MASTER'S THESIS

FPGA Optimization of Advanced Encryption Standard Algorithm for Biometric Images

Toke Herholdt Groth

2014

Master (120 credits)
Master of Science in Information Security

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

Acknowledgment

I would like to thank Dr. Qiang Liu from Tianjin University for sharing the fully pipelined AES encryption core, the code is an excellent piece of engineering and has been a central element in my master thesis. Ingus Krumins deserves special thanks for peer reviewing my thesis, his input was very valuable and I enjoyed very much working together with him. Last but not least would I like to thank Dr. Ali Ismail Awad for providing excellent supervision and sparing throughout the project. The supervision has guided me to both learn and understand how to perform research in the field of information security research.

Abstract

This is a master thesis in the field of information security. The problem area addressed is how to efficiency implement encryption and decryption of biometric image data in a FPGA. The objective for the project was to implement AES (Advanced Encryption Standard) encryption in a Xilinx Kintex-7 FPGA with biometric image data as the application. The method used in this project is Design Science Research Methodology, in total three design and development iterations were performed to achieve the project objectives. The end result is a FPGA platform designed for information security research with biometric image as application. The FPGA developed in this project, is the first fully pipelined AES encryption/decryption system to run physically in a Kintex-7 device. The encryption core was made by Dr. Qiang Liu and his team while the fully pipelined decryption core was designed in this project. The AES encryption/ decryptions was further optimized to support image application by adding Cipher-block chaining to both the encryption and decryption. The performance achieved for the system was 40 GB/s throughput, 5.27 Mb/slice efficiency with a power performance of 286 GB/W. The FPGA platform developed in this project is not only limited to AES, other cryptography standards can be implemented on the platform as well.

Content

List of figures

List of tables

Glossary

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
ASIC	Application specific integrated circuit
BRAM	Block random access memory
CBC	Cipher-block chaining
CFB	Cipher feedback
CLB	Configurable Logic Blocks
CPU	Central Processing Unit
DES	Data Encryption Standard
DCM	Digital clock manager
ECB	Electronic code book
FIFO	First in – first out memory structure
FPGA	Field programmable array
JTAG	Joint Test Action Group interface
KAT	Known answer test
LUT	Look up table
NA	Not available
MMCM	Mixed-Mode Clock Manager
ModelSim	VHDL/Verilog simulator
NIST	National Institute of Standards and Technology
RS232	Standard for Serial single-ended data and control interface
VHDL	Very-high-speed integrated circuits Hardware Description Language
PCIe	Peripheral Component Interconnect Express
USB	Universal Serial Bus
XOR	Exclusive-or logical operation.

1 Introduction

The society is becoming more and more digitalized – therefore Information security is becoming more important than ever. The need for each individual to identify themself in a digital way has spawned a wide variety of challenges, such as, for example, how to avoid fraud. Biometric data as fingerprint or iris scan is one way of identification, however in order to use the data that is reliable for identification purposes the data must stay confidential, for that reason information security is important. The biometric data is typical sampled by a physical terminal and the data is transmitted to a centralized server through a unsecured network for verification, in order to protect the data is encryption needed as soon as possible in the data path – thus in the terminal.

The physical terminal that samples the biometric data must have enough computational power to encrypt the data fast and reliable in a cost efficient way. Further is low power consumption a requirement for handheld terminals. However biometric data can be rather large, e.g. a passport image with the resolution 3300x4400 is 42.5MB uncompressed and 14.2 MB with lossless JPEG compression [1].

The problem of implementing encryption with image application has puzzled researched over the last decade. The encryption is often required to be real-time yet the processing cannot be done at a central server. The literature provides several suggestions how to overcome this problem, e.g. partial encryption of the fingerprint [2]. However this approach might not be feasible for iris scan or voice recognition – and complete encryption could be necessary to have sufficient security.

There are clear advantages by using a standard encryption algorithm for the image application. The reliability is well tested and the data can be shared between different platforms – encryption standards such as DES ((Data Encryption Standard), 3DES (Triple Data Encryption Standard) and AES (Advanced Encryption Standard) are standardized by National Institute of Standards and Technology. However the mentioned encryption algorithms require many calculations steps and storage of partial results in order to encrypt the data. Therefor is a CPU (Central Processing Unit) not ideal for this type of tasks, since it require many cycles for the CPU to perform the encryption calculations. Previous studies have showed that encryption can be performed much fast using a FPGA (Field Programmable Gate Array) compared to a CPU [3].

The CPU is a dedicated logical circuit designed to execute instruction and calculation in a sequential order. The FPGA is a programmable logic circuit – which means that is function of is not fixed after the silicon fabrication. The device consist of thousands of "building block" called CLB (configurable logic blocks), each of CLB's can be individually be configured to a specific logic function and each of the CLB can (with some limitation) be connected to any other CLB through the routing network. Figure 1 show a typical FPGA architecture, the CLB slices are located in a matrix pattern and is surrounded by several different types of dedicated blocks; multipliers, Block random access memory (BRAM) and digital controlled clocking managers (DCM). The configuration of the components and their interconnectivity implements the actual functionality of the FPGA.

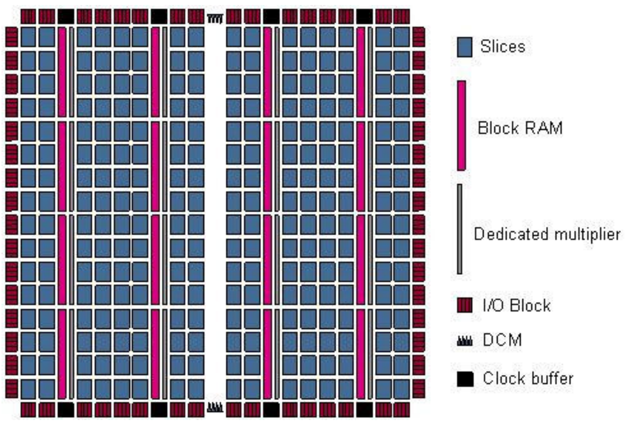


Figure 1: Principal drawing of the FPGA architecture [4]

The strength of an FPGA compared to a CPU is that many smaller circuits can be implemented to run in parallel, while the CPU is native a sequential circuit. The partial results for the encryption can be calculated in parallel and combined in a later stage, reducing the number of cycles required significantly. The FPGA cannot run at the same clock frequencies as a modern CPU (which is in the GHz range), the FPGA runs typically with an internal processing clock of 250-500 Mhz. Despite the lower clock frequency can a FPGA process data much faster than a CPU, due to parallel processing capabilities – thus encrypt data at a high rate.

The ideal technology for implementing AES in hardware is ASIC (Application specific integrated circuit). Here is it possible to custom design your chip to the application and achieving speeds that are superior to both CPU and FPGA. However the cost of design an ASIC far exceeds the scope of this project both in development time and production cost. Therefor is the second best choice the FPGA.

2 Problem Statement

2.1 Research question

How to efficiently implement encryption algorithm in a FPGA with image as the application?

2.2 Project description

This project consist of hardware oriented research; how to efficiently implement encryption algorithm with biometric image as an application. The focus of the project is to implement an encryption algorithm in a FPGA with image as an application. The knowledge contribution of the project is to address the problems of efficiently encryption biometric data in real-time using state of the art FPGA technology. The project will contribute with an power efficient hardware implementation aimed at the current marked leading FPGA technology, which can encrypt image data in real-time. The purpose of the project is also to create an FPGA image encryption platform to be used for further research in respect to comparing different encryption algorithm, cost optimization and power consumption optimization.

This project will focus on implementation of AES in a FPGA which is a standardized algorithm that recognized by the literature. The image data for encryption is biometric samples e.g. fingerprint images. The chosen architecture is Xilinx Kintex-7, since this FPGA family is the market leader on performance versus power versus price. The Kintex-7 utilizes 28 nm die technology, which is minimize the dynamic power consumptions compared to previous 40 nm die technology [5] The implementation should be portable to similar architectures, thus be compatible with Artix-7 and Virtex-7. The project includes performance measurements of the implementation, in respect to encryption speed, power consumption and data integrity.

Figure 2 shows a principal drawing of the FPGA system diagram that is being proposed. The AES encryption algorithm is located in the middle. The input and output buffers serves the purpose of avoiding real-time problems with data transfer, which complicates the performance analysis exclude the need for a high speed host interface.

The host starts by loading the master key into the master key register. The host sends the test data to the host interface through RS232 (Serial single-ended data and control interface). The host interface load the data block into the input buffer. The AES starts the encryption when a block of data is ready in the buffer; the data is loaded through a wide fast interface to avoid bandwidth problems. The AES encrypts the data and sends the result to the output buffer, the host interface transfers the finished block back to the PC for integrity analysis. The analysis monitor sends the performance statistics back to the PC for further analysis.

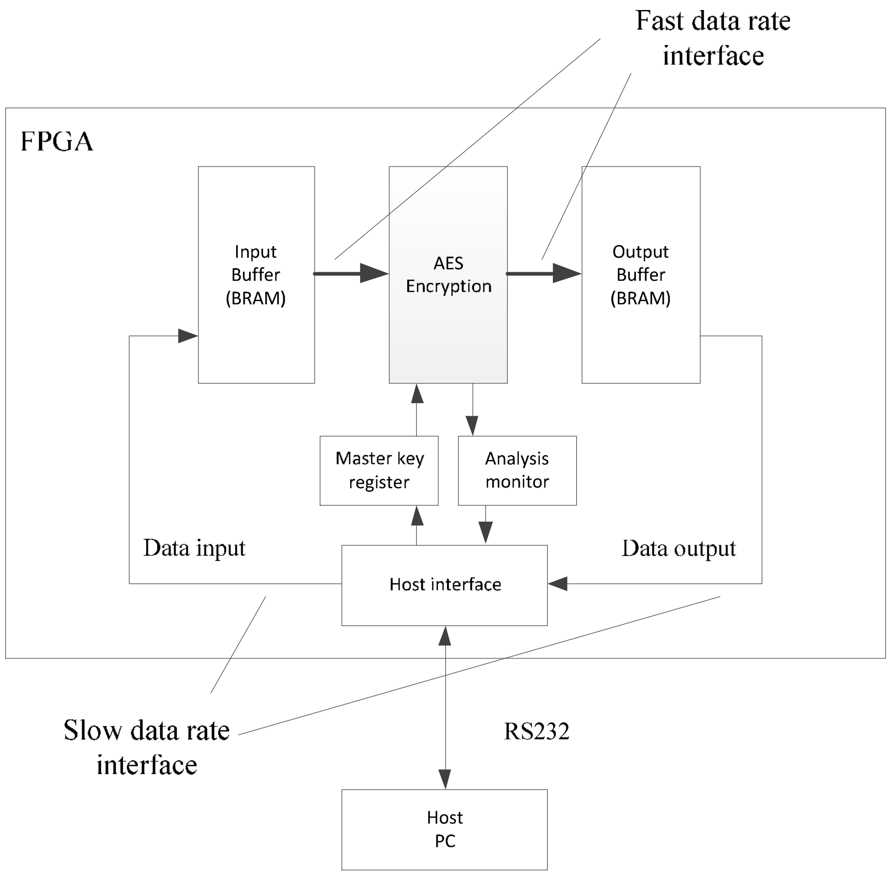


Figure 2: Principal drawing of the encryption FPGA

The structure of the FPGA allows porting of the AES algorithm to a similar FPGA architecture. The system design of the FPGA also allows inserting other encryption algorithm modules for analysis, e.g. 3DES.

2.3 Objectives

The objectives for the project are:

1. Implementation of AES encryption in a FPGA.
2. Use biometric image data as the application.
3. Research how to efficiently implement AES in Xilinx Kintex-7 architecture.
4. Performance test throughput rate, latency, data integrity and power consumption.
5. Create a hardware platform that can be used for further biometric information security research.

The goal is to implement a fully functional AES algorithm in a Xilinx Kintex-7 FPGA. The expected outcome is to test the AES in real hardware using a Xilinx development board thus power measurements. The throughput and latency of the AES running on Xilinx Kintex-7 will be measured. The throughput has to be high enough so biometric images can be used as the application and the latency has to be low enough so the performance can be considered as real-time. The power of the device under full speed operation is measured and the integrity of the data is verified.

2.4 Research contribution

The research contribution is to implement NIST (National Institute of Standards and Technology) standard AES encryption in leading edge FPGA technology to perform real time encryption for biometric image data. The study is aimed to be the first implementation of AES encryption in Kintex-7 architecture. To be the first that implement a high throughput, low latency AES, good enough to process biometric data in real-time. To be the first that perform power measurements of high throughput AES running at full speed in 28 nm FPGA technology for future reference and benchmarking.

2.5 Delimitations

In this project will we not consider any of the security aspects of handling encryptions key. The encryption key will be uploaded to the FPGA by the host before the encryption cycle is initiated, the key will be stored in regular register in FPGA, which will be vulnerable to JTAG (Joint Test Action Group interface) read-back. Side channel attack will not be considered in this project.

The image size for biometric data be rather large as mentioned in the introduction, in projected is limited to use smaller image sizes (maximum 132 kb). This limitation is due to the internal memory of the target FPGA.

This project is delimitated to only optimize the throughput of the AES encryptions. There are many possibility in FPGA development to optimize both clock speed and area. However these types of optimizations will not be part of this master thesis.

This project is delimitated not to develop host communication software, instead is scripting and a simple terminal programs used.

3 State of the art

3.1 Definitions

Throughput defined as the total data throughput of the encryption core in terms of Gigabits per second Gb/s, defined as the symbol **R** [6].

Latency is defined as the maximum time it take a word to pass though the encryption kernel measured in clock cycles, defined as the symbol $\mathbf{t_L}$.

Work size is defined as the amount of data to be encrypted, the work size measured in bit, defined as the symbol **W**.

Processing time is defined in this project as a benchmark for having a job to pass though the encryption kernel where both the throughput, work size and the latency are taken into consideration. The processing time is named $\mathbf{t_{proc}}$:

$$t_{proc} = \frac{W}{R} + t_L \text{ (eq. 1)}$$

The η (efficiency) of an FPGA AES implementation is measured as the throughput divided with the area of the FPGA used [6]. The area is measured in terms of FPGA slices, the unit for efficiency are mega bit per slice, Mb/slice.

$$\eta = \frac{R}{\text{area}} \text{ (eq. 2)}$$

The power performance for encryption kernel is measured in terms of encryption rate per Watt, defined in Gb /(W *s), and is defined as the symbol $\mathbf{P_p}$.

In this paper is the logical operation exclusive-or (XOR) used widely, the definition for the mathematic operator is showed in Figure 3 while the structural symbol is showed in Figure 4.

$$\oplus$$

Figure 3: Mathematical operator for XOR

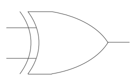


Figure 4: Structural symbol for XOR

The hex values notation used is VHDL standard: e.g. X"5A" correspond to 90 in decimal value.

3.2 Literature review

The literature review is divided into two areas: High performance AES FPGA implantations and Image applications using AES FPGA implementation. The first area are studied in order to find out how efficient encryption has been implemented in FPGA so far, the purpose is also to identify if there are consensus in how to implement the AES, to find if there is reference design or there are competing design suggestion how the AES is most efficiently implemented. The second area is studied is to have an overview of what have already been achieved, in respect to using FPGA for AES encryption of image data.

3.2.1 High performance AES FPGA implantations

Rahimunnisa, et al published a paper in December 2013 describing the Parallel sub-pipelined (PSP) architecture. The PSP architecture uses 128 bit data blocks which are divided into four blocks of 32 bit, each of these 32 bit blocks are processed in parallel, in order to achieve high throughput. The architecture is a mix of parallel and sequential processing, which has achieved a high efficiency. The design has been both implemented in a Virtex-6 LX75T FPGA and prototyped as an ASIC design. The throughput achieved on Virtex-6 LX75T was 59.59 Gb/s, the area used was 2597 slices, giving an efficiency of 22.94 Mb/slice. The results was retrieved be simulating the design using ModelSim (VHDL/Verilog simulator). The work included power simulations for 130 nm and 180 nm ASIC die technology [6].

Liu, Xu and Yuan published a paper in December 2013 where real time AES encryption was in focus. The paper describes a 66.1 Gb/s fully pipelined AES 128 bit FPGA implementation. The FPGA was implemented on the new Xilinx Virtex-7 VX690T device, they achieved 66.1 Gb/s using 3436 slices thus achieving an efficiency of 19.20 Mb/slice. The latency of the design is 22 clock cycles at a clock running at 516 Mhz, which is equal to 426 us. The paper further suggest to run two AES kernels in order to break the 100 Gb/s barrier, this should be possible with the chosen target, since only a fraction of the slices are used. The design was only simulated and no power estimations were performed [7].

Cai X, Sun R and Liu J has implemented AES on a FPGA by using pre-calculated liner combination of the keys and storing the results in FPGA rom. The solutions have a theoretical throughput of 40.96 Gb/s, with only a 10 clock latency. The designed was only simulated and there was no area estimations publish and therefore can the efficiency not be determined. There was no aspect of power consumptions mentioned [8].

Kumar and Sharma has improved the latency in the AES kernel by using an enhanced VLSI implementation. The SubBytes, which are part of the S-box in the AES encryption has been implemented in logic instead of placing them in ROM (BRAM), since the access time for the CLB's are much lower compared to BRAM access, then is the latency decreased. The designed was implemented in a Xilinx Virtex-2 device and the simulation shows that the latency can be reduced by 0.6-0.9 ns, which is roughly the penalty for accessing the BRAM. While the FPGA technology used are rather outdated, the study shows, that latency improvement can be achieved by reducing the access time for the parameters of the s-box [9].

