Implementation of an FPGA Based Accelerator for Virtual Private Networks

CHEUNG Yu Hoi Ocean (B. Eng.)

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

July, 2002


香港中文大學
大學
圖書館系統
07 MAR 2003
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

Abstract

Virtual Private Networks (VPN) are becoming increasingly popular network architectures for corporate networks. They enable corporations to connect Local Area Networks (LAN) in main and branch offices as if they were in the same network. As VPNs are built on the Internet infrastructure, the data exchange among different local area network will be passed through the Internet and thus can be easily eavesdropped, masqueraded, etc. Therefore, certain security measures must be used to deal with these privacy issues.

The Internet Protocol Security (IPSec) by the Internet Engineering Task Force (IETF) addresses the above mentioned security issues. A project called the Free Secure Wide Area Network (FreeS/WAN) was developed to provide an open source IPSec based VPN solution. This application use Triple-DES as default encryption mode for IPSec. Results show that the bottleneck in FreeS/WAN comes from encryption and decryption of the data.

As shown in this dissertation, the performance of FreeS/WAN with IPSec is 50% of that without FreeS/WAN. In order to improve performance of encryption, field programmable gate array (FPGA) based accelerators were built on a reconfigurable computing development platform called Pilchard. An implementation of Triple-DES on Pilchard was built to replace the current Triple-DES software based library (LibDES) used in FreeS/WAN. To compare performance of Triple-DES with that of another cipher, a Pilchard based accelerator for the International Data Encryption Algorithm (IDEA) was developed.

The resulting implementations achieved 120 Mb/sec for Triple-DES in CBC

mode and 248 Mb/sec for IDEA in ECB mode. These ciphers were used as a new cryptographic library for FreeS/WAN. Measurements show that this FPGA-based FreeS/WAN offers a 30% speedup on Triple-DES CBC mode over the original software library.

現場可編程門陣虛擬私有網路的加速器

作者 張如海

摘要

虛擬私有網路(VPN)正成爲商業機構愈來愈普遍採用的網路結構。商業機構可以利用虛擬私有網路將在總部和分部的局域網(LAN)連成一體。因爲虛擬私有網路架設在互聯網上,不同的局域網之間需要通過互聯網來進行資料交換。所以必須使用某些安全措施來處理這些保密性問題。

爲解決上述安全問題, 由互聯網工程小組(IETF)提出了互聯網安全協定(IPSec)。FreeS/WAN 的開發提供了一種在使用互聯網安全協定和開放來源碼的 VPN 應用程式。這應用程式使用三重數據加密標準(Triple-DES) 作爲 IPSec 的主要加密方式 。其結果顯示在 FreeS/WAN 的瓶頸來自於資料的加密和解密。

在這份學術論文中, FreeS/WAN 的性能表現爲沒有 FreeS/WAN 50%。爲了改進加密的性能, 在可重構的發展平臺-Pilchard 上發展了現場可編程門陣 (FPGA) 的加速器。三重數據加密標準在 Pilchard 上實施以取代在 FreeS/WAN 的三重數據加密標準的軟體庫(LibDES) 。

三重數據加密標準的加速器在 CBC 方式下性能可達 120 Mb/sec。而國際數據加密演算法(IDEA) 的加速器在 ECB 方式下可達 248 Mb/sec。這些加速器成爲 FreeS/WAN 的一個新的加密庫。測量結果顯示使用 FPGA 加速器的 FreeS/WAN 應用程式在三重數據加密標準下的性能比原來使用的軟體庫提升 30% 。

Acknowledgments

This thesis would not be possible to completed without help of many people. I would like to take this opportunity to thank them.

Firstly, I would like to thank my final year project and Master Degree supervisor, Prof. Leong Heng Wai Philip, for his guidance and encouragement in the past two years. He show his generosity and gave me numerous ideas for my research work.

I would like to thank Prof. Lee Kin Hong and Prof. Wei Keh Wei Victor for suggestions and comments for improving this work.

I would like to specially thank Mr. K.H. Tsoi and Mr. M.P. Leong. They gave me a lot of advices in completing this thesis.

I would like to thank my colleagues, Mr. C.K. Fung, Mr. C.H. Ho, Mr. Y.M. Lam, Ms. Ng Anny, Mr. C.W. Ng, Mr. C.W. Sham and Mr. W.C. Wong for their help in my research and bring me a pleasant working atmosphere.

I would like to thank my family for their endless support. This thesis is dedicated to my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In a private network of a business, information and resource are shared. Information flow is very important nowadays, for example, the operational costs of a business can be cut down if a better supply chain model is employed. The business can improve its services by sharing information internally and with its business partners. Also due to the globalization of business environments, corporations have offices all over the world. The different geographical locations make connections among different private networks difficult.

The Virtual Private Network is an architecture to realize the connections among different private networks over a public network. For example, the Internet can be used as a convenient and low cost channel for a virtual private network. Internet is a public channel and is not secure. Cryptographic algorithms can provide a way to secure channel between private networks over Internet.

Field-Programmable Gate Arrays (FPGAs) are hardware devices which are reconfigurable, i.e. programming an FPGA can change its functionally. Implementations of cryptographic hardware using FPGAs offer higher performance than software implementations. Software implementations of cryptographic algorithms are sequential in nature. However, in cryptographic hardware, algorithms can execute in parallel, offering a more efficient implementation. There are several advantages

to use FPGAs as the choice of hardware device for a virtual private network accelerator:

- most network applications offer various encryption standards as options. With FPGAs, it is possible to reconfigure the chip for different encryption standards.
- FPGAs offer lower costs for small volumes, shorter development times and faster time to market over application specific integrated circuit (ASIC) technology.
- the technology and capacity of FPGAs continue to improve over previous years. The performance of FPGA accelerator can be improved once a faster device is available without any further engineering.

1.2 Aims

The main aim of this work was to develop an FPGA based accelerator for Virtual Private Networks. The following features were desired.

- develop a hardware accelerator which is integrated into a real network application.
- design various cryptographic hardware accelerators to widen the choice of algorithm.
- devise a hardware interface which is fully compatible with an existing software cryptographic library for usage in other applications.
- provide a high performance hardware accelerator for Triple Data Encryption Standard in Cipher-Block Chaining mode and the International Data Encryption Algorithm (IDEA) in Electronic Code Book (ECB) mode by using a new reconfigurable hardware environment - Pilchard.

1.3 Contributions

This thesis presents a FPGA based cryptographic accelerator for virtual private network. The work presented in this thesis has the following features that distinguishes it over all previous designs:

- a study of tradeoffs in parallel and serial implementations of the International Data Encryption Algorithm was made [CTLL01]. In this work, the bit-serial implementation of IDEA was implemented by M.P. Leong [LCTL00]. In the bit-parallel implementation of IDEA, the pipelined IDEA core was my work and the control section was implemented by K.H. Tsoi.

- improvements to the device driver for the the Pilchard reconfigurable hardware environment were made in order to improve the bandwidth between the PC and the FPGA.

- a high performance cryptographic accelerator was integrated in a real VPN application and its performance measured. Although hardware based cryptographic accelerators (summized in Section 2.6) exist in commercial products, to the best of my knowledge, detailed reports of their design and performance have not been published.

1.4 Thesis Outline

Background information concerning virtual private network are presented in Chapter 2. Chapter 3 provides a description of previous work on the IDEA and DES algorithms as well as their implementation in hardware. Also the tools and reconfigurable hardware that were used in this research are introduced. Chapter 5 introduce the architectural details of an FPGA based Virtual Private Network. Chapter 6 contains the results and benchmarks for this research. In Chapter 7, conclusions and further directions for work are given.

Chapter 2

Virtual Private Network and FreeS/WAN

2.1 Introduction

In this chapter, background knowledge about virtual private networks and a VPN solution - FreeS/WAN are given. This chapter begins with a brief introduction to Virtual Private Networks using IPSec. A section discussing the Internet Protocol Security protocol (IPSec) is included, followed by information about LibDES which is a popular software cryptographic library. Finally, the details about FreeS/WAN is given.

2.2 Internet Protocol Security (IPSec)

IP packets are not secure over the Internet. It is trivial to fake the identity of an IP address, modify the content of packets, replay packets and intercept packets. In addition, we cannot guarantee that IP packets received are either coming from the original source or that the content is the original content.

Therefore, the IPSec protocol [KA98c] was introduced to solve the following problems:

1. Eavesdropping : an adversary eavesdrops on the Internet, capturing data packets, e.g. credit card numbers, login names and passwords, etc. can be obtained by eavesdropping. Using the IPSec protocol, data traffic is encrypted so that it is difficult to obtain useful information by eavesdropping.

2. Masquerading : an adversary fakes his IP address to masquerade as a trusted host when address-based authentication is used. The IPSec protocol provides a cryptographic authentication method to protect against masquerading. With the IPSec Authentication Header Protocol, a receiver can make sure the source of data is as claimed.

3. Session hijacking : an adversary takes over a connection after the source has been authenticated. This scenario will not occur with IPSec protocol since the adversary have no knowledge about the session keys, which is negotiated in the IPSec Internet Key Exchange protocol, so they cannot encrypt or decrypt the data packets.

4. Denial-of-service : An adversary sends a huge sequence of connection requests to the target in order to make the target system overflow the buffer space. For example, email bombing, TCP SYN flooding, etc. Although this kind of attack is still possible with the use of IPSec protocol, the adversary will expose his real IP address due to the fact that all data packets should be properly authenticated.

The IPSec protocol provides three functionalities using three different protocols:

- the Authentication Header (AH) protocol
- the Encapsulating Security Payload (ESP) protocol
- the Internet Key Exchange (IKE) protocol

The Authentication Header (AH) protocol [KA98a] can authenticate the source of data packets, protect the completeness of the data packets and protect against

replay attacks. ESP (Encapsulation Security Payload) protocol [KA98b] shares all properties that AH has and it can also protect data from unauthorized disclosure and provide protection against traffic flow analysis. The security provided by IPSec needs to use shared keys in order to authenticate and encrypt the data streams. The Internet Key Exchange (IKE) protocol [HC98] is used to dynamically authenticate parties involved in IPSec, negotiate the encryption method used, and produce shared keys.

2.3 Secure Virtual Private Network

In this work, the term Virtual Private Network (VPN) [DH99] is use to refer to the architecture that a private network constructed within a public network infrastructure. The connection of this network architecture can be either encrypted or unencrypted. Also it is possible to implement Virtual Private Networks on other networks, in this project, the Internet is assumed. IPSec is one of the protocol that may use in VPN architecture that provide security and privacy by cryptographic algorithms and hash functions. Another term, Secure Virtual Private Network (SVPN) refers to a VPN with IPSec.

The key concept of Virtual Private Network is tunneling. With tunneling, VPNs provide connection and protocol transparency among different Intranets of same organization or even different parties. For connection transparency, different parties are connected together as if they were on the same network. They do not need to know the mechanism and the details of connections. For protocol transparency, different parties using different protocol can be connected together as if they were using the same protocol. This is achieved by encapsulation of data packets at the end-points of a tunnel with a different protocol.

Although there are many advantages associated with using a VPN over the Internet, the data would be transmitted in plaintext over an insecure channel. The Intranet of a party is exposed to attacks from Internet which violates the major aim

of VPN to be a private network.

There are two major in common protocols for VPNs.

1. IPSec
2. PPTP

PPTP is the protocol used in Microsoft's VPN product and is proprietary. Thus, this work will focus on IPSec which will documented by the IETF in RFCs [iet].

Internet Protocol Security (IPSec) is a protocol proposed to solve the concerns about security and privacy. IPSec provide mechanisms to authenticate different parties and data packets are protected by encryptions.

To summarize here are some VPN characteristics:

1. The VPN uses the Internet as the underlying public network infrastructure.
2. The VPN uses the IPSec as protocol to ensure privacy at the network layer. This means encryption is done as per every packet.
3. Private addressing schemes can be used in Intranets and IP address is only used on communication of end-points of tunnel.

As suggested by most vendors of VPN solutions [vpnc], there are three scenarios that should be deal with in order to meet the requirements of a business. Three different types of users should be able to grant access to the VPN of a corporation. They are remote users, branch offices and business partners.

1. Remote access network - A remote user at home or on road needs access to his/her company's resources. The VPN should enable the remote users to work as if (s)he was at a workstation in the office. Different connections method should be provided in order to achieve the remote access into the network, e.g. dialup, ISDN, mobile IP, etc.

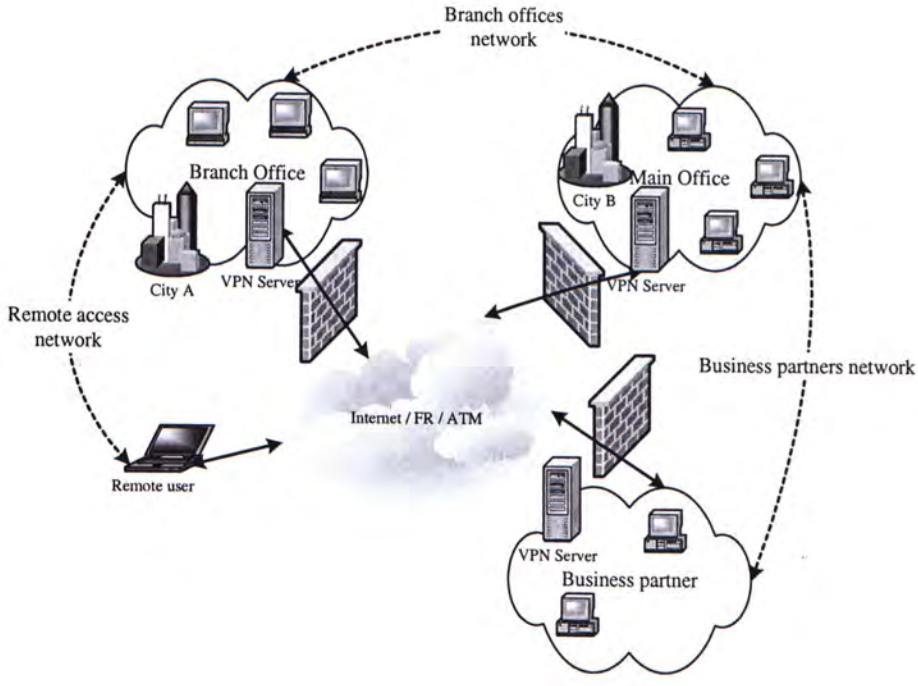

Figure 2.1: Virtual Private Network

2. Branch offices network - Two or more trusted Intranets, which represent different branch and remote offices of a corporation, are interconnected together by a VPN. Very often, Intranets are protected by firewalls which can act as secure gateway connect to the Internet. Client workstations do not have to worry about the security between Intranets since this is ensured by the VPN.

3. Business partners network - This is referred to as an Extranet by many VPN solution vendors. It should be the most recent trend for VPN usage; however, it is the scenario with least knowledge. Corporations can grant their business partner temporal and limited access to their Intranet. Electronic business applications among business partners include online quotations, order fulfillment, etc.

2.4 LibDES

LibDES is a publicity available software library for DES and Triple-DES, written by Eric Young [lib]. It offers a large variety of highly optimized DES and Triple-DES functions in different modes. For example, DES in Electronic Codebook Mode (ECB), DES in Cipher Block Chaining Mode (CBC), 3DES in ECB mode, 3DES in CBC mode, etc. LibDES is a common standard library which is used in various application such as openSSL.

