An Implementation and Experimental Evaluation of Hardware Accelerated Ciphers in All-Programmable SoCs

R. Cowart
Electrical and Computer Engineering,
The University of Alabama in Huntsville;
301 Sparkman Dr., Huntsville, AL 35899
rac0014@uah.edu

D. Coe
Electrical and Computer Engineering,
The University of Alabama in Huntsville;
301 Sparkman Dr., Huntsville, AL 35899
coed@uah.edu

J. Kulick
Electrical and Computer Engineering,
The University of Alabama in Huntsville;
301 Sparkman Dr., Huntsville, AL 35899
kulickj@uah.edu

A. Milenković
Electrical and Computer Engineering,
The University of Alabama in Huntsville;
301 Sparkman Dr., Huntsville, AL 35899
milenka@uah.edu

ABSTRACT

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. In this paper, we explore the performance benefits of an AES hardware accelerator versus the software implementation for multiple cipher modes on the Zynq 7000 All-Programmable System-on-a-Chip (SoC). The accelerator is implemented on the FPGA fabric of the SoC and utilizes DMA for interfacing to the CPU. File encryption and decryption of varying file sizes are used as the workload, with execution time and throughput as the metrics for comparing the performance of the hardware and software implementations. The performance evaluations show that the accelerated AES operations achieve a speedup of 7 times relative to its software implementation and throughput upwards of 350 MB/s for the counter cipher mode, and modest improvements for other cipher modes.

CCS CONCEPTS

• **Security and privacy** → *Block and stream ciphers;* • **Security and privacy** → *Hardware security implementation;* • **Hardware** → *Hardware accelerators*

KEYWORDS

IP cores, Hardware acceleration, FPGAs, OpenSSL, AES

ACM Reference format:

R. Cowart, D. Coe, J. Kulick, and A. Milenkovic. 2017. In *Proceedings of ACM Southeast Conf., Kennesaw, Georgia USA, April 2017*, 8 pages. DOI: http://dx.doi.org/10.1145/3077286.3077297

ACM SE '17, April 13-15, 2017, Kennesaw, GA, USA
 ISBN 978-1-4503-5024-2/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3077286.3077297

1 INTRODUCTION

The Digital Age has seen an ever increasing amount of data stored and transmitted across publicly accessed mediums. 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 encryption algorithms designed each with the goal of securing the data. The Advanced Encryption Standard (AES) 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. 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.

Recently, the computing industry has 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. One type of heterogeneous computing systems are the all-programmable Systems-on-a-Chip (SoCs), which contain 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 hard processor system and the hardware programmability of the FPGA fabric within a single chip. 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 efficiently.

This paper explores the performance benefits of implementing AES encryption and decryption for multiple cipher modes in hardware on the Zynq-7000 All-Programmable SoC. The AES cipher modes implemented are the Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Counter (CTR) mode. The ciphers are implemented in both software and hardware and the performance of each are compared to analyze the improvements achieved through the hardware accelerated ciphers. The software implementation of the ciphers is accomplished through the

use of the OpenSSL cryptography library. The OpenSSL library is highly optimized and widely used due to its user-friendly application programming interface and performance-minded implementation. The hardware accelerated implementation of the AES ciphers is achieved through the use of open source cores available from OpenCores [1] and Secworks [2] that interface to the hard processor system through the use of the Xillybus IP core [3] for data streaming.

Data files of different sizes were encrypted and decrypted separately using the software and hardware implementations for all three cipher modes. The performance metrics used to evaluate both implementations are execution times and data throughput of the cipher modes. The results of the performance analysis demonstrate that the hardware accelerated AES algorithms for all three ciphers perform up to 7 times better than the corresponding software implementations.

The remaining sections of this paper are organized as follows. Section 2 discusses related work. Section 3 discusses the ciphers that are implemented and analyzed. Section 4 gives the specifics of the hardware implementations. Section 5 describes the experimental environment and Section 6 gives the main results of the experimental evaluation. Finally, Section 7 gives the concluding remarks and discusses future work.

2 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.

