inactive attacks, the opponent acts on the information. He alters data integrity, authentication and confidentiality. He can try to alter the transmission of the message on the channel, for example, by modifying the message (deletion, addition, modification of the message sequences), by delaying (or preventing) its transmission, by repeating its sending.

In passive attacks, the adversary observes information on the channel without modifying it. He tries to recover information about the cryptosystem without altering it, such as the message, the secret key, etc. In this case, the adversary affects the confidentiality of the data[5].

I.3.7 Modern cryptography

In today's information society, the use of cryptology has become commonplace. We find it every day with credit cards, cell phones, Internet or even transport tickets.
Modern cryptology is the study of the methods that ensure integrity, authenticity and confidentiality in information and communication systems. Today, it also covers all the computer processes that must resist adversaries[10].

Cryptology is divided into two sub-disciplines:

- Cryptography which proposes methods to ensure these services.
- Cryptanalysis that looks for flaws in the proposed mechanisms[11].

To ensure the objectives of modern cryptography we can use key-based algorithms. These algorithms are defined by several types of modern cryptography, we distinguish two approaches:

- Symmetric cryptography.
- Asymmetric cryptography[10].

I.3.7.1 Symmetric Cryptography

Symmetric encryption (also called private key or secret key encryption), this type is based on the use of a key to encrypt and decrypt messages. The security of this solution is based on the fact that the key is known only by the sender and the receiver of the message (Figure I.3).

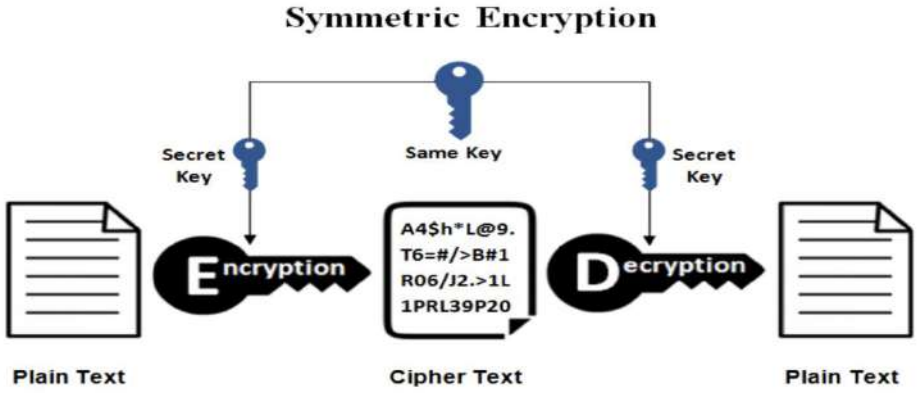


Figure I.3 Symmetric Cryptography

Symmetric algorithms are faster and easily implemented in hardware. They are simple substitution and transposition operations and are best suited for use on the wired Internet and in mobility. However, these systems are not entirely adequate to solve all security problems.

The encryption and decryption process depends on the same secret key shared between the sender and the receiver. The sender and receiver must therefore agree on the secret key before they can communicate in a secure manner. A secure channel for key exchange is difficult to establish.

Symmetric cryptosystems have a key management problem. In fact, when a large number of people wish to communicate together, the number of keys increases significantly and the situation becomes impracticable. For n communication participants, we will need: n * (n-1) / 2 registered secret keys.

For better security, it was destroyed and reset after each conversation[10].

Followings are the symmetric cryptography schemes.

- DES: DES stands for Data Encryption Standard. DES introduced in early 1970 at IBM. The early design of DES is based on Horst Feistel. DES is a symmetric cryptographic algorithm used for encryption and decryption of message. In DES, only one secret key is used for both encryption and decryption. The key size of DES is 56-bit. To perform encryption/decryption, the sender and receiver must have the same key. The DES performs encryption on a block of 64-bit. The DES algorithm is most widely used in many applications and some popular use in military, commercial, and security of communication system, same as DES but key size is different from DES. The key size of 3DES is 168 bits. The 3DES algorithm performs operation three times on each block of data. It is slower than DES[12].
- AES: AES stands for Advanced Encryption Standard which is the advancement of 3DES algorithm. It was introduced in 1997 by the NIST (National Institute of Standards and Technology). Basically, AES is based on the Rijndael cipher developed by two cryptographers, Joan Daemen and Vincent Rijmen. AES is different from DES and 3DES due to variables key sizes such as 128, 192, and 256 bits. Same as DES and 3DES, AES also performs encryption on blocks which are 128-bit. AES algorithm use in small devices for encrypting a message to send over a network. Some other applications are monetary transaction and security applications[12].

Symmetric encryption works in two different ways:

- Block encryption: encryption is performed on blocks of bits.
- Encryption by stream (or stream): the encryption is carried out continuously, bit by bit.

I.3.7.2 Asymmetric Cryptography

Asymmetric cryptography as shown on the (Figure I.4) is also in the category of cryptography schemes. Unlike symmetric cryptography, two keys are used: one is public and second is private. The public key is shared by anyone in the cryptographic system while the private key is kept secret by authenticated user. Followings are the asymmetric cryptography algorithms.

- RSA: RSA stands for Rivest, Shamir and Adleman who introduced the RSA algorithm in 1977[13]. RSA is an asymmetric cryptographic algorithm which is also used for encryption and decryption of the message. RSA is widely used in transferring of keys over an insecure channel. Due to asymmetric nature, there are two keys used in the

algorithm. One is public key and second is a private key. The public key is openly accessible to everyone in the cryptosystem and the private key is kept secret by authorized person. RSA provides confidentiality, integrity, authenticity, and nonrepudiation of data[12]. RSA is more commonly used in electronic industry for online money transfer. In future, RSA can be used in Java cards[14].

- ElGamal: ElGamal algorithm was introduced in 1985 by Taher ElGamal. ElGamal is an asymmetric key encryption algorithm that is based on the Diffie-Helman key exchange as an alternative to RSA for public key encryption. ElGamal is also used in digital signature generation algorithm called ElGamal signature scheme. A homomorphic algorithm named paillier used for its semantic security[12].
- ECC: ECC stands for Elliptic Curve Cryptography. ECC introduced in 1985 by Neal Koblitz and Victor S. Miller. ECC lies in the category of the asymmetric scheme that is based on elliptic curves. The applications of ECC are encryption, digital signatures and pseudo-random generators[15].

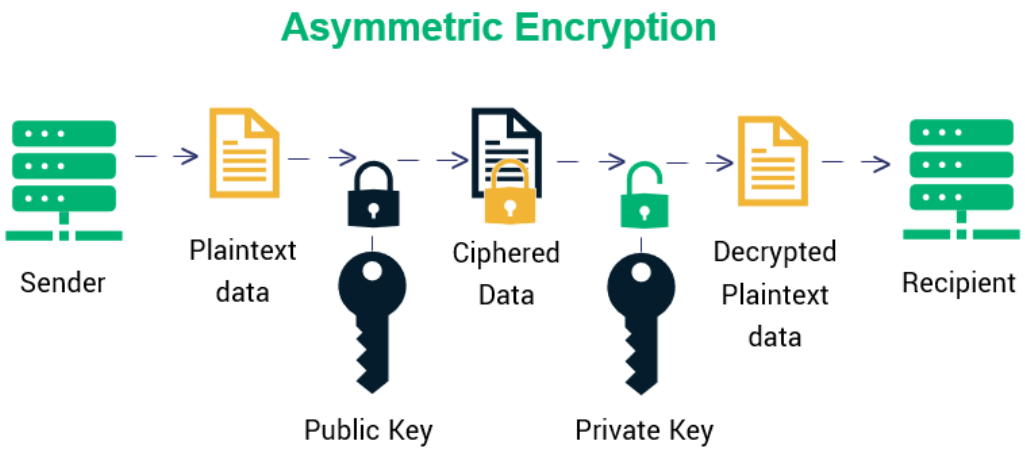


Figure I.4 Asymmetric Cryptography

I.4 DNA-based encryption

With the growing pace of the Internet and network technology day by day, security threats are also increasing for the users, due to lot of information flow on the network. There are various adversaries who always try to break into the system in order to steal crucial information or to destroy the integrity of data. So information security becomes necessity for modern computing systems. There are some sectors like government, banks, military can't afford any leaks to their secret data.

The most secure and presently used technique is the modern methods of cryptography which involves many mathematical computations and two types of keys, the public and private keys. Nowadays, there is another newly emerging cryptographic technique in the field of cryptography called DNA cryptography. The main objective of this method is to encrypt the plaintext and hide it in the DNA digital form. DNA cryptography enables the confidentiality of data higher than the modern methods with the use of OTP keys and theirs size. Also it is believed that in DNA cryptography the key can be generated for a huge length of data compared to the modern methods in which keys are generated only for a smaller length of the data. Also, with the breaking up of modern cryptographic algorithm like DES and MD5 new methods of information security are needed to protect our data. The concept of DNA computing plays an important role in the field of computer security which is assumed to be a more powerful and unbreakable cryptographic algorithm nowadays.

I.4.1 Biological Study

I.4.1.1 DNA

DNA stands for Deoxyribonucleic acid which store genetic information of the entire living organism ranging from human being to small viruses. It is also called an information carrier and consists of long polymer of small units called nucleotides. Further, nucleotides consist of three components: Nitrogenous base, five Carbon sugar and Phosphate group. The Nitrogenous base consists of four bases: Adenine, Thymine, Cytosine and Guanine (A, T, C, G), all the complex information about organism are stored with the combination of these bases. Adenine and Guanine are called purines, whereas Thymine and Cytosine are called pyrimidines. DNA is a double helix structure as shown on the figure below (Figure I.5).

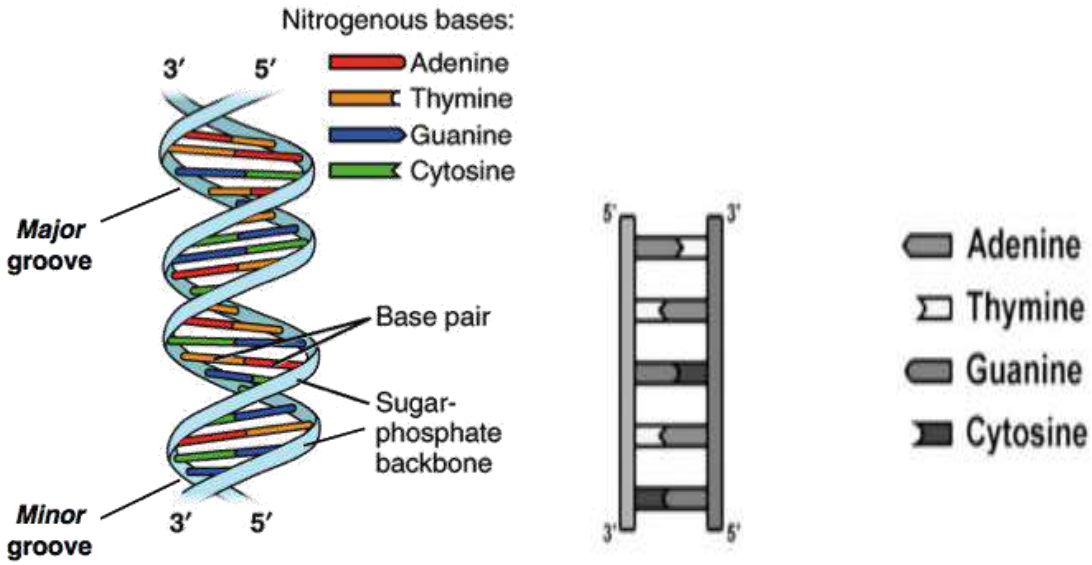


Figure I. 5 Double helical structure of DNA

DNA double-helical structure was discovered by the two Nobel laureate Watson and Crick and therefore it is also called a Watson-Crick complementary structure, where A and T form a hydrogen bond with each other, whereas C and G forms bond with one another. In this structure of DNA both the strands are antiparallel to each other, means if one strand starts from 3′ to 5′, then another strand is from 5′ to 3′[16].

I.4.1.2 DNA structure

Deoxyribose nucleic acid (DNA) has a helical shape, composed of two long strands of nucleotides. A nucleotide has one of four bases: A - adenine, G - guanine, C - cytosine or T - thymine, deoxyribose sugar and a phosphate group (Figure I.6). the sugars and the phosphates allow the nucleotides to bind in a single strand of DNA[17].

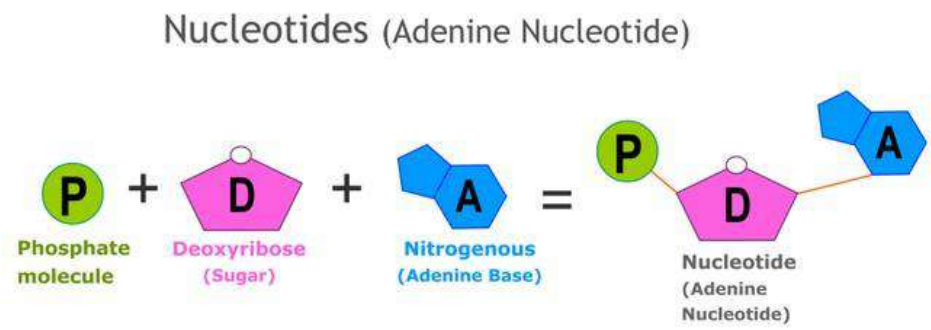


Figure I.6 Nucleotide structure

Hydrogen bonds hold 2 strands together and create a double-stranded DNA (Figure I.7).

Hydrogen bonds last only between complementary pairs: A-T and C-G

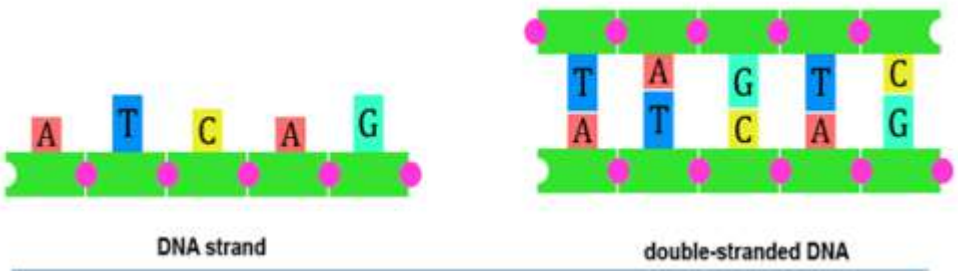


Figure I.7 DNA strands formed by nucleotide and hydrogen bonds

The three-dimensional structure of DNA was discovered in 1953 by Watson and Crick [18]. DNA strands twist together to form a helix (Figure I. 8).