2.5 FreeS/WAN

In this work, the baseline software used for the implementation of the VPN accelerator was FreeS/WAN. FreeS/WAN [Nap00, Fre00] stand for Free Secure Wide Area Network. FreeS/WAN is currently the most complete open source VPN solution available on Linux. In here, the version of FreeS/WAN used in this project is 1.5. It is currently built for Linux IPv4 network stack and work has commenced to integrate into the IPv6 network stack.

FreeS/WAN supports both remote access network and branch office network, however, it does not support business partner networks because the software does not have any mechanism for temporal and limited access to network.

LibDES is used in FreeS/WAN as the DES and Triple-DES library. In LibDES, DES and triple-DES in different modes are performed in software. Replacing the software DES with a hardware based implementation is the main focus of this work. The version of LibDES used in FreeS/WAN v1.5 is version 4.04.

2.6 Commercial VPN solutions

There are commercial VPN solutions using either software or hardware implementations for different cryptographic algorithms. Although different solutions may

have different build-in cryptographic algorithm options, Triple-DES is available for all VPN solutions. Performance of VPN solutions using Triple-DES from different vendors is compared in Table 2.1 and Table 2.2.

Cisco Systems Inc. has a wide range of VPN solutions with different specifications. Cisco 3015 uses software encryption method and hence a relatively low throughput of 4 Mb/sec is obtained. In Cisco 5000 series VPN solutions, different numbers of encryption processors can be used. For the highest throughput VPN solution in this series, 760 Mb/sec is achieved by using eight encryption processors. Intel provides two VPN solutions using software encryption with throughputs of 8 Mb/sec and 20 Mb/sec. They also have a VPN solution using a PCI encryption processor with a throughput of 85 Mb/sec.

Vendor	VPN Solution	Maximum throughputs (Mb/sec)	Reference
Cisco	Cisco 3015	4	[vpna]
Intel	Intel 3110 VPN gateway	8	[vpne]
Intel	Intel 3105 VPN gateway	20	[vpnd]

Table 2.1: Comparison of VPN solutions using software encryption

Vendor	VPN Solution	Maximum throughputs (Mb/sec)	Scalability	Reference
Cisco	Cisco VPN 5001	45	1	[vpnb]
	Cisco VPN 5002	190	2	[vpnb]
	Cisco VPN 5008	760	8	[vpnb]
Intel	Intel 3125 VPN gateway	85	1	[vpnf]

Table 2.2: Comparison of VPN solutions using hardware encryption

2.7 Summary

In this chapter, virtual private networks and the details about FreeS/WAN were discussed. The Virtual Private Network is an architecture to connect two separate LANs over a public network. The Internet is the most popular choice as the channel due to its accessibility and cost. Since the Internet is insecure, IPSec is used to deal with privacy issues. FreeS/WAN is an open source VPN solution using IPSec on Linux.

Chapter 3

Cryptography and Field-Programmable Gate Arrays (FPGAs)

3.1 Introduction

This chapter introduces the basic concepts of cryptography and Field-Programmable Gate Arrays. Firstly, DES, Triple-DES and IDEA algorithms and previous implementations are introduced. This is followed by description of different block cipher modes of operation. The architecture of FPGAs is discussed, in particular, information on the architecture of Xilinx Virtex-E FPGAs are given. The details concerning the reconfigurable computing environment, Pilchard, is then presented. Finally, Electronic Design Automation Tools and the FPGAs design flow is detailed.

3.2 The Data Encryption Standard Algorithm (DES)

The Data Encryption Standard (DES) [Nat94, Uni77] algorithm has been a popular secret key encryption algorithm and is used in many commercial and financial applications. Also, it was the first commercial cryptographic algorithm with fully

specified implementation details. It is defined by ANSI FIPS46-2. Although introduced in 1976, it has proved resistant to all forms of cryptanalysis. However, its 56-bit key is not large enough by today's standards. A DES key search engine called "Deep Crack" could search 88 billion keys per second and this machine solve RSA laboratories DES-III challenge [RSA99] on January 1999 in 22 hours.

DES is a block cipher as shown in Figure 3.1 which processes 64-bit plaintext blocks and produces 64-bit ciphertext blocks. The effective portion of the secret key is 56-bit out of 64-bit since although the key is 64-bit, 8-bits are used as parity bits.

Encryption of DES proceeds in 16 identical rounds. From the input key, sixteen 48-bit subkey K_i are generated, one for each round. Within each round, 8 fixed 6 to 4-bit substitution mappings known as S-Boxes are used.

The plaintext have an initial bit permutation (IP) as shown in Table 3.1 and are then divided into left L_0 and right halves R_0, each 32-bit. Each round takes 32-bit inputs L_{i-1} and R_{i-1} from previous rounds and produces 32-bit outputs L_i and R_i for $1 \leq i \leq 16$, as follows:

$$\begin{aligned} L_i &= R_{i-1} \\ L_i &= L_{i-1} \oplus f(R_{i-1}, K_i), where f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i)) \end{aligned}$$

E is a fixed expansion permutation mapping R_{i-1} from 32-bit to 48-bit. P is another fixed permutation on 32-bits. The equation shows the right half of each round go through an expansion permutation from 32-bits to 48-bits and is then exclusive-ored with the subkey of that round. The temporary result is passed through the S-Box and forms the new 32-bit product of the right half. For each round, right half and left half are exchanged. Finally both halves are combined together in the 16th round and permutated by the inverse of the initial bit permutation shown in Table 3.2 to form the ciphertext. Decryption uses the same key and algorithm, however, the subkeys in internal rounds are applied in reverse order. For encryption, the key schedule order is $K_1, K_2, K_3, \ldots, K_{16}$. For decryption, the decryption key

schedule is $K_{16}, K_{15}, K_{14}, \ldots, K_1$.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Table 3.1: Initial bit permutation (IP)

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	157	25

Table 3.2: Inverse of initial bit permutation (IP^{-1})

3.2.1 The Triple-DES Algorithm (3DES)

Triple-DES algorithm [Nat99] was introduced to increase the the key size of DES and maintaining compatibility with legacy DES software and hardware systems. For encryption, the plaintext is processed by three cascaded DES cores as shown in Figure 3.2, the first and the last DES cores are in encryption mode and the middle one is in decryption mode. If the same key is used for K_1 and K_2, Triple-DES is the same as DES with key K_3. For decryption of Triple-DES, the modes of three cascaded DES cores are inverted so that the first and the last DES cores are in

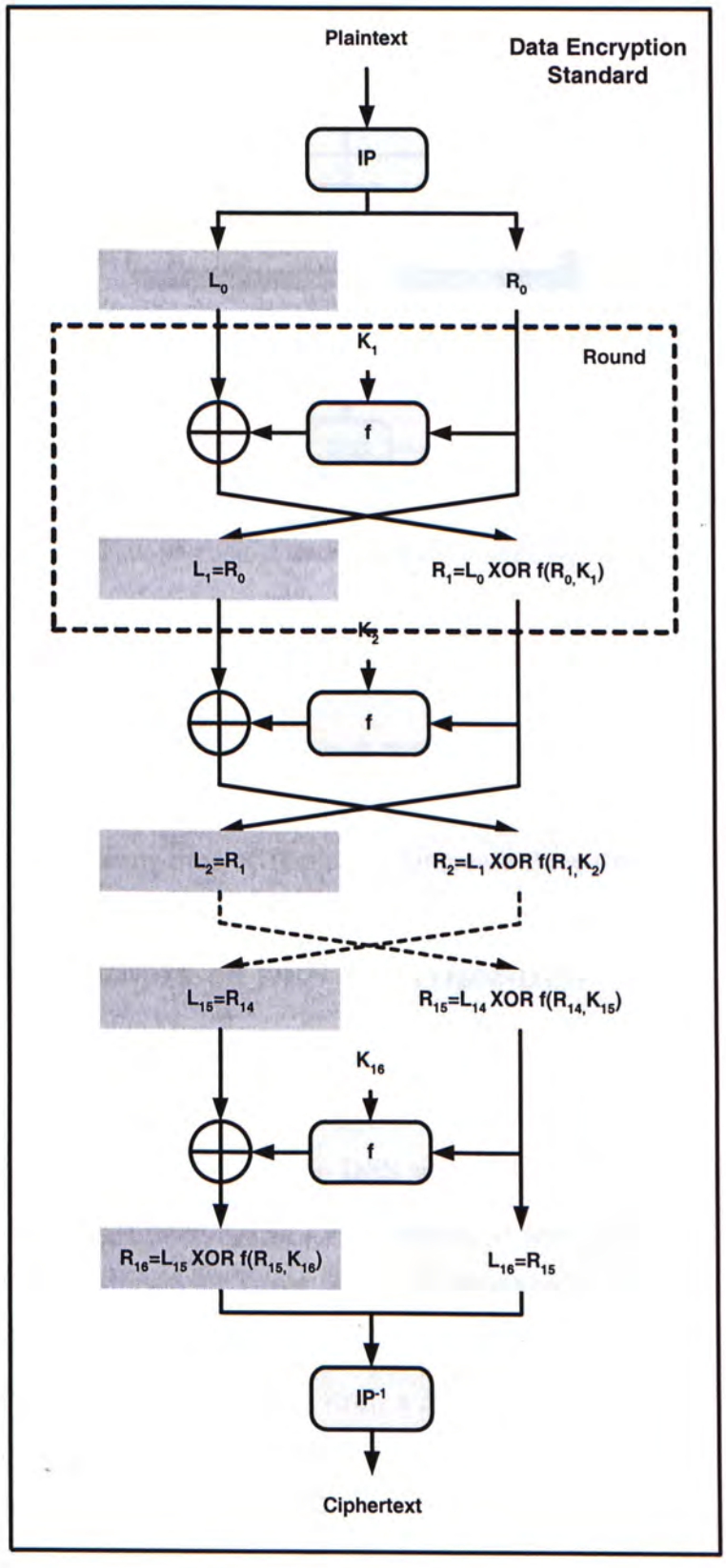

Figure 3.1: Data Encryption Standard algorithm

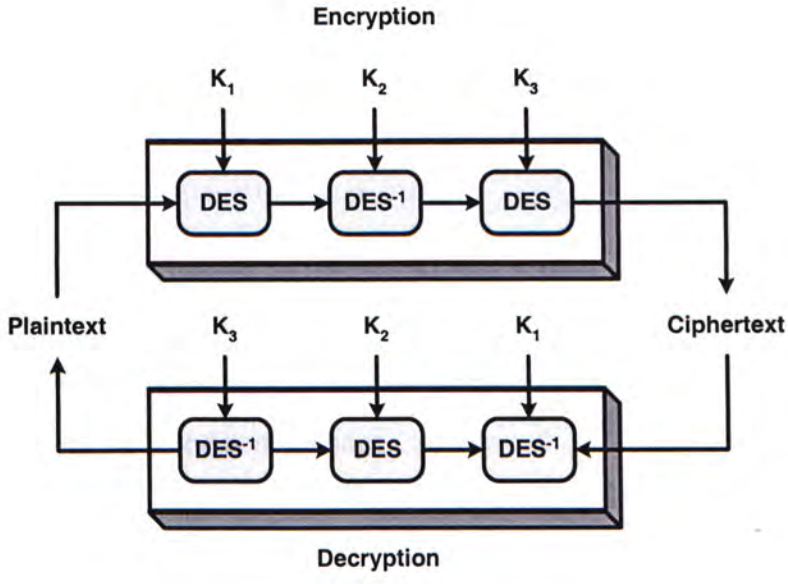

Figure 3.2: Triple-DES algorithm

decryption mode and the middle one is in encryption mode. Triple-DES algorithm increase the key size three times compared to DES, which is from 56-bit to 168-bit. However, the processing time of Triple-DES increase three times as well.

3.2.2 Previous work on DES and Triple-DES

A software implementation of DES and Triple-DES by Biham in 1997 in ECB mode achieved 46 Mb/second 22 Mb/sec respectively on an 300 MHz Alpha, which is a 64-bit processor . The most common DES software LibDES [lib] achieves 121.5 Mb/sec for DES ECB mode on an Intel Pentium III 866 MHz machine. LibDES also achieves 42.9 Mb/sec for Triple-DES CBC mode on the same machine.

Hardware implementations offer much higher performance than DES software implementation. In 1999, Free-DES [fre], a 3656 Mb/sec implementation of DES algorithm on Xilinx Virtex XCV400-6 with 60 MHz clock rate was reported. A 1280 Mb/sec implementation of IDEA was reported in 1999 [WPR+99] by Wilcox et. al. The Sandia National Laboratories developed an ASIC implementation of DES [WPR+99]

which achieves 6700 Mb/sec. The fastest hardware DES hardware implementation [Pat00] is proposed by Patterson which achieves 10752 Mb/sec. This implementation fully unrolls and pipeline the DES rounds and operates at a 168 MHz clock rate. It employs dynamic circuit specialization in an FPGA to achieve high performance.

Previous high performance implementation of DES in hardware fully maximize their throughput by unrolling and pipelining the design in Electronic Code Book mode (see Section 3.6). Due to the data dependencies, pipelined DES implementations cannot have the same performance.

Year	Implementation	Throughput (Mb/sec)	Reference
1997	software	121.5	[fre]
1999	Xilinx Virtex XCV400-6	3656	[fre]
1999	4 $\times$ Altera 10K100	1280	[WPR+99]
1999	ASIC 0.6 μm CMOS	9280	[WPR+99]
2000	Xilinx Virtex XCV150-6	10752	[Pat00]

Table 3.3: Comparison of DES implementations

3.3 The IDEA Algorithm

IDEA takes 64-bit plaintext inputs and produces 64-bit ciphertext outputs using a 128-bit key.

The design philosophy behind IDEA is mixing operations from different algebraic groups including XOR, addition modulo 2^{16}, and multiplication modulo the Fermat prime $2^{16} + 1$. All these operations work on 16-bit sub-blocks.

