IMPLEMENTATION OF A SOFT CORE PROCESSOR ON A FPGA

WOO CHI LIANG

A project report submitted in partial fulfillment of the requirements for the award of the degree of Bachelor (Hons.) of Electronic Engineering

Faculty of Engineering and Science
University Tunku Abdul Rahman

June 2011

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	___________________________
Name	:	WOO CHI LIANG
ID No.	:	07UEB06313
Date	:	13th MAY 2011

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"IMPLEMENTATION OF A SOFT CORE PROCESSOR ON A FPGA"** was prepared by **WOO CHI LIANG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	___________________________
Supervisor	:	Dr Lo Fook Loong
Date	:	___________________________

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Lo Fook Loong for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my previous supervisor Miss Florence Choong Chiao Mei that provides me guidance and reference sources throughout my project. Experiences that been shared her innovated me in designing my project.

I also would like to show my appreciation towards Dr. Goi Bok Min for his reference sources and guidance towards AES system. His explanation provides me a better understanding about the system.

Last but not least, I would like to thanks DreamCatcher for providing me tools and training for my project. With their guideline and training, my project progress smoothly.

IMPLEMENTATION OF A SOFT CORE PROCESSOR ON A FPGA

ABSTRACT

In today's modern, FPGAs has comes with embedded soft-core that can be customized for given application and synthesized for an FPGA target. In many applications, soft-core processors provide several advantages over custom designed processor such as cost, flexibility, platform independence and greater immunity to obsolescence. On the other hand, with today's sensitivity of data and privacy, cryptology had become a demanding application. The latest cryptology that been proven to be most efficient and effective is AES (Advance Encryption Standard). AES or Rijandael algorithm is propose by two Belgian cryptographers, Joan Daemen and Vincent Rijmen to NIST (National Institute of Standards and Technology) when a new standard of encryption is request. However, due to the growing of the mass of our data, process for AES encryption and decryption come into the problem. AES algorithm mostly was performed in software platform which will take long time of processing. In this paper, the combination of hardware and software implementation on AES algorithm will be discussed. Several version of hardware and software co-design have been introduced to the market lately, these implementation will be review and discuss on their implementation method, theory, and complexity of the implementation. As the growing of the soft-core of the FPGAs, it is expected that the usage of it customizable characteristic would make the soft-core processor to be more widespread and involve in complexity embedded system in the future.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

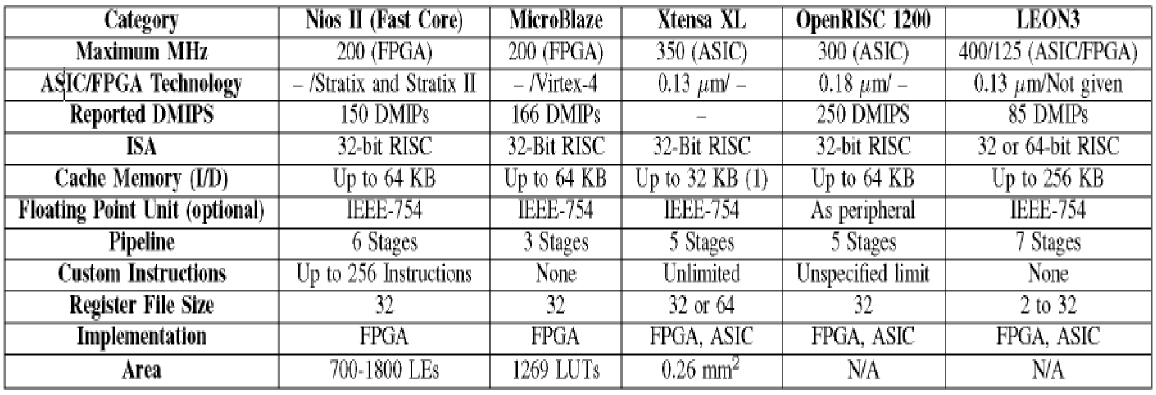
1.1 Background

In today's modern, flexibility plays an important role for dynamic and unforeseen changes in the product. According to Ralf Joost and Ralf Salomon (2005), nowadays FPGAs (Field Programmable Gate-Arrays) with high performance, reasonable price and adaptable are demanding the market. As we know, the configuration of FPGAs is described in abstract hardware description language such as verilog and VHDL; the system can be easily modified whenever is required.

However, in compete with application-specified microcontroller; FPGAs still could not reach the propagation. Soft core processor hence introduce to the market. A soft core processor is a hardware description language (HDL) model of a specific processor (CPU) that can be customized for a given application and synthesized for ASIC or FPGA target (Jason, Anderson & Mohammed, 2006). Ralf Joost and Ralf Salomon (2005) also state that soft-core processors can be considered as equivalents to a microcontroller or "computer on chip".

In today's market, there are several FPGAs vendor that provide soft-core processor implementation in their FPGAs. Nios and Nios II soft-core processor is one of the leading soft-core processor provided by Altera. Nios II will be use for implementation throughout this project as Nios has been obsolete. Table 1.1 shows the comparison of market's available soft-core processors.

Table 1.1: Comparison of Soft-Core Processor



Category	Nios II (Fast Core)	MicroBlaze	Xtensa XL	OpenRISC 1200	LEON3
Maximum MHz	200 (FPGA)	200 (FPGA)	350 (ASIC)	300 (ASIC)	400/125 (ASIC/FPGA)
ASIC/FPGA Technology	– /Stratix and Stratix II	– /Virtex-4	0.13 μm/ –	0.18 μm/ –	0.13 μm/Not given
Reported DMIPS	150 DMIPs	166 DMIPs	–	250 DMIPS	85 DMIPs
ISA	32-bit RISC	32-Bit RISC	32-Bit RISC	32-bit RISC	32 or 64-bit RISC
Cache Memory (I/D)	Up to 64 KB	Up to 64 KB	Up to 32 KB (1)	Up to 64 KB	Up to 256 KB
Floating Point Unit (optional)	IEEE-754	IEEE-754	IEEE-754	As peripheral	IEEE-754
Pipeline	6 Stages	3 Stages	5 Stages	5 Stages	7 Stages
Custom Instructions	Up to 256 Instructions	None	Unlimited	Unspecified limit	None
Register File Size	32	32	32 or 64	32	2 to 32
Implementation	FPGA	FPGA	FPGA, ASIC	FPGA, ASIC	FPGA, ASIC
Area	700-1800 LEs	1269 LUTs	0.26 mm^2	N/A	N/A

Nios-II is a 32-bit embedded-processor architecture designed specifically for the Altera family of FPGAs. It incorporates many enhancements over the original Nios architecture, making it more suitable for a wider range of embedded computing applications, from DSP to system-control.

Cryptography plays an important role in today's security of data information. It is widely used in communication information, national security, VPN, and others sensitive data storage or transmission. In September 1997, the NIST (National Institute of Standard and Technology) call for proposal of AES (Advance Encryption Standard) to replace the DES (Data Encryption Standard). In October 2000, Rijandel Algorithm was selected as the winner of AES development race (Arif Irwansyah & etc, 2009).

Normally, AES is done through software implementation. However, the process requires long time and high performance of PC. By using the combination of hardware and software implementation, acceleration can be achieved.

1.2 Aims and Objectives

The aim of this project is to accelerate the AES encryption and decryption process in an effective and efficient way. Although the acceleration reaches max when fully hardware implemented, but the device will be more costly. Hence, the combination of hardware and software implementation will be more convenient. The final goal for this project is where hardware and software implementation can be use together in a system so that efficiency and effectiveness can be achieved.

1.3 Thesis Organization

In this paper, there are 5 sections available, Introduction, Literature Review, Methodology, Result and Discussion, Conclusion and Recommendation. Introduction basically explained the brief ideas of FPGA and AES. Literature Review are majorly discussing the journal or research that been done by other people, the method of their implementation, the algorithm, platform, theory that they applied. Understanding people works can provide innovation to the projects ideas. Methodology illustrates the implementation method that I'm going to use and the theory about my implementation.It contain In short, methodology explains what I going to do to design this project the way of achieving it. Validation, Comparison and Discussion of my project will be done in Result and Discussion part. Last but not least, whole project conclusion and the recommendation of future improvement will be discussion on Conclusion and Recommendation section.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

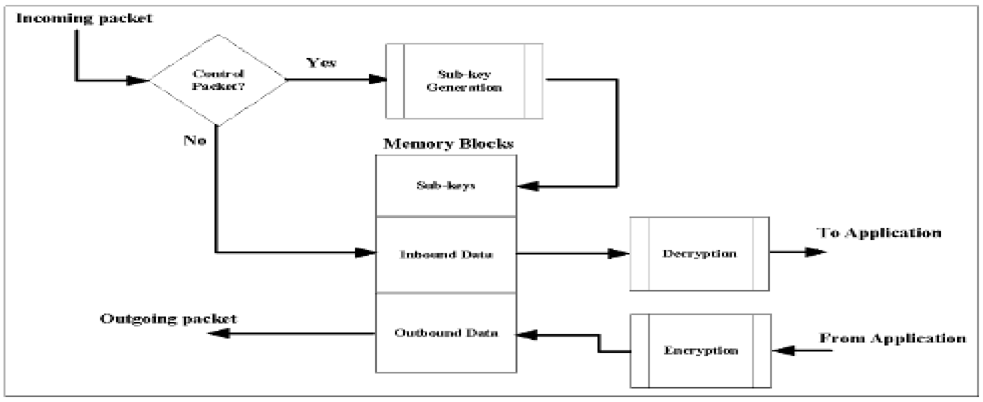
There are several journals that been review regarding AES implementation on various platform. The most common method is fully software implementation; however the process seems to be too slow for today’s mass data. Another method that been introduced lately is fully hardware implementation, although it reach high speed of encryption and decryption but due to the cost effective problem, it is still not the best solution ever. The latest technology is that AES been implement on the combination of hardware and software. This method is widely use nowadays because by the balance of hardware and software, cost effective and efficiency can be achieved.

2.2 Pure Software Implementation

2.2.1 FPGAs

The algorithm was developed using Xilinx Platform Studio 8.1i and uses C programming language. The reason why the evaluation was done by using C language was because the compiled high-level language like C is better adapted to optimizing performance compare to interpreted language like Java, besides C and C++ languages are supported by the development tools. There are 2 functions the design, the sub-key generation and the encryption/decryption process shown in Figure 2.1 (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007)

Figure 2.1: Software Implementation of AES in FPGA (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007)



The sub-key operation include bit-wise additions modulo 2 of 32-bit values obtained from user key combined with byte substitution, byte rotation and round constant (RCons) addition. After obtaining the key from the user, the sub-key functions start to generate 44 32-bit sub-keys and stored in memory. By storing the decryption keys just below the encryption key, we can assure that the decryption key can be use in the same order as encryption key which is different with the traditional method where encryption & decryption uses same sub-keys but is reverse order. The decryption key is generated by keeping the first and the last 128-bit sub-keys as it is and InvMixColumn operation on remaining intermediate 128-bit sub-keys. While having all the keys ready and stored in the memory for a given connection between source IP and destination IP. (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007)

Considered 128-bit data coming from memory into the encryption/decryption function that's operated in serial fashion, it takes 32-bits of data a time. The sub-function like SubBytes and RowShift are performed on 128-bit data while AddRoundKey and MixColumn are performed on 32-bits at a time. The final encrypted or decrypted data was stored in memory in a serial fashion, 32-bits at a time. This design that Chirag Parikh, M.S. & Parimal Patel, Ph.D (2007) develop was using 2 approaches: one without enabling any form of cache and one with instruction and data cache enabled. The reason of enabling the cache was to enable fast access to frequently used program instruction and data. (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007)

2.2.2 Desktop PC

The same developed C Code on the FPGAs was ported to the Visual C++ 6.0 complier and targeted it to Desktop PC. The code and design was similar with the FPGAs software implementation except that the platform and the environment had change (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007).

2.2.3 Symbian OS

The developed C Code as now targeted to Mobile platform with Symbian as operating system. The reason that Symbian was targeted as the choice of the application development environment is because the popularity of the Symbian operating system is coupled with excellent developer support. UIQ and Series 60 are the user interfaces that available for Symbian OS in which third-party developers can write C/C++ application. Simulation is done under Metrowerks CodeWarrior IDE (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007).

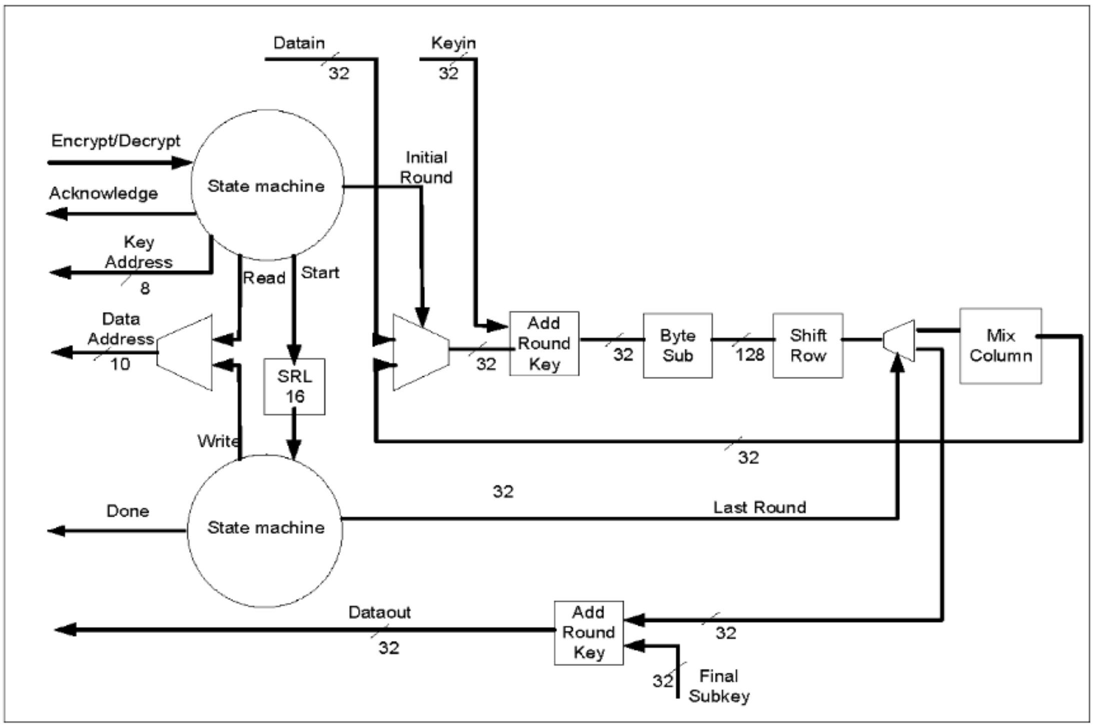
2.3 Pure Hardware Implementation

At first, the algorithm was developed in pure hardware using Xilinx ISE 8.1i tools and implemented in Xilinx's Virtex-IIPro (XC2VP3Off896-6) FPGA. The design was modeled in Verilog HDL, synthesized using Xilinx'x XST synthesis tools, simulated using Modeltech's Modelsim 6.0d simulator and implemented using Xilinx's Place and Route tools integrated in ISE 8.1i tools. (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007).

The algorithm for the hardware implementation is as below. As the data packet was received either from outside (inbound) or application (outbound), its then stored in the BRAM by the receiver engine and a start signal is generated. Upon the receiving of the start signal, the AES cores will decides the operation (encryption/decryption) based on the data transfer direction and sends back an appropriate acknowledge signal. The first 128-bits data is then taken from the BRAM

(32-bits per time) and pass the data on the Initial round. Due to the State bytes (Data) are operated individually, each AES round require 8-bit by 8-bit LUTs (Look Up Table) which will cause additional slice resources to be used up. BRAMS will be comes useful as the same purpose as they are provided by the family and will be wasted if unused. This technique can save some slice for other logic operation. By using implementing the S-BOX as LUT or ROM for SubBytes function, the operation is proven to be faster and more cost-effective than implementing the multiplicative inverse operation and affine transformation. There are no problems with ShiftRows and MixColumn operations as only AND and XOR logic included. The overall flow for AES Encryption Process is as Figure 2.2. (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007).

Figure 2.2: AES Encryption Process (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007)



2.4 Software & Hardware Combination Implementation

As above mention, the software implementation of AES is having slow processes and it’s having the tendency to expose the plaintext (origin data), while hardware implementation of AES require larger space of hardware which cause the increase of the cost. Lately, studies of software and hardware combination implementation have been done and it was found to be more efficient than software implementation and more cost effective compare to hardware implementation. There are various method that been use to balance the hardware and software implementation.

2.4.1 Optimized Design of Rijndael Algorithm Based on SOPC

From the analyzing the round transformation and key expansion of AES, it was clear that the algorithm can be optimized through the Look-Up Table. The design of optimized Rijandael algorithm can be done through SOPC (System on Programmable Chip) and implemented through software and hardware.

The AES algorithm based on SOPC system is shown in Figure 2.3. By using the standard version of Altera NIOS II embedded CPU, it guarantee for the large and systematic data processing. The system is composed of FPGA, memory and external interface. On the system, the peripheral circuit and the NIOS II are integrated to realize the control functions. As the function of the control core, NIOS II require a balance between its resource occupation and function when is generated. As the NIOS II was generated by SOPC Builder customization, the demand of the system resources is greatly reduced. Due to mass data need to be execute in algorithm, the algorithm round transformation is completed by using NIOS II and the key generation is executed by the key generator in FPGA. The external interface if FPGA is a part including some interface devices and circuit modules, which use for interfacing the data input/output and etc. The process flow for the optimized algorithm is shown as Figure 2.4. (Shunwen Xiao, Yajun Chen & Peng Luo, 2009).

Figure 2.3: The scheme of SOPC system (Shunwen Xiao, Yajun Chen & Peng Luo, 2009).

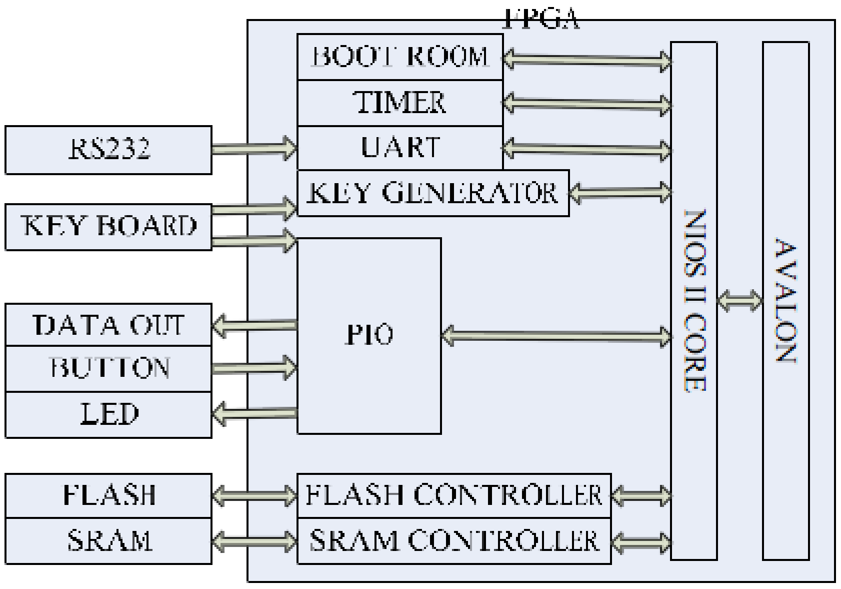
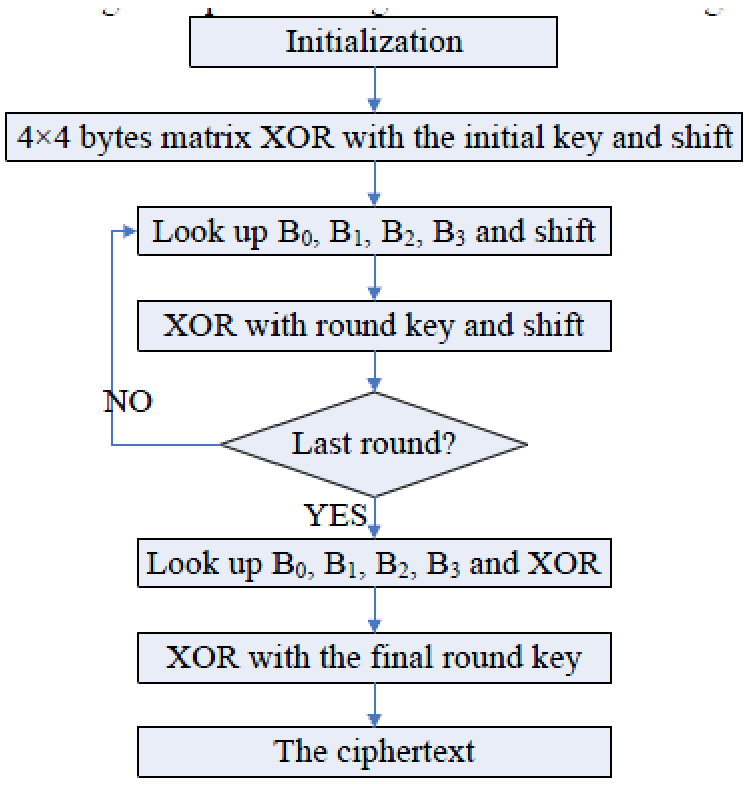
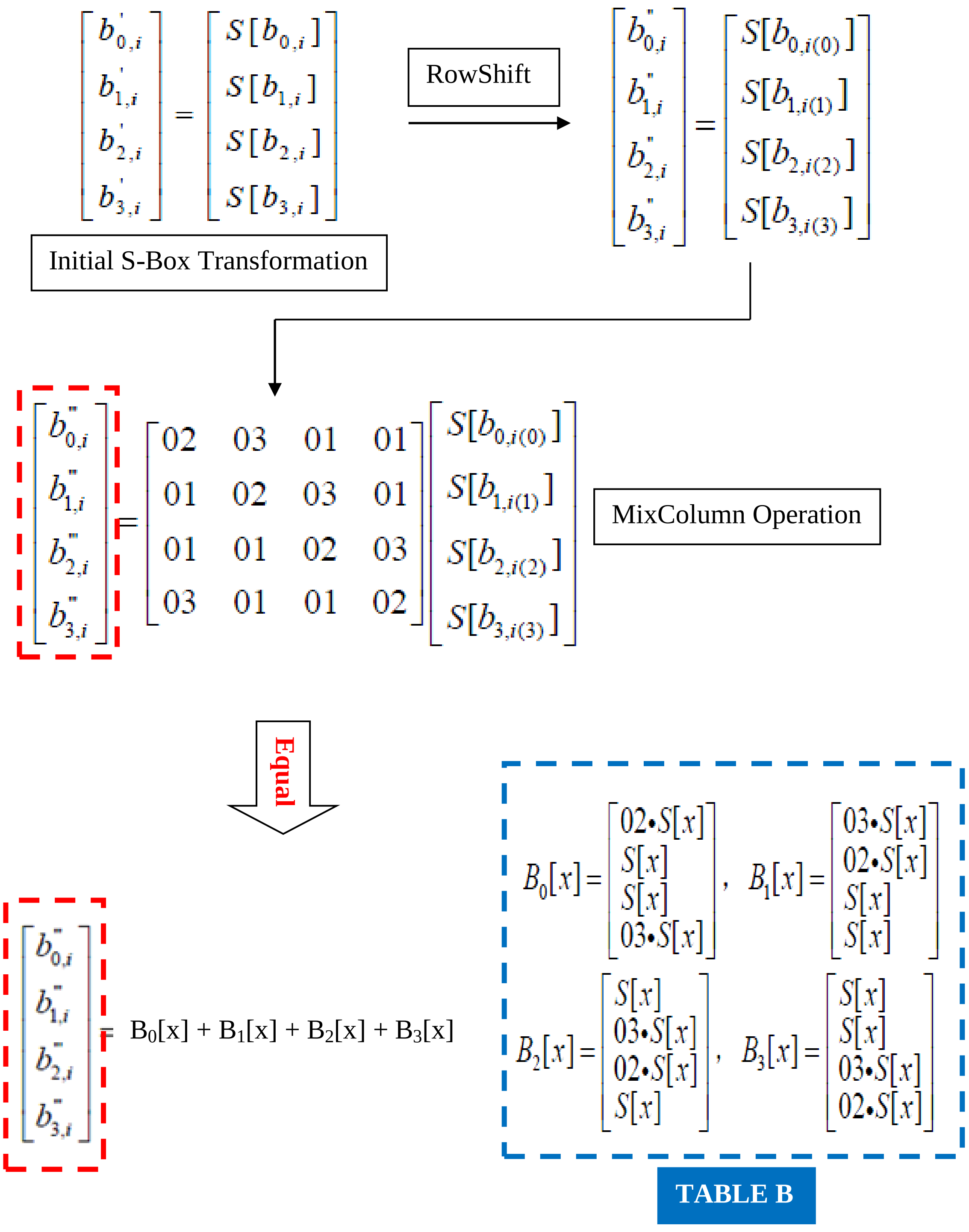


Figure 2.4: The design of optimized algorithm (Shunwen Xiao, Yajun Chen & Peng Luo, 2009).