Dogan and Saldamli studied in 2012 the design techniques for FPGA AES encryption to achieve low power consumption. The designed minimize the power by reusing calculations block in the AES kernel such as the S-box. Instead of performing true parallel calculations, input the recused kernel was time slot multiplexed, thus utilizing the periods where the blocks are idle. The design was targeted to a Xilinx Spartan-3 XC6SLX150L. The designed was running at low speed, 20 MHz and the throughput was low. The conclusion was that proposed design technique did lower the power consumption drastically. However no absolute power numbers was published [10].

This FPGA implementation of AES encryption as counter mode for 256 bits data width was done by Balwinder Singh, Harpreet Kaur and Himanshu Monga in 2010. They achieved to encrypt at 52.6124 Gbit/s with a master key length of 256 bits. The design was implemented in Xilinx Spartan 3, Xilinx Virtex II and Xilinx Virtex E devices [11].

Akman and Yerlikaya have recently published an article where they compare encryption performance in a FPGA versus a CPU. The FPGA implementation was in VHDL (Very-high-speed integrated circuits Hardware Description Language) while the CPU implementation was in C programming. The encryption algorithm was a 128 bit wide AES with a key length of 128 bit. The comparison was based on simulations results; the encryption processing time was 390ns for the FPGA and 11000 ns for the CPU. This article is relevant for our study since it provide an empirical example of the superior performance of the FPGA versus a CPU [3].

The fully pipelined AES implemented was implemented by Hodjat and Verbauwhede in 2004. They managed to fit the algorithm into one VirtexII-Pro FPGA. The latency for the algorithm was only 31 clock cycles and they achieved an encryption rate of 21.54 Gbit/s. The implementation used 84 BRAMs and 5177 CLB slices, giving an efficiency of 4,2 Mb/slice if the BRAM usage is not taken into consideration [12].

In 2011 did a team consisting of Hongying Liua, Ying Zhoub, Yibo Fanc, Yukiyasu Tsunood and Satoshi Goto study how to increase the security the FPGA implementation, by considering the possibility of side channel in form of differential power analysis - by using advance randomization where they able to hide data-dependent encryption in the power spectrum. The performance of the implementation was 2.56 Gbit/s [13].

Jason Van Dyken and José G. Delgado-Frias investigated in 2010 how encryption strength and power consumption was related in FPGA implementation of AES. The study showed how to lower the power consumption of the encryption with minimum effect on performance. They were able to lower the power consumption with 66% while only lowering the encryption strength with 27%. The target device was a Xilinx Virtex-II Pro [14].

The implementation of AES encryption and decryption in a FPGA was done in 2010 by Yogesh Kumar and Prashant Purohit, they have implemented a parallel 128 Bit AES in a Xilinx Spartan 3 device. The focus of their work was the achieving high hashing speed in a low-cost device [15].

The AES encryption was implemented in a FPGA in sequential and parallel architectures in 2003 by Nazar A. Saqib, Francisco Rodriguez Henriquez and Arturo Diaz-Pirez. The aim of the research was to compare sequential and parallel architectures in respect to area and speed. In sequential architecture did the implementation occupies 2744 CLB slices while the parallel architecture occupied 2136 CLB slices. There was not used any BRAM for the implementation. The sequential architecture was encrypting at 0.259 Gbit/s while the parallel architecture was encrypting at 2.868 Gbit/s. The target device was Xilinx Virtex E [16].

3.2.1 Image applications using AES FPGA implementation

The AES FPGA implementation was used as an image application by Chang et. al in 2009. They implemented a full encryption and decryption system in a Virtex-2 device, using a host PC to control the FPGA with a RS232 link. The application was aimed to be a low area low cost solution to image encryption. The AES core was a 32 bit and occupied only 104 slices, and had a throughput of 794 Mbps, giving an efficiency of 7.93 Mb/slice [17].

Image compression and image encryption were combined by Ou, Chung and Sung in 2006. By compressing an image before encryption, they addressed two problems at the same time: they decreased the amount of data to be encrypted and increased the entropy of the AES encryption. The design was using 128 bit AES, the encryption speed was 330 Mb/s. The efficiency and power consumption were not addressed [19].

Gupta, Ahmad, Sharif and Amira constructed in 2011 a wireless communication prototype system. The aim of the project was to demonstrate secure image transmitting of live images over Bluetooth. The system consisted of two Xilinx development boards, one for transmitting and one for receiving. A CMOS Camera was connected to the transmitter board, the image data was encrypted with an AES core and send over Bluetooth to the receiver board, where the data was decrypted and showed on a monitor. They used AES 128 bit for encryption and archived a throughput of 7.87 Gb/s with an efficiency of 2.23 Mb/slice. The decryption had a throughput of 7.03 Gb/s with an efficiency of 1.26 Mb/Slice [20].

The AES encryption/decryption was implemented in a Xilinx MicroBlaze processor by Gore and Deotare in 2013, specially aimed for image application. The Xilinx MicroBlazer is a soft microprocessor, meaning that it a real microprocessor but implemented in the logic of an FPGA. The MicroBlaze is native 32 bit and AES 128 bit was implemented by having four MicroBlaze in parallel. The design was implemented in a Xilinx Spartan 3 and they achieved throughput of 3.40 Gb/s with an efficiency 5.43 Mb/slice. The design was only simulated and no power estimates was made and you could expect that the latency was long due to the microprocessor architecture [21].

Manoj and Manjula implemented AES 128 bit as an image application in a Xilinx Spartan 3 device in 2012. The design was similar to what others have done, expect that the design could take 8 bit input (data pixels) and unroll them to 128 bit, which is a trivial task. The encryption throughput was 882.46 Mb/s, the efficiency was 0.53 Mb/slice and the latency was 24 clocks. The article includes plots with the relation between core voltage and power consumptions for the device. However, these drawings were not commented and no power estimate of the design was made [22].

The design proposed by Karimian, Rashidi and Farmani in 2012, was aimed to achieve high throughput and low power consumption for AES encryptions using image as the application. They implemented an AES 128 bit core in an Altrea Stratix device and achieved a throughput of 617 MB/s. The efficiency of the design was 0.76 Mb/slice. The approach for lowering the power was resource sharing, pipelining and signal gating. They estimated the power consumption to be 301 mW @100Mhz clock using the Xilinx XPower tool. There was no estimate of the power consumption at full clock speed (475 MHz). Moreover there was no evaluation of how much power saving techniques improved the power performance [23].

3.3 Literature review conclusion

The literature review has shown that there is no standard FPGA implementation of AES. Each implementation was aimed to achieve different goals. The review has revealed that preferred key length for the researcher was 128 bit, instead of the more secure AES-256 bit.

The most common goal for the researchers was to achieve as high throughput as possible. The highest throughput was achieved by Liu, Xu and Yuan by using a single pipelined architecture; they were able to reach 66 Gb/s on Virtex-7, while having an efficiency of 19.20 Mb/slice [6]. The closest competing implementation was done by Rahimunnisa, et al - they used a Parallel sub-pipelined architecture, which basically is a singled pipeline architecture, where part of the 128 bit kernel is split into four separate 32 bit blocks. They achieved a throughput of 59.59 Gb/s, with the efficiency 22.94 Mb/slice [6]. However, they used an FPGA technology that is one generation younger, virtex-6. They did not publish the latency for the core, nevertheless the design is comparable with the single pipeline architecture and we can therefore expect that the latency is approximately 20 clock cycles.

The literature review has revealed a knowledge gap - so far there has been little focus on power consumption. The reason could be that the published designs are only conceptual and the problem of reducing power consumption is left out for further research. Rahimunnisa, et al [6] did make an effort to simulate the power consumption for an older CMOS ASIC technology, but power simulation is not specifically accurate and only providing a rough estimate. Dogan and Saldamli had a study aimed directly at power reduction 0. However Dogan and Saldamli measured power on a low throughput design on older FPGA technology. There were no absolute measurements for the power consumption and the results cannot be transferred to new die technology since the ratio between static and dynamic current has changed drastically. Karimian, Rashidi and Farmani did an effort to estimate power consumption using the Xilinx XPower too, yet the XPower only provides a rough estimate which enables the hardware designer to dimension the power supply. Moreover, the estimate was not done at the target clock frequency, which only adds uncertainty to the estimate.

The conclusion of the literature review is that there are some excellent design ideas for AES FPGA implementation, such as the Parallel sub-pipelined architecture which have excellent performance on a modern FPGA technology, both in respect to throughput, efficiency. The performance summary for the articles is listed in
Table 1 (NA is acronym for Not available).

Table 1: Literature review performance summary

Author	**R** [Gb/s]	η **[Mb/slice]**	t_L **[clocks]**	P_p **[Gb/Ws]**
Liu, Xu and Yuan	66.10	19.20	20	NA
Rahimunnisa et. al	59.59	22.94	N/A	NA
Sing, Kaur and Monga	52.61	NA	NA	NA
Cai, Sun and Liy	40.96	NA	10	NA
Hodjat and Verbauwhede	24.54	4.20	31	NA
Gupta, Ahmad, Sharif and Amira	7.87	2.23	NA	NA
Gore and Deotare	3.40	5.43	NA	NA
Saqib, Rodríguez-Henríquez and Diaz-Pirez	2.87	1.34	NA	NA
Manoj and Manjula	0.88	0.53	24	NA
Chang et. al	0.79	7.93	NA	NA
Karimian, Rashidi and Farmani	0.617	0.76	NA	2.05[1]
Ou, Chung and Sung	0.33	NA	NA	NA

1. Based on estimate at 100Mhz clock frequency

Table 1 show that not all authors have shared the archived latency and efficiency. The table also emphasizes the gap in knowledge of power performance for AES FPGA implementation. Only one article was found where an attempt was made to quantize the power consumption.

The design published by Liu, Xu and Yuan is chosen for further studies, since they have the design with highest throughput and best efficiency. The design is an excellent candidate for power measurements and optimizations.

3.4 Theory of operation

The theory behind the AES algorithm is described in this section. The AES algorithm is specified by NIST [1], this section provides a summary of the specification defined by NIST with a few exceptions. The NIST has a programming approach to describing the algorithm including use of programming examples and pseudo code while this description has a hardware approach with the use of structural and logical diagrams. Further the NIST describes the mathematics behind Rijndael's galois field in detail, this is omitted from this description to keep the focus on the structural details. The AES algorithm supports key lengths of 128, 192 or 256 bit; however this paper only describes the 128 bit key, since only the 128 bit key length is being implemented.

The AES algorithm uses a fixed input size of 128 bit (called a data block), the purpose of the AES algorithm is to encrypt the information of the input data block and hide the correlation between the input data block, the key and the output data block. The AES algorithm is basically two mathematical functions: a function for encryption (AES_{enc}) and a function for decryption (AES_{dec}). The two functions are each other's inverse:

$$AES_{enc}^{-1} = AES_{dec} \text{ (eq.3)}$$

The AES_{enc} takes two inputs: a data block (D) and a key (K), the output of the function (Q) is the encoded data:

$$Q = AES_{enc}(D,K) \text{ (eq.4)}$$

The AES_{dec} takes two inputs: a block of encoded data (Q) and the inverse key (K^{-1}), the output of the function is the data block (D):

$$D = AES_{dec}(Q,K^{-1}) \text{ (eq.5)}$$

K is also referred to as the encryption key where K^{-1} is described as the decryption key, the two keys are each other's inverse and in this chapter we will later describe the mathematical relationship between them.

3.4.1 AES Encryption

The AES encryption is series of matrix calculations which are repeated ten times, each of these repetitions is called rounds. For each round is the state matrix "mixed" with the key, the result is a new state matrix and a new key which is used as input to the next round.

The AES Encryption is a rather complicated series of matrix operations. The order of the different operation is illustrated in the structural diagram in Figure 5.

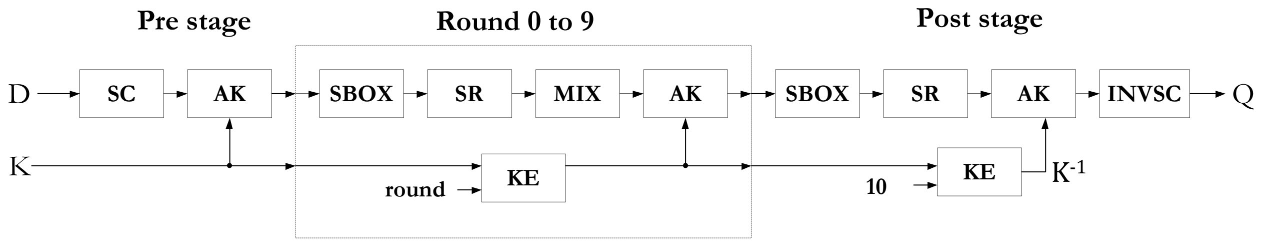


Figure 5: Structural diagram of AES Encryption

The pre stage consists of a state matrix conversion, to convert the 128 bit input vector to a 4x4 byte matrix followed by an Add key operation. The input key is used for the add key operation as well as transferred to the round 0.

Round 0-9 is cascaded, meaning that the same series of operation is performed ten times. The output of a round is input to the next round. Each round consists of a substitution box, shift row, mix column and add key operation. The key for the given round number is calculated with the key expander, the new key is used as input to the following round. The output from last round (round 9) is transferred to the post stage.

The post stage is the final round for the encryption, it consists of a substitution box, shift row and add key operation. The key for the last add key operation is calculated as well with a key expand operation. The last key calculated is in fact K^{-1}, however the key is not output under normal circumstances, since the key has the same level of confidentially as the input key. By having the K^{-1}, K can be found simply by performing 11 rounds of inverse key expanding.

3.4.2 AES Decryption

The AES decryption is the same series of matrix operations as for AES encryption; however the operations are mathematically inverse and are performed in opposite order. Figure 6 shows the structural diagram.

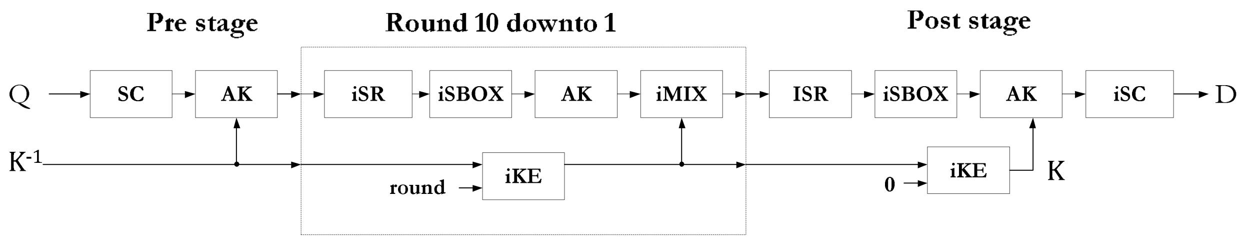


Figure 6: Structural diagram of AES Decryption

The pre stage consists of a state matrix conversion, to convert the 128 bit input vector to a 4x4 byte matrix followed by an Add key operation. The add key operation is its own inverse, therefore it is the same exact add key operation as for encryption. The inverse key is used for the add key operation as well as transferred to round 10.

Round 10-1 is cascaded in the same way as for encryption, the difference is that we start with round 10 and go down to round 1 - the post stage will process round 0, which is the last round for decryption. The output for each round is input to the next round. Each round consists of a inverse shift row, inverse substitution box, add key and inverse mix column operation. The key for the given round is calculated with the key expander, the new key is used as input to the following round. The output from last round (round 1) is transferred to the post stage.

The post stage consists of a inverse shift row, inverse substitution box and a final add key operation. The key for the last add key operation is calculated with a key expand operation. The last key calculated is in fact K

3.4.3 State matrix conversion (SC)

The input to the AES function is a 128 bit vector consisting of 16 bytes: $D = D_{15} \dots D_1, D_0$ where D_{15} is most significant byte and D_0 is least significant byte. The first step is the state matrix conversion; the operation is not an arithmetic operation but rather a conversion from vector format to matrix format. The 128 bit vector is converted into a 4x4 byte matrix, called the state matrix (Figure 7).

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Figure 7: State matrix

The four least significant bytes is filled into the first row, the next four bytes is filled in to next row etc. the pattern is showed in equation 6.

$V_0=S_{0,0}$, $V_1=S_{0,1}$, $V_2=S_{0,2}$, $V_3=S_{0,3}$, $V_4=S_{1,0}$ ... $D_{15}=S_{3,3}$ (eq.6)

3.4.4 Inverse state matrix conversion (iSC)

The inverse state matrix conversion converts a 4x4 byte matrix back to 128 bit vector. Matrix index 0,0 will be the least significant byte of the vector (D_0), matrix index 0,1 will be the next byte etc. the pattern is showed in equation 7.

$S_{0,0}=V_0$, $S_{0,1}= V_1$, $S_{0,2}=V_2$, $S_{0,3}=V_3$, $S_{1,0}=V_4$ ... $S_{3,3}=V_{13}$ (eq.7)

3.4.5 Substitution box (SBOX)

The substitution box (SBOX) is practically a look up table with 8 bit input and 8 bit output, which gives a total of 256 entries, see Figure 8.

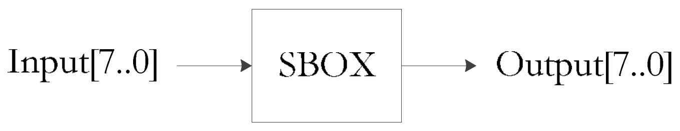
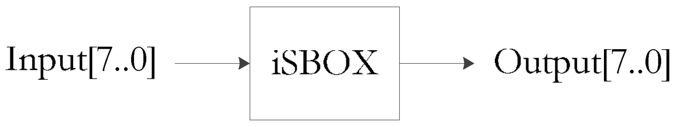


Figure 8: SBOX look up table

The content of the SBOX is specified by NIST [1], it is determined on the basis of the multiplicative inverse of the Rijndael's galois field. In this paper we will not describe the galois field in detail, instead will we acknowledge that the SBOX can be performed with the look up table provided by NIST. The SBOX Look up table is included in Appendix A.