Some of the prior AES coprocessor designs used softcore processors in an FPGA for interfacing to an AES hardware core. Hodjat et. al. [4] [5] used the LEON soft processor in the ThumbPod SOC to achieve a maximum throughput of 3.84 Gbit/s. Baskaran et. al. [6] implemented the AES operations 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 microprocessor and AES implementations with custom software executed on the microprocessor.

A couple other designs also used softcore processors, but instead of using custom software for cipher control certain cryptographic software libraries were extended to target the hardware accelerators as opposed to performing all the operations in software. Pedraza et. al. [7] ran Linux on a PowerPC softcore processor using a Virtex II FPGA and extended the functionality of the CryptoAPI Linux cryptographic library to utilize the AES hardware accelerators with a maximum throughput of 100 MB/s. Nambiar et. al. [8] extended the encryption function of the OpenSSL cryptographic library on the NIOS II softcore microprocessor running uClinux 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 full software implementation of OpenSSL. Hodjat et. al. [9] interfaced a hard CPU processor to an AES FPGA hardware accelerator for use in VPN and IPSec applications. They were able to achieve a throughput of 3.84 Gbit/s with a power consumption of 86 mW. Irwansyah et. al. extended the instruction set of the Nios II RISC processor to support AES encryption and decryption [10]. The designs in [8] and [10] only implement the base AES encryption and decryption algorithms.

Our work is different than the previously mentioned designs because we are using the Zynq 7000 All Programmable SoC that contains both a hardened dual core ARM processor with an FPGA fabric. The ARM cores use the AES coprocessor by passing data across the AXI data bus. Some of the previous designs only implement the base AES operations; whereas, our design implements fully functional AES ciphers of ECB, CBC, and CTR in the FPGA and requires no additional interaction with the ARM cores aside from the transfer of the raw data for the encryption or decryption. Our design also makes use of Xillybus which is an interface between the ARM cores and the FPGA fabric via either DMA or memory mapped interfaces. Our current design only makes use of the DMA interface of Xillybus which offloads the data transfer process from the CPU; whereas, the previous designs required the CPU to participate in the data transfer. Most of the previous designs used a softcore processor, but our design uses hardened ARM core processors that communicate with the FPGA via a bus architecture as opposed to the FPGA fabric that is used by the softcore processors.

3 BACKGROUND AND METHODOLOGY

The cryptographic algorithm implemented was the Advanced Encryption Standard (AES). AES is one of the most commonly used and trusted cryptographic algorithms for protecting sensitive information for governments, corporations, and individuals alike. The algorithm performs multiple rounds of operations on the raw data based on the bit width of the cipher key. The performance of the hardware and software AES implementations are compared by using file encryption and decryption as a workload. Encryption is the process of encoding the raw data using a private cipher key so that the data cannot be viewed by non-authenticated users. Decryption is the process of decoding the encrypted data using the same private key so that the user can view the raw data. OpenSSL is an open source cryptographic library that implements the AES operations in a highly optimized manner for different instruction set architectures. OpenSSL is used as the baseline software implementation due to its optimized performance and common use across the industry.

3.1 Advanced Encryption Standard (AES)

AES is an industry standard for encrypting electronic data for the purpose of data security and confidentiality. It was developed by two Belgian cryptographers Joan Daemen and Vincent Rijmen and was accepted as a standard in 2001 by the National

Institute of Standards and Technology (NIST) in the U.S. [11] and has quickly become the most widely used cipher due to its security strength and optimized implementations. AES is a symmetric cipher because it uses the same key to encrypt and decrypt the data.