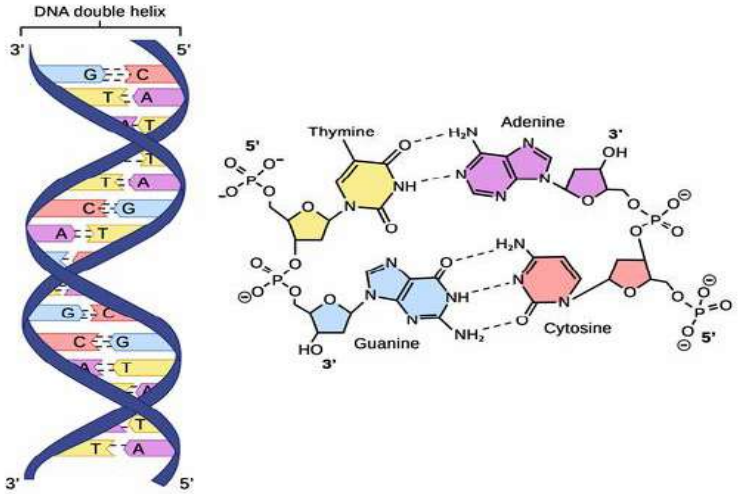


Figure I. 8 DNA structure

I.4.1.3 Biological Operations

The biological concepts of DNA cryptography are the main activity of the cell in our body which is incorporated in the field of DNA computing.

I.4.1.3.1 Hybridization

The double-stranded DNA is formed by combining single-stranded DNA sequences, In this process A always pairs with T and G always pairs with C according to Watson crick complement condition Consider an example of two single-stranded sequences 5'TGTGCGCG 3'and 5'GCGCACAC3'will combine to form a Double-stranded sequence in a double helix structure. 5'TGTGCGCG 3' 3' ACACGCGC 5'

I.4.1.3.2 Denaturation

The double-stranded DNA molecule can be heated up to 90 degrees Celsius, so that it is resolved into two single-stranded DNA.The biological concepts of DNA cryptography are the main activity of the cell in our body which is incorporated in the field of DNA computing[19].

I.4.1.4 Definition of some concepts:

I.4.1.4 .1 Achromosome

A chromosome (Figure I.9) is a microscopic cellular structure representing the physical support of genes and genetic information, always made of DNA, and often of proteins. Chromosomes exist in the cells of all living beings, in variable numbers, specific to each species.

Chromosomes are the hereditary material of cells. They carry the genetic information and the genes that are transmitted from generation to generation. Each gene occupies a precise location on a given chromosome: this is its locus.
The same gene will always be located on the same locus for all individuals of a given species.

A single chromosome is said to carry an average of 3000 genes. These coding sequences seem to occupy only 30% of the chromosomes. The remaining sequences correspond to responsible for regulating gene expression, as well as areas of unknown function[20].

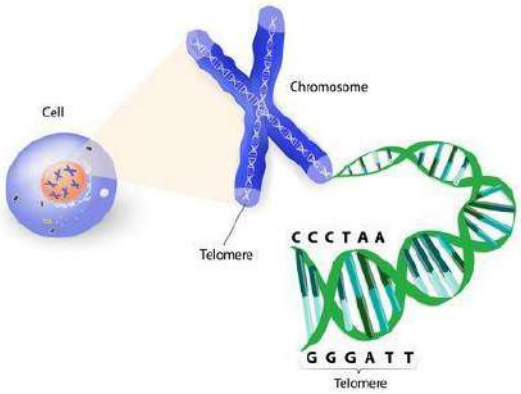


Figure I.9 Chromosome

I.4.1.4.2 Nitrogenous bases:

A nitrogenous base is also called a nucleobase. A nitrogenous base qualifies a nitrogenous heterocyclic body, an organic compound having basic properties. In the cells, the nitrogenous bases are constitutive of the nucleic acids. It is the element carrying the information of the nucleotides. DNA is made up of the following 4 bases: A = adenine; G = guanine; C = cytosine;T=thymine.

A rephrasing gives: DNA contains two purine bases, adenine and guanine, and two pyrimidine bases, cytosine and thymine (Figure I.10); if we compare the DNA molecule to a ladder, the nitrogenous bases constitute the rungs. The complementary bonds are realized by a different number of hydrogen bonds: 2 between adenine and thymine, 3 between cytosine and guanine, which determines a difference in the strength of these bonds[20].

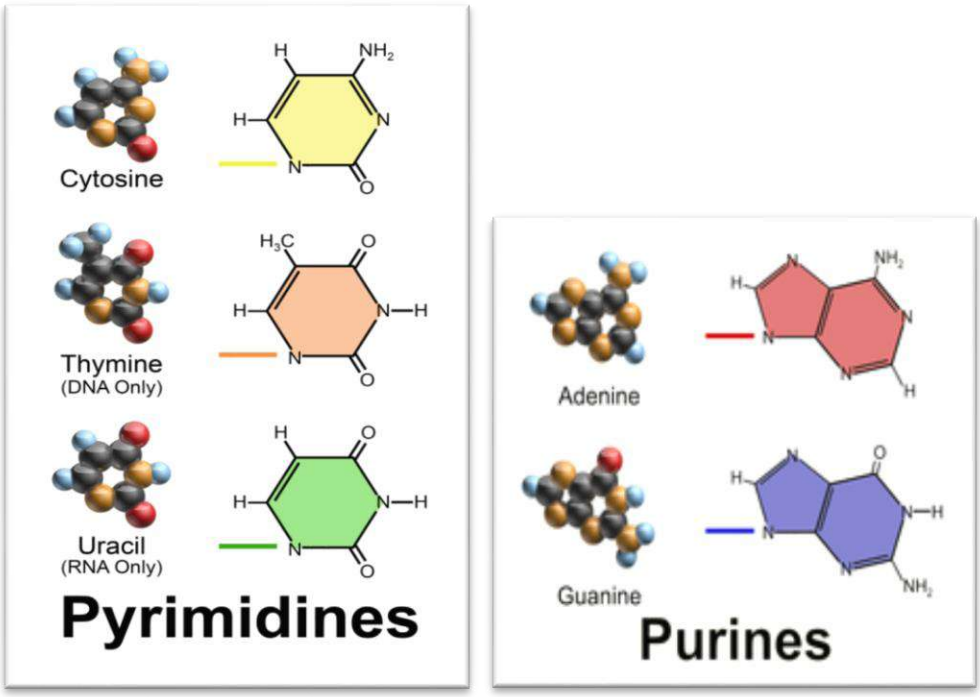


Figure I. 10 pyrimidines and purines

I.4.1.4.3 The nucleotide:

A nucleotide (Figure I.11) is a molecule formed by a nucleoside linked to a phosphate, i.e. a nucleoside monophosphate. The nucleic acids are polynucleotides (polymers of nucleotides)[20].

The sequence of nucleotides determines the succession of bases in the nucleic acid, which constitutes the genetic message. In DNA, the bases are involved in the hydrogen bonds responsible for the formation of the double helix (two bonds between adenine and thymine, and three between guanine and cytosine). In RNA, the OH group located on the ribose can be involved in biological molecular reactions (ribozyme). In addition to this role of formation of more complex molecules, nucleotides are also carriers of energy. Thus, ATP, or adenosine triphosphate, is the energy molecule par excellence, generated by metabolic pathways (glycolysis). ATP can then be used (hydrolyzed to ADP or adenosine diphosphate, loss of a phosphate) by enzymes to perform reactions in the cell[21].

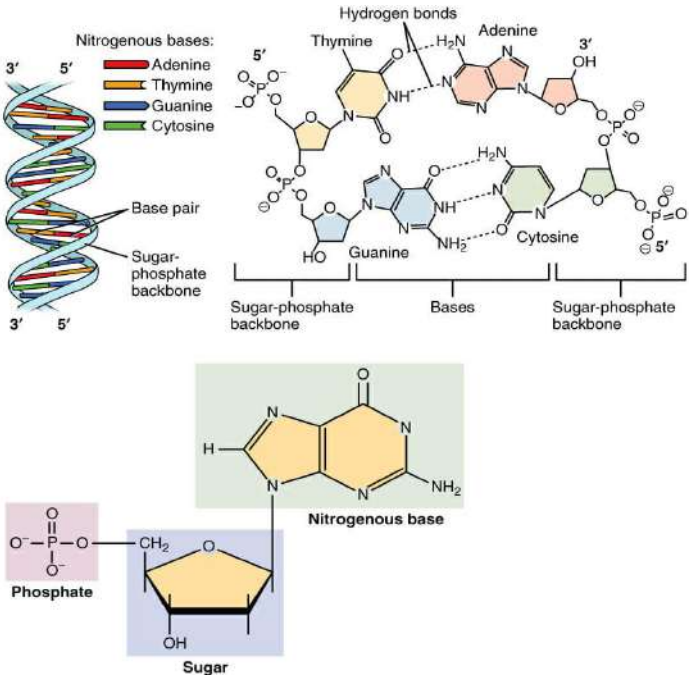


Figure I.11 Structure of Nucleotide

I.4.1.4.4 The nucleoside:

A nucleoside is the combination of a nitrogenous base and a sugar. In other words, it is a nucleotide without a phosphate group[20] as shown on the(Figure I.12).

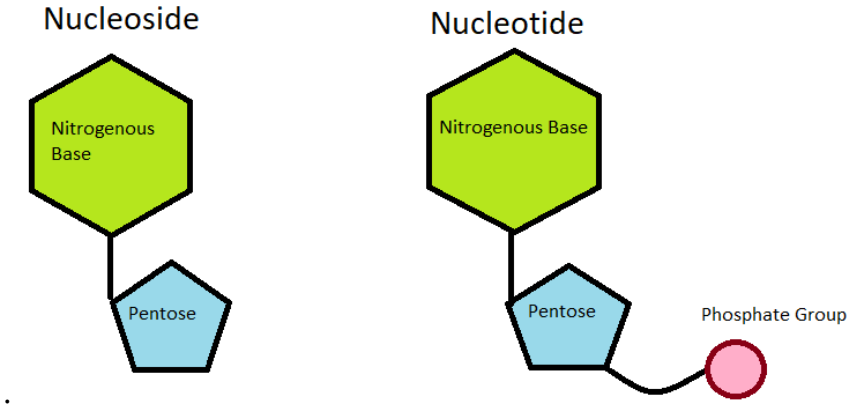


Figure I.12 nucleoside and nucleotide

I.4.2 DNA computing

The computation using DNA sequences is called DNA Computing. The information can be encoded in a DNA sequence and it can be computed. Due to the vast parallelism of DNA computing, DNA itself acts as a microprocessor in which the silicon processors are replaced in the future. As DNA stores a huge amount of information and perform immense parallel

computation, it can process a large number of problems in a fraction of a second. Each DNA molecule or group of molecules could be treated as a processor that can store billion times larger data than our personal computer perform parallel computation at the same time. DNA and RNA are alluring media to store data that exceeds the storage capacity of electronic, physical and optical media. A single molecule can also extend to store large memory and build a storage device that is much times faster than a traditional computer. It can be used to search for possible solutions simultaneously which is represented by DNA strands. By using the parallelism concept, it can be used in the field of encryption and decryption by generating public and private keys.DNA can be embedded with a computer chip and it can be used as a bio chip. DNA computation requires only less energy which can be automated, as it is the nature of DNA strands. The DNA computer hardware and software can all be made up of bimolecular where it does not require power efficiency. Adleman assigned a graph of seven cities to different sequences of DNA which is 20 molecules long, then dropped the sequences into a heat of millions of more sequences of DNA which is naturally bonded with the cities. It generated a random path of thousand sequences like the computer which can be used to break the codes by filter random numbers[19].

I.4.3 DNA cryptography

According to the Watson, The DNA strands can be useful to encode information. In encryption method, the data has been developed through DNA.As the DNA cryptography is the emerging topic only few algorithms have been proposed and it is highly far away from the real time implementation. Even though the DNA cryptography is effective but it is not as much effective than traditional cryptography, But it can be combined with existing cryptography to give hybrid security. This encryption methodology been used to hide the original data which also includes the set of methodologies to store and encrypt the message in terms of DNA sequences. The advantage of DNA encryption methodology is that it addresses the set of extended ASCIIs through which all kind of digitized information can be encrypted. The DNA sequences can be synthesized and merged with denatured DNA or dummy DNA strands of the DNA database[19].

There are different DNA cryptography techniques that have been developed Some of the DNA techniques that are widely used and developed recently are listed:

- Polymerase Chain Reaction is a molecular technique for in vitro gene amplification that produces large amounts of DNA or RNA sequences from small amounts of nucleic

acids. This is a recent method, dating back to the mid-1980s, and discovered by Kary Mullis. This discovery earned him the Nobel Prize in Chemistry in 1993. For a specific and scarce segment of DNA (either because the sample is too small, too damaged, or because the DNA of a component is mixed with other DNA), PCR will make it possible to find the sequence of the desired gene, a tiny fraction of the total DNA, and to duplicate it millions of times, in an exponential way, following a principle of denaturation of the DNA strands, hybridization of specific primers and synthesis (reconstitution of the complementary strands by the polymerase). Once there is a sufficient quantity, the targeted DNA sequence can be studied.

- Bimolecular DNA-based cryptographic design the bimolecular cryptographic system scrambles data into DNA code using oligonucleotide sequences. It uses the One Time Pad (OTP) cryptographic technique which is based on the unbreakable principle. The actual cryptographic scheme experience follows OTPs that have a limitation in transmission on convention electronic media due to the size of the OTP. The DNA strands are very neat because the storage media and a small amount of DNA is enough for a large amount of OTP. We use One Pad encryption, which uses a codebook to scramble the plaintext portion of the message. Text message. The codebook used here is a random codebook, meaning that a secret code is used only once for encryption and decryption, not repeatedly. The OTP of the plaintext message is distributed to the sender and receiver in advance[20].

I.4.4 Molecular storage and calculations

The biological environment of DNA molecules can be seen from the point of view of storage force material or computational. The human genome represents about 0.72 GB of data in only about 3.5 pictograms. This means that a large amount of information can be stored in a space invisible to the naked eye. Different concealment techniques have been proposed to explore a molecular environment. A successful experiment of DNA shorthand is presented in. DNA is presented in[22]. It consists of hiding a message coded by DNA in the background of other DNA sequences. Various techniques propose to introduce watermarks in living DNA. A watermark can be introduced into the coding regions of the DNA by using codon redundancy and in this way no functional modifications appear.

Watermarking information can also be introduced into non-coding regions of the DNA. To store data in DNA molecules, they must be encoded in DNA alphabet. In [23] a mapping method is presented where each letter of the English alphabet (A - Z), digits (0 to 9), and

some punctuation marks have been encoded into 3 DNA letters, such as: Q -AAC, or 9 - GCG. Considering that all numerical information is coded in binary, a simple method is to map the binary alphabet to DNA (Table I.1) [24]. Using this mapping method, any numerical information can be easily transformed into the DNA sequence.