3.4.6 Inverse substitution box (iSBOX)

The inverted substitution box is the exact opposite of the SBOX. E.g. if X"00" is the input to the SBOX the output is X"63", if X"63" is the input to the iSBOX then is X"00" the output. The content of the SBOX is specified by NIST [1], but can also be derived from the SBOX look up table.



The iSBOX Look up table is included in the Appendix A.

3.4.7 Rotate (ROT)

The rotate operation takes a vector of 4 bytes as input. The bytes are rotated left as illustrated in Figure 9.

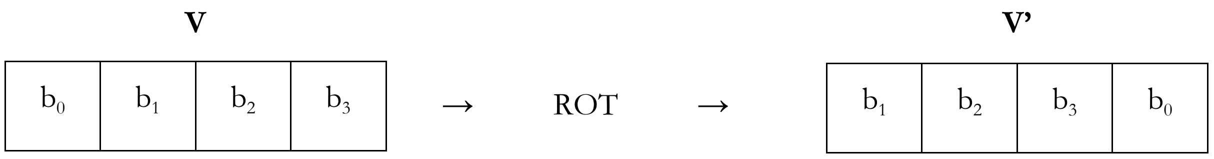


Figure 9: Rotate operation

3.4.8 Inverse rotate (iROT)

The inverse rotate operation takes a vector of 4 bytes as input, instead of rotating the byte left as in the rotate operation, the byte is rotated right as illustrated in Figure 10.

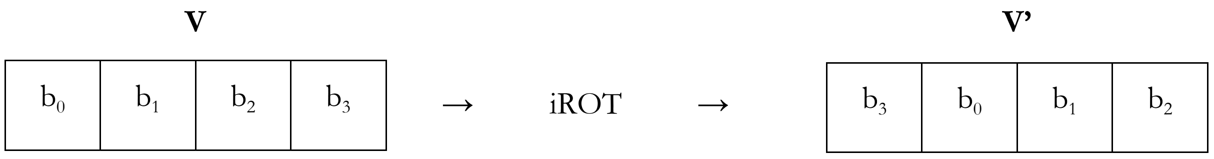


Figure 10: Inverse rotate operation

3.4.9 Shift row (SR)

The shift row operation takes a 4x4 byte matrix as input and performs a series of byte rotation on each row: row 0 is unchanged, row 1 is rotated one time, row 2 is rotated 2 times and row 3 is rotated 3 times, this operation is illustrated in Figure 11.

	S							**S’**		
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	→	Unchanged	→	$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	→	ROT x 1	→	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	→	ROT x 2	→	$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	→	ROT x 3	→	$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

Figure 11: Shift row operation

3.4.10 Inverse shift row (iSR)

The inverse shift row operation takes a 4x4 byte matrix as input and performs a series of inverse rotation on each row: row 0 is unchanged, row 1 is inverse rotated one time, row 2 is inverse rotated 2 times and row 3 is inverse rotated 3 times, this operation is illustrated in Figure 12.

	S							**S’**		
$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	→	Unchanged	→	$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	→	iROT x 1	→	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	→	iROT x 2	→	$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$	→	iROT x 3	→	$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

Figure 12: Inverse shift row operation

3.4.11 Mix column (MIX)

The mix column operation takes a 4x4 byte matrix as input, and performs a matrix multiplication for each column with a constant vector. The idea behind the matrix multiplications is that each column is treated as a four term polynomial in the Rijndael's galois field and is multiplied with a constant polynomial a(x). The constant polynomial is defined by NIST and is shown as equation

8, the two digits in the brackets for the constants emphasize that the constant is one byte in hexadecimal.

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \text{ (eq.8)}$$

The matrix multiplication for each column in the input matrix is shown in Figure 13, where c denotes the column number (0-3) for the operation.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 13: Mix column matrix multiplication [1]

In this paper will we not go into details with the mathematics behind the matrix multiplication, we recognize that each column of the input matrix has to be multiplied with a matrix of constant bytes as shown in Figure 13.

3.4.12 Inverse mix column (iMIX)

The inverse mix column operation takes a 4x4 byte matrix as input, and performs a matrix multiplication for each column with a constant vector. The operation is similar to the mix column operation, except that the inverse of the constant polynomial is used $a^{-1}(x)$. The inverse constant polynomial is defined by NIST and is shown as equation 9, the two digits in the brackets for the constants emphasize that the constant is one byte in hexadecimal.

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \text{ (eq.9)}$$

The matrix multiplication for each column in the input matrix is showed in Figure 14, where c denotes the column number (0-3) for the operation.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 14: Inverse mix column matrix multiplication [1]

3.4.13 Add round key (AK)

The add round key operation takes a 4x4 byte matrix and a 128 bit key as input. The operation performs a bitwise XOR between the matrix and the input key. The key is converted to a matrix before the XOR operation, this is done by placing the first four bytes in the column 0, the next four bytes in column 1 etc. The Add round key operation is illustrated in Figure 15.

$$\overset{\textbf{S}}{\begin{array}{|c|c|c|c|}\hline S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ \hline S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ \hline S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ \hline S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \\ \hline \end{array}} \oplus \overset{\textbf{W}}{\begin{array}{|c|c|c|c|}\hline w_0 & w_4 & w_8 & w_{12} \\ \hline w_1 & w_5 & w_9 & w_{13} \\ \hline w_{2,} & w_6 & w_{10} & w_{14} \\ \hline w_3 & w_7 & w_{11} & w_{15} \\ \hline \end{array}} = \overset{\textbf{S'}}{\begin{array}{|c|c|c|c|}\hline S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ \hline S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ \hline S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ \hline S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \\ \hline \end{array}}$$

Figure 15: Add round key operation

Notice that the inverse of the round key operation is exactly the same operation, since a bitwise XOR of S' will give S.

3.4.14 Round constant (RCON)

The round constant is a look up table, which takes the round number as input (0-10), which contains 4 bit, the output is the 8 bit value called the round constant, see Figure 16.

$$\begin{tikzcd} \text{round[3..0]} \arrow[r] & \boxed{\text{RCON}} \arrow[r] & \text{rcon[7..0]} \end{tikzcd}$$

Figure 16: Round constant look up table

The content of the look up table is specified by NIST [1], and is calculated by using equation 10.

$$\text{RCON} = 2^{\text{round}} \bmod 2^8+2^4+2^3+2+1 \text{(eq.10)}$$

The result of the calculation is truncated to 8 bit, the complete RCON look up table is found in the Appendix B.

3.4.15 Key expansion (KE)

The key expansion operation is fundamental for the AES algorithm, for each round is the key altered with the key expansion, the altered key is used as input to next round. The altered key is referred to as the key schedule since there is one unique key for each round. A top level description of this operation is:

$$w_{round+1} = KE(W_{round,}\ round), \text{ where } w_0 = K \text{ (eq.11)}$$

The last key (round 10) is the inverse key (K^{-1}) used in eq.5, $w_{10} = K^{-1}$.

The key consist of 16 bytes, however the key expansion processes the key as four sets of four bytes. The structure of the key expansion for one set of four bytes can be seen in Figure 17, this structure is repeated for each set of four bytes. The processing of the four sets all uses the same round number.

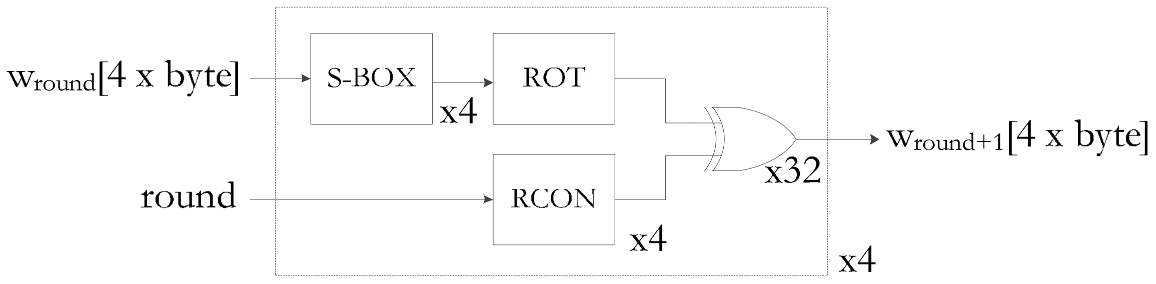


Figure 17: Key expansion for four bytes of the key

The first operation is applying the SBOX to each of the four bytes. The next step is a byte left rotation. The round key is used to find the round constant (RCON), which is a simple lookup table with 11 entries. The round constant is XOR'ed with the four bytes from the ROT operation. The result is four new bytes for the next key schedule.

3.4.16 Inverse key expansion (iKE)

The inverse key expansion operation could also be called the key compression, but for consistency the operation is called inverse key expansion. The inverse key expansion converts the inverse key (K^{-1}) back to the keys original state (K). The top level description of this operation is:

$$w_{round-1} = iKE(W_{round,}\ round), \text{ where } w_{10} = K^{-1} \text{ (eq.12)}$$

The inverse key expansion processes the key as four sets of four bytes. The structure of the key expansion for one set of four bytes can be seen in Figure 18, this structure is repeated for each set of four bytes. The processing of the four sets uses the same round number.

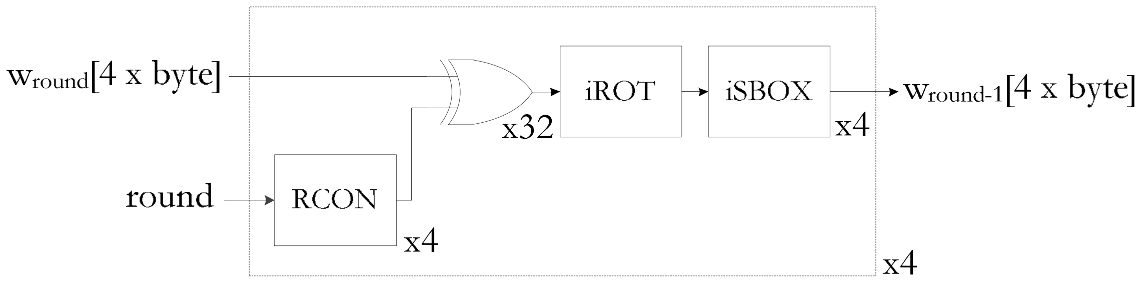


Figure 18: Key expansion for four bytes of the key

The first operation is applying the SBOX to each of the four bytes. The next step is a byte left rotation. The round key is used to find the round constant (RCON), which is a simple lookup table with 10 entries. The round constant is XOR'ed with the four bytes from the ROT operation. The result is four new bytes for the next key schedule.

3.5 Provided AES Cores

The AES encryption core used in this project was provided by Dr. Qiang Liu from Tianjin University in China. The source code was send directly to my supervisor and later the code was handed over to me, all rights for AES encryptions core are reserved to Dr. Qiang Liu and his team. The AES decryption core is downloaded from open cores and follows GNU Lesser General Public License. Table 2 shows the list of cores which was made by others and used in this project.

Table 2: List of AES cores used in this project

ID	Name	Author(s)	Reference
AES0	66.1 Gbps single-pipeline AES on FPGA	Liu Q, Xu Z, Yuan Y. A	[7]
AES1	Fast AES-128 Hemanth Satyanarayana	Hemanth Satyanarayana	[24]

4 Methodology

The research method used in this thesis is Design Science Research Methodology.

The Design Science Research Methodology is a relatively new method; it was first published in a journal article in 2007. The method is developed specifically for the field of Information Security and covers the gap between interpretive research in the field of information security and the discipline of engineering [25]

The objective of the project is to efficiently implement encryption algorithm with image as the application. We need the knowledge from Information Security to choose and evaluate the encryption algorithm used for the image application but we also need the discipline of engineering to implement and test the effectiveness of the chosen encryptions techniques, for this type of problem is Design Science Research Methodology an obvious choice. Other methods could also be chosen, but since this study aims to improve a current implementation of AES, is it difficult to predict what design changes that would produce good results. Therefor is iterative process like Design Science Research Methodology an effective approach.

4.1 Design Science Research Methodology

This section will briefly describe the design science research methodology, based on the journal article "A Design Science Research Methodology for Information Systems Research" [25].

The method uses six activities which are nominally executed in a sequence. However the method is not constraining the researcher to start at the first activity. Further, the method is an iterative process which means that the result of a given activity determines if the researcher goes forward to the next activity or choses to goes back to the previous activity and uses the new knowledge as input. The flow of the method is shown in Figure 19.

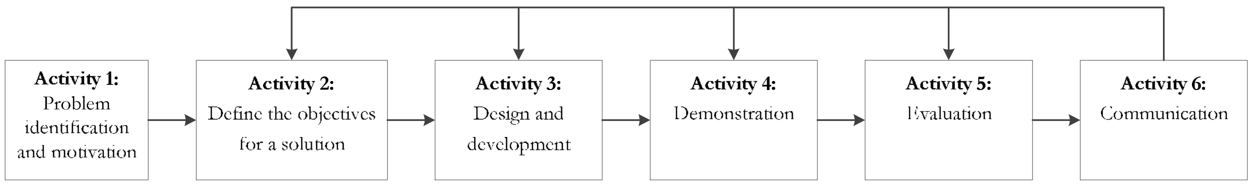

Activity 1: Problem identification and motivation
Activity 2: Define the objectives for a solution
Activity 3: Design and development
Activity 4: Demonstration
Activity 5: Evaluation
Activity 6: Communication

Figure 19: Design Science Research Methodology flow [25]

Activity 1: Problem identification and motivation

In this activity the research problem is identified and the research question is formulated. This activity also involves justification of the scientific contribution for the research and statement. The value of the solution is also defined in this activity.

Activity 2: Define the objectives for a solution

The objectives for the solutions are derived from the problem definition. The objectives can be either quantitative or qualitative as long as they are inferred rationally from the problem identification. The objectives must be based on what is possible and feasible based on the state of the art. Resources needed to reach the objective should also be taken into consideration, such as retrieval of source code from previous research or special hardware.

Activity 3: Design and development

The design and development of the artifact is done in this activity. The artifact must be a research design artifact, which means that the research contribution is embedded into the artifact. The design and development contains the normal steps used in engineering: determining the artifacts functionality, designing the architecture and implementing the solution, thus creating the artifact.

Activity 4: Demonstration

The created artifact is used to solve one or more parts of the research problem. The activity should not be confused with testing, which is part of the design and development, this activity assumes that the artifact is functioning as specified. The artifact is used to generate scientific data which can be evaluated in the next activity.

Activity 5: Evaluation

The purpose of this activity is to evaluate how well the artifact provides a solution to the research problem. The data generated in the demonstration activity is analyzed and the result is evaluated. The results are compared to the objectives of the project and the arguments for answering the research question are formed. If the researcher is satisfied with the result and the research question can be answered the next activity starts: communication. However if the argumentation for answering the research question is not solid, then the researcher can choose to execute another iteration of the design and development activity, where the result is generated in the evaluation activity used as design input to improve the design. In case where the results are incomplete and more data is generated, the researcher can choose to execute another iteration of the Demonstration activity.

Activity 6: Communication

The focus of this activity is to communicate the results of the research to other researchers, commercial professionals or other relevant audience such as students or the public. The activity involves communication concerning the importance of the research problem and presenting the created artifact and formulating how the artifact contributes to solving the research problem. The means of communication should take the target group into considerations, for example if the target groups are researchers then the output can be a paper publication in a scientific journal.

4.2 Project research process

The research process in this project follows the principals of Design Science Research Methodology. The flow of the process is illustrated in Figure 20, the Design Science Research Methodology allows us to start at any activity, however this project is problem centered and therefore the best activity to start with is number 1. The input to the project research process was the project idea, which was to create a research platform to solve problems with the privacy of biometric data.

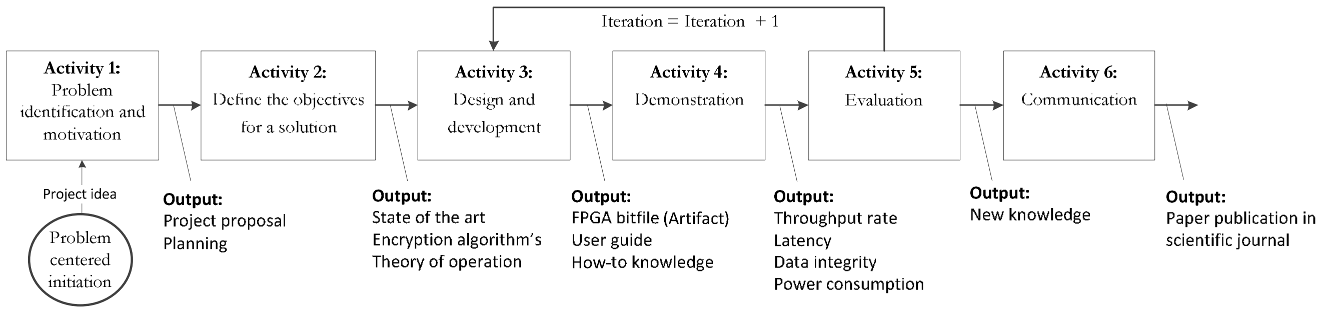

Iteration = Iteration + 1
Activity 1: Problem identification and motivation
Activity 2: Define the objectives for a solution
Activity 3: Design and development
Activity 4: Demonstration
Activity 5: Evaluation
Activity 6: Communication
Project idea
Problem centered initiation
Output: Project proposal Planning
Output: State of the art Encryption algorithm's Theory of operation
Output: FPGA bitfile (Artifact) User guide How-to knowledge
Output: Throughput rate Latency Data integrity Power consumption
Output: New knowledge
Output: Paper publication in scientific journal

Figure 20: Project research process flow

Activity 1: Problem identification and motivation

The project idea was synthesized into a master thesis project proposal. The problem to be addressed was how to implement encryption of biometric image data in hardware. The motivation for the project is that there is a need for real-time encryption in security terminals with power consumptions. The planning for project was also done in this activity, you can argue that the planning should be done in activity 2, since it is hard to plan are project where the objectives for the solutions are not fully known. However due to limited timeframe of master thesis project, the overall planning has to be done before you know how to create the solutions. This is not much different from the discipline of engineering, the project time frame is often fixed long time before the engineers know how to solve a problem.