The IDEA block cipher [Sch96] (depicted in Figure 3.3) consists of a cascade of eight identical blocks known as rounds, followed by a half-round or output transformation. In each round, XOR, addition and modular multiplication operations are applied. IDEA is believed to possess strong cryptographic strength because

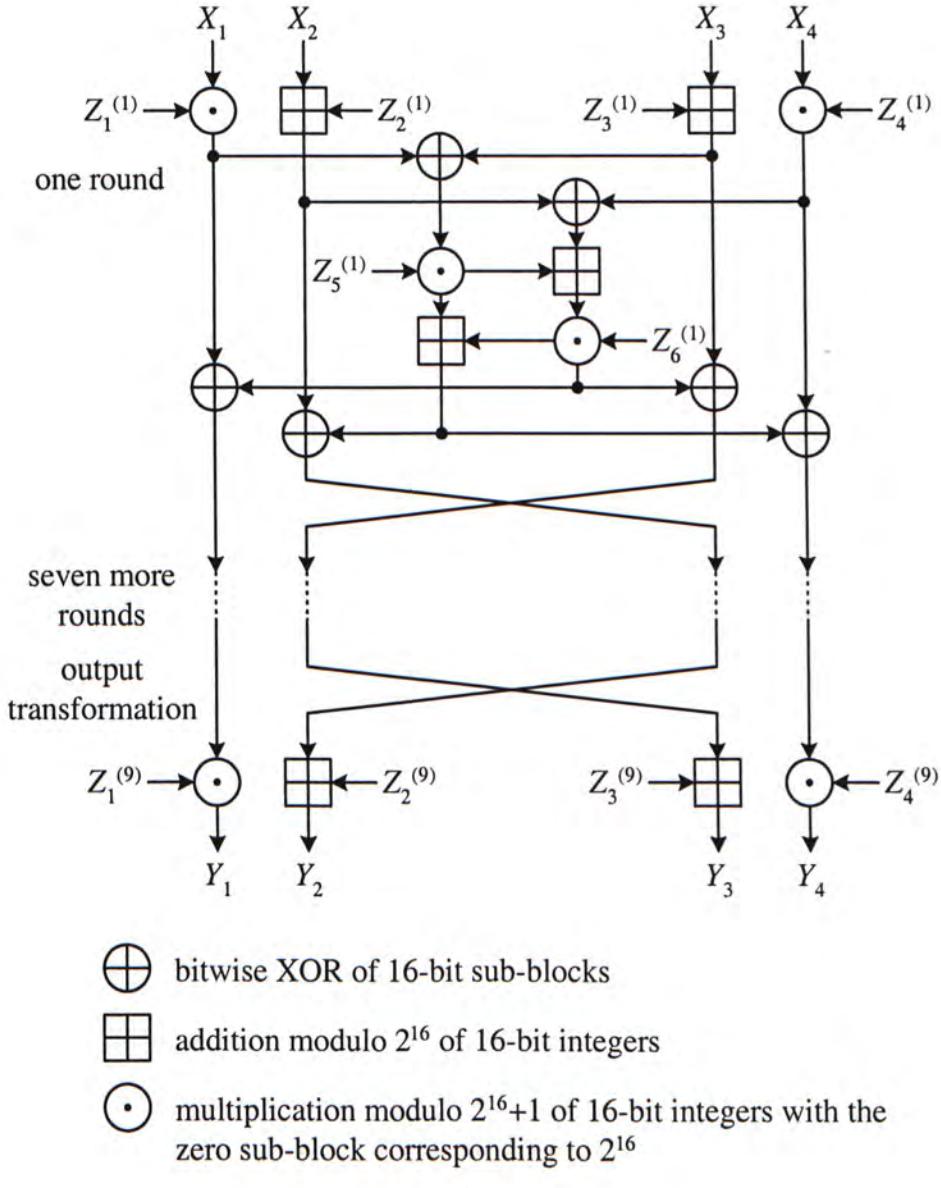

Figure 3.3: Block diagram of the IDEA algorithm.

- its primitive operations are of three distinct algebraic groups of 2^{16} elements
- multiplication modulo $2^{16}+1$ provides desirable statistical independence between plaintext and ciphertext
- its property of having iterative rounds made differential attacks difficult.

The encryption process is as follows. The 64-bit plaintext is divided into four 16-bit plaintext sub-blocks, X_1 to X_4. The algorithm converts the plaintext blocks into ciphertext blocks of the same bit-length, similarly divided into four 16-bit sub-blocks, Y_1 to Y_4. 52 16-bit subkeys, $Z_i^{(r)}$, where i and r are the subkey number and round number respectively, are computed from the 128-bit secret key. Each round uses six subkeys and the remaining four subkeys are used in the output transformation. The decryption process is essentially the same as the encryption process except that the subkeys are derived using a different algorithm [Sch96].

The algorithm for computing the encryption subkeys (called the key-schedule) involves only logical rotations. Order the 52 subkeys as $Z_1^{(1)}, \ldots, Z_6^{(1)}, Z_1^{(2)}, \ldots, Z_6^{(2)}, \ldots, Z_1^{(8)}, \ldots, Z_6^{(8)}, Z_1^{(9)}, \ldots, Z_4^{(9)}$. The procedure begins by partitioning the 128-key secret key Z into eight 16-bit blocks and assigning them directly to the first eight subkeys. Z is then rotated left by 25 bit, partitioned into eight 16-bit blocks and again assigned to the next eight subkeys. The process continues until all 52 subkeys are assigned. The decryption subkeys $Z'^{(r)}_i$ can be computed from the encryption subkeys with reference to Table 3.4.

In Electronic Codebook (ECB) mode [Sch96], the data dependencies of the IDEA algorithm have no feedback paths. Additionally, in practice, latencies of order of microseconds are acceptable. These features make deeply pipelined implementations possible.

	$r = 1$	$2 \leq r \leq 8$	$r = 9$
$Z'^{(r)}_1$	$(Z^{(10-r)}_1)^{-1}$	$(Z^{(10-r)}_1)^{-1}$	$(Z^{(10-r)}_1)^{-1}$
$Z'^{(r)}_2$	$-Z^{(10-r)}_2$	$-Z^{(10-r)}_3$	$-Z^{(10-r)}_2$
$Z'^{(r)}_3$	$-Z^{(10-r)}_3$	$-Z^{(10-r)}_2$	$-Z^{(10-r)}_3$
$Z'^{(r)}_4$	$(Z^{(10-r)}_4)^{-1}$	$(Z^{(10-r)}_4)^{-1}$	$(Z^{(10-r)}_4)^{-1}$
$Z'^{(r)}_5$	$Z^{(9-r)}_5$	$Z^{(9-r)}_5$	N/A
$Z'^{(r)}_6$	$Z^{(9-r)}_6$	$Z^{(9-r)}_6$	N/A

Table 3.4: IDEA decryption subkeys $Z'^{(i)}_r$ derived from encryption subkeys $Z^{(i)}_r$. $-Z_i$ and Z_i^{-1} denote additive inverse modulo 2^{16} and multiplicative inverse $2^{16}+1$ of Z_i respectively.

3.3.1 Multiplication Modulo $2^n + 1$

Of the basic operations used in the IDEA algorithm, multiplication modulo $2^{16}+1$ is the most complicated and occupies most of the hardware. Curiger et. al. [CBK91] described and compared several VLSI architectures for multiplication modulo 2^n+1 and found that an architecture proposed by Meier and Zimmerman [MZ91], using modulo 2^n adders with bit-pair recoding offers the best performance.

The C code for the multiplication modulo $2^{16}+1$ operation by modulo 2^{16} adders using bit-pair recoding is as follows.

```
1 uint16 mulmod(uint16 x, uint16 y)
2 {
3     uint16 xd, yd, th, tl;
4     uint32 t;
5     xd = (x - 1) & 0xFFFF;
6     yd = (y - 1) & 0xFFFF;
7     t = (uint32) xd * yd + xd + yd + 1;
8     tl = t & 0xFFFF;
9     th = t >> 16;
```

```
10     return (tl - th) + (tl <= th);
11 }
```

This algorithm requires a total of six additions and subtractions, one 16-bit multiplication and one comparison. However, in IDEA one of the operands of a modular multiplication operation is always a subkey, so the second subtraction can be eliminated if the associated subkeys are pre-decremented.

3.3.2 Previous work on IDEA

The holder of the patent on the IDEA algorithm, Ascom implemented the IDEA cipher in software which achieves 0.37×10^6 encryption per seconds, or an equivalent encryption rate of 23.53 Mb/sec on an Intel Pentium II 450 MHz machine. Another software implementation is proposed by Helger [Lip98] involve Intel MMX multimedia instructions set. This implementation offer 0.51×10^6 encryption per seconds or an equivalent encryption rate 32.9 Mb/sec on an Intel Pentium 233 MHz machine. In 2000, Helger developed his software implementation in parallel architecture. His 4-way IDEA implementation achieves 440 Mb/sec on an Intel Pentium III 800 MHz machine. The term 4-way means that there are 4 independent IDEA encryptions or decryptions done in parallel. Our optimized software implementation running on a Sun Enterprise E4500 machine with twelve 400 MHz Ultra-IIi processor, performs 2.30×10^6 encryptions per second or a equivalent encryption rate of 147.13 Mb/sec.

Hardware implementations offer significant speed improvements over software implementations by exploiting parallelism among operators. In addition, they are likely to be cheaper, having lower power consumption and smaller footprint than a high speed software implementation. The first VLSI implementation of IDEA was developed and verified by Bonnenberg et. al. in 1992 using a 1.5 μm CMOS technology [BCF+91]. This implementation had an encryption rate of 44 Mb/sec. In 1994, VINCI, a 177 Mb/sec VLSI implementation of the IDEA algorithm in 1.2 μm CMOS technology, was reported by Curiger et. al. [CBZ+93, ZCB+94]. A

355 Mb/sec implementation in 0.8 μm technology of IDEA was reported in 1995 by Wolter et. al. [WMSL95], followed by a 424 Mb/sec single chip implementation of 0.7 μm technology by Salomao et. al. [SAF98] was reported. A paper design of an IDEA processor which achieves 528 Mb/sec on four XC4020XL devices was proposed by Mencer et. al. [MMF98]. In 2000, Leong et. al. proposed a 500 Mb/sec bit-serial implementation of IDEA on an Xilinx Virtex XCV300-6 FPGA which is scalable on larger devices [LCTL00]. Later, Goldstein et. al reported an implementation on the PipeRench FPGA which achieves 1013 Mb/sec [GSB+00]. A commercial implementation of IDEA called the IDEACrypt Kernel developed by Ascom achieves 720 Mb/sec [Asc99b] in 0.25 μm technology. Another implementation derived from the IDEACrypt Kernel, called the IDEACrypt Coprocessor, has a throughput of 300 Mb/sec [Asc99a].

Year	Implementation	Throughput (Mb/sec)	Reference
1998	software	23.53	[Lip98]
2000	software	440	[Hel]
1992	ASIC 1.5 μm CMOS	44	[BCF+91]
1994	ASIC 1.2 μm CMOS	177	[CBZ+93, ZCB+94]
1995	ASIC 0.8 μm CMOS	355	[WMSL95]
1998	ASIC 0.7 μm CMOS	424	[SAF98]
1998	4 $\times$ XC4020XL	528	[MMF98]
1999	ASIC 0.25 μm CMOS	720	[Asc99b]
2000	Xilinx Virtex XCV300-6	424	[LCTL00]
2000	ASIC 0.25 μm CMOS	1013	[GSB+00]

Table 3.5: Comparison of IDEA implementations

Most of the previous hardware are known to use a precomputed key schedule or only the encryption process was implemented, due to the fact that if is more difficult to implement decryption for IDEA. For encryption, the whole key schedule ($52\times$ 16 bits) can be derived from the first six subkeys (16 bits each) by shifting, while the decryption key schedule is derived from the whole encryption key schedule. Also the decryption subkey K_i^{-1} is the multiplicative inverse mod $2^{16}+1$) of K_i, where

K_i is the corresponding encryption subkey. It can be simplified to be $K_i^{-1} \oplus K_i = 1$. The conversion of decryption subkey require an iterative process based on the encryption subkey and is more difficult to implement in hardware and hardware implementations can get around this difficulty by computing all subkeys in software and making them an input to the hardware [WMSL95].

3.4 Block Cipher Modes of operation

A block cipher is a mathematical function which maps n-bit plaintext blocks to n-bit ciphertext blocks, where n is the blocklength. In order to perform decryption uniquely, the encryption function is one-to-one mapping which is invertible. The inverse mapping is defined as the decryption function. In most block ciphers, the encryption and decryption process is similar so that the same hardware can be used.

A block cipher encrypts plainext in fixed-size n-bit blocks. However, if messages exceeds n-bit, there are different modes of operation can be used, namely:

There are four common modes :

1. Electronic Code Book (ECB) mode
2. Cipher-block Chaining (CBC) mode
3. Cipher feedback (CFB) mode
4. Output feedback (OFB) mode

These modes will be explained in the following sections.

3.4.1 Electronic Code Book (ECB) mode

Electronic Code Book mode (Figure 3.4) is the simplest approach for employing block ciphers. For messages exceeding n-bits, the message is partitioned into n-bit blocks and each of these blocks are encrypted independently. In ECB mode,

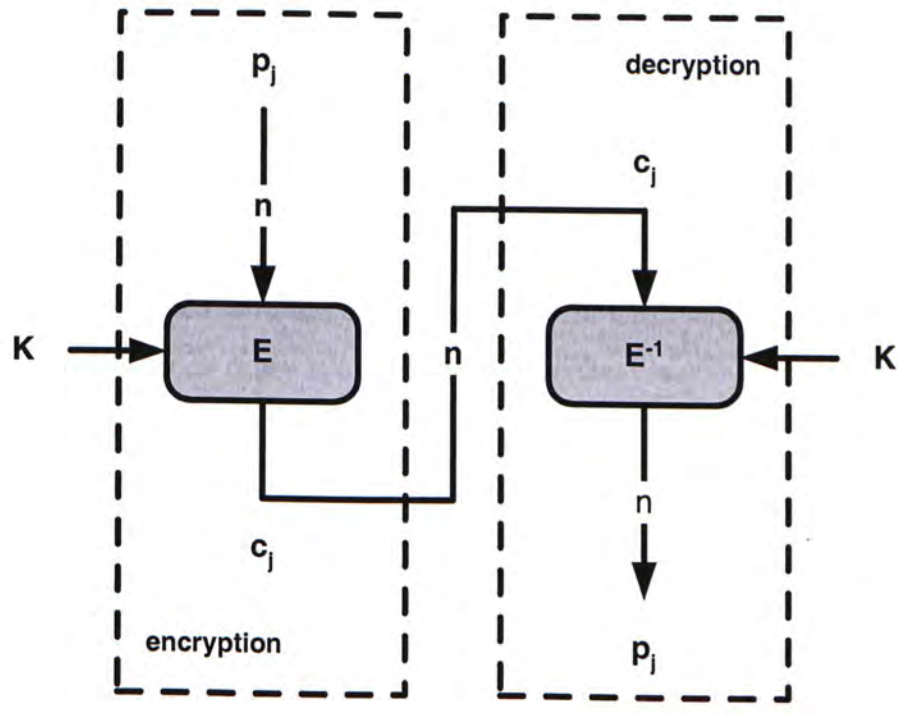

Figure 3.4: Electronic Codebook mode

encryption is independent of the sequence of blocks, e.g. the last block can be encrypted first and followed by first block and this will not affect the ciphertext. Also, identical plaintext blocks with the same key always result in identical ciphertext.

The ECB mode of operation is rather weak in security. For 8-bit blocks, once we know that 'e' is encrypted to 'z', we know whenever the ciphertext is 'z' if the plaintext was 'e'. Thus ECB mode allows simple frequency analysis to be applied.

The algorithm of the ECB mode of operation can be described as:

$$Encrpytion: \quad for\ 1 \leq j \leq t, \quad c_j \leftarrow E_K(p_j)$$
$$Decrpytion: \quad for\ 1 \leq j \leq t, \quad p_j \leftarrow E_K^{-1}(c_j)$$

where K is a k-bit Key, $p_1, \ldots, p_t$ are the plaintext blocks and $c_1, \ldots, c_t$ are ciphertext blocks. E_k and E_k^{-1} denotes encryption and decryption process with key K respectively.

3.4.2 Cipher-block Chaining (CBC) mode

In the CBC mode of operation (Figure 3.5), every plaintext block is exclusive-ored with the previous ciphertext block before being encrypted. For example, the first plaintext block p_1 is enciphered to produce c_1 before it is encrypted to produce c_2. The next plaintext block p_2 is exclusive-ored with ciphertext block c_1. The procedure is repeated until the end of message. In CBC mode, It is obvious that every ciphertext block depends on previous ciphertext blocks.

For the first encryption, there is no previous ciphertext. An initialization vector (IV) is introduced for initialization of the feedback value. The IV need not be secret.

