AN IMPLEMENTATION AND EXPERIMENTAL EVALUATION OF HARDWARE ACCELERATED CIPHERS IN ALL-PROGRAMMABLE SoCs ON EMBEDDED AND WORKSTATION COMPUTER PLATFORMS

by

RYAN A. COWART

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in
The Department of Electrical & Computer Engineering
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2017

In presenting this thesis in partial fulfillment of the requirements for a master's degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

_____ _____

(student signature) (date)

THESIS APPROVAL FORM

Submitted by Ryan A. Cowart in partial fulfillment of the requirements for the degree of Master of Science in Engineering in Computer Engineering and accepted on behalf of the Faculty of the School of Graduate Studies by the thesis committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate on the work described in this thesis. We further certify that we have reviewed the thesis manuscript and approve it in partial fulfillment of the requirements for the degree of Master of Science in Engineering in Computer Engineering.

Committee Chair

(Dr. Aleksandar Milenkovic) (Date)

(Dr. David Coe) (Date)

(Dr. Jeffrey Kulick) (Date)

Department Chair

(Dr. Ravi Gorur) (Date)

College Dean

(Dr. Shankar Mahalingam) (Date)

Graduate Dean

(Dr. David Berkowitz) (Date)

ABSTRACT

The School of Graduate Studies
The University of Alabama in Huntsville

Degree Master of Science in Engineering College/Dept. Engineering/Electrical & Computer Engineering

Name of Candidate Ryan A. Cowart
Title An Implementation and Experimental Evaluation of Hardware Accelerated Ciphers in All-Programmable SoCs on Embedded and Workstation Computer Platforms

The protection of confidential information has become very important with the increase of data sharing and storage on public domains. Data confidentiality is accomplished through the use of ciphers that encrypt and decrypt the data to impede unauthorized access. Emerging heterogeneous platforms provide an ideal environment to use hardware acceleration to improve application performance. This thesis explores the performance benefits of hardware accelerated ciphers versus their software counterparts for multiple cipher modes. The hardware accelerated ciphers are implemented on the FPGA fabric of the Zynq-7000 All-Programmable System-on-a-Chip (SoC) and utilizes DMA for communicating with the host processor. The design is implemented within an embedded and workstation computing environment. File encryption and decryption of varying file sizes and a hardware sink test are used as the workloads for testing the software and hardware ciphers, with execution time, speedup, and throughput as the metrics for comparing the performance of each. The performance evaluations show that the hardware accelerated ciphers performed significantly better than the software ciphers in the embedded environment with speedups upwards of 30x, but only achieved moderate improvements for the workstation environment with speedups upwards of 1.5x.

Abstract Approval: Committee Chair _____

Department Chair _____

Graduate Dean _____

ACKNOWLEDGMENTS

I would first like to thank my Lord and Savior Jesus Christ from whom all wisdom and knowledge originates (Proverbs 1:7). He has blessed me with the intellectual ability to pursue my graduate studies and has continuously blessed me and my family on a daily basis.

I also want to express my gratitude to those who have made this research and thesis possible. Foremost, I would like to thank my advisor, Dr. Aleksandar Milenkovic, for the opportunity to study under his mentorship, his initial ideas for the research, and for his continuous guidance and support throughout the process. I would also like to thank Dr. Kulick for loaning me the ZC706 development board for use in my research. I would like to thank Dr. Milenkovic, Dr. Coe, and Dr. Kulick for their time reviewing and providing feedback on my Thesis.

Most importantly I would like to thank my family, my wife Brooke and daughter Hadlie, for their unconditional love, support, and patience throughout the process. I consider the accomplishment of this Thesis just as much theirs as it is mine.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

The Digital Age has seen an ever increasing amount of data stored and transmitted across publicly accessed mediums, such as the Internet. A large portion of this data is confidential information that could harm individuals, corporations, and even governments if accessed by a malicious party. The critical importance of securing this data has led to the utilization of data encryption algorithms. Over time, there has been a multitude of cryptographic algorithms each designed with the goal of securing data more effectively. One such algorithm is the Advanced Encryption Standard (AES) which has emerged as a highly secure and easy to implement algorithm that is used by many corporations and government entities to secure their confidential information. AES, with the use of different cipher modes, has become the most widely used block cipher for securing information. Software implementations are the simplest and most common form of the AES algorithm; however, hardware implementations of the algorithm often improve speed, throughput, or save energy relative to their software counterparts, especially within an embedded computing environment with limited resources. The goal of this thesis is to explore any performance benefits of implementing hardware accelerated ciphers that use AES on an FPGA within an embedded and workstation environment. The rest of the Introduction section discusses the motivation, technology trends, the work done in the thesis and overview of the results, lists the contributions of this thesis, and gives the outline of the remainder of the thesis.

1.1 Need for Data Encryption

The cyber hacking of data and information has become a common occurrence in the twenty first century. So many companies and even governments store confidential information about their customers and constituents, respectively. Also, with the increased use of the Internet for things such as shopping, etc. the sharing of personal information on the Internet is on the rise. Unfortunately, there are individuals and groups of people that exploit vulnerabilities in the computing systems that store and transmit this confidential information in order to gain access to the data for personal gain. It has become very difficult to identify and correct all possible vulnerabilities of a system that contains this data especially as attackers continue to invent new techniques for attacking systems and mediums. Therefore, it has become extremely important to be able to protect the data itself regardless of the system it resides on. This is accomplished through the use of data encryption. Data encryption is the process of using a private key to transform readable data into a code that cannot be read or understood by a human or computing device. The only way for the encrypted data to be placed back into a readable form is to decrypt the data using the original private key used to encrypt the data. Therefore, if the private key is kept secret then only the original user can view the actual data. Many different encryption algorithms have been developed in the past few decades, such as DES, AES, IDEA, MD5, SHA 1, and many more. Some of the algorithms have emerged as being more secure and harder to hack than others. Data encryption does not control who can access data and information, but rather it controls who can view it. Indeed, as long as data is correctly encrypted before being transmitted or stored

on an accessible medium or device then even if another party gains access to the data it will be in an unknown and unusable form.

1.2 Technology Trends

Software implementations of cipher modes and block ciphers have long been the most common and easiest form of protecting data. However, the requirements for higher bandwidth and performance and lower power consumption in modern applications led to more hardware implementations of these ciphers. Most notably, Intel introduced the AES New Instructions (AESNI) that extends the x86 instruction set architecture. AESNI instructions invoked by software rely on dedicated hardware in the chipset that is used to perform the cryptographic operations. This allows for the cryptographic operations to be performed much faster than in software implemented algorithms. The computing industry has also seen a significant rise in interest and manufacturing of heterogeneous computing systems that offer more flexibility to developers and can achieve higher computational and data throughputs than a homogeneous system. These heterogeneous platforms may have alternative processors on the same integrated circuit (IC) as the main processor or off-chip to the main processor. An example of an off-chip heterogeneous system is a workstation that contains a general purpose graphics processing unit (GPGPU) that resides on the PCIe bus. An example of an integrated heterogeneous platform is an All-Programmable System-on-a-Chip (APSoC), which contains a hard processor system and FPGA fabric on the same silicon die. These all-programmable chips allow designers to combine the strengths of the software programmability of a hard processor system and the hardware programmability of the FPGA fabric. A common design approach is to offload computational overhead from the hard processor

system to a hardware accelerator in the FPGA fabric to perform tasks faster and more efficiently.

1.3 What Has Been Achieved

In this thesis, an experimental performance evaluation of software and hardware implementations of three AES-based cipher modes is performed. The ciphers are evaluated within an embedded and workstation environment. The three cipher modes that are implemented are the Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Counter (CTR) mode. Each of the cipher modes uses AES as the block cipher. The OpenSSL open-source cryptographic library is used for the software implementation of each cipher and the hardware implementations are hosted on the FPGA fabric of the Zynq-7000 All-Programmable System-on-a-Chip. There are two software applications for testing the performance of the ciphers. One application is multithreaded whose purpose is to attempt to achieve maximum throughput for the hardware accelerated ciphers. The second application uses an extension added to OpenSSL to simplify the programming API used for accessing both software and hardware ciphers. Furthermore, there are two tests executed within each software application. A file encryption/decryption test and a hardware sink test. The file encryption/decryption test is designed to evaluate the performance of each cipher in the scenario of protecting data at rest and the hardware sink test is designed to test each cipher in the scenario of protecting data prior to sending data out from the computer. The performance of the software and hardware ciphers are compared using metrics of execution time, speedup, and throughput.

1.4 Contributions

This thesis makes the following contributions to the field of hardware accelerated cryptographic processing and heterogeneous computing on embedded and workstation computing environments:

- Provides an accurate performance comparison of highly optimized software cipher implementations and FPGA hardware accelerated cipher implementations using modern processors and FPGAs.
- Evaluates the potential performance improvements of hardware accelerated ciphers on embedded and workstation platforms through the use of heterogeneous computing.
- Evaluates the effects of bus architectures, system hard drives, and FPGA families on the performance of hardware accelerated ciphers.
- Creates environment for experimental-based evaluation of applications utilizing software and hardware accelerated ciphers.

1.5 Outline of the Thesis

The remaining sections of this thesis are organized as follows. CHAPTER 2 discusses the background and motivation for the thesis. CHAPTER 3 discusses the related work from the open literature. CHAPTER 4 discusses the methodology for how the thesis work is accomplished. CHAPTER 5 discusses the specifics of the system designs for the embedded and workstation designs. CHAPTER 6 discusses the architecture and flow of the software applications used for testing and measuring the performance of the cipher implementations. CHAPTER 7 provides details regarding the embedded and workstation experimental environments for testing the implementations. CHAPTER 8 discusses the results of the micro-

benchmarking tests on different components of the system designs. CHAPTER 9 discusses the results for each test and software application combination. Finally, CHAPTER 10 discusses possible future work and conclusions drawn from the thesis work.

CHAPTER 2

BACKGROUND AND MOTIVATION

Data security is the process of protecting private or confidential information from being accessed by an unauthorized party on a public domain or private computer. Computers are owned in many more households and businesses now than ever before and they all contain personal information of individuals or proprietary information of companies and needs to be protected. Information is also transferred across public communication domains in extremely large amounts on a daily basis where some of the information transferred is not sensitive; however, much of the information is confidential to one or both of the communicating parties. Some of the information that is passed across the Internet domain includes banking information, personal identifiable information, tax information, and much more so it is very important to use data sharing techniques and algorithms in order to protect this information to guarantee that a malicious party that intercepts the data cannot read the information. A technique that has been used for many years is data encryption. There are many different data encryption algorithms and have all been successful at protecting the data entrusted to their algorithm. However, some have been more successful than others due to the evolution of more sophisticated software and hardware attacks against computers, networks, and the encryption algorithms themselves. Some examples of the software attacks are ciphertext only attack, known plaintext attack, chosen plaintext attack, chosen ciphertext attack, side channel attacks, brute force

attack, and birthday attacks. Examples of hardware attacks are man-in-the-middle attack, electromagnetic attack, and power analysis attacks [1].

Users can either use software-based or hardware-based encryption to protect data. Software encryption programs or much more prevalent than hardware solutions because it is cost effective, easily distributed, easy to use, upgrade, and update so it makes a good solution for individuals as well as large companies. Software encryption solutions are also readily available for all major operating systems. However, software solutions tend to only be as strong as the operating system of the base device and can also be very computationally intensive. Software solutions also have the possibility of being turned off by users or circumvented by attackers making them extremely vulnerable to attacks. The other option is hardware-based encryption solutions. Hardware solutions tend to be self-contained and does not require any additional software support making it essentially free from the possibility of contamination, malicious code infection, or vulnerability [2].

2.1 Advanced Encryption Standard (AES)

Symmetric cryptography is split into block ciphers and stream ciphers. Stream ciphers encrypt bits individually, as shown in Figure 2.1. This is achieved by adding a bit from a key stream to a plaintext bit. On the other hand, block ciphers encrypt an entire block of plaintext bits at a time with the same key, as shown in Figure 2.2. This means that the encryption of any plaintext bit in a given block depends on every other plaintext bit in the same block. Block ciphers are much more common than stream ciphers for public communication domains such as the Internet. Stream ciphers are smaller, faster, and better suited for embedded devices

with little computational power; whereas, block ciphers require more compute and storage resources and are thus more suited for workstation environments [3].

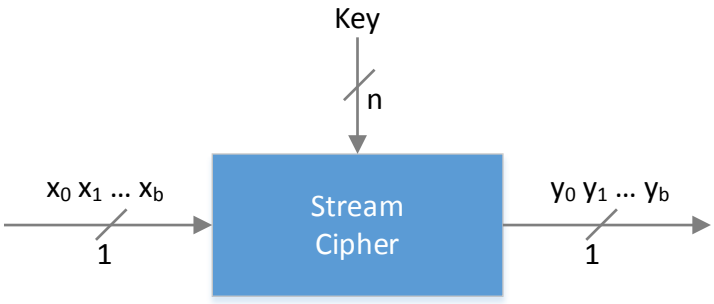


Figure 2.1 Stream Cipher [3]

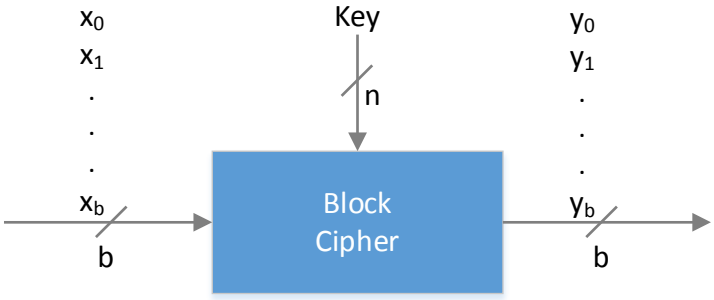


Figure 2.2 Block Cipher [3]

The Advanced Encryption Standard (AES) is the most widely used symmetric block cipher today. It is the standard block cipher for the US government and is also used in many industries and commercial systems. AES is used as the encryption standard for many protocols including Internet Protocol Security suite (IPsec), Transport Layer Security (TLS), the WLAN security protocols (IEEE 802.11i), and the secure shell network protocol (SSH) [3].

AES was developed by Joan Daemen and Vincent Rijmen and was adopted by the US National Institute of Standards and Technology (NIST) in 2001 as the new

encryption standard for the US government. AES has three different key sizes of 128, 192, and 256 bits. The block size is 128 bits, or 16 bytes. A basic block diagram of AES is shown in Figure 2.3. The number of internal rounds of the cipher is a function of the key length and it is 10, 12, or 14 rounds for key length sizes of 128, 192, and 256 bits, respectively. The AES algorithm consists of so-called layers where each layer manipulates all 128 bits of the data block. The data block is also referred to as the state. There are three different layers and each round of the algorithm, with the exception of the first round, executes all three layers. The three layers are the Byte Substitution layer (S-Box), Diffusion layer, and the Key Addition layer. The Byte Substitution layer is where each byte in the state is substituted with another byte from a known lookup table that contains values with special mathematical properties. This byte substitution is a nonlinear transformation. The Diffusion layer has two sublayers of ShiftRows and MixColumn which perform linear operations on the state. The ShiftRows sublayer permutes the data on a byte level and the MixColumn sublayer is a matrix operation that combines blocks of four bytes used for mixing the data. Lastly, the Key Addition layer is where a 128-bit round key, derived from the main key, is XORed with the state. The round keys, or subkeys, for each round are generated by taking the original cipher key, any of the three lengths, and performing the key scheduling operation to produce a unique 128-bit subkey [3].

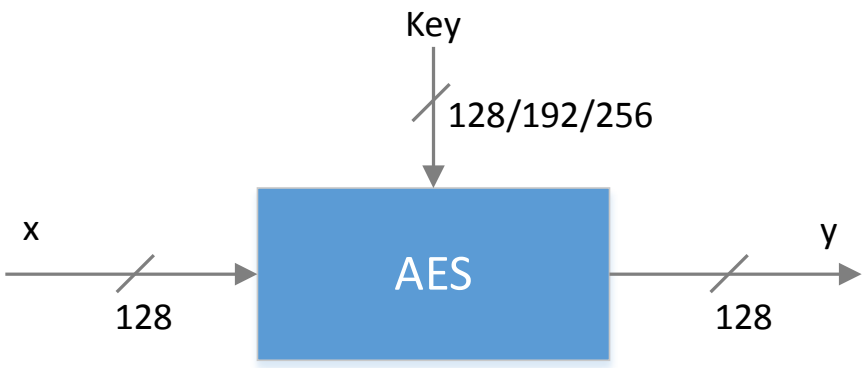


Figure 2.3 AES Block Cipher [3]

2.2 Cipher Modes of Operation

Block ciphers, such as AES, can be used as the base building block for a multitude of encryption schemes and to even create stream ciphers. The different ways of encryption are referred to as modes of operation. For this research, the modes of operation that were implemented were the Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Counter Mode (CTR). AES is the block cipher used to implement the different modes of operation.

2.3 Electronic Codebook (ECB)

Electronic Codebook (ECB) is the most straightforward and simple way of encrypting data. Figure 2.4 shows the functional block diagram of the ECB cipher mode. ECB just uses the cipher key to encrypt each block of data. Each block of data is encrypted or decrypted independently from each other with the same key. Thus, data synchronization between the sender and receiver is not necessary for the ECB mode because even if some encrypted blocks are not fully received the rest of the encrypted data is not corrupted and can still be fully decrypted. ECB mode also has the ability to be parallelized since there are no data dependencies between different

blocks of data. However, for this research, the ECB mode was not parallelized due to hardware resource constraints which will be discussed in a later section. On the other hand, the ECB cipher has a few cryptographic weaknesses. The biggest weakness is that the cipher mode is highly deterministic. In other words, the same block of plaintext data will be encrypted identically to produce the same output ciphertext. Therefore, a potential attacker could perform a traffic analysis attack by just looking at the output ciphertext to determine if the same data was used multiple times. Next, due to the lack of data dependency between the data blocks and the absence of a message authentication code, the data blocks could be reordered or replaced by an attacker and the receiver would have no means to be able to detect it [3].

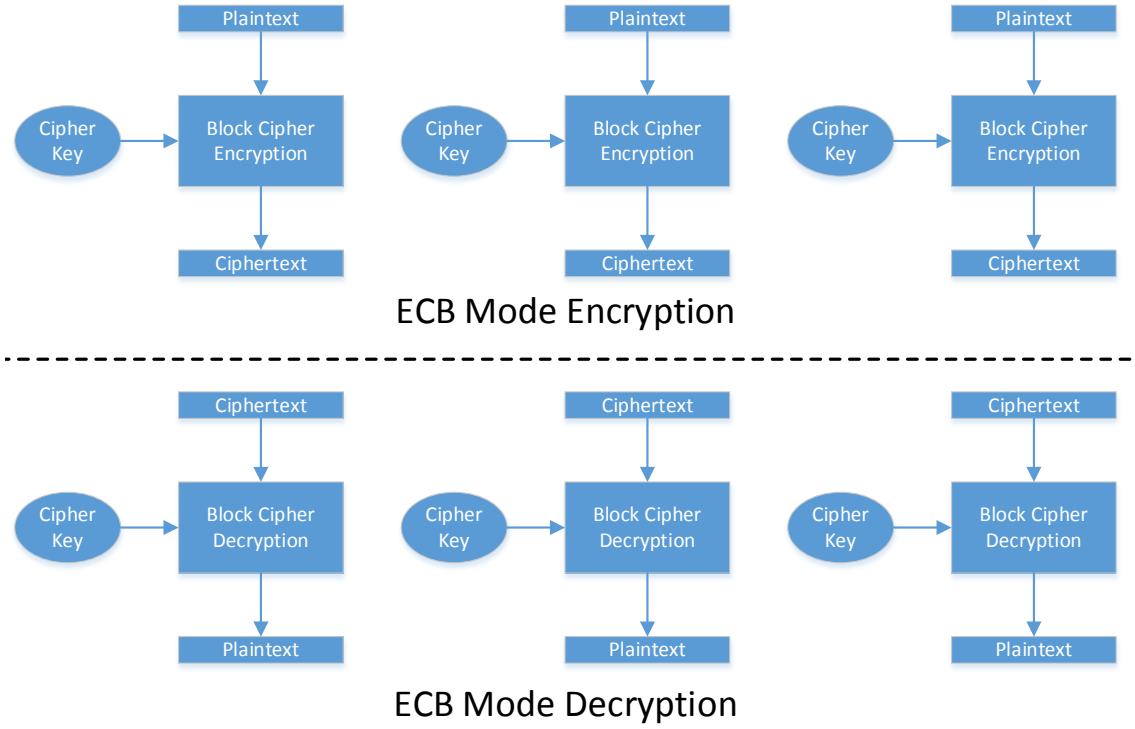


Figure 2.4 ECB Encryption/Decryption Block Diagrams

2.4 Cipher Block Chaining (CBC)

Cipher Block Chaining (CBC) is a mode in which encryption of all the blocks are chained together such that the current block is not only dependent on the previous encrypted block, but also on every other encrypted block before it. CBC uses a cipher key along with an initialization vector (IV) in order to encrypt the data. Figure 2.5 shows the functional block diagram of the CBC mode. The IV is used at the beginning of the encryption operation to XOR with the first plaintext block which is then passed through the block cipher, or AES in this case, to produce the output ciphertext. Furthermore, for each additional block of data to follow, the output ciphertext from the previous block is XORed with the current plaintext block before passed through the block cipher. The decryption process of CBC is just the inverse of the encryption process. The ciphertext is sent through the block cipher for decryption and then the output of the block cipher is then XORed with the input ciphertext from the previous data block. The last data block is XORed with the original IV to decrypt the data block. The end result is the original decoded data set. The strength of the CBC mode comes when a unique IV is used to encrypt each new data set. This is important because the same data set can be encrypted with a different IV and the resulting ciphertext outputs will be completely different which makes it impossible for an attacker to perform any sort of pattern detection. However, the IV does not have to be kept secret; only the cipher key must be kept secret [3].

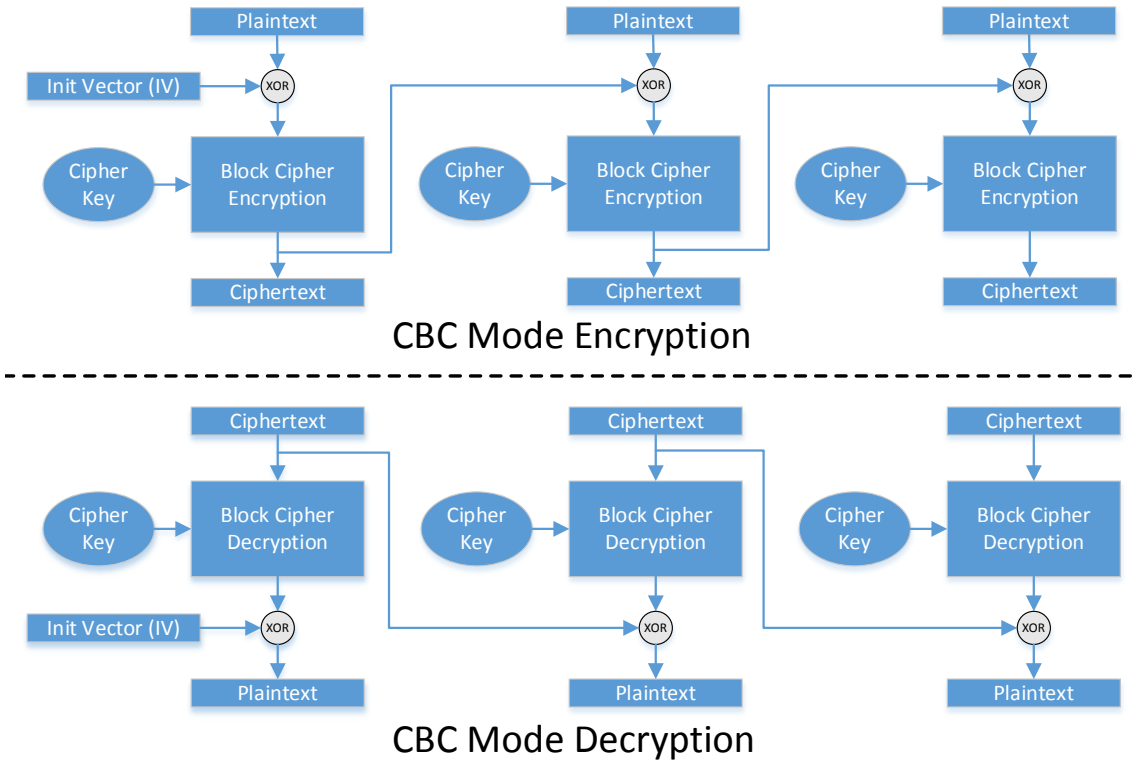


Figure 2.5 CBC Encryption/Decryption Block Diagrams

2.5 Counter Mode (CTR)

The Counter Mode (CTR) uses a block cipher to implement a stream cipher. CTR mode uses a cipher key, an initialization vector (IV), and a counter value. Figure 2.6 shows the functional block diagram of the CTR cipher mode. The cipher key is used in the block cipher to perform the encryption operation on the input data into the block cipher. The input into the block cipher is a combination of the IV and the counter value. The values can be combined in one of two ways. Either the concatenation of the two values or by XORing the two values to create the 128-bit input into the block cipher. It is extremely important that the same counter value is not used more than once during the encryption of a data set because if an attacker knows one of two of the plaintext data blocks that were encrypted with the same

input to the block cipher (i.e. the same counter value) then he could compute the key to the block cipher and have the ability to decrypt all the other ciphertext output blocks. In order to guarantee a unique counter value for each block of data a large bit-width counter is used, such as 32 or 64 bits wide, so that a very large set of data (greater than ~32GB) can be encrypted before having to change the IV. The counter value is just incremented by one with each new block of data and then concatenated with the IV. The concatenated value is then encrypted by passing through the block cipher. The encrypted output from the block cipher is XORed with the input plaintext block of data which produces the output ciphertext. This process is completed for each input plaintext block. The CTR mode is highly parallelizable and does not contain any data dependencies or feedback requirements between the different blocks of data. The only requirement is that the same IV and counter value combination cannot be used to encrypt two different blocks of data within the same data set [3].

