Performance Evaluation of AES Algorithm on Various Development Platforms

Chirag Parikh, M.S.
Department of Electrical and Computer Engineering
University of Texas at San Antonio
One UTSA circle
San Antonio, TX 78249
cparikh2000@gmail.com

Parimal Patel, Ph.D.
Department of Electrical and Computer Engineering
University of Texas at San Antonio
One UTSA circle
San Antonio, TX 78249
parimal.patel@utsa.edu

Abstract

With the rapid growth of low-cost, low-power handheld devices with wireless capability, more and more devices connect to wireless networks. This warrants secured data exchange and authentication to take place during the connection. 802.11i security scheme is becoming a de-facto standard of future WLAN, which uses AES algorithm in both encryption and authenticity of payload data. Cryptographic transformations of AES are computationally intensive, consuming significant power. Most of the published AES implementations target high-end products with multi-gigabit throughputs and have no regards for power consumption. Low-end products, including handheld devices like PDA rarely need more than 200 Mbps data transfer rate which necessitates alternative implementation of AES algorithm, with the aim of reducing power consumption and increasing the handsets' battery life.

In this paper we study and compare the outcome obtained from implementing AES algorithm on various platforms. We present results obtained through profiling the implemented algorithm on platforms like FPGA, Desktop PC (Microsoft Visual Studio) and handheld device (UIQ SDK for Symbian OS along with Metrowerks CodeWarrior). Our comparison shows that the 32-bit hardware implementation of AES on FPGA supports the required throughput and consumes less power allowing fully charged PDA to remain connected to WLAN for a longer duration.

Keywords: Cryptography, FPGA, AES, WLAN, SDK, Symbian, Encryption, Decryption

I. Introduction

The stupendous growth of the Internet and the explosive growth in computer systems and their interconnections via networks have increased the dependence of both organizations and individuals on the information stored and communicated using these systems. However, at the same time, it also brought about a plethora of new issues and concerns, chief among them being the need to protect data and resources from disclosure, guarantying the authenticity of data and messages, and protecting systems from network based attacks [2]. In the past, encryption algorithms such as DES had been sufficient to handle most security needs, but it became apparent that the time of DES had quickly approached its end. With the limitations of DES's 56-bit key and the advent of faster computers, DES could no longer be considered a secure algorithm. In recent years, Triple-DES which uses 168-bit key, has taken the place of DES in an attempt to create stronger security without compromising currently accepted encryption standards. For many purposes, however, Triple-DES is simply too slow [5]. With the eminent demise of DES, the National Institute of Standards and Technology (NIST) had issued a request for submissions of stronger encryption algorithms to replace the aging DES. These algorithms had to conform to several strict requirements: it must be a block cipher with longer key length, larger block size, fast in computation and greater flexibility. After several rounds of submissions and eliminations, the NIST had narrowed the applicant pool down to five finalists out of which Advanced Encryption Standard (AES) algorithm, also known as Rijndael algorithm, was selected. While such algorithms are relatively simple and efficient to implement on high-performance processors found in current desktop and laptop computers, such may not be the case for the smaller processors found in current palm computing devices or embedded systems. Handheld computing devices of today are as powerful as the old desktop PCs costing as little as one tenth. This has been made possible by advances in CPU, memory and integration technologies. Unfortunately, battery technology has not kept pace with this development. The slow improvement in battery lifetime shows that energy consumption is and will be one of the most important factors in designing embedded systems and portable devices participating in wireless networking [1].

Wireless Local Area Network (WLAN) solutions are being increasingly adopted by various industry segments both locally and globally. Wireless LAN products can provide LAN users with access to real-time information anywhere in their organization. This mobility supports productivity and service opportunities not possible with wired networks. The most appealing aspect of WLAN is

its convenience, flexibility and roaming. A user is not tied down to a LAN and can move around with relative ease while staying connected. WLANs are also easy to install. An entire network can be put together in a matter of hours rather than days without digging, drilling holes or altering construction. Finally, WLAN may be installed where rewiring is impractical. Wireless systems can be installed in different environments and users can communicate with the existing wired network through access points or wireless adapters.

Although WLANs solve some problems that exist in traditional wired LANs, they also introduce new security and power management issues [1]. The risks that WLAN services present can only be mitigated rather than completely eliminated. Although there is no single solution for perfect WLAN security, it is a strong belief that WLAN security can be enhanced to an acceptable level by a proper combination of countermeasures. As interest in wireless grew, IEEE expanded the plethora of standards under 802.11 by developing specific sub-standards with different specifications for bandwidth, frequency and transmission technologies [3]. 802.11i is an emerging standard and in the long term, necessitates a framework for using AES for its encryption services.