For CBC mode, identical ciphertext blocks are obtained if the same plaintext is enciphered using the same key and IV. If either IV, key, or first plaintext block is changed, a different ciphertext is obtained. Since ciphertext c_j is depends on p_j and all preceding plaintext blocks, the decryption order of ciphertext blocks needs to be maintained. Correct decryption requires all preceding ciphertext blocks to be correctly decrypted.

Since there are data dependencies of all plaintext and ciphertext blocks, if a plaintext block p_j is modified during encryption, it affects all following ciphertext blocks. Thus it is not possible to encrypt multiple blocks in parallel (like in ECB mode).

The algorithm of the CBC mode of operation is described as:

$$Encrpytion : c_0 \leftarrow IV. \quad for\ 1 \leq j \leq t, \quad c_j \leftarrow E_K(p_j \oplus c_{j-1})$$

$$Decrpytion : c_0 \leftarrow IV. \quad for\ 1 \leq j \leq t, \quad p_j \leftarrow c_{j-1} \oplus E_K^{-1}(c_j)$$

k-bit Key K, n-bit IV, n-bit plaintext blocks $p_1, \ldots, p_t$ are inputs and $c_1, \ldots, c_t$ is n-bit ciphertext blocks as output. E_k and E_k^{-1} denotes encryption and decryption process with key K respectively.

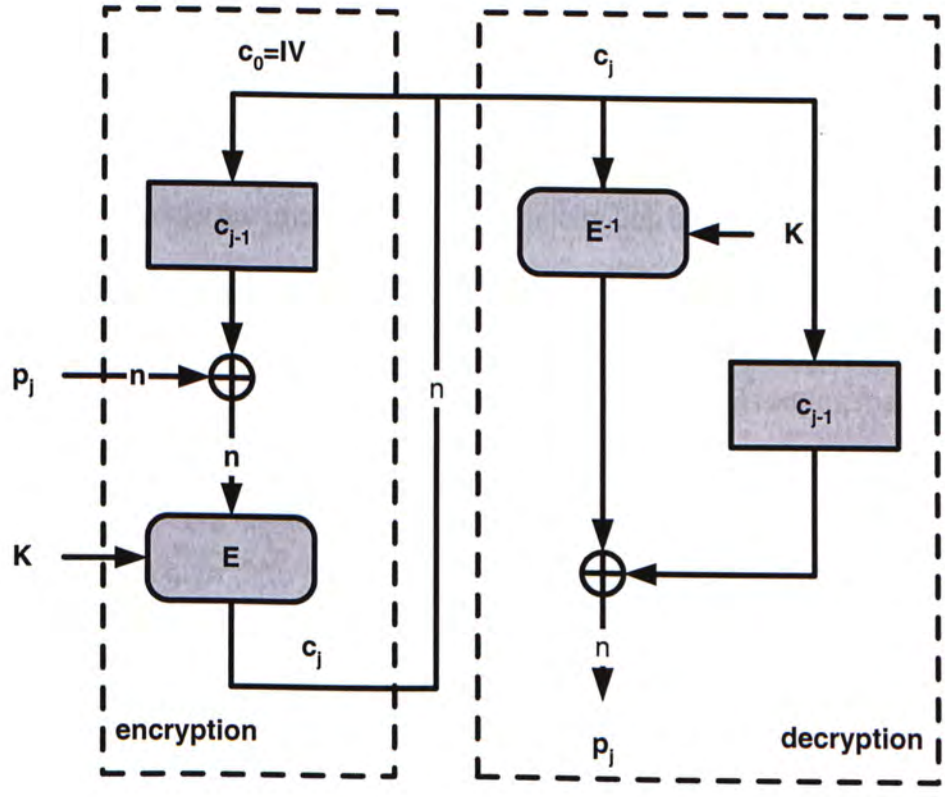

Figure 3.5: Cipher Block Chaining mode

3.5 Field-Programmable Gate Arrays

FPGAs are hardware devices that can change their functionality by programming the chips after fabrication. The programming process of FPGAs usually is less than a minute and can be done in the field, therefore they are very suitable for use as a reconfigurable platform. FPGAs consists of an array of configurable logic blocks (CLBs) surrounded by input/output blocks (IOBs) which provide an interlace between configurable logic block and package pins, and a network of routing resources called the general routing matrix (GRM) which interconnect the configurable logic blocks.

In most commercial devices, such as the Xilinx Virtex and Virtex-E family, configurable logic blocks are implemented as 4-input lookup tables together with an optional output register or latch. The array of CLBs in FPGA are arranged in columns and rows. Between CLBs, there are routing channels aligned horizontally and vertically. CLBs are interconnected by routing channels and general routing matrix. The GRM consists of an array of routing switches located at the conjunction of horizontal and vertical routing channels. Since every CLB has a Lookup table and registers , they maintain a high ratio of storage elements to computational elements. Also, since these computational and storage elements are coupled together in CLBs, these architectures are very suitable for the implementation of deeply pipelined designs.

3.5.1 Xilinx Virtex-E^{TM} FPGA

The Triple-DES accelerator are built on the Xilinx Virtex-E^{TM} FPGA. It is manufactured in a 6-layer-metal 0.18 μm CMOS process. The maximum synchronous clock rates for Virtex-E^{TM} FPGA is 240MHz.

The Logic cell (LC) is the basic building block for Virtex-E CLB. A logic cell consists of a 4-input function generator, carry logic, and a storage element. In fact, every Virtex-E CLB consists of four logic cells, which are organized in two similar slices as shown in Figure 3.7. Function generators in Virtex-E are implemented as

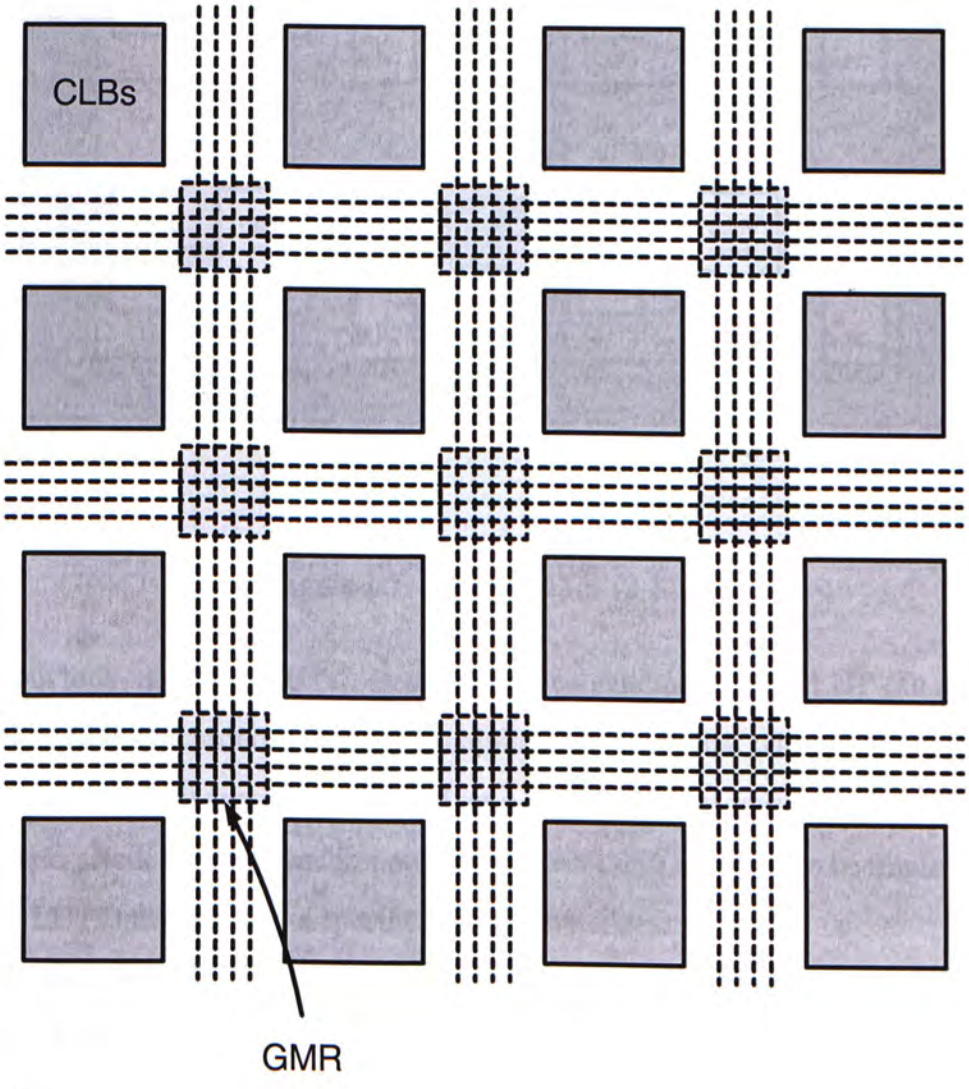

Figure 3.6: Architecture of FPGAs

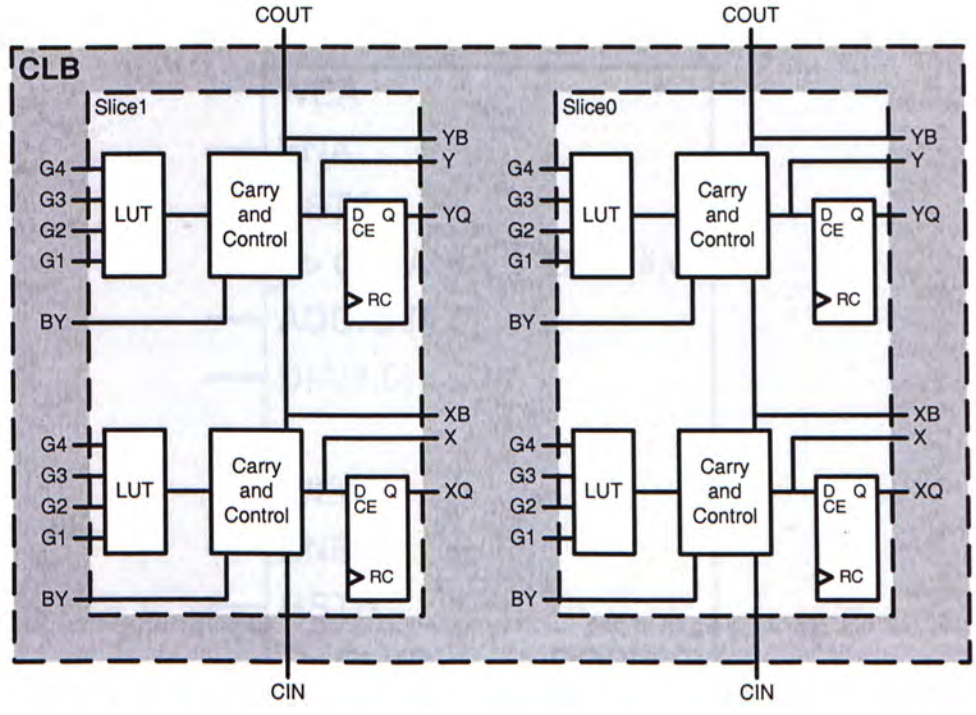

Figure 3.7: Virtex-E CLB (2-Slice)

4-input look-up tables (LUTs). Besides function generation, each LUT can serve as a 16 × 1-bit synchronous RAM. Combining two LUTs that within a slice, a 16 × 2-bit or 32 × 1-bit synchronous RAM, or a 16 × 1-bit dual-port synchronous RAM can be created. Besides synchronous RAM, 16-bit shift register can be implemented from LUTs but limited to a specific LC in every slices.

Also, Virtex-E FPGAs provide large block Select RAM memories. Each Block Select RAM component is a synchronous dual-ported 4096-bit RAM with independent control signals for each port as shown in Figure 3.8. Block SelectRAM is fully customized and data-widths of the two ports can be configured independently. The selection of data-width of a Block Select RAM range from 1 to 16 bits with depth 4096 to 256 correspondingly. These RAM provide a relatively large buffer for storing plaintext and ciphertext in Triple-DES and IDEA cipher. Also its fully customized feature enable the I/O interface operate in different clock rate as the

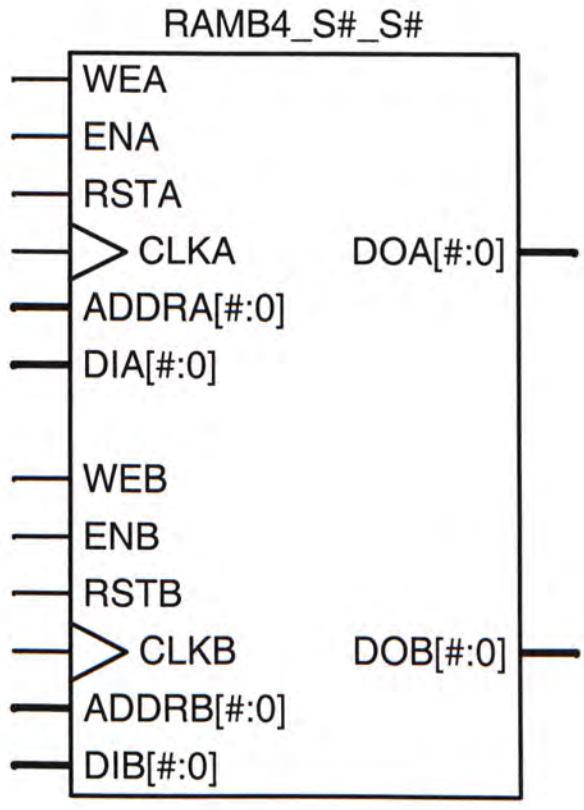

Figure 3.8: Dual-Port Block SelectRAM

clock rate for cipher.

3.6 Pilchard

Pilchard is a reconfigurable computing development environment, which employs a field programmable gate array. It was designed mainly to reduce the bottleneck of bus interface transfers between FPGA and personal computer by using the memory interface instead of the PCI bus interface. Nowadays FPGA systems can operate at clock frequencies over 100MHz with microprocessors operate at 1GHz. The speed of coprocessor systems are often limited by the bus interface, for example, the author's 496 Mb/sec implementation of IDEA cipher achieves only 39.2 Mb/sec

through a cardbus interface [LLC+01].

The Pilchard board was designed to be compatible with the 168 pin 3.3 Volt, 133 MHz, 72-bit, registered synchronous DRAM in-line memory modules (SDRAM DIMMs) PC133 standard. As SDRAM DIMMs PC66/100 standard shared the same pinouts as SDRAM DIMMs PC133 standard, therefore, Pilchard board can be operated under PC66/100 standard as well. The Pilchard board can populated with any Virtex or Virtex-E device in a PQ240 or HQ240 package.

The system interface of Pilchard board is developed using Linux. In order to access registers of Pilchard board, UNIX mmap() system call is used to map virtual addresses in user space to physical address of the Pilchard board. Data transfer between PC and Pilchard use the 64-bit MMX instruction "movq" embedded in inline assembly.

In benchmark of Pilchard board, write operation was reported to be 1063.04 Mb/sec using uncacheable mode [LLC+01]. Uncacheable mode guarantees that all reads and writes appear on the system bus as the same order in program. For read operations, Pilchard board achieves 422.40 Mb/sec. In a read/write benchmark, the transfer rate is 595.92 Mb/sec. Compared to the measured transfer rate of the PCI interface, which is 96.08 Mb/sec, Pilchard was 4 times faster.