Initialization
4×4 bytes matrix XOR with the initial key and shift
Look up B0, B1, B2, B3 and shift
XOR with round key and shift
NO
Last round?
YES
Look up B0, B1, B2, B3 and XOR
XOR with the final round key
The ciphertext

The key of optimized Rijindael algorithm is the Table B. Table B is a Look-Up table that is mixture of S-BOX with RowShift Operation and MixColumn Operation. The derivation of the table is shown below:

Figure 2.5: Table B generation Flow



$$\begin{bmatrix} b'_{0,i} \\ b'_{1,i} \\ b'_{2,i} \\ b'_{3,i} \end{bmatrix} = \begin{bmatrix} S[b_{0,i}] \\ S[b_{1,i}] \\ S[b_{2,i}] \\ S[b_{3,i}] \end{bmatrix} \qquad \begin{bmatrix} b''_{0,i} \\ b''_{1,i} \\ b''_{2,i} \\ b''_{3,i} \end{bmatrix} = \begin{bmatrix} S[b_{0,i(0)}] \\ S[b_{1,i(1)}] \\ S[b_{2,i(2)}] \\ S[b_{3,i(3)}] \end{bmatrix}$$

$$\begin{bmatrix} b'''_{0,i} \\ b'''_{1,i} \\ b'''_{2,i} \\ b'''_{3,i} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[b_{0,i(0)}] \\ S[b_{1,i(1)}] \\ S[b_{2,i(2)}] \\ S[b_{3,i(3)}] \end{bmatrix}$$

$$\begin{bmatrix} b'''_{0,i} \\ b'''_{1,i} \\ b'''_{2,i} \\ b'''_{3,i} \end{bmatrix} = B_0[x] + B_1[x] + B_2[x] + B_3[x]$$

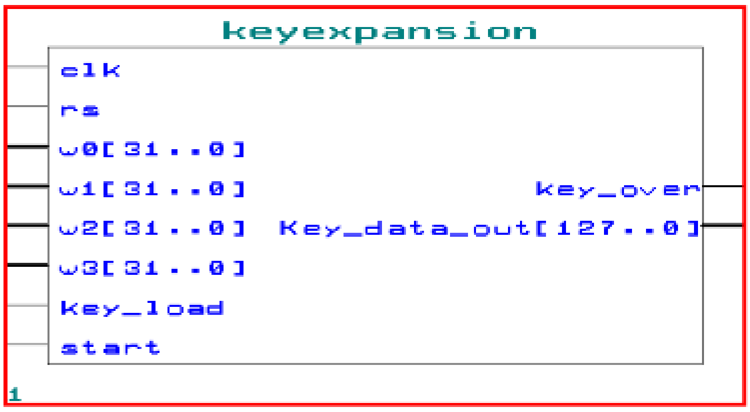
$$B_0[x] = \begin{bmatrix} 02 \bullet S[x] \\ S[x] \\ S[x] \\ 03 \bullet S[x] \end{bmatrix}, \quad B_1[x] = \begin{bmatrix} 03 \bullet S[x] \\ 02 \bullet S[x] \\ S[x] \\ S[x] \end{bmatrix}$$

$$B_2[x] = \begin{bmatrix} S[x] \\ 03 \bullet S[x] \\ 02 \bullet S[x] \\ S[x] \end{bmatrix}, \quad B_3[x] = \begin{bmatrix} S[x] \\ S[x] \\ 03 \bullet S[x] \\ 02 \bullet S[x] \end{bmatrix}$$

From the table, we can see that only Table B_0 required to be created as the other three Look-Up Tables B_1, B2 & B3 can be obtained by cyclical shift of the bytes. Due to there is no mix-column for the final round in the round operation, the Table B_0 is then change back to traditional S-BOX (Shunwen Xiao, Yajun Chen & Peng Luo, 2009).

As for the key generation operation, by using the initial key (w(0), w(1), w(2) and w(3)) the key generator generates w (4) ~ w (43) and stores them in the memory (complete memory initialization). During the period of round transformation, the quadruple frequency of the round clock is conducted by frequency multiplier and counting value is taken as the Look-up Table circuit address. There are 4 Look-up tables are implemented in a round clock period and w4i+0 ~ w4i+3 are sent out. During the same time, the 128-bits round key is exported through the serial-in parallel-out shift register. The function of description for the key generation module is as below(Shunwen Xiao, Yajun Chen & Peng Luo, 2009):

Figure 2.6: Key Generation VHDL generated module (Shunwen Xiao, Yajun Chen & Peng Luo, 2009).

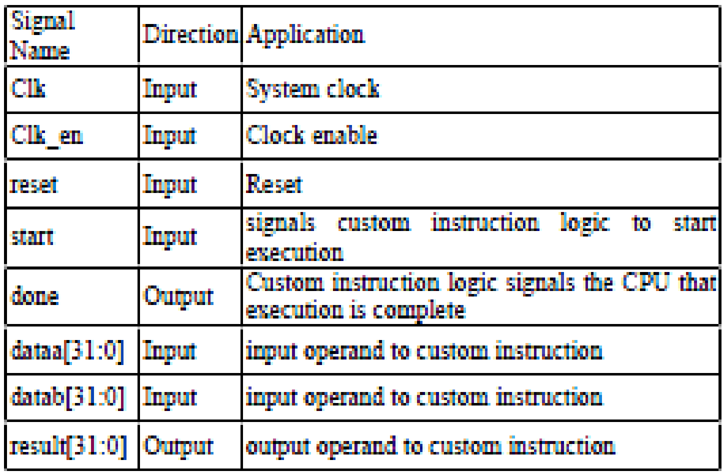


2.4.2 Exploring HW/SW Codesign of AES Algorithm Using Customs Instruction

Altera Nios II (Cyclone Version) have been use to implement the AES algorithm using custom hardware instructions. By using the custom instruction, the sequence of instruction can be reduced and the speed of processing can be accelerated by hardware (Kuan Jen Lin, Chin-Mu Hsiao and Ching Hung Jhan, 2009).

With the Nios II development kits, we can convert a hardware circuit into a custom instruction and treat it as the instruction set of the CPU. Depending on the data amount and execution cycle, NIOS II supports 4 types of custom instruction: combinatorial, multi-cycle, extended and register file. The design had selected multi-cycle custom instruction and the signal interface is given as Table 2.1

Table 2.1: The signal interface of multi-cycle customs instruction (Kuan Jen Lin, Chin-Mu Hsiao and Ching Hung Jhan, 2009).



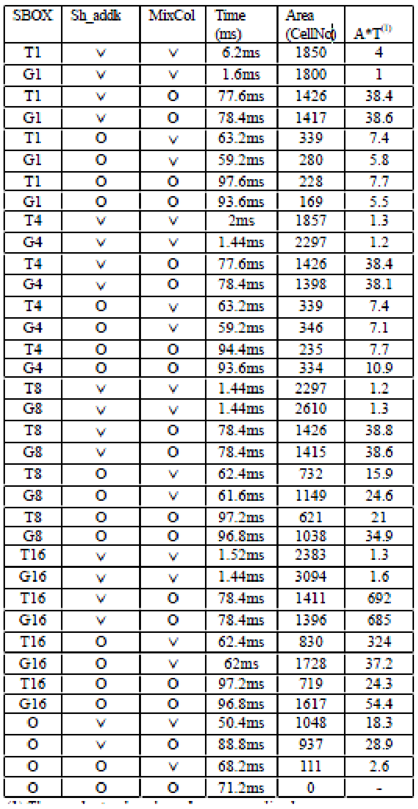
Signal Name	Direction	Application
Clk	Input	System clock
Clk_en	Input	Clock enable
reset	Input	Reset
start	Input	signals custom instruction logic to start execution
done	Output	Custom instruction logic signals the CPU that execution is complete
dataa[31:0]	Input	input operand to custom instruction
datab[31:0]	Input	input operand to custom instruction
result[31:0]	Output	output operand to custom instruction

By designing the circuit in accordance with the signal interface, the circuit is now ready for customs instruction conversion where is done through Quartus II. Now, the circuit can be called as a function in C programming. There are few design spec with parameterized synthesizable design have been explored. Relevant programmable parameters include:

i. SW, TSBOX or GSBOX: A user can choose software table (SW), pre-store hardware table (TSBOX), generating transformation by combinational logic to implement SBOX (GSBOX), which is realized by composite field arithmetic as stated in the third section.
ii. Number of SBOX: If using TSBOX or GSBOX, a user can choose how many SBOX to implement: 1, 4, 8 or 16.
iii. MixColumn: A user can choose whether to implement it using hardware.
iv. ShiftRow+AddRoundkey: A user can choose whether to implement it using hardware.

By using the combination of the relevant programmable parameter, 36 combinations can be made and Table 2.2 showing the performance of each parameter used. In table 2.2, T# indicates the number of SBOX(s) to implement the customs instruction, and G# indicates the number of SBOX(S) made using combinatorial logic. As for the Sh_addk(shiftrow-addkey), √ indicates that it was implemented by hardware custom instruction and O indicates it was adopted by software implementation. (Kuan Jen Lin, Chin-Mu Hsiao and Ching Hung Jhan, 2009).

Table 2.2: Comparison of area and time among various HW/SW mixed design



SBOX	Sh_addk	MixCol	Time (ms)	Area (CellNo)	A*T[1]
T1	v	v	6.2ms	1850	4
G1	v	v	1.6ms	1800	1
T1	v	O	77.6ms	1426	38.4
G1	v	O	78.4ms	1417	38.6
T1	O	v	63.2ms	339	7.4
G1	O	v	59.2ms	280	5.8
T1	O	O	97.6ms	228	7.7
G1	O	O	93.6ms	169	5.5
T4	v	v	2ms	1857	1.3
G4	v	v	1.44ms	2297	1.2
T4	v	O	77.6ms	1426	38.4
G4	v	O	78.4ms	1398	38.1
T4	O	v	63.2ms	339	7.4
G4	O	v	59.2ms	346	7.1
T4	O	O	94.4ms	235	7.7
G4	O	O	93.6ms	334	10.9
T8	v	v	1.44ms	2297	1.2
G8	v	v	1.44ms	2610	1.3
T8	v	O	78.4ms	1426	38.8
G8	v	O	78.4ms	1415	38.6
T8	O	v	62.4ms	732	15.9
G8	O	v	61.6ms	1149	24.6
T8	O	O	97.2ms	621	21
G8	O	O	96.8ms	1038	34.9
T16	v	v	1.52ms	2383	1.3
G16	v	v	1.44ms	3094	1.6
T16	v	O	78.4ms	1411	692
G16	v	O	78.4ms	1396	685
T16	O	v	62.4ms	830	324
G16	O	v	62ms	1728	37.2
T16	O	O	97.2ms	719	24.3
G16	O	O	96.8ms	1617	54.4
O	v	v	50.4ms	1048	18.3
O	v	O	88.8ms	937	28.9
O	O	v	68.2ms	111	2.6
O	O	O	71.2ms	0	-

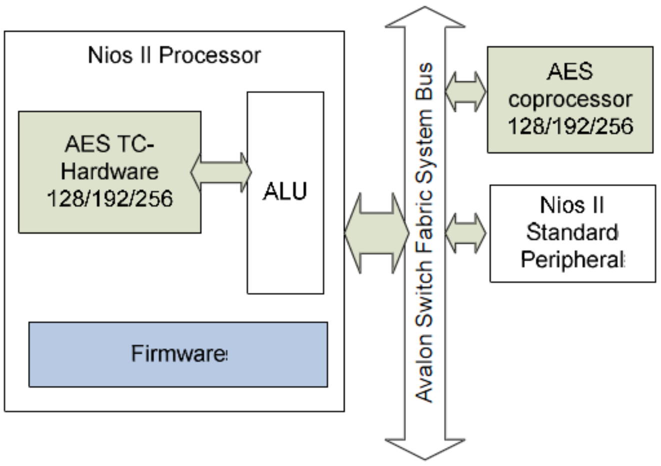
Throughout the design, the NIOS II is set to be run on 50MHz and the time is measured on running 32 packets of data with each having 128-bits. The key generation is done using same implementation method (LUT/combinational logic) as used in the data path. After the cipher keys are generated, data are encrypted sequentially (Kuan Jen Lin, Chin-Mu Hsiao and Ching Hung Jhan, 2009).

From table 2.2, we can see that the design with 4 S-Boxes of combinational logic require the least hardware area among those having the best performance (1.44ms), hence it is the best choice for high performance needs. If using less than 4 S-Boxes, the design using GSBOX has better performance compare to TSBOX. In other hand, when more than 4 S-Boxes required. GSBOX have similar performance but TSBOX implementation require less area. Hardware implementation for SBOX and MixColumn operation improve the performance, however the hardware implementation for AddRoundKey and ShiftRow may take the performance even worse than pure software implementation. Due to the limitation for the bus width, by increasing the S-Boxes that been used, the performance is not further improved (Kuan Jen Lin, Chin-Mu Hsiao and Ching Hung Jhan, 2009).

2.4.3 An AES Tightly Coupled Hardware Accelerator in an FPGA-based Embedded Processor Core

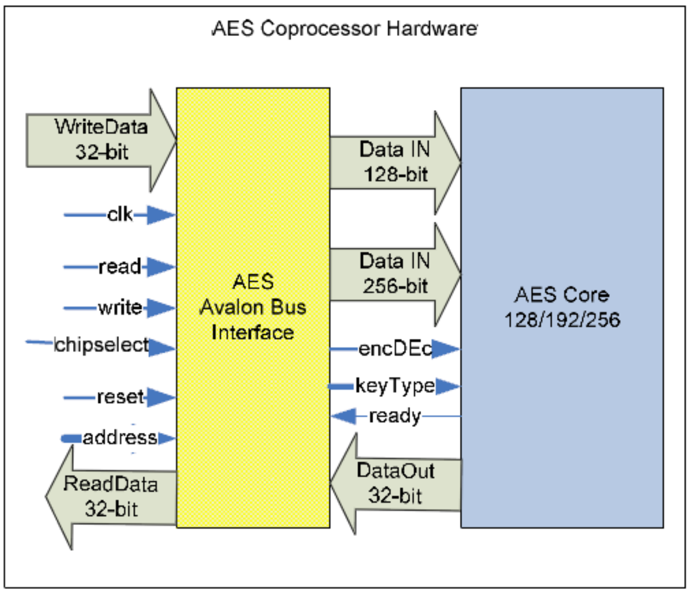
The common method to enhance the performance of the AES algorithm is to incorporate a crypto co-processor dedicated to execute certain parts of the algorithm, offloading the main embedded processor of specific compute-intensive routines, thus accelerating the execution the overall algorithm. The disadvantages on this implementation method are that the co-processor are loosely-coupled to the main processor and the interface between the main processor and the co-processor also incur severe performance bottleneck due to system bus communication and synchronization overhead. The new and recent trend of enhancing the AES algorithm is to extend the instruction set architecture (ISA) of the processor with custom instruction for performance critical operation. In this approach, some hardware implementation in custom logic is tightly-coupled to the embedded processor (Arif Irwansyah, Vishnu P. Nambiar & Mohamed Khalil-Hani, 2009)

Figure 2.7: TC-Hardware and Co-processor in NIOS II (Arif Irwansyah, Vishnu P. Nambiar & Mohamed Khalil-Hani, 2009)


Nios II Processor
AES TC-Hardware 128/192/256
ALU
Firmware
Avalon Switch Fabric System Bus
AES coprocessor 128/192/256
Nios II Standard Peripheral

As for co-processor design, an Avalon Switch Fabric System Bus is designed to interface the whole AES core with the Nios II. The AES hardware can be access through memory mapping. From figure 2.7, we can see that the co-processor is loosely coupled to the Nios II processor. The system structure of AES co-processor was illustrated as figure 2.8. From the figure, it can be seen that the AES co-processor have only 1 port (32-bits) input for data and cipher key to AES core where the port is named as WriteData port and 1 output port to have data transfer from AES core. (Arif Irwansyah, Vishnu P. Nambiar & Mohamed Khalil-Hani, 2009)

Figure 2.8: AES Coprocessor Hardware



Unlike coprocessor, TC-Hardware custom instruction is attach directly to the ALU in the main processor's data path. Custom instructions give the designer ability to accelerate time critical software algorithms by converting to custom hardware logic blocks. TC-hardware custom instructions also reduce the communication overhead between the AES core and the processor. In addition, it also allows us to fetch the data input or key input using two ports at the same time. This option reduces the time for supplying inputs to the AES core dramatically. The TC-hardware interface can be seen as figure 2.9. (Arif Irwansyah, Vishnu P. Nambiar & Mohamed Khalil-Hani, 2009)

Figure 2.9: TC-Hardware Interface. (Arif Irwansyah, Vishnu P. Nambiar & Mohamed Khalil-Hani, 2009)

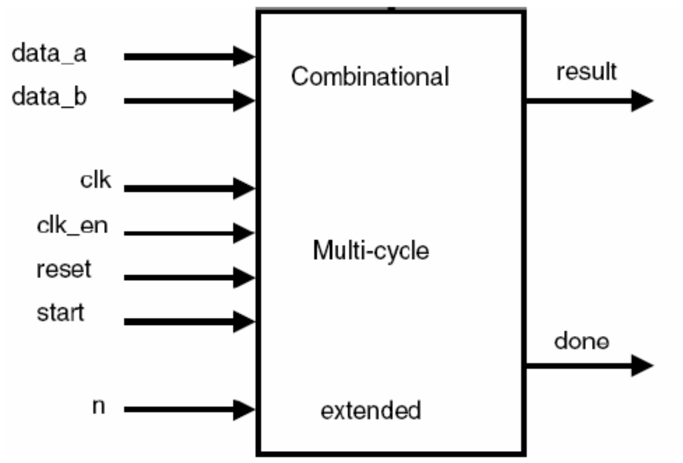
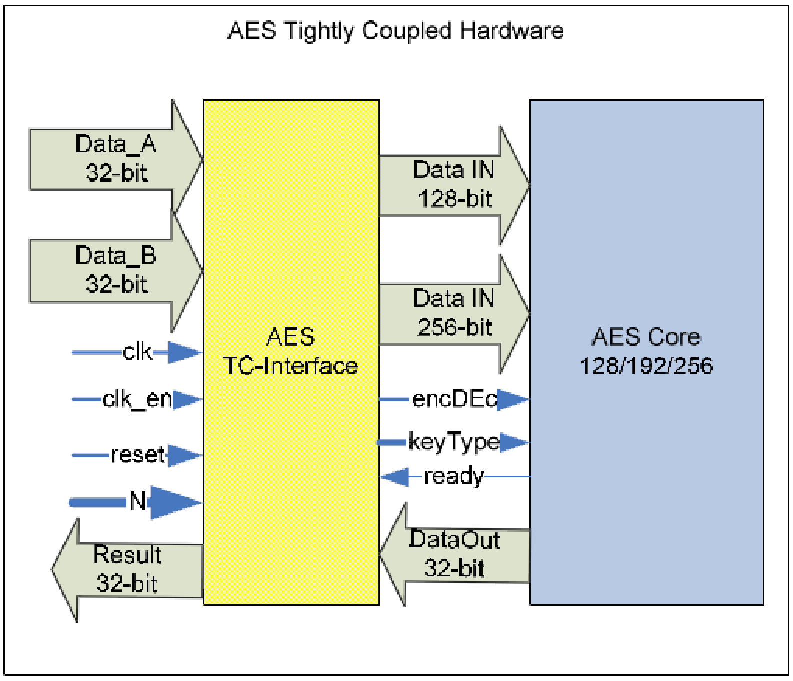


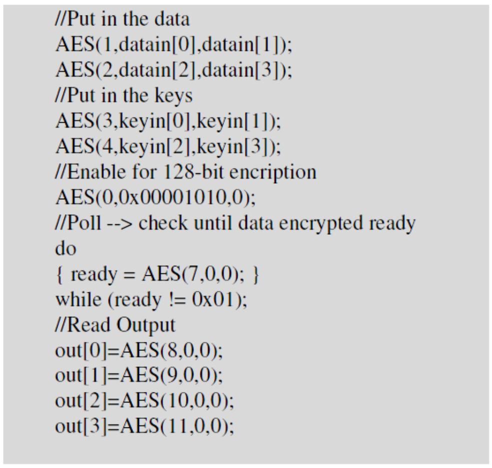
Figure 2.10 shows that the organization how AES works. Data_A and Data_B ports are 32-bit input port that transfer 128-bit of data input and 128 until 256 bits of keys for AES core. Both input transfer can occur at the same time, hence fetching 128-bit of data input just require 2 cycle as compare to coprocessor approach that require 4 cycles. As for the key input for 128,192 & 256 bits, the AES TC-hardware require 2,3 & 4 cycles which is contrary with the co-processor that require 4,6 & 8 cycles. The N-port is a 2-bit port that selects the operation in AES TC-interface and the result port is 32-bit output port that read data from AES core.