The AES cipher can be used with three different key sizes of 128, 192, and 256 bits. The strength of the cipher increases proportionally with the number of bits in the key. The base data component that the cipher operates on is known as the state matrix which consists of 128 bits of data. The cipher performs a number of operations on the state matrix for a specified number of rounds based on the key size. There are 10, 12, and 14 rounds for key sizes of 128, 192, and 256 bits, respectively. Within each round the cipher performs different layers of operations. Each layer manipulates all 128 bits of the state matrix. The different layers are the Key Addition layer, the Byte Substitution layer, and the Diffusion layer. In the Key Addition layer a 128-bit key that was derived from the original key is XORed to the state matrix. The Byte Substitution layer is just the operation of replacing the current byte with the value from a lookup table using the current byte value as the index to the lookup table. The Diffusion layer performs linear operations on the state matrix. The first operation is the Shift Rows operation which permutes the bytes within each row of the state matrix a different number of times based on the row number. The second is the Mix Columns operation which is a matrix operation that combines blocks of four bytes. The details of the cipher algorithm can be found in the Federal Information Processing Standards document FIPS 197 [11]. The AES algorithm can be used in multiple different cipher modes including Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Counter Mode (CTR) [12]. Each cipher mode uses AES as the encryption algorithm for securing the data. However, each cipher uses the AES algorithm in a slightly different place in its cipher implementation resulting in different cipher characteristics.

3.2 File Encryption/Decryption

Data encryption is the process of securing data by converting it into a cipher code that only the original owner can decode and read. Data decryption is the process of decoding encrypted data to its original form such that it can be read and understood by the accessing party.

To evaluate the effectiveness of the software and hardware implementations of the three ciphers we use file encryption and decryption as a test application. Data files varying from 32 KB to 64 MB are first encrypted and then decrypted using software-only and hardware-accelerated implementations of the ECB, CBC, and CTR ciphers.

The file encryption and decryption is performed within a single C/C++ application which runs on the dual ARM core processor of the Zynq. The application is used for controlling whether the file is encrypted/decrypted via the OpenSSL software implementation or the hardware implementation of the ciphers. The application has multiple command line options for specifying the different parameters of the file encryption/decryption. These command line options include one for specifying whether to execute the cipher in software or use the hardware accelerator, for specifying a security password for cipher key and initialization vector generation, to specify the AES cipher mode, and to specify input/output file names.

If executing fully in software, the file to be encrypted/decrypted is read incrementally and passed into the OpenSSL AES cipher functions which return the final data to be stored in the output file. If the hardware acceleration option is chosen then two separate threads are spawned where one thread reads the input file incrementally and sends the input data into the hardware accelerator; whereas, the second thread is continuously reading the output data from the hardware accelerator and writing the output data to the output file.

The application is verified by using known test vectors and verifying that the output of each cipher mode was correct. The decrypted files were compared to the original raw data files to verify that the decryption process produced the same data as the raw input files.

3.3 OpenSSL

OpenSSL is an open source library for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It also serves as a general-purpose cryptography library [13]. The library is compatible with 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.

In the context of this paper, OpenSSL is used solely for its cryptographic functionality, specifically the AES implementations. The reason OpenSSL was 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 library was 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. 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. There are only two function calls that were necessary to initialize and execute the cipher for any data set. This simplified the use of OpenSSL when implementing the software algorithms for AES ECB, CBC, and CTR ciphers. Overall, OpenSSL provided a highly optimized, easily implemented AES software ciphers that are then used for performance comparison with the corresponding hardware accelerated AES ciphers.

4 AES HARDWARE ACCELERATOR

The AES hardware acceleration design involved the use of AES cores with the Xillybus IP core to create the AES coprocessor.

The design wraps logic around the AES cores to implement the individual cipher modes. The ECB and CBC cipher modes use a non-pipelined AES core and the CTR mode uses a pipelined AES core. The dual ARM cores interface to the AES coprocessor via the AXI bus through the use of the Xillybus kernel driver code and the FPGA IP core. The data to be encrypted or decrypted is streamed to and from the FPGA over the bus and the full cipher functionality is performed within the AES coprocessor.

4.1 System View

The AES hardware accelerator is implemented in the FPGA fabric on the Zynq 7000 and is composed of three main components, the Xillybus AXI4 module, AES wrapper cores, and AES cipher cores. Fig. 1 shows the block diagram of the accelerator design connected to the ARM processor.