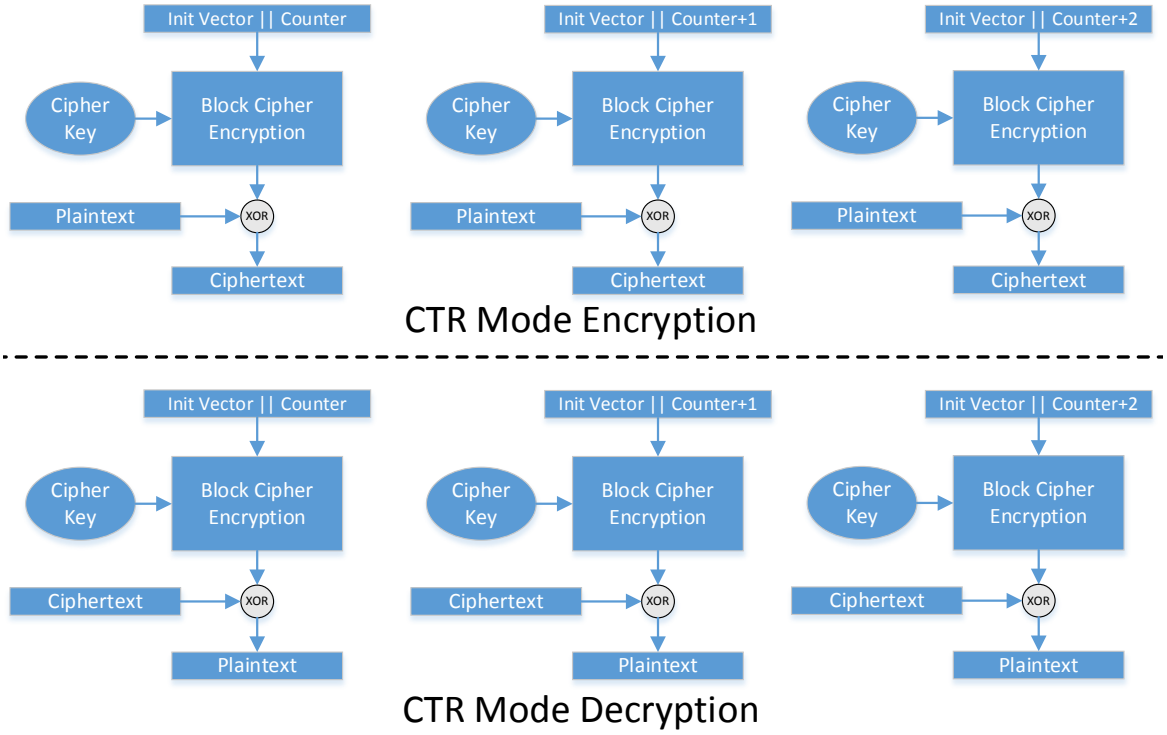


Figure 2.6 CTR Encryption/Decryption Block Diagrams

2.6 OpenSSL

The Secure Socket Layer (SSL) and Transport Layer Security (TLS) are used to secure applications that need to communicate over a network to ensure that the integrity of the data is upheld. OpenSSL is an open source library that implements the TLS and SSL protocols. OpenSSL is by far the most widely deployed, freely available implementation of these protocols. It also serves as a general-purpose cryptography library for numerous different types of block and stream ciphers [4]. The cryptographic library provides the most popular algorithms for symmetric key and public key cryptography, hash algorithms, and message digests, and a pseudorandom number generator. The library is full-featured and cross-platform, working on Windows, Linux, and Mac OS X operating systems (OS). The OS type is

detected at compile time so that the software is compatible and the compiler can optimize the library for the specific OS. The library is mainly used from within C and C++ applications, but can also be used within other languages such as Python, Perl, and PHP. One can also use OpenSSL from the command line [5].

In the context of this research, OpenSSL is used solely for its cryptographic functionality, specifically the highly optimized AES implementations. The reason OpenSSL is used is because it has become an industry standard for software implementations of cryptographic algorithms that can be used for both desktop and embedded environments. The library provides a high level Application Programming Interface (API) for interfacing to the different cryptographic functions. The high level API is known as EVP which provides the developer with similar function calls regardless of the cipher mode of operation in use. The library is also used for generating random cipher keys and initialization vectors during the encryption process by using a Password Based Key Derivation Function (PBKDF) available in the API. The PBKDF function implements a secure hashing algorithm for generating the cipher key and initialization vector [5]. The user has to then specify the correct password when decrypting data in order for the correct cipher key and initialization vector to be used to correctly decode the data.

2.7 All-Programmable SoCs (Zynq-7000)

Heterogeneous computing has emerged as computing platforms of choice that harness the Moore's Law to offer increased level of integration, customization, and computing power at minimal power consumption. Modern processor chips begin to reach their power consumption limits, which is inhibiting the clock rates from increasing any further. Heterogeneous computing allows for increasing

computational bandwidth, but does so with minimal power consumption compared to its homogeneous counterpart. An All-Programmable System-on-a-Chip (APSoC) is a type of heterogeneous computing platform that contains a hardened central processing unit (CPU) and a programmable logic fabric such as an FPGA. An example of an APSoC is the Zynq 7000 designed and manufactured by Xilinx, Inc.

The Zynq 7000 APSoC is a single integrated circuit that contains a hardened processor system (HPS) that includes a dual core ARM Cortex A9 and a programmable logic (PL) fabric to create a full heterogeneous computing system. Figure 2.7 shows an overview of the system architecture of the Zynq-7000 APSoC. The dual core ARM processors include multi-level cache hierarchy that maintains coherency between the two CPU cores, 256 KB of on-chip memory, 512 MB of DDR3 external memory, 8 channel DMA controller, vector processing units, and a large set of peripheral connectivity interfaces. Its peripheral interfaces include a gigabit Ethernet port, USB interfaces, CAN bus interfaces, SD card interface, and I2C interfaces. The PL, or FPGA fabric, on the Zynq 7000 is comparable to that of either the Artix-7 or Kintex-7 depending on the chip version. The FPGA fabric contains a host of block RAM units, hundreds of DSP slices, programmable I/O blocks, JTAG interface, PCI express block, high speed serial transceivers, and two analog-to-digital converters. It provides a low power and high design flexibility for embedded designs with its large number of resources [6]. The ARM cores and the programmable logic communicate via a version of the Advanced Microcontroller Bus Architecture (AMBA) known as Advanced Extensible Interface (AXI). The AXI bus protocol provides a separate address and data channel for both the read and write operations and has a data width of up to 64 bits. The processors and FPGA can communicate in both directions simultaneously without any loss of throughput. The

Zynq contains both general purpose and high performance ports for communication with the FPGA fabric. The general purpose ports are 32 bits wide, where the ARM cores are the master and the PL is the slave. The high performance ports are 64 bits wide where the PL is the master and the memory interfaces on the HPS are the slave.

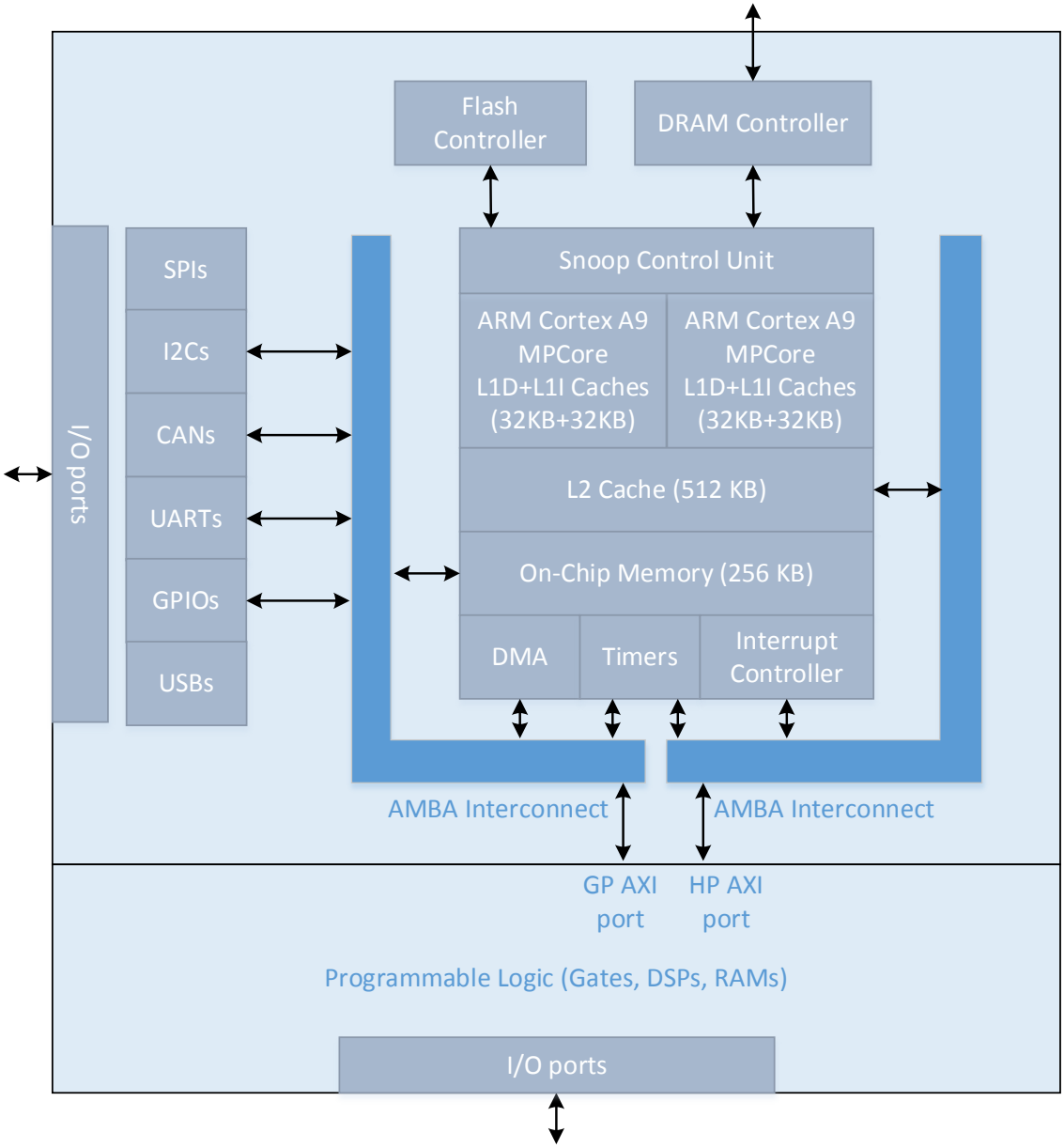


Figure 2.7 Zynq-7000 APSoC System Architecture

2.8 Xillybus

Xillybus is an open source Intellectual Property (IP) core developed by Xillybus, Ltd. that implements the necessary logic for the data transfer between IP cores implemented on a Field Programmable Gate Array (FPGA) and a host processor running Linux or Microsoft Windows [7]. Xillybus is provided with both a hardware IP core and a kernel driver module for full interaction between the host and FPGA. The host application and the FPGA design interact with well-known interfaces. The FPGA application logic connects to the Xillybus IP core through standard FIFO interfaces. The host application performs basic file I/O operations on pipe-like device files (i.e. open, read, write, close) such that there is no specific API for the Xillybus driver [8].

Xillybus works on both Xilinx and Altera FPGA's and System-on-a-Chip (SoCs). It is compatible with transport protocols of PCIe, AXI3, and AXI4. It can achieve maximum data throughput of 3.5 GB/s simultaneously in both directions depending on the FPGA and host capabilities [9]. It is compatible with Linux operating systems with kernel version greater than or equal to 2.6.36 and on Windows 7, 8, and 10. The applications that Xillybus is most suited for are data acquisition and playback, interfacing with hardware, custom computer peripherals, in-hardware logic verification, and coprocessing [7].

The AXI4 bus logic is for implementation on a Xilinx chip such as the Zynq-7000 APSoC and the PCIe bus version is for implementation within a workstation environment. The Xillybus IP core that resides in the FPGA has the ability to function as a slave or a master on the AXI4 bus. This dual functionality is very useful because Xillybus utilizes Direct Memory Access (DMA) in both the embedded

and workstation designs to move data between the processor and FPGA with minimal processor overhead [10]. Xillybus also provides a slower, memory mapped interface between the host and FPGA for basic register level access.

DMA is the process of transferring data to or from some destination without the interaction of the CPU. DMA not only decreases the computational load of the CPU, but also allows for an increase in the data throughput achieved by the communication interface because it makes use of burst transactions. With a burst transaction a large set of data is transferred from a source to a destination with a single address phase occurring on the bus, thus effectively increasing the effective bus throughput. If the CPU is involved in the data transfer it has to perform an addressing phase on the bus for every data transfer. This ultimately decreases the overall data throughput achieved on the bus which is why DMA is very appealing in high throughput applications.

Figure 2.8 shows a high level block diagram for Xillybus with the AXI4/PCIe bus. In the Xillybus design, the processor can initiate a DMA transaction when transferring data to the FPGA which behaves as the slave on the bus. The host processor will provide the DMA engine on the HPS with the memory address of the start of the data set to be transferred and a DMA burst transaction will transfer the data across the bus. A similar process is used for transferring data from the FPGA to the host processor. The user FPGA code just streams data into the provided Xillybus FIFOs, shown in Figure 2.8 as the Application FIFOs, and Xillybus uses an AXI DMA engine for initiating a DMA transfer across the bus. The data from the FPGA is transferred into a memory location in DRAM and the host processor can then access it. This means that the latencies experienced by the processor(s) when communicating with the FPGA is mainly affected by the latency of interfacing to the

DRAM and only when the DMA buffers are full is the latency affected by the actual bus latency. This significantly improves performance over the case where the processor is handling all transactions over the AXI4 bus since the processor does not have to wait for AXI4 transactions to complete before continuing with execution. Xillybus makes the communication across the AXI4 or PCIe bus transparent to the software/hardware developer which increases design simplicity and decreases development time. The only difference is in the bus protocol and possibly the DMA engine hardware on the host.

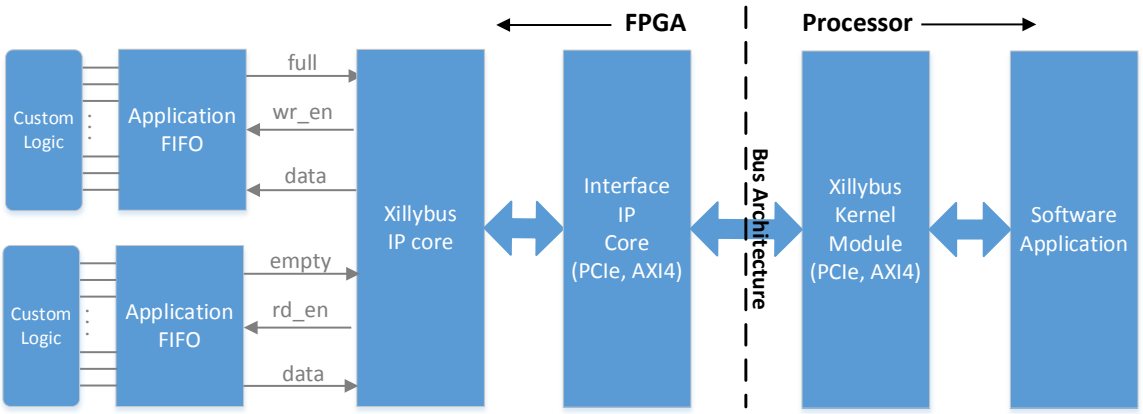


Figure 2.8 Xillybus Functional Block Diagram

The Xillybus website [7] provides a link called the IP Core Factory where one can create a custom IP core and kernel driver with any number of interfaces [11] [12]. The core designer can specify the interface names, bit widths, their purpose (i.e. coprocessing, data acquisition, video processing, etc.), and their expected bandwidth requirements. The website then uses this information to auto generate the VHDL and Verilog code for the FPGA IP core along with the kernel software driver code for interfacing with the FPGA core. This provides the user with the software and

hardware interface solutions for efficient communication between the processor and FPGA for a wide range of applications.

For this research, both the Xillybus AXI4 and PCIe versions are used for implementation on the Zynq-7000 APSoC on the Zedboard and ZC706 development boards, respectively. The use of the Xillybus IP core and kernel driver software greatly simplify the design process for the cipher hardware acceleration because all the necessary code for communicating between the processor and FPGA is provided, so more time and effort is spent on the development of the other system components.

2.9 Opportunity: Cryptographic Hardware Acceleration

Hardware accelerated cryptographic functionality is appealing because it not only can provide a stronger and more resilient form of security, but can also improve system performance and power consumption when performing cryptographic operations. Embedded computing is the most common form of computing, but it contains the least amount of resources and computing power compared to workstations, servers, etc. Therefore, the task of performing cryptographic operations on large amounts of data can be a very consuming task for embedded processors. Therefore, the possibility of offloading these cryptographic operations from the embedded CPU onto another form of computing platform can return large dividends in terms of performance and power consumption. Heterogeneous computing platforms provide and ideal environment for offloading such operations to a secondary computing platform. In this specific example, the cryptographic operations can be offloaded onto the secondary processing unit. Modern heterogeneous platforms, such as the Zynq-7000 APSoC, offer chipsets containing both a CPU and FPGA on the same die. The FPGA fabric of the Zynq can be used to

implement the cryptographic operations which has the potential to perform the operations faster while consuming considerably less power than the CPU.

CHAPTER 3

RELATED WORK

There has been an exceptionally large amount of research work done in the area of hardware acceleration for cryptographic operations, specifically AES. The majority of the accelerator implementations were completed on an FPGA and others on a general purpose graphics processing unit (GPGPU). Most of the implementations included multiple cipher modes such as ECB, CBC, CTR, GCM, and XTS that use AES as the block cipher.

Some of the prior AES coprocessor designs used soft-core processors in an FPGA for interfacing to an AES hardware core. Hodjat et. al. [13] [14] used the LEON soft processor in the ThumbPod SoC to implement the ECB, CBC, and OFB ciphers through a memory-mapped interface. They used a non-pipelined AES core clocked at 330 MHz to achieve a maximum throughput of 3.84 Gbit/s. Baskaran et. al. [15] implemented the AES block cipher using the Picoblaze microprocessor and other hardware cores on a Spartan 3E in order to achieve a very low-cost resource cryptographic design of only 460 slices on the FPGA. These softcore design approaches used the FPGA for the microprocessor and AES implementations with custom software executed on the soft-core microprocessor specifically designed for interfacing to the AES or cipher hardware implementations.

Other designs extended certain cryptographic software libraries to target the hardware accelerators as opposed to developing custom software for accessing and utilizing hardware coprocessors. Pedraza et. al. [16] ran Linux on a PowerPC hard

processor core using a Virtex II FPGA and extended the functionality of the CryptoAPI Linux cryptographic library to utilize the AES and DES hardware accelerators for implementing a secure file system. They were able to achieve a maximum throughput of 100 MB/s. Nambiar et. al. [17] extended the encryption function of the OpenSSL cryptographic library on the NIOS II soft-core microprocessor running uClinux real-time operating system (RTOS). It utilized a memory-mapped interface to the AES core inside the FPGA running at 50 MHz. They achieved a 2-3 times improvement over the full software implementation of OpenSSL. Hodjat et. al. [18] interfaced a hard CPU processor to an AES FPGA hardware accelerator for use in VPN and IPSec applications. They implemented the ECB, CBC, CTR, and CCM ciphers. They were able to achieve a throughput of 3.43 Gbit/s with a power consumption of 86 mW. Irwansyah et. al. [19] extended the instruction set of the Nios II reduced instruction set computer (RISC) processor to support AES encryption and decryption. The designs in [17] and [19] only implement the base AES encryption and decryption algorithms and not full cipher modes such as ECB or CBC.

An initial version of just the embedded design from this research was completed and published by myself et. al. [20] at an ACM Southeast Conference. This initial design did not have the ability to dynamically provide a private key or initialization vector to the FPGA from software, was only implemented on the Zedboard embedded platform, and did not include the necessary hardware and software for conducting the hardware sink test present in this Thesis.

The research presented in this Thesis is different than the previously mentioned designs in several different areas. The differences include that it uses a more modern and powerful chipset in the Zynq-7000 APSoC, it uses direct memory

access (DMA) for transferring data to/from the hardware ciphers, implements the design in both an embedded and workstation environment, utilizes two different bus architectures for data transfer. The Zynq-7000 APSoC contains both a hardened dual core ARM processor with an FPGA fabric on the same chip. Pedraza et. al. [16] used a PowerPC RISC hard processor on a Virtex II FPGA, but this research utilizes the ARM cores of the Zynq-7000 which are more powerful and full featured than the PowerPC processor. The FPGA fabric of the Zynq-7000 is also larger with more modern fabric technology than the legacy FPGAs that were used in the previously discussed designs. Another difference between this research and past research is the use of DMA for transferring data to/from the FPGA hardware accelerator. Most of the previous work used some form of a memory-mapped interface for interfacing the processor to the FPGA cores that caused the CPU to play a role in the transfer of data. The use of DMA in this research allows for data to be transferred to/from the FPGA without the CPU having to participate in the transfer. This increases the effective throughput of the data transfer and decreases the load on the CPU. This research also implements the same hardware acceleration design on both an embedded and workstation platform. The embedded implementation uses a Zedboard and the workstation implementation uses a Dell Precision Tower 7910 with the ZC706 development board. Both designs use the Zynq-7000 APSoC for implementing the hardware acceleration on the FPGA with either the ARM cores or Intel cores as the host processors for the embedded and workstation implementations, respectively. All the previously discussed research implemented the designs within an embedded environment, but did not attempt to implement the designs on multiple platforms. Based on the platform that the design is implemented on also determines the bus architecture that is used for transferring the data to/from

the FPGA hardware accelerator. The embedded platform utilized the AXI4 bus within the Zynq-7000 SoC, but the workstation platform utilized the PCIe bus. Even though a different bus architecture is used for each platform implementation, the overall system design and architecture of the hardware acceleration and the software for driving the experimental setup is the same. This portability of the design is possible through the use of Xillybus. This portability is unique to this research as none of the previously discussed research made any mention as to their designs being compatible on multiple computing platforms. This research also implements multiple ciphers; ECB, CBC, and CTR ciphers, whereas, some of the previous work only implemented and tested the AES block cipher independent of any cipher implementation. Nambiar et. al. [17] extended just the encryption function of the AES block cipher with the OpenSSL library; however, this research extends both the encryption and decryption function of the ECB, CBC, and CTR ciphers.

CHAPTER 4

METHODOLOGY

The main goals of this research is to experiment with the possible performance improvements of hardware accelerated ciphers compared to the equivalent software-only counterparts. There are three cipher modes implemented and tested for this research; the ECB, CBC, and CTR cipher modes with the 256-bit key AES as the block cipher. All three of the cipher modes are implemented in both software and hardware. The software ciphers use the OpenSSL cryptographic library while the hardware ciphers reside on the FPGA fabric of the Zynq-7000 APSoC. In order to test the software and hardware ciphers, two different tests are executed inside the software applications. One test is used for performing file encryption and decryption while the second test is used for sending encrypted/decrypted data to a hardware sink. The file encryption/decryption test evaluates the full functionality of the cipher modes implemented in software and hardware along with the performance of each. It is also used to test the use case of encrypting/decrypting data at rest on a computing platform, such as data on the hard drive. On the other hand, the test for sending encrypted/decrypted data to a hardware sink is used to experiment with the use case where raw data is sent to a hardware peripheral, such as an Ethernet port, and the cryptographic operations are performed in the hardware prior to arriving at the peripheral. This situation eliminates the case where the CPU has to wait for the return of the data from the hardware. The performance results of each test for the software and hardware

ciphers are recorded and then compared in order to analyze any performance improvements achieved by the hardware ciphers over the software ciphers.

Each of the two tests, file encryption/decryption and hardware sink test, are contained inside a single software application. Furthermore, there are two different software applications, or architectures, that are used for testing the software and hardware ciphers. Each software application can execute both types of tests. One software architecture is designed to maximize the data throughput of the hardware acceleration; whereas, the second software architecture is designed to present a more simplistic programming API to the software developer by extending the well-established OpenSSL EVP API. These two software architectures are used to conduct the tests on the ciphers using different programming models.

There are also two different hardware setups used for implementing the software and hardware ciphers. There is an embedded and workstation hardware setup. The embedded environment is hosted on the Zynq-7000 APSoC on the Zedboard development platform which uses the dual ARM core processors for the software ciphers and the FPGA for the hardware accelerated ciphers. The workstation hardware setup uses an Intel CPU to host the software ciphers and also uses the FPGA fabric of the Zynq-7000 APSoC to host the hardware accelerated ciphers similar to the embedded hardware setup. The workstation hardware setup uses the ZC706 development board, which contains a Zynq-7000 APSoC, to interface to the Intel CPU for the hardware acceleration.

CHAPTER 5

SYSTEM DESIGNS

There are two main system designs for this research. There is an embedded design and a workstation design. Both designs utilize the Zynq-7000 APSoC heterogeneous platform for implementing the hardware accelerated cipher modes on the FPGA fabric, but each system design consists of a different processor host for the software. The embedded design is implemented on the Zedboard development platform so it uses the dual ARM core processor to host the software portion of the system and communicates with the hardware accelerator via the AXI4 bus. The workstation design has the hardware accelerated ciphers implemented on the ZC706 development board and an Intel Workstation platform hosts the software portion of the system and communicates with the hardware accelerator via the PCIe bus.

The architecture of the system is similar between the two system designs with the exception of the bus architecture and host CPU. The embedded design utilizes the AXI4 bus and an ARM CPU; whereas, the workstation design utilizes the PCIe bus and an Intel CPU. The overall data flow and control of the systems are the same between the two designs. The main components of the system designs are the host processor, the Xillybus interface solution, the cipher mode cores, and the AES block cipher IP cores. The host processors and the Xillybus software and hardware cores are different between the embedded and workstation designs, but the cipher mode cores and AES IP cores are the exact same between the two designs. The software that executes on the host processor for each design is the same as well

with the exception of the memory mapped interface for providing the cipher key and initialization vector to the FPGA. The one feature of the designs that allows for this compatibility in functionality across the two design environments is the use of the Xillybus interface solution. Xillybus provides a full software and hardware solution for interfacing across the AXI4 and PCIe buses and provides the same software and hardware interfaces to the developer regardless of the bus type. Therefore, the same system architecture can be applied around the Xillybus solution transparent to the target platform.

There are two different AES block cipher IP cores used in the designs. There is a non-pipelined and a fully pipelined core. The non-pipelined core is used for the ECB and CBC cipher modes. Two instantiations of the non-pipelined core are present in each of the ECB and CBC cipher mode cores. One instantiation is used solely for the encryption process and the other is used solely for the decryption process. This allows for the simultaneous use of both the encryption and decryption process for these cipher modes. The fully pipelined core is used for the CTR cipher mode. Only one instantiation of the AES core is present for the CTR mode because the pipelined core consumes a large amount of resources so there was not enough resources in the FPGA fabric on the Zedboard. Therefore, the encryption and decryption processes can only be executed one at a time for the CTR cipher.

The following two sections, Section 3 and Section 5.2, will discuss both the embedded and workstation system designs in more detail. Each section will discuss details of each of the main components of the system and the roles each played in the overall design. The section discussing the workstation design will only elaborate on the parts of the system that is different from the embedded system.

5.1 Embedded System Design

The embedded system design is implemented on the Zedboard development platform and uses the onboard Zynq-7000 APSoC for hosting the software and hardware portions of the system design. The top level functional block diagram for the embedded system is shown in Figure 5.1. It shows how the dual ARM core processor is connected to the FPGA fabric via the AXI4 bus. The design uses the hardened DMA engine for transferring data from the CPU to the FPGA which uses a 32-bit General Purpose (GP) port for the data transfer. For transferring data from the FPGA to the CPU, the Xillybus IP core implemented in the FPGA design uses an AXI DMA engine. This DMA engine interfaces to the HPS of the Zynq-7000 via a 64-bit Accelerated Coherency Port (ACP) which allows the FPGA to transfer data directly into the DRAM memory without having to go through the CPU. The Xillybus IP core interfaces to each of the three cipher mode cores and the cipher mode cores interface to an AES block cipher IP core. The ECB and CBC cipher mode cores interface to the non-pipelined AES core and the CTR cipher mode core interfaces to the pipelined AES core.