Table I. 1 Binary coded DNA

ADN	Code binaire
A	00
C	01
G	10
T	11

I.4.5 Digital DNA in cryptography

The interest in discovering the exact sequence of DNA inside different living organisms brought an enormous amount of financial, technological and human resources to this subject. In 1985, the Human Genome Project was started. It was completed in 2003 thanks to the efforts of various international organizations. The basic procedure is DNA sequencing Data sharing was the basic principle of this project: "The adoption of data release and sharing was one of the major achievements of the Human Genome Project".

There is currently a wide variety of online sites with free access, genetic databases and analysis tools. A link to a complete list of genetic databases, organizations and tools is given in. There is a large amount of genomic and protein sequences available from different organisms. Two of the proposed ciphers in this thesis use genetic databases in cryptography.

The first method uses the DNA sequence for the substitution of values in plain text. The cipher text will contain the positions in the genetic sequence from which to read the plaintext values.

In the other method, a DNA sequence is transformed into a bitstream. Then, a bitwise XOR operation is performed with it and a plain text bitstream. The both methods use DNA sequences as a secret key in symmetric encryption [24].

I.5 Proposed method for DNA-based cryptography

Every three nitrogenous bases are called codons, and since there are 4 nucleotides involved in the synthesis of DNA (A, T, C, G) there are 64 ‘different’ codons that can be obtained from these bases. As shown on Figure I.13.

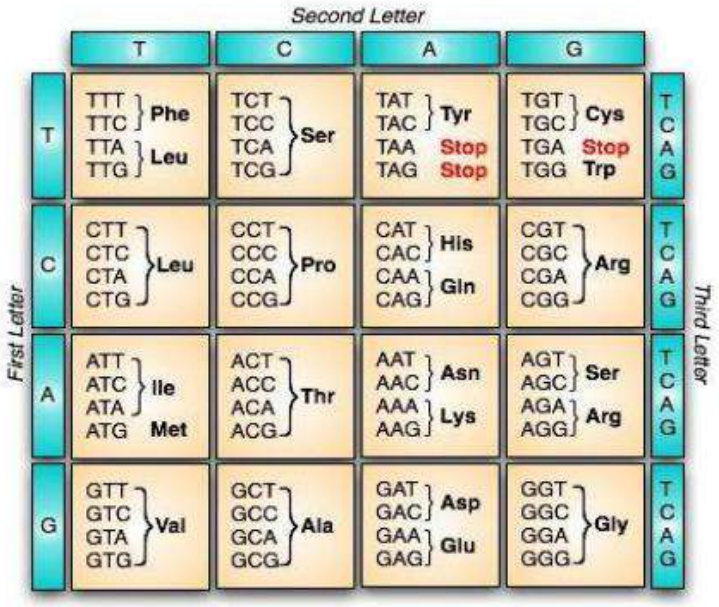


Figure I.13 Genetic code

The plain English text consists of a variety of sequences of the 26 characters (A-Z). Plain text characters are encoded into codons so that each character symbolizes a specific codon depending on the character's location within the clear text as the character located in an individual location (indicated by the small character.) symbolizes a codon that differs from the same character encoding but is located in an even position (indicated by the large character) within the clear text character string as shown on Table I.2.

Table I. 2 Translation table of alphabets, numbers and punctuation marks in DNA bases

Codons	Capital Lettre	Codons	Capital Lettre	Codons	Small Letter	Codons	Small Letter
TCT	N	TTG	A	TTA	n	TTC	A
TAT	O	TCG	B	TCA	o	TCC	B
TGG	P	TGC	C	TGT	p	TAC	C
CTG	Q	CTA	D	CTC	q	CTT	D
CAT	R	CCG	E	CCA	r	CCT	E
CGT	S	CAG	F	CAA	s	CAC	F
ATT	T	CGG	G	CGA	t	CGC	G
ACT	U	ATG	H	ATA	u	ATC	H
AAT	V	ACG	I	ACA	v	ACC	I
AGC	W	AGT	J	AAG	w	AAC	J
GTC	X	GTT	K	AGG	x	AGA	K
GCC	Y	GCT	L	GTG	y	GTA	L
GAC	Z	GAT	M	GCG	z	GCA	M

The transmitter and the recipient agree on a specific sequence of DNA. This information is available in databases at international centers specializing in genetic engineering and genetic studies, as well as on websites such as (NCBI [25] EMBL [26] GENBANK [27]), Whereas this agreed series forms part of the secret key, A process is underway to encode the nitrogenous bases of the DNA sequences (the first represents the clear text after encoding it into codons and the second the agreed DNA string) is done. The encoding product of the two rules is a letter from the English letters. As we note in table I.3, there is a 16 probability of encoding the two rules.

Table I. 3 the process of coding two nitrogenous bases into English letters

Encoded character	The DNA chain agreed between the two parties	DNA string resulting from clear text encoding	Encoded character	The DNA chain agreed between the two parties	DNA string resulting from clear text encoding
D	T	A	B	T	T
I	G	A	H	G	T
R	A	A	K	A	T
P	C	A	M	C	T
X	T	C	S	T	G
W	G	C	V	G	G
Z	A	C	Y	A	G
F	C	C	E	C	G

After you get the string of letters, you turn those letters into the binary system, depending on their sequence within the English alphabet (Table I.4), and since there are 26 'English' letters, then each character is coded five units in the binary system. An alternate encryption method is then used to rearrange the sequence of binary numbers.

Table I. 4 Conversion from alphabet to binary

Binary Code	Alphabet Sequence	Letter	Binary Code	Alphabet Sequence	Letter
10011	19	S	11010	26	Z
01011	11	K	00100	4	D
00100	4	D	01001	9	I
00010	2	B	11000	24	X
10011	19	S	10111	23	W
10110	22	V	00101	5	E
00110	6	F	01000	8	H
00100	4	D	01011	11	K
10000	16	P	10010	18	R
01101	13	M	11010	26	Z
00100	4	D	01011	11	K
01000	8	H	10010	18	R
00010	2	B	10010	18	R
10111	23	W	01011	11	K
01001	9	I	00110	6	F
01000	8	H	10111	23	W
11010	26	Z	00110	6	E
01011	11	K	00101	5	E

The resulting binary series following the substitution denotes nitrogenous bases based on (TableI.5), Another alternative encoding method is used to rearrange the resulting nitrogen rules and thus obtain a random series of nitrogenous rules that represent the encoding text.

Table I.6 Encoding of binary numbers in nitrogenous bases

ADN	Code binaire
A	00
C	01
G	10
T	11

I.5.1 An illustration of the method

To clarify how the proposed method works, a specific text will be encoded, Decoding is done in the same steps but in reverse.

Encryption is performed in the following steps:

1. Reading the clear text and suppose it is "send help soon".
2. Reading the DNA bar agreed between sender and recipient, We assume the agreement for bacteria (E. coli) is taken from EBI.com where the readable tape is:

G1=ATGTGCGAAAAAAACGCCTATTTGCTCCTGTGGGAAGGACCATGAACTA AGGCGCCT…

3. Encoding clear text characters to codons, From Table I.7, As shown:

Letters	s	e	n	d	h	e	l	p	s	o	o	n
letter position	o	e	o	e	o	e	o	e	o	e	o	e
Codons	CAA	CCG	TTA	CTA	ATC	CCG	GTA	TGG	CAA	TAT	TCA	TCT

From the encoding process you can see that the letter N in the individual location is different in its encoding from the letter N in the even position as well as the letter O whereas the letters S and E are in the same location so they have the same codon, Previously we found that the adoption of the letter location in the choice of codon encoded provided a high level of randomness and eliminated the problem of statistics of English letters where in certain locations of the letters within the clear text the same codon is encoded , and in other locations the coding is to a different codon.

* We refer to the characters that are in individual locations within the string of text by the letter O, while the characters in even locations are indicated by the code E.

4. The nitrogenous bases of the two ADN series are encoded in English letters using Table I.6 and as shown:

Table I. 8 the process of coding nitrogenous bases into English lettres (for the example taken)

Resulting English character string	Agreed DNA string	The DNA string resulting from step 3	Resulting English character string	Agreed DNA string	The DNA string resulting from step 3
S	T	G	Z	A	C
K	A	T	D	T	A
D	T	A	I	G	A
B	T	T	X	T	C
S	T	G	W	G	C
V	G	G	E	C	G
E	C	C	H	G	T
D	T	A	K	A	T
P	C	A	R	A	A
M	C	T	Z	A	C
D	T	A	K	A	T
H	G	T	R	A	A
B	T	T	R	A	A
W	G	C	K	A	T
I	G	A	F	C	C
H	G	T	W	G	C
Z	A	C	F	C	C
K	A	T	E	C	G

5. Converting characters to a binary number string as each character represents five units in the binary system, depending on their sequence within the English alphabet (Z-A). Thus the binary number series is:

11010001000100111000101110010101000010111001010010110100101110010100100101100110101110011000101100110101100100000101001110110001100010010000011010010001000001010111010010100011010010 11

6. Performing an cipher transposition process in one of the methods by adopting a coding key inferred from the agreed DNA string for the purpose of increasing the randomness in the binary chain, so the resulting chain is:

1100110001011001101000010111001010010100100000010100111011000110001000
1100101010101001011110111001000001100000110100101000101001001011001101
1101001010011010001000100111000101011010010 11

7. Converting the binary series into a DNA series, that is, a series of nitrogenous bases using table 8 with the resulting string of rules being :

TATACCGCTCACCAGAGTATATCACATCACACATACCTCAT

8. Performing an cipher transposition process of the nitrogenous Rules series and using the same encryption key as in step 6 the resulting string is :

ATATCACCGCTCATACCTTATACACCACACATAGAATC GTC

Through the resulting string of nitrogenous bases that represent the resulting numerical text, he shows us the high randomness and effectiveness of the proposed method.

I.6 Conclusion

In this chapter we have introduced the basic notions necessary to the understanding of cryptology in order to initiate the reader. First, we have recalled the different security services. Then, we recalled the basic cryptographic principles used in the field of the security of information, namely the principle of **Kerckhoffs** and that of Shannon. Asymmetric encryption functions to situate the context of the study and to understand the continuation of the work.

In the second part of this chapter, we presented the basic structure of the DNA molecule, as well as the tools and concepts of molecular biology. The remarkable development of DNA computing gives birth to DNA cryptography and this by exploiting the parallelism as well as the important capacity of storage that this molecule offers.

As we have presented in this chapter, a new method to encrypt text files using DNA. The researchers have built a set of coding tables and used them to obtain the encrypted text in the form of DNA.

Chapter II: FPGA and HLS

II.1 Introduction

Modern electronics are turning more and more to digital, which has many advantages over analog system: high insensitivity to interference, reconfigurability, easy storage of information, etc. where Digital electronics is concerned with circuits that represent information using a finite set of output states .An FPGA is a type of IC that can be programmed for different algorithms after fabrication. Modern FPGA devices consist of up to two million logic cells that can be configured to implement a variety of software algorithms. Additionally, HLS is particularly interesting for designing FPGA circuits, where hardware implementations can be easily refined and replaced in the target device. Recent years have seen much activity in the HLS research community, with a plethora of HLS tool offerings, from both industry and academia. All these tools may have different input languages, perform different internal optimizations, and produce results of different quality, even for the very same input description. Hence, it is challenging to compare their performance and understand which is the best for the hardware to be implemented.

This chapter presents fundamental information about FPGAs and the High-Level Synthesis (HLS) tool.

II.2 FPGA

II.2.1 History

Fixed logic devices, a name that implies devices that cannot be reprogrammed, were the first approach to system designs. Although they were widely used, the large number of time requirements for the transition from a design to a prototype along with the fact that error correction would demand a new design and implementation led the way to the fabrication of PLDs.

One of the first attempts in the PLD field was PLAs. PLAs consisted of a set of AND gates and another set of OR gates (Figure II.1) which could be conditionally complemented to produce an output. PLAs were mainly used for implementing combinatorial logic circuits. Programmable Array Logic (PLA) was an evolution of PLA. PAL devices consisted of a

small PROM core and additional output logic used to implement various logic functions with a few components.

The beginning of new technology and market occurred in 1985 when Xilinx co-founders R. Freeman and B. Vonderschmitt introduced the first FPGA which was the first device that had programmable gates and interconnects. Since then, and especially during the 1990s, FPGA production grew explosively. Various vendors entered the market and the competition increased. In the early 1990s FPGAs were primarily used in telecommunications and networking, however, by the end of the decade and lately FPGA usage expanded to consumer, automotive, and industrial applications [28].

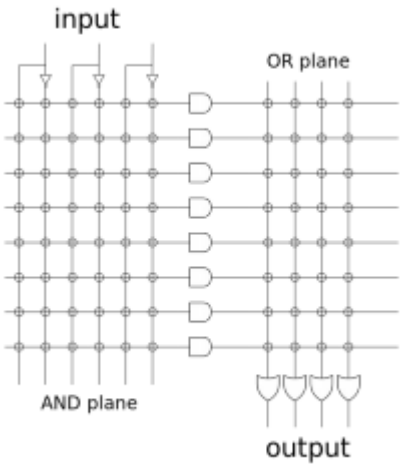


Figure II.1 Schematic paradigm

II.2.2 Definition

FPGAs are entirely reconfigurable components which allows them to be reprogrammed at will in order to significantly accelerate certain calculation phases. The advantage of this type of circuit is its great flexibility which allows them to be reused at will in different algorithms in a very short time.

FPGA board (Figure II.2) contain programmable logic blocks that can be wired in different configurations. These blocks create a physical array of logic gates that can be used to perform different operations. Because the gates are customizable, FPGAs can be optimized for any computing task. This gives FPGAs the potential to perform operations several times faster than a hard-wired processor.

The progress of these technologies allows us to make ever faster and more highly integrated components, which makes it possible to program important applications. This technology allows the implementation of many applications and offers a low-cost hardware implementation solution for small companies for whom the cost of developing a specific integrated circuit implies a too heavy investment[29].