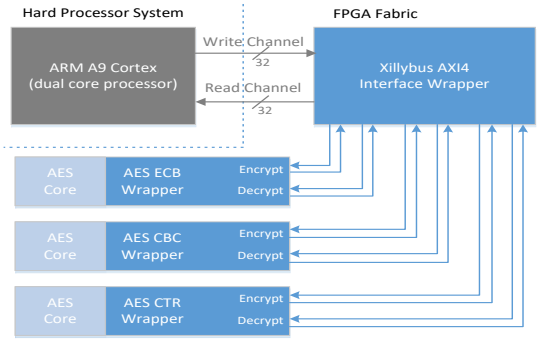

Figure 1: System Block Diagram

Both the Xillybus and AES cores are treated as black boxes during implementation. The Xillybus module implements the communication interface between the ARM processor and the FPGA fabric through the use of Direct Memory Access (DMA). It contains all the necessary logic for interfacing to the AXI4 bus as a slave and a master depending on the direction of communication. The AES cores implement the AES algorithms for either encryption or decryption for a single 128 bit state matrix. There are two different AES cores used, a non-pipelined core used for ECB and CBC ciphers and a pipelined core used for the CTR cipher. The AES cores themselves do not implement any particular cipher mode; therefore, the AES wrapper cores are used for implementing the specific functionality of each cipher mode. The wrapper cores are also used for bridging the interface between the Xillybus and AES cores. Each wrapper core is tailored specifically to the implementation specific features of each cipher mode while using an AES core for performing the actual AES encryption or decryption algorithm. For simplicity reasons, all the logic in the FPGA is clocked off of the AXI4 bus clock which runs at 100 MHz. The current design could run at a maximum clock frequency of 125 MHz which is determined by the maximum clock frequency of the non-pipelined AES core. The pipelined AES core has a maximum clock frequency of 325 MHz.

4.2 Xillybus Interface

Xillybus is an open source Intellectual Property (IP) core developed by Xillybus, Ltd. that implements the necessary logic for the data passing between a Field Programmable Gate Array (FPGA) and a processor via an AXI4 or PCIe bus. The AXI4 bus logic is for implementation on a Xilinx chip such as the Zynq 7000 System-on-a-Chip (SoC) and the PCIe bus version is for implementation on a PC. The desktop version is compatible on both Windows and Linux operating systems. Xillybus also includes the kernel driver software necessary to interface to the FPGA IP core logic. The Xillybus IP core that resides in the FPGA has the ability to function as a slave or a master on the AXI bus. Xillybus utilizes DMA in both the Xilinx and desktop designs to move data between the processor and FPGA with minimal processor overhead [3]. The processor can initiate a DMA transaction when transferring data to the FPGA and, likewise, the FPGA core can initiate a DMA transaction when transferring data to the processor. 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 Xillybus website [14] provides a link called the IP Core Factory where one can create a custom IP core with an arbitrary number of interfaces. The core designer can specify the interface names, their use (i.e. coprocessing, data acquisition, 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 interfaces for efficient communication between the processor and FPGA for a wide range of applications.

For this research, the Xillybus AXI4 version was used for implementation on the Zynq 7000 SoC. The use of the Xillybus IP core and kernel driver software greatly simplified the design process for the AES hardware acceleration because all the necessary code for communicating between the processor and FPGA was provided so more time and effort could be spent on the development of the AES ciphers. The interface between the processor and FPGA can be accessed from software in the form of standard block device files (i.e. in the /dev directory in Linux) and can be accessed with the standard open, read, write, close system calls.

The processor sends data to the FPGA by sending data to DMA buffers in RAM that are then transferred to the FPGA by the DMA engine. The Xillybus core stores the data received on the AXI4 bus in generic 32 bit FIFOs and provides an interface to these FIFOs at its backend for the user to interface to. The same applies when the user needs to send data back up to the processor from the FPGA. The data is stored in FIFOs that the Xillybus core has access to and the core initiates a DMA transaction which copies the data into DMA buffers in RAM. This means that the processor only ever has to interface to RAM when communicating with the FPGA fabric since the DMA engine handles the actual data transferring between RAM and the

FPGA. 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. Fig. 2 shows a high level block diagram for Xillybus with the AXI4 bus used for this particular implementation. The before mentioned user FIFOs for interfacing to Xillybus are shown here as the Application FIFOs.