3.6.1 Memory Cache Control Mode

Central processing unit (CPU) caching of reads and writes to Pilchard registers could lead to incorrect results. The Intel Pentium Pro, Pentium II and Pentium III has a Memory Type Range Register (MTRR), accessible from Linux, which allows different memory regions to be of different types [pen00].

The "Uncacheable" memory type guarantees that all reads and writes will appear on the system bus in the same order as the program. Furthermore, no speculative memory accesses, page-table talks or prefetches of speculated branch targets

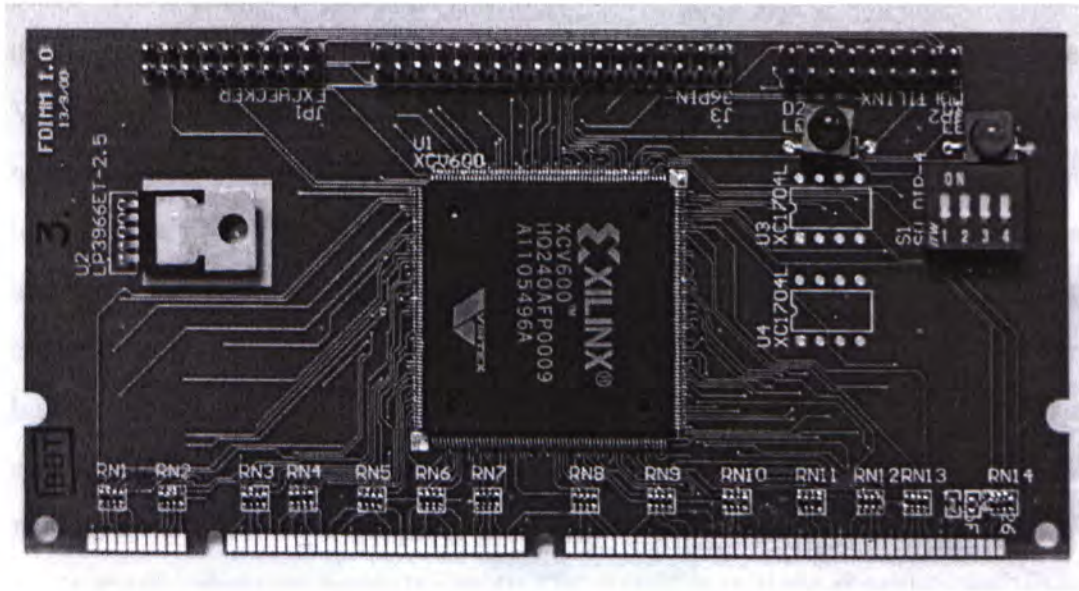

Figure 3.9: Picture of Pilchard

will occur [pen00]. Although the most conservative, it also leads to the lowest performance.

3.7 Electronic Design Automation Tools

The VPN accelerator was developed using the Very High Speed Integrated Circuit Hardware Description Language (VHDL). Hardware architectures can be described using VHDL. Besides as a descriptive language for hardware, it represent a design methodology.

The target hardware platform were Field Programmable Gate Arrays (FPGAs). FPGAs can be reconfigurable by downloading a software bitstream. The software bitstream is created from the VHDL description using the synthesis and implementation tools. The synthesis and implementation tools used was Synopsys FPGA Express™ 3.4 and Xilinx Xilinx Foundation™ 3.2i respectively.

Using FPGAs and VHDL offers a short turnaround time as shown in Figure 3.10.

Since VHDL designs offer a higher level of abstraction over say schematic capture, the design and debug time associated with this methodology is reduced. Once the VHDL description has been verified via simulation, a synthesis tool is used to generate a netlist. The implementation tool takes the netlist and maps the component to the target FPGA device. Then the place and route tools place the components and route the interconnections in the FPGA. In the implementation stage, timing constraints can be given and the tools will try to meet the constraint. Finally, the bitstream file of the circuit design is generated by the implementation tools for a specific FPGA. Downloading the specific bitstream to corresponding FPGA will make it work as per the description in VHDL.

In the development cycle, the target FPGA device is given after the simulation stage. Therefore, the same VHDL description can be used to synthesise netlists for different devices. This features offer portability in that different FPGA vendors, families and application specific integrated circuits (ASICs) can be targetted from the same VHDL description.

3.8 Summary

In this chapter, background on cryptography and some of cryptographic algorithm were given. Also FPGAs and the Pilchard environment were presented.

Most of the previous implementation of DES, Triple-DES and IDEA achieves high performance by using pipelining in ECB mode. However, most of applications recently suggest CBC mode which is more secure.

The decryption process in IDEA involve iterative calculation of subkeys, therefore, most IDEA hardware [WMSL95] uses precomputed keys or required the whole key schedule (832 bits) to be an input. This creates a large overhead on data transfer of key material in IDEA cipher.

FPGAs are hardware devices very suitable for rapid prototyping as its nature to change its functionality after fabrication. Pilchard is a reconfigurable computing

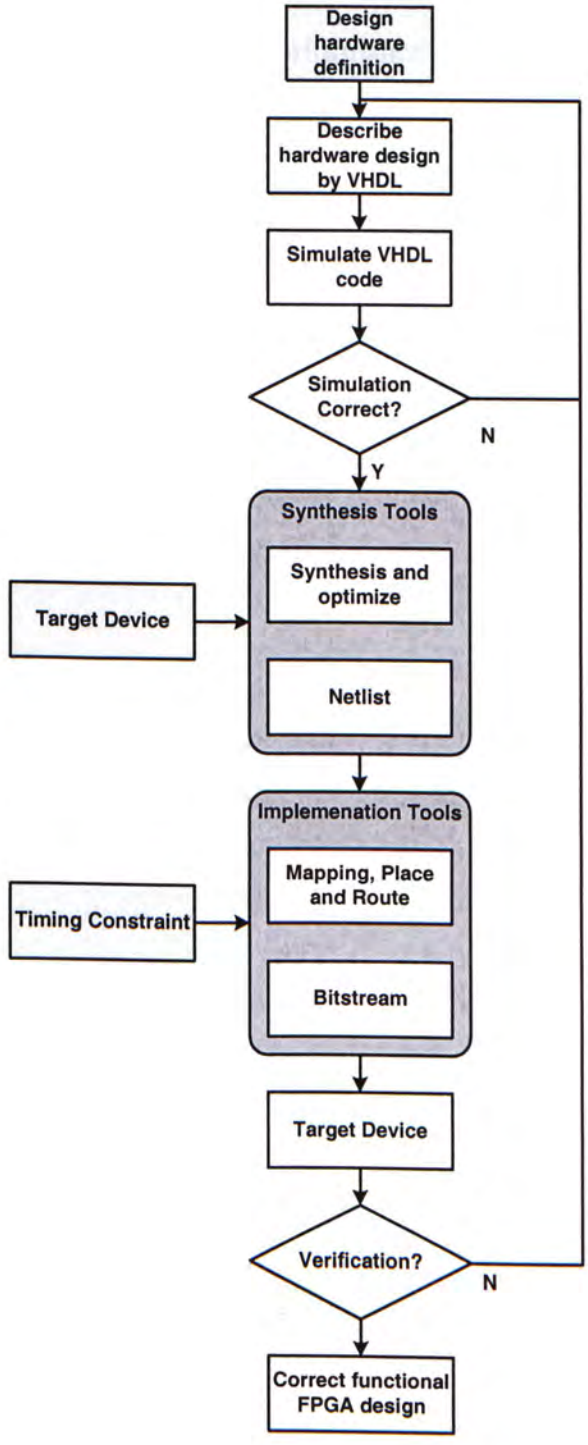

Figure 3.10: Development cycles for FPGA design using VHDL

development environment that employ FPGAs with a DIMM slot interface in order to improve transfer rate between PC and FPGA systems. This environment is used in this project as it offer a reasonable performance and easy to use interface.

Chapter 4

Implementation

4.1 Introduction

In this chapter, the implementation details for the VPN accelerator are presented. Firstly, the hardware platform used for the implementation is introduced, followed by the implementation details of DES, Triple-DES and IDEA cipher in different operation modes. Then the encapsulation of VPN accelerator in LibDES and FreeS/WAN is discussed.

4.1.1 Hardware Platform

4.1.2 Reconfigurable Hardware Computing Environment

A reconfigurable hardware computing environment, called Pilchard, was used to implement the Triple-DES accelerator. Pilchard is a DIMM RAM based FPGA system. The Pilchard board used for Triple-DES and IDEA accelerator was populated with a Virtex-E XCV1000E device in a HQ240 package with speed grade 6. The XCV1000E device contains 12288 slices aligned in a 64 $\times$ 96 CLB array, which equivalent to 1.5 million system gates. In addition, there are 96 $\times$ 4096-bit Block Select RAMs for storing data and eight delay-locked loops for clock multiplication and division.

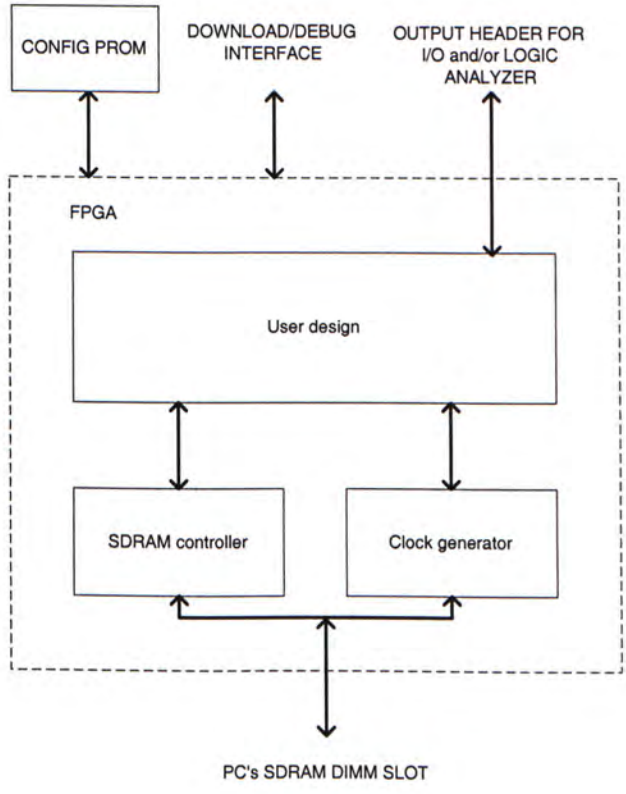

Figure 4.1: Block diagram of the Pilchard board

4.1.3 Pilchard Software

There are two major components in the Pilchard software namely the device driver and application programming interface (API). The device driver enables Pilchard to be accessible via a DIMM slot and specifies the physical memory range for Pilchard. The API perform memory mapping between virtual and physical addresses for Pilchard upon initialization. A series of read and write function calls in the API enables data transfer between the PC and Pilchard as shown below.

- write32(d, a) A 32-bit word referenced by pointer d is written to physical memory address a. This function call fails if a is not in the memory range specified in Pilchard driver.

- read32(d, a) A 32-bit word is read from physical memory address a and transferred to memory referenced by pointer d. This function call fails if a is not in the memory range specified in Pilchard driver.

The original Pilchard software developed by M.P. Leong [LLC+01] already has a fully functional device driver and 32-bit read and write function calls, namely write32(d,a) and read32(d,a). The original version of Pilchard VPN accelerator had limitations which its application to ciphers such as DES, Triple-DES and IDEA employ a block length of 64-bit. Therefore, two write32() / read32() function calls are needed in order to transfer every plaintext or ciphertext between the PC and Pilchard.

As a result, two new API function, write64(d,a) and read64(d,a), were developed based on the original read32 and write32 function calls. In order to perform 64-bit data transfer, "movq" [mmx] (move quad word) assembly MMX instruction was used. Both 64-bit API function calls are written in inline assembly which can embedded into any C code.

- write64(d, a) A 64-bit double word referenced by pointer d is written to physical memory address a. This function call fails if a is not in the memory range

specified in Pilchard driver.

- read64(d, a) A 64-bit double word is read from physical memory address a and transferred to memory referenced by pointer d. This function call fails if a is not in the memory range specified in Pilchard driver.

For 64-bit write function calls, a "movl" (move long word) instruction is issued to copy the 32-bit destination address to a 32-bit general purpose register. Then a "movq" (move quad word) instruction is issued to transfer the 64-bit data to MMX register. Finally, another movq instruction copied the 64-bit data from MMX register to the memory address that referenced by 32-bit general purpose register. Also an emms instruction was issued to clear the MMX register, otherwise, floating point calculations will result in Not A Number (NAN) in some cases. For 64-bit read function calls, a "movl" instruction is first issued to copy the source address. Then "movq" instructions are used to copy data from Pilchard to an MMX register and finally to the user program memory.

4.2 DES in ECB mode

4.2.1 Hardware

The DES algorithm is a cascade of sixteen identical rounds of operations in between an initial permutation and final permutation. A module for one round of computation is formed using ROM32 [Xil00a] and XOR primitives.

The DES core used in here is derived from the VHDL code written by Chris Eilbeck. Since there are no data dependencies in ECB mode. The DES core is pipelined into 16 stages corresponding to 16 rounds in DES shown in Figure 4.2.

The maximum clock rate achieve by the proposed DES core was 60.7 MHz. Since the clock rate of interface is limited to 100 MHz, a clock divider was used to divide the system clock by 2. Therefore, the DES core works at 50 MHz and

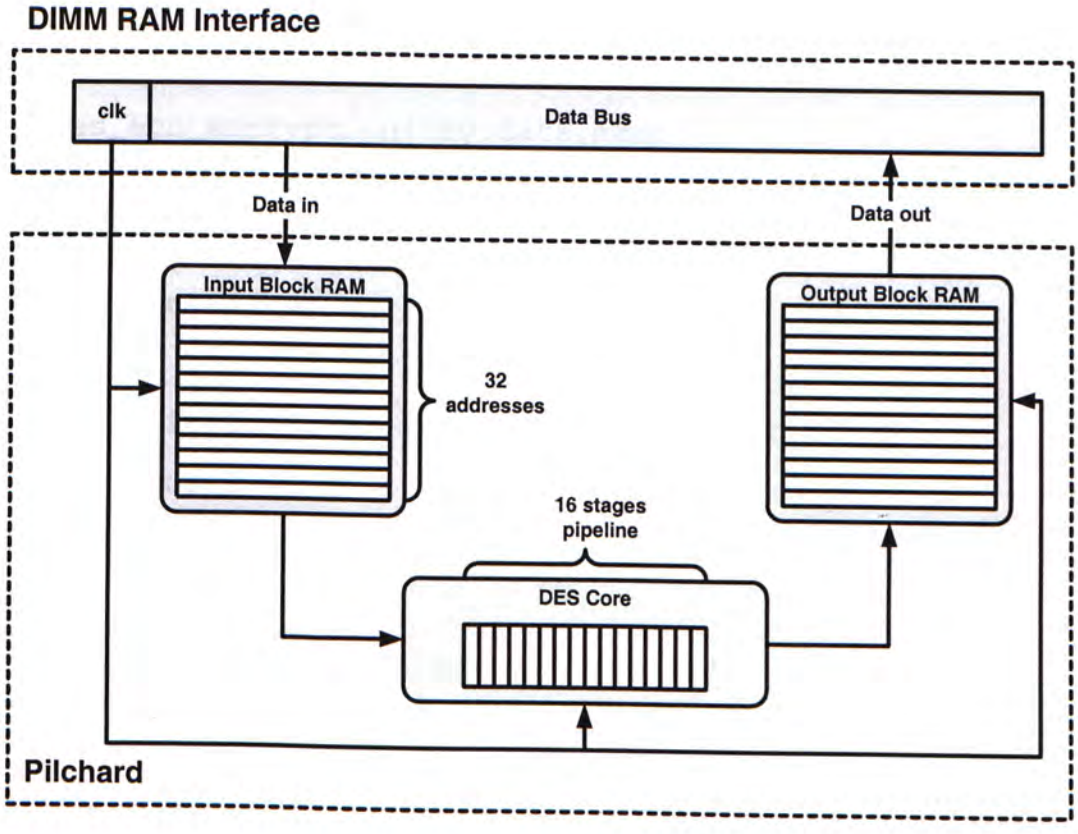

Figure 4.2: System architecture of DES accelerator in ECB mode

the maximum encryption rate of the DES implementation on Pilchard is around 248 Mb/sec.