Activity 2: Define the objectives for a solution

The creation of State-of-the-art has been the driver for this activity. The key component is the literature review, since it is fundamental for defining the objectives for solution that can generate new knowledge. The state-of-the-art has also served another important role, it has shown that encryption in FPGA has been done before; therefore it has little scientific value simply to repeat what others have done before, instead our solution is based on reusing a AES encryption algorithm already implemented and tested. Part of this activity has been to contact the researchers who have already implemented AES encryption in FPGA and retrieve their source code.

Activity 3: Design and development

The activity has a number of steps which is typical for FPGA design:

1. System design for the FPGA. The system design defines all modules in the FPGA and how they are interconnected.
2. Detailed design. The functionality for all the modules was described and the interface between all modules was defined.
3. VHDL code writing. The VHDL code was written based on the detailed design. Further, a test bench for each module was designed. There was also designed a test bench for the overall FPGA design.
4. Simulation. All modules are simulated and verified against the detail design specification. As well as the entire FPGA was simulated.
5. Implementation. The implementation of the FPGA is primarily a tool driven task, the actual synthesis and PAR (place and route) is done by the development tools. However the tools operate based on the input from the designer, the primary input is the VHDL code and the UCF file (user constraint file). The output is a bitfile that can be loaded into a FPGA.

After the last step the creation of the artifact is complete.

Another output from this activity was the user guide, which is a manual how to control the FPGA from a user perspective. The user guide is often referred to as the programmer's reference, since the user of a FPGA is typically a piece of software that controls the FPGA. The design and development activity also output "how-to knowledge", since the process of designing and implementing a solution gives the researcher knowledge on how to use the artifact.

Activity 4: Demonstration

The artifact (The FPGA design) was used to perform the following measurements:

- Verification
- Throughput rate
- Latency
- Data integrity
- Power consumption.

The artifact provides a complete research platform. The test data is loaded into the FPGA through the host interface. The encryption is initiated by the start command and the results are read out from the registers in the analyses module. Throughput rate, latency, data integrity is all measured by the analyze module. However the FPGA is not able to measure its own power consumptions, this was done external with an oscilloscope and a current clamp probe.

Activity 5: Evaluation

The measurements from the previous activity were used to determine if the objectives for the project had been reached. This activity shows the effectiveness of the solutions, and the result was used to improve the design. The design science research methodology allows us to do iterations, therefore the knowledge from this activity was used to perform another iteration of the design and development, in respect to optimize throughput, latency and power consumption. This does not mean that the whole design and development was redone; only adjustment and optimizations were performed. At each of the iterations the Demonstration activity was redone, since a new design gave new results. The output from this activity is new knowledge to the field of information security.

Activity 6: Communication

This is out of the scope for this project, nevertheless the results from this studies could be considered for publication in a scientific journal.

5 Design and development iterations

This chapter describes the design and development iterations for this project, there has been a total of three iteration, which is named with the suffix #1, #2 and #3. Each iteration follows the Design Science Research Methodology; first section is a design and development activity, followed by a demonstration activity and is concluded with an evaluation activity.

5.1 Design and development #1

This chapter describes iteration #1 (the first iteration) of the design and development. The design and development consist of following steps:

1. System design
2. Detailed design
3. Simulation
4. Implementation

Each of the steps is described in the following sections.

5.1.1 System design

The system design synthesizes the project objectives into solution on a high abstraction level. The key considerations in the system design are that the FPGA design must be as generic as possible, meaning that it should be suited as a research platform for this problem but also for future projects. The next consideration is the main objective in the research: how encryption can be implemented efficiently in a FPGA, thus different encryptions must be compatible with design. Further, in order to enable the design to be used for further research, the architecture must be as simple as possible, yet give enough flexibility to solve different problems. The science contribution of this system design is not to design the AES cores, but rather to provide the surrounding architecture for the AES core. This contribution is important since it enables comparison of different AES cores, power measurements and optimization.

The FPGA architecture for the FPGA is shown in Figure 21. The center is the encryption and decryption core. This core can be any AES core the only requirement is that the core must be described in Verilog or VHDL, this design uses a wrapper around the core in order to comply with the interface specification of **i2** and **i3**.

The FPGA is controlled by the host interface; the protocol for communication is RS232. This enables the researcher to use a generic program to control the FPGA. The list of commands available is described in the user guide.

Before encryptions can be done, the host must initialize the control register in the register bank, load the master key into the register bank and load a block of input data into the input buffer. The FPGA can bypass both the encryption and decryption, which enables the FPGA to be used as encryption only or decryption only. In most cases it is preferred to run the FPGA with both encryption and decryption since it makes the evaluation activity faster. However, in some case it can be convenient to have access to raw encrypted data or decrypt encoded data.

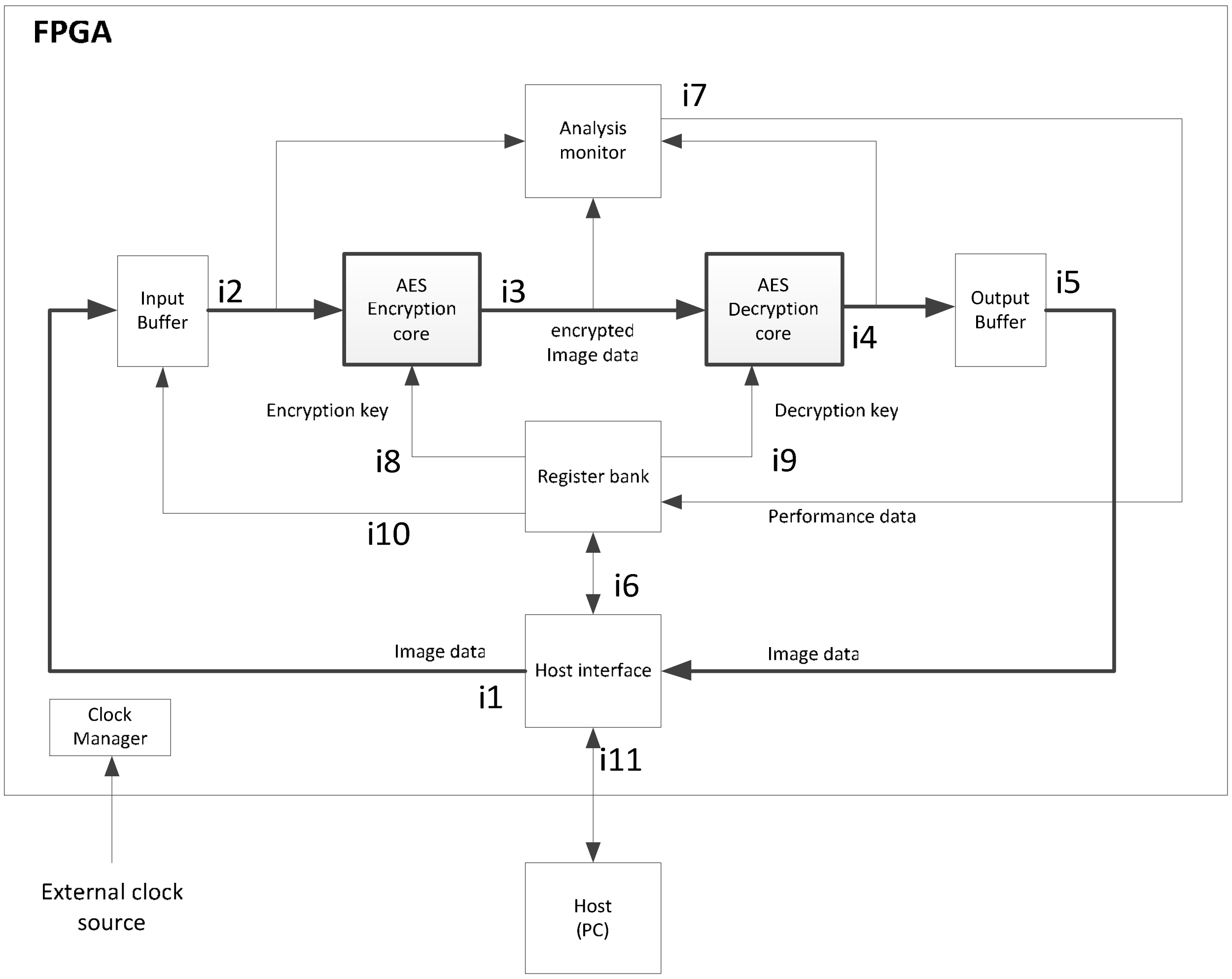


Figure 21: FPGA architecture

The FPGA has built-in analyzing capabilities; this is implemented in the analysis monitor. The analyze monitor can measure the latency from the input to the output of the AES cores and validate the data integrity. The data validation is done by comparing the input to the encryption core with the output of the decryption core. The results are available for the user in the register bank.

5.1.1.1 Interface specification

This section describes all interfaces in the FPGA, it is considered good design practice to have only one common interface description for the entire system, thus avoiding that two modules disagree on how the interface is specified. In order to keep the design as generic as possible, a standard interface protocol is chosen between the modules. The image data is transported with AXI4-Stream protocol and the control data is using AXI4-Lite protocol. The AXI protocol is used in the AMBA (Advanced Microcontroller Bus Architecture), and is the standard communication protocol used by Xilinx. Other protocols or even designing one could have been chosen, however by using a standard protocol it will be easy for other researcher to use, understand and modify the design. The RS232 protocol for host communication is chosen since it is a simple and easy protocol to implement; the downside is that is very slow. However the bandwidth to host is not of any concern in this project. Nevertheless, the protocol could fairly easy be substituted with a faster protocol such as PCIe (Peripheral Component Interconnect Express) in a future version. The detailed interface specification is found in Appendix C while a summery is showed in Table 3.

Table 3: Interface description summery

Name	Type	Description
i1	AXI4-S	Image data from host interface to input buffer
i2	AXI4-S	Image data from input buffer to encryption core
i3	AXI4-S	Image data from encryption core to decryption core
i4	AXI4-S	Image data from decryption core to output buffer
i5	AXI4-S	Image data from output buffer to host interface
i6	AXI4-Lite	Control data from host interface to register bank
i7	Parallel	Status registers 4 x 32 bit parallel.
i8	Parallel	Encryption key 128/256 bit parallel
i9	Parallel	Decryption key 128/256 bit parallel
i10	Parallel	Start signal to initiate data stream
i11	RS232	Host communication

5.1.2 Detailed design

5.1.2.1 Clock manager

The clock system in the FPGA is straightforward - the FPGA is driven by one external clock, which in principal could be at any frequency, for the development board a 66 MHz clock is available, which will be used in this project. The clock manager synthesizes the clock into the target frequency by using an MMCM (Mixed-Mode Clock Manager). The MMCM is a dedicated Xilinx component which uses a Phase-locked loop to generate the master clock (ACLK). The FPGA design is fully synchronously to the master clock, expect for the RS232 link which runs at a slow asynchronous clock. The clock manager is shown in Figure 22.

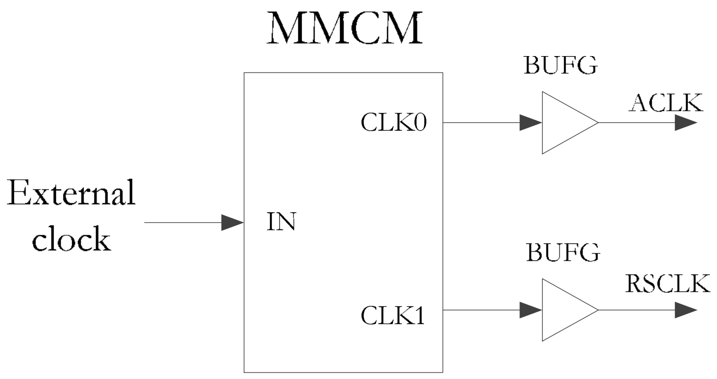


Figure 22: Clock manager

The encryption rate is direct proportional to master clock, which means higher clock frequency is equal to higher encryption rate. In this design the master clock frequency is:

f_{ACLK} = 180 MHz

The baud rate of the RS232 link is chosen to be 9600 baud:

F_{RCLK} = 9600 Hz

5.1.2.2 Host interface

The host interface uses a standard UART to convert parallel data into serial data for host communication. The state machine interprets the command words from the host, depending on the command either data will be written on i1 or i6, or data will be read from i5 or i6. The principal drawing of the module is shown in Figure 23.

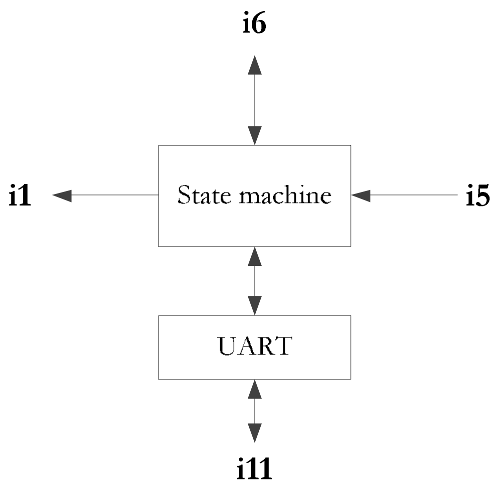


Figure 23: Host interface

The host writes data to the FPGA sending a command byte followed by a payload. The command specifies if read/write mode, destination and length of the payload.

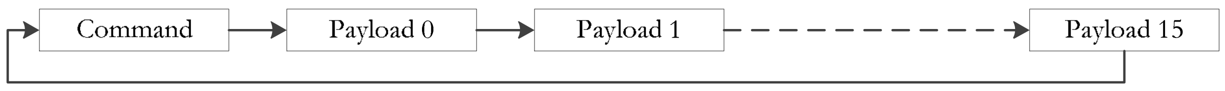


Figure 24: Example of host communication with 16 byte payload

The command byte is coded as shown in Figure 25.

MSB		LSB
LENGTH	DEST	RD/WR
7..3	2..1	0

Figure 25: Command byte coding

RD/WR: 0 = Read, 1 = Write

DEST: 00 = Image data, 01 = Control register, 10 = Encryption key, 11 = Analysis data

LENGTH: Payload (in bytes) = (LENGTH+1)*8

In case of a read command, the host interface will return [PAYLOAD] of data bytes to host interface. In case there is no data to return, the host interface returns 0xFF.

This host protocol has some limitations, e.g. the maximum payload is
Max Payload = (31+1)*8 = 256 bytes.

In case more data needs to be transferred, a new command has to be issued.

5.1.2.3 Register bank

The register bank is 128 bit wide and has three registers. Table 4, Table 5 and Table 6 show the layout of the registers.

Table 4: Control register

Parameter	Bit	Description
Run	0	Start/stop the encryption/decryption of the input buffer content
Loop	1	Loop through data in the input buffer
Bypass Encoding	2	Bypass the encoding module
Bypass Decoding	3	Bypass the decoding module
Image size	17..4	Image size in 128 bit words.
Reset	18	Reset the system, the transition from 0 to 1 initiates the reset.
Reserved	127..19	Not used

Table 5: Encryption key register

Parameter	Bit	Description
Encryption key	127..0	128 bit encryption key

Table 6: Analysis register

Parameter	Bit	Description
Latency encoding	8..0	Measured data latency for the encoding in clocks
Latency decoding	16..9	Measured data latency for the decoding in clocks
Data error	31..17	Number of detected errors (stop counting at 16383)
Reserved	127..32	Not used

The register bank includes a key convert, which can convert the encryption key into the decryption key. This is done automatically and is not of the user concern.

5.1.2.4 Input buffer

The size of the input is 1M bit. This enables a maximum image size of 132 kb.
The FPGA has 14M bit internal memory so the maximum image size could be increased if needed. The input buffer is implemented as a dual port BRAM, where the AXI4-Stream data is written into memory. When the run command is issued the content of BRAM is transferred word by word to the encryption core. The amount of data transferred depends on the image size specified.

5.1.2.5 Output buffer

The output buffer has the same size as the input buffer. The output buffer is basically implemented as FIFO (First in – first out memory structure). The received image data from the decryption module is written into the output buffer memory. The output buffer forwards the data to the host interface. The host interface controls the flow with the AXI4-Protocol (using the TREADY signal).

5.1.2.6 Analysis monitor

The analysis monitor is implemented as three counters: latency encoder counter, latency decoder counter and data error counter. The latency counter starts when first data arrives to a module, and stops when first data is outputted. This gives effectively the modules latency in clocks. The data error counter does as the name implies - it counts the amount of data errors. The data errors are measured by storing the data transferred to the encryption module in a FIFO. The data outputted by the decryption module is compared to original data before encryption. The depth of the FIFO is 1024 (we assume that the latency for encoding+decoding is less than 1024).

5.1.2.7 Encryption core

The encryption core used in this design iteration is provided by Liu Q, Xu Z, Yuan Y. A [7]

In order to use the core in the design, it needs a wrapper to make it compatible with the AXI4-S interfaces. The wrapper is shown in Figure 26.

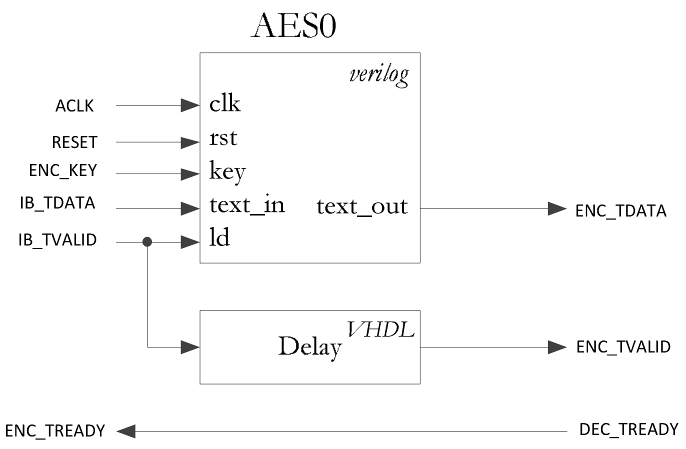


Figure 26: Encryption core wrapper

The most important issue with the AES0 core that needed to be addressed is that the core has no data valid output, this means that there is no way to know when the data on the output is valid. The latency is 20 clocks, this number is found by analyzing the source code. Therefore the valid signal on the input is delayed and used as the valid signal for the output.