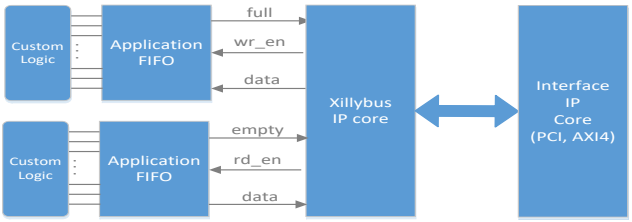

Figure 2: Xillybus Block Diagram

4.3 AES Cores

There are two AES cores used to implement the AES ciphers in the FPGA. One is downloaded from OpenCores.org [1] and the other from Secworks [2]. Both are shown in Fig. 3. The Secworks core is fully pipelined and only implements the encryption algorithm; whereas, the OpenCores core is non-pipelined and implements both the encryption and decryption algorithms. The non-pipelined core can only operate on a single state matrix at a time, but the pipelined core can operate on multiple state matrices at any given time. The AES cores are treated as a black box during implementation. However, they were validated by using known test vectors and verifying the output of the encryption and decryption algorithms were correct.

Fig. 3 shows all the input/output (I/O) signals for the non-pipelined AES core. Due to its sequential functionality, it can only operate on a single state matrix at a time so it must complete the encryption/decryption on the current state matrix before beginning the process on the next state matrix. The core has the ability to dynamically swap between 128 and 256 bit keys and to load different key values. Our implementation only used a 256 bit key so the KEY_LENGTH signal was hardcoded to reflect this. The MODE signal tells the core whether to perform encryption or decryption on the input data so this signal is set based on the desired operation. This core is used to implement the ECB and CBC cipher modes.

Fig. 3 shows all the I/O signals for the pipelined AES core. The pipelined core can operate on multiple state matrices at a time; therefore, increasing its overall throughput through the core. A drawback to this core is that it only implements the encryption algorithm which eliminates its use for ciphers that require both encryption and decryption. However, it is adequate for the CTR cipher mode since it only requires the encryption operation when encrypting or decrypting data. The core can accept a new state matrix every clock cycle. There are 29 pipeline stages 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 what time to read the output state matrix from the core.

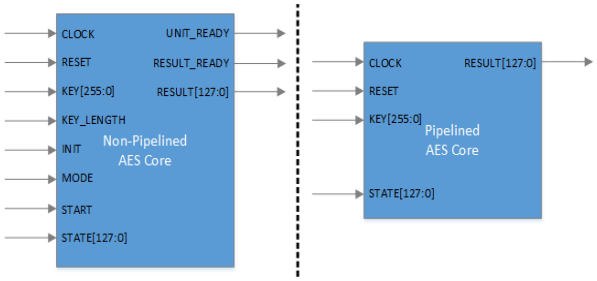

Figure 3: Non-Pipelined and Pipelined AES Cores

4.4 AES Wrapper

The AES wrapper core is used to control the data flow of the AES hardware accelerator by creating a bridge between the Xillybus core and the AES cores. The core is also used as control logic for handling the interaction and handshaking with the AES cores. As seen in Figure 1 there is an AES wrapper core for each cipher mode implemented. The I/O signals for the wrapper core for each cipher mode is shown in Fig. 4. The cores are written in such a way that only the plaintext and cipher text data flow interfaces are externally exposed and all the implementation specific logic for the ciphers are contained internally. This allows for easy integration with the Xillybus IP core because the data flow signals only need to be connected to the data FIFOs provided by Xillybus and the data flow from the processor to the AES cores is complete.

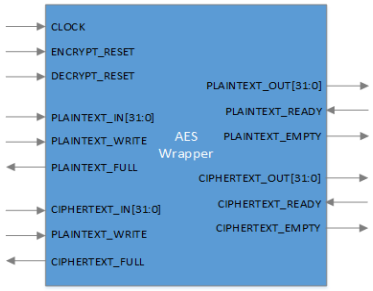

Figure 4: AES Wrapper Core

The wrapper core implements a state machine for both the encryption and decryption process. The state machines follow the same basic flow of operations of receive data from the processor, send data through the AES core, and then send the output data from the AES core back to the processor. The main differences between each of the cipher wrapper cores is the specific