Figure 2.10: AES Tightly Coupled Hardware (Arif Irwansyah, Vishnu P. Nambiar & Mohamed Khalil-Hani, 2009)



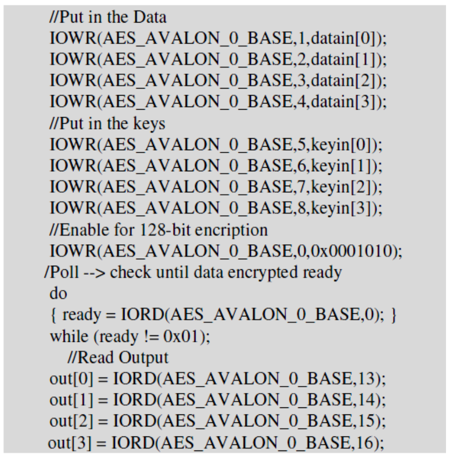
In terms of coding design, the C program for Nios II using TC-hardware is simpler and effective compare to coprocessor version. Comparison can be seen as below figure 2.11. The execution times of Encryption/Decryption for Co-processor and TC-hardware is illustrated on table 2.3.

Figure 2.11: Comparison of coding between TC-hardware and Coprocessor(Arif Irwansyah, Vishnu P. Nambiar & Mohamed Khalil-Hani, 2009)



```
//Put in the data
AES(1,datain[0],datain[1]);
AES(2,datain[2],datain[3]);
//Put in the keys
AES(3,keyin[0],keyin[1]);
AES(4,keyin[2],keyin[3]);
//Enable for 128-bit encription
AES(0,0x00001010,0);
//Poll --> check until data encrypted ready
do
{ ready = AES(7,0,0); }
while (ready != 0x01);
//Read Output
out[0]=AES(8,0,0);
out[1]=AES(9,0,0);
out[2]=AES(10,0,0);
out[3]=AES(11,0,0);
```

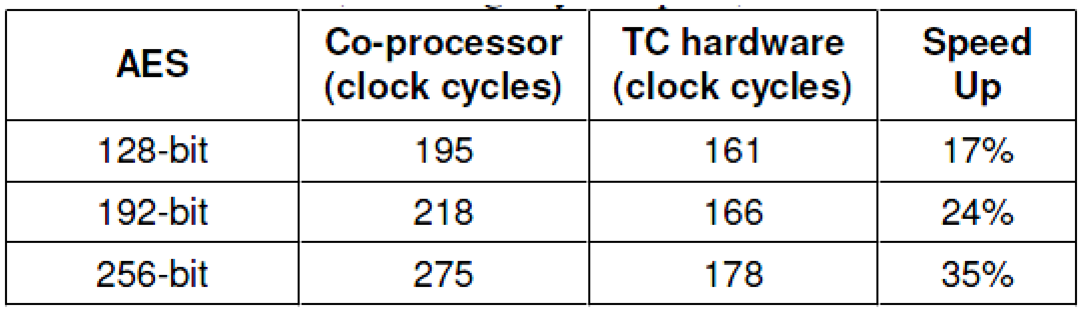
TC-Hardware version



```
//Put in the Data
IOWR(AES_AVALON_0_BASE,1,datain[0]);
IOWR(AES_AVALON_0_BASE,2,datain[1]);
IOWR(AES_AVALON_0_BASE,3,datain[2]);
IOWR(AES_AVALON_0_BASE,4,datain[3]);
//Put in the keys
IOWR(AES_AVALON_0_BASE,5,keyin[0]);
IOWR(AES_AVALON_0_BASE,6,keyin[1]);
IOWR(AES_AVALON_0_BASE,7,keyin[2]);
IOWR(AES_AVALON_0_BASE,8,keyin[3]);
//Enable for 128-bit encription
IOWR(AES_AVALON_0_BASE,0,0x0001010);
/Poll --> check until data encrypted ready
do
{ ready = IORD(AES_AVALON_0_BASE,0); }
while (ready != 0x01);
   //Read Output
out[0] = IORD(AES_AVALON_0_BASE,13);
out[1] = IORD(AES_AVALON_0_BASE,14);
out[2] = IORD(AES_AVALON_0_BASE,15);
out[3] = IORD(AES_AVALON_0_BASE,16);
```

Co-Processor Version

Table 2.3: Execution times of Encryption/Decryption (Arif Irwansyah, Vishnu P. Nambiar & Mohamed Khalil-Hani, 2009)



AES	Co-processor (clock cycles)	TC hardware (clock cycles)	Speed Up
128-bit	195	161	17%
192-bit	218	166	24%
256-bit	275	178	35%

2.4.4 Implementation of High Throughput Sequential and Fully Pipelined AES Processor on FPGA

In this implementation, FPGA chips is used to realize the high throughput 128-bits AES cipher processor by new high-speed and hardware sharing functional blocks. As we know, AES functional calculation includes SubBytes, ShiftRows, MixColumns and AddRoundKey. By replacing the old fashion ways of ROM mapping for SubBytes with CAM (content-addressable memory) to achieve new proposed high-speed SubBytes block. The new hardware sharing architecture is applied to implement the proposed high-speed MixColumns block. Efficient low-cost AddRoundKey architecture is used for real-time key generations.(Chih-Peng Fan and Jun-Kui Hwang,2007)

For the high speed realization of the SubBytes and InvSubBytes hardware the traditional ROM-based concept could not reach very high speed operation. By applying the content-addressable memory (CAM) based architecture as Figure 2.12 to realize SubBytes and InvSubBytes circuit, high speed operation can be achieve. From the figure, we can see that as we enable the SubBytes operation, the registers a_i, for i= 1,2,3,4,….,256, will output the 8 most significant bits to the inputs of CMP circuit(Figure 2.13). In order for further high-speed full pipelined AES implementation, the SubBytes and InvSubBytes can be divided into 3 pipelining stage by adding 2 pipelined register arrays. The 3 phase pipelined

SubBytes/InvSubBytes module can achieve higher operational frequency than the traditional ROM-based scheme. (Chih-Peng Fan and Jun-Kui Hwang,2007)

Figure 2.12: Proposed new realization for SubBytes and InvSubBytes Transformation (Chih-Peng Fan and Jun-Kui Hwang,2007)

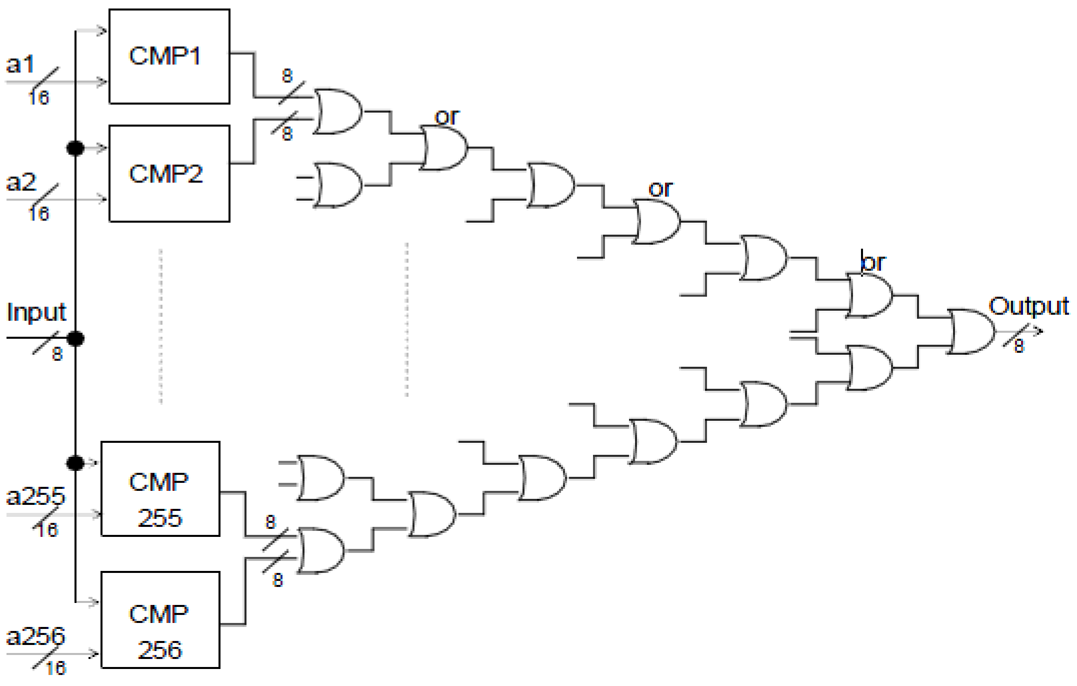
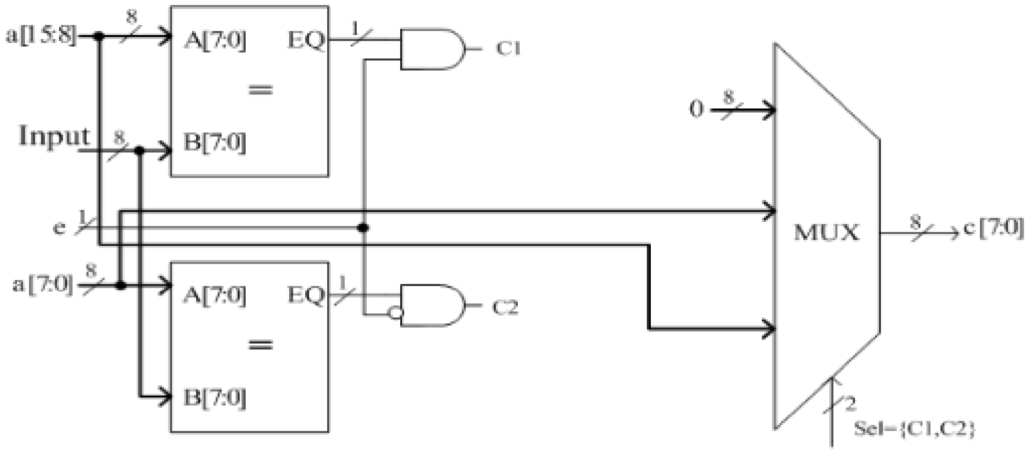


Figure 2.13: Realization of CMP Circuit (Chih-Peng Fan and Jun-Kui Hwang,2007)



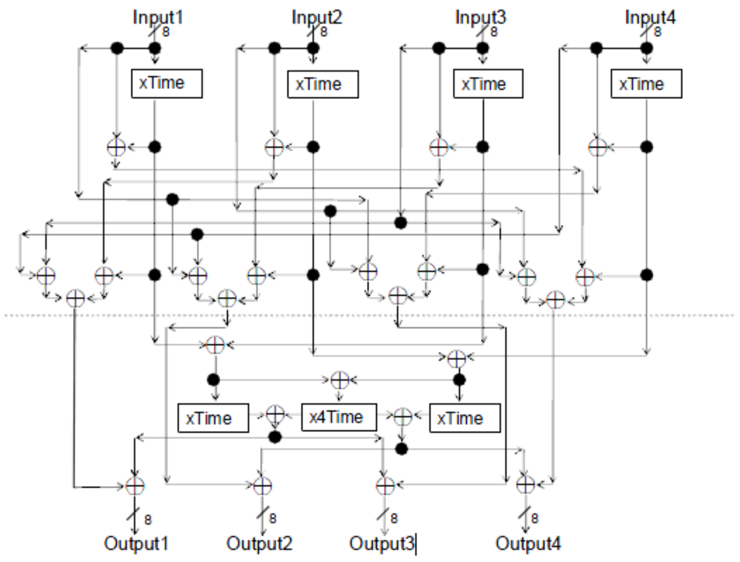
As the AES theory states that the operation of MixColums and Inverse Mixcolumns transformation is having different corresponding matrix polynomial. Instead of creating two separate hardware architecture, hardware sharing architecture are design for both operation. Firstly, the operation of InvMix was decomposed so that it will have common factor with MixColumns operation. The decomposition can be illustrated as figure 2.14. By using these common factors, high-speed hardware sharing circuits can the design to implement these transformations.

Figure 2.14: Decomposition of InvMixColumns(Chih-Peng Fan and Jun-Kui Hwang,2007)

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

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} + \begin{bmatrix} 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} + \begin{bmatrix} 04 & 00 & 04 & 00 \\ 00 & 04 & 00 & 04 \\ 04 & 00 & 04 & 00 \\ 00 & 04 & 00 & 04 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Figure 2.15: Circuit Architecture of MixColumns and InvMixColumns(Chih-Peng Fan and Jun-Kui Hwang,2007)



A real time high speed Key expansion for generation of 128-bit was designed. The realized Key expansion circuits can generates keys for AES encryption and decryption. Due to the asymmetric of the decryption process, the key expansion circuit for decryption needs to collocate the InvMixColumns circuits. In the operation of Key expansion, the 128-bits keys is segmented into 4 32-bits data and stored in 4 corresponding a, b, c, d registers. The output of register d must be pass through the operation of ROT, S-Box and RCON. Figure 2.16 showing the circuit architectures of sequential on-the-fly key expansions and figure 2.17 shows the circuit architecture for non-sequential on-the-fly key expansion. (Chih-Peng Fan and Jun-Kui Hwang,2007)

Figure 2.16: Circuit architectures of sequential on-the-fly key expansions(Chih-Peng Fan and Jun-Kui Hwang,2007)

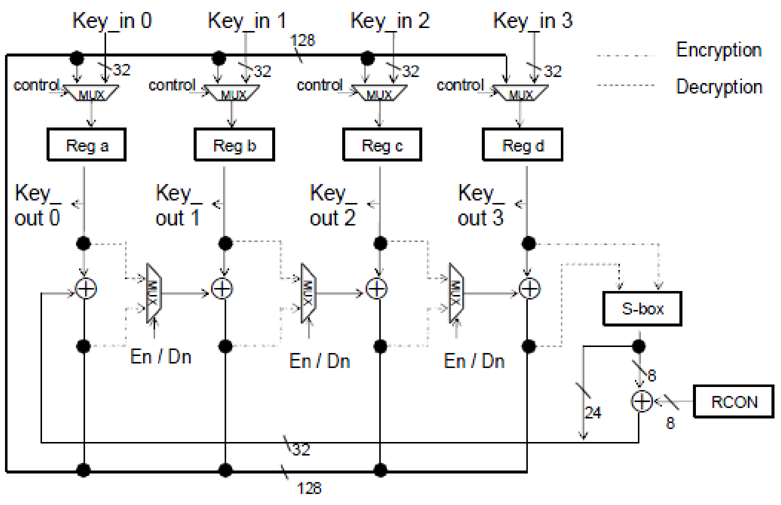
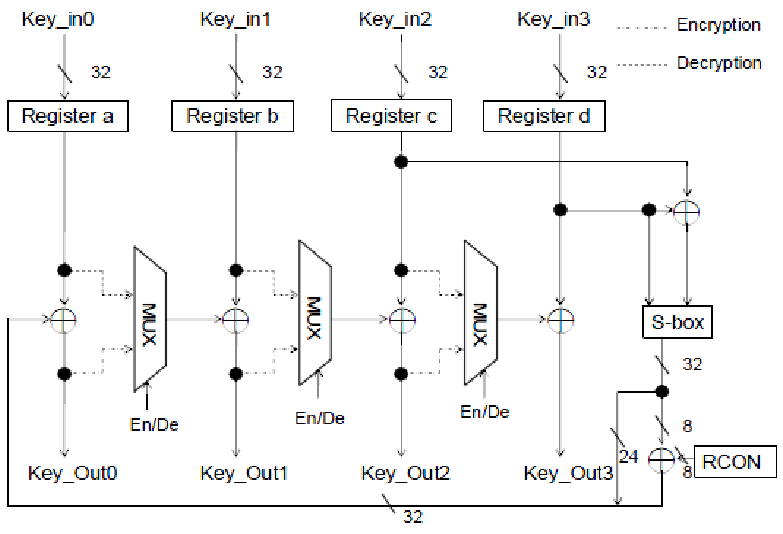


Figure 2.17: Circuit architectures of non-sequential on-the-fly key expansions(Chih-Peng Fan and Jun-Kui Hwang,2007)



From what have discuss, there are two architectures that provide high-speed processing, which are sequential and full pipelined schemes. Figure 2.18 shows the Hardware architecture of the proposed sequential AES processor and figure 2.19 shows the Hardware architecture of the proposed full pipelined AES processor. (Chih-Peng Fan and Jun-Kui Hwang,2007)

Figure 2.18: Hardware architecture of the proposed sequential AES processor(Chih-Peng Fan and Jun-Kui Hwang,2007)

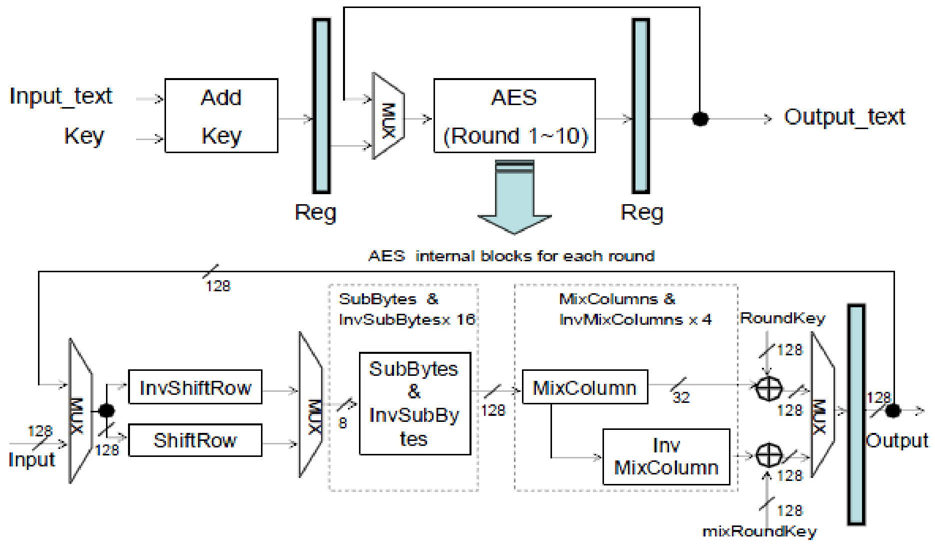
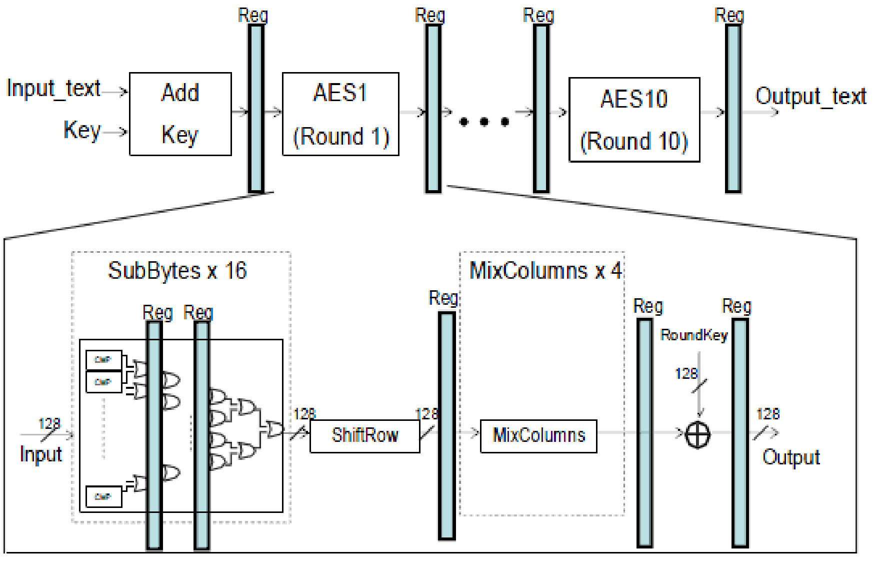


Figure 2.19: Hardware architecture of the proposed full pipelined AES processor(Chih-Peng Fan and Jun-Kui Hwang,2007)



CHAPTER 3

METHODOLOGY

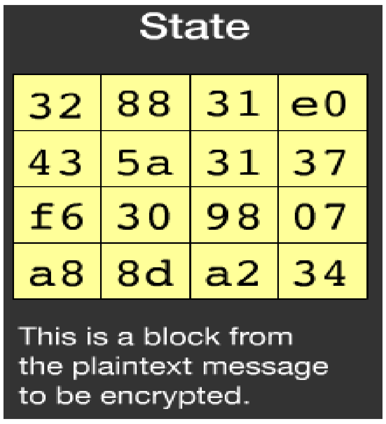
3.1 AES

3.1.1 Introduction of AES

AES (Advance Encryption Standard) is a symmetric-key encryption standard adopted by the U.S. government. It comprises three block ciphers, AES-128, AES-192 and AES-256. Encryption is the process of transforming information (normally referring as plaintext) using a string of bits (called key) to make it unreadable to anyone except those possessing the key. Inversely, decryption is to transform the cipher text to readable information by using the key and the proper algorithm. In our case, AES is the algorithm that going to be use in encryption.