This is an excellent example on an experimental design (AES0) which needs to be improved in order to be used in a real hardware implementation.

5.1.2.8 Decryption core

The encryption core used in this design iteration is provided by Hemanth Satyanarayana [24].

The AES1 core is did not any need modification to be compatible with the FPGA. The wrapper is shown on Figure 27.

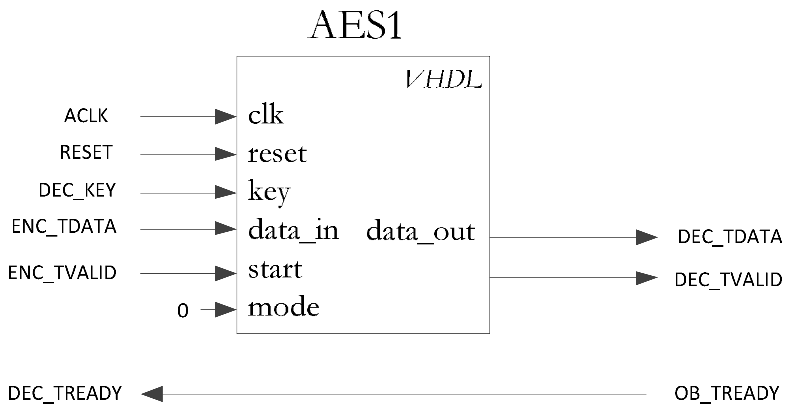


Figure 27: Decryption core wrapper

The AES1 core can do both encryption and decryption, this is controlled with a mode bit, in this case the mode bit is hardwired to zero (decryption).

5.1.3 Simulation

The primary clock (aclk) is set to 1.934 ns (517 Mhz), the same clock speed that Liu, Xu and Yuan used in their study.

5.1.3.1 AES Core

NIST has provided different sets of AES Known answer test (KAT) vectors [1]:

- The encryption key is constant, 128 different input vectors and the encoded vectors.
- The input vector is constant, 128 different encryption keys and the encoded vectors.

For this simulation will we use the vector set with constant encryption key and variable input data. The simulation uses only the first 16 sets of vectors.

The encryption key is:

`X"00000000000000000000000000000000"`

The decryption key is:

`X"b4ef5bcb3e92e21123e951cf6f8f188e”`

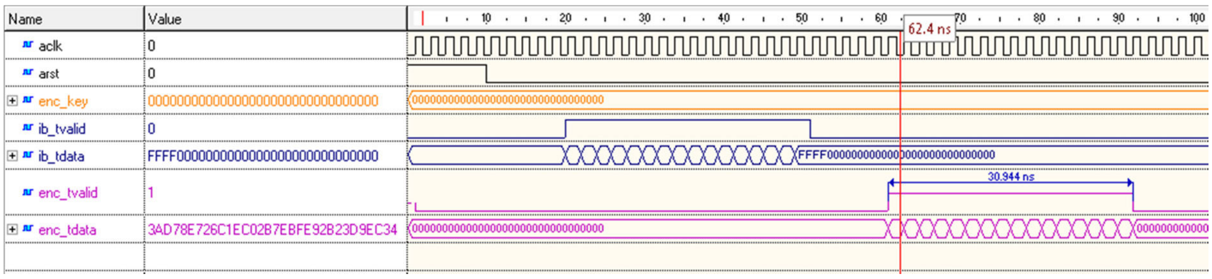
The input vectors follow a specific pattern, where the first four vectors are:

```
X"80000000000000000000000000000000"
X"c0000000000000000000000000000000"
X"e0000000000000000000000000000000"
X"f0000000000000000000000000000000"
```

The encoded vectors do not follow any pattern (since they are the result of an encryption), the first four encoded vectors are:

```
X"3ad78e726c1ec02b7ebfe92b23d9ec34"
X"aae5939c8efdf2f04e60b9fe7117b2c2"
X"f031d4d74f5dcbf39daaf8ca3af6e527"
X"96d9fd5cc4f07441727df0f33e401a36"
```

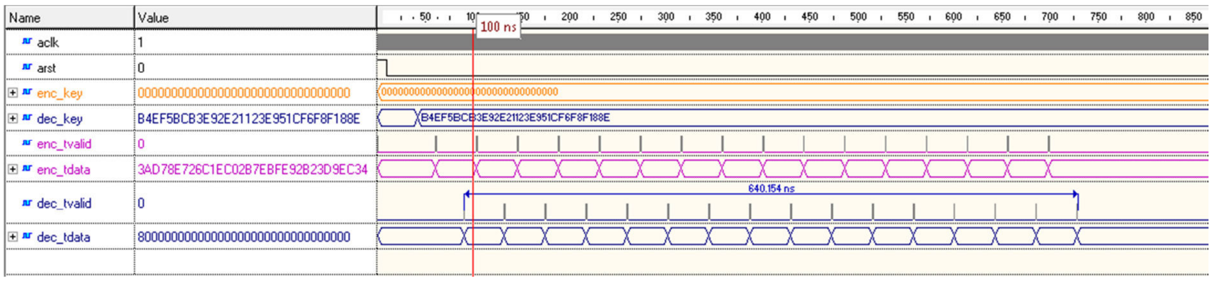
Figure 28 shows the simulation of the encryption core, the enc_tvalid signal shows that the core is capable of running at full data rate (128 bit data is processed at each clock). The processing time for the test vectors (16x128bit) is 30.944 ns, which correspond to 66.18 Gb/s. The output of the core (enc_tdata) corresponds to the KAT vectors.



Name	Value
aclk	0
arst	0
enc_key	00000000000000000000000000000000
ib_tvalid	0
ib_tdata	FFFF0000000000000000000000000000
enc_tvalid	1
enc_tdata	3AD78E726C1EC02B7EBFE92B23D9EC34

Figure 28: AES core encoder simulation result #1

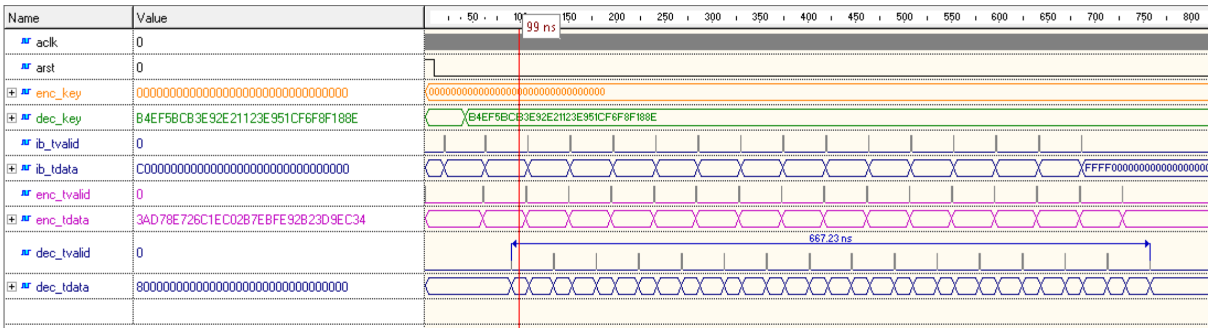
Figure 29 shows the simulation of the decryption core, the dec_tvalid signal shows that the core is only cable of processing one data value at the time. The data rate is 1/22, which is equal to the core latency. This highly reduces the speed of the core. The processing time for the test vectors (16x128bit) is 640.154 ns, which correspond to 3.20 Gb/s. The output dec_tvalid corresponds to the KAT vectors.



Name	Value
aclk	1
arst	0
enc_key	00000000000000000000000000000000
dec_key	B4EF5BCB3E92E21123E951CF6F8F188E
enc_tvalid	0
enc_tdata	3AD78E726C1EC02B7EBFE92B23D9EC34
dec_tvalid	0
dec_tdata	80000000000000000000000000000000

Figure 29: AES core decoder simulation result #1

Figure 30 shows the simulation of the encryption core connected to the decryption core. The decryption core is setting the pace, which means that the high throughput of the encryption core is not utilized. The processing time for the test vectors (16x128bit) is 640.154 ns, which corresponds to 3.20 Gb/s. The output vectors correspond to the input vectors, meaning that the encryption / decryption functionality is verified.



Name	Value
aclk	0
arst	0
enc_key	00000000000000000000000000000000
dec_key	B4EF5BCB3E92E21123E951CF6F8F188E
ib_tvalid	0
ib_tdata	C0000000000000000000000000000000
enc_tvalid	0
enc_tdata	3AD78E726C1EC02B7EBFE92B23D9EC34
dec_tvalid	0
dec_tdata	80000000000000000000000000000000

Figure 30: AES core encode/decode simulation results #1

5.1.4 Implementation

The target device was:

Xilinx Kintex-7, 7K325T-FFG900-2

The FPGA has been implemented in three different ways:

1. The AES encryption core only
2. The AES decryption core only
3. The complete FPGA

This gives us the possibility to evaluate both AES cores as well as the complete FPGA design. The results are summarized in Table 7.

Table 7: Implementation of results #1

Implementation	Number of slices	Number BRAM	Achievable clock
The AES encryption core only	3684	105	3.385 ns
The AES decryption core only	1361	1	3.178 ns
The complete FPGA	5845	163	3.412 ns

All nets were successfully routed.

The bit file was successfully generated.

5.2 Demonstration #1

The setup for the artifact demonstration can be seen in figure x. The host PC communicates with Xilinx evaluation board with a USB (Universal Serial Bus) link. The evaluation board has a USB to RS232 converter connected to the FPGA. Hyper terminal program running on the host PC is used to send commands to the FPGA

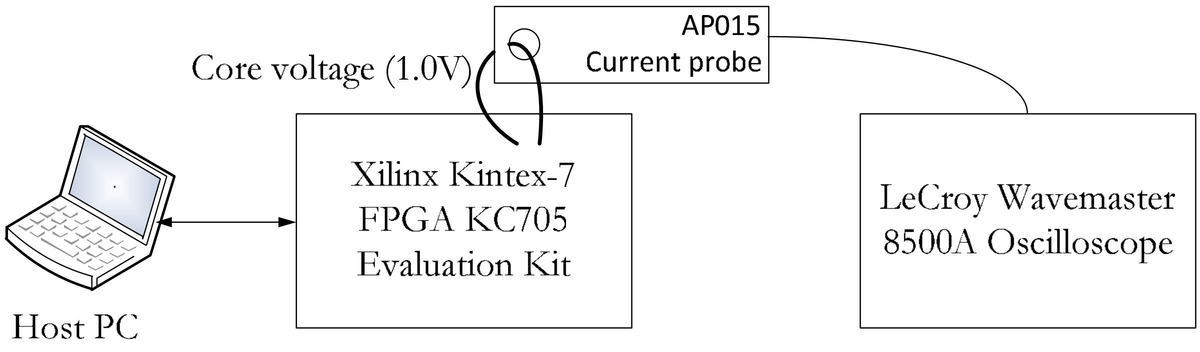


Figure 31: Artifact setup for demonstration #1

The output coil from the core voltage supply (VCCINT) is lifted and a thick wire is welded in serial with this supply. This allows a magnetic current probe to measure the current to FPGA core.

List of equipment:

- Xilinx Kintex-7 FPGA KC705 Evaluation Kit.
- LeCroy Wavemaster 8500A Oscilloscope.
- AP015 Current probe.
- Host PC.
- Hyper terminal.

The image data used is listed in Table 8.

Table 8: Image data for demonstration #1

Image name	Width [pixel]	Height [pixel]	Color [bit]	Size [kB]
Pattern	150	200	24	83.3
Fingerprint	151	190	24	84.6

The different imagess are downloaded to the FPGA. Both the encrypted and decrypted data is uploaded back to the host. The encryption key used is:
`X"000102030405060708090a0b0c0d0e0f"`

The results can be seen in Figure 32 to Figure 37.

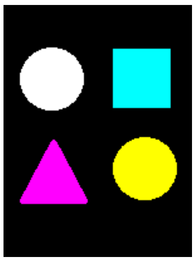


Figure 32: Pattern (original)

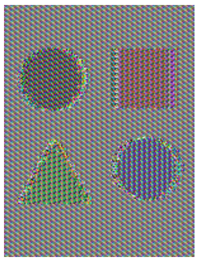


Figure 33: Pattern (encrypted)

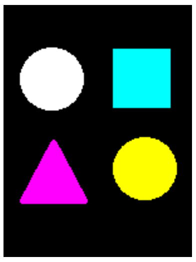


Figure 34: Pattern (decrypted)

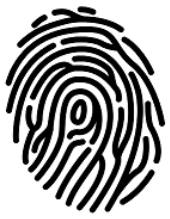


Figure 35: Fingerprint (original)

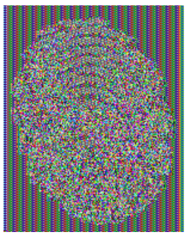


Figure 36: Fingerprint (encrypted)

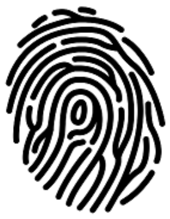


Figure 37: Fingerprint (decrypted)

The data integrity was intact; the error count was measured to be 0. The original and decrypted images was in all three cases both visually and bitwise identical.

The key performance parameters for the demonstration can be seen in Table 9.

Table 9: Performance results for demonstrations #1

FPGA modes	**R** [Gb/s]	η [Mb/slice]	t_L [clocks]	$P_{Average}$ [mW]	P_p [Gb/Ws]
AES encryption core only	37.51	10.18	20	NA	NA
AES decryption core only	1.83	1.35	22	NA	NA
AES encryption and decryption	1.83	6.4	42	106	18

5.3 Evaluation #1

This chapter describes the evaluation of the results presented in the demonstration for iteration #1. The purpose is to evaluate if the project objectives (Section 2.3) has been achieved. The evaluation is also used to identify if we were able to generate new scientific knowledge.

5.3.1 Project objectives

Implementation of AES encryption in a FPGA

The first object objective has been achieved. Both AES encryption and decryption have been implemented in a FPGA. The FPGA design has been implemented in a Kintex-7 device and the functionality has been verified.

Use biometric image data as the application

The FPGA design developed in iteration #1 is not specifically designed for biometric image data, the design is generic and any types of data can be used. Nevertheless, biometric data in form of a fingerprint and portrait has been used in demonstration. The demonstration has shown that the basic AES in electronic code book mode (ECB) is not sufficient for biometric data. The colors are scrambled in the pattern image, but the shapes are still visible after encryption (Figure 33). The outline of the fingerprint can be still seen after encryption (Figure 36).

At first glance, the results can be surprising, since 128 bit AES is considered to be a very secure encryption method. However, the electronic code book AES is a deterministic mathematical operation. A specific encryption key and a specific input vector always result in the same corresponding output vector. Therefore AES in ECB mode is not well suited for biometric data for several reasons: shapes are not fully hidden in the picture, since one color code in the bitmap is just converted to another color code.

The objective of using use biometric image data as the application has not been achieved in the first design and development iteration, since the security of the application has obvious weaknesses.

Research how to efficiently implement AES in Xilinx Kintex-7 architecture

The AES has been implemented into a Kintex-7 architecture. The implementation is not in particular efficient, the throughput is 1.83 GB/s and the efficiency is 6.4 Gb/Slice. The literature has provided examples of better throughput and efficiency with an older FPGA technology [6]. However, we have to take into consideration that the complexity of this design is greater than the designs presented in the literature review, therefore is not correct to make a direct comparison. Nevertheless the achieved clock frequency for iteration #1 was only 293 MHz and the theoretical clock speed for Kintex-7 (speed grade 2) is 650 MHz [5]. It is expected that the clock frequency can be improved if the design is optimized for speed.

The conclusion is that this objective has been achieved only partially.

Performance test throughput rate, latency, data integrity and power consumption.
This objective has been achieved. The throughput, latency, data integrity and power consumption has been measured. The performance test has been performed in real hardware, which only few researchers have attempted. The literature review has shown that most researchers only reached to the point where they had a simulation output. This is also the reason why now power measurements exist for AES encryption/decryption in FPGA. Now power consumption has been quantized in terms of Gb/s per Watt.

Create a hardware platform that can be used for further biometric information security research
This objective has been achieved. A complete hardware platform for information security research has been developed.

5.3.2 New scientific knowledge for this iteration

AES encryption/decryption has been implemented in Xilinx Kintex-7 for the first time.
Power measurements for AES encryption/decryption running in real hardware has been performed for the first time.

5.3.3 New design iteration decision

The most important problem with the design is that it is not suited for biometric image data since the information is not securely encrypted. For that reason a new design and development iteration is needed to address this problem.

5.4 Design and development #2

This chapter describes design and development iteration #2. Not all steps described in the previous iteration are redone, e.g. the system design is still the same, thus there is no need to describe it again.

The objective with this design iteration is increase the security of the AES encryption, to make the FPGA applicable for a biometric image application.

The litter review has shown an approach to this problem - Ou, Chung and Sung used image compression before the encryption, this secured that the color codes were not directly translated into another code [19]. However, if using the same encryption key, it poses a problem with identity tracking, the method protects the actual image but you can trace the identity, since the image will be encrypted exactly the same way for each transfer. This can be solved only by using an encryption key one time, but this poses a major challenge to key management.