logic needed to implement the functionality of the cipher and then interfacing to the AES core in use. The ECB and CBC ciphers interface to the same AES core, but the CTR cipher interfaces to a completely different 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 its input for AES operation. The maximum data width of the Xillybus IP core is 32 bits since it uses the general-purpose ports on the AXI bus which introduces the need to buffer multiple 32 bit words until an entire state matrix is received. Thus, the first responsibility of the AES wrapper core is to perform such a task; to concatenate four 32-bit words that were received from the ARM processor in order to create a single 128-bit word to pass into the AES core for encryption or decryption. The wrapper core can continuously receive data from the Xillybus core as long as data does not back up further downstream 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 before executing the encryption/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. CTR cipher mode encrypts a concatenated initialization vector and incrementing counter value which is then XORed with either the plaintext (for encryption) or ciphertext (for decryption). The wrapper core for the CTR cipher 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 AES core used for the CTR cipher because the incoming data does not have to wait for the AES core to complete its operation.

The CTR cipher uses the pipelined AES core and the other two modes use the non-pipelined core. The wrapper cores for the ECB and CBC ciphers 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 significant backpressure on the Xillybus FIFOs which could ultimately result in the stalling of the DMA controller when trying to send new data to the FPGA. On the other hand, the wrapper core for the CTR mode can push new data to the pipelined core every clock cycle to buffer up encrypted blocks for later use. This makes the AXI bus the limiting factor on performance for CTR mode because it becomes the component with the highest latency in the system.

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 AES core outputs the resultant state matrix as 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.

5 EXPERIMENTAL ENVIRONMENT

The AES coprocessor is implemented on the Zynq 7000 All Programmable SoC. The AES operations are implemented on the FPGA fabric and the ARM cores are used for initializing the AES coprocessor and streaming the data to the coprocessor. The Zedboard is used as the development platform for the design. It contains the Zynq 7000 chip along with many more hardware peripherals at the disposal of the developer. The Zynq ARM cores are booted with a flavor of Ubuntu Linux called Xillinux which is produced by Xillybus to be used on the Zedboard. The test parameters that are used to test the performance of the three ciphers are file size and cipher operation (i.e. encryption or decryption).

5.1 Zedboard and Zynq 7000 AP SoC

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-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 and includes many more design features [15]. The Zedboard can be programmed using the Xilinx software tool suites of Vivado and SDK.

The Zynq 7000 All Programmable (AP) System on a Chip (SoC) is a single integrated circuit manufactured by Xilinx that contains a hardened dual core ARM Cortex A9 processor (PS) and a programmable logic (PL) fabric to create a full heterogeneous computing system. The dual core ARM processors are feature-rich including multi-level cache hierarchy, 8 channel DMA controller, vector processing units, on-chip memory, external memory interfaces, and a large set of peripheral connectivity interfaces. The FPGA fabric on the Zynq 7000 is comparable to that of either the Artix-7 or Kintex-7 depending on the chip version [16]. It provides a low power and high design flexibility for embedded designs with its large number of resources. 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 [16].

5.2 Xillinux

The Zynq 7000 SoC was booted with a special Linux distribution kernel developed by Xillybus. 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 [17]. 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. The VGA output port of the Zedboard is used for the computer's screen output. Xillinux comes with the Xillybus (discussed in Section 4.2) driver software installed in the kernel for interfacing to the Xillybus IP core. Xillinux is used because it is free for evaluation purposes and is fairly easy to implement. Documentation was provided by Xillinux for how to download, compile, and boot the Xillinux kernel on the Zedboard [17].

5.3 Measurement Setup

The performance metrics that are of interest are the execution time and data throughput for each of the AES cipher modes for both the software and hardware implementations. The parameters that are varied between measurements are the file size, AES cipher mode, and cipher implementation. There are 7 different file sizes (32 KB to 64 MB), 3 cipher modes (ECB, CBC, CTR), 2 cipher operations (encryption and decryption), and 2 cipher implementations (software and hardware); resulting in 84 different parameter configurations. The C/C++ application is executed for each parameter configuration and the execution time of each run is recorded. As a result of the program executing within the Xillinux OS, there is a very small variability in the execution times between runs with the same parameter configuration. To account for this variability, each parameter configuration is executed 3 times and the mean execution time is recorded for post analysis.