With the ever-expanding market for palm computers, and the inevitable networking of palm devices, it appears that encryption on limited processors may warrant closer investigation. The primary objective of this paper is to study and compare the outcome obtained from implementing AES algorithm on various platforms. In next section the algorithm is described, followed by a section on design methodology and considerations needed for implementing the algorithm. Section IV presents the design and implementation results.

II. AES Algorithm

The AES algorithm is a block cipher that encrypts blocks of 128, 192, or 256 bits using symmetric keys of 128, 192 or 256 bits. The standard requires that the AES must implement a symmetric block cipher with a block size of 128 bits, hence larger block sizes of 192 and 256 bits has not been implemented. Electronic Codebook Mode (ECB) has been used in the design. ECB mode simply takes the piece off the input message one block (128 bits) at a time and encrypts it using the same key until no more pieces are left. This process is shown in **Figure 2**, which displays the computation for both serial (one block at a time) and parallel encryption. Serial encryption and decryption supports slower data rate, whereas parallel encryption/decryption provides higher data rate using more resources.

Operations in AES algorithm are performed on a two-dimensional byte array of four rows and four columns or *State* as shown in **Figure 1**.

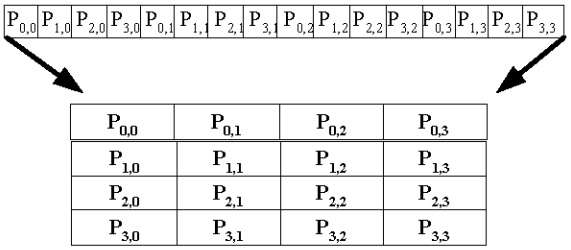

Figure 1: State Rectangular Array

The algorithm consists of Initial round, nine uniform rounds, followed by Final round [4]. The Initial round performs **AddRoundKey** function (as explained below). The reason that the rounds have been listed as "nine uniform rounds followed by Final round" is because the Final round involves a slightly different manipulation from the others. The nine uniform rounds consist of four functions:

- **ByteSub** - Transformation using non-linear byte substitution table (S-box) that operates on each of the bytes independently
- **ShiftRow** – Transformation that processes the State 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.
- **MixColumn** – 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
  $C(x) = '03'x^3 + '01'x^2 + '01'x + '02'$
- **AddRoundKey** – Transformation in which a round key is added to the State using an ex-or operation.

The Final round only performs ByteSub, ShiftRow and AddRoundKey transformations.

Round keys K_i are derived from the 128-bit user key by means of the key schedule [6]. It consists of two components: key expansion and round key selection. The total number of round keys required is equal to $N_r + 1$ (where N_r = 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 as shown in **Figure 3**.

There are two ways decryption engines can be designed; one is straightforward method and second is equivalent method, In the straightforward decryption method, the sequence of the transformations differs from that of the encryption approach. It comprises of an inverse of the final round, inverses of the nine rounds, followed by the initial data/key addition. The inverse of the round

is found by inverting each of the transformations in the round. The inverse of ByteSub is obtained by applying the inverse of the affine transformation and taking the multiplicative inverse in $GF(2^8)$ of the result. In the inverse of the ShiftRow transformation, row 0 is not shifted, row 1 is shifted left by three places, row 2 by two places and row 3 by one place. The polynomial, C(x), used to transform the state columns in the inverse of MixColumn is given by,

$$C(x) = \text{`0B'}x^3 + \text{`0D'}x^2 + \text{`09'}x + \text{`0E'}$$

The same set of keys are used in encryption and decryption process, but are used in reverse order. In the equivalent approach that the decryption has the same sequence of transformations as encryption (with transformations replaced by their inverses) but with different set of keys. The decryption keys are generated by keeping the first and the last 128-bit encryption sub-keys as it is and performing InvMixColumn operation on remaining intermediate 128-bit sub-keys.

III. Design Methodology and Implementation

The aim of this paper is to measure the performance of the AES algorithm on various platforms like FPGA, Desktop PC and handheld device, and based on the results recommend which platform implementation supports the required throughput. In design and implementation of any system, it is very important that the target technology resources are exploited to full extent to achieve highest performance in terms of speed and area.

Pure Hardware Implementation of AES in FPGA

To begin with, we first developed the algorithm in pure hardware using Xilinx ISE 8.1i tools [12]. The algorithm is implemented in Xilinx's Virtex-IIPro (XC2VP30ff896-6) FPGA [11]. The system is modeled in Verilog HDL, synthesized using Xilinx's XST synthesis tools, verified using Modeltech's Modelsim 6.0d simulator, and implemented using Xilinx's Place and Route tools integrated in ISE 8.1i tools. The targeted FPGA has configurable logic blocks (CLB), BlockRAM (BRAM), DCM, and selectable I/O blocks (IOB) as the major resources. Each CLB consists of four slices, and each slice consists of two function generators and two storage elements. The function generators F & G are configurable as 4-input look-up tables (LUTs), as 16-bit shift registers, or as 16-bit distributed SelectRAM+ memory. In addition, the two storage elements are either edge-triggered D-type flip-flops or level-sensitive latches. In our design we used them as flip-flops. Next, we explain how these resources are exploited.