NIST published in 2001 a method that addresses the problem of the deterministic nature of AES in ECB mode [26]. NIST defines four modes: Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB) and Counter (CTR). From hardware perspective, the CBC mode is the cheapest to implement. The principle is to form a chain, the output from the AES encryption is mixed (XOR'ed) with the input vector for the next output calculation. The output vector will therefore not only be depended on the encryption key but also on previous outputs.

For the first input vector is there no previous output vector, instead an initialization vector is used. This vector can be randomly generated for each block of data. The initialization vector is not secret; it can be attached to header of the image or transmitted separate. Server and Client can also generate the initialization vector locally in a pseudorandom way.

The approach is to redesign the FPGA to support AES in CBC mode.

5.4.1 Detailed design

There has only been made changes in the encryption and decryption core wrapper. The initialization vector has not been implemented in the register bank, since it is out of scope for this project to have initialization vector that can be changed by the host. This means that the FPGA uses a fixed initialization vector, for simulations can any vector be used, but when the FPGA is one specific vector chosen and hardcoded into the FPGA.

5.4.1.1 Encryption core

The encryption core wrapper has been redesigned in order to support AES in CBC mode. The AES core does not need to be changed, since CBC works on the input/output of the AES.

The output of AES encryption in CBC mode can be described with the following formula:

$$Q_i = AES_{Enc}(D_i \oplus Q_{i-1}), \text{ where } Q_0 = \text{Initialization vector} \quad \text{(Eq.13)}$$

The expression means that input (D_i) is exclusively or'ed with the previous output (Q_{i-1}). For the first input sample (i=1) the initialization vector is used instead of the previous output. The new wrapper is shown in Figure 38. The output value (Q_{i-1}) is stored in a D-Flip-flop. The D-Flip-flop is initialized with initialization vector on reset.

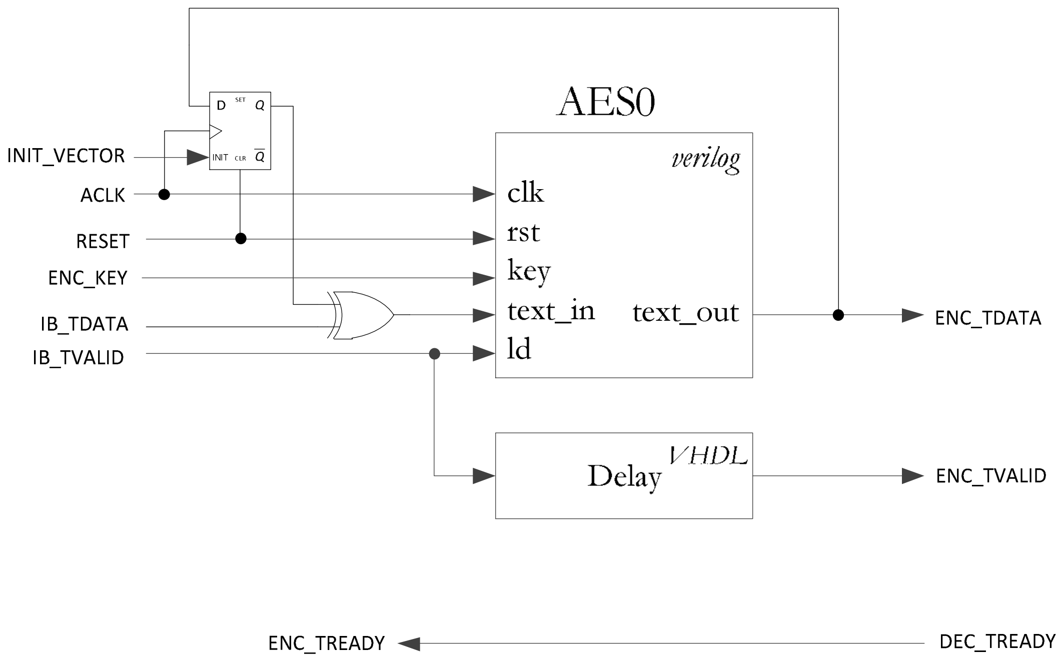


Figure 38: Encryption core wrapper

5.4.1.2 Decryption core

The decryption core wrapper has been redesigned in order to support AES in CBC mode. The AES core does not need to be changed, since CBC works on the input/output of the AES.

The output of AES decryption in CBC mode can be described with the following formula:

$$Q_i = AES_{Dec}(D_i) \oplus D_{i-1}, \text{ where } D_0 = \text{Initialization vector} \quad (\text{Eq.14})$$

This equation is the inversee operation as for AES CBC encryption, the input to the AES decoding is just the normal input (D_i), the output of the AES decoding is xor'ed with the previous input (D_{i-1}). For the first input (i=1) the initialization vector is used in the xor operation. The new wrapper is shown in Figure 39.

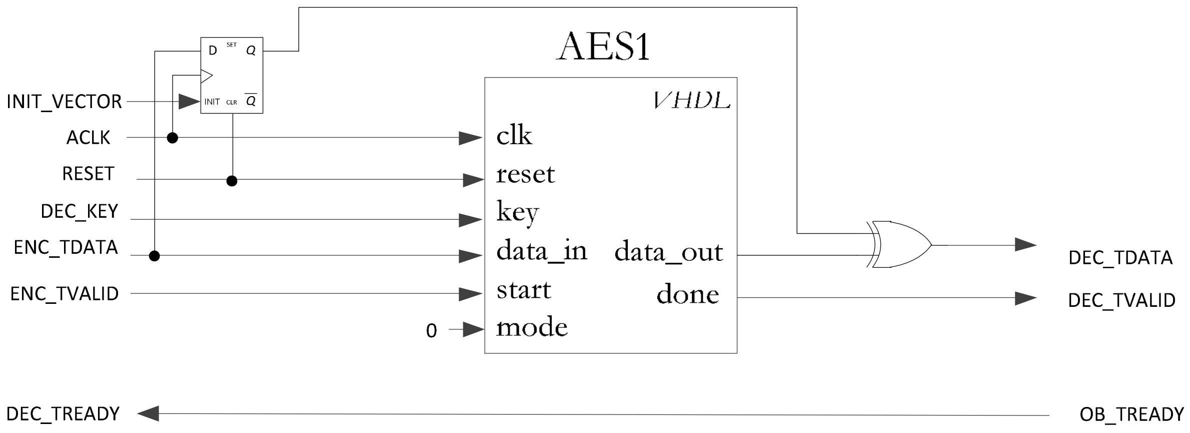


Figure 39: Decryption core wrapper

The input value (D_{i-1}) is stored in a D-Flip-flop. The D-Flip-flop is initialized with initialization vector on reset.

5.4.2 Simulation

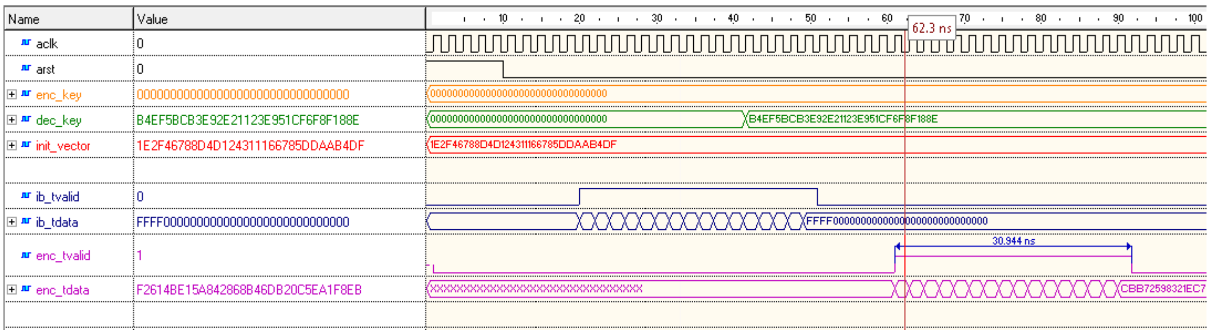
5.4.2.1 AES Core

The same KAT vectors from iteration #1, have been used for this simulation.

The initialization vector used is:

`X"1E2F46788D4D124311166785DDAAB4DF"`

Figure 40 shows AES encryption in CBC mode, the encryption key and input vectors are the same KAT vectors used in iteration #1. However the output of the encryption is completely different, the first output vector is `X"f2614be15a842868b46db20C5ea1f8eb"` as a result of the xor operation of the input.

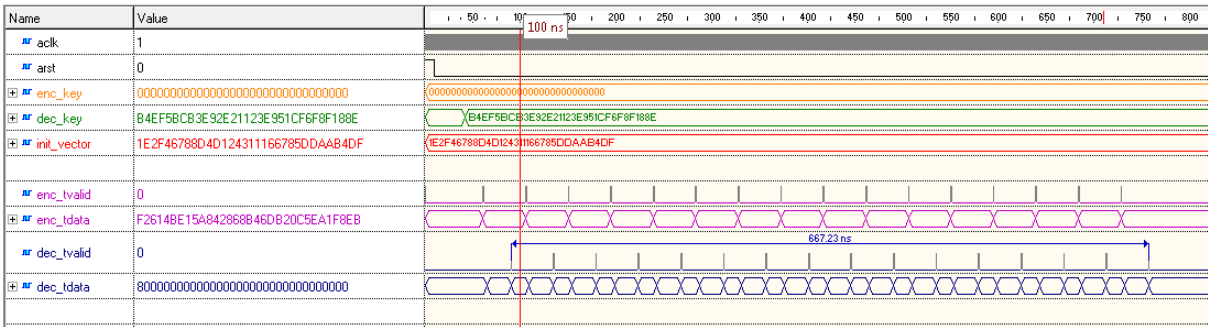


```
Name         Value
aclk         0
arst         0
enc_key      00000000000000000000000000000000
dec_key      B4EF5BCB3E92E21123E951CF6F8F188E
init_vector  1E2F46788D4D124311166785DDAAB4DF

ib_tvalid    0
ib_tdata     FFFF0000000000000000000000000000
enc_tvalid   1
enc_tdata    F2614BE15A842868B46DB20C5EA1F8EB
```

Figure 40: AES core encoder simulation result #2

Figure 41 shows the decryption, the input vectors are no longer KAT vectors, but the output is still the same as the KAT vectors. E.g. the `X"f2614be15a842868b46db20C5ea1f8eb"` vector results in `X"80000000000000000000000000"` on the output.

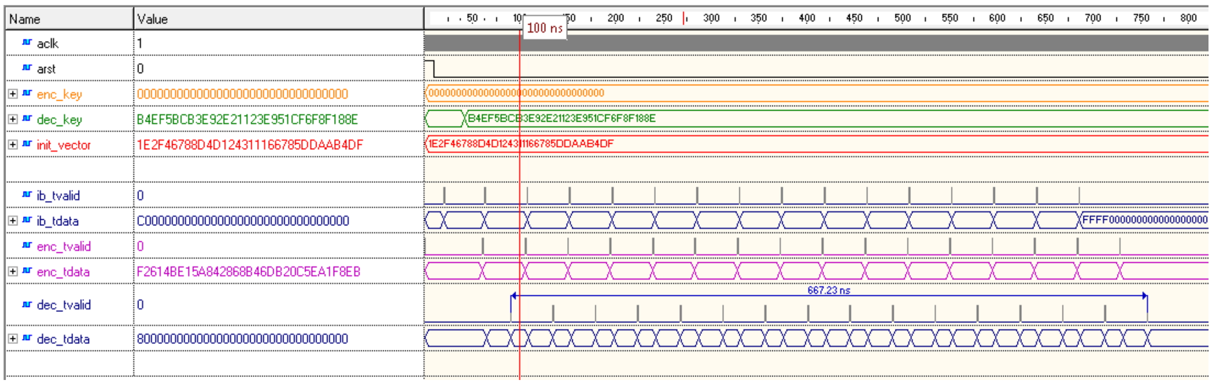


```
Name         Value
aclk         1
arst         0
enc_key      00000000000000000000000000000000
dec_key      B4EF5BCB3E92E21123E951CF6F8F188E
init_vector  1E2F46788D4D124311166785DDAAB4DF

enc_tvalid   0
enc_tdata    F2614BE15A842868B46DB20C5EA1F8EB
dec_tvalid   0
dec_tdata    80000000000000000000000000000000
```

Figure 41: AES core decoder simulation result #2

Figure 42 shows the complete AES encryption / decryption in CBC mode. All the KAT vectors input is reconstructed on the decryption modules output.



Name	Value
aclk	1
arst	0
enc_key	00000000000000000000000000000000
dec_key	B4EF5BCB3E92E21123E951CF6F8F188E
init_vector	1E2F46788D4D124311166785DDAAB4DF
ib_tvalid	0
ib_tdata	C0000000000000000000000000000000
enc_tvalid	0
enc_tdata	F2614BE15A842868B46DB20C5EA1F8EB
dec_tvalid	0
dec_tdata	80000000000000000000000000000000

Figure 42: AES core encode/decode simulation results #2

The simulation has also been tested with other initialization vectors, the performance is consistent, the original input vectors are reconstructed on the decryption output.

5.4.3 Implementation

The implementation was done with the same parameters as design iteration #1. The results are summarized in Table 10.

Table 10: Implementation of results for iteration #2

Implementation	Number of slices	Number BRAM	Achievable clock
The AES encryption core only	3812	105	3.427 ns
The AES decryption core only	1570	1	3.423 ns
The complete FPGA	6182	163	3.412 ns

All nets were successfully routed.

The bit file was successfully generated.

5.5 Demonstration #2

The same setup is used as for the demonstration in iteration #1.

The image data and encryption key are also the same, the initialization vector for CBC mode is:

`X"00000000000000000000000000000000"`

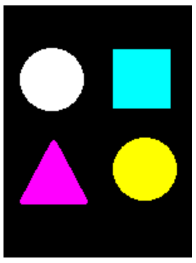


Figure 43: Pattern (original)

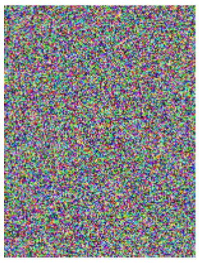


Figure 44: Pattern (cbc encrypted)

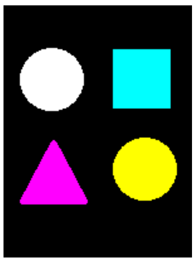


Figure 45: Pattern (cbc decrypted)

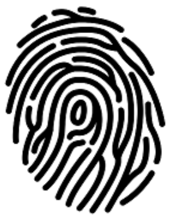


Figure 46: Fingerprint (original)

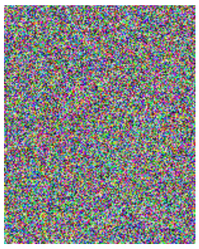


Figure 47: Fingerprint (cbc encrypted)

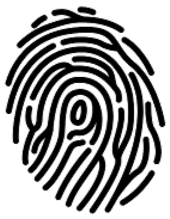


Figure 48: Fingerprint (cbc decrypted)

Data integrity was intact; the error count was measured to be 0. The original and decrypted images were in all cases both visually and bitwise identical. The performance parameters for the demonstration are listed in Table 11.

Table 11: Performance results for demonstration #2

FPGA modes	**R** [Gb/s]	η [Mb/slice]	t_L [clocks]	$P_{Average}$ [mW]	P_p [Gb/Ws]
AES encryption core only	37.35	9.80	20	NA	NA
AES decryption core only	1.70	1.08	22	NA	NA
AES encryption and decryption	1.70	0.27	42	103	17

5.6 Evaluation #2

This chapter describes evaluation of the results presented in the demonstration for iteration #2. The project objectives which were not achieved in the previous iteration will be reevaluated in this chapter. The evaluation is also used to identify if we were able to generate new scientific knowledge in this iteration.

5.6.1 Project objectives

Use biometric image data as the application

This objective has been achieved - the demonstration showed that AES in CBC mode provides a strong protection of the image information.

Research how to efficiently implement AES in Xilinx Kintex-7 architecture

The efficiency of the design was decreased; in iteration #1 the efficiency was 6.4 Gb/slice whilst in this iteration the efficiency was only 0.27 Gb/slice. The reason why this number is lower is that we used more logic and the throughput is slightly lower. This shows that it has a price in terms of efficiency to implement more security. However, the reason for the low efficiency is the decryption core, if the decryption core could run at same speed as the encryption core the efficiency would drastically increase. Therefore a better decryption core is needed to fully achieve this goal.

5.6.2 New scientific knowledge for this iteration

AES encryption/decryption in CBC mode was implemented in a Xilinx Kintex-7 FPGA for the first time. Power measurements for AES encryption/decryption running in a real FPGA were performed for the first time. FPGA hardware platform aimed for information security research, with biometric image as the application were developed and verified.

5.6.3 New design iteration decision

The project could end at this point. We have generated new scientific knowledge which could be a valued contribution to the field of information security. The only objective which has not been fulfilled is how to research how to efficiently implement AES in Xilinx Kintex-7 architecture. We have definitely done research, yet there are good reasons to believe that the results could be improved by redesigning the decryption core. The performance could also be improved by making speed, area and power optimization on the overall FPGA design. Therefore a new design iteration is initiated, with the objective to optimize the performance of the FPGA.

5.7 Design and development #3

This chapter describes design and development iteration #3. The objective with this design iteration is to increase the efficiency of the design. There are two options for increasing the efficiency: either to increase the throughput or decrease the area (or do both).

Previous evaluation showed that very low efficiency is caused by the low throughput of the AES decryption core. The AES decryption core has a throughput of 1.70 GB/s, this is very low compared to the AES encryption core which has throughput of 37.35 GB/s. The AES decryption core uses only 1570 slices compared to 3812 for the AES encryption core. Since the AES decryption core is already relatively compact, the best way to optimize the design will be to increase the throughput of the AES decryption core.