The execution time of the ciphers are measured by latching the system time of the Zynq at the start and end of the cipher operation and then computing the difference between the two times. This technique of computing the execution time was used for both the software and hardware implementations. The execution time involved in the generation of the cipher key and initialization vector is not of interest since the hardware accelerator modules did not implement these functions. The throughput of the AES ciphers is calculated by dividing the file size that is encrypted/decrypted by the recorded execution time.

6 RESULTS

The performance measurements of all the runs show that the AES hardware acceleration did improve the performance of the data encryption and decryption. However, the ECB and CBC ciphers did not experience a significant performance improvement as did the CTR cipher.

The speedups of the AES hardware accelerated implementations for the three cipher modes is shown in Fig. 5 for the encryption and decryption processes. The ECB and CBC ciphers only achieved a speedup of about 1.3 for encryption and 1.1 for decryption. The reason these ciphers did not achieve a significant speedup is mainly attributed to the use of the non-pipelined AES core. This core forced the encryption and decryption to operate on one state matrix at a time which created a major bottleneck in the hardware implementation. Therefore, the latency involved with encrypting/decrypting a state matrix in the non-pipelined core is the main contributing factor to the lack of speedup in the hardware implementation of the ECB and CBC ciphers. The CTR cipher, on the other hand, did experience significant speedup for the hardware implementation over the OpenSSL software implementation. It achieved a speedup between 6 and 7 for encryption and about 6 for decryption. The CTR cipher achieved significant speedup because it uses the pipelined AES core so it is capable of encrypting/decrypting the state matrix in a single clock cycle once it's received from the processor. It can do this because the encrypted values from the initialization vector and counter value are pre-computed and buffered. The bottleneck for the CTR cipher becomes the AXI4 bus because there has to be four write transactions to create a complete state matrix for the algorithm to operate on so the AES logic must idle while it waits for the state matrix. The speedup for all the ciphers for file sizes less than or equal to 128 KB was less than 1. The reason for the slower execution times for these file sizes is due to the latency involved with launching the threads used to stream data to and from the FPGA or the data size is too small to hide the latency of the bus transfer.

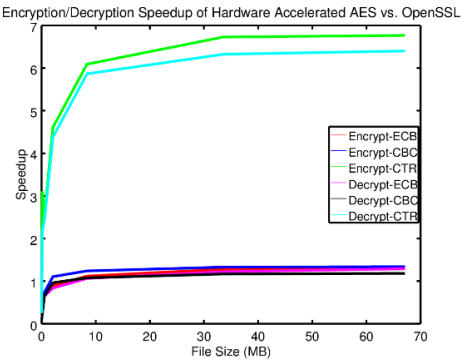

Figure 5: Hardware Accelerated Encryption/Decryption Speedup

The throughput for both the hardware and software implementations of the three ciphers is shown in Figs. 6, 7, and 8. Figs. 6 and 7 show the throughputs for the ECB and CBC cipher which are approximately 18 MB/s for the OpenSSL implementation and about 20 to 25 MB/s for the hardware implementation. This further illustrates that the hardware implementation of these cipher modes did not achieve substantial performance improvements. It is expected that the hardware implementation of these ciphers experienced roughly the same throughput because they used the same AES core. The CTR cipher achieved a throughput of about 55 MB/s for the OpenSSL implementation and about 350 MB/s for the hardware implementation. This was expected since the CTR hardware implementation was fully pipelined and should not have experienced the same data streaming stalls the ECB and CBC modes did. As noted, the FPGA clock rate was limited to 100 MHz.