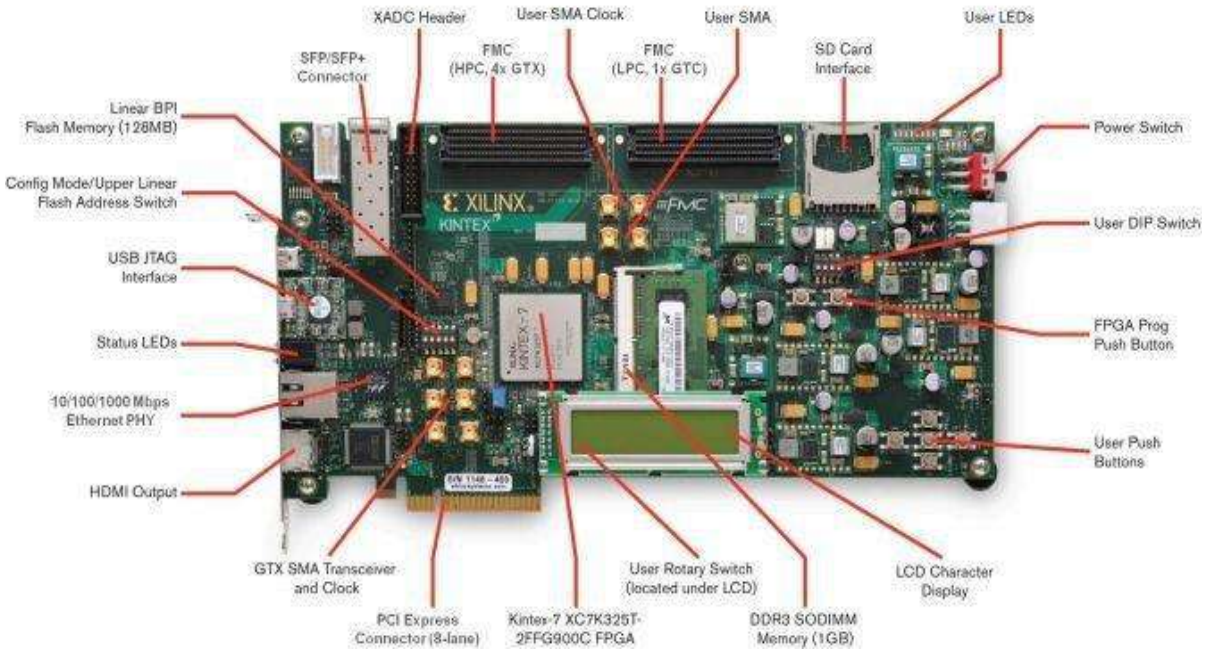


Figure II.2 Xilinx FPGA board

II.2.3 FPGA Architecture

FPGA circuits consist of a matrix of programmable logic blocks surrounded by programmable input-output blocks. The whole is connected by a programmable interconnection networkas shown the figure II.3.

FPGAs are quite distinct from other families of programmable circuits while offering the highest level of logic integration. There are currently several manufacturers of FPGA circuits, Xilinx and Altera are the best known, and several technologies and organizational principles. The architecture, chosen by Xilinx, is in the form of two layers.

- A layer called configurable circuitry.
- An SRAM memory network layer[29].

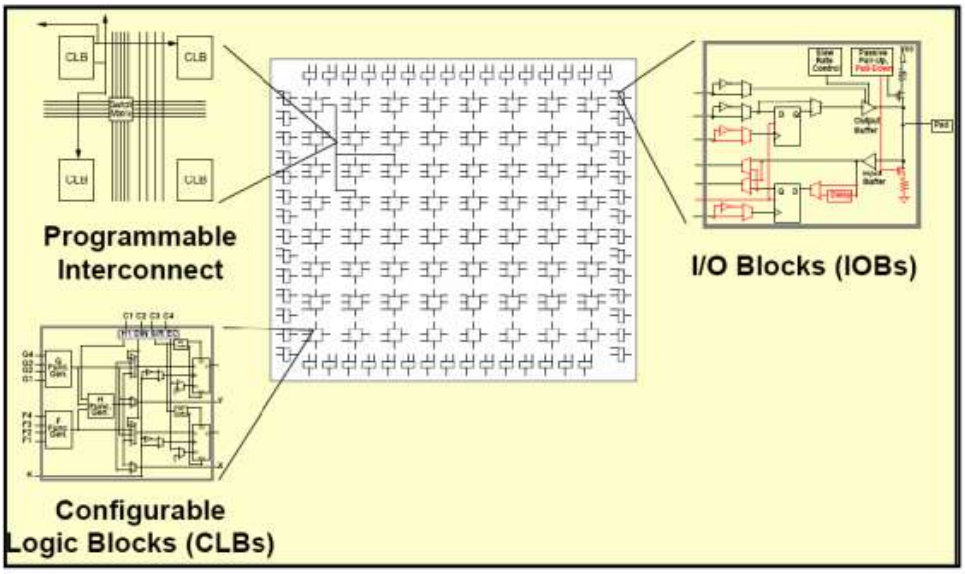


Figure II.3 Internal architecture of the Xilinx manufactured FPGA.

II.2.4 Internal structure

Basic structure and essential components of a generic FPGA. The programmable logic consists of a set of fine-grained blocks used to implement the logic of the application. This is sometimes referred to as the FPGA structure. Logic blocks are generally based on a (LUT) architecture, allowing them to implement any arbitrary function of the inputs. The logic blocks are usually arranged in a mosaic in a grid structure and interconnected via a programmable routing matrix that allows the blocks to be connected in arbitrary configurations. The input and output blocks (I/O) interface between the internal components or the FPGA core and external devices. Routing means that virtually any signal can be routed to any I/O pin on the device.

In addition to these basic features, most FPGAs provide some form of clock synchronization to control the timing of a clock signal with respect to an external source.

A clock distribution network provides clock signals to all parts of the FPGA while limiting the clock offset between different sections of a design. There is also dedicated logic to load the configuration into the FPGA. This logic does not directly form a part in the user's design, but it is the additional overhead required for FPGAs to be programmable and compatible provides the mechanisms for loading the user's design onto an FPGA. Each of these components is discussed in a little more detail in the following sections[30].

II.2.4.1 FPGA Logic

The smallest logic unit in the FPGA is the logic cell. It is the finest grained logic unit producing a single-bit output. It usually consists of a small LUT, conversion table and a lock on the output, the latch is used to record the output. On some FPGAs, both outputs can be provided, as on Figure II.4, on others, a multiplexer can be used to select whether the recorded or unrecorded signal is output. Registers can be used to build finite state machines, counters and to record data in pipelined systems.

The LUT can implement any arbitrary function of its inputs. More complex functions can be implemented by cascading multiple LUT layers by connecting the output, X, to the input of the next layer. FPGA manufacturers must make a trade-off when choosing the size of the LUTs. Consider implementing a simple two-input AND gate. With a large LUT, a significant fraction of the logic would be unused smaller LUTs would allow for greater resource utilization. However, when implementing complex functions with small LUTs, several logic cells should be cascaded, which can significantly increase the propagation delay. For complex functions, it would be better to use a larger mapping table. To balance this, the silicon area of the table grows exponentially with the number of inputs is more efficient for splitting a complex function into multiple LUTs[30].

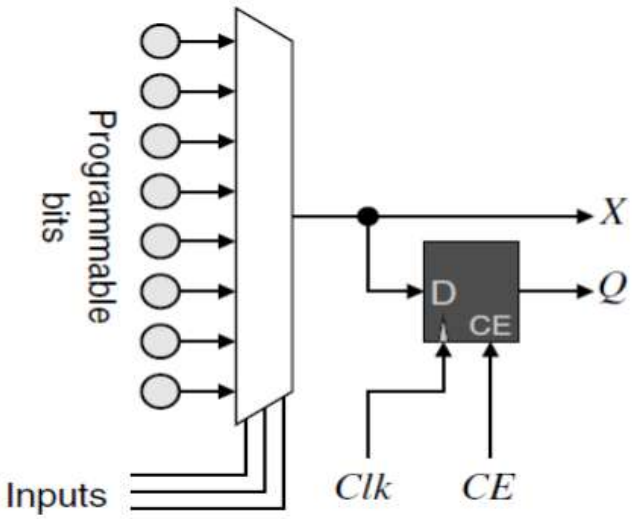


Figure II.4 A logic logic cell is the foundation of an FPGA

Homogeneous fine-grained FPGAs consist of a sea of identical logic blocks. However, most current FPGAs have moved away from this homogeneous structure, with a heterogeneous mix of both fine-grained logic blocks and larger coarse-grained blocks intended for acceleration

commonly used operations. This has both advantages and disadvantages in terms of design. Heterogeneous systems will generally be faster, as coarse-grained blocks will not have the same level of overhead as those However, heterogeneous devices are also more difficult to program, and if the mix of specialized blocks is not correct, much of the functionality may remain underutilized[30].

II.2.4.2 Interconnection

The purpose of the programmable interconnection is to allow the different logic blocks to be used flexibly connected to implement the desired functionality. It is impossible to provide a dedicated direct connection from all possible outputs to all possible inputs. In each application, only a very small fraction of these connections would be necessary; however, none of the connections can potentially be required. The solution is to have a smaller set of connection resources (routing lines) that can be used. Flexibly shared to create the connections required for a given application.

Since the interconnect takes place on silicon, there is a trade-off. If only a few routing lines are provided, more space is available for logic, but complex circuits cannot be built due to insufficient interconnect resources. Alternatively, if too many routing lines are provided, many will not be used, reducing the area available for logic. There is an optimum somewhere between these extremes, where sufficient routing lines are available for most applications, without using too much space.

The connection network for most FPGAs are based on a grid structure. At each grid intersection, a form of crossbar switch allows a programmable connection between the horizontal and vertical routing lines.

Another factor that must be considered is the propagation delay through the interconnection network. Each switch that the signal passes through will add to the delay. This has led to a segmented structure, where not all routing lines are switched at each junction. In addition, specific direct connections can be provided between adjacent logic blocks to reduce the number of signals that must pass through the interconnected matrix. While having a range of different interconnect resources improves flexibility and speed, it can make the process of finding good routes significantly more complex, especially for designs that use most of the logic available on the FPGA.

It is important that only one output drives a routing line. If two outputs are connected together with one high and the other low, the power supply will effectively short-circuit, damaging the chip. One of the roles of FPGA programming tools is to ensure that this never happens. Peripherals using bus routing structures must limit the current of the control transistors or use an open drain and a passive pull-up. Both approaches affect velocity and power dissipation[30].

II.2.4.3 Input and output

The input and output blocks allow the connection of signals between the FPGA and external devices. The flexibility of FPGAs requires that they be able to connect to a wide range of other components. Most of the input/output pins can be programmed as inputs, outputs or both and can be connected directly to the internal logic memory of the FPGA, or through the registers. This range of functionality can be provided by a generic I/O device. Such as the one shown on Figure II.5, each multiplexer being controlled by a programmable configuration bit. The main complexity of I/O comes from the large number of different standard signals used by different devices and ensuring proper synchronization between the FPGA and the off-chip components.

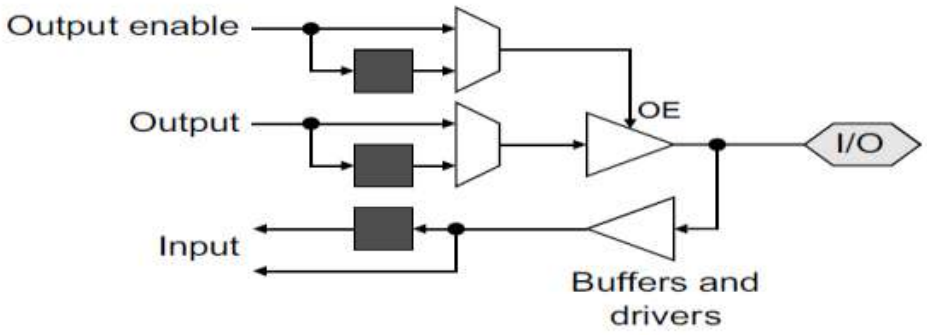


Figure II.5 Generic input / output block

When interfacing with high-speed signals, it is important to consider the effects on the transmission line. As a general rule, any connection longer than about one-eighth of the rise time should be properly terminated to minimize signal reflections. As FPGAs have become larger and faster, it has become common practice to provide on-chip programmable termination. If not provided, it is necessary to ensure that the appropriate termination is built off-chip for fast switching signals. Many high-level communication protocols would require significant FPGA resources just to handle the protocol. While this can be implemented in the

FPGA fabric, the design is often quite complex and is best implemented using a dedicated interface chip. These chips typically handle media access (MAC) and physical signaling (PHY) requiring only standard logic signals in their interface with the FPGA. Some FPGAs incorporate dedicated MAC and PHY circuits for commonly used communications protocols, such as PCI Express and Gigabit Ethernet. Most FPGAs offer a range of general-purpose and dedicated I/O connections[30].

II.2.4.4 Clock

FPGAs are designed primarily as synchronous devices, so they require a clock signal. Although multiple clocks can be used in a design, registers, synchronous memories and I/O blocks can only be controlled by one clock. A clock domain includes all circuits controlled by a particular clock. Since a clock potentially drives many inputs throughout the FPGA, there are two special options. Characteristics of clock lines that distinguish them from other interconnects in an FPGA. First, it is important that all registers of a clock domain are synchronized simultaneously. The clock offset is the difference in the arrival time of the clock signal at successive registers in a design (Friedman, 2001). Any bias will reduce the synchronization margin and impact the maximum clock speed of a design. Although asymmetry can be minimized by using H-tree or X-tree clock distribution networks, these do not match well with the FPGA grid structure (Lamoureux andWilton, 2006). Instead, a spine and rib structure is typically used (Figure II.6), where the clock is distributed to the rows using a spine and then to the elements in the row using the ribs (Lamoureux and Wilton, 2006). A series of buffers is used in the distribution network to handle the large fan-out. In practice, it is a bit more complex than that, as the network must support multiple clock domains, and many FPGAs also have a mix of local and global clocks[30].

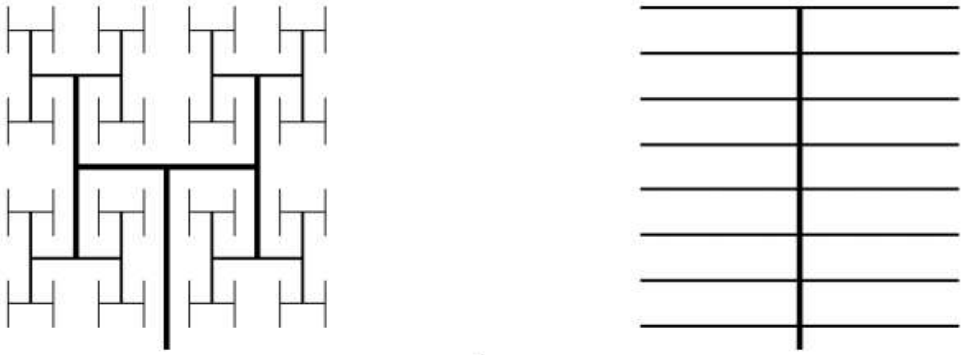


Figure II.6 Clock distribution networks. Left: H-structure; right: backbone and rib structure

II.2.4.5 Configuration

All switches, structures and options of the FPGA are controlled by bits of more generally, the configuration data is contained in the FPGA in static memory cells (SRAM), although flash programmable devices are available from some manufacturers. FPGAs programmed in flash are non-volatile; retain their configuration after power off, so that they can start working immediately after being switched on. Static memory FPGAs lose their configuration when they are power off, which requires the configuration file to be reloaded each time the system is turned on. Static memory had the advantage of being infinitely reprogrammable; flash memory usually has a limited number of write cycles.