Basically AES can be divided into 3 processes: Encryption, Decryption, and Key Expansion. According to the theory of AES, the data in groups of 128-bits will be initially transformed into a 4 x 4 matrix with each slot containing 1 byte of data and called a State.

Figure 3.1: Transformed Data Matrix



State

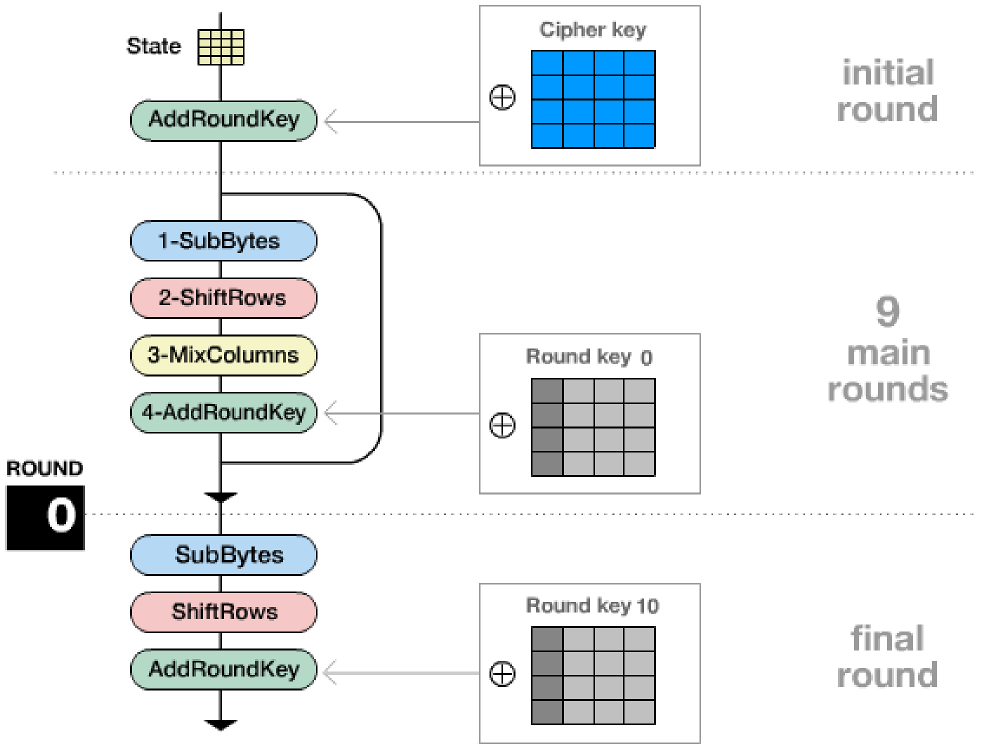
32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

This is a block from the plaintext message to be encrypted.

3.1.2 Encryption

There are 4 functions inside the encryption process (SubBytes, ShiftRows, MixColumns & AddRoundKey). Based on the selected block ciphers, the number of rounds the functions will be applied is determined: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys.

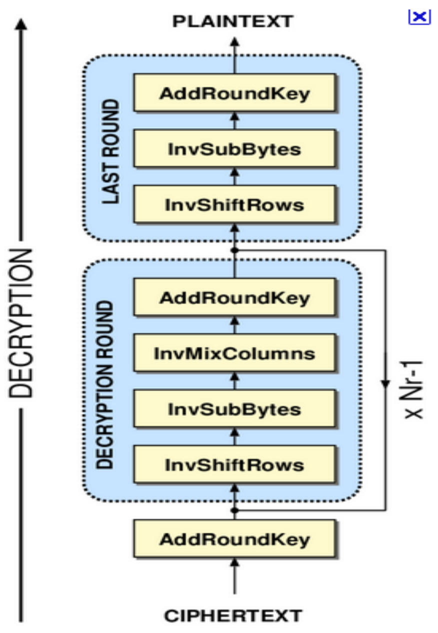
Figure 3.2: AES-128 Encryption Flow



3.1.3 Decryption

The process of decryption is similar to that of encryption. The differences are: each of the SubBytes, ShiftRows, MixColumns function is replaced with InvSubBytes, InvShiftRows & InvMixColumns, while Add Round Key remains unchanged. The sequence of the functions is also ~~is~~ rearranged in a reversed way.

Figure 3.3: Decryption Flow



3.1.4 Key Expansion

Throughout each round, the Add Round Key function uses a different key that has been expanded from a short key (cipher key). This expansion is called Rijndael key schedule. The total number of round keys required is equal to Nr + 1 (where Nr = Number of rounds = 10). Although there are 10 rounds, eleven keys are needed because one extra key is needed in the Initial round. The key expansion algorithm uses bit-wise additions modulo 2 of 32-bit values obtained from user key combined with byte substitution, byte rotation to right and round constants (RCons) addition (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007).

3.2 Implementation Process and Flow

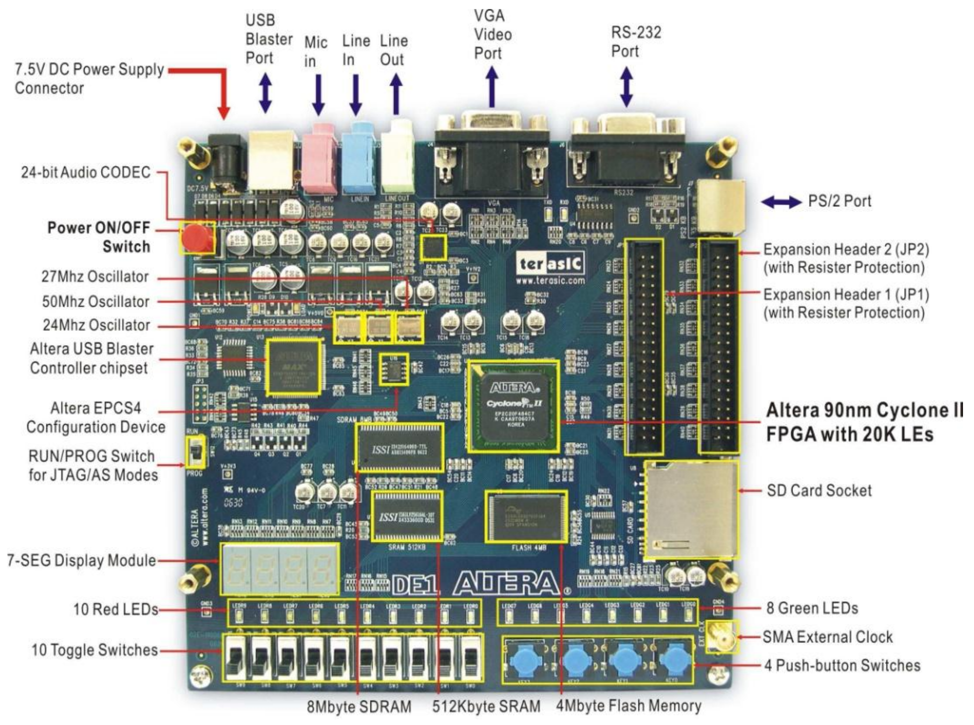
The above explained AES algorithm is based on 8-bits processing scheme. As for the Nios II where it’s having 32-bits of processing power, modification on the traditional algorithm would make the system to be more efficient. Based on the AES theory, we have encryption, decryption and key expansion process. Firstly, modification and implementation method of encryption will be explained as decryption is just an inverse of encryption.

Encryption process include of SubBytes, RowShift, MixColumns and AddRoundKey. Traditionally, S-Box for SubBytes is meant for 8-bit substitution. However, now the designs are made in 32-bits architecture, modification of S-Box can be made to come across 32-bits substitution.

Basically the design I implemented can be categorised into 3 stages where at first SOPC builder will be used to generate blocks for the customized module with Nios II embedded with custom instructions. This custom instruction will be first written in a Verilog file. After completing the system for the Nios II, Quartus II schematic diagram will be used to draw the connection between the built Nios II system with peripherals and other modules of the overall system such as S-Box substitution ROM. As the last stage, Nios II IDE software development kit will be used to write a C code program~~,~~ which will be loaded to the Nios II module. The program includes some simple logic operation such as XOR for the AddRoundKey function.

Evaluation will be done on DE1 board manufactured by Altera. The figure 3.4 below shows the diagram of the DE1 development board.

Figure 3.4: Altera DE1 Board



3.3 Hardware

3.3.1 Nios II

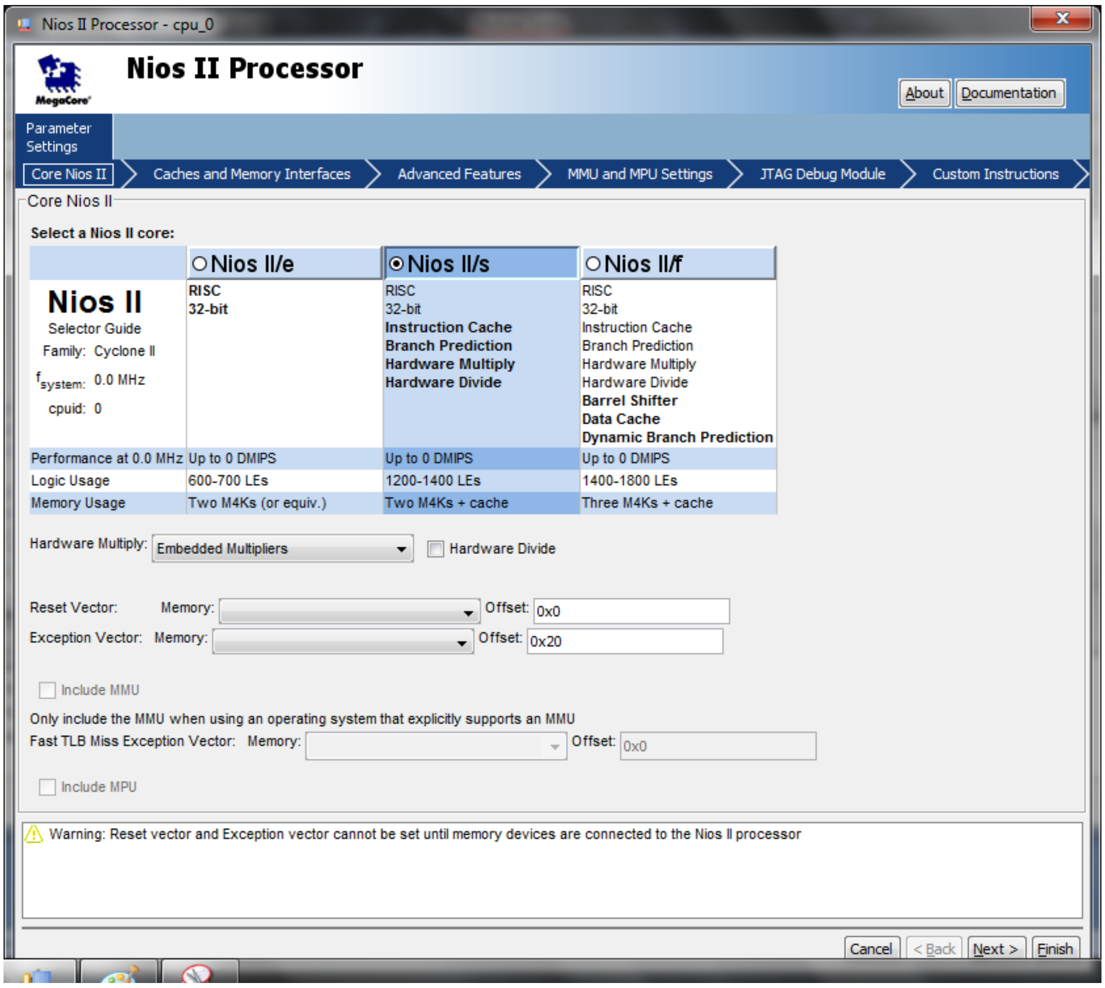
Nios II is designed by one of the leading vendors of Programmable Logic Devices, Altera Corporation. Nios II can be implemented in Stratix, Stratix II ,and Cyclone Families of FPGA that are also manufactured by Altera.

Nios II soft-core processor is a general purpose Reduced Intruction Set Computer (RISC) processor core and features Harvard memory architecture (Jason, Anderson & Mohammed, 2006). According to the specifications provided by Altera Corporation, Nios II is featured with full 32-bit Instruction Set Architecture (ISA), 32 general purpose registers, single-instruction 32x32 multiply and divide operation, and dedicated instructions for 64-bit and 128-bit products of multiplication.

Based on Altera, Nios II processor comes in three version of design: economy, standard and fast core. Each core version is different in terms on number of pipelining stages, instruction & data cache memories and hardware components for multiply and divide operations. Based on the requirements of the system, one of cores can be selected.

Peripherals can be added to Nios II through the Avalon Interface Bus which contains the necessary logic to interface the processor with the off-the-shelf IP cores or custom-made peripherals (Jason, Anderson & Mohammed, 2006).

Figure 3.5: Nios II Wizard


Nios II Processor - cpu_0

Nios II Processor

About | Documentation

Parameter Settings

Core Nios II > Caches and Memory Interfaces > Advanced Features > MMU and MPU Settings > JTAG Debug Module > Custom Instructions

Core Nios II

Select a Nios II core:

Nios II Selector Guide Family: Cyclone II f_{system}: 0.0 MHz cpuid: 0	○Nios II/e	⦿Nios II/s	○Nios II/f
	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction
Performance at 0.0 MHz	Up to 0 DMIPS	Up to 0 DMIPS	Up to 0 DMIPS
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs
Memory Usage	Two M4Ks (or equiv.)	Two M4Ks + cache	Three M4Ks + cache

Hardware Multiply: Embedded Multipliers ☐ Hardware Divide

Reset Vector: Memory: Offset: 0x0

Exception Vector: Memory: Offset: 0x20

☐ Include MMU

Only include the MMU when using an operating system that explicitly supports an MMU

Fast TLB Miss Exception Vector: Memory: Offset: 0x0

☐ Include MPU

⚠ Warning: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor

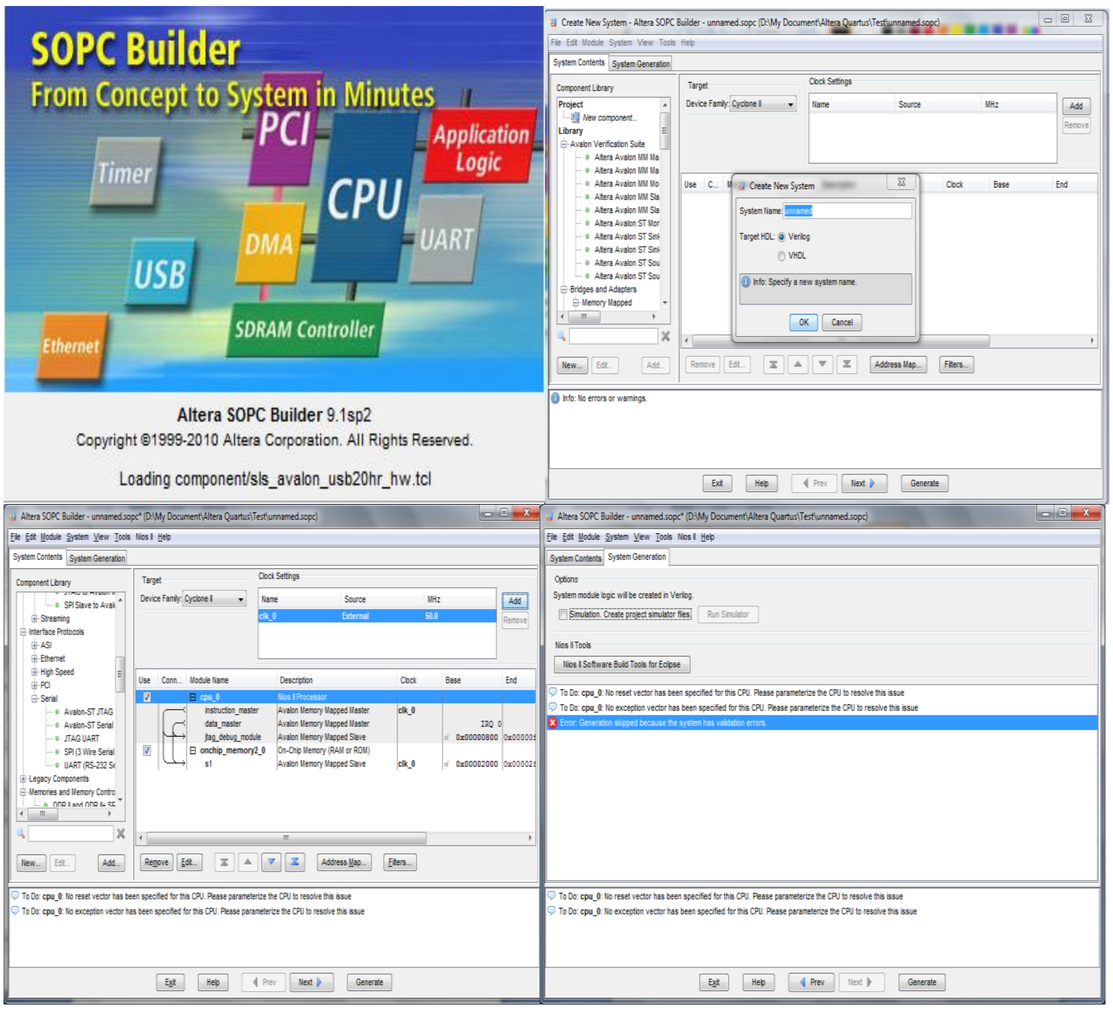
Cancel | < Back | Next > | Finish

3.3.2 System Structure

In order to produce a workable embedded system, the structure for the system must be known. There are various software and systems that have been provided by the vendor to help the users in their system design. As for Altera Corporation, SOPC builder, Ouartus II, Eclipse IDE, etc., are systems and software that can be downloaded from their website.

According to Wikipedia, FPGA-based SOPC (system on Programmable Chip) is a platform made by Altera that automates connecting soft-hardware components to create a complete system that runs on any of its various FPGA chips. SOPC Builder incorporates a library of pre-made components (including the flagship Nios II soft processor, memory controllers, interfaces, and peripherals) and an interface for incorporating custom ones. Interconnections are made though the Avalon bus. Bus arbitration, bus width matching, and even clock domain crossing are all handled automatically when SOPC Builder generates the system (SOPC Builder, Wikipedia). By using SOPC builder, we can describe the relationship between modules and link the whole system up. Below is shown a screen capture of SOPC Builder software

Figure 3.6: SOPC Builder ScreenShots



Altera SOPC Builder 9.1sp2
Copyright ©1999-2010 Altera Corporation. All Rights Reserved.
Loading component/sls_avalon_usb20hr_hw.tcl

Create New System - Altera SOPC Builder - unnamed.sopc (D:\My Document\Altera Quartus\Test\unnamed.sopc)

Create New System
System Name: unnamed
Target HDL: Verilog
Info: Specify a new system name.

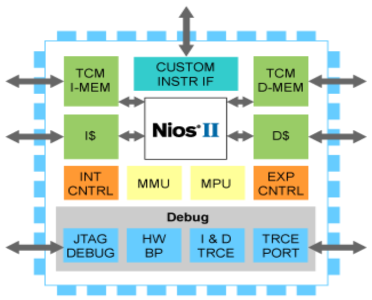
Info: No errors or warnings.

Altera SOPC Builder - unnamed.sopc* (D:\My Document\Altera Quartus\Test\unnamed.sopc)

System Generation
Options
System module logic will be created in Verilog.
Simulation. Create project simulator files
Nios II Tools
Nios II Software Build Tools for Eclipse

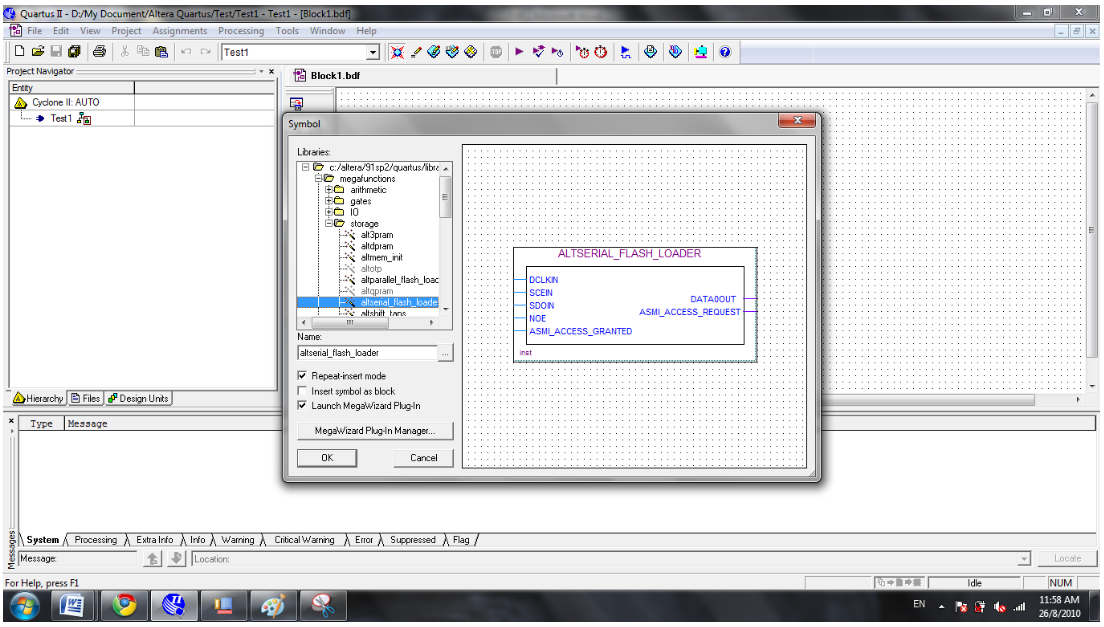
To Do: **cpu_0**: No reset vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
To Do: **cpu_0**: No exception vector has been specified for this CPU. Please parameterize the CPU to resolve this issue
Error: Generation skipped because the system has validation errors.