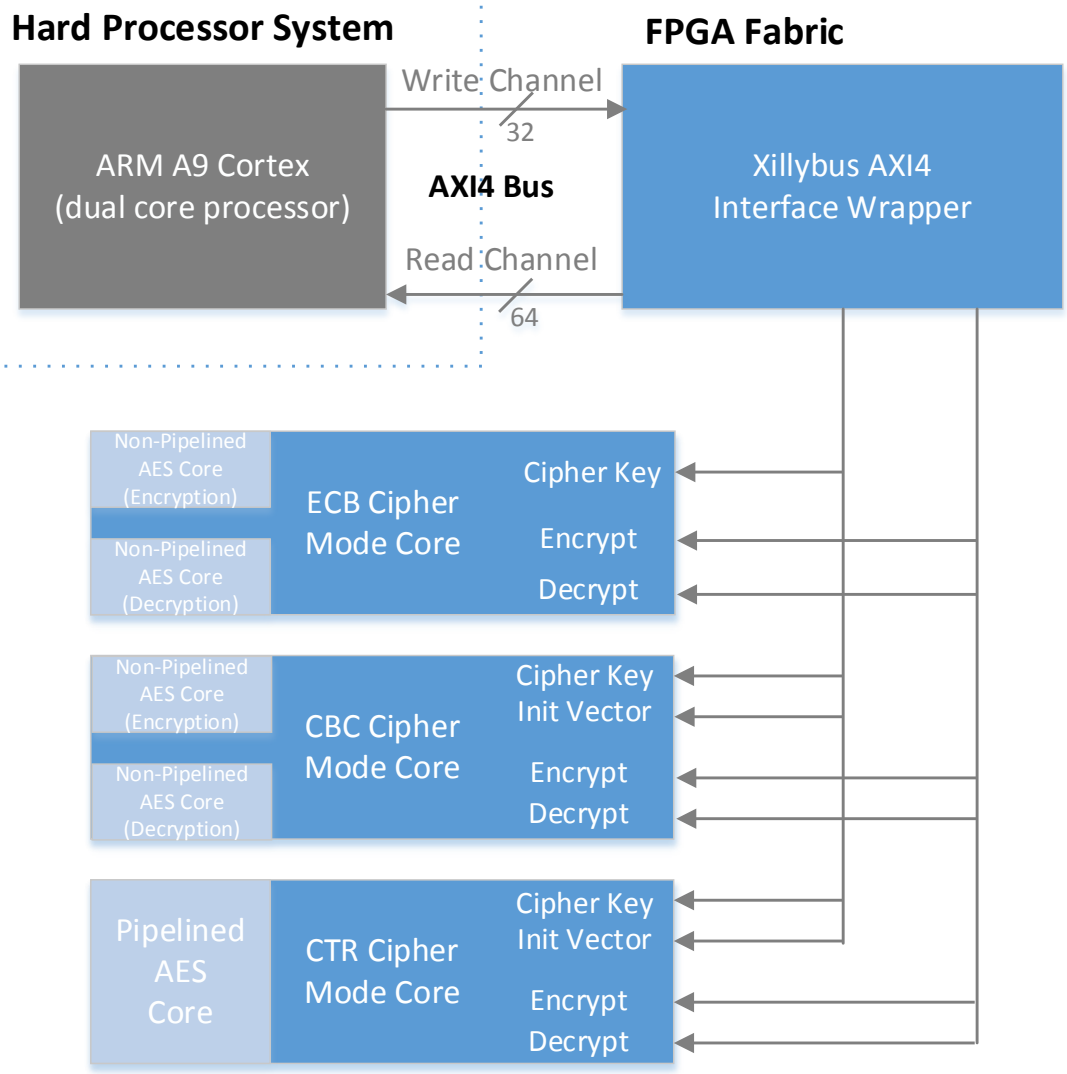


Figure 5.1 Top Level System View of Zedboard Embedded Design

5.1.1 ARM Cores

The hardened processor system (HPS) of the Zynq-7000 APSoC consists of a dual ARM Cortex A9 Application Processor Unit (APU). It has a computation throughput of 2.5 Dhrystone Million Instructions per Second (DMIPS)/MHz per CPU. Each CPU can be clocked at a maximum clock frequency of 1 GHz. Therefore, at 2.5 DMIPS/MHz, if each CPU was clocked at 1 GHz then each CPU would ideally

achieve 2.5*1000 = 2,500 DMIPS. This produces a combined throughput of 5000 DMIPS across both cores. For the design implemented for this research, each core was provided a clock at a rate of 666 MHz. The Zynq also maintains memory coherency across both processors. The Cortex A9 cores implement the ARMv7-A hardware architecture which contains TrustZone security features and the Thumb-2 instruction set architecture (ISA). The cores also contain the NEON vector processing unit for high throughput vector processing. It contains both single and double precision Vector Floating Point Unit (VFPU). It has three watchdog timers, one global timer, and two triple-timer counters. Each CPU core contains a separate instruction (L1I) and data (L1D) caches that are 32 KB each with a 4-way set-associativity. Both CPU cores share a level 2 unified cache (L2U) that is 512 KB with an 8-way set-associativity [6].

The dual ARM core processors are the supporting hardware for executing the software test applications. The ARM cores are booted with a version of the Ubuntu 12.04 kernel known as Xillinux. Xillinux is a Xillybus produced kernel and will be discussed in more detail in Section 7.2. The Xillinux OS is used for developing and executing the test software necessary for testing the software and hardware ciphers of the system design. The OS is also where the OpenSSL cryptographic library is installed and accessed for the software ciphers. The OS also has the Xillybus kernel driver software installed for access while interfacing to the hardware ciphers.

5.1.2 Xillybus Kernel Driver

Xillybus offers both a software and hardware package for interfacing across the AXI4 bus. The software package, in the form of a kernel module driver, is designed to present the Linux host with a simple and well-known interface. The host

kernel driver generates device files that behave like named pipes. They are opened, read from and written to just like any file, but behave much like pipes between processes or TCP/IP streams. To the program running on the host, the difference is that the other side of the stream is not another process (over the network or on the same computer), but a FIFO in the FPGA. Just like a TCP/IP stream, the Xillybus stream is designed to work well with high-rate data transfers as well as single bytes arriving or sent occasionally [21].

The interface to the Xillybus driver software from any user application is done through device files. The device files are auto generated by the kernel driver when it is loaded into the OS and detects a Xillybus compatible device on the bus. Also at driver load, the DMA buffers for bus communication are allocated in DRAM and their location(s) are provided to the Xillybus IP core for all the Xillybus streams present. During application execution, a handshake protocol between the FPGA and host makes an illusion of a continuous data stream. However, behind the scenes, DMA buffers are filled, handed over, and acknowledged in both directions, thus hiding the latency experienced by the transfer of data across the bus [21].

For the embedded design on the Zedboard, there are seven Xillybus interfaces present. One of the interfaces is a Xillybus Lite interface, which is the memory-mapped interface, for providing the cipher key and initialization vector to the hardware ciphers after being generated in software. The memory-mapped transactions are handled by the CPU and not the DMA engine. The other six interfaces are stream interfaces for sending and receiving data to and from the hardware ciphers. There is a separate interface for the encryption process and an interface for the decryption process for each cipher. Thus, each cipher has two separate full-duplex interfaces. The AXI4 bus is a full duplex bus so data can be

transferred in both directions without a decrease in bandwidth. Since there are separate interfaces for each of the ciphers and separate interfaces for the encryption and decryption process, multiple operations can be executed simultaneously. However, by performing multiple operations on the hardware accelerated ciphers the overall throughput of each operation will be diminished since the bus is shared for all data streams. Each of the stream interfaces are configured to achieve maximum bandwidth on the AXI4 bus and in order to do so Xillybus reserves DMA buffers in the DRAM. For each of the six stream interfaces in the embedded design there are 32 x 128 KB, or 4 MB, buffers reserved for the DMA buffers.

The host processor sends data to the FPGA through the use of DMA transactions. A DMA transaction is initiated by the processor by providing the DMA engine with the DRAM memory location for the beginning of the data set to be transferred across the bus. The DMA engine then initiates a DMA transaction on the AXI4 bus and transfers the entire data set to the FPGA, or at least a data block for which there is room in the FPGA FIFOs. The data is transferred from the host to the FPGA via the 32-bit general purpose (GP) AXI ports where the DMA engine is the master and the FPGA is the slave. If there is not enough room in the FPGA FIFOs to receive all the data from the processor then the DMA only transfers what it can and waits until the next transaction to send the remaining data.

5.1.3 AXI4 Bus

The Advanced eXtensible Interface (AXI) protocol is part of the ARM Advanced Microcontroller Bus Architecture (AMBA) which is proprietary to ARM. The AXI protocol is designed and used for inter device communication via a bus interface for high throughput communication. The first version of AXI was released

in 2003 and the second version (AXI4) was released in 2010. The Zynq-7000 APSoC uses the AXI4 architecture. There are three types of AXI4 interface; AXI4, AXI4-Lite, and AXI4-Stream. The base AXI4 interface is used for high-performance memory-mapped interfaces, the AXI4-Lite interface is for simple, low-throughput memory-mapped interfaces, and the AXI4-Stream interface is for high-speed data streaming [22].

The AXI bus protocol allows for any number of devices to reside on the bus. Furthermore, for any transaction to occur on the bus, there has to be a master that initiates the transaction and a slave that is the recipient of the transaction from the master node. Both the AXI4 and AXI4-Lite interfaces consist of five different signal channels: read address channel, write address channel, read data channel, write data channel, and write response channel. Data can be transferred in both directions on the bus between the master and slave devices simultaneously. The separate address and data lines for both the read and write channels is what allows for this simultaneous bidirectional communication [22].

The bus transactions for memory-mapped interfaces, AXI4 and AXI4-Lite, involve the idea of a destination address within a system memory space; therefore, each transaction that occurs on the bus includes an address phase and a data phase. The address phase specifies that target address for the data and the data phase provides the data for the target address. The AXI4-Lite interface only allows a single data transfer per transaction which makes it ideal for register access type interfaces. The AXI4 interface allows a burst transaction of up to 256 data transfers per transaction. The AXI-Stream interface is different from the memory-mapped interfaces in that it does not require an address phase in a transaction and the AXI4-Stream interface is unidirectional. The write channel of the AXI4-Stream

interface is modeled after the AXI4 interface, except there is no address channel and the Stream interface can burst an unlimited amount of data in any transaction [22].

The embedded design implemented on the Zedboard utilizes both the AXI4-Lite and AXI4-Stream interfaces. The AXI4-Lite interface is used for providing the hardware accelerator cores with the cipher key and initialization vector prior to executing the cipher modes. The AXI4-Lite interface is used for this functionality because the basic register level access needed for providing this information to the hardware did not call for a high throughput interface, so the simplistic Lite interface is adequate. The AXI4-Stream interface is used for streaming the raw data to and from the hardware for performing the cipher operations on the data. The AXI4-Stream interfaces provided the ability to burst an unlimited amount of data in any given transaction on the bus which allowed the system to achieve maximum throughput for the bus. It is also worth noting that the AXI4 bus has a maximum clock frequency of 125 MHz and the design implemented for this research uses a clock frequency of 100 MHz.

5.1.4 Xillybus IP Core

The Xillybus IP core is the full solution for interfacing with the host ARM processor via the AXI4 bus from the FPGA fabric of the Zynq-7000 APSoC. The overall Xillybus FPGA solution includes multiple sources and core modules all connected together, but for simplicity it will all be referred to as a single block box module which is how the solution is used in the design. The purpose of Xillybus is to provide the developer with a well-known and easy-to-use interface for communicating with the ARM processors via the AXI4 bus.

Xillybus contains three main components: the AXI4 slave interface, the AXI4 master interface, and the interface to the backend user application. The AXI4 slave interface is used for receiving data from the host ARM processors through the memory-mapped or data stream interfaces. The data is received on this interface through the 32-bit GP ports. The memory-mapped transactions across the slave interface are basically just passed straight through to the user application with all the necessary signals for the user design to react to the transactions, but the Xillybus core still handles all the low level AXI4 handshaking. The data stream interface(s) from the CPU to the slave interface is handled much differently than the memory-mapped interface. Each Xillybus interface present in the design has its own FIFO of at least 2 KB deep in each direction to help hide the latencies of the bus transactions. These FIFOs are used to receive a stream of data from the processor through DMA transactions. The FIFOs help buffer data as it is received from the bus in the burst transactions just in case the backend user logic is not consuming the data at the same rate as the data transfer. However, eventually, if the user logic is consuming data too slow and the Xillybus FIFOs become full, then the DMA engine on the HPS will stall due to backpressure from the FPGA slave interface. The master AXI4 interface of the Xillybus core is naturally just the reverse of the slave interface. Xillybus implements an AXI DMA engine inside the core for interfacing to the 64-bit Accelerated Coherency Port (ACP) of the HPS. The ACP provides direct access to the DRAM. Therefore, the AXI DMA engine can transfer data from the FPGA back to the DRAM (and host processor) through the ACP interface. Thus, all the user logic must do when transferring data to the host is move data into the outgoing data FIFO provided by Xillybus and the AXI DMA engine will transfer the data into the DMA buffers present in the DRAM. As eluded to, the interface signals

that are provided to the backend user application logic are the memory-mapped interface signals and standard FIFO interface signals for the data stream interfaces for communicating in both directions. The FIFO interfaces are standard FIFOs without the first-word-fall-through (FWFT) feature. The entire Xillybus module is clocked off of the 100 MHz AXI4 bus clock. This makes the entire core synchronous to the AXI4 bus [23].

The Xillybus IP core is also the place in the design where the decision is made whether to send the encrypted/decrypted data back to the host processor or to perform the sink operation on the data. The software application writes to a register telling the hardware which operation to perform during the execution of the current test and the top level of the Xillybus IP core is modified to act based on the contents of that register. It will either send the output data from the cipher operations back to the processor via the AXI DMA engine or it will allow the data to die upon arrival from the cipher cores.

5.1.5 Cipher Mode Cores

The cipher mode cores are designed to implement the specific functionality of each cipher mode. The cores have an instantiation of one of the AES IP cores which is used as the block cipher for the cipher modes. One of the main functionalities of the core is to control the data flow of the hardware accelerated ciphers by interfacing to the Xillybus IP core and the AES cores. The cipher cores are used as control logic for handling the interaction and handshaking with the AES block cipher cores while passing data through it. As seen in Figure 5.1, there is a cipher mode core for each of the three cipher modes implemented in the design. Therefore, there is an independent cipher mode core for the ECB, CBC, and CTR ciphers. The input and

output (I/O) signals for the top level interface for the cipher mode cores are shown in Figure 5.2 and Figure 5.3. The clock signal sent into the cipher mode cores is the 100 MHz bus clock that is driving the AXI4 bus. The only externally exposed signals from the cores include the input signals for the cipher key and initialization vector, if needed, and the signals needed to control the data flow in and out of the Xillybus IP core. The plaintext and ciphertext data stream interface signals are used to interface directly to the Xillybus IP core FIFOs for sending and receiving data to and from the host processor. All the implementation specific logic for the cipher modes are contained internal to the cores. This allowed for the easy integration with the Xillybus IP core by just connecting the necessary signals to the Xillybus FIFOs and the cipher key and init vector input signals. This results in the completion of the full data flow path from the processor to the cipher mode cores, to the AES block cipher cores, and then back.

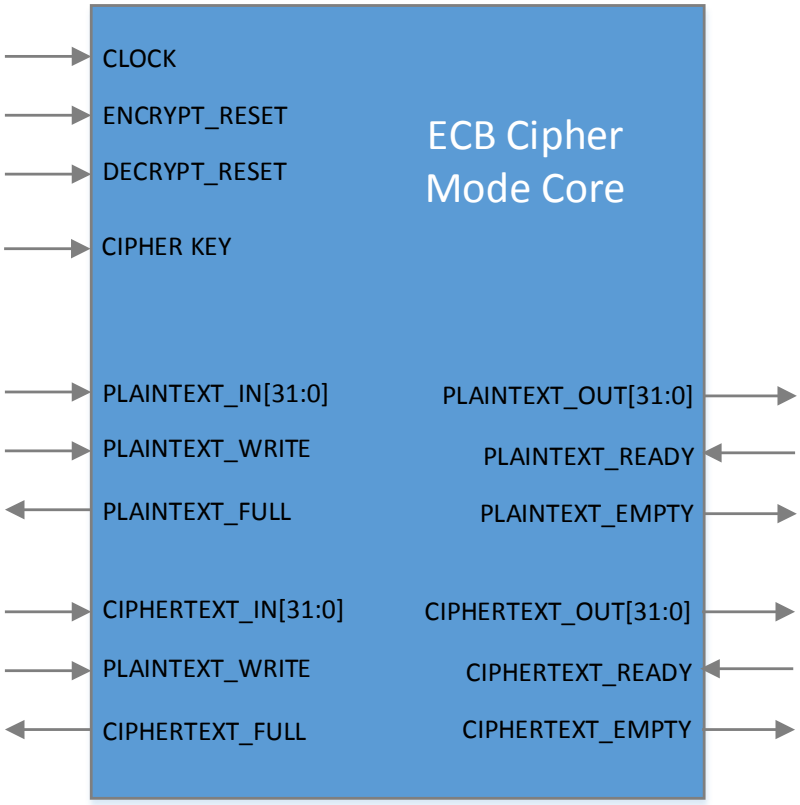


Figure 5.2 I/O Signals for ECB Cipher Mode IP Core

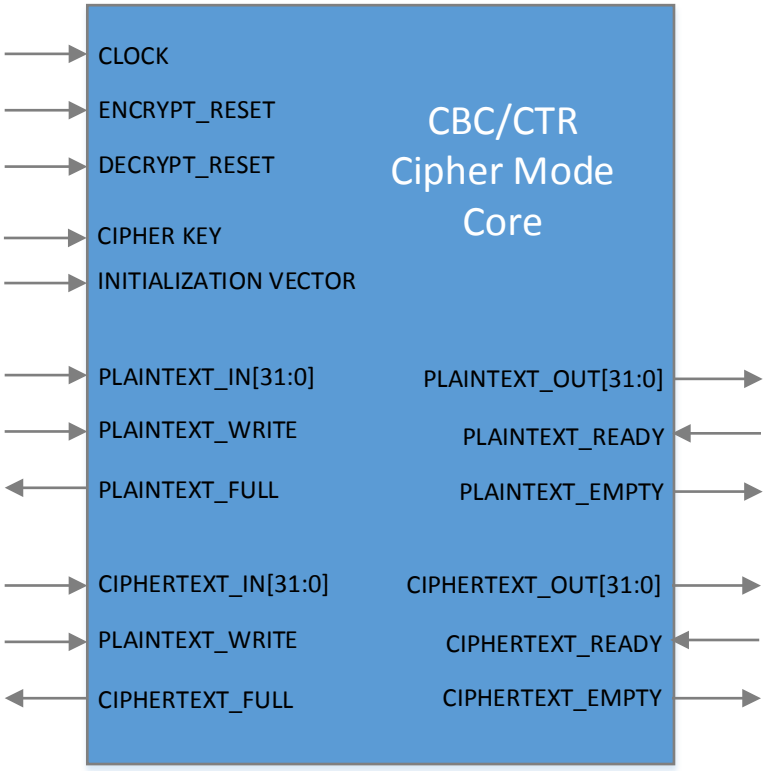


Figure 5.3 I/O Signals for CBC and CTR Cipher Mode IP Cores

The cipher cores implement a finite state machine for both the encryption and decryption processes. The state machines step through the same basic flow of operations for all three ciphers. These operations include receiving data from the processor, sending the data through the AES core, performing specific cipher operations on the data, and then sending the output data back to the processor. The main differences between each of the cipher mode cores is the specific logic needed to implement the functionality of the cipher and the AES core used. The ECB and CBC ciphers interface to the non-pipelined AES core, but the CTR cipher core interfaces to the pipelined AES core.

The first major operation of receiving the data from the processor is the same for all three cipher modes. Both AES cores require an entire 128 bit state matrix as the data input into the core. The maximum data width of the Xillybus IP core for receiving data from the processor is 32 bits since it uses the general-purpose ports on the AXI bus. This introduces the need to buffer multiple 32 bit words until an entire state matrix is received. Thus, the first responsibility of the cipher mode cores is to perform such a task; to concatenate four 32-bit words that are received from the HPS in order to create a single 128-bit word to pass into the AES core for encryption or decryption. The cipher core can continuously receive data from the Xillybus core as long as data does not back up further downstream (i.e. the AES core or the return path of Xillybus) and stall the flow of data.

The second operation of sending the data through the AES core is different for each of the cipher modes. This is because each cipher mode requires different operations to be performed on the data specific to the cipher algorithm before executing the encryption or decryption operation on the data. The ECB cipher is able to just send the input state matrix directly into the AES core because it just encrypts each state matrix separately. The CBC cipher XORs either the initialization vector or the previously generated ciphertext to the plaintext prior to encrypting the data. For decryption, CBC decrypts the ciphertext and then XORs either the initialization vector or previous ciphertext to the decryption output. The CTR cipher core encrypts a concatenated initialization vector and incrementing counter value which is then XORed with either the plaintext (for encryption) or ciphertext (for decryption). The CTR cipher core buffers up multiple encrypted values for the initialization vector and counter value so that the incoming plaintext or ciphertext can immediately be XORed and sent back to the processor. This hides the latency present in the

pipelined AES core used for the CTR cipher because the incoming data does not have to wait for the data to progress through the pipeline of the AES core.

The CTR cipher uses the pipelined AES core and the other two ciphers use the non-pipelined core. The ECB and CBC cipher cores cannot send a new state matrix into the AES core until it receives the output for the previous state matrix. This causes the data flow to stall while waiting for the AES core to complete its operation. This can create backpressure on the Xillybus FIFOs once all the hardware FIFOs fill up, which could ultimately result in the stalling of the DMA controller when trying to send new data to the FPGA. The AES cores are the bottleneck for the ECB and CBC cipher modes. On the other hand, for the CTR cipher, the AXI4 bus becomes the bottleneck since the CTR cipher uses the fully pipelined AES core.

The final operation in the state machine of sending the output data to the processor is the same for each cipher mode as it was for receiving data from the processor. Each output state matrix is a 128-bit word which has to be broken into four 32-bit words and written into the Xillybus FIFOs for transmission back to the processor. Figure 5.4 and Figure 5.5 show the control and data flow paths of the ECB/CBC and CTR cipher mode cores, respectively.

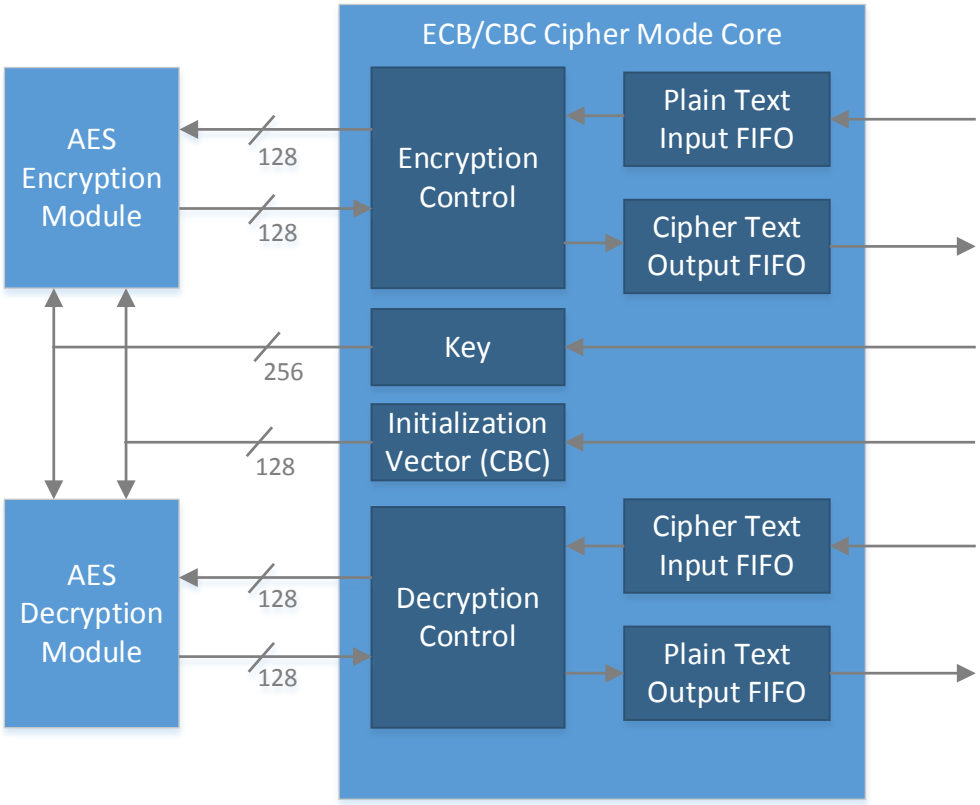


Figure 5.4 Control and Data Flow of the ECB/CBC Cipher Mode Cores

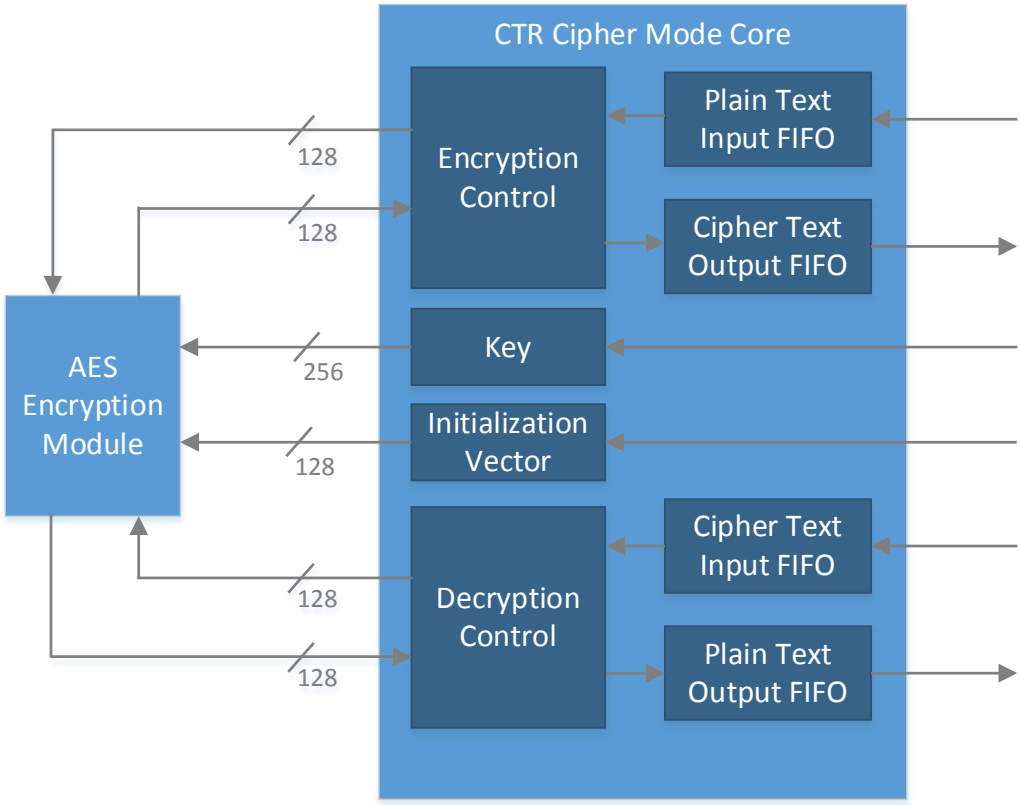


Figure 5.5 Control and Data Flow of the CTR Cipher Mode Core

5.1.6 AES IP Cores

The AES block cipher IP cores implement the AES algorithm using a hardware description language. There are two different AES IP cores used in the hardware design; a pipelined and non-pipelined core. The non-pipelined AES core is used for the ECB and CBC ciphers and the pipelined core is used for the CTR cipher. The AES IP cores are third party IP cores and are used as black boxes in the design. However, the correctness of the output of the cores are verified with known AES test vectors. Section 5.1.6.1 will discuss the non-pipelined core in more detail and Section 5.1.6.2 will discuss the pipelined core in detail.

5.1.6.1 Non-Pipelined Core

The non-pipelined AES IP core is an open-source Verilog core that is downloaded from a GitHub repository called secworks/aes [24]. The core can only operate on a single block of data at any given time. This means that the core must complete the current operation on the current state matrix before accepting the next state matrix. One feature of the non-pipelined core is that it implements both the encryption and decryption algorithms for the AES block cipher. This makes the core a good solution for both the ECB and CBC ciphers since both ciphers require both the AES encryption and decryption algorithms for encrypting and decrypting data, respectively. The ECB cipher mode has the ability to be implemented in a pipelined architecture, but the pipelined AES core does not implement the decryption algorithm for AES so the non-pipelined core is used.

The non-pipelined core has both the 128-bit and 256-bit key versions of AES implemented. For this design, the 256-bit key implementation of AES is used as the block cipher for the ECB and CBC ciphers.