An FPGA is configured by loading the configuration file into internal memory. This is usually accomplished by transmitting the configuration file to the FPGA either serially, either in parallel 8, 16 or in some cases 32 bits at a time. In a standalone system, the configuration file is usually contained in a file. External flash memory, such as that shown on Figure 20, with the board configured to automatically configure the FPGA at startup. FPGAs can also be programmed serially via a JTAG interface, and this interface is commonly used when programming FPGAs from a host computer system. The need to transfer the configuration to the FPGA each time the system is powered up exposes the configuration file making the intellectual property contained in it vulnerable to hacking or reverse engineering. High-end FPGAs overcome this limitation by encrypting the configuration file and decrypting the configuration once it is in the FPGA. Some FPGAs also support configuration replay from the FPGA. This feature can be useful for debugging a design with the appropriate tools, but is generally not used in normal operation[30].

II.2.4.6 Energy consumption

In embedded applications, the power consumption of the system, and in particular the FPGA, is critical. The power dissipated by an FPGA can be divided into two components: the static power and the dynamic power. Dynamic power is the power required to switch signals from 0 to 1 and vice versa. Inversely, the static energy is the one that is consumed without switching, simply because of the device turned on.

Most FPGAs are based on CMOS technology. A CMOS inverter, the most basic CMOS circuit, is shown on Figure II.7. It consists of a P-channel pull-up transistor and an N-channel pull-down transistor. Only one of the transistors is turned on at a time, and because MOS transistors have very high shock resistance, CMOS circuits generally have very low static power dissipation[31].

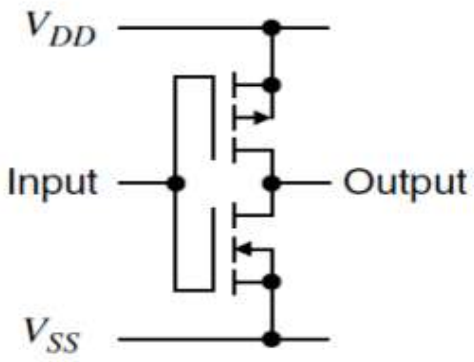


Figure II. 7 CMOS inverter

II.2.5 FPGA Applications

Due to their programmable nature, FPGAs are an ideal fit for many different markets. As the industry leader, Xilinx provides comprehensive solutions consisting of FPGA devices, advanced software, and configurable, ready-to-use IP cores for markets and applications such as:

- **Aerospace & Defense** - Radiation-tolerant FPGAs along with intellectual property for image processing, waveform generation, and partial reconfiguration for SDRs.
- **ASIC Prototyping** - ASIC prototyping with FPGAs enables fast and accurate SoC system modeling and verification of embedded software.

- **Automotive** - Automotive silicon and IP solutions for gateway and driver assistance systems, comfort, convenience, and in-vehicle infotainment. - Learn how Xilinx FPGA's enable Automotive Systems.
- **Broadcast & Pro AV** - Adapt to changing requirements faster and lengthen product life cycles with Broadcast Targeted Design Platforms and solutions for high-end professional broadcast systems.
- **Consumer Electronics** - Cost-effective solutions enabling next generation, full-featured consumer applications, such as converged handsets, digital flat panel displays, information appliances, home networking, and residential set top boxes.
- **Data Center** - Designed for high-bandwidth, low-latency servers, networking, and storage applications to bring higher value into cloud deployments.
- **High Performance Computing and Data Storage** - Solutions for NAS, SAN, servers, and storage appliances.
- **Industrial** - Xilinx FPGAs and targeted design platforms for Industrial, Scientific and Medical (ISM) enable higher degrees of flexibility, faster time-to-market, and lower overall non-recurring engineering costs (NRE) for a wide range of applications such as industrial imaging and surveillance, industrial automation, and medical imaging equipment.
- **Medical** - For diagnostic, monitoring, and therapy applications, the Virtex FPGA and Spartan® FPGA families can be used to meet a range of processing, display, and I/O interface requirements.
- **-Security** - Xilinx offers solutions that meet the evolving needs of security applications, from access control to surveillance and safety systems.
- **-Video & Image Processing -** Xilinx FPGAs and targeted design platforms enable higher degrees of flexibility, faster time-to-market, and lower overall non-recurring engineering costs (NRE) for a wide range of video and imaging applications.
- **Wired Communications -** End-to-end solutions for the Reprogrammable Networking Linecard Packet Processing, Framer/MAC, serial backplanes, and more
- **Wireless Communications -** RF, base band, connectivity, transport and networking solutions for wireless equipment, addressing standards such as WCDMA, HSDPA, WiMAX and others[32].

II.2.6 FPGA Development tools

The development tools allow the designer to program the circuit from the description of the function to perform. This description can be textual in this case one generally uses languages of development: Verilog and VHDL. This description can also be graphical and this by means of chronograms, state graphs and function symbols[29].

II.2.7 Advantages of using FPGAs

FPGAs have undergone important architectural evolutions. Resulting mainly from the increase of integration capacities the main advantages of the FPGA technology are:

- Very fast execution.
- Easy and flexible to use.
- Suitable for rapid prototyping.
- Cost-effective manufacturing and long-term maintenance[10].

II.3 Presentation of the zynq 7000 card

II.3.1 Zynq-7000 Family Description

The Zynq-7000 family offers the flexibility and scalability of an FPGA, while providing performance, power, and ease of use typically associated with ASIC and ASSPs. The range of devices in the Zynq-7000 family allows designers to target cost-sensitive as well as high-performance applications from a single platform using industry-standard tools. While each device in the Zynq-7000 family contains the same PS, the PL and I/O resources vary between the devices. As a result, the Zynq-7000 and Zynq-7000S SoCs are able to serve a wide range of applications[33].

II.3.2 Zynq-7000

Zynq-7000 devices are equipped with dual-core ARM Cortex-A9 processors integrated with 28nm Artix-7 or Kintex®-7 based programmable logic for excellent performance-per-watt and maximum design flexibility. With up to 6.6M logic cells and offered with transceivers ranging from 6.25Gb/s to 12.5Gb/s, Zynq-7000 devices enable highly differentiated designs for a wide range of embedded applications [25] .

Zynq-7000 has two main parts: Processing System (PS) in which there is a dual-core processor ARM Cortex-A9, and Programable Logic (PL) that works like Field-Programmable Gate Array (FPGA). In general the Zynq architecture can be seen as in Figure II.8 [25].

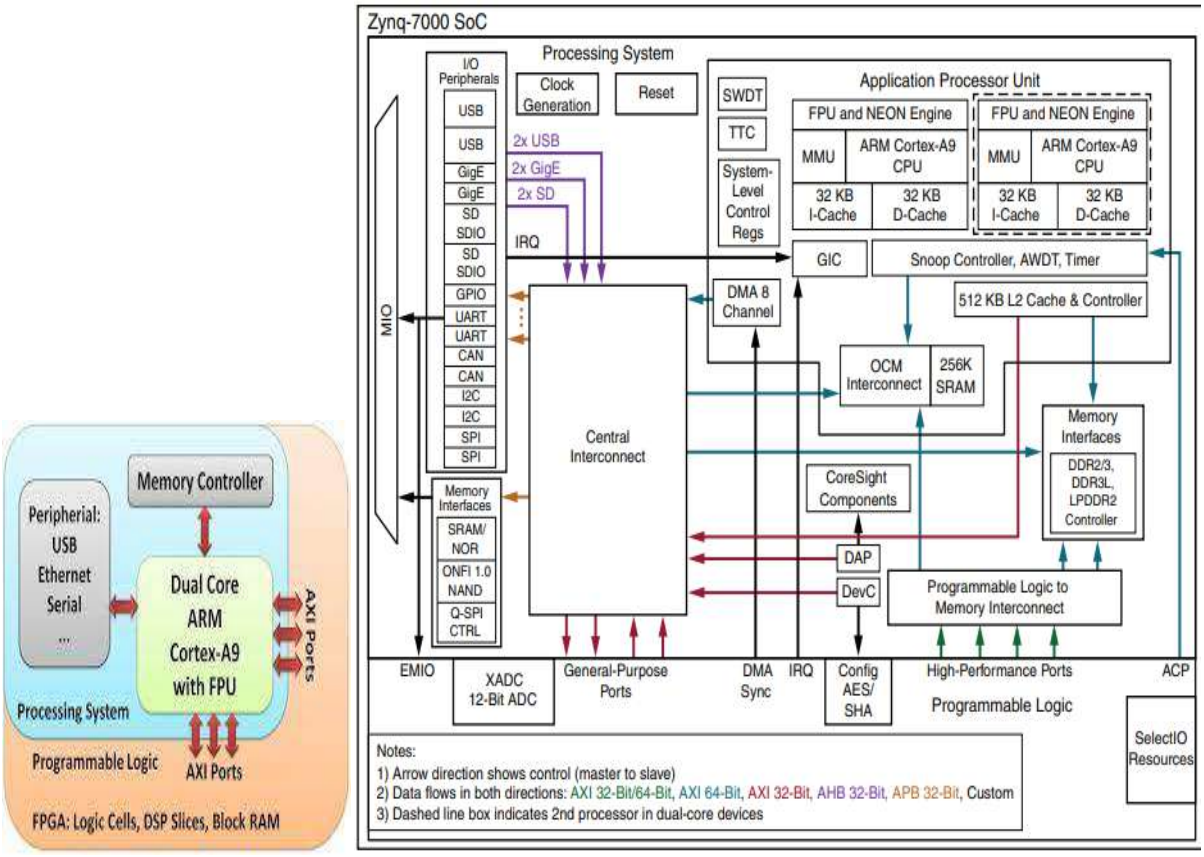


Figure II.8 Architectural Overview

On the PS there is ARM Cortex-A9 with maximum clock up to 1 GHz with three watchdog timers, one global timer, and two triple-timer counters. The PS is ideal for doing software routines. Whereas in the PL there are DSP Blocks, PCI Express Block, and two 12-Bit Analog-to-Digital Converters. The PL is ideal for implementing high-speed logic, arithmetic, and data flow subsystems. While on the interconnect between PS and PL using Advanced eXtensible Interface 4 (AXI4) which is the fourth generation of AMBA interface[25].

II.4 High-Level Synthesis

II.4.1 Definition

HLS is an automated design process that takes as input an algorithmic description in order to create the digital hardware that implements the desired function.

The Xilinx Vivado High-Level Synthesis (HLS) is a tool that transforms a C specification into an RTL implementation that you can synthesize into a Xilinx field programmable gate array (FPGA).

HLS input is specified in languages like C, C++, System C, or even Matlab, and allows the use of advanced language features like loops, arrays, structs, classes, pointers, inheritance, overloading, template, polymorphism, etc. This is higher level than RTL description languages and allows concise, reusable, and readable design descriptions.

II.4.2 High-Level Synthesis for FPGAs

FPGA devices have long been an attractive option for the energy-efficient acceleration of applications with high computation demand. However, hardware development targeting FPGAs remains challenging and time-consuming_often requiring hardware design expertise and a register transfer level (RTL) algorithm description for efficient implementation. Manual design of RTL hardware often takes many months_an orders of magnitude longer than software implementations even when using available hardware IP.

High-level synthesis (HLS) targets this problem: HLS tools synthesize algorithm descriptions written in an HLL such as C/C++/SystemC. An HLL description can typically be implemented faster and more concisely, reducing design effort and susceptibility to programmer error. Thus, HLS provides an important bridging technology—enabling the speed and energy efficiency of hardware designs with significantly reduced design time. In recent years, HLS has made significant advances in both the breadth of HLS compatible input source code and the quality of the output hardware designs. Ongoing development of HLS has led to numerous industry and academia-initiated HLS tools that can generate device-specific RTL descriptions from popular HLLs such as C, C++, SystemC, CUDA, OpenCL, MATLAB, Haskell, and specialized languages or language subsets.

The advancements in a language support for HLS mean that many implementations can be synthesized to hardware, but the original software design may not be suitable for hardware implementation. High-level languages such as C have been shown as an effective means for capturing FPGA circuits when the C description is written specifically for HLS and hardware (e.g., C implementation is derived from manual FPGA implementation) . However, although software specifically written for HLS is sometimes available, must of the software is not designed for synthesis.

II.4.3 HLS Benefits

High-level synthesis bridges hardware and software domains, providing the following primary benefits:

- Improved productivity for hardware designers Hardware designers can work at a higher level of abstraction while creating high-performance hardware.
- Improved system performance for software designers Software developers can accelerate the computationally intensive parts of their algorithms on a new compilation target, the FPGA[34].

II.5 Conclusion

In this chapter, we have presented the basic of the FPGA board and the intern architecture ,also the configuration and the applications as well as we talked about the zynq-7000 family which include the ZYBO board that we used for next chapter to implement the DNA-encryption algorithm.

In the second part of this chapter, we presented the High-Level Synthesis HLS that we used to create the digital hardware for the desired functions in the next part of this work.

Chapter III: Experiments and results

III.1 Introduction:

In this chapter, we will present the implementation part of our DNA encryption/decryption system, as well as the different results obtained. The system is first developed in C language for software execution on FPGA. Then, parts of the encryption and decryption program are transformed into custom IP-core using the Vivado-HLS tool, the aim is to speed up the execution in the FPGA platform, and the different implementation steps are presented in detail.

III.2 Presentation of the development tools:

III.2.1 Vivado

Vivado Design Suite is a software produced by Xilinx for synthesis and analysis of HDL designs, superseding Xilinx ISE with additional features for system on chip development and high-level synthesis. Vivado represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE).

III.2.2 Vivado HLS

The Xilinx Vivado High-Level Synthesis (HLS) is a tool that transforms a C specification into a register transfer level (RTL) implementation that you can synthesize into a Xilinx field programmable gate array (FPGA).

High-level synthesis bridges hardware and software domains, providing the following primary benefits:

Hardware designers can work at a higher level of abstraction while creating high-performance hardware.

Software developers can accelerate the computationally intensive parts of their algorithms on a new compilation target "the FPGA".

III.2.3 C Language

C is a computer programming language. That means that you can use C to create lists of instructions for a computer to follow. The C program is the human-readable form, while the

executable that comes out of the compiler is the machine-readable and executable form.

III.2.4 ZYBO Board

The Zybo Z7 is a ready-to-use embedded software and digital circuit development board built around the Xilinx Zynq™-7000 family. The Zynq-7000 tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic.