Once data packet is received either from outside (inbound) or from application (outbound) and stored in BRAM by the receiver engine, a *start* signal is generated. On receiving the *start* signal, the AES core decides based on the data transfer direction, whether to perform encryption (outbound) or decryption (inbound) and sends back an appropriate *acknowledge* signal. It then takes the first 128-bit data from the BRAM (32 bits at a time) and then passes it on to the Initial round. Since the State bytes (Data) are operated on individually, each AES round requires 8-bit by 8-bit LUTs. If implemented as LUTs, additional slice resources will be used up. BRAMs can be used for the same purpose as they are provided by the family and will be wasted if unused. This can save the slices for other logic. In the Initial round, sub-key is XORed with the data and then sent to the First round. In first round, S-box is implemented as look-up table (LUT) or ROM. This proves faster and more cost-effective method than implementing the multiplicative inverse operation and affine transformation. Next comes the ShiftRow and MixColumn operations, which are nothing but rearrangement of bits and simple AND and XOR operations. In our implementation of the algorithm, we have developed the hardware for one encryption/decryption round and re-used the same piece of hardware to complete the whole encryption/decryption process as shown in **Figure 4**.

Each encryption/decryption round has been implemented as 32-bit architecture as compared to 128-bit architecture. A significant advantage of the 32-bit architecture of the design is to reduce the number of S-Boxes from four in 128-bit implementation to one instance in 32-bit implementation at the expense of four clock cycles. The 128-bit data takes four clock cycles to enter the first round from the BRAM (in chunks of 32 bits), then each round takes eight clock cycles and final round takes seven clock cycles. As there are nine rounds and a final round, so in all ((9 * 8) + (1 * 7)) = 79 clock cycles are used. Final data takes four clock cycles to store into the BRAM. Thus overall it takes 87 clock cycles to encrypt/decrypt one set of 128-bit data.

Pure Software Implementation of AES in FPGA

Next step was to implement the algorithm in software and still use FPGA as the target technology. The algorithm was developed using Xilinx Platform Studio 8.1i using C programming language. The reason for selecting C as programming language was because compiled high-level languages like C are generally better adapted to optimizing performance than interpreted language like Java and the development tool supports C and C++. A program translated by compiler is often much faster than an interpreter executing the same program[10]. Therefore a software implementation of AES algorithm in standard C would produce better performance results than say a Java implementation. In this approach the design had two functions: (i) The sub-key generation and (ii) The encryption/decryption process as shown in **Figure 5**. The sub-key generation process involves bit-wise additions modulo 2 of 32-bit values obtained from user key combined with byte substitution, byte rotation and round constants (RCons) addition. After reading the control packet containing the key, the sub-key generation function starts sub-key generation process and 44 32-bit sub-keys are generated as output and stored in memory. In

order to use decryption keys in the same order as encryption keys, we generated the decryption keys (different than the encryption keys) and stored them in the memory just below the encryption sub-keys. The decryption keys were generated by keeping the first and the last 128-bit sub-keys as it is and performing InvMixColumn operation on remaining intermediate 128-bit sub-keys. Thus all the sub-keys were generated before hand and stored in the memory for a given connection between source IP and destination IP. Upon completion of sub-key generation, data is either encrypted or decrypted depending on whether it is inbound or outbound data. The 128-bit data coming from memory into the encrypt/decrypt function was operated in a serial fashion, taking 32 bits at a time. However, sub-functions like Byte-Sub and Shift row are performed on 128-bit data whereas, sub-functions like AddRoundKey and MixColumn are performed on 32 bits at a time. The final encrypted/decrypted data is stored in the memory in a serial fashion i.e. 32 bits at a time. The design was developed using two approaches: one without enabling any form of cache and one with instruction and data cache enabled. The purpose of enabling the caches was to provide fast access to frequently used program instruction and data.

Software Implementation on Desktop PC

We then ported the same developed C code using Visual C++ 6.0 compiler and targeted it to Desktop PC. The code and design was similar to the above mentioned software approach except the environment was different.

Software Implementation on Symbian OS

Last we targeted the above developed code to Mobile platform with Symbian as operating system. The popularity of the Symbian operating system coupled with the excellent developer support was the key factor in our choice of application development environment. Along with this, it also provides an opportunity to represent a targeted device type. UIQ and Series 60 are the two user interfaces available for Symbian OS for which third-party developers can write C/C++ applications. The simulator runs under Metrowerks CodeWarrior IDE.