Figure 5.6 shows all the input/output (I/O) signals for the non-pipelined AES IP core. The CLOCK signal is the signal used by the core for clocking the logic. The clock frequency has a documented maximum clock frequency of 100 MHz for the Xilinx Spartan 6 FPGA. Therefore, if another FPGA model is used other than the Spartan 6 then the maximum clock frequency would vary. For this design, the Zynq-7000 contains an Artix 7 FPGA so the core is provided a clock frequency of 100 MHz. This is the documented maximum clock frequency for the core, but since the Artix 7 is a much newer FPGA model than the Spartan 6 the actual maximum clock frequency is 125 MHz. The RESET signal just forces the core back to an initialized

state. The KEY_LENGTH signal tells the core which length of key, either 128 or 256, based on the value of the signal. This signal is hardcoded to a value of "1" to program to core for the 256-bit key implementation. The KEY signal is the input signal for passing in the cipher key into the core. The MODE signal is used to tell the core which AES operation to perform whether encryption or decryption. A value of "0" is used for decryption and a value of "1" for encryption. The INIT signal is used to instruct the core to latch the values that are present on the KEY_LENGTH and MODE input signals and configure the core for the configuration specified by those input signals. The UNIT_READY output signal signifies when the core is ready to accept new data and perform the requested operation. The STATE input vector is how the input state matrix is passed into the core. The START input signal is used to instruct the core to use the vector present on the state input lines and perform the desired AES operation. Lastly, when the core has completed the operation on the input data, it signals that the output data is ready via the RESULT_READY signal and provides the output state matrix on the RESULT output lines [24].

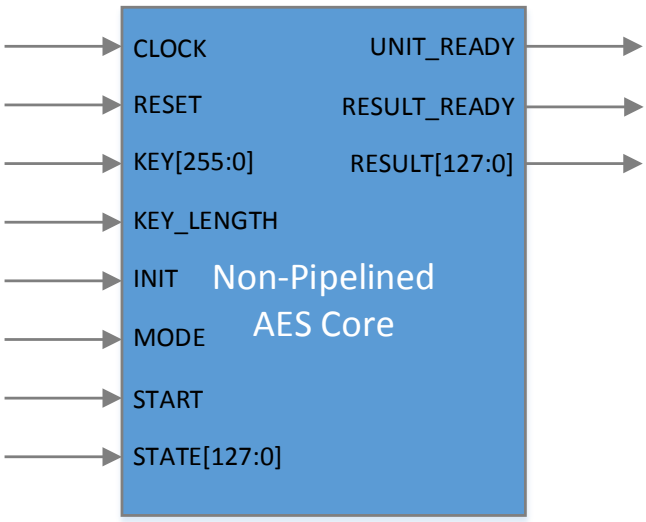


Figure 5.6 I/O Signals for Non-Pipelined AES IP Core

5.1.6.2 Pipelined Core

The pipelined AES IP core is an open-source Verilog core that is downloaded from OpenCores.com [25]. The core can operate on multiple state matrices at any given time; therefore, increasing its overall throughput through the core. A drawback to this core is that it only implements the AES encryption algorithm which rules out its use for ciphers that require both encryption and decryption algorithms such as ECB and CBC. However, the CTR cipher mode only requires the encryption algorithm to both encrypt and decrypt data. This fact, along with the fact that the CTR cipher mode does not have any data dependencies between any two blocks of data makes the pipelined AES IP core the ideal option for implementing the CTR cipher mode.

The pipelined core has an implementation for the 128-bit, 192-bit, and 256-bit key versions of AES, but for this design only the 256-bit key version is used. The core consists of two pipelines. The first pipeline manipulates the 16-byte state matrix and the second pipeline computes the 16-byte expanded key used in each round. The pipelined architecture of the core allows the core to accept a new state matrix every clock cycle. There are 29 pipeline stages implemented in the core so the ciphertext for a corresponding input state matrix is available 29 clock cycles after being input into the core. The core, however, does not have any control signals for handshaking with the core for input or output data. In order to control the flow of data into and out of the core, a wrapper has been developed around the core synchronizing the time at which new data is sent to the core so it knows the time that its corresponding output data is ready to be received from the core.

Figure 5.7 shows all the I/O signals available for the pipelined AES core. The CLOCK signal is the signal used by the core for clocking the logic. The maximum documented clock frequency the core can accept is 324.6 MHz for a Xilinx Virtex 6 FPGA. However, if the core is implemented within a different FPGA then the maximum clock frequency may vary slightly. For this design, an Artix 7 FPGA is used and the core is provided a clock with a frequency of 100 MHz. This clock frequency is used because the AXI bus is limited to the 100 MHz clock frequency and with the pipelined architecture of the core the limiting factor for performance of the core is the AXI bus. Therefore, there is no need to provide the core with a clock faster than the 100 MHz provided to the AXI bus. The RESET signal just forces the core back to an initialized state. The KEY signal is where the core is provided the 256-bit key for the AES block cipher. The STATE signal is where the 16-byte state matrix is passed into the core. The signals present at the key and state input lines of the core are clocked into the pipeline at every rising clock edge. Furthermore, the RESULT output signal is driven with a new output state matrix every clock cycle. The FPGA resources that are consumed by the core is approximately 6,800 slice registers, 6500 slice LUTs, 500 bonded IOBs, 120 block RAMs, and 1 BUFG. The expected throughput of the core if clocked with a 300 MHz clock frequency is 38.4 Gbps. Since this design clocks the core with a 100 MHz clock frequency then the nominal throughput of the core becomes 38.4/3 = 12.8 Gbps [25].

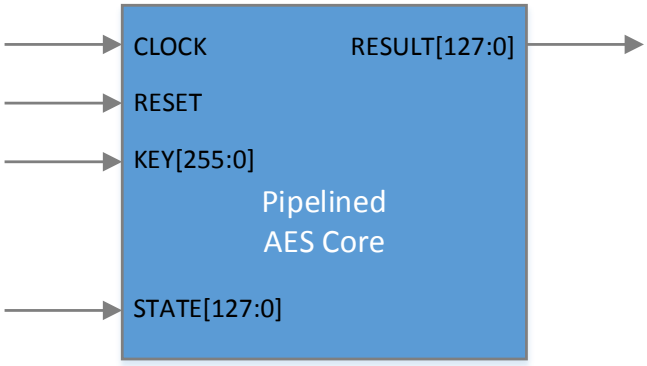


Figure 5.7 I/O Signals for Pipelined AES IP Core

5.2 Workstation System Design

The workstation system design is implemented on the combination of a ZC706 development platform and an Intel host processor. The ZC706 is used for hosting the hardware design of the ciphers and the Intel processor is used for hosting the software portions of the system design. The top level functional block diagram for the workstation system is shown in Figure 5.8. It shows how similar the workstation design is to the embedded design shown in Figure 5.1 with the exception of just a couple differences. The workstation design uses the Intel processor of the workstation to interface to the FPGA hardware accelerator via the PCIe x4 interface. The host processor and Xillybus IP core use a DMA engine for transferring data over the PCIe bus. The following subsections regarding the workstation design will only focus on the implementation details that are different from the embedded design which is discussed in Sections 0 to 5.1.6.

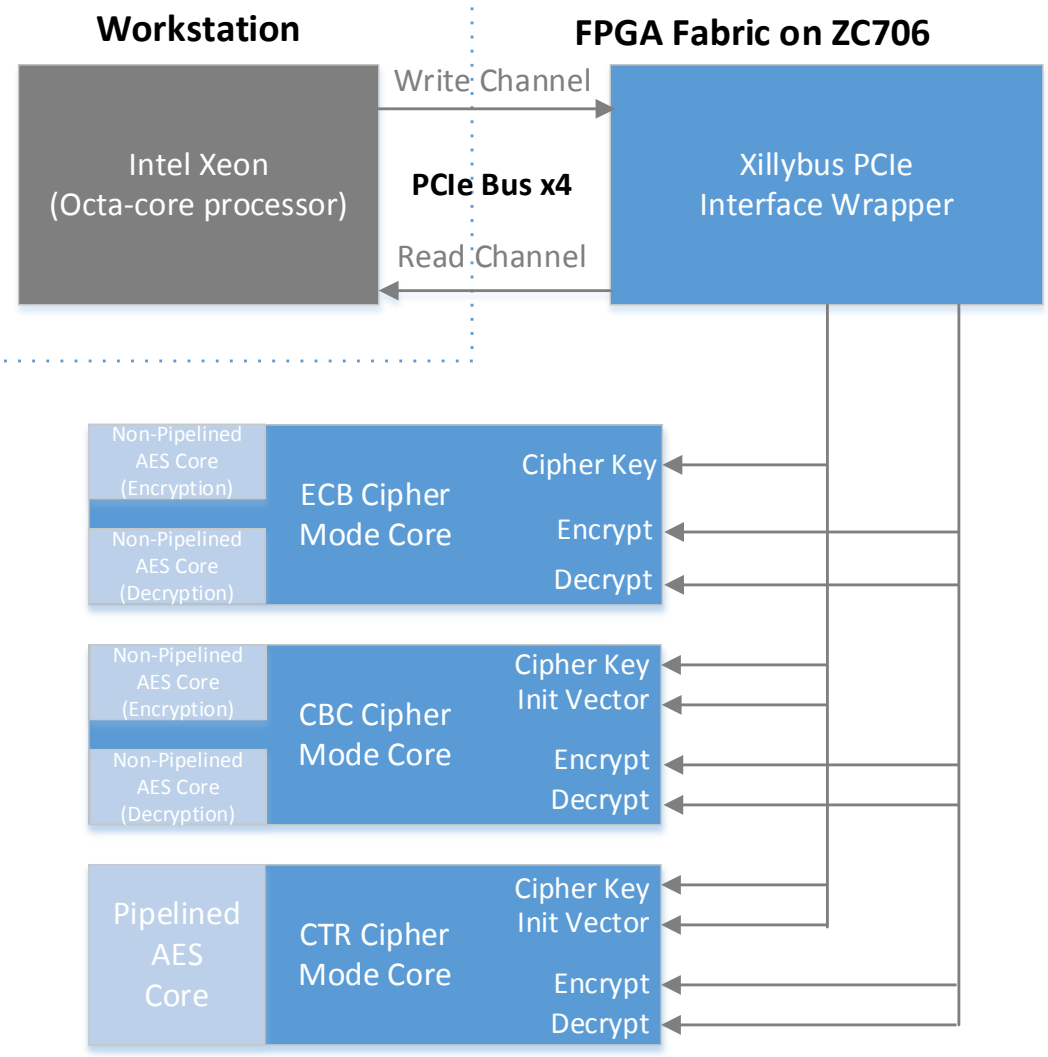


Figure 5.8 Top Level System View of Workstation Design

5.2.1 Intel Processor

The host processor for the workstation design is the Intel Xeon processor E5-4655 V4. This Intel processor contains eight cores with two threads per core giving a total of sixteen threads for the whole processor. The base frequency for the processor is 2.5 GHz with a max turbo frequency of 3.2 GHz. It contains 30 MB of cache at a bus speed of 9.6 GT/s over Intel's Quickpath Interconnect (QPI). The processor can handle a memory size of up to 1.5 TB of DDR4 memory. It can have up to four

memory channels and achieve a maximum memory bandwidth of 68 GB/s. The processor also has support for x4, x8, and x16 PCIe configurations with a maximum of 40 PCIe lanes [26].

5.2.2 Xillybus Kernel Driver

The Xillybus kernel driver module for the workstation design is very similar to that of the embedded design. The software interface provided to the software developer for accessing the kernel driver is identical to that of the kernel driver for the embedded design. The kernel driver is still accessed via device files on Linux with the same calls to open, read, write, and close. The interface looks and behaves the exact same as the kernel driver in the embedded design. The workstation design contains the same seven Xillybus interfaces. It contains one memory-mapped interface and six streaming interfaces. The only difference in the kernel driver module for the workstation design compared to the embedded design is the software calls that are made inside of the kernel module. The workstation kernel driver is interfacing to the PCIe bus and the DMA engine that is present on the workstation platform. This only difference is transparent to the software developer interfacing to the kernel module in the workstation design because all the specific functionality for interfacing to the bus is contained inside the kernel module. The DMA buffers allocated in the host RAM is 64 x 128 KB, or 8 MB, for each of the six stream interfaces present. The specifics of the Xillybus kernel module driver software and interface is explained in Section 5.1.2. The majority of the details discussed in that section are common in the Xillybus kernel module between the workstation and embedded designs except for the previously mentioned differences.

5.2.3 PCIe Bus

The Peripheral Component Interconnect Express (PCIe) bus actually does not function like a bus, but more like a network. It is still referred to as a bus in many instances because multiple devices can reside on the PCIe interconnect which gives it that bus feel. Instead of multiple devices communicating on the same data lines, the PCIe bus provides dedicated connections to all the peripherals and the data lines all run into a switch that controls the routing of data to the correct destination. This allows for much higher bandwidths than the legacy PCI bus because peripherals do not share the bandwidth of a single bus. At boot-up, a computer enumerates all the devices present on the PCIe bus so the switch knows where how to route the traffic. Every PCIe link contains four pairs of wires; one pair for transmit and the other for receive. Each set of transmit and receive lines construct a link. A single transmit and receive pair is an x1 link; whereas, an x2 link will have two transmit lines and two receive lines resulting in eight total wires. PCIe allows for up to x32 lanes meaning there are 32 transmit and receive pairs [27].

5.2.4 Xillybus IP Core

The Xillybus IP core for the workstation design is also almost identical to that implemented in the embedded design. Section 5.1.4 discusses the details of the Xillybus IP core for the embedded design where the majority of those details are still applicable to the Xillybus core for the workstation design. The main difference between the embedded and workstation cores is that the workstation Xillybus core interfaces to the PCIe bus instead of the AXI4 bus. The Xillybus core uses the Xilinx PCIe IP core available for the Kintex 7 FPGA on the ZC706 development board. The PCIe core is used to interface directly to the PCIe bus for communicating with the

host PC. The only other difference with the workstation Xillybus core is the clock rate that drives the core. This core is driven with a 250 MHz clock as opposed to the 100 MHz clock in the embedded design. All the interfaces for communicating with the Xillybus core via the memory-mapped interface and the data stream FIFOs are exactly the same between the workstation and the embedded Xillybus cores. This allows for the rest of the FPGA VHDL code to interface to the Xillybus cores in both designs the same way without having to modify any code.

5.2.5 Cipher Mode Cores

The cipher mode cores implemented in the workstation design are identical to those implemented in the embedded design with just one exception. The only difference in the cipher mode cores between both designs is the clock provided to the cores for clocking the logic. The clock provided to all the cipher cores in the embedded design is the raw 100 MHz AXI4 bus clock, but the clock provided to the cipher cores in the workstation design differs between the cipher cores. The ECB and CBC cipher cores use the non-pipelined AES block cipher core which has a maximum clock rate of 100 MHz which means these cipher cores could not use the raw bus clock from Xillybus which is 250 MHz. Therefore, the design divides the 250 MHz Xillybus bus clock down to 100 MHz and provides the divided clock to the ECB and CBC cipher cores. On the other hand, the CTR cipher core uses the pipelined AES core so it is able to use the 250 MHz clock from Xillybus. This increases the theoretical throughput of the CTR cipher, but maintains the same throughput for the ECB and CBC ciphers since they have the same clock frequency. The implemented functionality of the cipher cores are identical between the designs. The functionality of the cipher cores are discussed in detail in Section 5.1.5.

5.2.6 AES IP Cores

The AES block cipher IP cores used in the workstation design are the same as the ones used in the embedded design. The same non-pipelined and pipelined cores are used. The specific details of each core is discussed in Section 5.1.6.1 and Section 5.1.6.2 for the non-pipelined and pipelined core, respectively. The non-pipelined core is used for the ECB and CBC ciphers and the pipelined core is used for the CTR cipher. The only minor difference between the way the AES cores are used in the embedded and workstation designs is the workstation design provides a 250 MHz clock to the pipelined AES core and a 100 MHz clock to the non-pipelined AES core instead of the same 100 MHz clock provided to both AES cores as in the embedded design.

CHAPTER 6

SOFTWARE ARCHITECTURE

There are two different software architectures that are developed to experiment with the performance of the hardware accelerated ciphers compared to the OpenSSL software ciphers. The first software architecture is designed to maximize the overall data throughput of the hardware accelerator by maximizing the throughput of the Xillybus interface. The second software architecture utilizes an extension of the OpenSSL software where the top level application only interfaces to the OpenSSL library for utilizing both software and hardware cryptographic functionality.

6.1 Maximum Throughput

The first software architecture maintains the goal of maximizing the data throughput of the hardware accelerated ciphers through the use of programming techniques for maximizing the throughput of the Xillybus interface. As a result of the Xillybus interface using DMA transactions to move data to and from the FPGA and CPU a multithreaded software technique can be used to send data to the FPGA at the same time it is receiving output data from the FPGA [21]. This is a direct benefit of using DMA to perform data transfer because this does not require the CPU to be involved in the transfer of data so it can just focus on reading and writing the data to the DRAM memory that is used by the DMA engine for the DMA data transfers. The multithreaded technique used to maximize the data throughput of the Xillybus interface is successful because it is able to hide the latency of both the bus

communication interface with the presence of DMA buffers and the FPGA cryptographic logic since it can produce and consume data to and from the FPGA simultaneously.

This software application contains separate code for accessing the OpenSSL software library and accessing the hardware accelerated ciphers. If the software ciphers are selected at runtime then the application will execute specific code for sending the data through the OpenSSL EVP API for performing the appropriate cipher mode and then writing the output data to the output file or to the hardware sink. On the other hand, if the hardware accelerated ciphers are selected at runtime then the software will spawn two additional threads. One thread is incrementally reading from the input file and sending the data to the FPGA that will perform the cipher algorithm on the data. The second thread is continuously polling for output data from the FPGA and writing the received output data to the output file. The second thread is only used if conducting the file encryption/decryption test. Upon receiving the last output data from the FPGA the main thread joins the two worker threads and completes the application execution. Figure 6.1 shows a flow chart for the software execution for this software architecture. Note that Figure 6.1 specifically shows the software flow for the file encryption and decryption test. The test using the hardware sink will only spawn a single thread for reading the input file and writing the data to the FPGA.

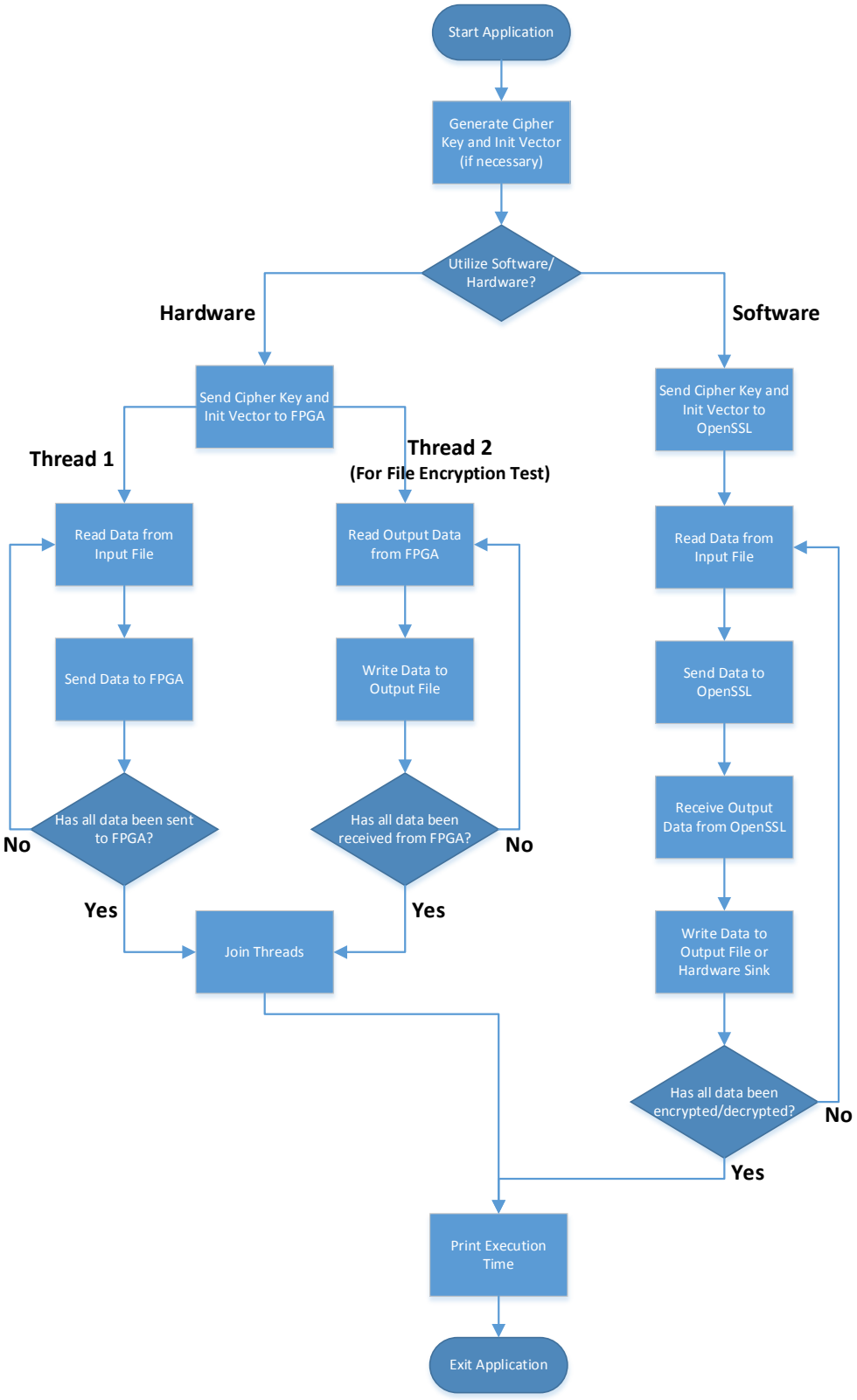


Figure 6.1 Flow Chart for Maximum Throughput Software Architecture

6.2 OpenSSL Extension

The main focus and goal of the second software architecture is to provide a more simplistic API to the software developer by only requiring the developer to interface to the well-known OpenSSL EVP API for accessing both the software and hardware cryptographic functionality. This is achieved by both modifying the OpenSSL library software along with the top level application software to only interface to the OpenSSL library as opposed to having separate code for interfacing to Xillybus. The OpenSSL cryptographic library is modified to be able to utilize the hardware accelerated ciphers at the same software level that the equivalent software algorithms are executed. The EVP API of OpenSSL is slightly modified to include two additional parameters in the initialization function call that specifies whether to execute the software or hardware ciphers and also to specify whether to use the hardware sink in the hardware design or not. The software/hardware selection parameter at initialization of OpenSSL will trigger the initialization of Xillybus if the hardware option is selected. OpenSSL will fully initialize the Xillybus interface and provide the hardware accelerator with the cipher key and initialization vector, if necessary, so that in subsequent calls into OpenSSL with data all that is needed is to pass the data into Xillybus and then wait for the return of the data from the FPGA. OpenSSL will also provide the information to the hardware accelerator whether to use the hardware sink option or not. This software architecture experiences a major decrease in throughput compared to the previous software architecture because OpenSSL forces the interaction with Xillybus and the FPGA to become synchronous, or sequential, as opposed to asynchronous. Figure 6.2 shows a flow chart for the software execution for this software architecture.

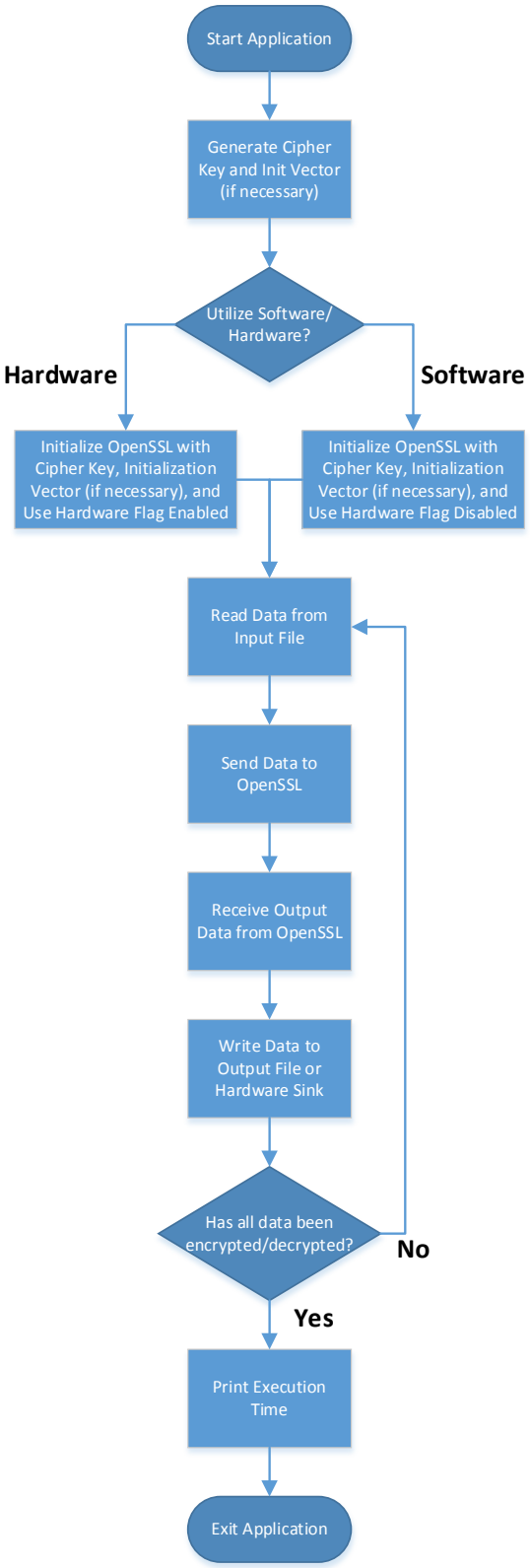


Figure 6.2 Flow Chart for OpenSSL Extension Software Architecture

CHAPTER 7

EXPERIMENTAL ENVIRONMENT

The experimental environment sections include information on the development platforms used for both the embedded and workstation designs along with the approach used for testing the software and hardware ciphers. The embedded environment is discussed first, then the workstation environment, followed by the specifics of the experimental tests and performance metrics.

7.1 Zedboard

The Zedboard is a low cost development board designed and built by Digilent. It includes a Zynq-7000 All-Programmable (AP) SoC and an array of peripherals and standardized connectors including USB, HDMI, VGA, Ethernet, audio connectors, etc. The Zynq chip is the XC7Z020 version which contains an FPGA with the equivalent amount of resources as that of an Artix 7 generation FPGA. The Zynq-7000 has a dual core ARM processor with an adjacent FPGA fabric connected via an AXI4 bus. The board has an additional 512 MB of DDR3 RAM external to the Zynq, 256 Mb Quad-SPI flash memory, SD card interface, onboard USB-JTAG programming, 10/100/1000 Ethernet, USB OTG 2.0 and USB-UART, and multiple displays including HDMI, VGA, and a 128 x 32 OLED [28]. The Zedboard can be programmed using the Xilinx software tool suites of Vivado and SDK. The target applications that the Zedboard is suited for is video processing, motor control, software acceleration, Linux/Android/RTOS development, embedded ARM

processing, and Zynq-7000 APSoC prototyping. Figure 7.1 shows a photo of the Zedboard.

The Zedboard is the development platform that the embedded system design is implemented on. The FPGA hardware design and operating system are stored on an SD card and the Zedboard is loaded with the information from the SD card. After booting the hardware design and the operating system, the SD card then operates as the hard drive for the system.