III.3 Vivado HLS design flow

The Xilinx Vivado HLS tool synthesizes a C function into an IP block that you can integrate into a hardware system. It is tightly integrated with the rest of the Xilinx design tools and provides comprehensive language support and features for creating the optimal implementation for your C algorithm.

The following Figure shows an overview of the Vivado HLS design flow (figure III.1).

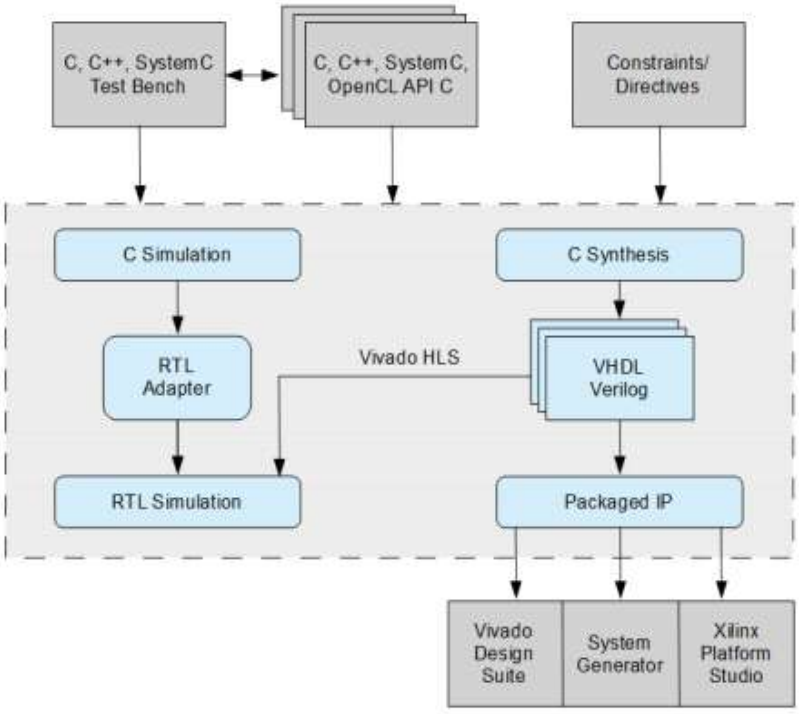


Figure III.1 Vovado HSL design folow

III.3.1 Design step

The figure below (figure III.2) demonstrates the steps followed to realize an HLS design:

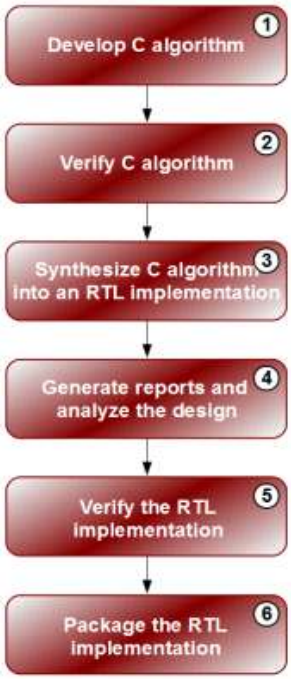


Figure III.2 Design steps

III.3.1 functions design

On Zynq-7000 there are two main parts, which are PL and PS.

IP core for decryption case for the function "convert from binary to character " and for encryption case "convert from character to binary "will be implemented on the PL, while data processing for data input and display will be on the PS. Furthermore, on the PL part will be designed using Xilinx Vivado HLS 2018.2. While on the PS will be designed using Xilinx SDK.

In Zynq-7000, the system frequency on the PS has a maximum frequency of up to 1 GHz while on the PL has a system frequency of 50 MHz. To communicate between PL and PS, then the data will pass through AXI4-Lite interface first. The IP core will be connected via Pmod connectors found on Zybo.

The data received by the IP Core and stored in the AXI register will be read and sent via UART and finally displayed by the SDK Console. While the process of sending data is done in PS by entering data through AXI register.

III.3.2 Develop C Algorithm

III.3.2.1 Developed algorithm at the C-level

Work at a level that is abstract from the implementation details, which consume development time.

We developed our functions to encrypt and decrypt texts .we used 5 functions to realize that as we explained in the second chapter.

III.3.2.2 Verified the algorithm at the C-level

Validate the functional correctness of the design more quickly than with traditional hardware description languages.

III.3.3 Developing custom IP core using HLS

In this step we will synthesized the C algorithm into an RTL implementation.

III.3.3.1 Create a new project

We launched and navigated the Vivado High-Level Synthesis (HLS) tool, created a project using New Project Creation Wizard.

In the Project Configuration dialog box specified the name and the location of the new project:

Open HLS--> Create new project --> Choose the location and file name as shown on the figure III.3.

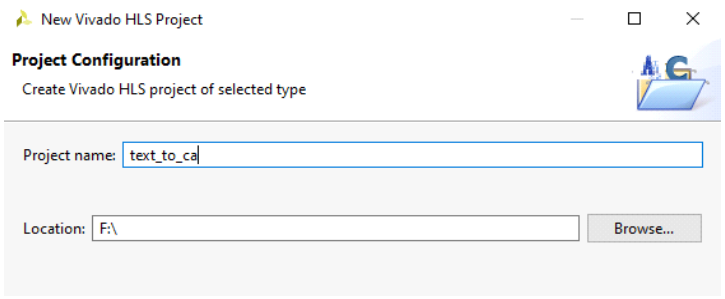


Figure III.3 Create new vivado project dialog box

III.3.3.2 Add C based source files

Than added the C-based design file and specified the top function (Figure III.4), In any C

program, the top-level function is called main(). In the Vivado HLS design flow, us can specify any sub-function below main() as the top-level function for synthesis. Us cannot synthesize the top-level function main().

Only one function is allowed as the top-level function for synthesis.

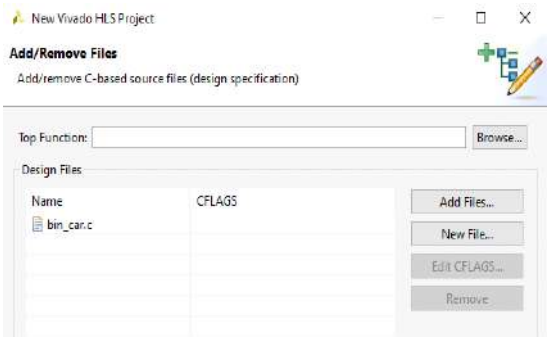


Figure III.4 add/remove design file dialo box

III.3.3.3 Add C-based testbench file

After clicking next we added the C-based testbench files (Figure III.5). The C test bench includes the function main () and any sub-functions that are not in the hierarchy under the top-level function for synthesis.

These functions verify that the top-level function for synthesis is functionally correct by providing stimuli to the function for synthesis and by consuming its output.

Vivado HLS uses the test bench to compile and execute the C simulation.

Note: Because Vivado HLS uses the test bench to both verify the C function prior to synthesis and to automatically verify the RTL output, using a test bench is highly recommended.

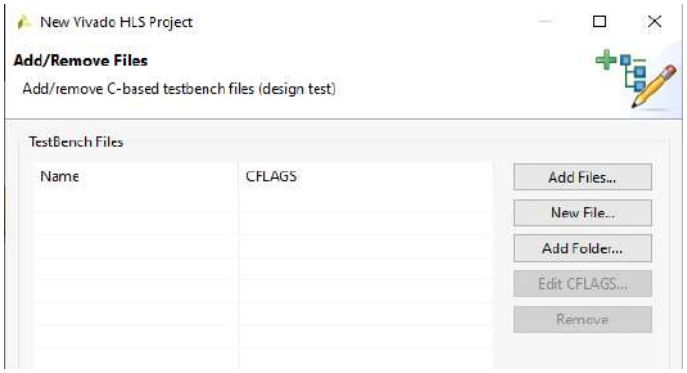


Figure III.5 add/ remove c testbench file

III.3.3.4 The Solution Configuration

Clicking next, the Solution Configuration window (shown on the Figure III.6) specifies the technical specifications of the first solution.

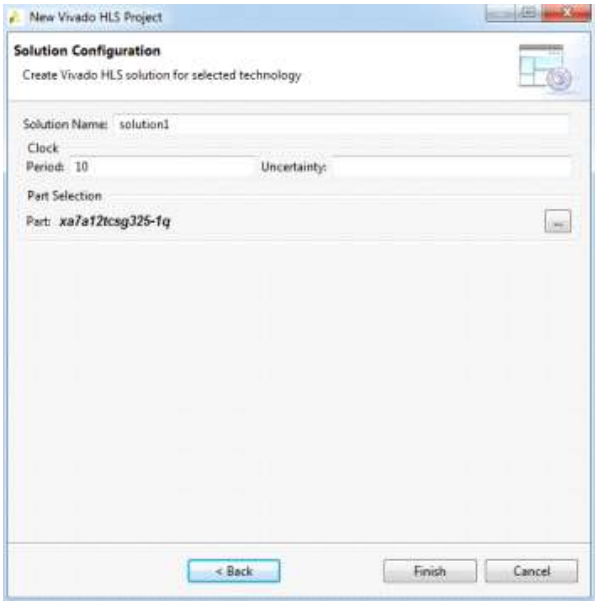


Figure III.6 the Solution Configuration window

In the Solution Configuration dialog box accept the default solution name (solution1), clock period (10 ns), and blank clock uncertainty (defaults to 12.5% of the clock period, when it is left blank then it is undefined).

The Solution Configuration dialog box allows us to specify the details of the first solution:

- Solution Name: Vivado HLS provides the initial default name solution1, but us can specify any name for the solution.
- Clock Period: The clock period specified in units of ns or a frequency value specified with the MHz suffix (for example, 100 MHz).
- Uncertainty: The clock period used for synthesis is the clock period minus the clock uncertainty. Vivado HLS uses internal models to estimate the delay of the operations for each FPGA. The clock uncertainty value provides a controllable margin to account for any increases in net delays due to RTL logic synthesis, place, and route. If not specified in nanoseconds (ns) or a percentage, the clock uncertainty defaults to 12.5% of the clock period.

Choosing the Part: Click to select the appropriate technology, as shown on the following figure (Figure III.7).

In our project we used the board xc7z010clg400-1, so in the Default Part dialog.

Box we selected Parts option and set the filter parameters on the same way as it is shown on the figure below:

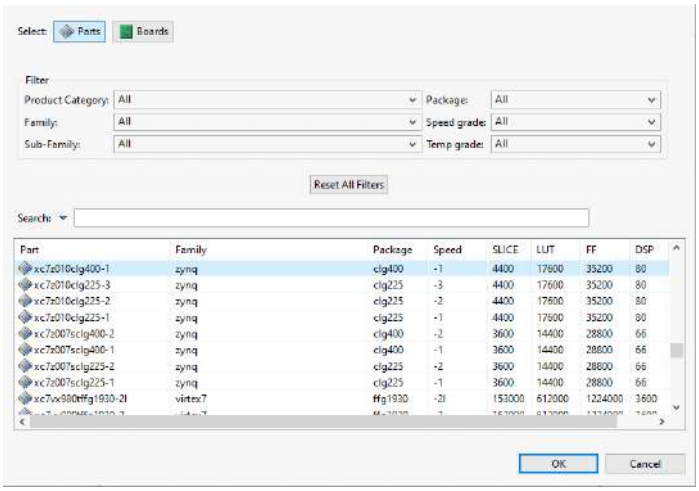


Figure III.7 select board dialog box

In the Solution Configuration dialog box (Figure III.8), the selected part name now appears under the Part Selection heading.

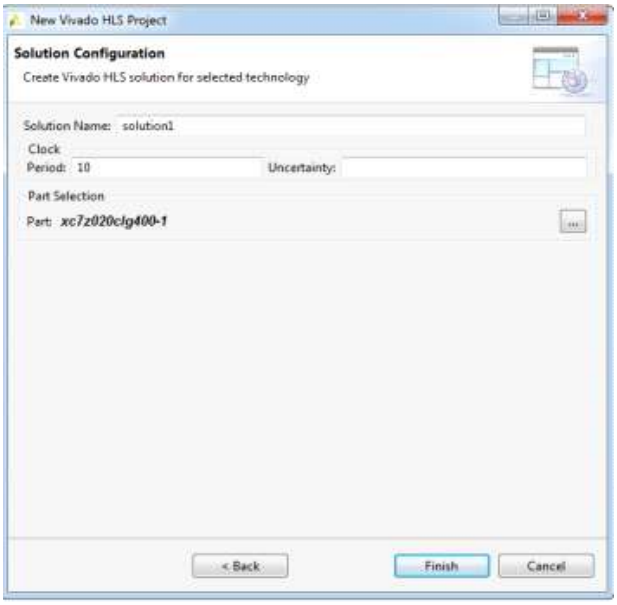


Figure III.8 Solution Configuration dialog box

After clicked finish the new project will open, in the encryption HLS project we created a project which include the functions name "convert caractere to binaire" and in the decryption the function was "convert binaire to caractere ".

When Vivado HLS creates a new project, it also creates a directory with the name and at the location .That means that the all project data will be stored in the project_name directory.

III.3.4 Verify C Algorithm

The second step within an HLS project is to con_rm that the C code is correct, this process is called C Validation or C Simulation.

In this step we removed and changed functions which is not supported.

Vivado HLS supports many C, C++, and System C language constructs and all native data types for each language, including _float and double types. However, synthesis is not supported for some constructs, including:

- Dynamic memory allocation :An FPGA has a fixed set of resources, and the dynamic creation and freeing of memory resources are not supported such as function malloc which is "Memory Allocation "In C, the library function malloc is used to allocate a block of memory on the heap. The program accesses this block of memory via a pointer that malloc returns. When the memory is no longer needed, the pointer is passed to free which deallocates the memory so that it can be used for other purposes. We removed this function by giving a constant size for our register.
- The strlen() function calculates the length of a given string ,we changed this function by creating a function do the same role.
- strcpy() which is a standard library function in C/C++ and is used to copy one string to another also not supported and we did a function with same role.
- strcat()the strcat function appends a copy of the string pointed to by s2 to the end of the string pointed to by s1. It returns a pointer to s1 where the resulting concatenated string resides we changed it by a function with same role.

Those 4 functions were not supported in our project we changed them for having a simulated program on HLS.

III.3.4.1 Run C Simulation

Click the Run C Simulation toolbar button to open the C Simulation dialog box shown the Figure III.9.



Figure III.9 the Run C Simulation button

The C code is compiled and the C simulation is automatically executed without any problems, and the designed is ready for synthesis.

When C simulation completed, a folder csim is created inside the solution1 folder (Figure III.10).