The best way to start is to study why Liu, Xu and Yuan [6] AES encryption is so efficient. The reason why the throughput is so high is because they have been successful in designing a fully pipelined architecture for the encryption with a data rate of 1/1. This means that for each clock cycle is the core able to process a 128 bit data block. If we compare this to the data rate of the AES decryption core which was 1/22, it is apparent that that the fully pipelined architecture will be a factor 22 faster at the same clock speed.

The best choice to increase the efficiency of the FPGA is to design a fully pipelined architecture for the AES decryption. Liu, Xu and Yuan [6] did only design the AES encryption core, but the general idea to fully unroll the matrix operations can be reused in this project.

5.7.1 Detailed design

In this section we will describe the detailed design for the new AES decryption core. As described in section 3.4 Theory of operation, the AES decryption is the same series of matrix operations as for AES encryption. However the matrix operations have to be inverse (except for the Add round key), for that reason the modules in the AES encryption core cannot be reused. The inverse matrix operations have to be designed and placed in the order specified in section 3.4.2 AES Decryption.

5.7.1.1 Decryption core

The structure of the AES decryption is described in section 3.4.2 AES Decryption. In order to design a fully pipelined structure we need to unroll all ten rounds. This means that each matrix operation, for each round, will have its own dedicated logic resources. In this way it is possible to achieve a data rate of 1/1, since each data block will pipeline through the structure and will not have to wait for the previous data block to finish. The top level design for the fully pipelined AES decryption core can be seen in Figure 49.

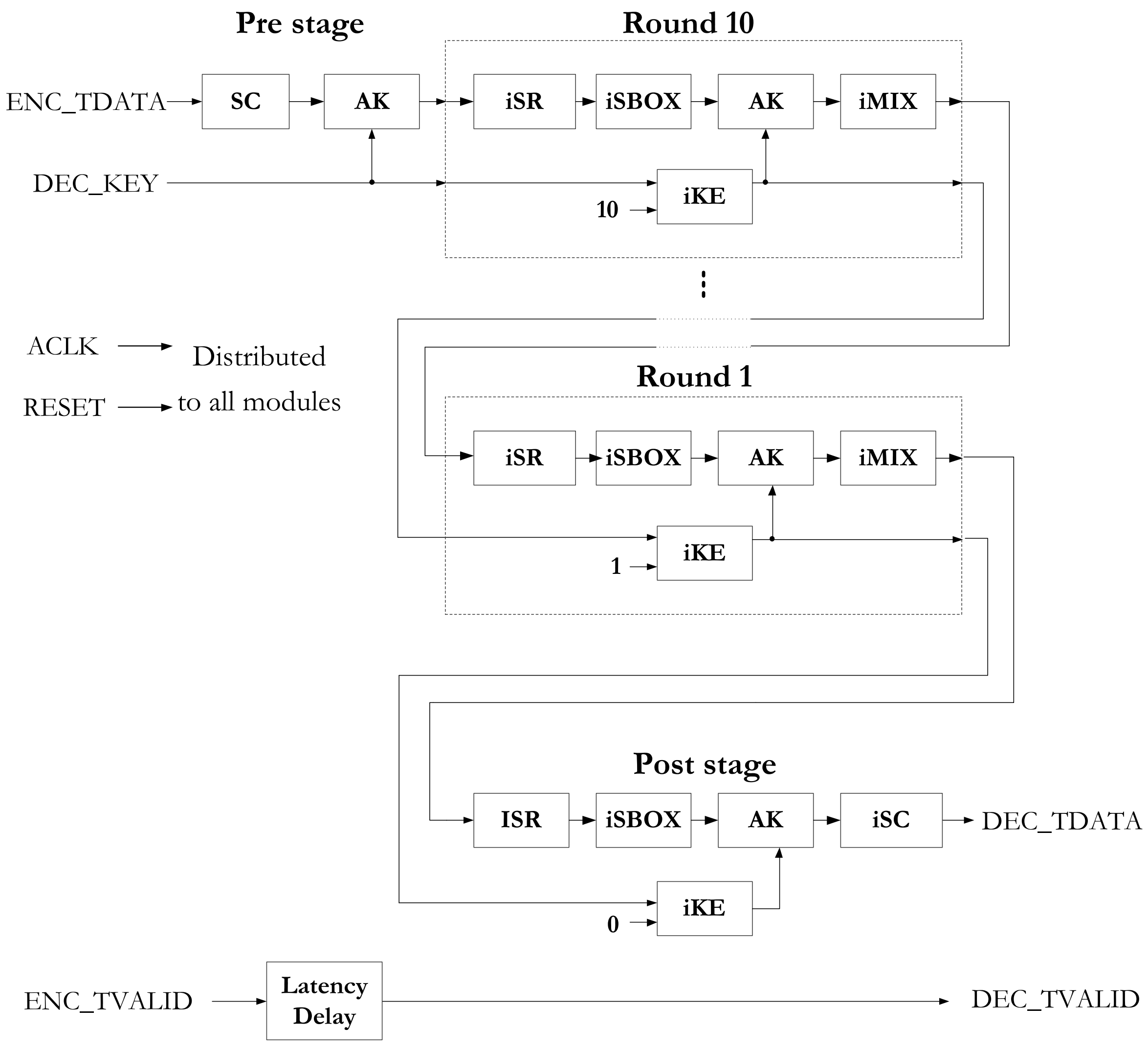


Figure 49: Fully pipelined AES decryption

The pre stage, round 10 down 1 and the post stage has its own dedicated logic, notice that Figure 49 only shows round 10 and round 1, in between is round 9 down to 2.

The key schedule can either be placed in a separate module, since a typical AES encryption will use the same key for many data blocks, however, by placing a key expander in each of the stages it will be possible to change the key for each data block, since the key will pipeline through the structure aligned with the data blocks.

The design also features a tvalid signal to support AXI4-Stream interface, the tvalid signal could be pipelined through the rounds aligned with the data blocks. However it is much simpler to use a shift register to implement a latency delay. This construction assures that tvalid will go high when the first valid data block appears on the output. The decryption core wrapper has already been designed to support CBC mode, therefore it is not necessary to include CBC logic in the core.

5.7.1.2 State conversion (iSC)

The treatment of a logic vector as a matrix is layer of abstraction in the VHDL coding. Therefore the state conversion does not require any logic to implement, all bits are practically implemented as separate wires, but defining a group of signals as a matrix allows us to write the VHDL code on a higher abstraction layer. The following VHDL example shows the conversion:

```
for row in 0 to 3 loop
   for col in 0 to 3 loop
      Matrix (col,row)<=Vector((row*32)+(col*8+7) downto (row*32)+(col*8));
   end loop;
end loop;
```

Notice that this is not a sequential statement like other programming languages, it requires no clock cycles to perform this operation, it is merely a mapping of wires.

5.7.1.3 Inverse State conversion (iSC)

The inverse state conversion has the same properties as state conversion; the wire mapping is just in opposite direction. The following VHDL example shows the conversion:

```
for row in 0 to 3 loop
   for col in 0 to 3 loop
      Vector((row*32)+(col*8+7) downto (row*32)+(col*8))<=Matrix (col,row);
   end loop;
end loop;
```

5.7.1.4 Add round key (AK)

The Add key operation is simply a XOR operation between the state matrix and the key (Figure 50). In order to optimize the timing of the FPGA, flip flop is placed after the operation. This flip flop will be free in terms of slices, since each XOR gate will be placed in a FPGA slice which already houses an output flip flop. The only drawback is that one is added to the module latency; however this will not influence the throughput.

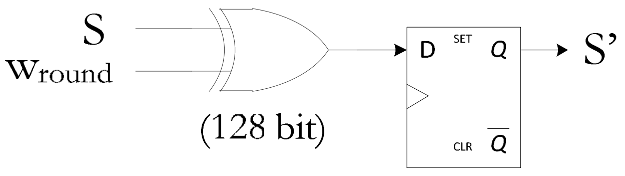


Figure 50: Add round key logic

5.7.1.5 Inverse shift row (iSR)

This operation does not require any logic to implement. The rotate operations in shift row are completely static, since all input always rotates in the exact same way. Therefore the shift row operation is rather a specification how the wires between the Add key module and inverse SBOX module should be connected.

5.7.1.6 Inverse substitution box (iSBOX)

The inverse substitution box is implemented as a 256 x 8 bit look up table, since the state matrix is 128 bit is 16 look up table needed for each iSBOX module. Because of the fully pipelined architecture 11 instances iSBOX are needed, one for each round and one for the post stage.

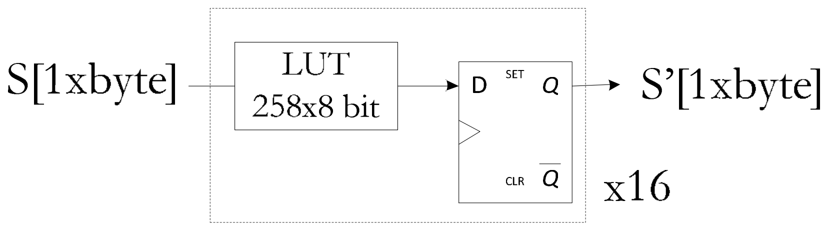


Figure 51: Inverse substitution box logic

The look up table is an exact implementation of the values specified by NIST, which can be found in the Appendix A.

5.7.1.7 Inverse mix column (iMIX)

The inverse mix column operations implements the matrix multiplication described in 3.4 Theory of operation. The implemented equations for one column can be seen in Figure 52. The calculation is done each for all 4 columns in parallel.

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

Figure 52: Inverse mix column logic equations [1]

Notice that this calculation does not require logic multipliers, since all multiplication has one variable and one constant. Therefore the synthesize engine will be able to implement this construction very cheap with XOR gates.

5.7.2 Inverse key expander (iKE)

The implementation of the inverse key can be seen in Figure 53, the picture only shows the logic for 4 bytes, the structure is repeated 4 times in order to support a 128 bit key. The round constant is implemented as a 10x8 bit look up table. The content is calculated based on by equation 10 in section 3.4.14 Round constant (RCON), the result of the calculation can be seen in Appendix B.

The iROT is a free operation; it only specifies the mapping between the XOR gate output and the input to the inverse substitution. A pipeline register is placed on the output in order to align the key with the Add round key module.

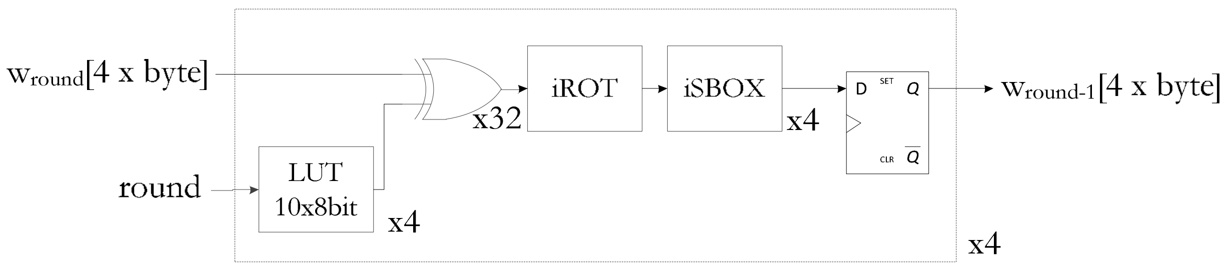


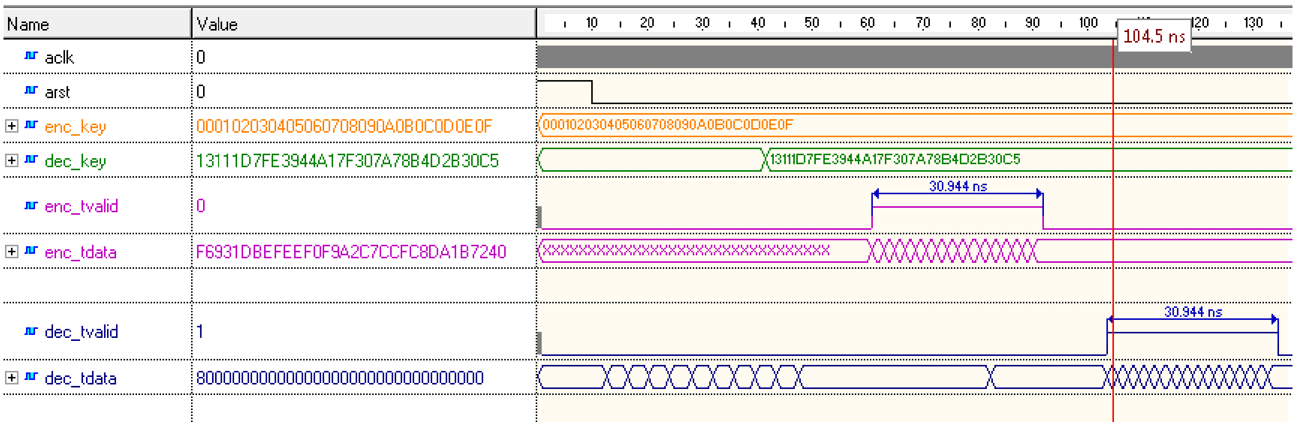
Figure 53: Inverse key expander logic

5.7.3 Simulation

5.7.3.1 AES Decryption core

The same KAT vectors from iteration #1 and #2, have been used for this simulation. This simulation is not using the CBC mode, since this functionality is already tested and is independent of the AES core.

Figure 54 shows the simulation of the designed decryption core, the enc_tvalid signal shows that the input is at full data rate (128 bit data is processed at each clock), the input consist of 16x128 bit. The dec_tvalid signal is the output from the decryption core is also at full data rate, it takes 30.944 ns to process the input. This corresponds to 66.18 Gb/s, which matches Liu, Xu and Yuan [6] AES encryption implementation.



Name	Value
aclk	0
arst	0
enc_key	000102030405060708090A0B0C0D0E0F
dec_key	13111D7FE3944A17F307A78B4D2B30C5
enc_tvalid	0
enc_tdata	F6931DBEFEEF0F9A2C7CCFC8DA1B7240
dec_tvalid	1
dec_tdata	80000000000000000000000000000000

10
20
30
40
50
60
70
80
90
100
120
130
104.5 ns
000102030405060708090A0B0C0D0E0F
13111D7FE3944A17F307A78B4D2B30C5
30.944 ns
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
30.944 ns

Figure 54: AES core decoder simulation result #3

Figure 54 also shows that the data integrity is intact, the first KAT vector is decrypted back to its original value, the output of dec_tdata is: `X"800000000000000000000000"`. The rest of the KAT vectors was also decoded correct

5.7.4 Implementation

The implementation was changed for this iteration in order to increase the efficiency. The implementation engine was set to optimize for speed, optimization effort was set to high and extra effort was enabled. The results are summarized in Table 12.

Table 12: Implementation of results for iteration #3

Implementation	Number of slices	Number BRAM	Achievable clock
The AES encryption core only	3812	105	3.120 ns
The AES decryption core only	3520	105	3.101 ns
The complete FPGA	7774	268	3.125 ns

All nets were successfully routed.

The bit file was successfully generated.

5.8 Demonstration #3

The same setup is used as for the demonstration in iteration #1 and #2. The initialization vector for CBC mode is `X"00000000000000000000000000000000"`.

The same images used in demonstration #2 has been tested with the new AES decryption core, the result is identical to demonstration #2. It does not add much value to show the same images again, therefor is a set of new test images (Table 13) used for the final demonstration. The images are sourced from the open source forensic data base [27] and used under the license of fair use.

Table 13: Image data for demonstration #3

Image name	Width [pixel]	Height [pixel]	Color [bit]	Size [kB]
Fingerprint 2	129	191	24	72.4
Iris	200	127	24	74.4

Figure 55 to Figure 60 demonstrate that the designed decryption core function correctly.

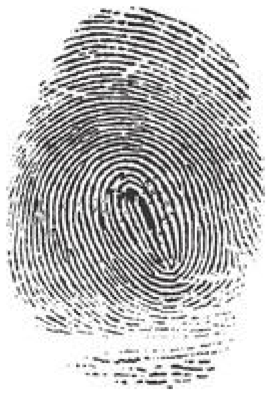


Figure 55: Fingerprint 2 (original)

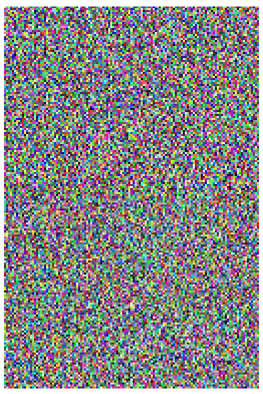


Figure 56: Fingerprint 2 (cbc encrypted)

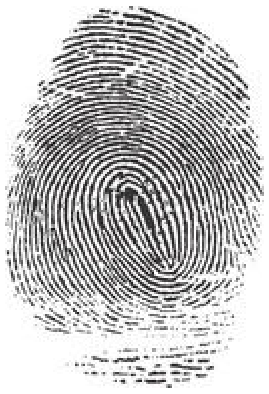


Figure 57: Fingerprint 2 (cbc decrypted)

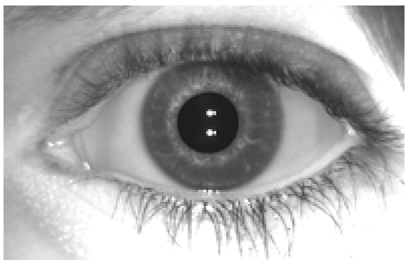


Figure 58: Iris (original)

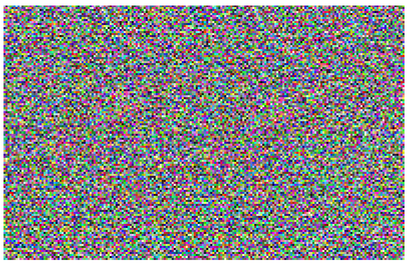


Figure 59: Iris (cbc encrypted)

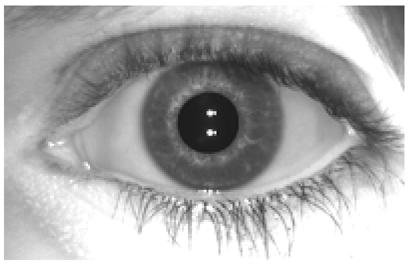


Figure 60: Iris (cbc decrypted)

Data integrity was intact; the error count was measured to be 0. The original and decrypted images were in all three cases both visually and bitwise identical. The performance parameters for the demonstration are listed in Table 14.