Figure 7.1 Zedboard Development Board [29]

7.2 Xillinux

The Zynq-7000 APSoC is booted with a special Linux distribution kernel developed by Xillybus. This Linux kernel is called Xillinux. Xillinux is a software + FPGA code kit for running a full-blown graphical desktop on the Zedboard and some other development boards that contain a Zynq-7000. Xillinux is based on Ubuntu LTS 12.04 kernel for ARM and allows the Zedboard to behave like a PC with the SD

card as its hard disk drive. A keyboard and mouse can be plugged into the Zedboard for full desktop interaction with either a Linux test console or the full blown Gnome desktop environment [30]. The VGA output port of the Zedboard is used for the computer's display output. As a result of Xillinux being developed by Xillybus, it comes pre-installed with the Xillybus kernel driver software for interfacing to the Xillybus IP core on the FPGA. Xillinux is used as the operating system in the embedded design on the Zedboard because it is free for evaluation purposes, easy to implement, and comes installed with the Xillybus software. Documentation is provided by Xillinux for how to download, compile, and boot the Xillinux kernel on the Zedboard [31].

7.3 ZC706

The ZC706 Evaluation board is a very feature rich engineering board containing many different hardware peripherals with a Zynq-7000 APSoC at the heart of it. The Zynq-7000 chip is the XC7Z045 version which contains a much larger FPGA than the 7020 chip that comes installed on the Zedboard. The FPGA on the 7045 chip on the ZC706 board is equivalent in the amount of resources as that of the Kintex-7 generation FPGA. It contains 1 GB of DDR3 memory SODIMM on the PL side of the chip so that the FPGA can interface to the memory. It also contains 1 GB of DDR3 component memory on the hardened processor side for it to interface to. The board also contains two 128 Mb Quad SPI (QSPI) flash memory chips. It contains a USB 2.0 transceiver along with a micro-B USB connector as well. It has a Secure Digital (SD) connector with a USB JTAG interface. The board also has a PCI Express endpoint connectivity for either Gen1 4-lane (x4) and a Gen2 4-lane (x4). It has an SFP+ connector for high speed Ethernet, HDMI connector, I2C bus, a

multitude of GPIO pins, and dual 12-bit analog to digital converters [32]. Figure 7.2 shows a photo of the ZC706 board.

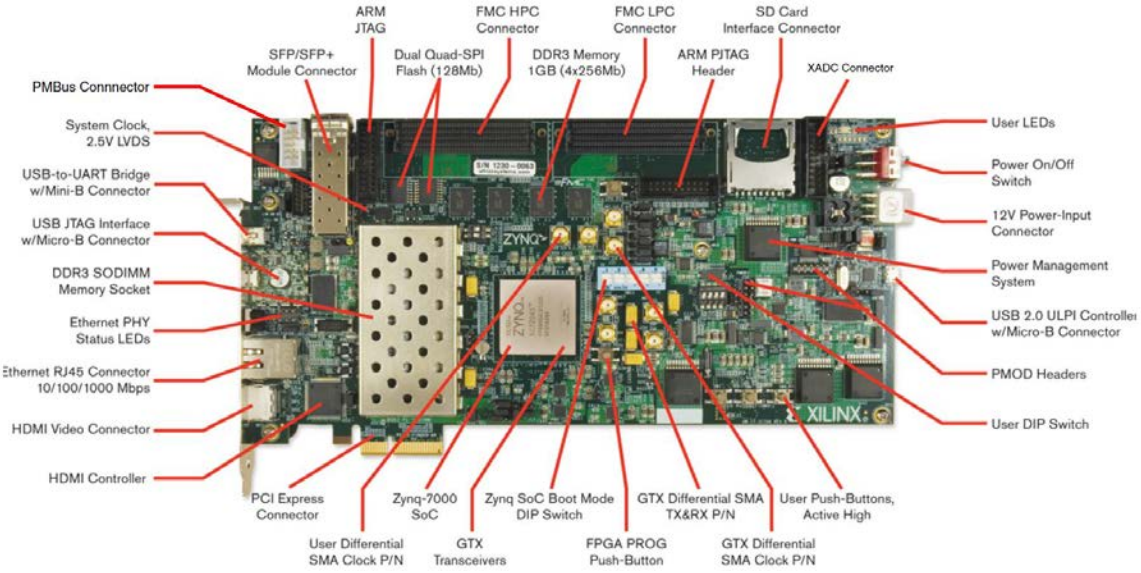


Figure 7.2 ZC706 Development Board [33]

For this research, the ZC706 board is used for its PCIe connectivity for interfacing to the Intel processor for the hardware accelerated ciphers for the workstation system design. The Xillybus IP core can be configured to operate on the PCIe bus the same as it does on the AXI4 bus. The Xillybus IP core is capable of utilizing the full 4-lane bus width of the ZC706 PCIe interface at the 66 MHz bus clock. Although the ZC706 development board has the Zynq-7000 APSoC chip with the dual-core ARM processors available, the ARM cores are not used in the workstation design for the hardware accelerated ciphers. This is because the host software is executed on the Intel processor host and the data flows from the host PC to the ZC706 board via the PCIe bus. Thus, the data is received by the ZC706 board through the PCIe core inside the FPGA fabric of the Zynq which then passes the

data through the hardware ciphers and ultimately back to the PC. Therefore, the ARM cores are never needed to perform the cipher operations in the workstation system design.

7.4 Workstation

The workstation that is used for the implementation of the software and hardware ciphers is the Dell Precision Tower 7910 high end personal computer. It contains an octa-core Intel Xeon processor, 32 GB of RAM, and a 1 TB solid state hard drive. The workstation runs the Ubuntu 14.04 Linux operating system. There are several expansion slots in the workstation for adding peripheral cards on the PCI/PCIe bus. The ZC706 board sits on the PCIe bus for the workstation design.

7.5 Measurement Setup

The performance metrics used for comparing the software and hardware ciphers are the execution time, data throughput, and speedup. The execution time to complete the file encryption/decryption test or hardware sink test for each of the cipher modes is recorded and then used for calculating data throughput and speedup. Each of the software applications, as discussed in CHAPTER 6, are executed for each test and cipher. Based on the test being executed, either file encryption/decryption or hardware sink, determines how the execution time is computed in the application. For the file encryption/decryption test, the execution time for the software ciphers is computed as the time spent executing inside of the API calls into OpenSSL and the execution time for the hardware ciphers is computed as the time spent in the API "write" call of Xillybus. For the hardware sink test, the execution time for the software ciphers is computed as the time spent

inside the API calls into OpenSSL and the time spent in the API "write" call of Xillybus. The execution time for the hardware ciphers remains just the time spent inside the Xillybus "write" API function. Note, the execution times for the hardware ciphers are highly influenced by the speed of the file I/O operations on the platform executing the software applications because the Xillybus "write" API call initiates a transfer of data into DMA buffers in RAM and then a DMA transaction for transferring data across the bus. However, if data is written to the DMA buffers at a faster rate than can be transferred across the bus then the DMA buffers will cause the "write" function of Xillybus to stall which causes the execution time to increase. This is the case for the workstation platform and will be discussed further in the micro-benchmarking and results sections. The file encryption/decryption test is discussed in more detail in Section 7.6 and the hardware sink test is discussed in further detail in Section 7.7.

Each test is executed for every combination of the input parameters of the software applications. The input parameters that are varied for all the tests are the file size, cipher mode, block cipher operation, and cipher implementation. There are sixteen different file sizes ranging from 32 KB to 1 GB as shown inTable 7.1. There are three cipher modes (ECB, CBC, CTR), two block cipher operations (encryption and decryption), and two cipher implementations (software and hardware). All the varied parameters result in 16*3*2*2 = 192 different parameter configurations. All the different parameter configurations are executed on both the embedded and workstation designs.

The execution time of the software and hardware ciphers are used to compute the throughputs of each along with the speedup of the hardware ciphers relative to the software ciphers. Throughput is defined as the amount of data that passes

through the ciphers within the time period of a second. Throughput is computed by dividing the amount of data from the input file by the execution time to complete the test. Speedup is defined as the performance improvement, in terms of execution time, of the hardware ciphers relative to the software ciphers. Speedup is computed as the execution time of the software ciphers divided by the execution time of the hardware ciphers.

Table 7.1 File Sizes used for Test Applications

File Sizes
32 KB
64 KB
128 KB
256 KB
512 KB
1 MB
2 MB
4 MB
8 MB
16 MB
32 MB
64 MB
128 MB
256 MB
512 MB
1 GB

7.6 File Encryption/Decryption Test

One of the tests used to evaluate the performance of the software and hardware ciphers is file encryption and decryption. Data files varying from 32 KB to 1 GB are encrypted and then decrypted using either the software or hardware ciphers.

The two C++ software test applications are used to control the file encryption or decryption process by either interfacing to the OpenSSL library API or interfacing to the hardware accelerated ciphers in the FPGA. The application has multiple

command line options that can be passed into the software at run-time. These command line options include selecting which cipher mode to execute (i.e. ECB, CBC, CTR), which cryptographic operation to perform (i.e. encryption/decryption), whether to perform the operation in software or hardware, to specify a password for the cipher key and initialization vector generation or authentication, to specify the file name of the output file for the output data (either encrypted or decrypted) to be written to, and finally the name of the input file containing the data to be manipulated. After processing the command line parameters, the application begins by generating the cipher key and initialization vector to be used in the cipher operation. If preparing for the encryption operation, the application uses the Password-Based Key Derivation Function (PBKDF) available in the OpenSSL library to derive a cipher key and initialization vector from the password specified by the user. If preparing for the decryption operation, the application still generates the cipher key and initialization vector from the specified password, but after doing so it compares the output for each against the hashed values read from the input file in order to authenticate the password that is provided by the user. If the values do not match, then the user did not specify the correct password in order to successfully decrypt the data file and the application aborts.

Once the application has the cipher key and initialization vector it can continue to perform the necessary cryptographic operation specified by the user. The application incrementally reads the data from the input file and passes the data buffer through the cryptographic operation. The cryptographic operation can either be performed in software by OpenSSL or in hardware on the FPGA. The data is passed into the OpenSSL library via the EVP API function calls provided by the OpenSSL library which then just returns a data buffer with the output data from

the operation. The interface to the FPGA hardware acceleration is accomplished through the use of function calls into the Xillybus kernel module driver. The hardware accelerated ciphers on the FPGA manipulate the data set and then transfers the data back to the CPU. The output data from either OpenSSL or the FPGA is then written to the output file. This process is repeated as many times as necessary to read the entire input file and perform the cipher operation on all the input data. The same process of events are performed for both the maximum throughput and OpenSSL extension software applications, but the OpenSSL extended software only interfaces to the OpenSSL EVP API and the OpenSSL library uses either the software or hardware ciphers.

During the execution of the applications for the file encryption/decryption test, the execution time is computed in order to do post-analysis of the performance of the software and hardware ciphers. The execution time is recorded for the amount of time it takes to complete the cipher operations on the entire input data file. The application reads the input file in increments so the system time is latched right before passing the data to the ciphers and then latched again once the data is returned from the ciphers and the delta between the two times is then added to the overall execution time. For the OpenSSL extension application, the execution time is just the period of time spent calling the OpenSSL EVP functions. The same applies for the maximum throughput application when accessing the OpenSSL software ciphers. The execution time is computed differently for the hardware ciphers for the maximum throughput application because it operates in a multithreaded environment. In order to not capture the latencies involved with the file I/O operations, which are present in both threads interfacing with Xillybus, the execution time is computed as the time spent making calls into the “write” function

of the Xillybus kernel module plus the period of time between the write thread joining with the main thread and the read thread joining with the main thread. The execution time being computed in this manner is highly influenced by the file I/O operations of the platform because if data is read from the platform hard drive at a fast enough rate and written to the DMA buffers then once the DMA buffers become full the application will stall in the "write" function of Xillybus and cause the execution time to increase.

7.7 Hardware Sink Test

The hardware sink test is used to experiment with the situation where data is sent to a hardware peripheral that communicates externally to the host platform (i.e. an Ethernet port). Assuming the data being transmitted by the hardware peripheral needs to be encrypted then the data will have to be encrypted in software or hardware prior to transmission. Logically, if the data is encrypted in software then it must be transferred to the hardware peripheral after the necessary software operations, but if the data is encrypted in hardware then the raw data can immediately be transferred to the hardware and forgotten because the hardware performs the remaining operations needed before passing it to the peripheral. This is simulated with the hardware sink test. For this test, the same software applications are used, but given the command line option for the hardware sink test. For the software ciphers, the same process is executed as in the file encryption/decryption test, but instead of sending the output data to an output file the data is sent across the bus to the FPGA to a hardware sink. This simulates the case where the software has to encrypt data and then send the data to a hardware peripheral. Utilizing the hardware accelerated ciphers works differently. The process is the same as the file

encryption/decryption test up to where the input data is sent to the FPGA. Once the data is sent to the FPGA it never returns to the host processor because it is sent to the hardware sink after being sent through the cipher. This simulates the "send it and forget it" situation where the processor can send the raw data to the hardware accelerated ciphers that encrypts the data and then sends it to a peripheral without the processor ever touching the data again.

The execution times for this test are recorded in a similar manner to the file encryption/decryption test. For the software ciphers, the time required to perform the cipher in the OpenSSL library and then send the data to the FPGA is recorded as the execution time. For the hardware ciphers, just the time required to send the data to the FPGA is recorded as the execution time.

CHAPTER 8

MICRO-BENCHMARKING

A set of micro-benchmarking tests are executed within each of the different design environments to better understand the influence different components of the system have on the performance of the overall design. The components that are analyzed are the bus architectures, the hard drives, and the hardware accelerated ciphers. The bus architectures and the hard drives are evaluated using actual experimental data retrieved through micro-benchmarking applications, but the hardware ciphers are evaluated using analytical techniques.

8.1 Embedded Design Micro-benchmarking

The embedded design components that are evaluated include the AXI4 bus, the SD card used by the Zedboard as the hard drive, and the ciphers implemented on the FPGA fabric of the Zynq-7000 APSoC.

8.1.1 AXI4 Bus

The AXI4 bus in the context of the micro-benchmarking refers to the bus architecture itself and the Xillybus interface solution that wraps the bus architecture. Due to the use of Xillybus, there was no other way to test the performance of the AXI4 bus without using the Xillybus interface since their kernel module is what drives the DMA transactions on the bus. Furthermore, the performance of the AXI4 bus as seen from the perspective of the software developer includes the use of the DMA engine and the available DMA buffers allocated in the

system. These greatly improve the performance of the AXI4 bus because the DMA buffers allow the software to just transfer data around in the DRAM memory and the DMA engine performs large bursts across the bus. The application used to test the performance of just the AXI4 bus incrementally reads a 1 GB data file and streams the data across the bus. The total execution time to complete all the "write" function calls into the Xillybus kernel module for writing all the data across the bus is used to compute the throughput of the AXI4 bus which is found to be 500 MB/s.

8.1.2 Zedboard HDD (SD Card)

The Zedboard uses an SD memory card as the hard drive for the embedded system. The performance of the file I/O operations using the SD card affect the overall system performance because it affects how fast the test application can provide new data to the ciphers and retrieve output data from the ciphers. The test applications must read from an input file before providing new data to the ciphers and must write to an output file before it can retrieve more output data from the ciphers. The test application used to test the performance of the SD card attempts to read the 1 GB file into DRAM and the execution time to complete the file reading is used to calculate the throughput of the file I/O which is found to be about 11 MB/s.

8.1.3 Hardware Accelerated Ciphers

The ECB and CBC ciphers use the non-pipelined AES core and the CTR cipher mode uses the pipelined AES core. Thus, the ciphers have different performance characteristics. Some analytical techniques are used to evaluate the throughputs of each of the cipher modes based on the implementations and timing of the cipher cores and the AES cores. By taking the clock rate of the AES cores and the number

of clock cycles it takes to complete the cipher for any input state matrix the throughput is calculated. The ideal throughputs of the ECB and CBC ciphers are found to be about 35 MB/s and the ideal throughput of the CTR cipher is about 1.6 GB/s. These throughputs are computed using the 100 MHz clock used in the embedded design.

8.2 Workstation Design Micro-benchmarking

The workstation design components that are evaluated include the PCIe bus, the spinning disk hard drive of the workstation, and the ciphers implemented on the FPGA fabric of the Zynq-7000 APSoC on the ZC706.

8.2.1 PCIe Bus

The micro-benchmarking of the PCIe bus is similar to that of the AXI4 bus for the embedded design. It also includes the bus architecture itself and the Xillybus interface solution that wraps the PCIe bus. Therefore, the results of the micro-benchmarking include the use of the DMA engine and the available DMA buffers allocated in the system by Xillybus. The application used to test the performance of just the PCIe bus incrementally reads a 1 GB data file and streams the data across the bus. The total execution time to complete all the "write" function calls into the Xillybus kernel module for writing all the data across the bus is used to compute the throughput of the PCIe bus which is found to be 600 MB/s.

8.2.2 Workstation HDD

The workstation uses a spinning disk hard drive. The rate at which the workstation can perform file I/O operations affects the overall system performance because it affects how fast the test application can provide new data to the ciphers

and retrieve output data from the ciphers. The test application used to test the performance of the hard drive attempts to read the 1 GB file into RAM and the execution time to complete the file reading is used to calculate the throughput of the file I/O which is found to be about 1.7 GB/s.

8.2.3 Hardware Accelerated Ciphers

The performance of the hardware ciphers in the workstation design is computed the same as it is for the embedded design through the use of analytical techniques. The ideal throughputs of the ECB and CBC ciphers are found to be about 35 MB/s which is the same as for the embedded design because the same 100 MHz clock frequency is used by these cores in both designs. However, the ideal throughput of the CTR cipher is about 4 GB/s which is greater than the CTR cipher in the embedded design because the workstation design provides a 250 MHz clock to the CTR cipher instead of the 100 MHz clock. These throughputs are based on the throughput of the hardware ciphers if they were standalone and were not affected by anything else in the system.

CHAPTER 9

RESULTS

The results section discusses the performance results of the software and hardware ciphers. The following sections will provide plots for the throughputs achieved by both the software and hardware ciphers and also the speedups of the hardware ciphers compared to the software ciphers. Section 9.1 and its subsections will discuss the results for the embedded design hosted on the Zedboard development board. Section 9.2 and its subsections will discuss the results for the workstation design hosted on an Intel workstation and ZC706 development board.

9.1 Results for Zedboard Embedded Design

The execution times for the software and hardware ciphers are recorded by the test applications on the Zedboard and used to compute the throughput and speedup of the ciphers. The speedup is used to show the performance improvement, if any, of the hardware ciphers versus the software ciphers. The results of the embedded design show that the hardware ciphers do achieve a considerable performance improvement over the software ciphers. The results show drastically different results for the file encryption/decryption test between the maximum throughput application and the OpenSSL extension application. The maximum throughput application achieved a maximum speedup and throughput of 25x and 450 MB/s, respectively, for the hardware ciphers. The OpenSSL extension application achieved a maximum speedup and throughput of 6x and 130 MB/s, respectively, for the hardware ciphers. However, the results are very similar

between the two applications for the hardware sink test. Both software applications achieved a maximum speedup and throughput of 30x and 500 MB/s for the hardware ciphers.

In the following subsections, the plots will be labeled with either “File” or “HW Sink” which just refers to the test that the plots correspond to. The label “File” corresponds to the file encryption/decryption test and the label “HW Sink” corresponds to the hardware sink test.

9.1.1 Results for File Encryption/Decryption Test using Maximum Throughput Application

The results for the file encryption/decryption test using the maximum throughput software application are shown in Figure 9.1 to Figure 9.2. Figure 9.1 (A) and (B) show the throughputs of the software ciphers for encryption and decryption, respectively. The ECB and CBC ciphers have a throughput of about 18 MB/s and the CTR cipher has a throughput of about 22 MB/s. Figure 9.1 (C) and (D) show the throughputs of the hardware ciphers for encryption and decryption, respectively. They show that all three hardware ciphers achieved throughputs of about 400 MB/s. The reason all three hardware ciphers achieved about the same throughput even though the CTR cipher is fully pipelined and has a better nominal throughput is due to the effects of the use of DMA and the latency involved with file I/O on the SD card of the Zedboard. For the hardware ciphers, the execution time is computed as the total time spent in the “write” function call to Xillybus which just transfers memory within the DRAM and then begins transferring data to the FPGA. Since it takes a significant amount of time for the processor to read another block of data from the input file most of the data has already been transferred across the bus

and even possibly returned from the FPGA. This effectively hides the latency involved with performing the ciphers in the FPGA.

Figure 9.2 shows the speedups achieved by the hardware ciphers for encryption and decryption, respectively. They show that the ECB and CBC ciphers achieved speedups of about 25x for both processes. The CTR cipher achieved a speedup of about 20x. The reason the ECB and CBC ciphers achieved a higher speedup even though these hardware ciphers have a lower throughput than the CTR cipher is because the performance of these ciphers in OpenSSL are considerably less than the OpenSSL implementation of the CTR cipher. Therefore, in comparison, the hardware ECB and CBC ciphers have a greater overall improvement over OpenSSL than the CTR cipher.

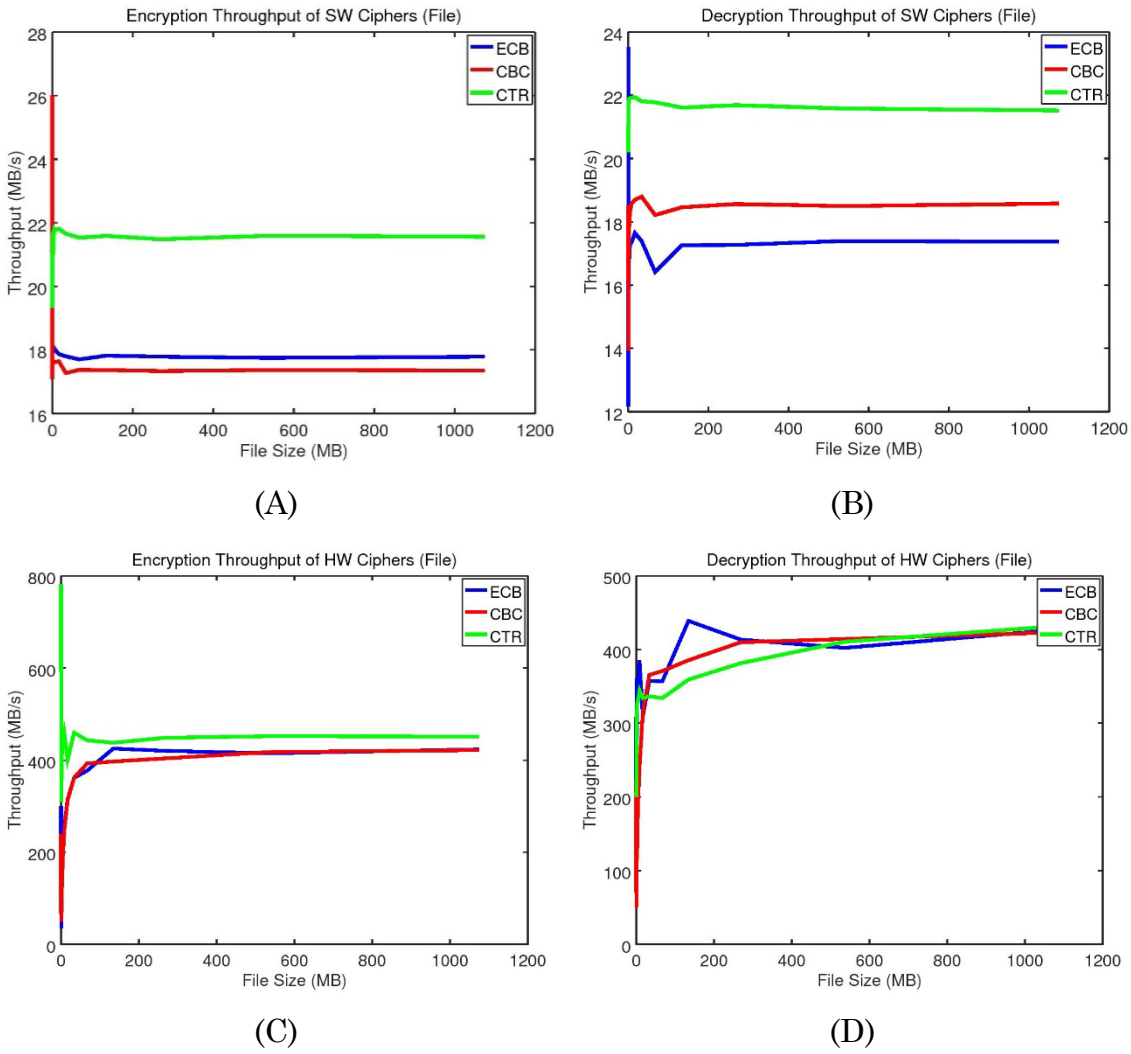
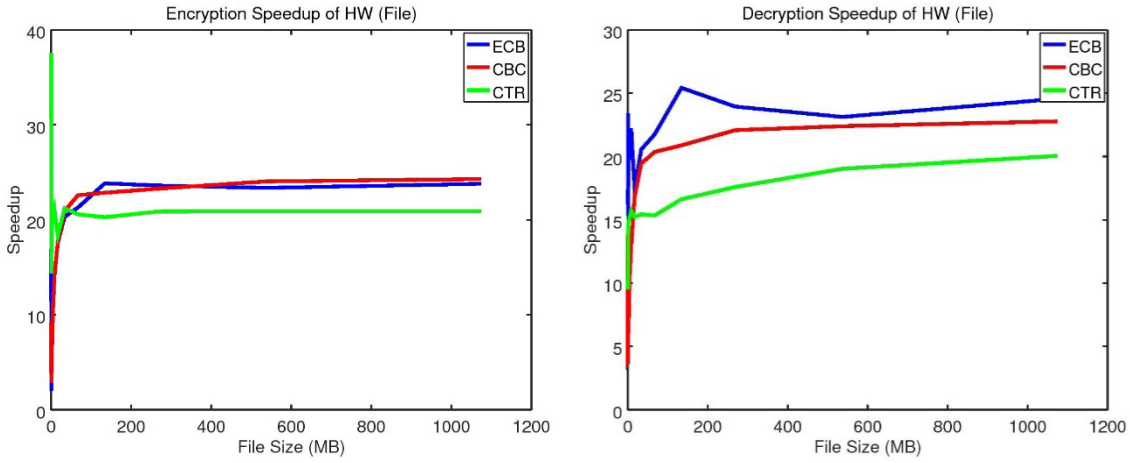


Figure 9.1 (A) Encryption Throughput of Software Ciphers (File), (B) Decryption Throughput of Software Ciphers (File), (C) Encryption Throughput of Hardware Ciphers (File), (D) Decryption Throughput of Hardware Ciphers (File)



(A) (B)

Figure 9.2 (A) Encryption Speedup of Hardware Ciphers (File), (B) Decryption Speedup of Hardware Ciphers (File)

9.1.2 Results for Hardware Sink Test using Maximum Throughput Application

The results for the hardware sink test using the maximum throughput software application are shown in Figure 9.3 and Figure 9.4. Figure 9.3 shows the throughputs of the software and hardware ciphers while using the hardware sink. Figure 9.3 (A) and (B) show that the ECB and CBC software ciphers have a throughput of about 17 MB/s and the CTR cipher has a throughput of about 21 MB/s. These throughputs are less than that achieved for the file encryption/decryption test which is expected because the data must be encrypted or decrypted before it is sent across the bus to the hardware sink. This adds an extra step to the process; therefore, decreasing the overall throughput. Figure 9.3 (C) and (D) show that the ECB and CBC hardware ciphers achieved a throughput of about 450 MB/s; whereas, the CTR cipher achieved a throughput of about 500 MB/s. This is expected because as the processor continuously reads data from the input file and transfers it into the DMA buffers the non-pipelined functionality of the ECB and CBC ciphers could cause the DMA buffers to fill up and put backpressure on the processor from transferring more data into the DMA buffers. This causes the application to stall while waiting on space to become available in the DMA buffers to receive the current data block. However, this is not occurring with the CTR cipher because it is fully pipelined and is clocked off the same clock as the AXI4 bus so data will never get backed up in the FPGA and thus will not cause the DMA buffers in RAM to become full.