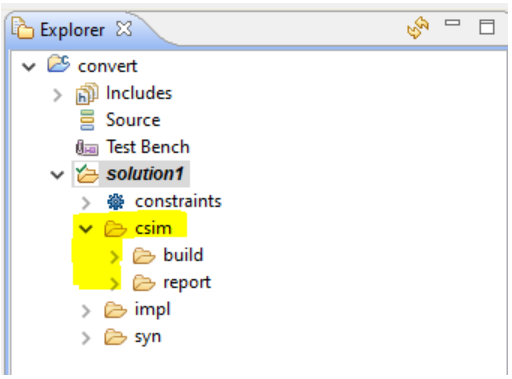


Figure III.10 the csim folder

The folder csim/build is the primary location for all files related to the C simulation:

- Any files read by the test bench are copied to this folder.
- The C executable file csim.exe is created and run in this folder.
- Any files written by the test bench are created in this folder.
- After synthesis, Vivado HLS automatically creates synthesis reports to help us understand the performance of the implementation.

The next step in the Vivado HLS design flow is to execute synthesis.

III.3.4.2 Synthesize C Algorithm into an RTL Implementation (High-Level Synthesis)

In this step, we synthesized the C design into an RTL design and reviewed the synthesis report clicked on the button shown on the figure III.11.



figure III.11 the Run C synthesis button

When synthesis completes, the synthesis report for the top-level function open automatically in the Information pane.

The synthesis report provides details on both the performance and area of the RTL design.

III.3.4.3 Applying optimization directives

Directives are optional and direct the synthesis process to implement a specific behavior or optimization.

Vivado HLS applies the directive to the top-level function (scope), and the Interface port (object) is identified in the directive.for example in our decryption project "convert_binaire_caractere" is specified as an s_axilite interface, our apply optimization directive is "INTERFACE" and port return which means it's return the result.

INTERFACE specifies how RTL ports are created from the function description.

When we apply directives to an interface, Vivado HLS applies the directive to the top-level function, because the top-level function is the scope that contains the interface.

And the following destination was a "Source File"(Figure III.12).



Figure III.12 choose a source file

Choosing a source file because when Vivado HLS inserts the directive directly into the C source file as a pragma. the optimization directives are embedded into the C source code. Ideal when the C sources files are shipped to a third-party as C IP. No other files are required to recreate the same results. Useful approach for directives that are unlikely to change, such as INTERFACE.

In our decryption example we inserted also a directive for "tab" and "resultat " but as a slave offset fort the top function and as a master axilit "m_axi" interface and we specified the depth necessary (same for the encryption only with differents functions name).

The Figure III.13 demonstrate our decryption directive for the function convert_binaire_caractere:

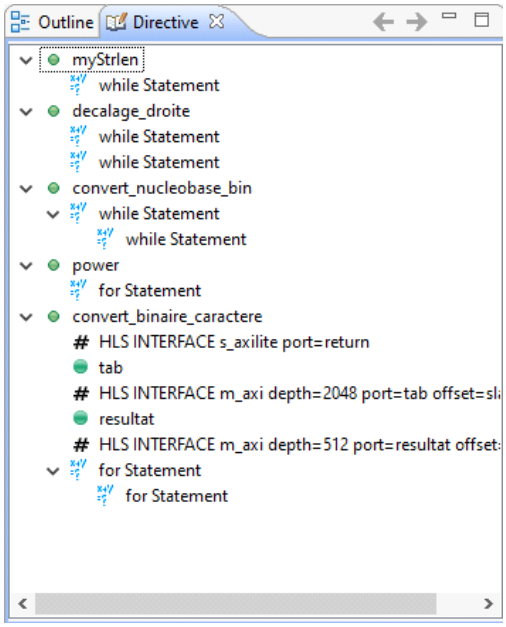


Figure III.13 the decryption directive for the function convert_binaire_caractere

III.3.4.4 Package the RTL Implementation:

The final step in the Vivado HLS flow is to export the RTL design as a block of Intellectual Property (IP) which can be used by other tools in the Xilinx design flow.

The RTL design can be packaged into the following output formats:

-IP Catalog formatted IP for use with the Vivado Design Suite.

-System Generator for DSP IP for use with Vivado System Generator for DSP.

-Synthesized Checkpoint (.dcp).

The RTL synthesis is executed in the verilog or vhdl directories, but the results of

RTL synthesis are not included in the packaged IP.

Last step to export the RTL as IP for Vivado designe suite by clicking on the button shown on

the Figure III.14:



Figure III.14 the export RTL button

III.4 Using developed IP core in vivado design suite

How to integrate a custom IP using Xilinx Vivado IDE and SDK tool, will be shown on this part**.**

III.4.1 Create a New Project with Included Developed IP Core:

First, a new project must be created, In the Vivado IDE tool we created a new project and we specified the type as an RTL Project, targeting the ZYBO board (Figure III.15) and saved it.

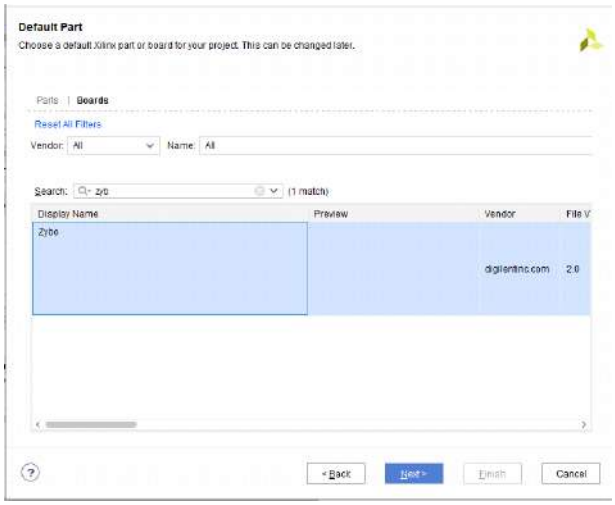


Figure III.15 Choosing the ZYBO board

In the Settings dialog box we added the IP Repository which is for adding directories to the list of repositories, then add additional IP to a selected repository.

From seeting -> Ip ->Repository and then we selected the IP from its location ->apply->ok.

On the figure below (Figure III.16) our Decryption function's IP Repository "decryption case".

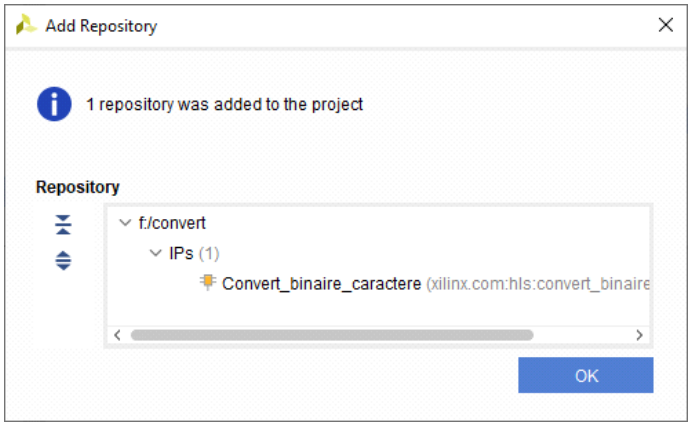


Figure III. 16 Repository add to the project

III.4.2 Block diagram with added necessary IPs and generate Bitstream:

The next step is to create the Block Design, then add the we added the IP and the ZYNQ7 Processing System as shown on the figure below (Figure III.17).

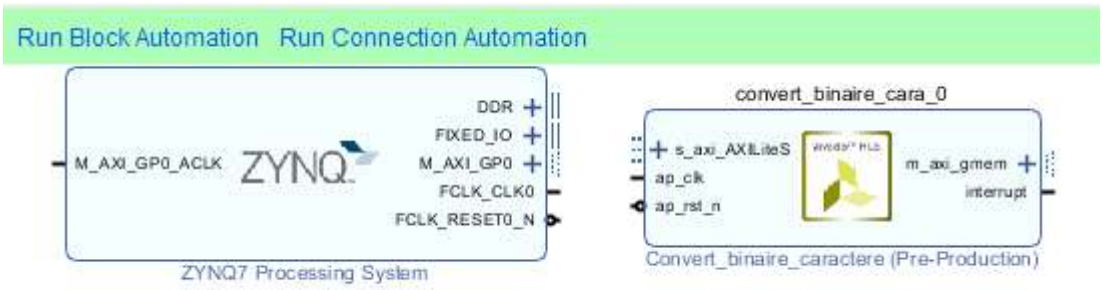


Figure III.17 add the IP and the ZYNQ7000 Processing System

After adding all the necessary IPs into the design canvas, and running the Block automation and the Connection Automation, the design look as it is shown on the following figure (Figure III.18).

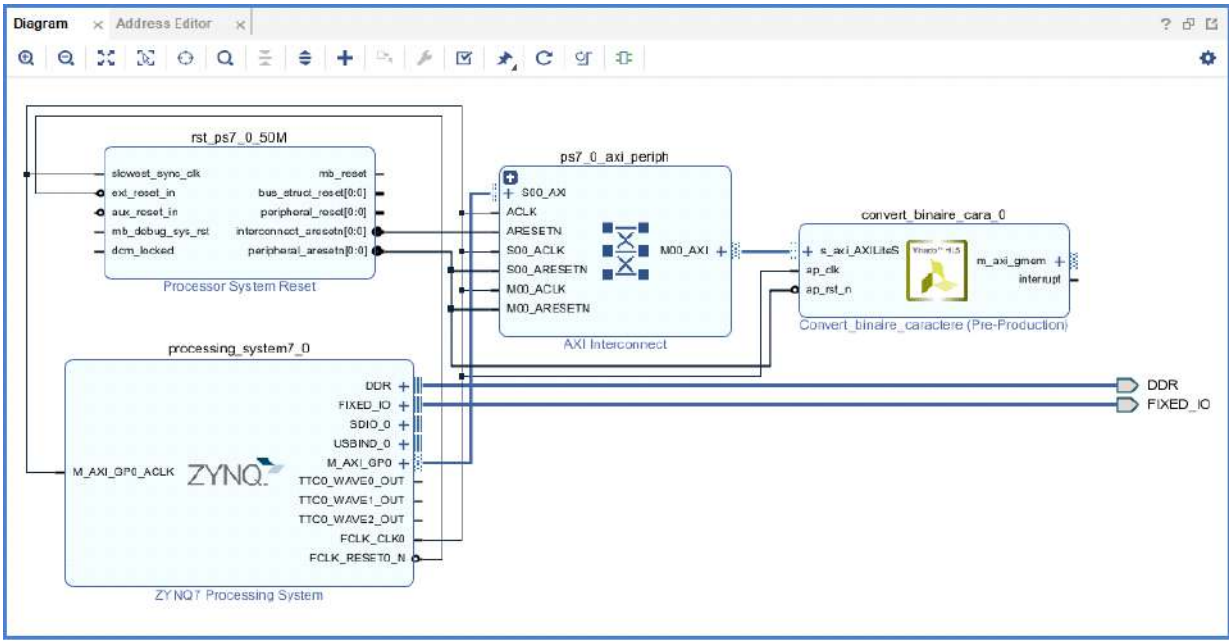


Figure III.18 architecture the design on vivado suite

Then we validated and Saved Block Design From the sidebar menu of the design canvas, and checked by clicking the Validate Design button then create the Design wrapper.

The last step is to generate bitstream and export hardwar to launch SDK.

III.4.3 Result obtained:

Launch SDK and then the designe wraper appread including the DRIVERS for creating a file.c, the figure bellow show the two functions chosen for HLS synthesize in both cases ,encryption case where the function is "convert_texte_binaire " ,and "convert_to_caractere" for the decryption case as shown on the Figure III.19.

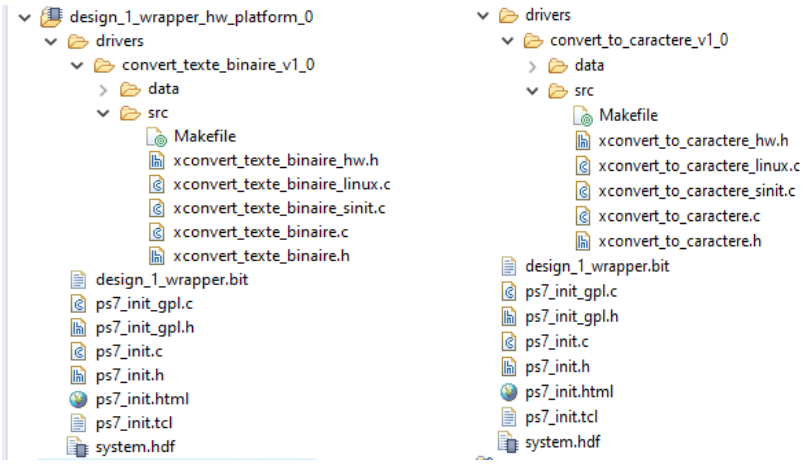


Figure III.19 Encryption_decryption function drivers

The files "xconvert_to_caractere.h", and "xconvert_texte_binaire.h" including all functions necessaries to run the IP core created.

after extracted all the functions necessaries for running the IP core function's created by the design wrapper , we added all other functions built on c and all libraries necessaries.

And the library necessary are:

```
#include "platform.h"
#include "xil_printf.h"
#include <xparameters.h>
#include <xil_cache.h>
#include <stdlib.h>
#include <string.h>.
#include <xconvert_to_caractere.h> | #include <xconvert_texte_binaire.h>
```

Where the last 2 libraries are for introduce the IP core functions to convert binary to character for decryption algorithm and convert text to binary for encryption case.

We built the whole program for both encryption and decryption on file.c and then we have programed the FPGA board to reside the bitstream and the ELF files in BRAM memory for each case (encryption, decryption).

To program the FPGA board we clicked on the button program FPGA from the Toolbar (Figure III.20), here the bitstream with the design needed will be programed on the FPGA board.