4.2.2 Software Interface

memp is the base address pointer to Pilchard mapped memory range. The key material is transferred to Pilchard at the beginning and then the control register is reset to zero in order to reset the DES core. Then 32 plaintext blocks are transferred to DES core. When the first plaintext block is transferred to Pilchard, the control register is set to 1 in order to trigger the start of DES core. Finally, 32 ciphertext blocks are received from Pilchard.

The following is the pseudocode for a block for 32 DES encryptions in ECB

mode:

```
1 des_ecb_encryption(key,data,memp)
2 {
3     /* copy key into key register */
4     write64(key,key_reg);
5
6     /* initialize control register to reset */
7     control=0;
8     write64(control,control_reg);
9
10    /* write 32 * 64 bit plaintext */
11     for(i=0;i<32;i++){
12       write64(data[i],memp+i*8);
13       /* early trigger DES core */
14       if(i==0){
15          /* set control register to 1
16          to trigger starting of DES encryption */
17          control=1;
18          write64(control,control_reg);
19          }
20    }
21
22    /* read 32 * 64 bit ciphertext */
23    for(i=0;i<32;i++)
24       read64(&data[i],memp+i*8);
25 }
```

Since the Pilchard interface does not offer interrupts, for the software (PC) to know about the status in hardware (Pilchard), polling is the only option. In order to achieve highest performance in DES ECB mode, the assumption that DES core can output the first ciphertext in the period that the 32 blocks of plaintext are written to Pilchard was made. Correctness of operation was extensively tested by comparison with software.

4.3 DES in CBC mode

4.3.1 Hardware

The DES core was modified to be totally combinational. Since in CBC mode, the current encryption depends on previous ciphertext, only one encryption is processed at a time so pipelining is not effective.

4.3.2 Software Interface

The following is the pseudocode for a block for 32 DES encryptions in CBC mode:

```
1 des_cbc_encryption(key,data,memp)
2 {
3     /* copy key into key register */
4     write64(key,key_reg);
5
6     /* initialize control register to reset */
7     control=0;
8     write64(control,control_reg);
9
10    /* write 32 * 64 bit plaintext */
```

```
11    for(i=0;i<32;i++){
12       write64(data[i],memp+i*8);
13
14    /* set control register to 1
15    to trigger starting of DES encryption */
16    control=1;
17    write64(control,control_reg);
18
19    /* poll control register for
20    the end of encryption */
21    do{
22       read64(&control,control_reg);
23    }while(control==0);
24
25    /* read 32 * 64 bit ciphertext */
26    for(i=0;i<32;i++)
27       read64(&data[i],memp+i*8);
28 }
```

The interface for DES CBC mode (Figure 4.3) first requires reset of the finite state machine. Then 248 different 64-bit plaintext are transferred to Pilchard from PC via DIMM RAM data bus. This is followed by setting the control register to 1 in order to trigger the finite state machine and DES core. No interrupt routine is provided in DIMM RAM interface, therefore, polling has to be done on control register to detect completion of all encryptions. Finally, all ciphertext are read back from Pilchard. The buffer size was chosen to be 248 because there is an 8-bit effective address space for Pilchard, but 8 address were reserved for control and key registers.

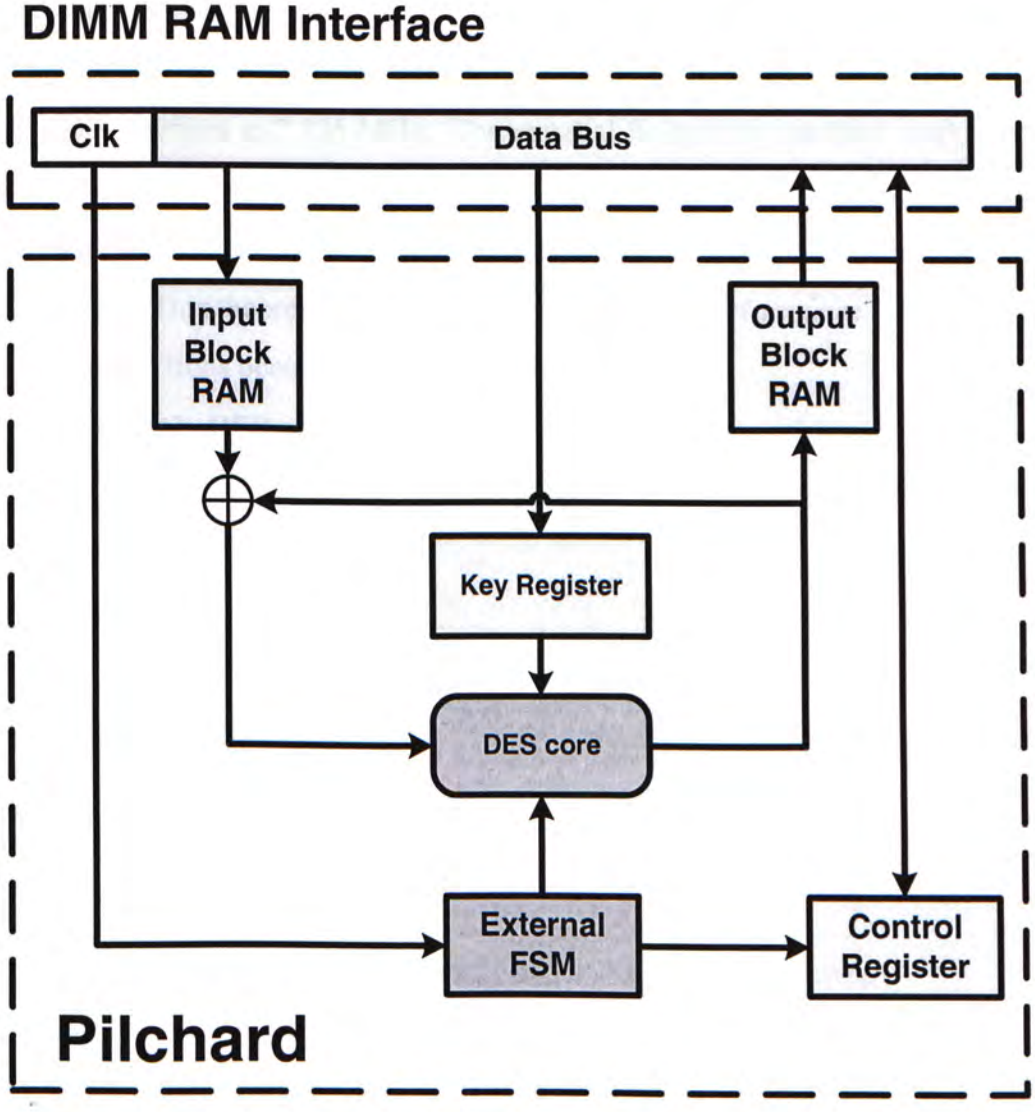

Figure 4.3: System architecture of DES accelerator in CBC mode

4.4 Triple-DES in CBC mode

4.4.1 Hardware

The Triple-DES core (Figure 4.4) is formed by cascading three combinational DES CBC cores. The Triple-DES core is combinational, but an external finite state machine was used to determine the readiness of input and output. The proposed Triple-DES core operates at 2.135 MHz. The external finite state machine works at 50 MHz which is the system clock (100 MHz) divided by two. A 64-bit ciphertext is obtained every 32 cycles. Therefore the performance of core is 50 $\times$ 64-bit / 32 = 100 Mb/sec. This theoretical result agrees with the real performance of Triple-DES hardware functions benchmark embedded in LibDES.

In our Triple-DES core, a throughput of 96 Mb/sec was achieved. Triple-DES consists of three cascaded DES core and thus, it require more processing time than DES.

4.4.2 Software Interface

The following is the pseudocode for a block of 248 Triple-DES encryptions in CBC mode:

```
1 3des_cbc_encryption(key1,key2,key3,data,memp)
2 {
3     /* copy key into key register */
4     write64(key1,key_reg1);
5     write64(key2,key_reg2);
6     write64(key3,key_reg3);
7
8     /* initialize control register to reset */
9     control=0;
```

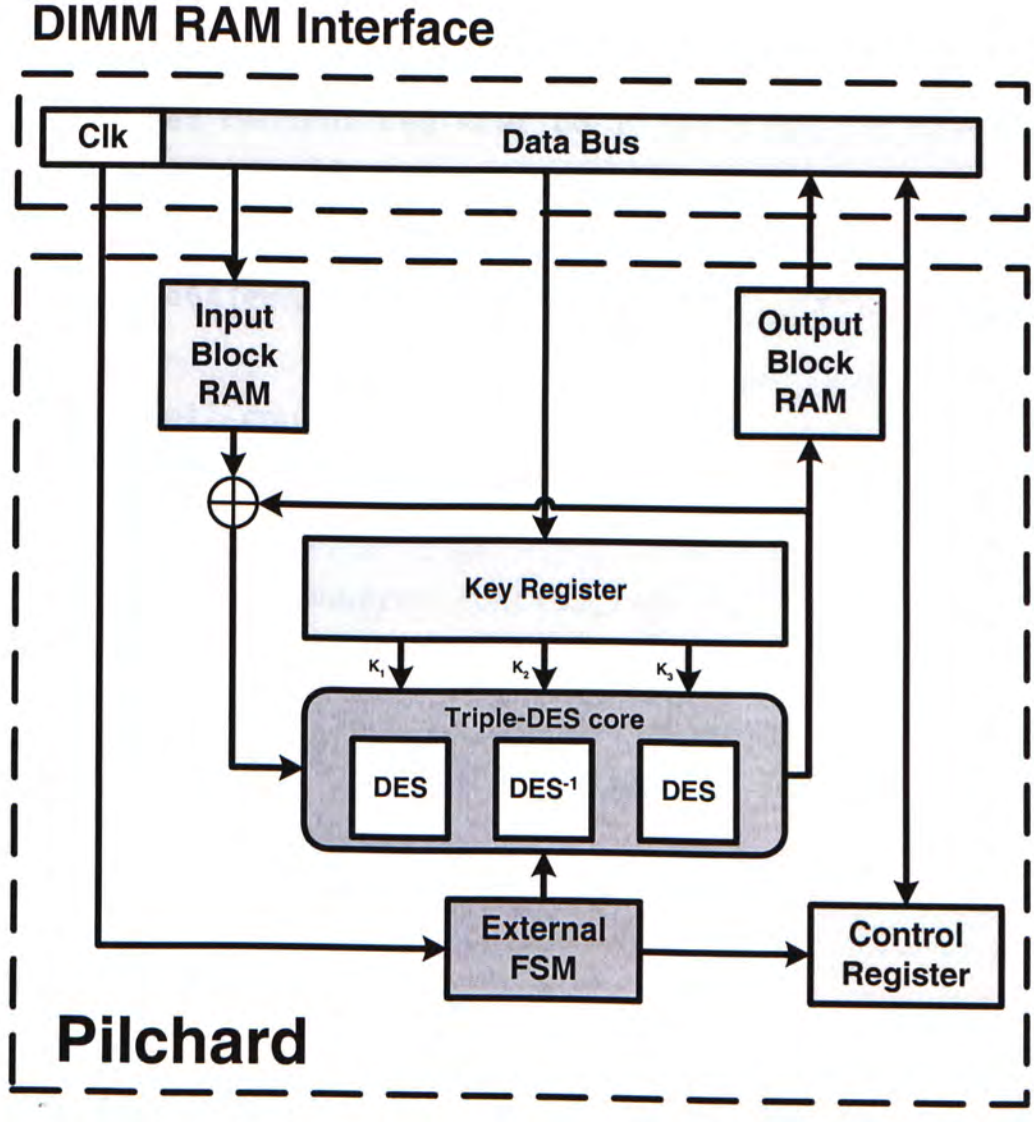

Figure 4.4: System architecture of Triple-DES accelerator

```
10    write64(control,control_reg);
11
12    /* write 248 * 64 bit plaintext */
13    for(i=0;i<248;i++){
14       write64(data[i],memp+i*8);
15
16    /* set control register to 1
17    to trigger starting of DES encryption */
18    control=1;
19    write64(control,control_reg);
20
21    /* poll control register for
22    the end of encryption */
23    do{
24       read64(&control,control_reg);
25    }while(control==0);
26
27    /* read 248 * 64 bit ciphertext */
28    for(i=0;i<248;i++)
29       read64(&data[i],memp+i*8);
30 }
```

The Triple-DES CBC mode interface is similar to DES CBC mode, the only difference being the key size. Also the throughput of Triple-DES CBC mode is lower than DES CBC mode since the datapath for Triple-DES is three times longer.

4.5 IDEA in ECB mode

4.5.1 Multiplication Modulo $2^{16} + 1$

Modulo multiplication is the bottleneck in the IDEA algorithm. In a single round of the algorithm there are four modular multiplications so a well-designed multiplication modulo $2^{16} + 1$ operator is crucial since it directly affects the system performance both in terms of area and throughput.

The modular multiplication algorithm described in Section 3.3.1 was used in our design, but instead of taking x and y as inputs, the operator takes x and y_d as inputs. As one of the operands is a subkey which is regarded as a constant, the modification eliminates one subtraction operator by taking the advantage of pre-decremented subkeys (Section 3.3.1, pseudocode line 6).

In order to implement a well-designed multiplication modulo $2^{16} + 1$ operator, the throughput of the operator is maximized by introducing more pipeline stages. In our design, 16-bit multiplier used in Section 3.3.1 (pseudocode line 7) is constructed by Xilinx CORE Generator [Xil00b] which has a latency of 4 cycles. And the multiplication modulo $2^{16} + 1$ operator pipeline has a latency of 7 cycles.

4.5.2 Hardware

The IDEA algorithm is a cascade of eight identical rounds of operations, followed by a output transformation. By instantiating building blocks, that is, additions, XORs and modular multiplications, and inserting appropriate stage latches for time-alignment, a module for one round of computation is formed. For the best area-efficiency, stage latches are constructed by Virtex SRL16E primitives [Xil99, GA99].