Figure 9.4 shows the speedups achieved by the hardware ciphers for encryption and decryption, respectively. The plots show that the ECB and CBC

ciphers achieved speedups of about 30x for encryption and decryption. The CTR cipher achieved a speedup of about 20x to 25x. The reason the ECB and CBC ciphers achieved a higher speedup than the CTR cipher is the same reason as in the file encryption/decryption test. It is because the performance of the CTR cipher in OpenSSL is better than the ECB and CBC ciphers so the comparative improvements are better for the ECB and CBC ciphers.

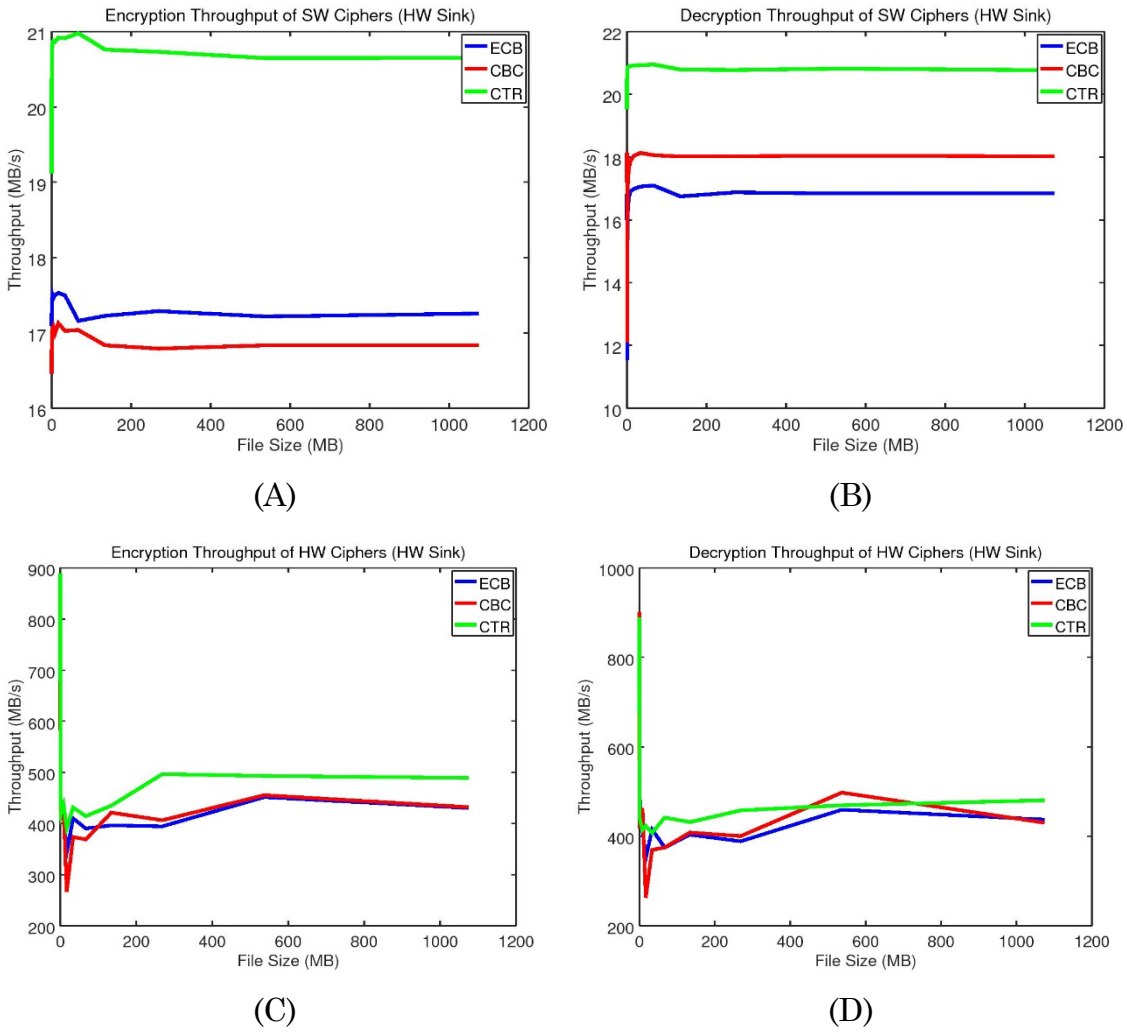
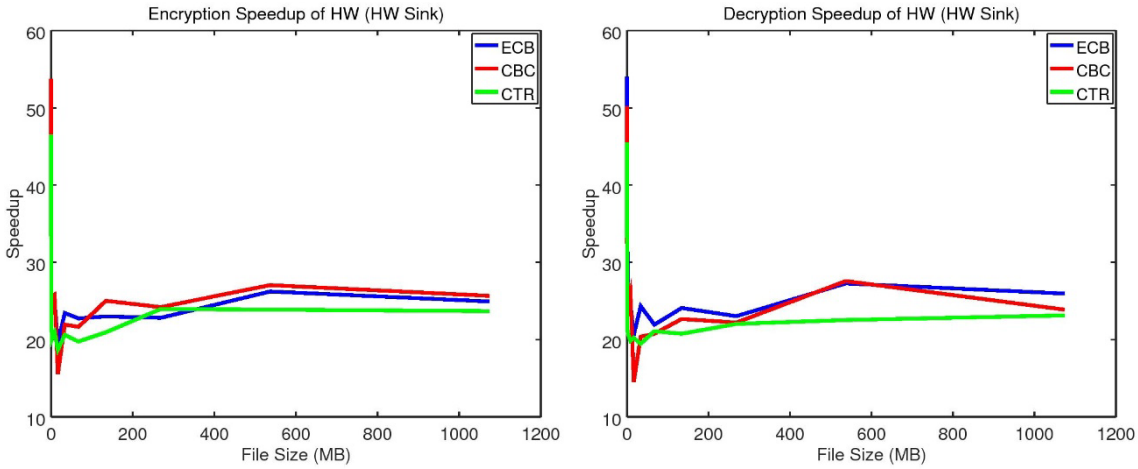


Figure 9.3 (A) Encryption Throughput of Software Ciphers (HW Sink), (B) Decryption Throughput of Software Ciphers (HW Sink), (C) Encryption Throughput of Hardware Ciphers (HW Sink), (D) Decryption Throughput of Hardware Ciphers (HW Sink)



(A) (B)

Figure 9.4 (A) Encryption Speedup of Hardware Ciphers (HW Sink), (B) Decryption Speedup of Hardware Ciphers (HW Sink)

9.1.3 Results for File Encryption/Decryption Test using OpenSSL Extension Application

The results for the file encryption/decryption test using the OpenSSL extension software application are shown in Figure 9.5 and Figure 9.6. The throughputs for the software and hardware ciphers are shown in Figure 9.5. Figure 9.5 (A) and (B) show the software ciphers achieved throughputs of about 18 MB/s for the ECB and CBC ciphers and 21 MB/s for the CTR cipher. Comparatively, the hardware ciphers, shown in Figure 9.5 (C) and (D), achieved throughputs of about 18 MB/s for the ECB and CBC ciphers and 130 MB/s for the CTR cipher.

The speedups of the hardware accelerated ciphers for encryption and decryption are shown in Figure 9.6. The ECB and CBC hardware ciphers did not experience any speedup over their software counterparts and the CTR cipher only achieved a speedup of 6x. This is due to the sequential nature of the OpenSSL library. The extension to the OpenSSL library allows for the utilization of both the software and hardware ciphers through the same API to the top level application. Therefore, the top level application must wait until the data is returned from the OpenSSL library before continuing execution regardless of performing the ciphers in software or hardware. Thus, the use of DMA buffers that hid the latency of the bus transfers and hardware ciphers in the maximum throughput application are now present in the OpenSSL extension application which results in a decreased performance when using the hardware ciphers compared to the maximum throughput application.

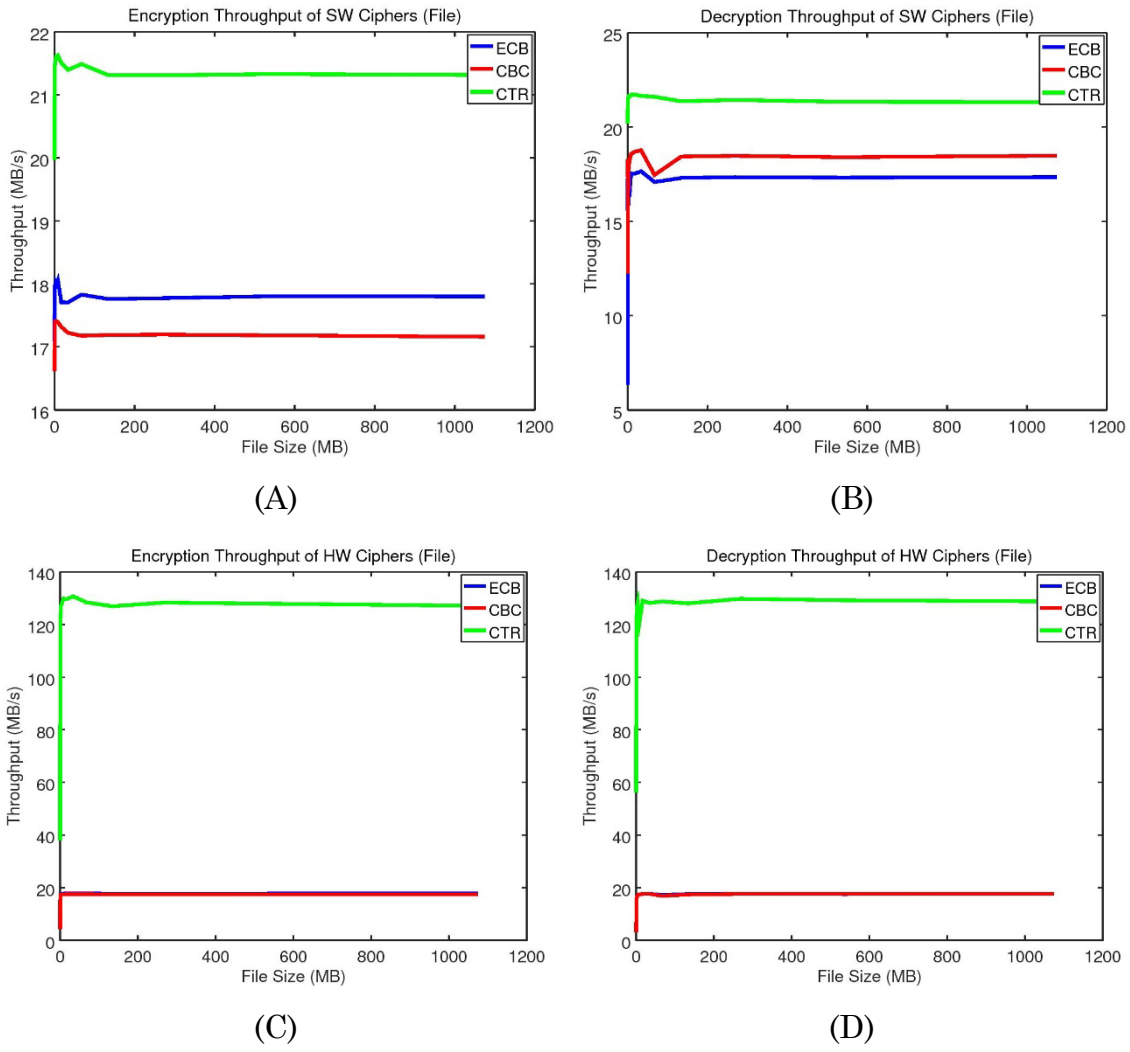
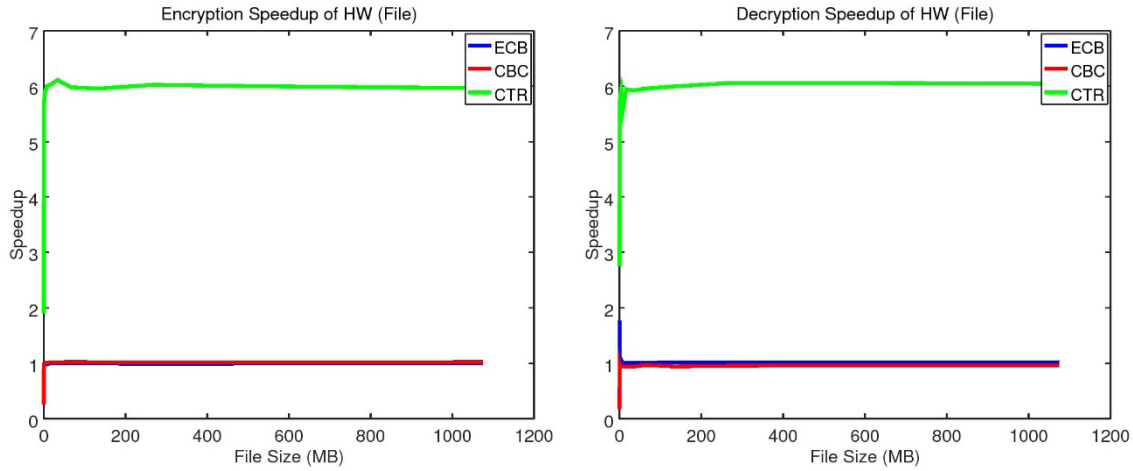


Figure 9.5 (A) Encryption Throughput of Software Ciphers (File), (B) Decryption Throughput of Software Ciphers (File), (C) Encryption Throughput of Hardware Ciphers (File), (D) Decryption Throughput of Hardware Ciphers (File)



(A) (B)

Figure 9.6 (A) Encryption Speedup of Hardware Ciphers (File), (B) Decryption Speedup of Hardware Ciphers (File)

9.1.4 Results for Hardware Sink Test using OpenSSL Extension Application

The results for the hardware sink test using the OpenSSL extension software application are shown in Figure 9.7 and Figure 9.8. The throughputs for the software and hardware ciphers are shown in Figure 9.7. The software ciphers achieved throughputs of about 17 to 18 MB/s for the ECB and CBC ciphers and 20 MB/s for the CTR cipher. The hardware ciphers achieved throughputs of about 450 to 500 MB/s for all three ciphers.

The speedups of the hardware ciphers for encryption and decryption are shown in Figure 9.8. The ECB and CBC ciphers achieved a speedup upwards of 30x and the CTR cipher achieved a speedup of about 20x. The reason why the ECB and CBC ciphers have a higher speedup than the CTR cipher is because the ECB and CBC ciphers in OpenSSL do not perform as well as the CTR cipher so there is more room for improvement with those two ciphers. All three hardware ciphers achieved very high speedups which is attributed to the fact that with the hardware sink the software is able to send the data into the DMA buffers and immediately return control to the top level application from the OpenSSL library. This is because for the hardware sink test OpenSSL does not have to wait for the data to return from the FPGA before returning control to the top level application. This removes the latency involved with actually performing the ciphers in the FPGA from the computation of the execution time.

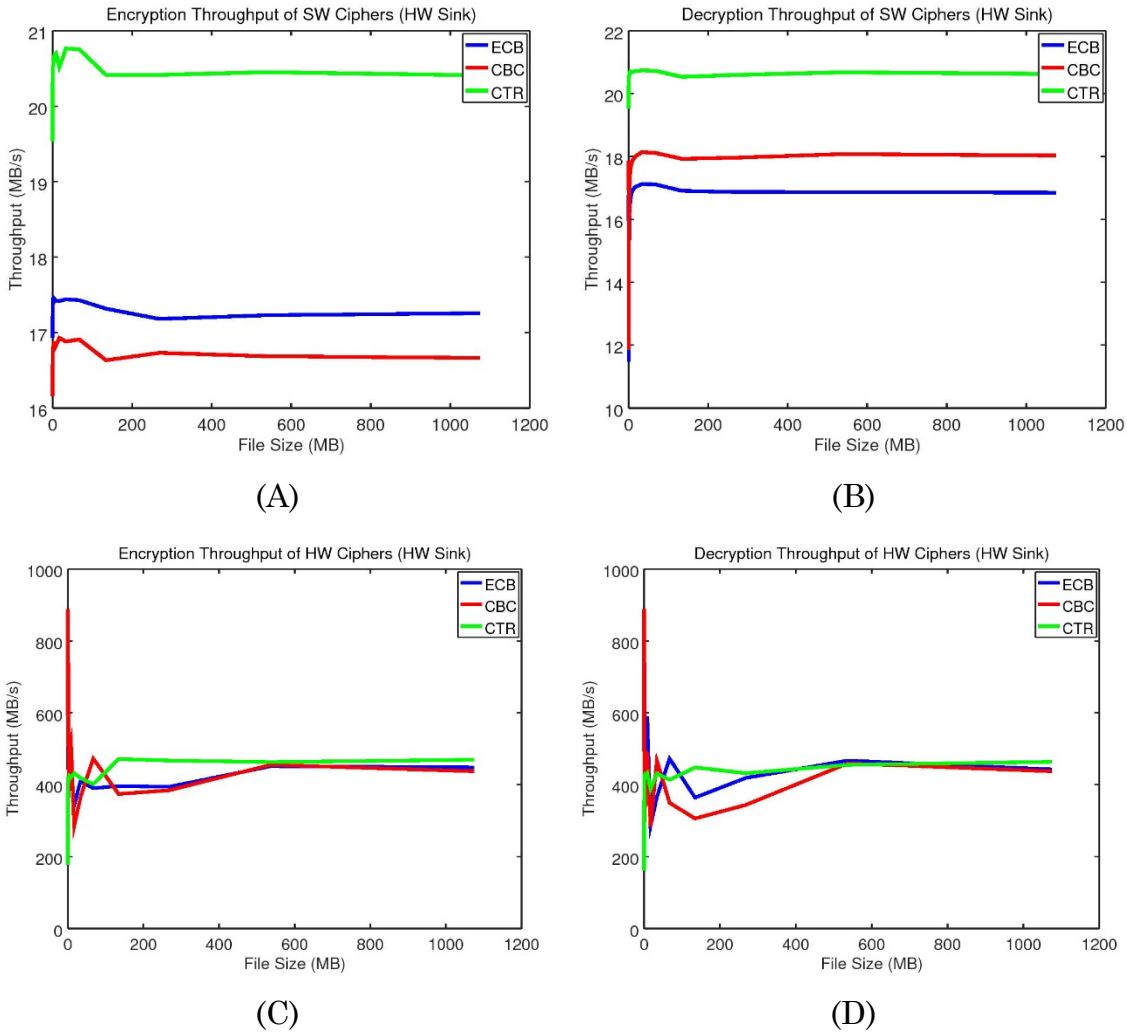
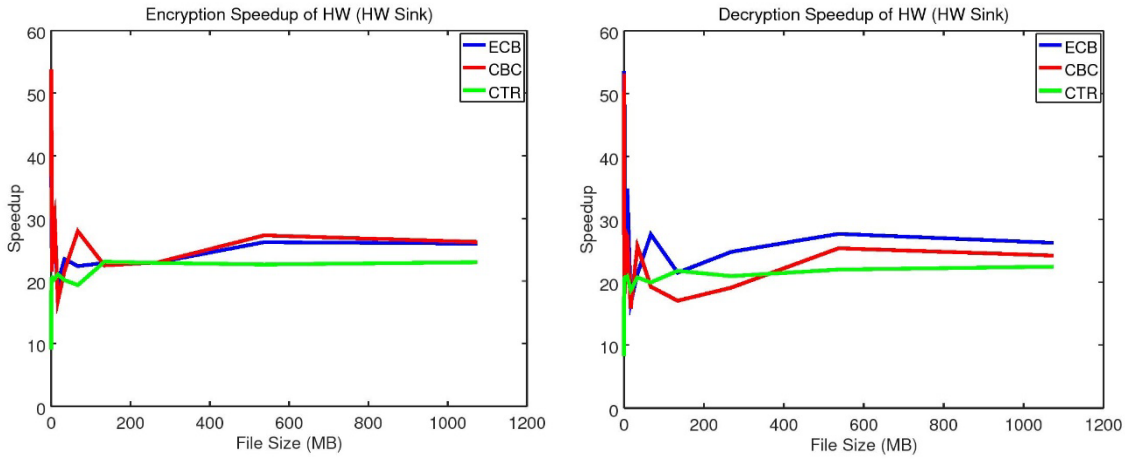


Figure 9.7 (A) Encryption Throughput of Software Ciphers (HW Sink), (B) Decryption Throughput of Software Ciphers (HW Sink), (C) Encryption Throughput of Hardware Ciphers (HW Sink), (D) Decryption Throughput of Hardware Ciphers (HW Sink)



(A) (B)

Figure 9.8 (A) Encryption Speedup of Hardware Ciphers (HW Sink), (B) Decryption Speedup of Hardware Ciphers (HW Sink)

9.2 Results for ZC706 Workstation Design

The execution times for the ciphers for the OpenSSL software implementations and the FPGA hardware implementations are recorded by the test applications on the Intel workstation using the ZC706 for the hardware accelerated design. The execution times are used to compute the speedup and throughput of the ciphers. The speedup is used to show the performance improvement, if any, of the hardware ciphers versus the software ciphers. The Intel AESNI hardware acceleration option for Intel processors is available on the workstation used for this design so it is also used for performance comparisons with the FPGA hardware acceleration. The OpenSSL library is used for executing the software ciphers with and without the AESNI option. The option with the use of AESNI is referred to in this paper as the software+ option.

The results of the workstation design show that the hardware ciphers do not achieve a considerable performance improvement over the software ciphers and experience a drastic decrease in performance compared to the software+ ciphers. The results are similar between the maximum throughput application and the OpenSSL extension application. The hardware accelerated ECB and CBC ciphers experienced speedups of less than 1 while the CTR cipher achieved speedups upwards of 1.5x. The software+ ciphers produced speedups between 9x and 14x compared to the software ciphers and about 150x compared to the hardware accelerated ECB and CBC ciphers and 7x compared to the hardware accelerated CTR cipher. The software ciphers achieved throughputs upwards of 350 MB/s, the software+ ciphers have throughputs upwards of 3.3 GB/s, and the hardware ciphers have throughputs upwards of 550 MB/s.

In the following subsections, the plots will be labeled with either “File” or “HW Sink” which just refers to the test that the plots correspond to. The label “File” corresponds to the file encryption/decryption test and the label “HW Sink” corresponds to the hardware sink test.

9.2.1 Results for File Encryption/Decryption Test using Maximum Throughput Application

The results for the file encryption/decryption test using the maximum throughput software application are shown in Figure 9.9 through Figure 9.12. The throughputs for the software and hardware ciphers are shown in Figure 9.9 and Figure 9.10. Figure 9.9 (A) and (B) show the software ciphers achieved throughputs of about 225 MB/s for the ECB and CBC ciphers and 350 MB/s for the CTR cipher. Figure 9.9 (C) and (D) show that the software+ ciphers achieved throughputs of about 3 to 3.5 GB/s. The encryption process shows a decreased throughput for the CBC cipher down to about 2.25 GB/s. Lastly, Figure 9.10 shows that the hardware ciphers achieved throughputs of about 22 MB/s for the ECB and CBC ciphers and 500 MB/s for the CTR cipher.

Figure 9.11 shows the speedup of the software+ ciphers for encryption and decryption. The ECB cipher achieved a speedup of about 14x and the CBC and CTR ciphers have a speedup of about 10x. Figure 9.12 (A) and (B) show the speedups of the hardware ciphers relative to the software ciphers where the ECB and CBC hardware ciphers have a speedup of about 0.1x and the CTR cipher has a speedup of about 1.5x. Figure 9.12 (C) and (D) show the speedups of the hardware ciphers relative to the software+ ciphers where the ECB and CBC ciphers have a speedup of about 0.005x and the CTR cipher has a speedup of about 0.15x. The hardware CTR

cipher has a much greater throughput than the ECB and CBC ciphers which is why the CTR cipher does not have as low of a speedup when compared to the software+ ciphers.

The reason for the decrease in performance of the ECB and CBC hardware ciphers compared to the software ciphers can be attributed to a couple different factors. One is the non-pipelined functionality of the ciphers and the second is the performance of the file I/O operations on the workstation. The PCIe data transfers over the x4 interface are providing data to the FPGA at a much faster rate than the non-pipelined ciphers can consume the data thus causing the buffers in the FPGA and, ultimately, the DMA buffers in RAM to get backed up. Also, the workstation HDD performs file I/O operations so much faster than the PCIe bus can transfer data to the FPGA that the application is writing new data into the DMA buffers in RAM faster than the PCIe interface can transfer data causing the buffers to fill up and the proceeding calls to the Xillybus write function will have to block until space becomes available in the DMA buffers. Recalling that the execution time for these runs is the time spent in the Xillybus write function means that these memory stalls cause the execution time to increase. These results clearly show the presence of bottlenecks in the system for the hardware accelerated ciphers. For the ECB and CBC ciphers, the bottleneck is the AES block cipher core since it is non-pipelined. The throughput of these ciphers are comparable to the 35 MB/s theoretical throughputs for these ciphers found in the micro-benchmarking section. On the other hand, the bottleneck for the CTR ciphers shifts to the PCIe bus. Since the CTR cipher is fully pipelined it can consume data at the same rate the PCIe bus can provide it; thus, making the PCIe bus the limiting factor of the performance for the

CTR cipher. This is supported by the fact that it achieves a throughput of 500 MB/s in this test and the PCIe bus has a theoretical throughput of about 600 MB/s.