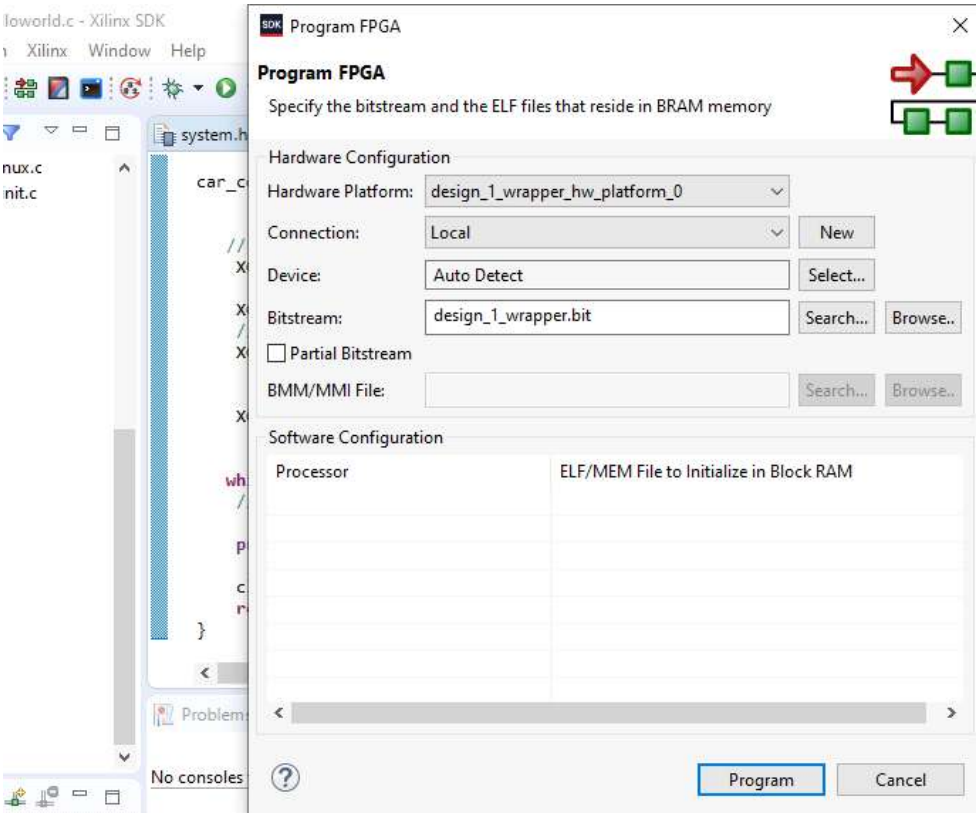


Figure III.20 tab to program FPGA doard

When finished we have run the file desired (encryption/decryption) clicking on:

Run As ->Launch on Hardwar (system debugger).

And the result appeared on the terminal "in our project we used Hercules".

We did an example which is about the encryption and the decryption of our Names "**BELGACEMI BENSABER**" where the agreed DNA string was a part of Fennec DNA "TGCTATCCCCTATATCGGAACCGATCTAGTAGAATGAATTTGAGGGGGCTTCTCA GTAGACATAGTAGAAGCGGGTGCGGGAACTGGGTGAACCGTATATCCCCCATTA GCTGGTA", and the result obtained for the Encryption and Decryption are shown on the figures (Figure III.21, Figure III.22):



Figure III.21 encryption results

Explanation of the results display:

**: The text desired to encrypt.

***: representation of the transfer of text to serial codons.

****: from codons to character.

*****: the binary representation.

******: the shifted binary representation.

*******: from binary to nucleotide.

********: the shifted nitrogenous series representation.

Le texte chiffre est: Represent the ciphertext.

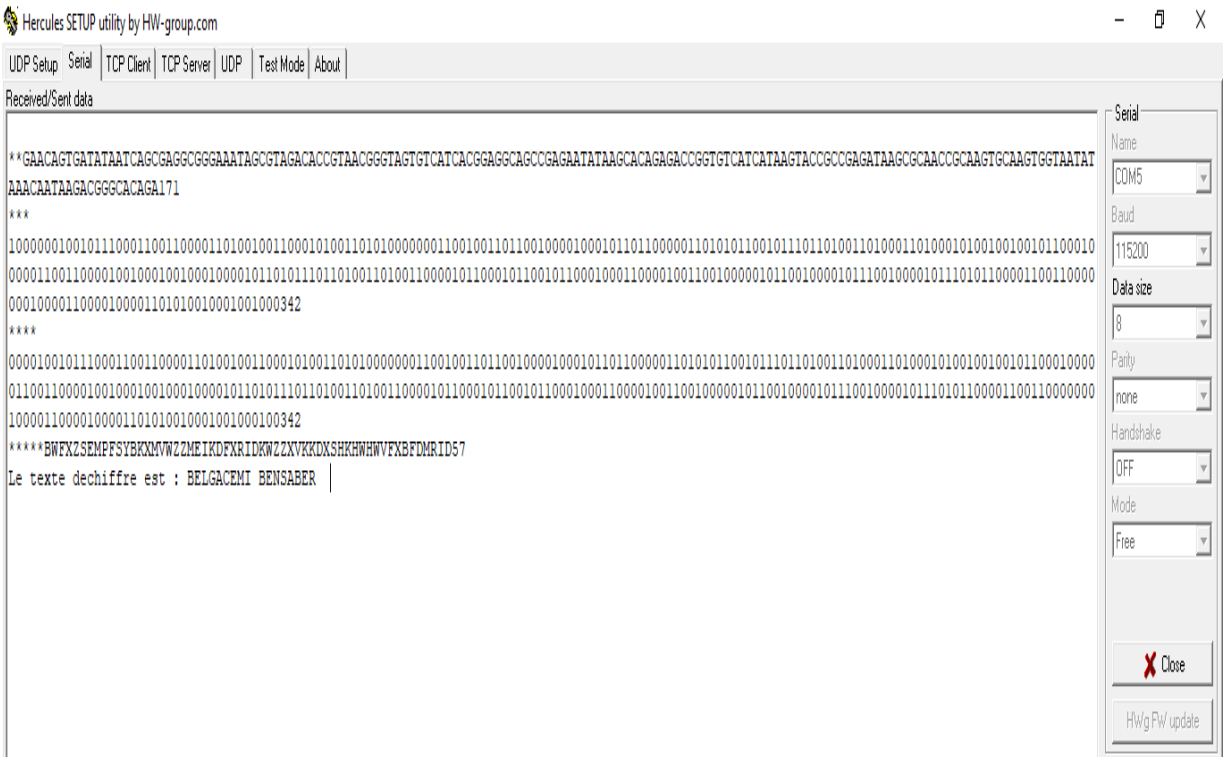


Figure III.22 decryption results

Explanation of the results display:

**: the nitrogenous series representation from the cipher text.

***: is the binary representation of the nitrogenous series.

****: the shifted binary representation.

*****: represent the character form "from binary to character".

Le texte dechiffre est: represent the result which is the deciphered text "original text".

III.5 Conclusion

In this chapter we have presented a hardware design of the encryption-decryption function of the ADN-Encryption algorithm .that allowed us to encrypt texts. During this chapter, we have developed two different algorithms, the first for encryption and the second for decryption where the synthesized was on HLS which takes functions from software to hardware IP core. Finally, an execution was performed using a ZYBO board.

General conclusion

ADN encryption have proven their efficiency through the 5 functions followed to create the Algorithm to secure information and hide the real meaning, in this dissertation we have shown how it's complicated and secured algorithm.

The HLS is a great choice for develop an IP core using C-Algorithm, we demonstrated how it could create a practical design and synthesize which is useful on FPGA board.

We have shown how to proceed this implementation using C and synthesize on HLS step by step using the ZYBO board from Xilinx board.

In this work we focused on the part of transferring software functions to hardware "IP core". It was just for specified functions because there is a lot of complexity to make a program on HLS, it doesn't include all the desired C program functions, so we have prepared some functions for encryption and others for decryption to show how and why the use of HLS, but it's not going to stop here, our next destination is to transfer all the functions and solve all errors that appear to have an IP core which contain all the encryption and decryption functions.

References bibliography

1. **JASHKOTHARI1 *Cryptography and its Types*. 2020.**

2. **Lamas, D., *La cryptographie*, 2015, Haute école de gestion de Genève.**

3. **Kahn, D., *The Codebreakers: The comprehensive history of secret communication from ancient times to the internet*. 1996: Simon and Schuster.**

4. **Onwutalobi, A.-C., *Overview of Cryptography.* Available at SSRN 2741776, 2011.**

5. **de Dieu Nkapkop, J., J.Y. Effa, and L. Bitjoka, *Design et implémentation d'un crypto-système d'images basé sur le chaos*. 2018: Editions universitaires europeennes.**

6. **Mahdi, H., *étude et comparaison des principaux systèmes de cryptage et les technique y afférentes*, 2016, FACULTE DES MATHEMATIQUES ET-UNIVERSITE MOHAMED BOUDIAF-M'SILA.**

7. **Masuda, N. and K. Aihara, *Cryptosystems with discretized chaotic maps.* Ieee transactions on circuits and systems i: fundamental theory and applications, 2002. 49(1): p. 28-40.**

8. **Kerckhoffs, A., *La cryptographic militaire.* Journal des sciences militaires, 1883: p. 5-38.**

9. **Shannon, C.E., *Communication theory of secrecy systems.* The Bell system technical journal, 1949. 28(4): p. 656-715.**

10. **Zouhir, D., *Implémentation d'un algorithme de cryptage sur un circuit FPGA*, 2017, Faculte de technologie/UniversiteMohamed Boudiaf-M'sila.**

11. **Amine, F.M., *Etude et Implémentation Pipeline sur FPGA de***

***L'algorithme de Chiffrement AES* 2015, Université des Science et Technologie d'Oran**

Mohamed Boudiaf

12. **Maqsood, F., et al., *Cryptography: A comparative analysis for modern techniques.* International Journal of Advanced Computer Science and Applications, 2017. 8(6): p. 442-448.**

13. **Rivest, R.L., A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 1978. 21(2): p. 120-126.**

14. **Bernabé, G. and N. Clarke, *Study of RSA Performance in Java Cards.* Advances in Communications, Computing, Networks and Security Volume 10, 2013: p. 45.**

15. **Anoop, M.S., *Elliptic Curve Cryptography.* Infosecwriters, 2015: p. 1-11.**

16. Kaundal, A.K. and A. Verma, *DNA based cryptography: a review.* International Journal of Information and Computation Technology, 2014. 4(7): p. 693-698.

17. Joseph, K., *Molecular Genetics, Biology, DNA Technology And Fingerprinting Structure Kindle Edition.* 2020.

18. Crick, J.D.W.a.F.H.C., *,Molecular Structure of Nucleic Acids: A Structure*

for Deoxyribose Nucleic Acid. Nature, 1953: p. 171

19. Jeevidha, S., M.S. Basha, and P. Dhavachelvan, *Analysis on DNA based cryptography to secure data transmission.* International Journal of Computer Applications, 2011. 29(8): p. 16-20.

20. Miloud, C. and B. Mohammed, *La cryptographie a base d'ADN :*

Application sur les images, 2017/2018, Université Dr Tahar Moulay - Saïda. p. p.30-34

21. Imane, M. and B. Kadda, *Cryptographie à base d'ADN: Un crypto-système à clé symétrique basé sur les séquences ADN.*

22. Crick, J.D.W.a.F.H.C., *Molecular Structure of Nucleic Acids: A Structure*

for Deoxyribose Nucleic Acid. Nature

1953.

23. Head, T.J., N. Jonoska, and G. Paun, *Aspects of molecular computing: essays dedicated to Tom Head on the occasion of his 70th birthday*. 2004: Springer Science & Business Media.

24. Namasudra, S., *Taxonomy of DNA-based security models*, in *Advances of DNA computing in cryptography*. 2018, Chapman and Hall/CRC. p. 37-52.

25. ; Available from:

www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

26. 2020

accessed March.19]; Available from: www. EMBL.com.

27. 2020 Accessed March.

19]; Available from: www.GENBANK.com

28. Railis, K., *Integrating high-level synthesis derived hardware accelerators on an FPGA-based SoC: evaluation and analysis of design alternatives.* 2016.

References bibliography

29. Amar, G. *Les circuits FPGA : description et applications*.

, 2012. p. 1.

30. Mohamed, F. and Z. Abdelhak, *Etude et implémentation d'un réseau de neurones*

***sur une plateforme de type Pynq.*, 2019, UNIVERSITE YAHIA FARES DE MEDEA. p. 63.**

31. Touzet, C., *les réseaux de neurones artificiels, introduction au connexionnisme*. 1992: Ec2.

32. *Field Programmable Gate Array (FPGA)*

Available from: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html.

33. xilinx, *Zynq-7000 SoC Data Sheet: Overview*, 2018.

34. *Basic HLS Tutorial*

2018 12/19]; Available from: www.so-logic.net.

Annex

The decryption case where this program represent the use of IP core to convert from binary to character.

```
#include "platform.h"

#include "xil_printf.h"

#include <xparameters.h>s

#include <xil_cache.h>

#include <stdlib.h>

#include <string.h>

#include <xconvert_to_caractere.h>

char*
code_binaire_decale="0000100101110001100110000110100100110001010011010100000001100100110110010000100010111011000001101010110010111011010011010001101000101001001001011000100000110011000001011010111010011011010010111001011001011011000000100001101001001000010000001000000100101100010110101110101110101110101110101101100001100110000001000100000001000100000010110101100000100110100010010110010000110010000010";

XConvert_to_caractere car;

XConvert_to_caractere_Config* car_conf;

int main()

{

   init_platform();

   Xil_DCacheDisable();

   char* tab_caractere=(char*)malloc(strlen(code_binaire_decale)/6);
```

```
car_conf=XConvert_to_caractere_LookupConfig(XPAR_CONVERT_TO_CARACTERE_0_
DEVICE_ID);

    XConvert_to_caractere_CfgInitialize(&car, car_conf);

    XConvert_to_caractere_Set_tab(&car,(u32)code_binaire_decale);

    XConvert_to_caractere_Set_resultat(&car,(u32)tab_caractere);

    XConvert_to_caractere_Start(& car);

   while(!XConvert_to_caractere_IsDone(&car));

    printf("\n******%s",tab_caractere);

    cleanup_platform();

    return 0;

}
```

The encryption case where this program represent the use of ip core for convert character to binary:

```
#include <stdio.h>

#include "platform.h"

#include "xil_printf.h"

#include <xparameters.h>

#include <xil_cache.h>

#include <stdlib.h>

#include <string.h>

#include <xconvert_texte_binaire.h>


char*texte_chiffre="XVPXZSEMPFXYBRXMIHYZPEIKDMDKWBRWZKSWKKXDBH
RWWHIHMBD";
```

```
XConvert_texte_binaire_Config* bin_conf;

XConvert_texte_binaire bin;

int main()

{

   init_platform();

   Xil_DCacheDisable();

   char* message_chiffre_binaire=(char*)malloc(strlen(texte_chiffre)*6);

bin_conf=XConvert_texte_binaire_LookupConfig(XPAR_CONVERT_TEXTE_BINAIRE_0
_DEVICE_ID);

   XConvert_texte_binaire_CfgInitialize(&bin,bin_conf);

   XConvert_texte_binaire_Set_texte_chiffre(&bin, (u32)texte_chiffre);

   XConvert_texte_binaire_Set_message_chiffre_binaire(&bin,
(u32)message_chiffre_binaire);

   XConvert_texte_binaire_Start(&bin);

   while(!XConvert_texte_binaire_IsDone(&bin));

   printf("\n*****%s\t%d",message_chiffre_binaire,strlen(message_chiffre_binaire));

   cleanup_platform();

   return 0;

}
```