Table 14: Performance results for demonstration #3

FPGA modes	**R** **[Gb/s]**	η **[Mb/slice]**	t_L **[clocks]**	$P_{Average}$ **[mW]**	P_p **[Gb/Ws]**
AES encryption core only	41.02	10.76	20	NA	NA
AES decryption core only	41.28	11.72	22	NA	NA
AES encryption and decryption	40.96	5.27	42	143	286

5.9 Evaluation #3

This chapter describes evaluation of the results presented in demonstration #3. The project objectives of how to efficiently implement AES in Kintex-7 architecture was not fully achieved in Evaluation #2, this objective will be reevaluated in this chapter, based on the new results. The evaluation is also used to identify if we were able to generate new scientific knowledge-

5.9.1 Project objectives

Research how to efficiently implement AES in Xilinx Kintex-7 architecture

The efficiency of the design has been increased significantly; the efficiency is 5.27 Mb/slice compared to 0.27 Gb/slice in the previous iteration, which correspond to a factor 20 improvement of the efficiency. The fully pipelined AES decryption core does use more logic, 3520 slices compared to 1517 slices in the previous iterations, but the efficiency is still remarkably improved since the throughput is much higher.

If we compare this to the state of the art, then is two designs more efficient: Liu, Xu and Yuan achieved 19.20 Mb/slice and Rahimunnisa et. al achieved 22.94 Mb/slice. However it is not applicable to do a direct comparing of the efficiency parameter, since they have only implemented AES encryption (you need approximate twice the logic to have both encryption and decryption), there was no support logic and the design used ECB mode only. Both designs was only simulated, therefore is it not proven that an efficiency of e.g. 22.94 Mb/slice is practically possible. This project on the other hand has proven that a full AES encryption/ decryption running 40 GS/s with an efficiency of 5.27 Mb/slice is practically possible to implement in a FPGA.

The fully pipelined AES decryption core has caused an increase in power which is a natural consequence of have more logic implemented. However the overall power performance is increased due to high throughput. The power performance is 286 GB/Ws, which mean that you get 286 GB/s of throughput for each watt you spend. The state of the art provides only one case for comparison, Karimian, Rashidi and Farmani estimated the power performance for their design to be 2.05 GB/Ws. This project as provided a solutions that is a least a factor 100 better. But more important, the power performance has been quantized, so any future project will have a measurement from a practical implementation to compare with.

5.9.2 New scientific knowledge for this iteration

For the first time has a fully pipelined AES encryption and decryption core in CBC mode been implemented in a Xilinx Kintex-7 FPGA. The performance achieved is 40 GB/s throughput, 5.27 Mb/slice efficiency with a power performance of 286 GB/Ws. The throughput and efficiency is not the highest achieved. However it is the highest achieved for a real FPGA implementation. Further has the power performance been measured and can be used to benchmark other designs in the future.

The FPGA design developed in the project is a new valuable artifact for Information security research. The artifact has demonstrated its value through design and development iteration #2 and #3, implementing and testing new information security features is very simple once the FPGA platform is built. This artifact will enable the researcher to generate new knowledge faster and more efficiently.

5.9.3 New design iteration decision

All project objectives have been fulfilled; therefore no new design iterations will be initiated.

6 Conclusion

The result of the project objectives are summarized in Table 15. All the objectives were achieved. The most challenging objective was to research how to efficiently implement AES in Kintex-7. The solution was to use a fully pipelined AES design for both encryption and decryption. Liu, Xu and Yuan [7] solved this problem for encryption in 2013. In this project have we solved the problem for decryption - which is an important scientific contribution.

Table 15: Project achievements

Objective	Achieved
Implementation of AES encryption in a FPGA.	Yes
Use biometric image data as the application.	Yes
Research how to efficiently implement AES in Xilinx Kintex-7 architecture.	Yes
Performance test throughput rate, latency, data integrity and power consumption.	Yes
Create a hardware platform for biometric information security research.	Yes

The project has also demonstrated the major weakness of AES in ECB mode; the deterministic output vectors are not suited for s image application. There are different approaches to solve this problem, in this project was AES CBC mode chosen, since it is a cheap simple and efficient solution. The project has showed that CBC mode is a good choice for image applications.

The project has produced a valuable artifact in form of a FPGA hardware platform which can be used for further information security research. Many before have implemented AES in a FPGA, yet the literature review identified a gap in knowledge, no attention was paid to the power performance of the AES FPGA implementation. The developed platform provides an excellent opportunity for further research in power consumption. The hardware platform is not only limited to AES encryption other encryptions algorithms could be implemented as well. The platform would be a great assert for developing new encryption algorithm standards.

The knowledge contribution of this thesis includes not only how to implement AES encryption/ decryption efficiently in a Xilinx Kintex-7, but also how to implemented the algorithm in a real physical FPGA. The research has shown that it is possible to simulate and achieve 66 GB/s throughput, but as soon as you implements the design with a usable interface and supporting logic then will the throughput rate will drop. This has learned us, that when reviewing a scientific paper, the throughput rate should be treated as the maximum theoretical rate, and not the rate one practically can expect to run in hardware.

The power measurements in this paper can be used both to benchmark new AES implementation methods, but also to have a quantified number of how much power it costs to add AES encryption to a data stream in a FPGA. This information can be valuable when designing a power supply for a FPGA or upgrading an existing FPGA design with AES encryption.

6.1 Reflections

There are many things aspects of this project which could be explored further. However a master thesis paper has a limited scope and it is not possible to research everything. The throughput achieved in this project is not nearly as high as what others has done, the reasons for this has been stated. Nevertheless the design developed in this project has the potential to exceed the performance of the competing design, since no clock speed optimizations has been performed. The project was delimitated not to optimize the FPGA for area or clock speed, but the Kintex-7 device should be cable of running at least with 40% higher clock speed and thereby exceeding the performance of the competing designs. This can be done e.g. by inserting pipeline registers between every matrix operation in the AES algorithm, this should increase the clock speed significantly. However this will cost more area, therefor is it not certain that the efficiency will increase. But this project has learned us that increasing the throughput is good way to increase the efficiency.

Another approach could be to reduce the area of the FPGA. The most logic consuming module is the substitution look up tables. With the current implementation is one invers substitution look up table used for each round and one substitution look up tables is used for each key expander. The key expander could be implemented as a separate key engine, since one key expander could sequential calculate the keys for each round - this alone will save 10 substitution look up tables.

The communication with the FPGA is rather difficult for the user, since there has not been developed any software application to run on the host PC, which was out of scope for this project. The development of a real software application with a graphical user interface will greatly improve the usability of the FPGA hardware platform.

The FPGA design has a major limitation; it can only handle image sizes of maximum 132 kb. This could be improved by using external memory (the Xilinx Kintex-7 evaluation board has

4GB external memory). However the input/output buffers is only present to compensate for the slow RS232 connection to the Host. The Xilinx Kintex-7 FPGA supports PCIe interface, the Xilinx Kintex-7 evaluation board can be inserted directly into a PCIe x16 slot on the host PC. If the FPGA is redesigned to support PCIe, then would it be possible to stream data directly to the FPGA and avoiding the use of large buffers. This would also enable the PC to use the FPGA as a cryptography co-processor, which could be very valuable asset both for research and commercial use.

6.2 Further research

This section provide a list of topics where the FPGA hardware platform can used for further research and development:

- Increase throughput of the AES core.
- Reduce the area of the AES core.
- Increase the power performance.
- Implement variable initialization vectors.
- Implement other AES mode, such as propagating cipher-block or cipher feedback.
- Implementing other standard cryptography cores such as 3DES.
- Develop host application software.
- Develop a high-speed PCIe interface for fast host communication

7 References

[1] National Institute of Standards and Technology (NIST), Advance Encryption Standard (AES), Processing Standards Publication 197, pp. 5-26, 2001.

[2] Moon D., Chung Y., Moon K., Pan S., “Secure and Efficient Transmissions of Fingerprint Images for Embedded Processors Image Analysis and Recognition,” Lecture Notes in Computer Science, vol. 3656, pp. 1106-1117, 2005.

[3] Akman Y. and Yerlikaya T., “Encryption Time Comparison of AES on FPGA and Computer,” Advances in Intelligent Systems and Computing, vol. 225, pp. 317-324, 2013.

[4] Magdaleno E. and Rodríguez M., “Acceleration of Computation Speed for Wavefront Phase Recovery Using Programmable Logic,” Topics in Adaptive Optics, p. 210, 2012.

[5] Xilinx, “Kintex-7 Family Overview,” Xilinx publications, pp 1-2, 2013.

[6] Rahimunnisa K. and Karthigaikumar P., “PSP: Parallel sub-pipelined architecture for high throughput AES on FPGA and ASIC,” Central European Journal of Computer Science, vol. 3, no. 4, pp. 173-186, 2013.

[7] Liu Q., Xu Z., Yuan Y., ”A 66.1 Gbps single-pipeline AES on FPGA. Field-Programmable Technology (FPT),” in Proc. IEEE International Conference on Field Programmable Technology, Kyota, pp 378-81, 2013.

[8] Cai X., Sun R. and Liu J., “An Ultrahigh Speed AES Processor Method Based on FPGA,” in Proc. 5th IEEE International Conference on Intelligent Networking and Collaborative Systems (INCoS), pp. 633–636,2013.

[9] Kumar S., Sharma V.K. and Mahapatra K.K.., “An Improved VLSI Architecture of S-box for AES Encryption,” IEEE International Conference on Communication Systems and Network Technologies (CSNT), pp. 753-756, 2013.

[10] Dogan A., Ors S.B. and Saldamli G., “Analyzing and comparing the AES architectures for their power consumption,” Journal of Intelligent Manufacturing, vol. 25, no. 2, pp. 263-271, 2012.

[11] Singh B., Kaur H., Monga H., “FPGA Implementation of AES Co-processor in Counter Mode,” Communications in Computer and Information Science, vol. 70, pp. 491-496, 2010.

[12] Hodjat A. and Verbauwhede I., “A 21.54 Gbits/s fully pipelined AES processor on FPGA,” in Proc. 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, CA, pp. 1-2, 2004.

[13] Liua H., zhoub Y., Fanc Y., Tsunood Y. and Gotoa S., “Information Hiding for AES Core Based on Randomness,” Procedia Engineering, vol. 15, pp. 2113–2117, 2011.

[14] Dyken J. and Delgado-Frias J., ”FPGA schemes for minimizing the power-throughput trade-off in executing the Advanced Encryption Standard algorithm,” Journal of Systems Architecture, vol. 56, no. 2, pp. 116–123, 2010.

[15] Kumar Y. and Purohit P., “Hardware Implementation of Advanced Encryption Standard” in Proc. International Conference on Computational Intelligence and Communication Networks, Bhopal, pp.440-442, 2010.

[16] Saqib N.A., Henriquez F.R. and Diaz-Pirez A., “AES Algorithm Implementation-An efficient approach for Sequential and Pipeline Architectures,” in Proc. 4th Mexican International Conference on Computer Science, Apizaco, p. 126, 2003.

[17] Chang K.H., Chen Y.C., Hsieh C.C., Huang C.W. and Chang C.J., “Embedded a low area 32-bit AES for image encryption/decryption application” in Proc. IEEE International Symposium on Circuits and Systems, Seoul, pp. 1922–1925, 2009.

[18] Huang C.W., Kuo S.W, Chang C.J., “Embedded 8-bit AES in wireless Bluetooth application,” in Proc. IEEE International conference on System Science and Engineering (ICSSE), Budapest, pp. 87-92, 2013.

[19] Ou S.C., Chung H.Y. and Sung W.T., “Improving the compression and encryption of images using FPGA-based cryptosystems,” Multimedia Tools and Application, vol. 28, no. 1, pp. 5-22, 2006.

[20] Gupta A., Ahmad A., Sharif M.S. and Amira A., “Rapid prototyping of AES encryption for wireless communication system on FPGA,” in Proc. 15th IEEE International Symposium on Consumer Electronics (ISCE), Singapore, pp. 571–575, 2011.

[21] Gore M. and Deotare V., “FPGA Implementation of Area Optimized AES for Image Encryption/Decryption Process,” Design and reuse online journal. p. 1, 2013.

[22] Manoj B. and Manjula N., “Image Encryption and Decryption using AES. International,” Journal of Engineering and Advanced Technology (IJEAT), vol. 5, pp. 2108-2112, 2012.

[23] Karimian G.H., Rashidi B. and Farmani A., “A High Speed and Low Power Image Encryption with 128-Bit AES Algorithm,” International Journal of Computer & Electrical Engineering, vol. 4, no. 3, pp. 367-372, 2012.

[24] Satyanarayana H., ”Fast AES-128 Encryption,” Open source project, Open cores, 2010.

[25] Peffers K., Tuunanen T., Rothenberger M.A. and Chatterjee S. A., “A Design Science Research Methodology for Information Systems Research,” Journal of Management Information Systems, vol. 24, no. 3, pp 45–77, 2012.

[26] Dworkin, M., "Recommendation for Block Cipher Modes of Operation. Methods and Techniques 2001 Edition. Computer security," NIST special publication 800-38A, pp. 9-15, 2001.

[27] Forensic Informatics Biometric Repository, "The open source forensic database", 2014.

Appendix A

Substitution lookup table

		Least significant 4 bits of input															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**b**	**c**	**d**	**e**	**f**
Most significant 4 bits of input	**0**	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	Be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	Da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	A	e0	32	3a	0a	49	06	24	5c	c2	d3	Ac	62	91	95	e4	79
	B	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	C	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	D	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	E	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Inverse substitution lookup table

		Least significant 4 bits of input															
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
Most significant 4 bits of input	**0**	52	9	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	8	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	0	8c	bc	d3	0a	f7	e4	58	5	b8	b3	45	6
	7	d0	2c	1e	8f	ca	3f	0f	2	c1	af	bd	3	1	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	A	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	B	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	C	1f	dd	a8	33	88	7	c7	31	b1	12	10	59	27	80	ec	5f
	D	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	E	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	4	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Appendix B

RCON Look up table

Input	**Output**
00	01
01	02
02	04
03	08
04	10
05	20
06	40
07	80
08	1b
09	36
others	00

Appendix C

Detailed interface specification

Host interface – Input buffer (i1)

Signal name	Bits	Direction	Description
HOST_TDATA	256	HI->IB	Host data bus
HOST_TVALID	1	HI->IB	Host valid signal
HOST_TLAST	1	HI->IB	Host last signal
IB_TREADY	1	IB-> HI	Input buffer ready signal

Input buffer – Encryption core (i2)

Signal name	Bits	Direction	Description
IB_TDATA	256	IB->ENC	Input buffer data
IB_TVALID	1	IB-> ENC	Input buffer valid signal
IB_TLAST	1	IB-> ENC	Input buffer last signal
ENC_TREADY	*1*	*ENC -> IB*	*Encryption ready signal*

Encryption core – Decryption core (i3)

Signal name	Bits	Direction	Description
ENC_TDATA	256	ENC->DEC	Encryption data
ENC_TVALID	1	ENC-> DEC	Encryption valid signal
ENC_TLAST	1	ENC-> DEC	Encryption last signal
DEC_TVALID	*1*	*DEC -> ENC*	*Decryption ready valid*

Decryption core – Output buffer (i4)

Signal name	Bits	Direction	Description
DEC_TDATA	256	DEC->OB	Decryption data
DEC_TVALID	1	DEC-> OB	Decryption valid signal
DEC_TLAST	1	DEC-> OB	Decryption last signal
OB_TREADY	*1*	*OB ->DEC*	*Output buffer ready signal*

Output buffer – Host interface (i5)

Signal name	Bits	Direction	Description
OB_TDATA	256	OB-> HI	Ouput buffer data
OB _TVALID	1	OB-> HI	Ouput buffer valid signal
OB _TLAST	1	OB-> HI	Ouput buffer last signal
HI_TREADY	1	HI->OB	Host interface ready signal

Host interface – Register bank (i6)

Signal name	Bits	Direction	Description
HI_AWDATA	4	HI->RB	Address write data
HI_AWVALID	1	HI->RB	Address write valid signal
RB_AWREADY	1	RB->HI	Address write ready signal
HI_ARDATA	4	HI->RB	Address read data
HI_ARVALID	1	HI->RB	Address read valid signal
RB_ARREADY	1	RB->HI	Address read ready signal
HI_WDATA	8	HI->RB	Write data
HI_WVALID	1	HI->RB	Address read valid signal
RB_WREADY	1	RB->HI	Address read ready signal
RB_RDATA	8	RB->HI	Read data
RB_RVALID	1	RB->HI	Read valid signal
HI_RREADY	1	HI->RB	Read ready signal

Analysis monitor – Host interface (i7)

Signal name	Bits	Direction	Description
STATUS_0	32	AM->RB	Status register 0
STATUS_1	32	AM->RB	Status register 1
STATUS_2	32	AM->RB	Status register 2

Register bank – Encryption core (i8)

Signal name	Bits	Direction	Description
Encryption_key	256	RB->ENC	Encryption master key

Register bank – Decryption core (i9)

Signal name	Bits	Direction	Description
Decryption_key	256	RB->ENC	Decryption master key

Register bank – Input buffer (i10)

Signal name	Bits	Direction	Description
Run	1	RB->IB	Start streaming data

Host interface – Host (i11)

Signal name	Bits	Direction	Description
TX	1	HI->HO	Transmit host data
RX	1	HO->HI	Receive host data