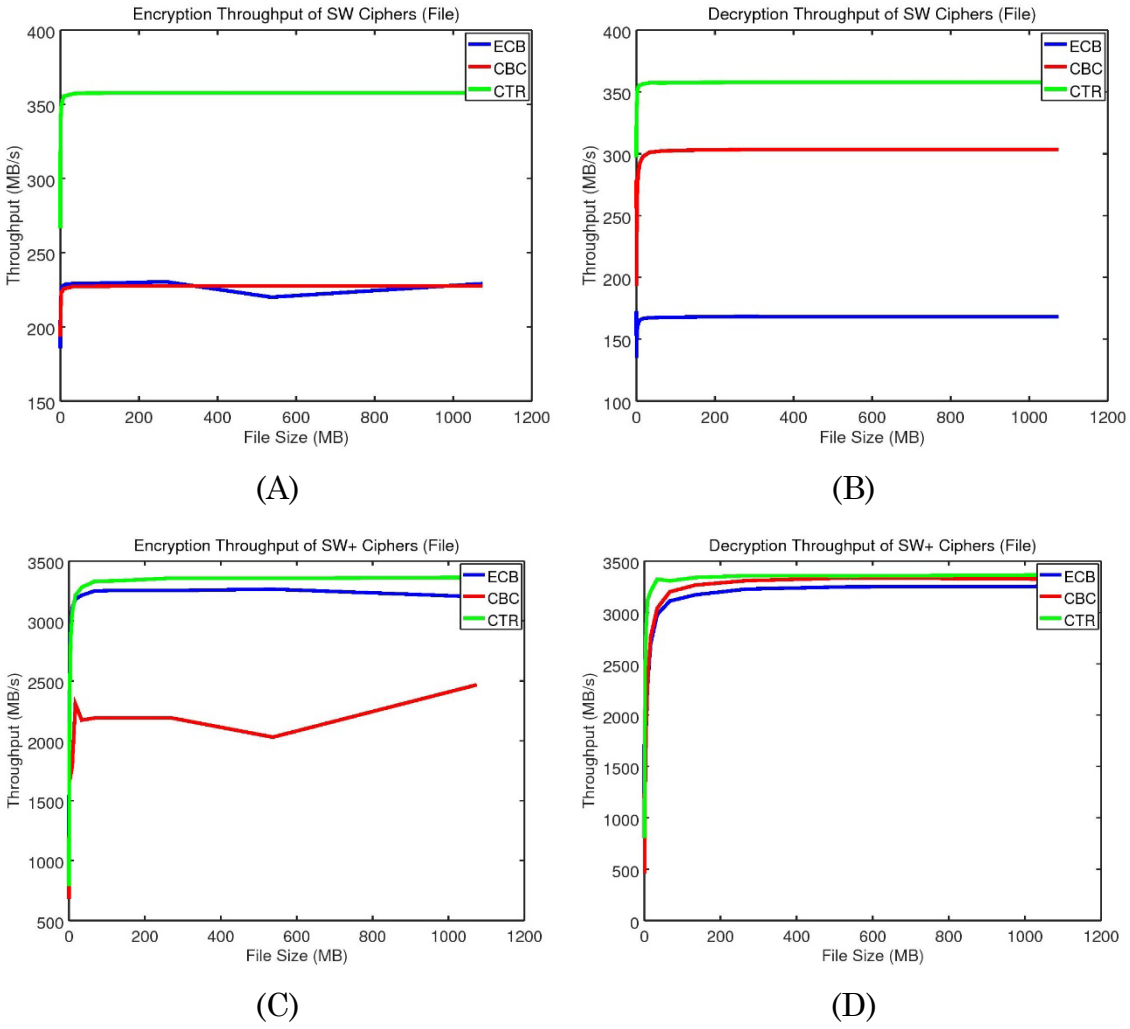
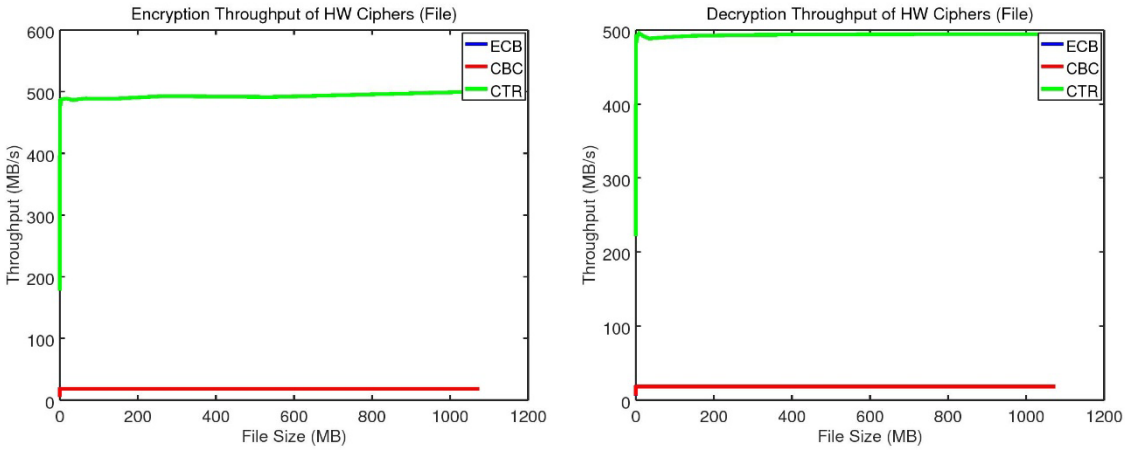
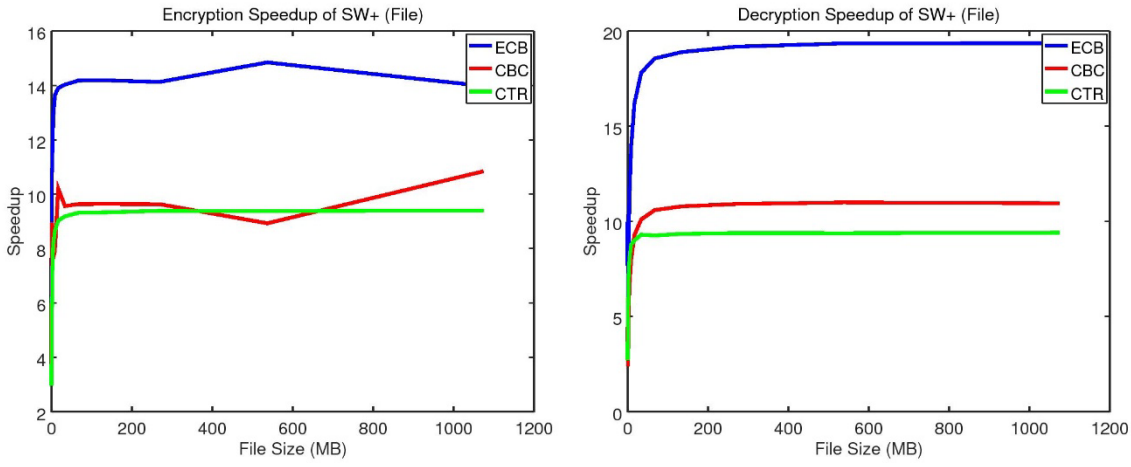


Figure 9.9 (A) Encryption Throughput of Software Ciphers (File), (B) Decryption Throughput of Software Ciphers (File), (C) Encryption Throughput of Software+ Ciphers (File), (D) Decryption Throughput of Software+ Ciphers (File)



(A) (B)

Figure 9.10 (A) Encryption Throughput of Hardware Ciphers (File), (B) Decryption Throughput of Hardware Ciphers (Decryption)



(A) (B)

Figure 9.11 (A) Encryption Speedup of Software+ Ciphers (File), (B) Decryption Speedup of Software+ Ciphers (File)

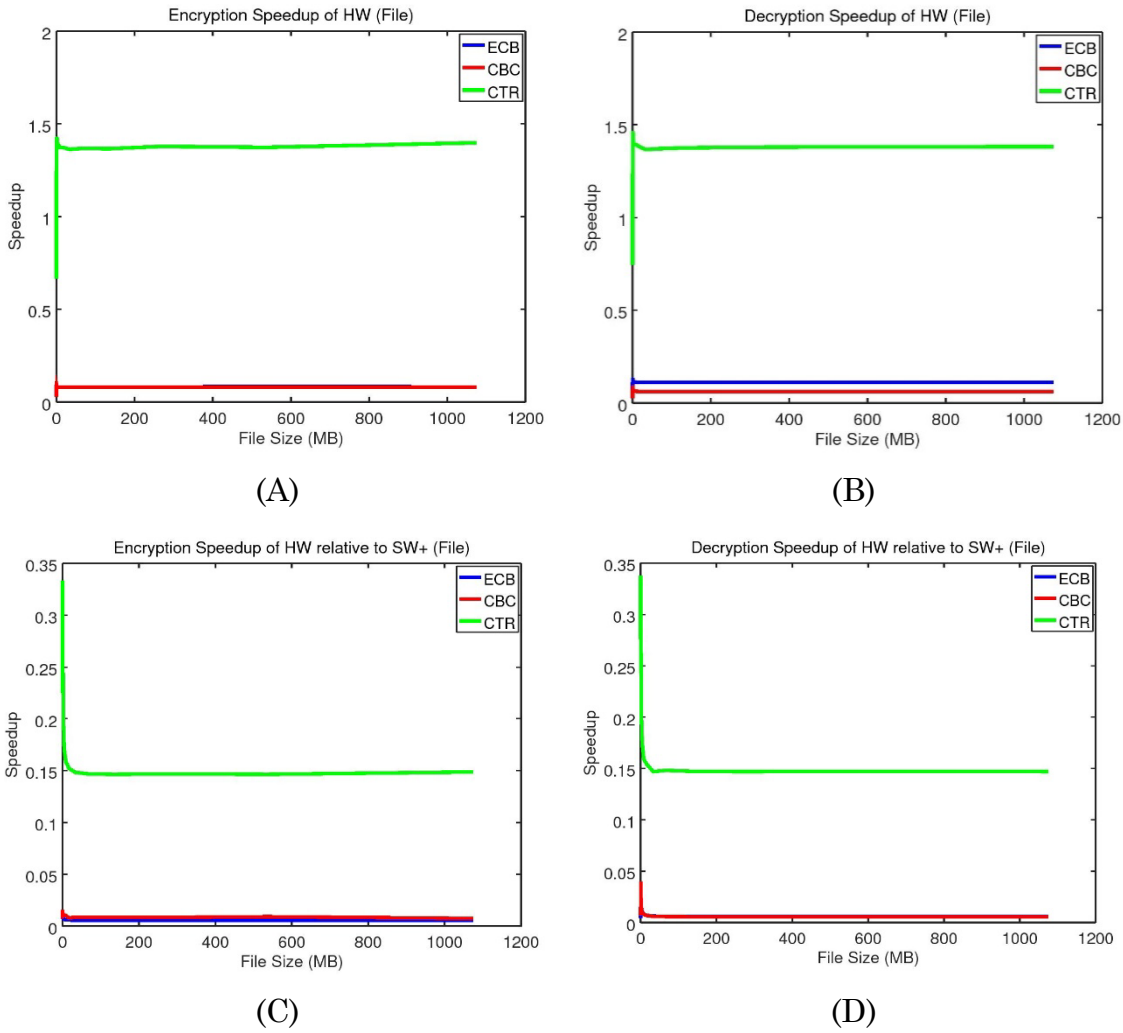


Figure 9.12 (A) Encryption Speedup of Hardware Ciphers (File), (B) Decryption Speedup of Hardware Ciphers (File), (C) Encryption Speedup of Hardware Ciphers relative to Software+ Ciphers (File), (D) Decryption Speedup of Hardware Ciphers relative to Software+ Ciphers (Decryption)

9.2.2 Results for Hardware Sink Test using Maximum Throughput Application

The results for the hardware sink test using the maximum throughput software application are shown in Figure 9.13 through Figure 9.15. The throughputs for the software and hardware ciphers are shown in Figure 9.13 and Figure 9.14. Figure 9.13 (A) and (B) show the software ciphers achieved throughputs of about 225 MB/s for the ECB and CBC ciphers and 325 MB/s for the CTR cipher. It is unclear why the ECB and CBC cipher throughputs in Figure 9.13 (B) differ so much in the decryption process. It could possibly be attributed to the variability in the OS for those runs. Figure 9.13 (C) and (D) show that the software+ ciphers achieved throughputs of about 1 GB/s which is drastically lower than the throughput achieved in the file encryption/decryption test. This can be attributed to the fact that the data is sent to the hardware sink via the PCIe bus after OpenSSL performs the cipher operations on the data; thus, showing that data transfer to the FPGA is the limiting factor for the throughput for this implementation. The encryption process shows a decreased throughput for the CBC cipher which could possibly be OS variability as seen in the software cipher throughput plots. Lastly, Figure 9.14 shows that the hardware ciphers achieved throughputs of about 22 MB/s for the ECB and CBC ciphers and 500 MB/s for the CTR cipher which is the same as the file encryption/decryption test.

Figure 9.15 (A) and (B) show the speedups of the hardware ciphers relative to the software ciphers for encryption and decryption, respectively. The ECB and CBC ciphers have a speedup of about 0.1x and the CTR cipher has a speedup of about 1.5x which are similar to the file encryption/decryption test. Figure 9.15 (C) and (D)

show the speedups of the hardware ciphers relative to the software+ ciphers. The plots show speedups of 0.02x for the ECB and CBC ciphers and 0.5x for the CTR cipher. It is expected that the ECB and CBC software+ ciphers have a much greater improvement over the hardware accelerated ECB and CBC ciphers than for the CTR cipher since the CTR cipher in the FPGA has much greater throughput.

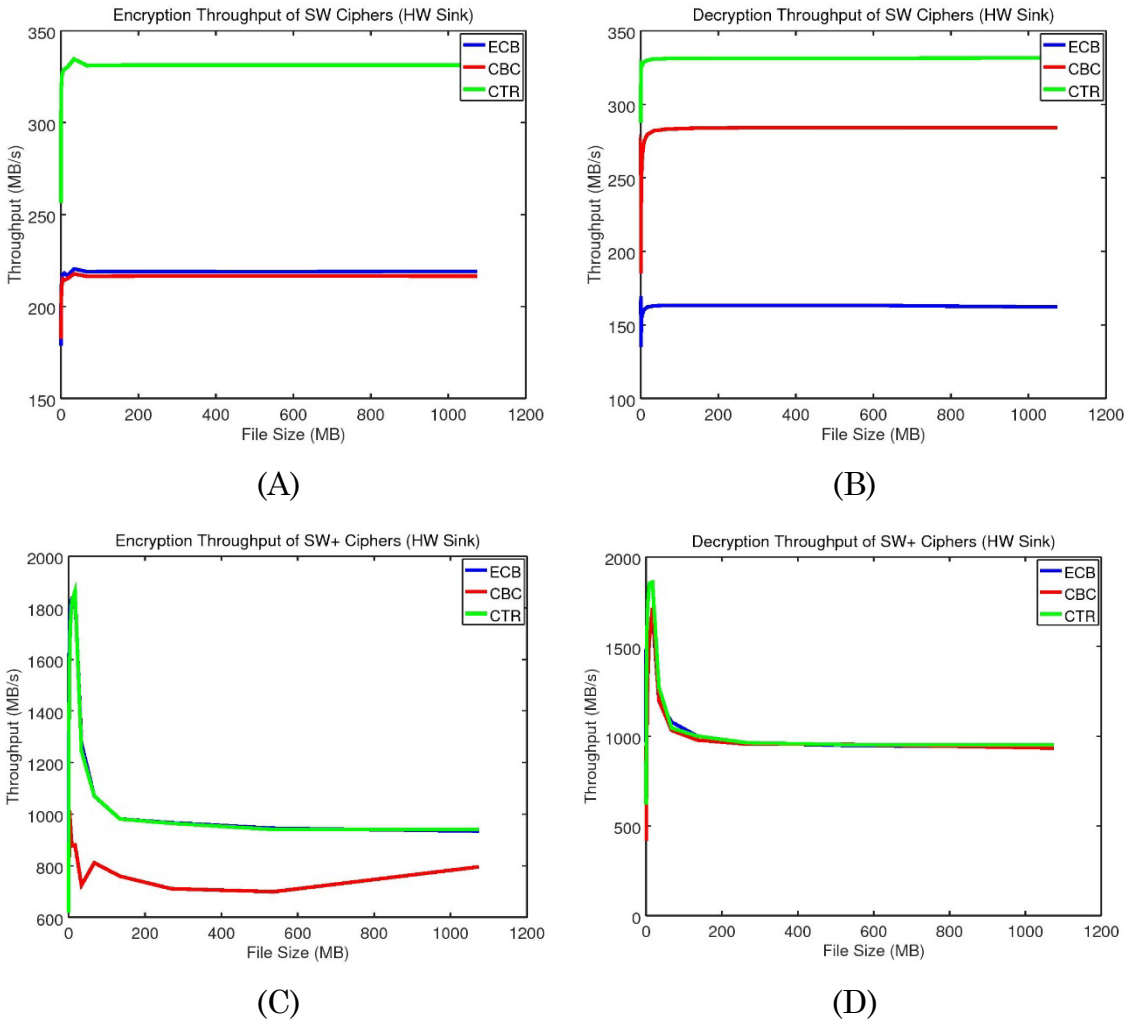
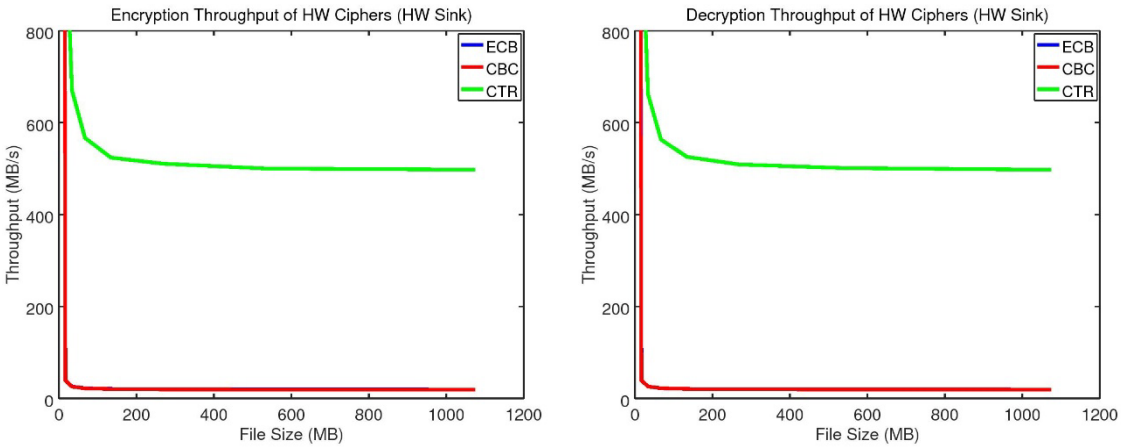


Figure 9.13 (A) Encryption Throughput of Software Ciphers (HW Sink), (B) Decryption Throughput of Software Ciphers (HW Sink), (C) Encryption Throughput of Software+ Ciphers (HW Sink), (D) Decryption Throughput of Software+ Ciphers (HW Sink)



(A) (B)

Figure 9.14 (A) Encryption Throughput of Hardware Ciphers (HW Sink), (B) Decryption Throughput of Hardware Ciphers (HW Sink)

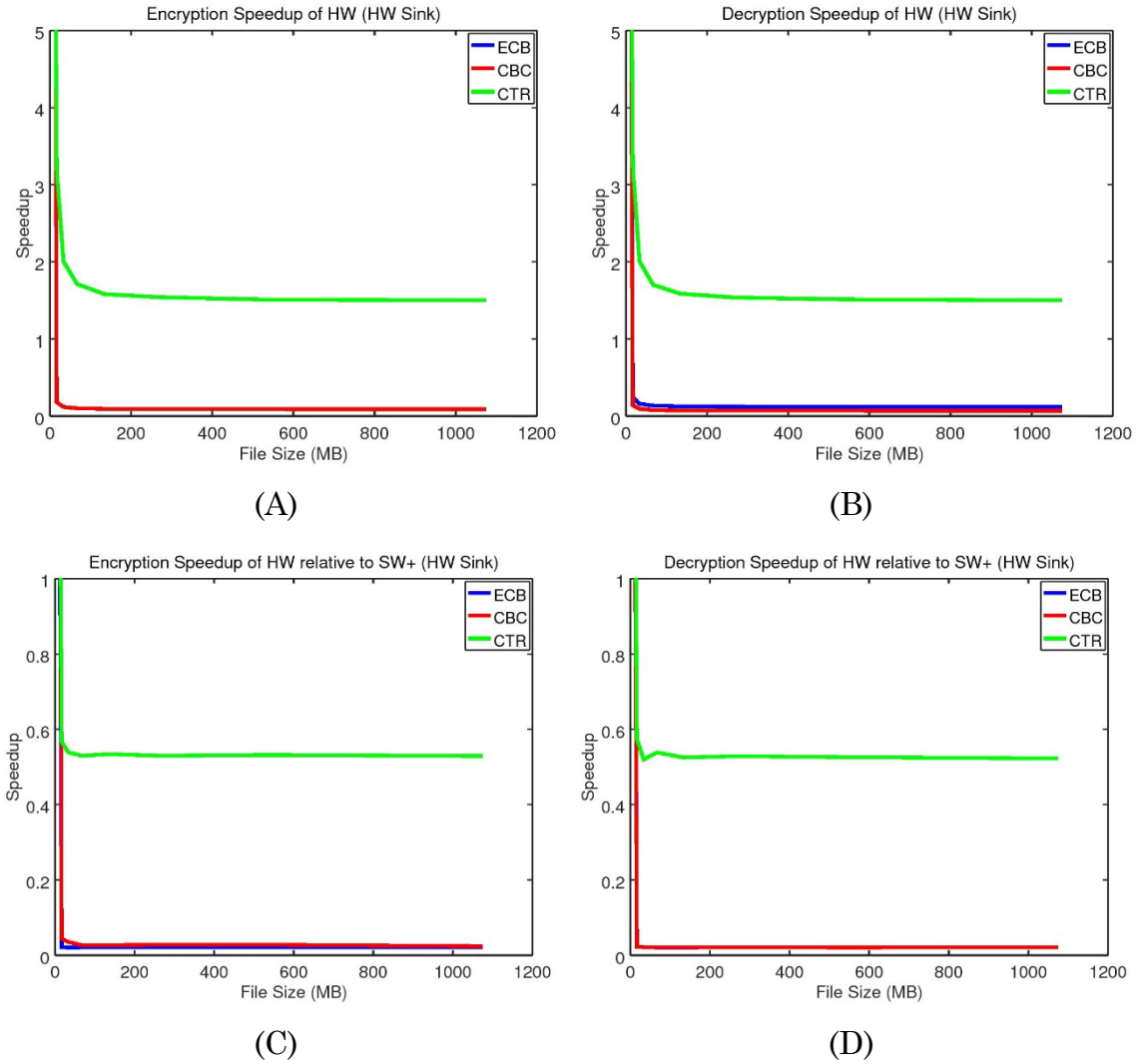


Figure 9.15 (A) Encryption Speedup of Hardware Ciphers (HW Sink), (B) Decryption Speedup of Hardware Ciphers (HW Sink), (C) Encryption Speedup of Hardware Ciphers relative to Software+ Ciphers (HW Sink), (D) Decryption Speedup of Hardware Ciphers relative to Software+ Ciphers (HW Sink)

9.2.3 Results for File Encryption/Decryption Test using OpenSSL Extension Application

The results for the file encryption/decryption test for the OpenSSL extension software application are shown in Figure 9.16 through Figure 9.19. Figure 9.16 (A) and (B) show the throughputs of the software ciphers for encryption and decryption, respectively, to be about 225 MB/s and 300 MB/s for the ECB and CBC ciphers. The CTR cipher has a throughput of 350 MB/s for both operations. Figure 9.16 (C) and (D) show the throughputs for the software+ ciphers. These ciphers achieved throughputs for encryption of about 3.25 GB/s for the ECB and CTR ciphers and about 2.25 GB/s for CBC. For decryption, all three ciphers achieved throughputs of about 3.25 GB/s. Finally, the throughputs of the hardware ciphers are shown in Figure 9.17. The CTR cipher has a throughput of about 400 MB/s; whereas, the ECB and CBC ciphers only have throughputs of about 20 MB/s.

Figure 9.18 shows the speedups achieved by the software+ ciphers relative to the software ciphers. The plots show that for encryption the ECB cipher achieved a speedup of about 14x and the CBC and CTR cipher have a speedup of about 10x. For decryption, all three ciphers have a speedup between 9x and 11x. Figure 9.19 (A) and (B) show the speedups of the hardware ciphers relative to the software ciphers. It can be seen that the CTR cipher is the only cipher to achieve a positive speedup, but it is still very minimal at about 1.1x. The ECB and CBC ciphers both achieved a speedup of about 0.1x. The hardware ciphers did not perform any better than the software ciphers because of the latencies involved with transferring data across the PCIe bus and also due to the fast file I/O operations of the workstation which causes the DMA buffers to fill up causing the test application to spend more time in the

Xillybus write function. As expected, the performance of the hardware ciphers are even more diminished compared to the software+ ciphers. The speedups of the hardware ciphers relative to the software+ cipher is shown in Figure 9.19 (C) and (D). The speedups for encryption and decryption are about 0.005x for the ECB and CBC ciphers and about 0.125x for the CTR cipher.

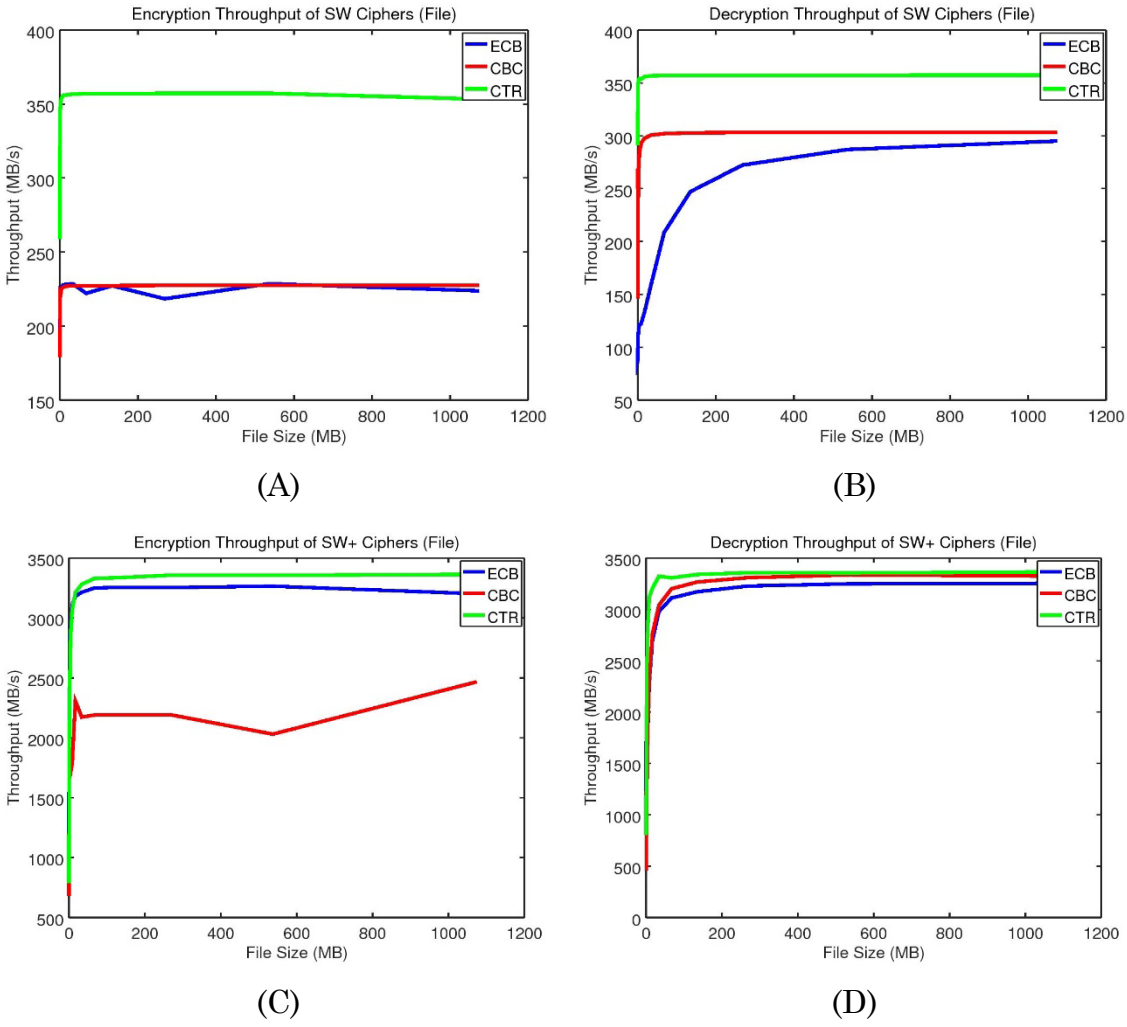
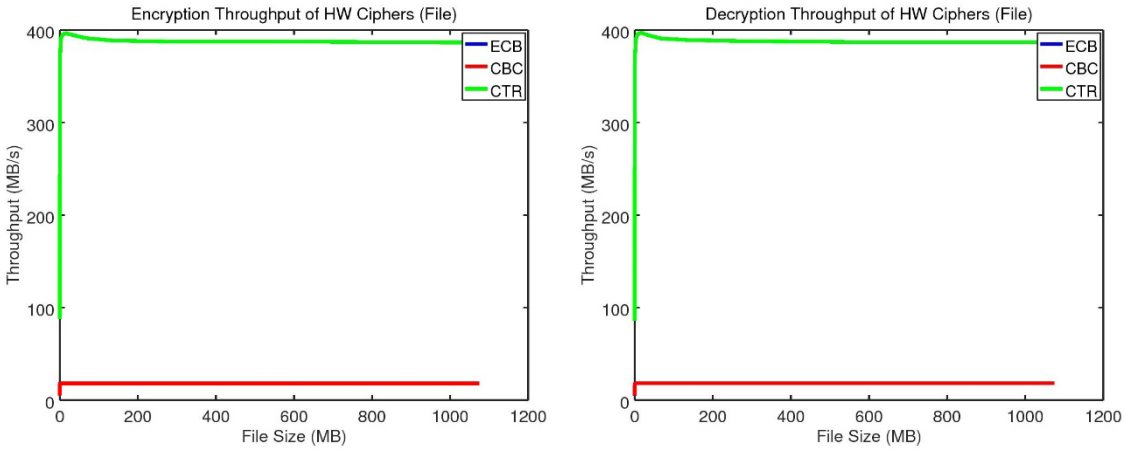
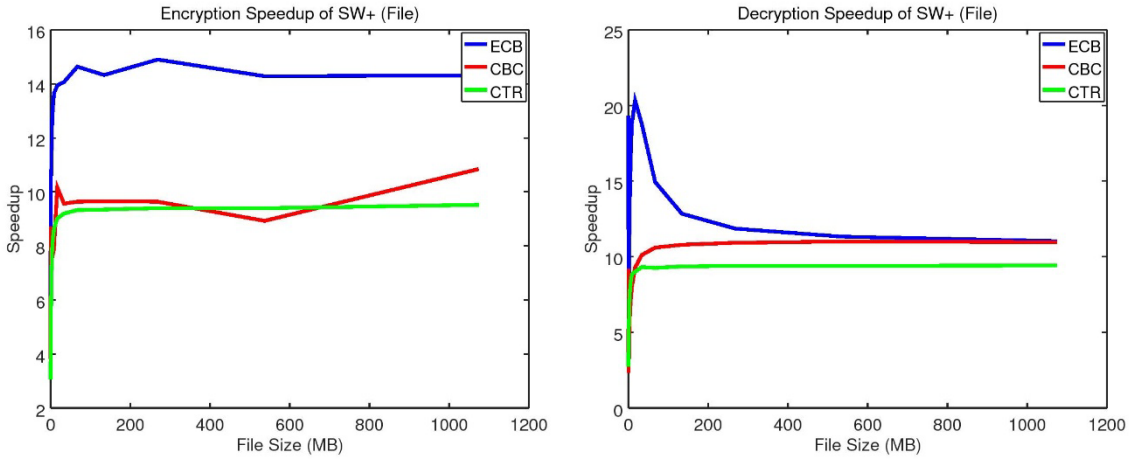