Figure 3.7: SOPC Example



Quartus II software is used for analysis and synthesis of HDL designs. Designers can compile their design, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer (Quartus II, Wikipedia). Besides simulation purpose, the schematic design system that is embedded inside the software can be used for attaching the design modules with other peripherals such as 7-segment display, button switch, etc.

Figure 3.8: Schematic Diagram Platform, Quartus II

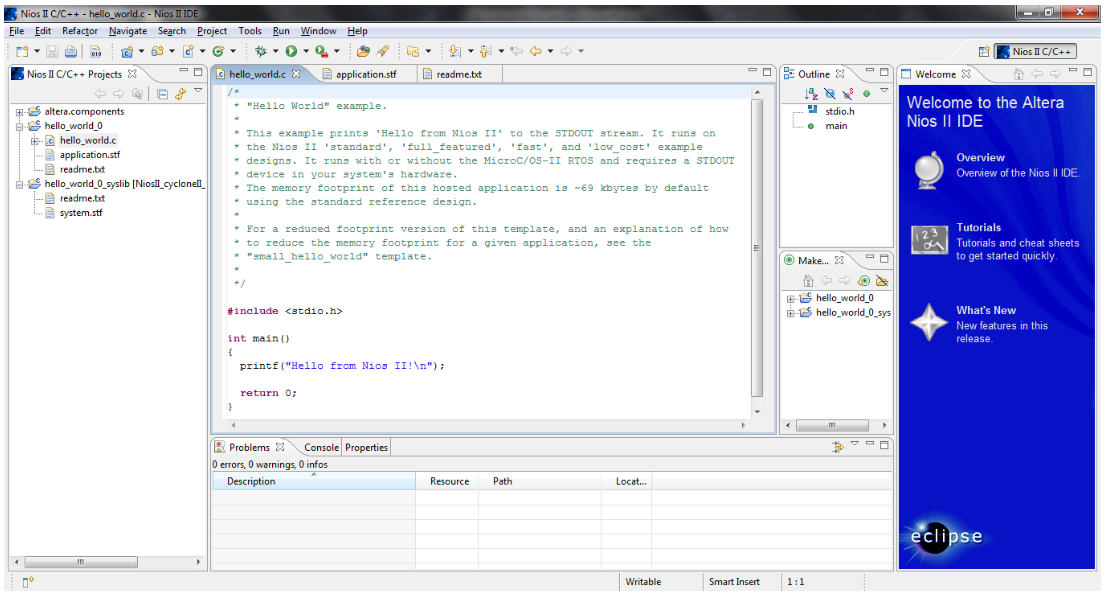


3.4 Software

Altera Corporation provides software development tools such as Eclipse IDE, Nios II IDE and so forth. These software development tools are used for writing programs for the system that we created. It provides tools to accomplish software development tasks such as editing, building, and debugging programs.

Altera Nios II IDE will be used for the software implementation design. The Nios II integrated development environment (IDE) is a primary software development tool for Nios II family of embedded processors. We can accomplish all software development tasks within the Nios II IDE, including editing, building and debugging. The Nios II IDE provides a consistent development platform that work for all Nios II processor systems. With a PC, an Altera FPGA and JTAG download cable; the whole process of developing the software for any Nios II processor system can be accomplished.

Figure 3.9: Screenshots of Nios II IDE tools (Hello world!! Example)



```
/*
 * "Hello World" example.
 *
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires a STDOUT
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an explanation of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */

#include <stdio.h>

int main()
{
  printf("Hello from Nios II!\n");

  return 0;
}
```

```
0 errors, 0 warnings, 0 infos
Description                    Resource    Path              Locat...
```

3.5 Functional Description

The designed system is a typical 128-bit AES system, whereby blocks of 128-bit data will be encrypted/decrypted at a time with a 128-bit key.

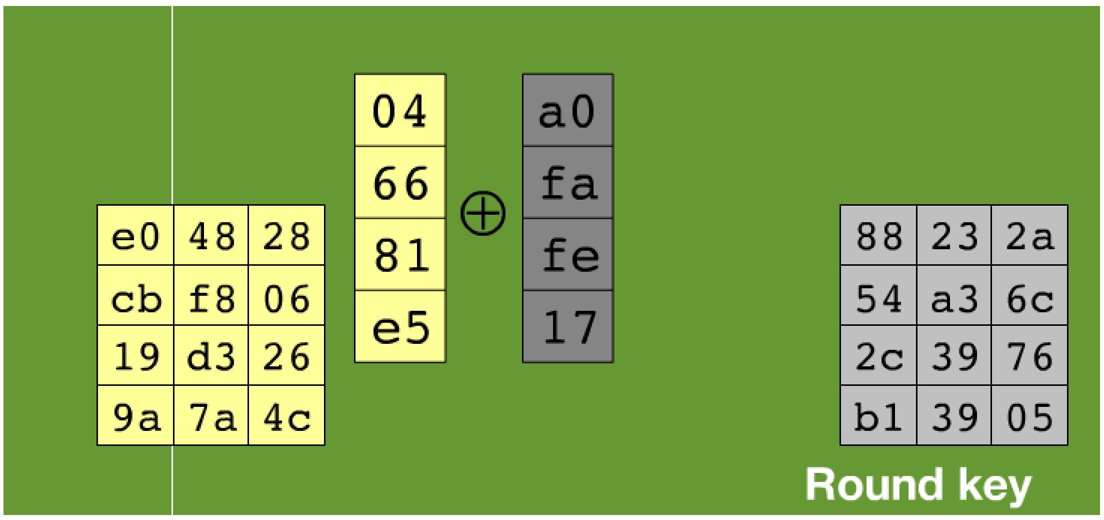
Designed AES system basically can be separated into 3 major functions: Key Expansion, Encryption and Decryption.

3.5.1 Encryption

3.5.1.1 Add Round Key

Add Round key is the transformation in which a round key is added to the State using an ex-or operation. The process of round key will be explained in the Key Expansion sections (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007).

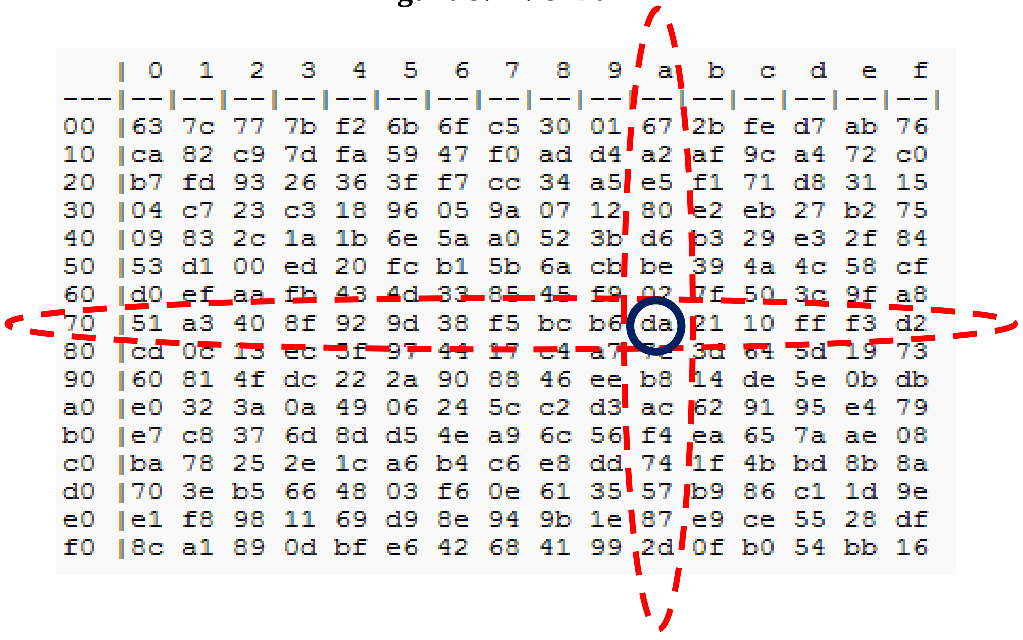
Figure 3.10: Add Round Key Transformation



3.5.1.2 SubBytes

SubBytes is the Transformation using non-linear byte substitution table (S-box) that operates on each of the bytes independently (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007). Inside each slot of 1-byte data, the input high order 4 bits or a nibble is used as the row value of the S-box, the low order 4 bits or a nibble is used as the column value of the S-box. The corresponding row and column element is taken out from the S-box as an output (Shunwen Xiao, Yajun Chen & Peng Luo, 2009). For instance, from the S-Box table below, input of hexadecimal “7a” will result hexadecimal “da”.