Due to limited hardware resources, each round of the algorithm shares the same physical resource, but with different key-schedules. The output transformation also reuses the resources. In our implementation the key-schedules are stored inside

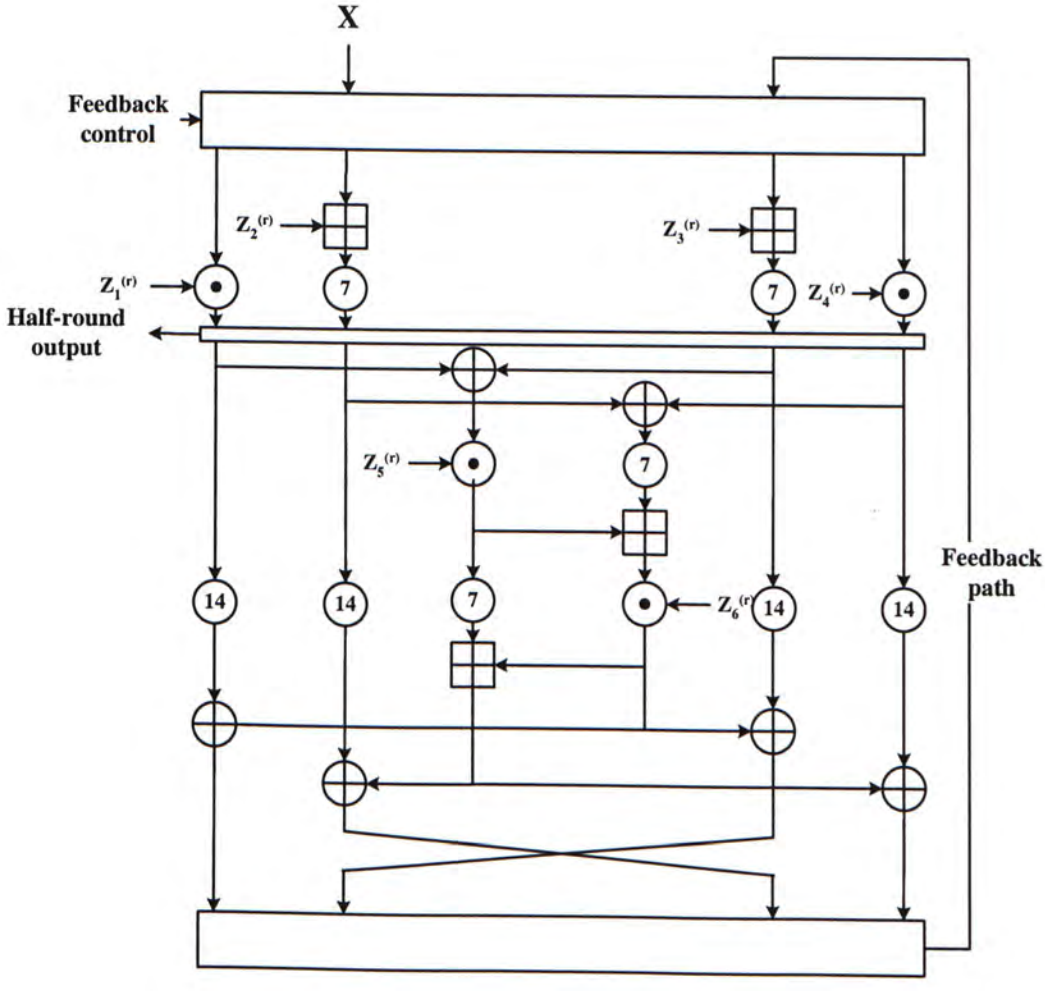

Figure 4.5: Architecture of the IDEA core.

ROM primitives. The architecture of the bit-parallel IDEA core is shown in Figure 4.5.

As mentioned earlier, for ECB mode operations, data dependencies of the IDEA algorithm have no feedback paths. This property enabled the round architecture to take input values until the pipelined is filled, and output values are redirected to the input of the pipeline subsequently. In an IDEA round, the data passes through three multiplication modulo $2^{16} + 1$ operators, each of which has a latency of 7 cycles. Thus the full round pipeline has a latency of 21 cycles For an output transformation,

the data must pass through a single multiplication modulo $2^{16} + 1$ operator with pipeline latency of 7 cycles. Therefore the core has a total latency of $21 \times 8 + 7 = 175$ cycles. The core takes 21 64-bit plaintexts per $21 \times 9 = 189$ cycles, equivalently performing encryption at $(21 \div 189) \times 64 \times f$ Mb/sec with a system clock rate of f MHz. For instance, at a 82 MHz clock rate, the core delivers an encryption rate of 583 Mb/sec with a latency of 2.134 μs.

4.5.3 Software Interface

The following is the pseudocode for a block of 175 IDEA encryptions in ECB mode:

```
1 idea_ecb_encryption(data,memp)
2 {
3
4     /* initialize control register to reset */
5     control=0;
6     write64(control,control_reg);
7
8     /* write 248 * 64 bit plaintext */
9     for(i=0;i<175;i++){
10       write64(data[i],memp+i*8);
11
12    /* set control register to 1
13    to trigger starting of DES encryption */
14    control=1;
15    write64(control,control_reg);
16
17    /* poll control register for
18    the end of encryption */
19    do{
```

```
20      read64(&control,control_reg);
21   }while(control==0);
22
23   /* read 175 * 64 bit ciphertext */
24   for(i=0;i<175;i++)
25      read64(&data[i],memp+i*8);
26 }
```

The IDEA ECB mode interface is similar to DES CBC mode and Triple-DES CBC mode interfaces. However, in the IDEA ECB mode, the key-schedule is pre-computed and stored in FPGA. The data transfer of key-schedule is elimated.

4.6 Triple-DES accelerator in LibDES

In the above sections, the Triple-DES cipher interface for Pilchard was introduced. Another Triple-DES functions was implemented in LibDES which uses Pilchard as hardware accelerator. Both kernel mode version and user mode version has been developed. Most of the cryptographic softwares operates in user mode, however, because FreeS/WAN manipulates IPSec packet in kernel mode, a kernel version was also required. The user mode and kernel mode functions differ from the representation of memory address mapping for Pilchard. In user mode, a virtual address is used in interface for Pilchard, however, a direct access to a physical address is used in kernel mode.

As LibDES is a widely used encryption library for openSSL and other applications, the encapsulation of the Pilchard based accelerator interface in LibDES enables other application to easily utilize the Pilchard board accelerator.

4.7 Triple-DES accelerator in FreeS/WAN

In our implementation of a Triple-DES accelerator on Pilchard, a simple software interface was built in order to perform verification benchmark and performance. Modifications were needed to integrate the Triple-DES hardware accelerator and FreeS/WAN.

The architecture of VPN and IPSEC protocols was unchanged, therefore, no major modification on FreeS/WAN was required. Modifications were made on LibDES, and the software based DES functions were replaced by calls to the hardware accelerator on Pilchard.

In FreeS/WAN, there is a data structure that stores the encryption key in the form of a key-schedule. In Triple-DES accelerator on Pilchard, the key provided should be a raw-key. There are two solutions to this problem.

The interface of Triple-DES accelerator on Pilchard could be modified to accept a key schedule as input. However, this modification will have great impact on the performance of the Triple-DES accelerator. For Triple-DES encryption, a raw-key of 3×64-bit = 192-bit is needed, but if key schedule were used, $3 \times 16 \times 48$-bit = 2304-bit are needed to store the key schedule. This method requires 12 times more storage and transfer, therefore, this method was not used.

On the other hand, LibDES can be modified such that it can accept a raw key. However, problems with FreeS/WAN compatibility are encountered. In FreeS/WAN, there is a data-structure tdb (tunnel descriptor block) storing information about VPN connections. This includes the session key for the connection. Unfortunately, tdb only has entry for the key schedule. To minimize modifications on FreeS/WAN codes, a series of functions were rewritten so that tdb data structure does not need to be changed.

Before the connection is ready and tdb data structure is filled, a raw-key would be used during the process. The des_setkey() function is called to transform this raw-key into a key schedule. Then the key schedule is filled into the tdb $\rightarrow$ tdb_key_e

entry of the structure. For Triple-DES, tdb → tdb_key_e is an array of $3 \times 16 \times$ 64-bit for storing 3 different encryption keys with 16 rounds. For data alignment, each 48-bit subkey is stored in a 64-bit array element. There are 3 different keys used in Triple-DES so des_setkey () is called 3 times.

First of all, modification were made to the des_setkey() function. The 64-bit raw key is passed to the first array element without modification. Therefore calling des_setkey 3 times will result in raw key 1 being stored in tdb → tdb_key_e[0] and raw key 2 stored in tdb → tdb_key_e[15]. Finally, the last raw key is stored in tdb → tdb_key_e[31]. Other array elements are empty since are no longer used for key manipulation.

Then the interface for Triple-DES accelerator was embedded in des_ede3_cbc_encrypt(). Due to the fact that they have different data structures, conversion of data structure between inputs and outputs is necessary.

4.8 IDEA accelerator in FreeS/WAN

In FreeS/WAN, the available options for encryption are DES and Triple-DES. The IDEA accelerator was made in order to demonstrate the possibility for adding other encryption algorithms and as a high speed accelerator for FreeS/WAN. The IDEA ECB mode cipher that was discussed in Section 4.5 was implemented achieving 248 Mb/sec on Pilchard board, double the performance of the Triple-DES CBC core.

The IDEA accelerator interface is similar to Triple-DES CBC interface with a difference in key management. In IDEA, the determination of the decryption key-schedule was done in software. For an IDEA encryption, $16 \times 52 = 832$-bit is needed. The overhead for input of key-schedule is huge and as a result, the IDEA core has a hard-wired key-schedule for achieving high performance.

4.9 Summary

In this chapter details of the implementation for different cryptographic algorithms in different modes of operation were discussed.

A Triple-DES accelerator and an IDEA accelerator were implemented on Pilchard which employs Virtex-E™ XCV1000E FPGAs.

Between the Pilchard interface of the Triple-DES accelerator and LibDES, a different key representation was used. Since in hardware, data transfer and data storage should be minimized, a raw-key rather than a key schedule was used. LibDES was modified accordingly.

In the IDEA accelerator, the calculation of decryption key-schedule require iterations which is difficult to realize in hardware. The 832-bit key-schedule create large overhead on data transfer. As a result, key-schedule was hard-wired in the IDEA accelerator.

Chapter 5

Results

5.1 Introduction

In this chapter, results obtained from the Pilchard system are presented. Firstly, the benchmarking and testing environment is introduced. This is followed by performance measurements of the IDEA and Triple-DES accelerator. Finally, benchmarks using FreeS/WAN are presented.

5.2 Benchmarking environment

In this work, two computers set up with identical configuration were used for benchmarking and obtaining all results. These two computers connect to a 100Mbit network via a hub running FreeS/WAN version 1.5 with Linux Kernel 2.2.16 as shown in Figure 5.1.

CPU	P-III 866
RAM	128 MB
Motherboard	Asus CUSL2 (Intel 815EP chipset)
Network card	3COM 590 (100 Mbit network card)
OS	Mandrake v7.2 with kernel 2.2.16

Table 5.1: Configuration of machine for benchmark.

5.3 Performance of Triple-DES and IDEA accelerator

The Triple-DES and IDEA processors on Pilchard were verified using the Synopsys VHDL Simulator, and was synthesized using Synopsys FPGA Express 3.5 and Xilinx Foundation Series 3.3i, with Xilinx Virtex-E XCV1000E-6 as the target device.

Both processors were successfully implemented on Pilchard board. All implementations were tested using Pilchard card with a memory slot interface with an Xilinx Virtex-E XCV1000E-6 FPGA as Processing Element (PE).

5.3.1 Performance of Triple-DES core

Triple-DES core is made of three DES cores and therefore, the throughput of the Triple-DES core is directly proportional to throughput of DES core. A study of area and speed tradeoffs for a DES core with different number of rounds was conducted in order to choose an efficient Triple-DES core with high throughput. Table 5.2 show the performance of different DES cores in ECB mode.

Number of combinational rounds	Area (slices)	Clock rate	Throughput (Mb/sec)
1	747	58.42	233.68
2	765	51.3	410.4
4	877	23.38	374.08
8	1121	12.32	394.24
16	1666	5.94	380.16

Table 5.2: Area and Speed Tradeoff among DES core with different rounds

From Table 5.2, the performance is similar among the DES cores. Therefore, a core with 16 rounds was chosen since it has a simpler control and host interface.

The Triple-DES CBC core uses three combinational DES cores with 16 combinational rounds. It requires 5368 Virtex slices, which is 43.68% of the total 12288

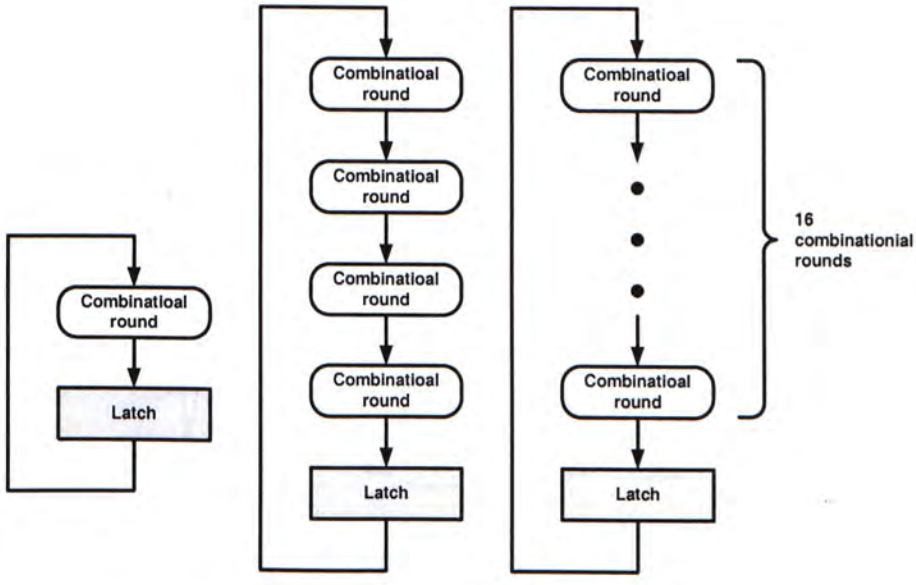

Figure 5.1: Architecture of the DES core with different number of combinational rounds

slices in a Xilinx Virtex-E XCV100E device, and operates at 2.135 MHz, achieving throughput of 2.135 MHz$\times$ 64-bit = 136.64 Mb/sec.

The Triple-DES accelerator was tested on the machine described in Table 5.1. Performance was taken as the time to process data using Triple-DES encryption. The Linux kernel function do_gettimeofday() was used for timing. Including software overhead, our Triple-DES accelerator achieves a measured throughput of 120 Mb/sec. According to figure 5.2, the performance of the Triple-DES accelerator for small amounts of data is much lower than software. As data size increases, the performance increases quickly and achieves a higher performance than software. This figure does not reach to the 136.64 Mb/sec performance above due to handshaking overheads.

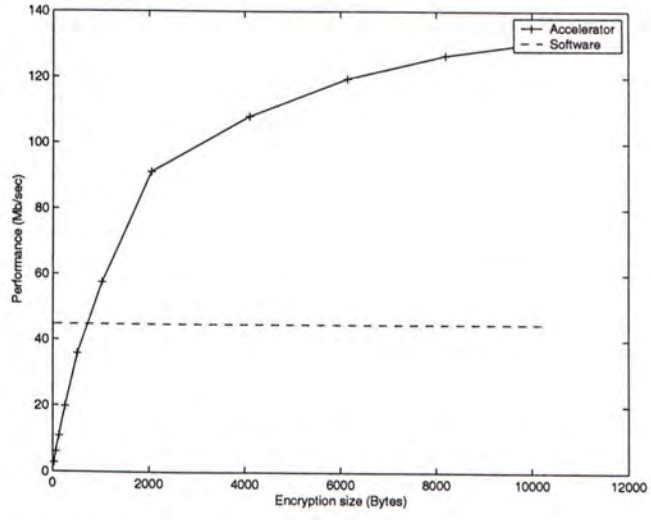

Figure 5.2: Performance of Triple-DES accelerator with different encryption size

5.3.2 Performance of IDEA core

The IDEA core is a fully-pipelined ECB implementation (8 rounds with output transformations), which requires 9568 Virtex slices and occupies 77.86% of a Xilinx Virtex-E XCV100E device. It achieves clock rate of 60.14 MHz. The expected throughput of the IDEA core is $\frac{175}{175+175}$ cycles $\times$ 60 MHz $\times$ 64 bits = 1920Mb/sec.