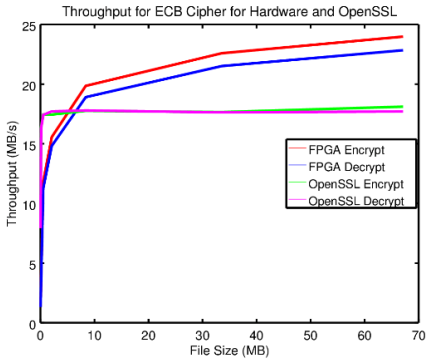

Figure 6: Throughput of ECB Cipher

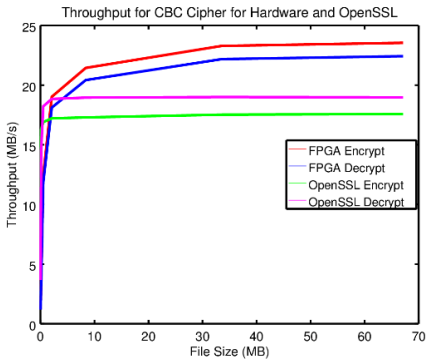

Figure 7: Throughput of CBC Cipher

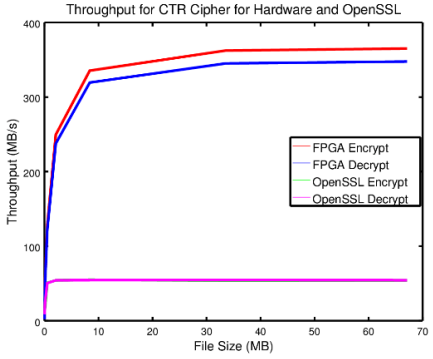

Figure 8: Throughput of CTR Cipher

7 CONCLUSIONS AND FUTURE WORK

The AES ECB, CBC, and CTR ciphers are implemented on the Zynq 7000 AP SoC using software and hardware implementations. The OpenSSL cryptography library is used for the software implementations of the AES ciphers, and AES cores from OpenCores and Secworks are used for the hardware implementations.

File sizes ranging from 32 KB to 64 MB are encrypted and decrypted using the software and hardware implementations of three ciphers. The speedup and throughput is measured for each implementation. The results show moderate speedups for the hardware accelerated ECB and CBC ciphers; however, the CTR cipher achieves up to a 7x speedup and 350 MB/s throughput.

Some possible modifications to the design could be to increase the clock rate for the non-pipelined AES core and the AXI bus to decrease its latency of encryption or data transfers. Lastly, the design could be ported to other embedded or desktop platforms that the Xillybus IP core and driver are compatible with.

ACKNOWLEDGMENTS

This work has been supported in part by the NSF grant CNS-1217470 and the NSA grant H98230-15-1-0268.

REFERENCES

[1] H. Hsing, "AES Core." [Online]. Available: https://opencores.org/project,tiny_aes. [Accessed: 15-Dec-2016].
[2] "Verilog implementation of the symmetric block cipher AES," *Secworks/AES*. [Online]. Available: https://github.com/secworks/aes. [Accessed: 15-Dec-2016].
[3] "Xillybus: Principle of Operation," *Xillybus*. [Online]. Available: http://xillybus.com/doc/xilinx-pcie-principle-of-operation. [Accessed: 15-Dec-2016].
[4] 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.
[5] 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.
[6] 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.
[7] C. Pedraza, J. Castillo, J. I. Martínez, P. Huerta, and C. S. de La Lama, "Self-reconfigurable secure file system for embedded Linux," *IET Computers & Digital Techniques*, vol. 2, no. 6, pp. 461–470, 2008.
[8] 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.
[9] 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.
[10] 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.
[11] NIST, *FIPS PUB 197: Advanced Encryption Standard (AES)*. 2001.
[12] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners*. Heidelberg; New York: Springer, 2010.
[13] "OpenSSL: Cryptography and SSL/TLS Toolkit." [Online]. Available: https://www.openssl.org/. [Accessed: 15-Dec-2016].
[14] "Xillybus: An FPGA IP core for easy DMA over PCIe with Windows and Linux." [Online]. Available: http://xillybus.com/.
[15] "Zedboard." [Online]. Available: http://zedboard.org/.
[16] Xilinx, "Zynq." [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Accessed: 15-Dec-2016].
[17] "Xillinux: A Linux distribution for Zedboard, ZyBo, MicroZed and SocKit," *Xillybus*. [Online]. Available: http://xillybus.com/xillinux. [Accessed: 15-Dec-2016].