Figure 3.11: S-Box

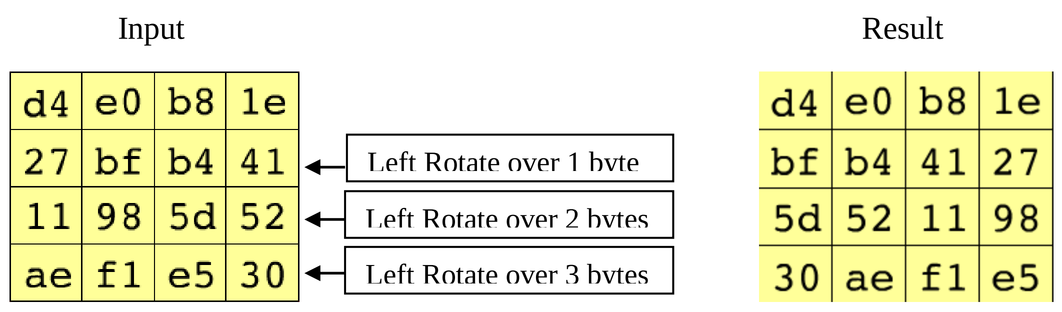


```
   | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
00 |63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab 76
10 |ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0
20 |b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15
30 |04 c7 23 c3 18 96 05 9a 07 12 80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

3.5.1.3 ShiftRows

ShiftRows is the Transformation that processes the State(refer Figure 3.1 explanation) cyclically shifting the last three rows of the State by different offsets; Row 1 is circular left shift by one place, Row 2 by two, Row 3 by three places whereas, Row 0 remains unchanged(Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007).

Figure 3.12: ShiftRows Transformation



Input

d4	e0	b8	1e
27	bf	b4	41
11	98	5d	52
ae	f1	e5	30

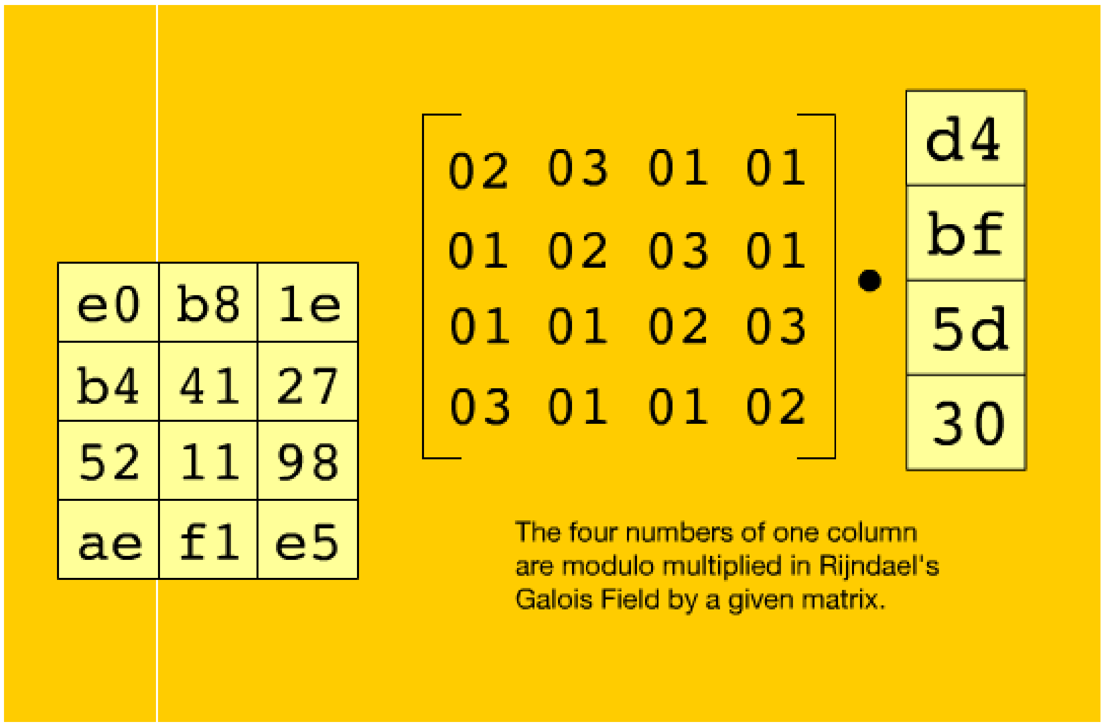
Result

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

3.5.1.4 MixColumns

MixColumns is the transformation that takes all the columns of the State and mixes their data (independently of one another) to produce new columns. Each column is considered a polynomial over GF(2^8) and multiplied modulo $X^4 + 1$ with a fixed polynomial C(x), where $C(x) = 3x^3 + x^2 + x + 2$ (Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007).

Figure 3.13: MixColums Transformation


e0 b8 1e
b4 41 27
52 11 98
ae f1 e5
02 03 01 01
01 02 03 01
01 01 02 03
03 01 01 02
d4
bf
5d
30
The four numbers of one column are modulo multiplied in Rijndael's Galois Field by a given matrix.

3.5.2 Decryption

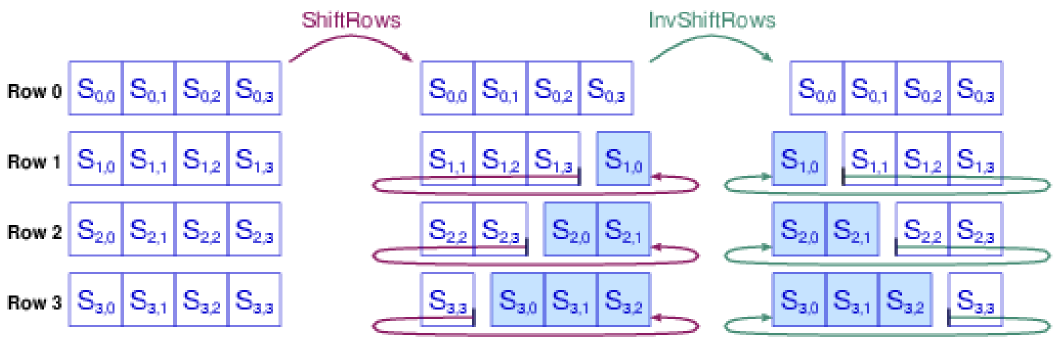
3.5.2.1 Add Round Key

As for the decryption of the Add Round Key, the sequence of the keys used for addition is no longer round key 0 until round key 10. The sequence is instead reversed, from round key 10 until round key 0.

3.5.2.2 InvShiftRows

InvShiftRows operation is similar to ShiftRows operation, but instead of rotating the bytes toward the left, now it rotates them towards the right.

Figure 3.14: Differences between ShiftRows & InvShifRows


ShiftRows
InvShiftRows
Row 0
Row 1
Row 2
Row 3
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,1}$ $S_{1,2}$ $S_{1,3}$ $S_{1,0}$
$S_{2,2}$ $S_{2,3}$ $S_{2,0}$ $S_{2,1}$
$S_{3,3}$ $S_{3,0}$ $S_{3,1}$ $S_{3,2}$
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$

3.5.2.3 InvSubBytes

InvSubBytes operates exactly the same as SubBytes operation. However, now the S-Box is replaced with the InvS-Box table.

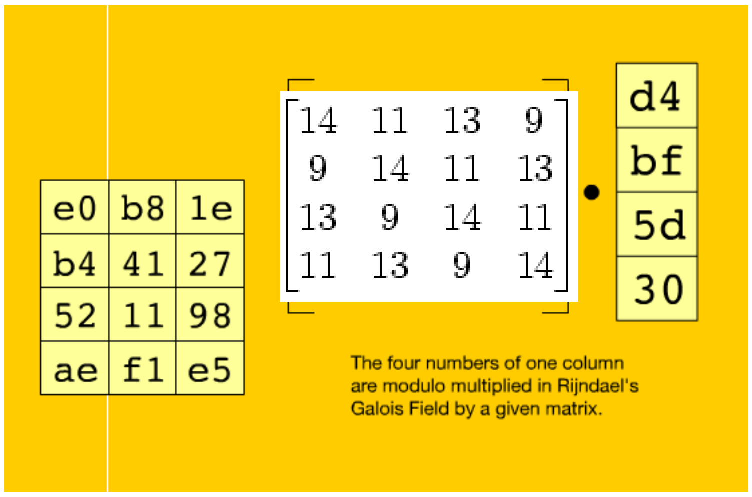
Figure 3.15: InvS-Box

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

3.5.2.4 InvMixColumns

InvMixcolumns performs the same operation as the MixColumns function. The only difference is that the polynomial used for multiplication is changed to $C^{-1}(x) = 11x^3 + 13x^2 + 9x + 14$.

Figure 3.16: InvMixColumns


e0 b8 1e
b4 41 27
52 11 98
ae f1 e5
14 11 13 9
9 14 11 13
13 9 14 11
11 13 9 14
d4
bf
5d
30
The four numbers of one column are modulo multiplied in Rijndael's Galois Field by a given matrix.

3.5.3 Key Expansion

Throughout each round, the Add Round Key function uses different keys that been expanded from a short key (cipher key). This expansion is called Rijndael key schedule. The total number of round keys required is equal to Nr+ 1 (where Nr = Number of rounds = 10). Although there are 10 rounds, eleven keys are needed because one extra key is needed in the Initial round. The key expansion algorithm uses bit-wise modulo-2 additions of 32-bit values obtained from the user key combined with byte substitution, byte rotation to right and round constants (RCons) addition(Chirag Parikh, M.S. & Parimal Patel, Ph.D, 2007). The total key schedule is 44 words (32-bits) for 128-bit key.

3.6 Program Architecture

3.6.1 Overall System Architecture

Using the SOPC builder to design Nios II system, I found through analysis and research, that Nios II(f) is the most suitable processor for the system as it has higher stage of pipelining and instruction cache which would highly increase the performance of the system. Furthermore, using this processor would actually give us higher flexibility in the future should we want to enhance our system. During the generation of the Nios II system, custom instructions and other peripherals that are required are also included, especially SDRAM as Nios II system requires a larger amount of RAM compared to other systems. Outside the Nios II processor, connecting with other custom made peripherals would complete the system. The flowchart and block diagram of the system are shown below:

Figure 3.17: System Block Diagram

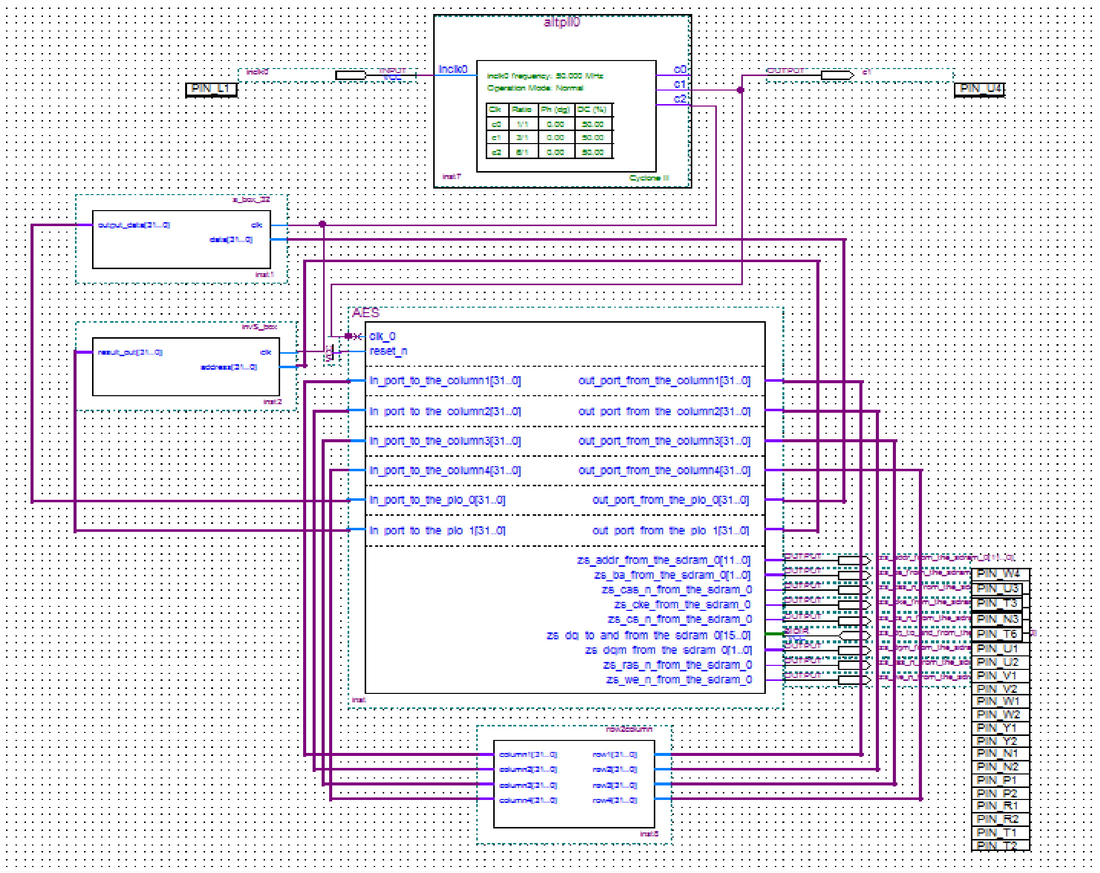
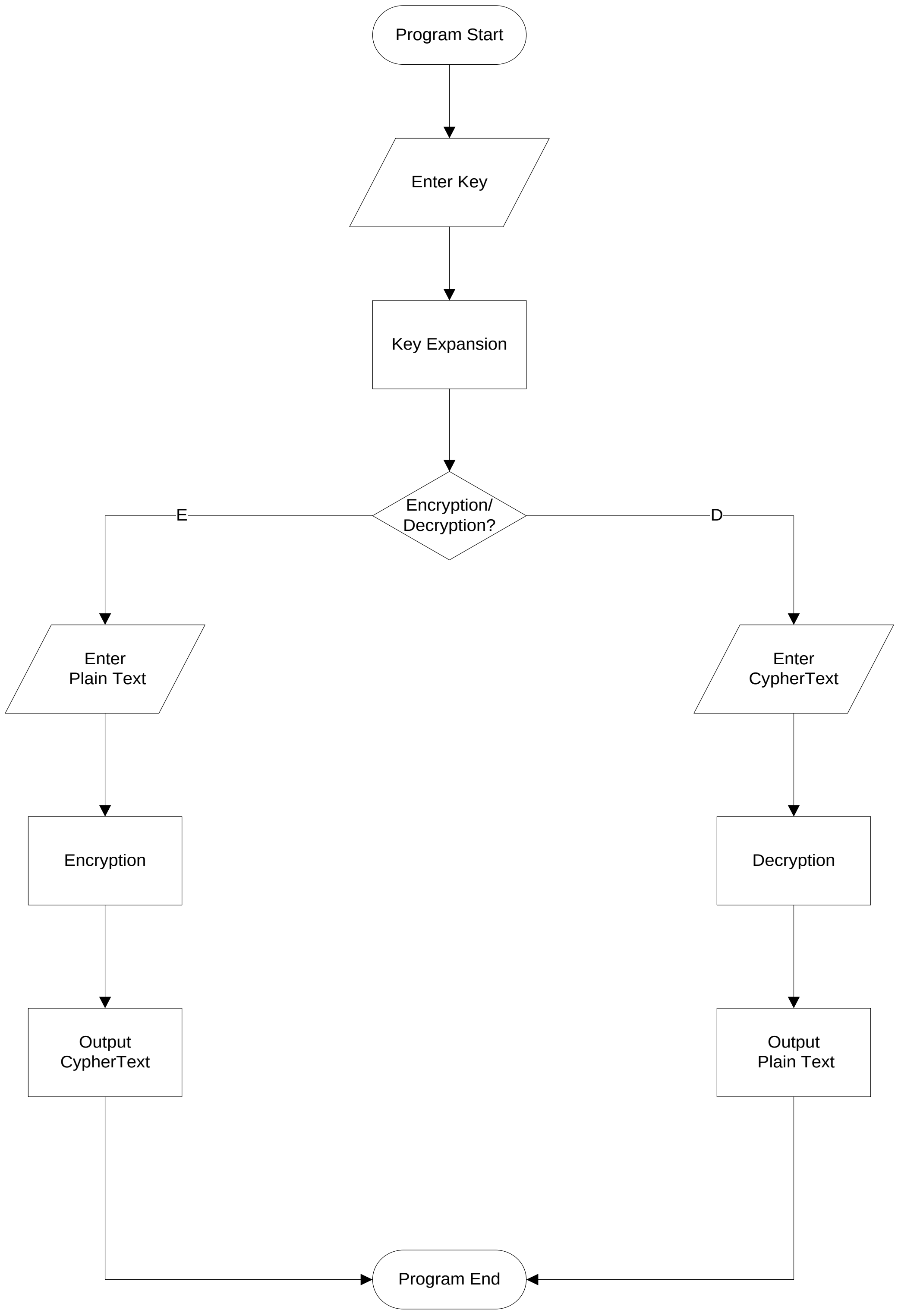


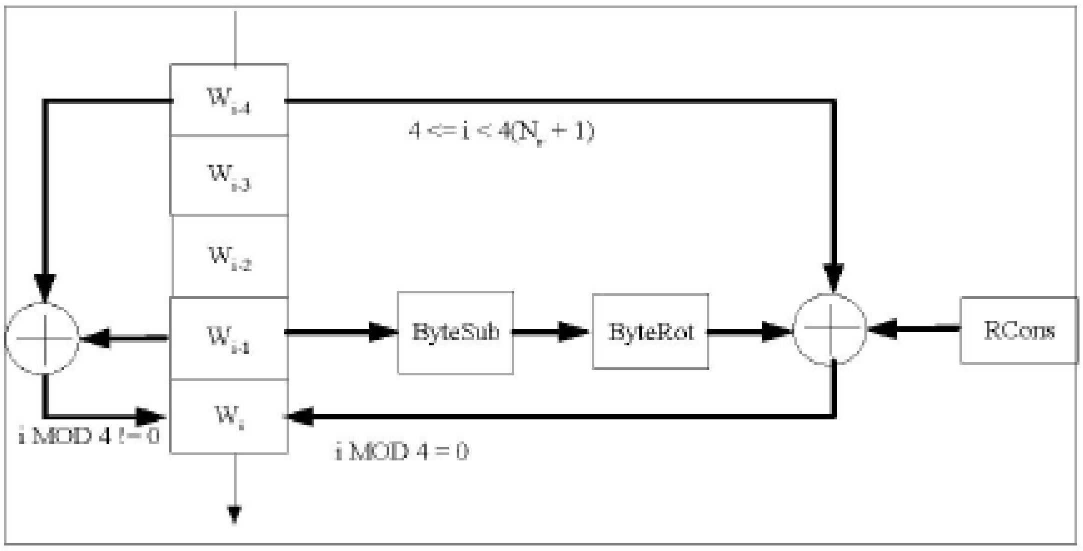
Figure 3.18: System Flow Chart



3.6.2 Key Expansion

As explained before, there are a total of 44 keys that will be expanded in the Key Expansion process. Due to the requirement in AddRoundKey process in decryption whereby the key required is in reverse order, keys will be expanded before the process of encryption/decryption started. The process of the Key Expansion is shown as below:

Figure 3.19: Key Expansion Process



The ByteSub & ByteRot is a shared function of Encryption. In enhancing the efficiency of the 32-bit Nios II processor, ByteSub has been hardware implemented, and details will be discussed in Encryption.

3.6.3 Encryption

We can divide the 4 functions of the encryption into 2 implementation categories: hardware or software. Hardware consists of MixColumn & SubBytes whereas AddRoundKey & Shiftrows are software implemented.

3.6.3.1 Software

Software implementation requires fewer resources compared to hardware implementation but with the drawback that effectiveness is lower as hardware parallel execution is faster compare to software serial execution. Due to the AddRoundKey and Shiftrow functions requiring only basic arithmetic and logical operations; software implementation will be more suitable.

3.6.3.1.1 AddRoundKey

This function does not consist of complex algorithm, with just XoR operations, the function is implemented in software.

3.6.3.1.2 ShiftRow

The data type that has been used in the software for the data is selected to be unsigned integer byte which I do believe is more efficient for a 32-bit processor. Hence in order to rotate the integer to the left, with the MSB moving to the LSB side, a customized algorithm is implemented.

Example: 5-bit rotate to left “01101011”

0	1	1	0	0	0	0	0	LSL 5-bit“01101011”
0	0	0	0	1	1	0	1	LSL 3-bit“01101011”
0	1	1	0	1	1	0	1	Bitwise OR above

Using this algorithm, rotating an integer type in software is easily achievable.

3.6.3.2 Hardware

Hardware implementation of complex functions would simplify the algorithm and indeed increase the performance of the functions.

3.6.3.2.1 SubBytes

S-BOX has been used in SubBytes function. Typical AES S-Box consists of 256 of 8-bit data, however due to the Nios II being a 32-bit processor, 4 typical AES S-Boxes are combined so that 32-bits of data can be directly mapped with the S-Box in 1 cycle. Hence the new S-Box would actually be 1Kbyte in size. This process is done by designing a 1Kbyte ROM with initialization. The ROM is then connected to the Nios II system generated by the SOPC builder in the schematic diagram.

3.6.3.2.2 MixColumn

Due to the complex mathematic operations in MixColumn, hardware implementation would be more effective. However, it was implemented differently compared to SubBytes which was implemented using Parallel IO. MixColumn makes use of custom instructions inside the Nios II. This is because we can set the number of cycles that the function in custom instruction will require to finish a job before the system reads the return result and this will ensure the precision of the returned result. Initially, MixColumn algorithm will be written in Verilog file and by using the timing analyzing, clock cycle that been require for the process to complete is identified and being specific during custom instructions integration in SOPC builder.

3.6.4 Decryption

Similar to encryption, the decryption process also consists of 4 functions: InvSubBytes, InvShiftRows & InvMixColumns, & AddRoundKey. As explained in encryption, InvShiftRow and AddRoundKey will be software implemented and InvSubBytes & InvMixColumn will be hardware implemented.

3.6.4.1 Software

3.6.4.1.1 AddRoundKey

Function is the same as in encryption, the only difference is that the key being used will be in reverse order, the 44th key would be the first key followed by 43rd, and so on. As the key expansion is done before the encryption or decryption processes, timing problems will not appear

3.6.4.1.2 InvShiftRow

The algorithm for this function is similar to encryption ShiftRow function, except that ShiftRow function rotates the data to the left while InvShiftRow rotates the data to the right.

Example: 5-bit rotate to right “01101011”

0	0	0	0	0	0	1	1	LSR 5-bit“01101011”
0	1	0	1	1	0	0	0	LSR 3-bit“01101011”
0	1	0	1	1	0	1	1	Bitwise OR above

By using the same example, we can see that by modifying the algorithm, we can achieve integer rotate operation.

3.6.4.2 Hardware

3.6.4.2.1 InvSubBytes

Due to the same reason that was mentioned in encryption, InvS-Box has been designed to match the performance of Nios II 32-bit words. Using the same approach by combining 4 InvS-Boxes, higher efficiency can be obtained.

3.6.4.2.2 InvMixColumn

Looking at the theory of AES, we see that the only difference between MixColumn and InvMixColumn is the multiplier of the matrix. Due to multiplication of the Finite Field for higher multiplier requiring more mathematical operations, InvMixColumn requires more resources compared to MixColumn. In order to minimize the resources used, I have factored out the common factor of the multiplier between MixColumn and InvMixColumn so that some resources can be shared.

Figure 3.20: InvMixColumn Multiplier

$$\begin{array}{|c|c|c|c|}\hline 0E & 0B & 0D & 09 \\ \hline 09 & 0E & 0B & 0D \\ \hline 0D & 09 & 0E & 0B \\ \hline 0B & 0D & 09 & 0E \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 0B & 08 & 0B & 08 \\ \hline 08 & 0B & 08 & 0B \\ \hline 0B & 08 & 0B & 08 \\ \hline 08 & 0B & 08 & 0B \\ \hline \end{array} + \begin{array}{|c|c|c|c|}\hline 02 & 03 & 01 & 01 \\ \hline 01 & 02 & 03 & 01 \\ \hline 01 & 01 & 02 & 03 \\ \hline 03 & 01 & 01 & 02 \\ \hline \end{array}$$

By summing the result from new finite field multiplication and MixColumn, InvMixColumn results can be obtained. This provides a better efficiency for the resources.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Result Validation

Validation of the system is done by comparing the results with AES java calculator that has been designed by Lawrie Brown from ADFA, Canberra, Australia. Details in each cycle are compared to validate the AES system that has been designed here.

Table 4.1: Key Expansion Comparison

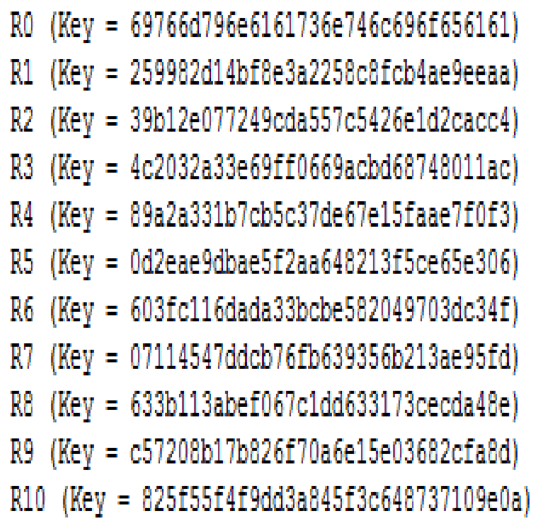
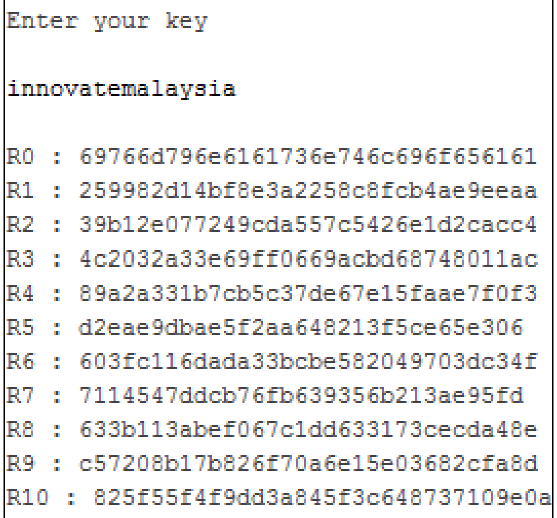
Java Calculator	Nios II FPGA
 R0 (Key = 69766d796e6161736e746c696f656161) R1 (Key = 259982d14bf8e3a2258c8fcb4ae9eeaa) R2 (Key = 39b12e077249cda557c5426e1d2cacc4) R3 (Key = 4c2032a33e69ff0669acbd68748011ac) R4 (Key = 89a2a331b7cb5c37de67e15faae7f0f3) R5 (Key = 0d2eae9dbae5f2aa648213f5ce65e306) R6 (Key = 603fc116dada33bcbe582049703dc34f) R7 (Key = 07114547ddcb76fb639356b213ae95fd) R8 (Key = 633b113abef067c1dd633173cecda48e) R9 (Key = c57208b17b826f70a6e15e03682cfa8d) R10 (Key = 825f55f4f9dd3a845f3c648737109e0a)	 Enter your key innovatemalaysia R0 : 69766d796e6161736e746c696f656161 R1 : 259982d14bf8e3a2258c8fcb4ae9eeaa R2 : 39b12e077249cda557c5426e1d2cacc4 R3 : 4c2032a33e69ff0669acbd68748011ac R4 : 89a2a331b7cb5c37de67e15faae7f0f3 R5 : d2eae9dbae5f2aa648213f5ce65e306 R6 : 603fc116dada33bcbe582049703dc34f R7 : 7114547ddcb76fb639356b213ae95fd R8 : 633b113abef067c1dd633173cecda48e R9 : c57208b17b826f70a6e15e03682cfa8d R10 : 825f55f4f9dd3a845f3c648737109e0a

Table 4.2: Encryption Comparison

Java Calculator	Nios II FPGA

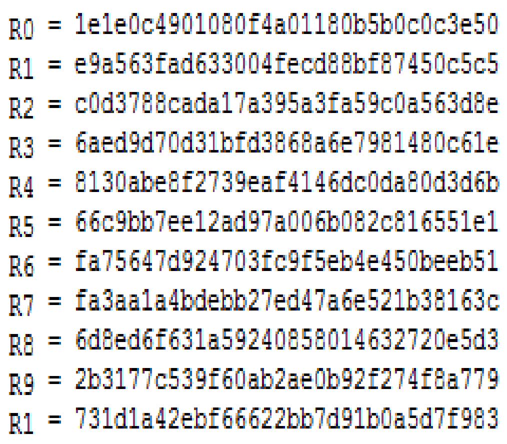
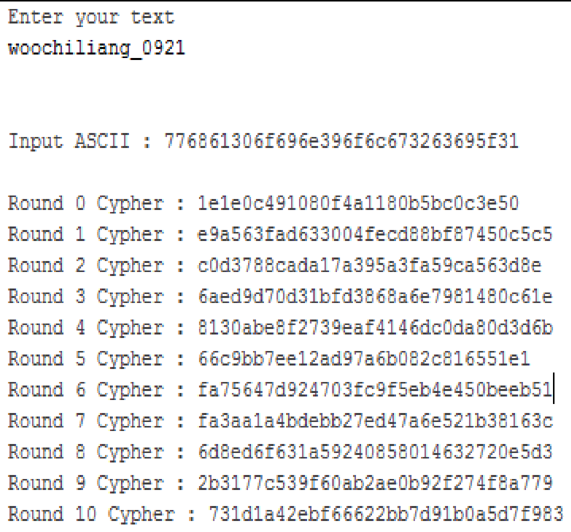
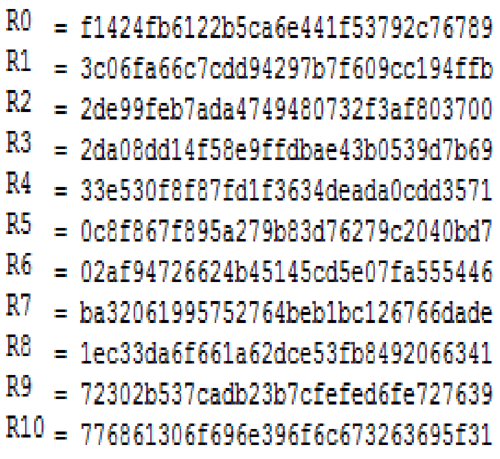
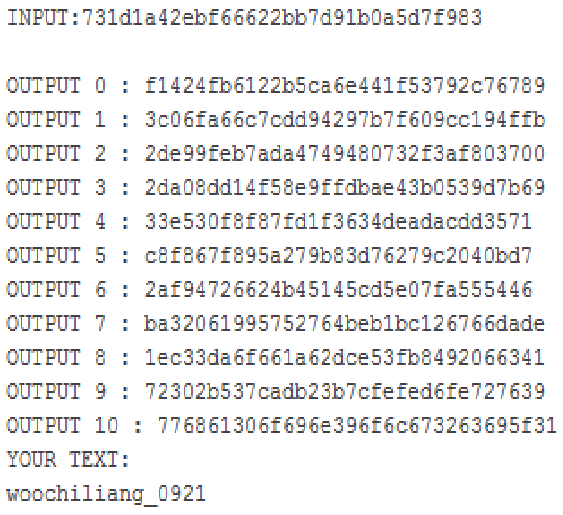

R0 = 1e1e0c4901080f4a01180b5b0c0c3e50
R1 = e9a563fad633004fecd88bf87450c5c5
R2 = c0d3788cada17a395a3fa59c0a563d8e
R3 = 6aed9d70d31bfd3868a6e7981480c61e
R4 = 8130abe8f2739eaf4146dc0da80d3d6b
R5 = 66c9bb7ee12ad97a006b082c816551e1
R6 = fa75647d924703fc9f5eb4e450beeb51
R7 = fa3aa1a4bdebb27ed47a6e521b38163c
R8 = 6d8ed6f631a59240858014632720e5d3
R9 = 2b3177c539f60ab2ae0b92f274f8a779
R1 = 731d1a42ebf66622bb7d91b0a5d7f983
	
	Enter your text
	woochiliang_0921
	
	Input ASCII : 776861306f696e396f6c673263695f31
	
	Round 0 Cypher : 1e1e0c491080f4a1180b5bc0c3e50
	Round 1 Cypher : e9a563fad633004fecd88bf87450c5c5
	Round 2 Cypher : c0d3788cada17a395a3fa59ca563d8e
	Round 3 Cypher : 6aed9d70d31bfd3868a6e7981480c61e
	Round 4 Cypher : 8130abe8f2739eaf4146dc0da80d3d6b
	Round 5 Cypher : 66c9bb7ee12ad97a6b082c816551e1
	Round 6 Cypher : fa75647d924703fc9f5eb4e450beeb51
	Round 7 Cypher : fa3aa1a4bdebb27ed47a6e521b38163c
	Round 8 Cypher : 6d8ed6f631a59240858014632720e5d3
	Round 9 Cypher : 2b3177c539f60ab2ae0b92f274f8a779
	Round 10 Cypher : 731d1a42ebf66622bb7d91b0a5d7f983

Table 4.3: Decryption Comparison

Java Calculator	Nios II FPGA


R0 = f1424fb6122b5ca6e441f53792c76789
R1 = 3c06fa66c7cdd94297b7f609cc194ffb
R2 = 2de99feb7ada4749480732f3af803700
R3 = 2da08dd14f58e9ffdbae43b0539d7b69
R4 = 33e530f8f87fd1f3634deada0cdd3571
R5 = 0c8f867f895a279b83d76279c2040bd7
R6 = 02af94726624b45145cd5e07fa555446
R7 = ba3206199575276 4beb1bc126766dade
R8 = 1ec33da6f661a62dce53fb8492066341
R9 = 72302b537cadb23b7cfefed6fe727639
R10 = 776861306f696e396f6c673263695f31
	
	INPUT:731d1a42ebf66622bb7d91b0a5d7f983
	
	OUTPUT 0 : f1424fb6122b5ca6e441f53792c76789
	OUTPUT 1 : 3c06fa66c7cdd94297b7f609cc194ffb
	OUTPUT 2 : 2de99feb7ada4749480732f3af803700
	OUTPUT 3 : 2da08dd14f58e9ffdbae43b0539d7b69
	OUTPUT 4 : 33e530f8f87fd1f3634deadacdd3571
	OUTPUT 5 : c8f867f895a279b83d76279c2040bd7
	OUTPUT 6 : 2af94726624b45145cd5e07fa555446
	OUTPUT 7 : ba32061995752764beb1bc126766dade
	OUTPUT 8 : 1ec33da6f661a62dce53fb8492066341
	OUTPUT 9 : 72302b537cadb23b7cfefed6fe727639
	OUTPUT 10 : 776861306f696e396f6c673263695f31
	YOUR TEXT:
	woochiliang_0921

From the comparison above, both systems produce the same values. Hence, we can conclude that the designed AES system is verified to be fully functional.

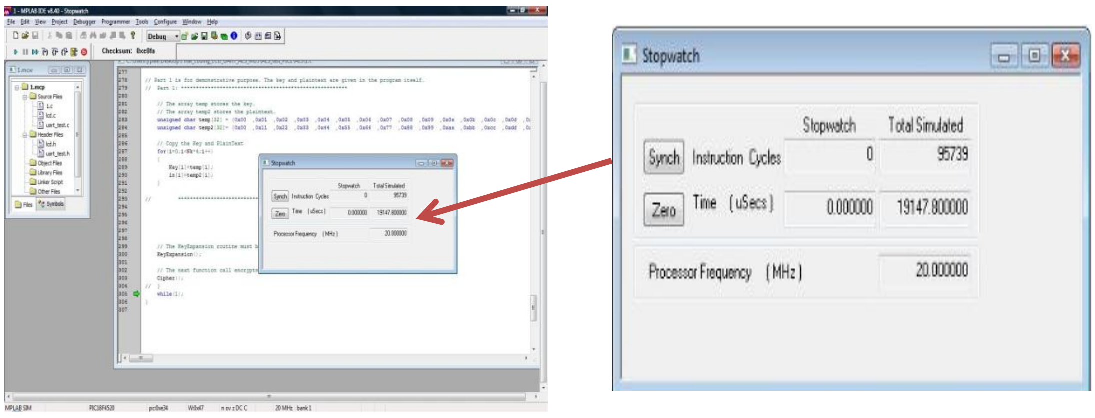
4.2 Performance Benchmark

4.2.1 Platform Benchmark

In recent years, softcore is claimed to have higher flexibility and performance compared to a microcontroller. Hence, I have chosen an AES system based on a microcontroller to be benchmarked with my softcore AES system. Fortunately, there is a student in UTAR developing AES on microcontroller for his project. Due to the similarity of our algorithms, differences in performance can be observed and compared.

Microcontroller

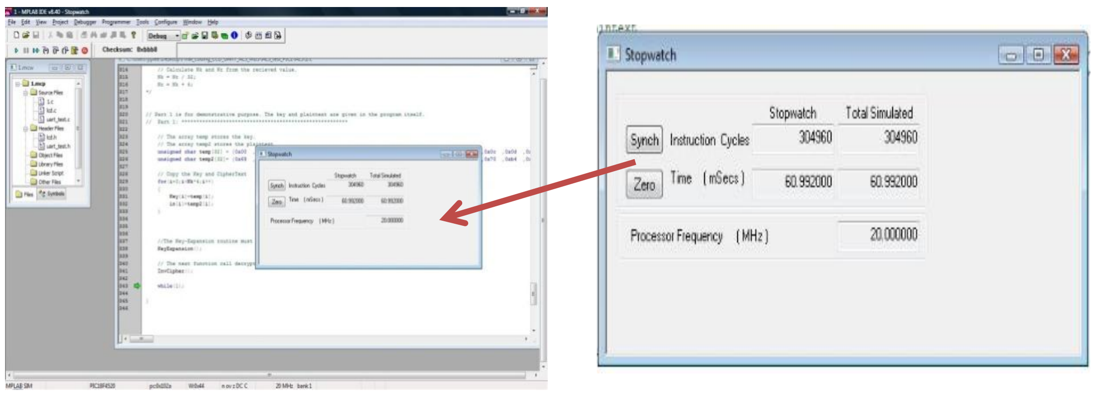
Figure 4.1: Encryption