The IDEA accelerator was tested on the same machine and same setup as the Triple-DES accelerator. The IDEA accelerator achieves 248 Mb/sec with all overheads included. Compared to the expected performance of the IDEA core, it is relatively low. However, Pilchard board can only achieve around 248 Mb/sec for I/O access in uncachable mode.

5.4 Benchmark of FreeS/WAN

5.4.1 Triple-DES

In the series of tests for FreeS/WAN, the encryption standard was chosen to be Triple-DES and authentication algorithm MD5-96, which is referred as 3des-md5-96 in FreeS/WAN. 3des-md5-96 is the default encryption and authentication mode suggested by FreeS/WAN.

ttcp [ttcb, ttca] was used to measure the throughput of the benchmark and the benchmark was conducted for both TCP and UDP protocols. Different parameters for ttcp were selected and tested. However, the ttcp parameters did not have major effect on the benchmark. As a result, the following ttcp benchmarks were done using the default settings of 8192 (source buffer) and 2048 (network buffer) bytes respectively.

Another utility iperf was used to measure the throughput and similar test results were obtained as ttcp.

Protocol	Side	Throughput no FreeS/WAN (in Mb/sec)	Throughput FreeS/WAN (in Mb/sec)	Performance degradation (%)
TCP	sender	67.024	35.448	47.72
TCP	receiver	66.968	35.360	47.19
UDP	sender	93.848	45.560	51.45
UDP	receiver	93.536	45.536	51.32

Table 5.3: Benchmark of ttcp with/without FreeS/WAN

For every packet sent out in single way connection, an acknowledgment packet is received. The acknowledgment packet is small in size and which not favor the use of Triple-DES accelerator. In this work, the encryption of acknowledgment packet is handled by LibDES which has better performance when the encryption size is small. However, this factor limits the speed up of the use of Triple-DES accelerator in FreeS/WAN.

Protocol	Side	Throughput Mb/sec	Performance Improvement (%)
TCP	sender	45.788	29.1
TCP	receiver	45.660	29.1
UDP	sender	53.021	16.4
UDP	receiver	52.882	16.1

Table 5.4: Benchmark of ttcp with FreeS/WAN using Pilchard based accelerator

As shown in Table 5.3, throughput using IPSec is around 50 % of throughput without IPSEC. The performance of FreeS/WAN without IPSEC may represent the bandwidth of a 100 Mbit network with overhead. In theory, a 100 Mbit network offers 100 Mb/sec. This is slightly higher than the 93 Mb/sec performance without FreeS/WAN. Thus it can be seen that the performance of VPN using IPSec is limited by the speed of the software cryptographic library.

In Table 5.4, FreeS/WAN with the Triple-DES accelerator offers a 30 % speed up over the original software cryptographic library.

5.4.2 IDEA

Since FreeS/WAN does not have IDEA library, no software performance can be provided. However, as shown in Table 5.3, the performance of FreeS/WAN with software cryptographic functions is limited by the performance of LibDES. According to the IDEA library provided in openSSL written by Eric Young, the estimated performance of FreeS/WAN using IDEA is 116.32 Mb/sec. FreeS/WAN can offer same performance using this library as a 100Mbit network.

The IDEA accelerator was verified by using ping and tcpdump command. The receiver side use tcpdump to monitor the incoming packet pattern, and the sender use ping command to send out ping packet with special pattern.

5.5 Summary

In this chapter, the performance of Triple-DES and IDEA accelerator were presented. Also benchmarks of FreeS/WAN using the Triple-DES accelerator is discussed.

FreeS/WAN in IPSEC with LibDES achieves only 50% of performance as without IPSEC, this shows that the current software cryptographic library is not capable using in network applications. FreeS/WAN using our Triple-DES accelerator achieves 55% - 70% of the performance without FreeS/WAN, which is a 30% improvement over software.

Chapter 6

Conclusion

The objective of this thesis was to develop a FPGA-based accelerator for virtual private network. A study of FreeS/WAN and LibDES was conducted, which showed that the performance of FreeS/WAN is limited by the speed of the Triple-DES cipher. Therefore, an Triple-DES FPGA-based accelerator was proposed to increase the performance of FreeS/WAN.

Area and performance tradeoffs among different DES cores were studied. It was shown that DES core with a different numbers of combinational rounds give similar performance. However, the area of a DES core increases linearly with the number of combinational rounds. To simplify the implementation, a DES core with 16 combinational rounds was implemented. Besides DES core, various of hardware implementations of different ciphers were compared. The results concerning these candidates and the accelerator are as follows:

- hardware implementation of IDEA in ECB mode, DES in ECB mode, DES in CBC mode and Triple-DES in CBC mode were implemented on Pilchard, which populated with Xilinx Virtex-E XCV1000E device. The estimated throughput of the cores were 3848 Mb/sec, 1942.4 Mb/sec, 360 Mb/sec and 136.8 Mb/sec respectively.

- the cores were tested on the Pilchard platform in uncachable mode and the performance of IDEA in ECB mode, DES in ECB mode, DES in CBC mode

and Triple-DES in CBC mode were 248 Mb/sec, 248 Mb/sec, 248 Mb/sec, 120 Mb/sec respectively.

- a virtual private network (FreeS/WAN) was integrated with the FPGA-based accelerator and tested. Benchmarks showed that virtual private network offer 30% improvement using the hardware accelerator over a software library. Note that improvement of 89% is achievable for infinitely fast accelerator.

- the FPGA-based Triple-DES accelerator for VPNs offers an advantage of high computional power only for large data sizes. According to Figure 5.2, the hardware implementation is slower than software for small data size due to data transfer overheads. However, if the data size is large, the hardware implementation provides a three times speedup over software. Due to this issue, the overall speedup of the VPN accelerator application was lower than expected.

The bottleneck for VPN solutions was verified to be the encryption throughput. The Triple-DES and IDEA accelerator were implemented to increase the encryption throughput and hence the performance of VPN solutions. VPN solutions can attain the same speed as a 100 Mbit network if a faster Triple-DES and IDEA cipher were implemented. This work demonstrates the effect of employing cryptographic hardware in network applications.

6.1 Future development

Rijndael is announced to be Advanced Encryption Standard (AES), the new encryption standard as a replacement for DES and Triple-DES, by NIST in FIPS-197. Rijndael offers a faster hardware implementation compared with DES as well as a longer keysize. Therefore, it is feasible to implement a Rijndael accelerator for virtual private network.

As the number of system gates of available in an FPGA increase according to Moore's Law, it is feasible to implement several cryptographic algorithms in a single chip. Different cryptographic algorithms can be used for a virtual private network with different encryption options without reconfiguring the hardware.

In this work, the throughput of Triple-DES and IDEA accelerators are greatly below the maximum throughput that Pilchard card can attain because of handshaking overhead. If a network card interface and the corresponding TCP/IP and IPSec packet handling modules are all implemented in the FPGA, many transfers between Pilchard card and the host can be eliminated, greatly improving performance.

Bibliography

[Asc99a] Ascom. *IDEACrypt Coprocessor Data Sheet*, 1999. http://www.ascom.ch/infosec/downloads/IDEACrypt_Coprocessor.pdf.

[Asc99b] Ascom. *IDEACrypt Kernel Data Sheet*, 1999. http://www.ascom.ch/infosec/downloads/IDEACrypt_Kernel.pdf.

[BCF+91] H. Bonnenberg, A. Curiger, N. Felber, H. Kaeslin, and X. Lai. VLSI implementation of a new block cipher. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computer and Processors*, pages 501–513, 1991.

[CBK91] A. V. Curiger, H. Bonnenberg, and H. Kaeslin. Regular VLSI architectures for multiplication modulo $2^n + 1$. *IEEE Journal of Solid-State Circuits*, 26(7):990–994, July 1991.

[CBZ+93] A. Curiger, H. Bonnenberg, R. Zimmerman, N. Felber, H. Kaeslin, and W. Fichtner. VINCI: VLSI implementation of the new secret-key block cipher IDEA. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 15.5.1–15.5.4, 1993.

[CTLL01] O.Y.H. Cheung, K.H. Tsoi, P.H.W. Leong, and M.P. Leong. Tradeoffs in parallel and serial implementations of the international data encryption algorithm IDEA. In *Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES)*, pages 333–347. LNCS 2162, Springer, 2001.

[DH99] Naganand Doraswamy and Dan Harkins. *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1999.

[fre] http://www.free-ip.com/des/index.html.

[Fre00] FreeS/WAN org. Linux FreeS/WAN 1.5 HTML Online Document, 2000.

[GA99] M. George and P. Alfke. *Linear Feedback Shift Registers in Virtex Devices*. Xilinx, Inc., August 1999. Application Note XAPP210, Version 1.0.

[GSB+00] S. C. Goldstein, H. Schmit, M. Budiu, M. Moe, and R. R. Taylor. Piperench: A reconfigurable architecture and compiler. *Computer*, 33(4):70–77, April 2000.

[HC98] D. Harkin and D. Carrel. *The Internet Key Exchange (IKE) (RFC 2409)*, 1998.

[Hel] http://home.cyber.ee/helger/implementations/fastidea/.

[iet] http://www.ietf.org/rfc.html.

[KA98a] S. Kent and R. Atkinson. *IP Authentication Header (RFC 2402)*, 1998.

[KA98b] S. Kent and R. Atkinson. *IP Encapsulating Security Payload (ESP) (RFC 2406)*, 1998.

[KA98c] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol (RFC 2401)*, 1998.

[LCTL00] M. P. Leong, O. Y. H. Cheung, K. H. Tsoi, and P. H. W. Leong. A bit-serial implementation of the international data encryption algorithm (IDEA). In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 122–131, April 2000.

[lib] ftp://ftp.psy.uq.oz.au:/pub/crypto/des/libdes-x.xx.tar.gz.

[Lip98] Helger Lipmaa. Idea: A cipher for multimedia architectures? In *Selected Areas in Cryptography '98*, pages 253–268, August 1998.

[LLC+01] P.H.W. Leong, M.P. Leong, O.Y.H. Cheung, T. Tung, C.M. Kwok, M.Y. Wong, and K.H. Lee. Pilchard - a reconfigurable computing platform with memory slot interface. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, page (to appear), April 2001.

[MMF98] O. Mencer, M. Morf, and M. J. Flynn. Hardware software tri-design of encryption for mobile communication units. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 5, pages 3045–3048, May 1998.

[mmx] http://developer.intel.com/design/archives/processors/mmx/index.htm.

[MZ91] C. Meier and R. Zimmerman. A multiplier modulo $(2^n + 1)$. Diploma thesis, Institut für Integrierte Systeme, ETH, Zürich, Switzerland, February 1991.

[Nap00] Duncan Napier. Introducing FreeS/WAN and IPsec. *Sys Admin: The Journal for UNIX Systems Administrators*, 9(11):63, 65–69, November 2000.

[Nat94] National Institute of Standards and Technology (U. S.). Data Encryption Standard (DES). Federal information processing standards publication 46-2, National Institute for Standards and Technology, Gaithersburg, MD, USA, 1994. Supersedes FIPS PUB 46-1-1988 January 22. Category: computer security, subcategory: cryptography. Shipping list no.: 94-0171-P. Reaffirmed December 30, 1993.

[Nat99] National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). Federal Information Processing Standards Publication 46-3 (FIPS PUB 46-3), October 1999.

[Pat00] C. Patterson. High performance DES encryption in Virtex FPGAs using JBits. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 113–121, April 2000.

[pen00] *IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, 2000.

[RSA99] RSA Labs. DES III Challenge. 1999.

[SAF98] S. L. C. Salomao, V. C. Alves, and E. M. C. Filho. HiPCrypto: A high-performance VLSI cryptographic chip. In *Proceedings of the Eleventh Annual IEEE ASIC Conference*, pages 7–11, 1998.

[Sch96] B. Schneider. *Applied Cryptography*. John Wiley & Sons, second edition, 1996.

[ttca] http://www.cisco.com/warp/public/471/ttcp.html.

[ttcb] http://www.dtic.mil/ttcp/.

[Uni77] United States. National Bureau of Standards. *Data Encryption Standard*, volume 46 of *Federal Information Processing Standards publication*. U.S. National Bureau of Standards, Gaithersburg, MD, USA, 1977.

[vpna] http://www.cisco.com/univercd/cc/td/doc/pcat/3000.htm.

[vpnb] http://www.cisco.com/univercd/cc/td/doc/pcat/5000.htm.

[vpnc] http://www.cisco.com/warp/public/779/largeent/design/vpn.html.

[vpnd] http://www.intel.com/design/network/products/security/vpn3105y.htm.

[vpne] http://www.intel.com/design/network/products/security/vpn3110y.htm.

[vpnf] http://www.intel.com/design/network/products/security/vpn3125y.htm.

[WMSL95] S. Wolter, H. Matz, A. Schubert, and R. Laur. On the VLSI implementation of the international data encryption algorithm IDEA. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 1, pages 397–400, 1995.

[WPR+99] D. C. Wilcox, L. G. Pierson, P. J. Robertson, E. L. Witzke, and K. Gass. A DES ASIC suitable for network encryption at 10gbps and beyond. In *Proceedings of first International Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*, pages 37–48, 1999.

[Xil99] Xilinx, Inc. *Xilinx Libraries Guide*, 1999.

[Xil00a] Xilinx. Xilinx Libraries Guide Version 3.1i. 2000.

[Xil00b] Xilinx, Inc. *Xilinx Coregen Reference Guide*, 2000. Version 3.1i.

[ZCB+94] R. Zimmermann, A. Curiger, H. Bonnenberg, H. Kaeslin, N. Felber, and W. Fichtner. A 177Mb/sec VLSI implementation of the International Data Encryption Algorithm. *IEEE Journal of Solid-State Circuits*, 29(3):303–307, March 1994.

Publications

Full Length Conference Papers

- O.Y.H. Cheung, K.H. Tsoi, P.H.W. Leong, and M.P. Leong. Tradeoffs in parallel and serial implementations of the International Data Encrytion Algorithm IDEA. In Proceedings of the Cryptographic Hardware and Embedded Systems Workshop (CHES), pages 333–347. LNCS 2162, Springer, 2001.

- M.P. Leong, O.Y.H. Cheung, K.H. Tsoi, and P.H.W. Leong. A bit-serial implementation of the International Data Encryption Algorithm (IDEA). In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, pages 122–131, April 2000.

- P.H.W. Leong, M.P. Leong, O.Y.H. Cheung, T. Tung, C.M. Kwok, M.Y. Wong, and K.H. Lee. Pilchard - a reconfigurable computing platform with memory slot interface. In Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, pages (to appear), April 2001.

Submitted Papers

- O.Y.H. Cheung, and P.H.W. Leong. Implementation of an FPGA Based Accelerator for Virtual Private Networks. IEEE International Conference on Field-Programmable Technology (FPT), December 2002.