Figure 9.16 (A) Encryption Throughput of Software Ciphers (File), (B) Decryption Throughput of Software Ciphers (File), (C) Encryption Throughput of Software+ Ciphers (File), (D) Decryption Throughput of Software+ Ciphers (File)



(A) (B)

Figure 9.17 (A) Encryption Throughput of Hardware Ciphers (File), (B) Decryption Throughput of Hardware Ciphers (File)



(A) (B)

Figure 9.18 (A) Encryption Speedup of Software+ Ciphers (File), (B) Decryption Speedup of Software+ Ciphers (File)

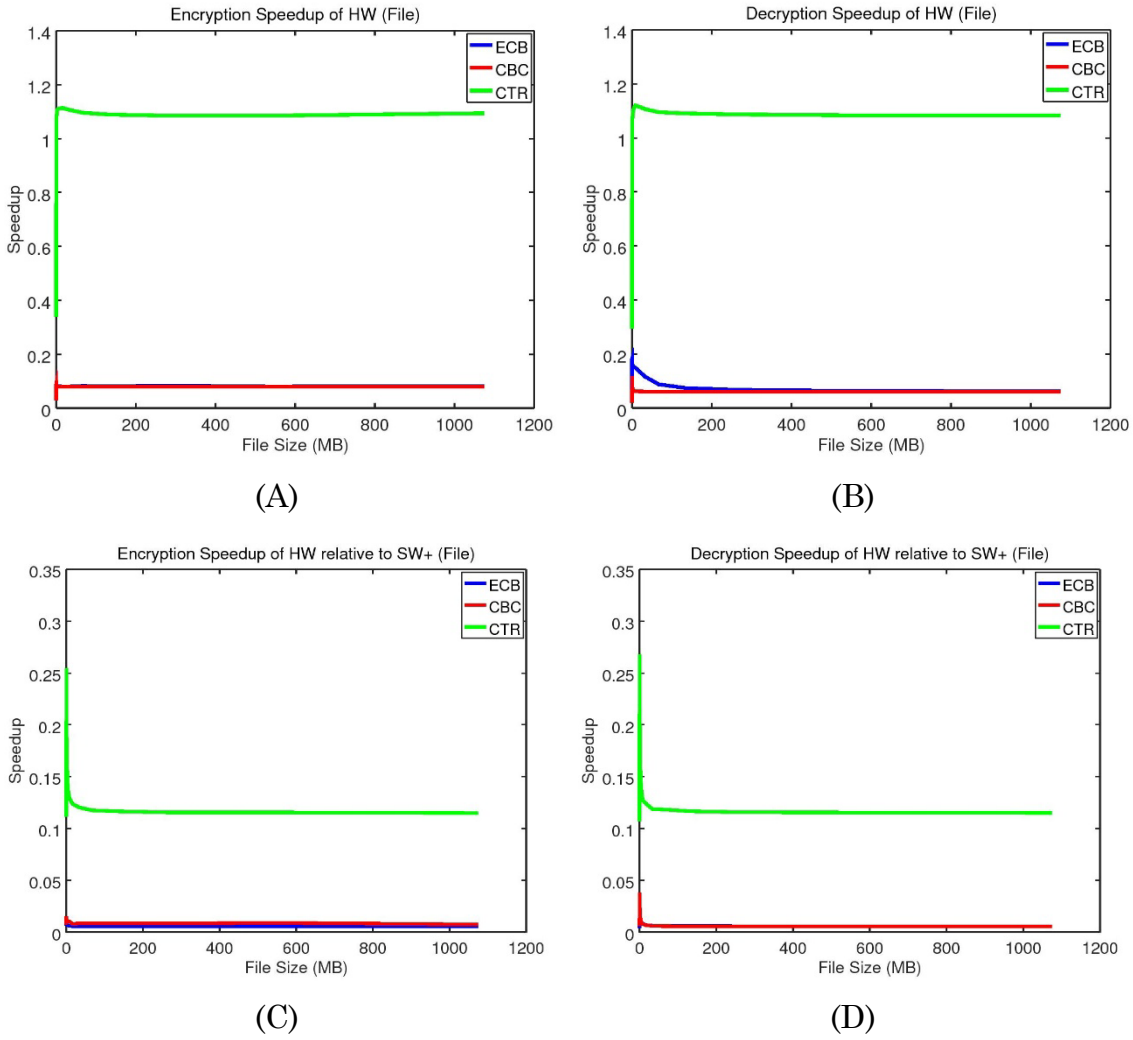


Figure 9.19 (A) Encryption Speedup of Hardware Ciphers (File), (B) Decryption Speedup of Hardware Ciphers (File), (C) Encryption Speedup of Hardware Ciphers relative to Software+ Ciphers (File), (D) Decryption Speedup of Hardware Ciphers relative to Software+ Ciphers (File)

9.2.4 Results for Hardware Sink Test using OpenSSL Extension Application

The results for the hardware sink test using the OpenSSL extension software application are shown in Figure 9.20 through Figure 9.22. Figure 9.20 (A) and (B) show the throughputs of the software ciphers which turned out to be roughly the same as for the file encryption/decryption test except a little slower due to the added step of sending the data to the hardware sink. The throughputs for the ECB and CBC ciphers are about 215 MB/s and 275 MB/s for encryption and decryption, respectively. The CTR cipher has a throughput of 325 MB/s for both operations. Figure 9.20 (C) and (D) show the throughputs for the software+ ciphers. The software+ ciphers achieve throughputs for encryption and decryption of about 1 GB/s; however, the CBC cipher only has a throughput of 800 MB/s for encryption. These throughputs are much less than for the file encryption/decryption test which is due to having to send the data across the bus to the hardware sink. The DMA buffers and PCIe bus become the bottleneck for this test and cause the throughput for the software+ ciphers to decrease dramatically. Lastly, the throughputs of the hardware ciphers are shown in Figure 9.21. The CTR cipher has a throughput of about 580 MB/s; whereas, the ECB and CBC ciphers only have throughputs of about 20 MB/s.

Figure 9.22 (A) and (B) show the speedups achieved by the hardware ciphers relative to the software ciphers for encryption and decryption, respectively. The plots show that the ECB and CBC ciphers have a speedup of about 0.1x and the CTR cipher has a speedup of about 1.5x. The speedups of the hardware ciphers relative to the software+ ciphers are shown in Figure 9.22 (C) and (D). The ECB and CBC ciphers have a speedup of 0.02x and the CTR cipher has a speedup of 0.5x. This

illustrates that the cost of having to transfer data across the PCIe bus is very costly to performance over executing the operations locally to the processor with AESNI.

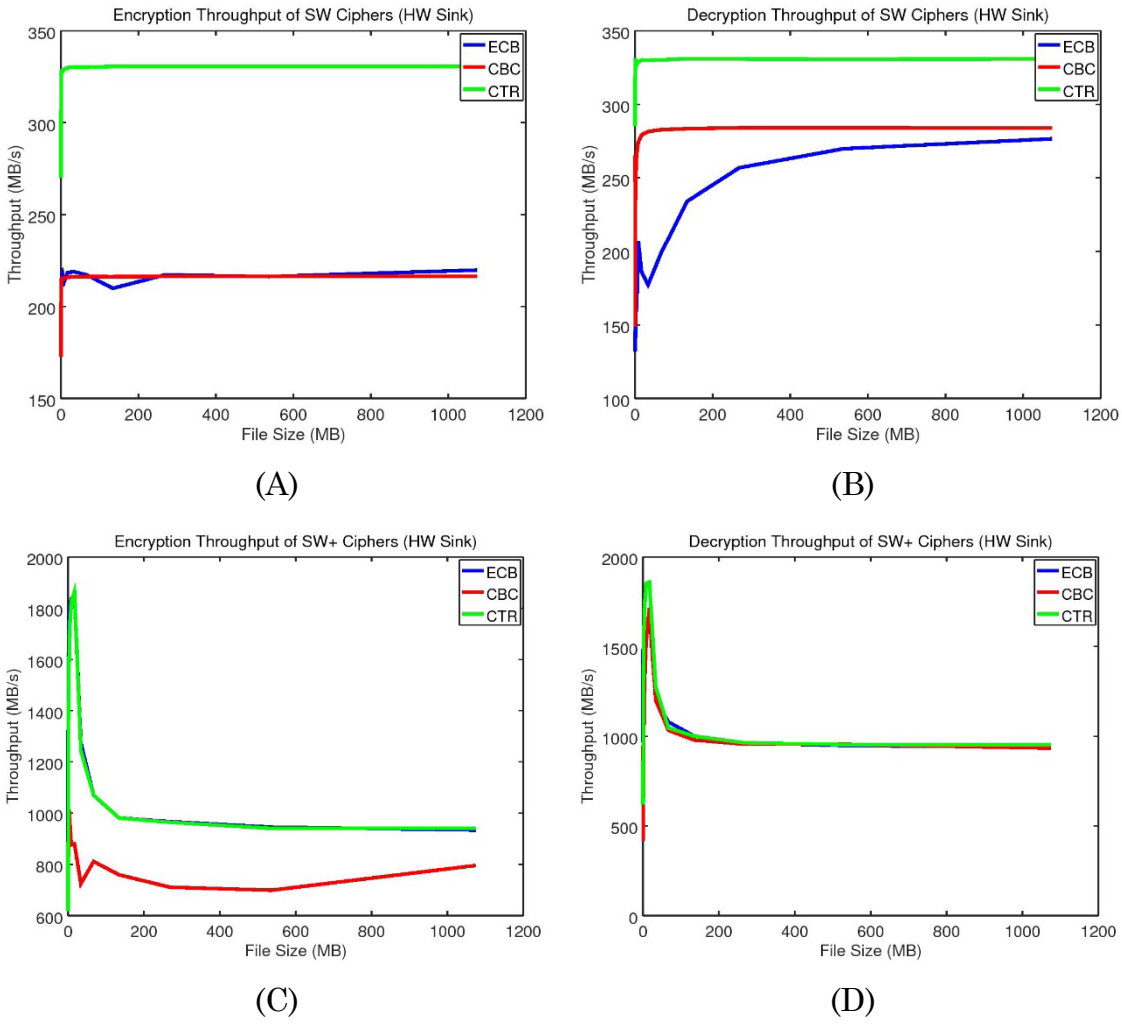
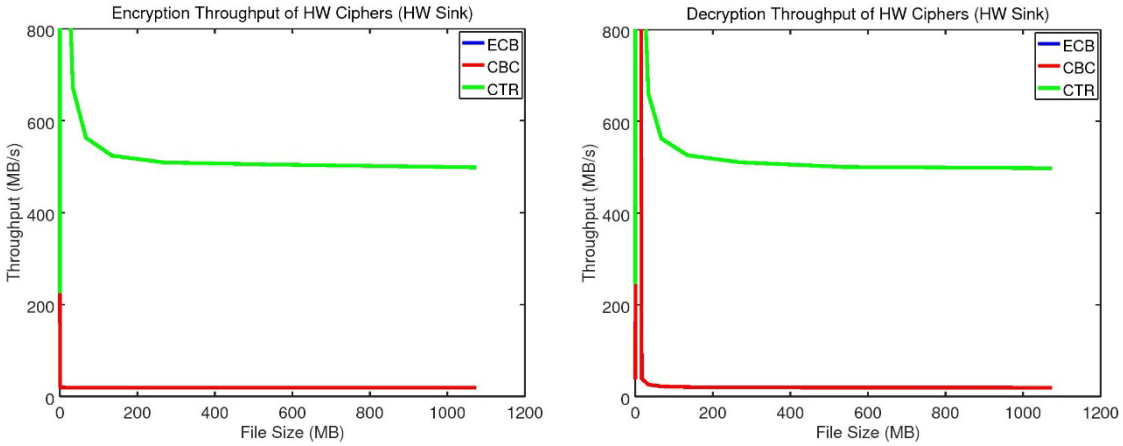


Figure 9.20 (A) Encryption Throughput of Software Ciphers (HW Sink), (B) Decryption Throughput of Software Ciphers (HW Sink), (C) Encryption Throughput of Software+ Ciphers (HW Sink), (D) Decryption Throughput of Software+ Ciphers (HW Sink)



(A) (B)

Figure 9.21 (A) Encryption Throughput of Hardware Ciphers (HW Sink), (B) Decryption Throughput of Hardware Ciphers (HW Sink)

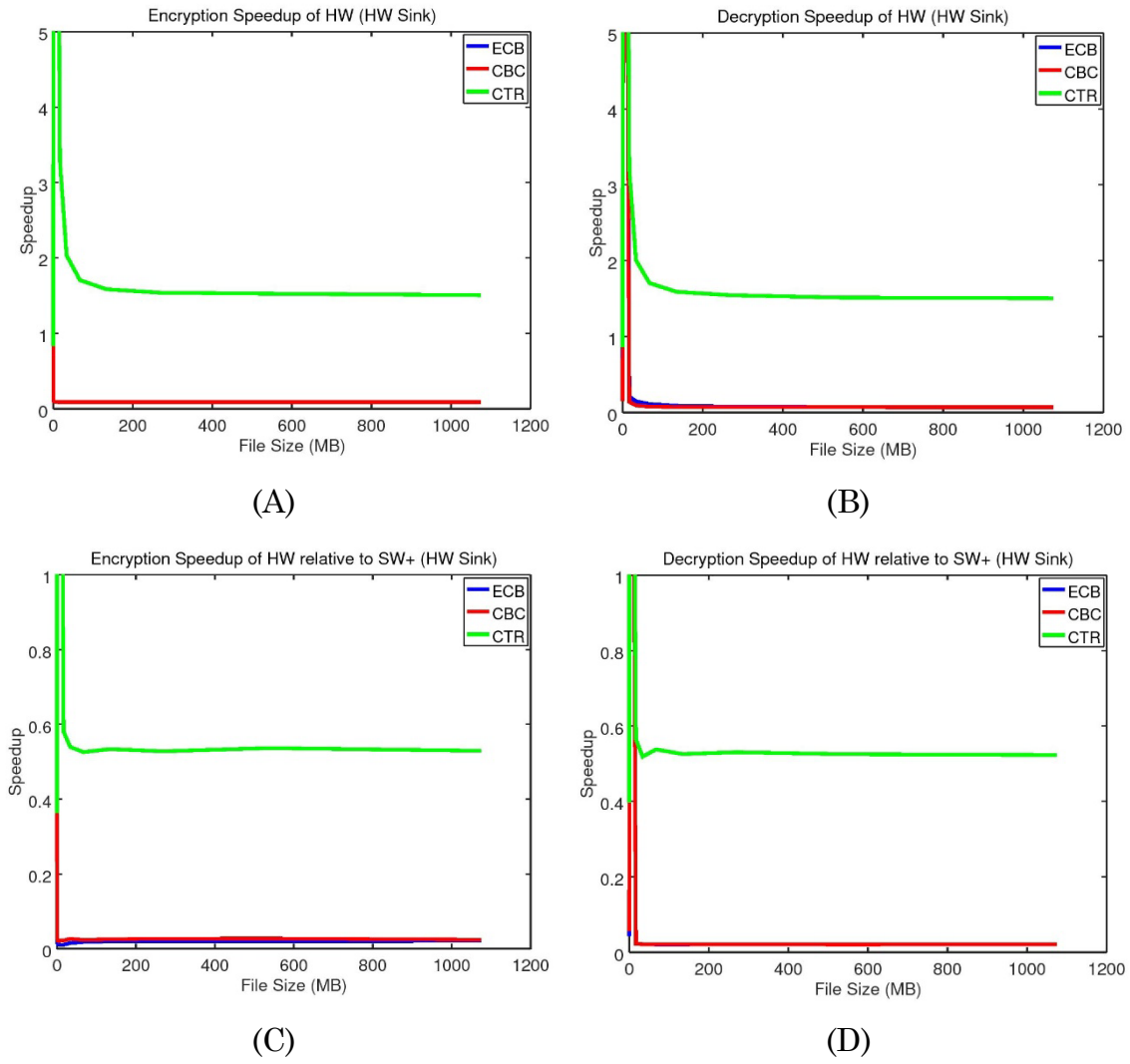


Figure 9.22 (A) Encryption Speedup of Hardware Ciphers (HW Sink), (B) Decryption Speedup of Hardware Ciphers (HW Sink), (C) Encryption Speedup of Hardware Ciphers relative to Software+ Ciphers (HW Sink), (C) Decryption Speedup of Hardware Ciphers relative to Software+ Ciphers (HW Sink)

CHAPTER 10

FUTURE WORK AND CONCLUSIONS

For this thesis, a heterogeneous Zynq-7000 APSoC is used to implement hardware accelerated ciphers for the ECB, CBC, and CTR cipher modes. The performance of the hardware accelerated ciphers are compared to the software ciphers from the OpenSSL cryptographic library using metrics of execution time, throughput, and speedup. Performance evaluations are completed for both embedded and workstation computing environments. The Zedboard development platform is used for implementing the hardware accelerated ciphers in an embedded environment. The ARM processor of the Zynq-7000 is used on the Zedboard as the host processor. The ZC706 development board is used for implementing the hardware accelerated ciphers in a workstation environment. Only the FPGA fabric of the Zynq-7000 is used on the ZC706 board and the Intel processor of the workstation is the host processor.

Two different software applications are used for testing the performance of the software and hardware ciphers. One application is used for attempting to maximize the throughput of the hardware accelerated ciphers while the second application is used for simplifying the programming interface for accessing both the software and hardware ciphers. There are also two different tests that are run by each software application. The first is just a basic file encryption/decryption where data is either encrypted or decrypted from an input file and the resultant data is written to an output file. This test simulates the case of protecting data-at-rest on a

computing platform. The second test is a hardware sink test which simulates the case of protecting data that will be transmitted to some platform external to the host platform.

The results of this thesis show that the hardware accelerated ciphers on the embedded platform experienced significant performance improvements over the software ciphers; however, the same hardware accelerated ciphers did not achieve significant performance improvements on the workstation platform. On the embedded Zedboard platform, the ECB and CBC ciphers achieved a maximum speedup of about 30x with a maximum throughput of about 450 MB/s. The CTR cipher achieved a maximum speedup of about 28x compared to its software counterpart with a maximum throughput of about 500 MB/s. On the workstation platform the hardware accelerated ciphers do not offer any performance improvement. Thus, the ECB and CBC ciphers achieved a maximum speedup of about 0.1x relative to the software ciphers and about 0.02x relative to the software+ ciphers. The ECB and CBC ciphers achieved a maximum throughput of about 22 MB/s. The hardware-accelerated CTR cipher achieved a maximum speedup of about 1.5x relative to the corresponding software CTR cipher and 0.5x versus the software+ CTR cipher. The CTR cipher achieved a maximum throughput of about 500 MB/s.

The overall conclusion that can be made from the work accomplished in this thesis is that the hardware accelerated ciphers are much more beneficial when used in an embedded environment than in a workstation environment. Embedded computing platforms are usually resource limited and cannot afford to have a very powerful processor due to power limitations; therefore, it becomes very advantageous to offload computationally intensive operations, such as cryptographic

operations, to an alternative computing platform that could perform such operations faster while consuming less power. On the other hand, the processors in a workstation environment, along with modern cryptographic hardware extensions, perform these operations at a very high level which makes it difficult to use off-chip alternative platforms for these operations and achieve significant performance improvements.

Some future efforts that could emanate from the work accomplished in this thesis could include trying to modify the ciphers in the FPGA design to improve performance. This could involve either using other open-source IP core(s) that have better performance statistics through higher operational clock rates, shorter/longer pipelines, or better optimized algorithms. A custom AES block cipher core could be developed to meet new/improved design and performance criteria if no open-source IP cores are available to meet the criteria. One could also explore the possibility of implementing the ECB cipher with a fully pipelined AES core which could elevate its performance to be comparable to the CTR cipher in the design used in this thesis. Another possible option for future work could be to experiment with altering the parameters in the Xillybus IP Core Factory [12] to increase the size of the DMA buffers in RAM and DMA FIFOs on the FPGA to try and improve the effective throughput of the bus architectures. Other future work could involve using the ZC706 development board with the ARM processors as the host for the design and utilizing the larger FPGA for allocating more resources to the ciphers for a potential performance improvement. Lastly, work could be done to improve the fidelity of the timing measurements with the experimental setup in order to remove any affects the file I/O operations have on the measured execution times for the ciphers.

REFERENCES

[1] "Software and Hardware Cipher Attacks." [Online]. Available: https://www.experts-exchange.com/articles/12460/Cryptanalysis-and-Attacks.html. [Accessed: 20-Jul-2017].

[2] "Software vs. Hardware Security." [Online]. Available: https://www.infosecurity-magazine.com/magazine-features/tales-crypt-hardware-software/. [Accessed: 20-Oct-2017].

[3] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners*. Heidelberg; New York: Springer, 2010.

[4] "OpenSSL: Cryptography and SSL/TLS Toolkit." [Online]. Available: https://www.openssl.org/. [Accessed: 15-Dec-2016].

[5] J. Viega, M. Messier, and P. Chandra, *Network security with OpenSSL*, 1st ed. Sebastopol, CA: O'Reilly, 2002.

[6] Xilinx, "Zynq." [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf. [Accessed: 15-Dec-2016].

[7] "Xillybus: An FPGA IP core for easy DMA over PCIe with Windows and Linux." [Online]. Available: http://xillybus.com/. [Accessed: 01-Oct-2016].

[8] "Xillybus: Principle of Operation." [Online]. Available: http://xillybus.com/doc/xilinx-pcie-principle-of-operation. [Accessed: 15-Dec-2016].

[9] "Xillybus Bandwidth." [Online]. Available: http://xillybus.com/doc/xillybus-bandwidth. [Accessed: 05-Dec-2016].

[10] E. Billauer, "Xillybus Overview." [Online]. Available: https://www.kernel.org/doc/Documentation/xillybus.txt. [Accessed: 04-Dec-2016].

[11] "Xillybus IP Core Factory Info." [Online]. Available: http://xillybus.com/custom-ip-factory. [Accessed: 03-Oct-2016].

[12] "Xillybus IP Core Factory." [Online]. Available: http://xillybus.com/ipfactory/. [Accessed: 03-Oct-2016].

[13] A. Hodjat, D. D. Hwang, B. Lai, K. Tiri, and I. Verbauwhede, "A 3.84 gbits/s AES crypto coprocessor with modes of operation in a 0.18-µm CMOS technology," in *Proceedings of the 15th ACM Great Lakes symposium on VLSI*, 2005, pp. 60–63.

[14] A. Hodjat and I. Verbauwhede, "Interfacing a high speed crypto accelerator to an embedded CPU," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, 2004, vol. 1, pp. 488–492.

[15] S. Baskaran and P. Rajalakshmi, "Hardware-software co-design of AES on FPGA," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, 2012, pp. 1118–1122.

[16] C. Pedraza, J. Castillo, J. I. Martínez, P. Huerta, and C. S. de La Lama, "Self-reconfigurable secure file system for embedded Linux," *IET Comput. Digit. Tech.*, vol. 2, no. 6, pp. 461–470, 2008.

[17] V. P. Nambiar, M. Khalil-Hani, and M. M. A. Zabidi, "Accelerating the AES encryption function in OpenSSL for embedded systems," in *Proceedings of the International Conference on Electronic Design, ICED 2008*, 2008, pp. 1–5.

[18] A. Hodjat, P. Schaumont, and I. Verbauwhede, "Architectural design features of a programmable high throughput AES coprocessor," in *Proceedings of the International Conference on Information Technology: Coding and Computing*, 2004, vol. Vol. 2, pp. 498–502.

[19] A. Irwansyah, V. P. Nambiar, and M. Khalil-Hani, “An AES Tightly Coupled Hardware Accelerator in an FPGA-based Embedded Processor Core,” in *Proceedings of the International Conference on Computer Engineering and Technology ICCET ’09*, 2009, pp. 521–525.

[20] R. Cowart, D. Coe, J. Kulick, and A. Milenković, “An Implementation and Experimental Evaluation of Hardware Accelerated Ciphers in All-Programmable SoCs,” 2017, pp. 34–41.

[21] “Xillybus Host Programming Guide Linux.” [Online]. Available: http://xillybus.com/downloads/doc/xillybus_host_programming_guide_linux.pdf. [Accessed: 10-Jan-2017].

[22] “AXI Reference Guide.” [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf. [Accessed: 30-Jul-2017].

[23] “Xillybus FPGA Interface.” [Online]. Available: http://xillybus.com/downloads/doc/xillybus_fpga_api.pdf. [Accessed: 02-Nov-2016].

[24] “Non-Pipelined AES IP Core.” [Online]. Available: https://github.com/secworks/aes. [Accessed: 10-Dec-2016].

[25] “Pipelined AES IP Core.” [Online]. Available: https://opencores.org/project,tiny_aes. [Accessed: 15-Dec-2016].

[26] “Intel Xeon Processor.” [Online]. Available: [https://www.intel.com/content/www/us/en/products/processors/xeon/e5-processors/e5-4655-v4.html. [Accessed: 10-Aug-2017].

[27] “PCIe Bus.” [Online]. Available: http://computer.howstuffworks.com/pci-express.htm. [Accessed: 08-Aug-2017].

[28] “Zedboard.” [Online]. Available: http://zedboard.org/. [Accessed: 24-Sep-2016].

[29] “Zedboard Image.” [Online]. Available: http://zedboard.org/product/zedboard. [Accessed: 20-Jul-2017].

[30] “Xillinux: A Linux distribution for Zedboard, ZyBo, MicroZed and SocKit.” [Online]. Available: http://xillybus.com/xillinux. [Accessed: 15-Dec-2016].

[31] “Getting started with Xillinux for Zynq-7000 EPP.” [Online]. Available: http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf. [Accessed: 15-Dec-2016].

[32] “ZC706 Evaluation Board.” [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf. [Accessed: 20-Jun-2017].

[33] “ZC706 Image.” [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/zc706/2015_4/ug961-zc706-GSG.pdf. [Accessed: 24-Jul-2016].