```
Stopwatch
                                 Stopwatch     Total Simulated
Synch   Instruction Cycles               0               95739
Zero    Time   ( uSecs )          0.000000        19147.800000

Processor Frequency   ( MHz )                        20.000000
```

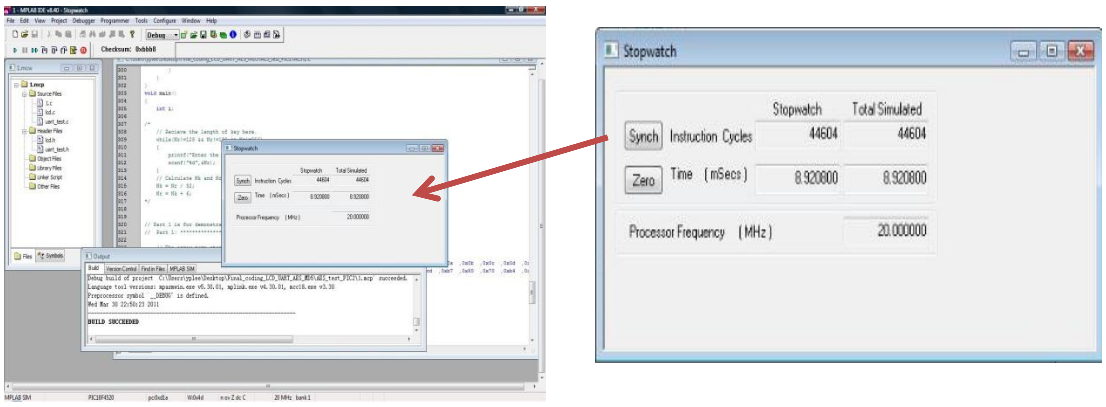
Figure 4.2: Decryption



```
Stopwatch
                                 Stopwatch     Total Simulated
Synch   Instruction Cycles          304960              304960
Zero    Time   ( mSecs )         60.992000           60.992000

Processor Frequency   ( MHz )                        20.000000
```

Figure 4.3: Key Expansion



```
Stopwatch
                                  Stopwatch     Total Simulated
Synch  Instruction Cycles            44604           44604
Zero   Time (mSecs)               8.920800        8.920800

Processor Frequency (MHz)                        20.000000
```

Nios II FPGA

Figure 4.4: Nios II Performance Counter Report

```
--Performance Counter Report--
Total Time: 16.0211 seconds  (2403172145 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|Encryption     |0.000281|    0.00005|           6752|          1|
+---------------+-----+-----------+---------------+-----------+
|Decryption     |0.000291|    0.00005|           6983|          1|
+---------------+-----+-----------+---------------+-----------+
|Key Generator  |0.0002|    0.00003|           4802|          1|
+---------------+-----+-----------+---------------+-----------+
```

From the comparison above, we can see that the AES in Nios II system is way faster than the AES system in microcontroller. There are a few reasons that we can find to explain the differences in performance. The advantage of Nios II is that it is a 32-bit processor whereas the microcontroller has an 8-bit processor. The most important factor that determines the extraordinary performance is that the Nios II system can support external custom peripherals & custom instructions which allows for some of the functions to be accelerated. Memory in the microcontroller is also limited, and this is very critical for AES system as the S-Box required in SubBytes function requires quite a bit of memory. The pipelining of instructions and the instruction cache that is found in Nios II(f) also help the system to achieve high performance.

4.2.2 Implementation Benchmark

An AES system typically can be categorie~~s~~d into fully hardware and fully software implementation. In this project, traditional method is replaced with co-design where both software and hardware are implemented into the same system for high efficiency.

Table 4.4: Fully Software Performance

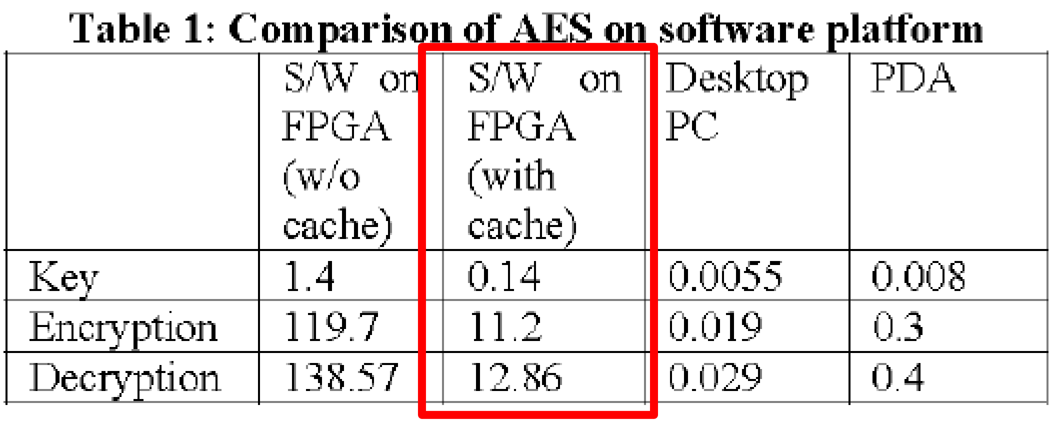


Table 1: Comparison of AES on software platform

	S/W on FPGA (w/o cache)	S/W on FPGA (with cache)	Desktop PC	PDA
Key	1.4	0.14	0.0055	0.008
Encryption	119.7	11.2	0.019	0.3
Decryption	138.57	12.86	0.029	0.4

The benchmark above is obtained from "Performance Evaluation of AES Algorithm on Various Development Platforms" (Chirag Parikh, M.S. , Parimal Patel, Ph.D., 2007). As mentioned in the article, the unit for the readings is millisecond. Looking at the FPGA with cache (Nios II(f)) performance, we can see that the performance for the fully software implementation is still slower than this project's system. This can be explained by the custom made hardware functions which are used for certain hardware acceleration. As mention in the article, the clock speed of the processor in the Desktop (Pentium 4) and PDA (UIQ emulator (ARM9)) platform is higher than FPGA applied clock speed and is faster than the fully software implementation in FPGA. However, my designed of AES system is yet faster than above platforms.

Figure 4.5: Fully Hardware Performance

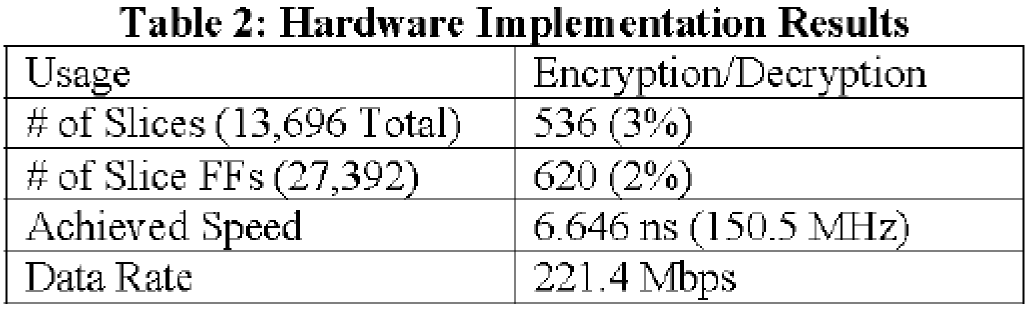


Table 2: Hardware Implementation Results

Usage	Encryption/Decryption
# of Slices (13,696 Total)	536 (3%)
# of Slice FFs (27,392)	620 (2%)
Achieved Speed	6.646 ns (150.5 MHz)
Data Rate	221.4 Mbps

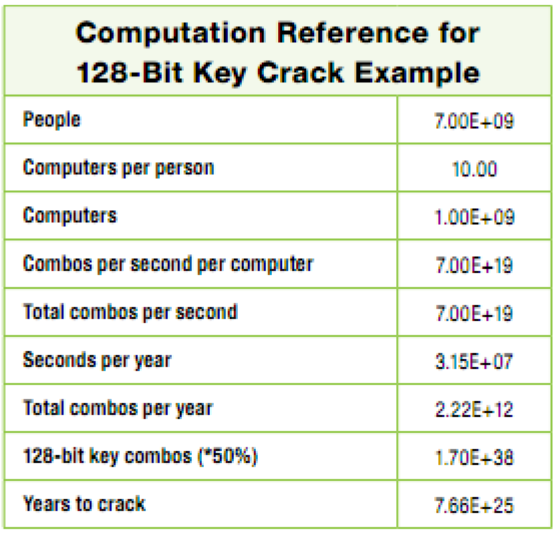
The performance for full hardware implementation is outstanding and is even faster when compared to my AES design. However, hardware implementation consumes a lot of resources and will be very costly. Furthermore, fully hardware implementation has as its largest drawback its flexibility whereby it requires add-on resources during future modifications or improvements. This problem will not affect software implementation much as we just need to add in some extra code for extra features. By using the combination of hardware and software implementation, minor future improvements or modifications that do not require hardware implementation can actually be done without adding on any LEs (Logic Element), it would save a lot of resources.

4.3 Overall Discussion

By having 128-bits of key, we would have 2 to the 128th power, or 3.4 x 10 to the 38th power numbers. Seagate Technology had come out the calculation where if presume that:

- Every person on the planet owns 10 computers
- There are 7 billion people on the planet.
- Each of these computers can test 1 billion key combinations per second.
- On average, you can crack the key after testing 50 percent of the possibilities

Table 4.5: Time require for Key Cracking



Computation Reference for 128-Bit Key Crack Example	
People	7.00E+09
Computers per person	10.00
Computers	1.00E+09
Combos per second per computer	7.00E+19
Total combos per second	7.00E+19
Seconds per year	3.15E+07
Total combos per year	2.22E+12
128-bit key combos (*50%)	1.70E+38
Years to crack	7.66E+25

it will require 77,000,000,000,000,000,000,000,000 years to crack a single key. According to NIST (National Institute of Standards and Technology), AES would be secure for at least 20-30 years.

Encryption process in fully software implementation is observable in memory, and it would give a path for the attacker to reveal the key. By using co-design where hardware and software are implemented together, key would be secure during the process.

Table 4.6: Overall Comparison Table

	Nios II FPGA (my design)	Microcontroller	Fully Software Implementation	Fully Hardware Implementation
Encryption	0.04 ms	19.00 ms	11.2 ms	6.646ns
Decryption	0.05 ms	60.00 ms	12.86 ms	6.646ns
Key	0.03 ms	8.00 ms	0.14 ms	6.646ns
Resources	5,259 LEs	-	N/A	13,696 slices

As the comparison above shows, the performance of my design is slightly better than that of a fully software implementation and is much worse compared to that of a fully hardware implementation. This can be explained as the algorithm that has been used is optimized for the fully hardware implementation. As for my algorithm, typical

AES algorithm is applied which is a disadvantage for my design. It should achieve a higher performance if the algorithm is optimized. The performance of my design is also slower than my expectation as I thought that it would be faster compared to the fully software implementation and only slightly slower than the fully hardware implementation.

Typical AES system takes in hexadecimal as its input, this would be very troublesome as people would have to find the hexadecimal representation for their input. As for my design of AES, the input to the system is character type where symbols or characters will be converted into hexadecimal based on their ASCII code. This is more convenient compared to a typical AES system.

From the comparison table, my design of AES scores the speed of 3.2Mbps for encryption and 2.56Mbps for decryption. Hence it can be applied in devices which require moderate performance with limited resources such as VOIP, Radio Frequency device, ATM machine, transceiver, video conferencing, etc. With the performance that been achieved, typical home-based internet usage or WIFI communication can be supported.

There are two major bugs that can be found in my design: spacing input problem, overflow input problem. As these two scenarios occur, the process of my system will result abnormal. These 2 problems is due to the usage of "scanf("%s")" as input command where according to Wikipedia, "*scanf("%s") scan a character string. The scan terminates at whitespace. A null character is stored at the end of the string, which means that the buffer supplied must be at least one character longer than the specified input length.*" This means that input shall not contain any spacing in between else the scanning will be terminated by the spacing. As declared input length for my design to be 16 words (1 byte each), overflow input would cause the system to be malfunction. These can be fix by replacing "scanf("%s")" with other input command such as fgets or fscanf. However further research on the functionality and characteristic of the command shall be done before being applied so that similar bugs won't occur.

The advantages where further application or function can be built directly on the Nios II system without much modification can save a lot of resources and space compared to hardware implementation and the performance is faster compared to fully software or a microcontroller platform.

I have tried 3 types of Nios II processor during the implementation and Nios II (f) gives the best performance and it can support up to 150Mhz. Despite the performance, resources of Nios II(f) is just slightly higher compare to Nios II(e) & Nios II(s) . The major disadvantage of this soft core is where it require license from Altera Technology for commercial purpose.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

There are few conclusions that we can draw from this project “Implementation of Soft Core” and the system for the implementation is AES, Advance Encryption Standard.

AES on Nios II system is not as effective as it expected to be. The reason is where major flow of the system is still software implemented and the algorithm that been applied is a typical AES algorithm where optimizations are not applied.

Efficiency of the system is acceptable where compare to a fully hardware implementation system (with optimization) where it require more than 10000 slices of resources, my design only require approximately 5000 slices of resources in the FPGA.

Soft core have a significant advantages in performance and resources compare to a microcontroller. From the result, soft core system performs at least 3 times faster than the microcontroller system. However, it is known that FPGA is more expensive compare to a microcontroller. Hence, only system which requires higher performance spec is recommend to design on soft core, FPGA system.

Designed AES system has been validated on its functionality with comparison with with AES java calculator that has been designed by Lawrie Brown from ADFA, Canberra, Australia. The result is positive and is conclude to be fully functional

There are still some bugs in the software where spacing in the sentence and overflow of the input are not allowed for the system. As explained in discussion, these bugs can be fixed by replacing scanf with other command.

Nios II(f) is found to be most suitable soft core for the system and having the highest specification among the Nios II family provides us with higher flexibility in future improvement. Any software application or design can develop directly on the Nios II system without any extra resources.

My design of AES system score 3.2Mbps in encryption and 2.56Mbps in decryption whereby normal audio or video communication can be secure with real time operation.

5.2 Recommendation

Future improvement has to implement for the system commercialization. There are few recommendations that I think would help in system improvement.

Research on AES algorithm shall be done for finding the optimize algorithm for the software/hardware codesign platform. By implementing the optimized algorithm, it is believe that the performance can dramatically improve.

Current design of the system is using standard input (keyboard) as the input interface. In future, other transmission interface can be used for replacement. This would allow the user to transfer their file to the FPGA for encryption/decryption, which would be more convenient. Serial port interface would be recommended as the DE1 is supplied with RS232 port. PCI Express interface can be used if require high speed transmission. However, PCI Express interface would be more difficult to program and use compare to RS232 port.

As mention earlier, audio or video transmission application can be applied to the system. DE1 development board contain DSP (Digital Signal Processing) core and it can be integrated to the system if necessary. Integrating a DSP core, audio or video processing can be done through the FPGA.

REFERENCES

(n.d.). Retrieved 2010, from Animation of AES: http://www.formaestudio.com/rijndaelinspector/

Advanced Encryption Standard. (n.d.). Retrieved 2010, from Wikipedia: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Arif Irwansyah, Vishnu P. Nambiar, Mohamed Khalil-Hani. (2009). An AES Tightly Coupled Hardware Accelerator in an FPGA-based.

Ashwini M. Deshpande, Mangesh S. Deshpande and Devendra N. Kayatanavar. (2009). FPGA Implementation of AES Encryption and Decryption.

Chen JianHong, Liu Yu, and Chia-Hau Shiu. (2005). *High Aberrance AES System Using a Reconstructable Function Core Generator.* Taiwan.

Chih-Peng Fan and Jun-Kui Hwang. (2007). Implementations of High Throughput Sequential and Fully Pipelined AES.

Chirag Parikh, M.S. , Parimal Patel, Ph.D. (2007). Performance Evaluation of AES Algorithm on Various Development Platforms.

Gürkaynak, F. K. (n.d.). *Cryptographic Accelerators*. Retrieved 2010, from GALS System Design:Side Channel Attack Secure Cryptographic Accelerators: http://www.iis.ee.ethz.ch/~kgf/acacia/c3.html#tth_sEc3.3.2

Jason G. Tong, Ian D. L. Anderson and Mohammed A. S. Khalid. (2006). Soft-Core Processors for Embedded Systems.

Jianzhuang Wang, Youping Chen, Jingming Xie, Bing Chen, Haiping Lin. (2008). System Structure for FPGA-Based SOPC Design Using Hard Tasks.

Kuan Jen Lin, Chin-Mu Hsiao and Ching Hung Jhan. (2009). Exploring HW/SW Codesign of AES Algorithm Using Custom Instructions.

Nios II. (n.d.). Retrieved 2010, from Wikipedia: http://en.wikipedia.org/wiki/Nios_II

Ralf Joost, Ralf Salomon. (2005). Advantages of FPGA-Based Multiprocessor Systems in Industrial Applications.

scanf. (n.d.). Retrieved from cplusplus.com: http://www.cplusplus.com/reference/clibrary/cstdio/scanf/

Shunwen Xiao, Yajun Chen, Peng Luo. (2009). The Optimized Design of Rijndael Algorithm Based on SOPC.

Somsak Choomchuay1, Surapong Pongyupinpanich, and Somsanouk Pathumvanh3. (2008). A Compact 32-bit Architecture for an AES System.

Sopc builder. (n.d.). Retrieved 2010, from Wikipedia: http://en.wikipedia.org/wiki/Sopc_builder

Stalling, W. (2003). *Cryptography and Network Security*. New Jersey: Person Education.

Techonology, S. (n.d.). *128-Bit Versus 256-Bit.* Retrieved from http://www.seagate.com/staticfiles/docs/pdf/whitepaper/tp596_128-bit_versus_256_bit.pdf

Wikipedia. (n.d.). *Scanf.* Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Scanf

APPENDICES

APPENDIX A: Verilog File (S-BOX)

```
/* This module is designed to combine four 256-bytes ROM so that the system can
perform 32-bit substitution*/

module s_box_32
(clk,data,output_data);
input clk;
input [31:0] data;
output[31:0] output_data;

lpm_rom0 s_box1(      //lpm_rom0 is a 256-byte ROM with initialization
        .address(data[7:0]),
        .clock(clk),
        .q(output_data[7:0]));

lpm_rom0 s_box2(
        .address(data[15:8]),
        .clock(clk),
        .q(output_data[15:8]));

lpm_rom0 s_box3(
        .address(data[23:16]),
        .clock(clk),
        .q(output_data[23:16]));

lpm_rom0 s_box4(
        .address(data[31:24]),
        .clock(clk),
        .q(output_data[31:24]));
endmodule
```

APPENDIX B: Verilog File (Inverse S-Box)

```
/* This module is designed to combine four 256-bytes ROM so that the system can
perform 32-bit substitution*/

module invS_box
(clk,address,result_out);

input clk;
input [31:0] address;
output [31:0] result_out;

lpm_rom2 invsbox1    //lpm_rom2 is a 256-byte ROM with initialization
(.clock(clk),
.address(address[31:24]),
.q(result_out[31:24]));

lpm_rom2 invsbox2
(.clock(clk),
.address(address[23:16]),
.q(result_out[23:16]));

lpm_rom2 invsbox3
(.clock(clk),
.address(address[15:8]),
.q(result_out[15:8]));

lpm_rom2 invsbox4
(.clock(clk),
.address(address[7:0]),
.q(result_out[7:0]));

Endmodule
```

APPENDIX C: Verilog File (256-byte ROM)

/*This function is generated using Altera Mega Function. This is S-box ROM module. Inverse S-Box ROM module is similar with S-box ROM module. The only different is the initialization file name and the module name*/

```
`timescale 1 ps / 1 ps
module lpm_rom0 (
        address,
        clock,
        q);

        input   [7:0]  address;
        input     clock;
        output  [7:0]  q;

        wire [7:0] sub_wire0;
        wire [7:0] q = sub_wire0[7:0];

        altsyncram      altsyncram_component (
                                .clock0 (clock),
                                .address_a (address),
                                .q_a (sub_wire0),
                                .aclr0 (1'b0),
                                .aclr1 (1'b0),
                                .address_b (1'b1),
                                .addressstall_a (1'b0),
                                .addressstall_b (1'b0),
                                .byteena_a (1'b1),
                                .byteena_b (1'b1),
                                .clock1 (1'b1),
                                .clocken0 (1'b1),
                                .clocken1 (1'b1),
                                .clocken2 (1'b1),
                                .clocken3 (1'b1),
                                .data_a ({8{1'b1}}),
                                .data_b (1'b1),
                                .eccstatus (),
                                .q_b (),
                                .rden_a (1'b1),
                                .rden_b (1'b1),
                                .wren_a (1'b0),
                                .wren_b (1'b0));
        defparam
                altsyncram_component.clock_enable_input_a = "BYPASS",
                altsyncram_component.clock_enable_output_a = "BYPASS",
`ifdef NO_PLI
                altsyncram_component.init_file = "S_BOX.rif" //initialization file
`else
                altsyncram_component.init_file = "S_BOX.hex"//initialization file
`endif
,
                altsyncram_component.intended_device_family = "Cyclone II",
                altsyncram_component.lpm_hint =
"ENABLE_RUNTIME_MOD=NO",
                altsyncram_component.lpm_type = "altsyncram",
                altsyncram_component.numwords_a = 256,
                altsyncram_component.operation_mode = "ROM",
                altsyncram_component.outdata_aclr_a = "NONE",
                altsyncram_component.outdata_reg_a = "CLOCK0",
                altsyncram_component.widthad_a = 8,
                altsyncram_component.width_a = 8,
                altsyncram_component.width_byteena_a = 1;


endmodule
```

APPENDIX D: Verilog File (MixColumn)

```
module mixcolum_en_32

(data_in,clk,result_out);

input clk;
input [31:0] data_in;
output[31:0] result_out;
wire [7:0]
result_2i,result_3i,result_2ii,result_3ii,result_2iii,result_3iii,result_2iv,result_3iv;

GF_2x mix2i
(.data(data_in[31:24]),
.clk(clk),
.result(result_2i));

GF_3x mix3i
(.data(data_in[23:16]),
.clk(clk),
.result(result_3i));

assign result_out[31:24] = result_2i ^ result_3i ^ data_in[15:8] ^ data_in[7:0];

GF_2x mix2ii
(.data(data_in[23:16]),
.clk(clk),
.result(result_2ii));

GF_3x mix3ii
(.data(data_in[15:8]),
.clk(clk),
.result(result_3ii));
```

```
assign result_out[23:16] = result_2ii ^ result_3ii ^ data_in[31:24] ^ data_in[7:0];

GF_2x mix2iii
(.data(data_in[15:8]),
.clk(clk),
.result(result_2iii));

GF_3x mix3iii
(.data(data_in[7:0]),
.clk(clk),
.result(result_3iii));

assign result_out[15:8] = result_2iii ^ result_3iii ^ data_in[31:24] ^ data_in[23:16];

GF_2x mix2iv
(.data(data_in[7:0]),
.clk(clk),
.result(result_2iv));

GF_3x mix3iv
(.data(data_in[31:24]),
.clk(clk),
.result(result_3iv));

assign result_out[7:0] = result_2iv ^ result_3iv ^ data_in[23:16] ^ data_in[15:8];
endmodule


module GF_2x
(data,clk,result);

input clk;
input [7:0] data;
output reg [7:0] result;
```

```
always @(posedge clk)
begin
result = data<<1;
if(data[7] == 1)
result=result ^ 8'b00011011;
end
endmodule

module GF_3x
(data,clk,result);

input clk;
input [7:0] data;
output reg [7:0] result;

always @(posedge clk)
begin
result = data<<1;
if(data[7] == 1)
result=result ^ 8'b00011011;
result = result ^ data;
end
endmodule
```

APPENDIX E: Verilog File (InvMixCoulumn Factor)

```
module invmixcolumn
(clk,data_in,mix_data,result);

input clk;
input [31:0] data_in,mix_data;
wire [15:0] result_12_8x;
output [31:0] result;

invmixcolumn_12x_8x invmix
(.clk(clk),
.data_in(data_in),
.result(result_12_8x));

assign result[31:24] = mix_data[31:24] ^ result_12_8x[15:8];
assign result[23:16] = mix_data[23:16] ^ result_12_8x[7:0];
assign result[15:8] = mix_data[15:8] ^ result_12_8x[15:8];
assign result[7:0] = mix_data[7:0] ^ result_12_8x[7:0];

endmodule

module invmixcolumn_12x_8x
(data_in,clk,result);

input [31:0] data_in;
input clk;
output [15:0] result;
wire [7:0]
result_12i,result_8i,result_8ii,result_12ii,result_8iii,result_12iii,result_12iv,result_8i
v;
```

```
GF_12x invmix12i
(.clk(clk),
.data(data_in[31:24]),
.result(result_12i));

GF_8x invmix8i
(.clk(clk),
.data(data_in[23:16]),
.result(result_8i));

GF_12x invmix12ii
(.clk(clk),
.data(data_in[15:8]),
.result(result_12ii));

GF_8x invmix8ii
(.clk(clk),
.data(data_in[7:0]),
.result(result_8ii));

assign result[15:8] = result_12i ^ result_8i ^ result_12ii ^ result_8ii;

GF_8x invmix8iii
(.clk(clk),
.data(data_in[31:24]),
.result(result_8iii));

GF_12x invmix12iii
(.clk(clk),
.data(data_in[23:16]),
.result(result_12iii));

GF_8x invmix8iv
(.clk(clk),
.data(data_in[15:8]),
.result(result_8iv));

GF_12x invmix12iv
(.clk(clk),
.data(data_in[7:0]),
.result(result_12iv));

assign result[7:0] = result_12iii ^ result_8iii ^ result_12iv ^ result_8iv;

endmodule

module GF_8x
(data,clk,result);

input [7:0] data;
input clk;
output [7:0] result;
wire [7:0] result_temp1,result_temp2;

GF_2x gf2
(.clk(clk),
.data(data),
.result(result_temp1));

GF_2x gf4
(.clk(clk),
.data(result_temp1),
.result(result_temp2));

GF_2x gf8
(.clk(clk),
.data(result_temp2),
.result(result));
```

```
Endmodule
module GF_12x
(data,clk,result);

input [7:0] data;
input clk;
output [7:0] result;
wire [7:0] result_temp1,result_temp2;

GF_2x gf2
(.clk(clk),
.data(data),
.result(result_temp1));

GF_2x gf4
(.clk(clk),
.data(result_temp1),
.result(result_temp2));

GF_3x gf12
(.clk(clk),
.data(result_temp2),
.result(result));

endmodule
```

APPENDIX F: AES system C Code

```
/*
Copyright (C) 2010-2011 Woo Chi Liang
This is an AES system that been develop using soft-core with
hardware acceleration.
Please email woochiliang@msn.com for details
 *
 */

#include <stdio.h>
#include <system.h>
#include <stdlib.h>
#include <altera_avalon_pio_regs.h>
#include <string.h>
#include <io.h>
#include<altera_avalon_performance_counter.h>



///FUNCTION
INITIALIZATION//////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////
void character_handler (char* char_store);
void Key_Scheduler(unsigned int KEY_IN[4]);
void encryption(unsigned int TEXT_IN[4]);
unsigned int rotate_left(unsigned int data,int shift);
void character_handler2(char* char_store);
void column2row(unsigned int input[4]);
void decryption(unsigned int CYPHER_IN[4]);
////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////

///GLOBAL VARIABLE
DECLARATION/////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////
char char_swap[4][4],char_swap2[4][4];

char input_key [][4]=
{{0x2b,0x28,0xab,0x09},{0x7e,0xae,0xf7,0xcf},{0x15,0xd2,0x15,0x4f},{
0x16,0xa6,0x88,0x3c}};
char input_text [][4] =
{{0x32,0x88,0x31,0xe0},{0x43,0x5a,0x31,0x37},{0xf6,0x30,0x98,0x07},{
0xa8,0x8d,0xa2,0x34}};
char input_cypher[][4] =
{{0x39,0x02,0xdc,0x19},{0x25,0xdc,0x11,0x6a},{0x84,0x09,0x85,0x0b},{
0x1d,0xfb,0x97,0x32}};
unsigned int cypherkey[44];
unsigned int process_data[4];
unsigned int
RCON[]={0x01000000,0x02000000,0x04000000,0x08000000,0x10000000,0x200
00000,0x40000000,0x80000000,0x1B000000,0x36000000};
unsigned int process_data_int[4];

////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
int main()
{
      char key [4][4],text[4][4];
      char* char_pointer;
      unsigned int test_cypher[4];
      int select;
      //char key_swap [4][4];
      //unsigned int char_int[4];
      //unsigned int KEY_IN[4],i,j;
       PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
      while(1)
      {
      printf("\nEnter your key\n");
      fflush(stdin);
      char_pointer = &key[0][0];
      scanf("%s",char_pointer);
      character_handler(&key);
      Key_Scheduler(char_swap);

      printf("\n 1:Encryption \n 2:Decryption");
      scanf("%d",&select);
      if(select == 1)
      {
                  printf("Enter your text               ");
                  fflush(stdin);
                  scanf("%s",&text);
                  character_handler2(&text);
                  encryption(char_swap2);
                  //character_handler2(&process_data[0]);
                  printf("\nYOUR
CYPHERTEXT:\n%x %x %x %x\n",process_data[0],process_data[1],process_
data[2],process_data[3]);
      }

      else if (select == 2)
      {
            printf("Enter your text                  ");/*
            fflush(stdin);
            scanf("%x",&test_cypher[0]);
            fflush(stdin);
            scanf("%x",&test_cypher[1]);
            fflush(stdin);
            scanf("%x",&test_cypher[2]);
            fflush(stdin);
            scanf("%x",&test_cypher[3]);*/

      scanf("%x %x %x %x",&test_cypher[0],&test_cypher[1],&test_cyph
er[2],&test_cypher[3]);
            //character_handler2(&text);
            decryption(test_cypher);
            //character_handler2(process_data);

            character_handler2(&process_data[0]);
            /*
            process_data[0] = rotate_left(process_data[0],3);
            process_data[1] = rotate_left(process_data[1],3);
            process_data[2] = rotate_left(process_data[2],3);
            process_data[3] = rotate_left(process_data[3],3);
*/

            printf("\nYOUR TEXT:\n%s\n",char_swap2);
```

```
            perf_print_formatted_report(
            (void *)PERFORMANCE_COUNTER_0_BASE, // Peripheral's HW
base address
            alt_get_cpu_freq(), // defined in "system.h"
            3, // How many sections to print
            "Encryption", // Display-names of sections
            "Decryption",
            "Key Generator");
            }
      }

      /*
      for(i=0;i<=3;i++)
      {
            for(j=0;j<=3;j++)
            {
                  key_swap[j][3-i]=key[i][j];
            }
      }
      for(i=0;i<=3;i++)
      printf("%x%x%x%x\n",key_swap[i][0],key_swap[i][1],key_swap[i][
2],key_swap[i][3]);
      */
/*
      character_handler(input_key);
      Key_Scheduler(char_swap);
      //character_handler2(input_text);
      //encryption(char_swap2);
      character_handler2(input_cypher);
      decryption(char_swap2);
*/

  return 0;
}

void Key_Scheduler(unsigned int KEY_IN[4])
{

             PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);

      unsigned int j,y,x,KEY_TEMP,i;
      unsigned int KEY_OUT[10][4];

      PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,3);
      cypherkey[0]=KEY_IN[0];
      cypherkey[1]=KEY_IN[1];
      cypherkey[2]=KEY_IN[2];
      cypherkey[3]=KEY_IN[3];
      //printf("\nR0 : %x%x%x%x\n",cypherkey[0],cypherkey[1],cypherk
ey[2],cypherkey[3]);
      for(j=1;j<=10;j++)
      {
            KEY_TEMP = KEY_IN[0];
            IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, KEY_IN[3]);
            KEY_IN[0] = IORD_ALTERA_AVALON_PIO_DATA(PIO_0_BASE);
            KEY_IN[0] = rotate_left(KEY_IN[0],1);
            KEY_IN[0] = KEY_IN[0] ^ RCON[j-1] ^ KEY_TEMP;
            for(i=1;i<=3;i++)
                  KEY_IN[i] = KEY_IN[i-1] ^ KEY_IN[i];
            cypherkey[j*4]=KEY_IN[0];
            cypherkey[(j*4)+1]=KEY_IN[1];
            cypherkey[(j*4)+2]=KEY_IN[2];
            cypherkey[(j*4)+3]=KEY_IN[3];

      //printf("R%d : %x%x%x%x\n",j,cypherkey[(j*4)+0],cypherkey[(j*
4)+1],cypherkey[(j*4)+2],cypherkey[(j*4)+3]);
      }
      PERF_END(PERFORMANCE_COUNTER_0_BASE,3);
}

unsigned int rotate_left(unsigned int data,int shift)
{
      unsigned int result;
      result = (data << (shift*8) | (data >> (32 - (shift*8))));

      return result;
}

unsigned int rotate_right(unsigned int data,int shift)
{
      unsigned int result;
      result = (data >> (shift*8) | (data << (32 - (shift*8))));

      return result;
}

void character_handler(char* char_store)
{

      int i,j;
      for(i=0;i<=3;i++)
            {
                  for(j=0;j<=3;j++)
                  {
                        char_swap[j][3-i]=*((char_store+j)+(i*4));
                  }
            }

}

void character_handler2(char* char_store)
{
      int i,j;
      for(i=0;i<=3;i++)
                  {
                        for(j=0;j<=3;j++)
                        {
                              char_swap2[i][3-j]=
*((char_store+j)+(i*4));
                        }
                  }
}

void encryption(unsigned int TEXT_IN[4])
{
      int round;

      PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,1);
      //printf("\n\n%x\n%x\n%x\n%x\n",TEXT_IN[0],TEXT_IN[1],TEXT_IN[
2],TEXT_IN[3]);

      column2row(TEXT_IN);
```

```
	//printf("\n\nInput 
ASCII : %x%x%x%x",process_data[0],process_data[1],process_data[2],pr
ocess_data[3]);
	process_data[0]^=cypherkey[0];
	process_data[1]^=cypherkey[1];
	process_data[2]^=cypherkey[2];
	process_data[3]^=cypherkey[3];

	//a[0]=rotate_left(TEXT_IN[0],0);
	//printf("\n\nRound 0 
Cypher : %x%x%x%x",process_data[0],process_data[1],process_data[2],p
rocess_data[3]);
	column2row(process_data);
	//printf("\n\n%x\n%x\n%x\n%x",process_data[0],process_data[1],
process_data[2],process_data[3]);


	for(round=1;round<=10;round++)
	{

	     IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,  process_data[0]);
	      process_data[0]=IORD_ALTERA_AVALON_PIO_DATA(PIO_0_BASE);
	     IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,  process_data[1]);
	      process_data[1]=IORD_ALTERA_AVALON_PIO_DATA(PIO_0_BASE);
	     IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,  process_data[2]);
	      process_data[2]=IORD_ALTERA_AVALON_PIO_DATA(PIO_0_BASE);
	     IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE,  process_data[3]);
	      process_data[3]=IORD_ALTERA_AVALON_PIO_DATA(PIO_0_BASE);
	      //printf("\n\nSUB-
BYTE\n\n%x\n%x\n%x\n%x",process_data[0],process_data[1],process_data
[2],process_data[3]);
	      process_data[1]=rotate_left(process_data[1],1);
	      process_data[2]=rotate_left(process_data[2],2);
	      process_data[3]=rotate_left(process_data[3],3);

	//printf("\n\nSHIFT\n\n%x\n%x\n%x\n%x",process_data[0],process
_data[1],process_data[2],process_data[3]);

	      column2row(process_data);
	      if(round!=10)
	      {

	process_data[0]=ALT_CI_MIXCOLUMN_EN_32_INST(process_data[0]);

	process_data[1]=ALT_CI_MIXCOLUMN_EN_32_INST(process_data[1]);

	process_data[2]=ALT_CI_MIXCOLUMN_EN_32_INST(process_data[2]);

	process_data[3]=ALT_CI_MIXCOLUMN_EN_32_INST(process_data[3]);
	      }

	//printf("\n\nMIX\n\n%x\n%x\n%x\n%x",process_data[0],process_d
ata[1],process_data[2],process_data[3]);
	      process_data[0]^=cypherkey[0+(round*4)];
	      process_data[1]^=cypherkey[1+(round*4)];
	      process_data[2]^=cypherkey[2+(round*4)];
	      process_data[3]^=cypherkey[3+(round*4)];
```

```
			//printf("\nRound %d
Cypher : %x%x%x%x",round,process_data[0],process_data[1],process_dat
a[2],process_data[3]);
			column2row(process_data);
		}
		//printf("\n\CYPHER
TEXT:\n\n%x\n%x\n%x\n%x",process_data[0],process_data[1],process_dat
a[2],process_data[3]);

		PERF_END(PERFORMANCE_COUNTER_0_BASE,1);


		//printf("\nCYPHER
TEXT:\n\n%x\n%x\n%x\n%x",process_data[0],process_data[1],process_dat
a[2],process_data[3]);
}

void decryption(unsigned int CYPHER_IN[4])
{
		int round;
		unsigned int temp[4];
		PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE,2);
		column2row(CYPHER_IN);
		//printf("\n\nINPUT:%x%x%x%x",process_data[0],process_data[1],
process_data[2],process_data[3]);
		process_data[0]^=cypherkey[40];
		process_data[1]^=cypherkey[41];
		process_data[2]^=cypherkey[42];
		process_data[3]^=cypherkey[43];
		//printf("\n\nOUTPUT
0 : %x%x%x%x",process_data[0],process_data[1],process_data[2],proces
s_data[3]);
		column2row(process_data);
		for(round=9;round>=0;round--)
			{
				process_data[1] = rotate_right(process_data[1],1);
				process_data[2] = rotate_right(process_data[2],2);
				process_data[3] = rotate_right(process_data[3],3);
				IOWR_ALTERA_AVALON_PIO_DATA(PIO_1_BASE,
process_data[0]);

		process_data[0]=IORD_ALTERA_AVALON_PIO_DATA(PIO_1_BASE);
				IOWR_ALTERA_AVALON_PIO_DATA(PIO_1_BASE,
process_data[1]);

		process_data[1]=IORD_ALTERA_AVALON_PIO_DATA(PIO_1_BASE);
				IOWR_ALTERA_AVALON_PIO_DATA(PIO_1_BASE,
process_data[2]);

		process_data[2]=IORD_ALTERA_AVALON_PIO_DATA(PIO_1_BASE);
				IOWR_ALTERA_AVALON_PIO_DATA(PIO_1_BASE,
process_data[3]);

		process_data[3]=IORD_ALTERA_AVALON_PIO_DATA(PIO_1_BASE);
				column2row(process_data);
				process_data[0]^=cypherkey[0+(round*4)];
				process_data[1]^=cypherkey[1+(round*4)];
				process_data[2]^=cypherkey[2+(round*4)];
				process_data[3]^=cypherkey[3+(round*4)];
				if(round!=0)
				{
```

```
	temp[0]=ALT_CI_MIXCOLUMN_EN_32_INST(process_data[0]);

	temp[1]=ALT_CI_MIXCOLUMN_EN_32_INST(process_data[1]);

	temp[2]=ALT_CI_MIXCOLUMN_EN_32_INST(process_data[2]);

	temp[3]=ALT_CI_MIXCOLUMN_EN_32_INST(process_data[3]);
				process_data[0] =
ALT_CI_INVMIX_32_NEW_INST(process_data[0],temp[0]);
				process_data[1] =
ALT_CI_INVMIX_32_NEW_INST(process_data[1],temp[1]);
				process_data[2] =
ALT_CI_INVMIX_32_NEW_INST(process_data[2],temp[2]);
				process_data[3] =
ALT_CI_INVMIX_32_NEW_INST(process_data[3],temp[3]);

			}
			//printf("\nOUTPUT %d : %x%x%x%x",((round-9)*-
1)+1,process_data[0],process_data[1],process_data[2],process_data[3]
);
			column2row(process_data);
		}
	PERF_END(PERFORMANCE_COUNTER_0_BASE,2);
}

void column2row(unsigned int input[4])
{
	IOWR_ALTERA_AVALON_PIO_DATA(COLUMN1_BASE, input[0]);
				IOWR_ALTERA_AVALON_PIO_DATA(COLUMN2_BASE,
input[1]);
				IOWR_ALTERA_AVALON_PIO_DATA(COLUMN3_BASE,
input[2]);

	IOWR_ALTERA_AVALON_PIO_DATA(COLUMN4_BASE,input[3]);

	process_data[0]=IORD_ALTERA_AVALON_PIO_DATA(COLUMN1_BASE);

	process_data[1]=IORD_ALTERA_AVALON_PIO_DATA(COLUMN2_BASE);

	process_data[2]=IORD_ALTERA_AVALON_PIO_DATA(COLUMN3_BASE);

	process_data[3]=IORD_ALTERA_AVALON_PIO_DATA(COLUMN4_BASE);
}
```