IV. Design and Implementation Results

The AES algorithm implementation was carried out in XC2VP30 FPGA hardware, software using XC2VP30 FPGA, on Desktop PC (Pentium 4) and on UIQ emulator (ARM9) which is similar to mobile environment. The Table 1 shows the time taken, in milliseconds, by AES algorithm in software to encrypt/decrypt one full-size packet i.e. 1504 bytes of data on different platforms. As can be seen from the table, none of the software implementations of AES on FPGA support data rate of more than 10 Mbps and is therefore not suitable for current WLAN security protocol.

However, implementation of AES on Desktop PC and PDA environment supports the sustained data rate of 245 Mbps and 40 Mbps respectively. Since the sustained date rate of 54 Mbps is never achieved, the current implementation on PDA environment supports the required data throughput. But for future WLAN, software implementation will not be sufficient and will require a separate piece of hardware to perform the cryptographic operations. Thus we also implemented the algorithm in pure hardware and the results obtained are shown in Table 2.

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

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

Hardware implementation easily supports the current data throughput of wireless networks and by using VirtexII-Pro FPGA of higher speed grade, it can easily support future data rates up to 250 Mbps.

V. Conclusion

This paper presents an efficient implementation of AES block cipher on different platforms like FPGA, Desktop PC and handheld device. Algorithm was successfully designed, developed and implemented on Xilinx's XC2VP30 device, Desktop PC and mobile environment. Multi-cycle technique was employed to process 128-bits data in 32-bits chunk in both hardware and software. The operations were implemented in serialized and concurrent fashion to meet the current IEEE 802.11 operating data rates of 54Mbps and 108Mbps (Super G mode). Our results show that only pure hardware implementation of AES algorithm on FPGA supports the required throughput and is suitable for hand-held devices.

References:

1. Wang Shunman, Tao Ran, Wang Yue and Zhang Ji, "*WLAN and It's Security Problems,*" Parallel and Distributed Computing, Applications and Technologies, Proceedings of the Fourth International Conference, Chengdu, China, August 27-29, 2003.

2. William Stallings, *Cryptography and Network Security: Principles and Practice,* 2nd ed., Prentice-Hall, Inc. 2000.
3. Joon S. Park and Derrick Dicoi, "*WLAN Security: Current and Future,*" IEEE Internet Computing, September 2003.
4. FIPS, *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, November 26, 2001.
5. Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
6. John Carroll, *Computer Security*, 3rd ed., Butterworth-Heinemann 1997.
7. Pawel Chadowiec and Kris Gaj, *"Very Compact FPGA Implementations of AES Algorithm,"* CHES, Proceedings, LNCS, Vol. 2779, pp 319-333, Cologne, Germany, September 2003.
8. Sumio Morioka and Akashi Satoh, *"An Optimised S-Box Circuit Arcitecture for Low power AES design,"* Cryptographic Hardware and Embedded Systems – CHES, 4th International Workshop Redwood Shores, CA, USA, August 13-15, 2002.
9. Jon Edney and William A. Arbaugh, *Real 802.11 Security Wi-Fi Protected Access and 802.11i*, 3rd print, Addison-Wesley, September 2004.
10. K. Aoki and H. Lipmaa, "Fast implementation of AES candidates" Third AES candidate conference, New York City, USA, April 2000.
11. Xilinx Virtex-II Pro Datasheet available at www.xilinx.com
12. Xilinx ISE 8.1i tools information available at www.xilinx.com

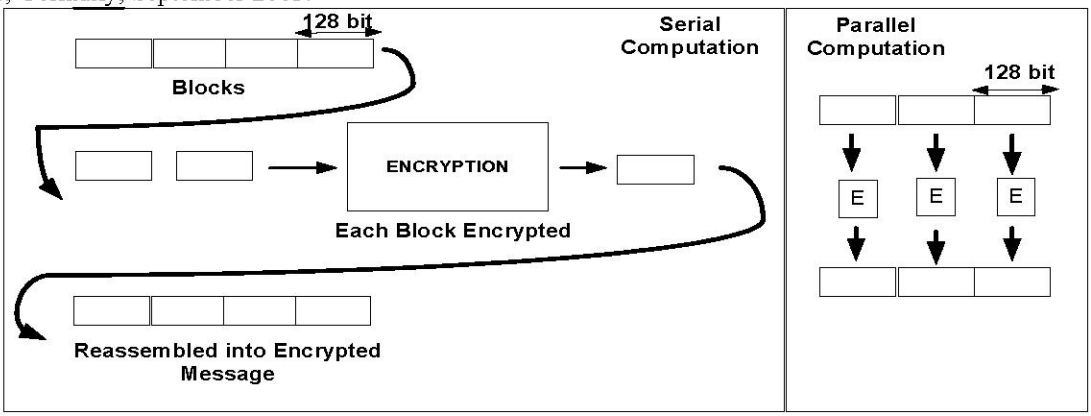

Figure 2: ECB Operating Mode

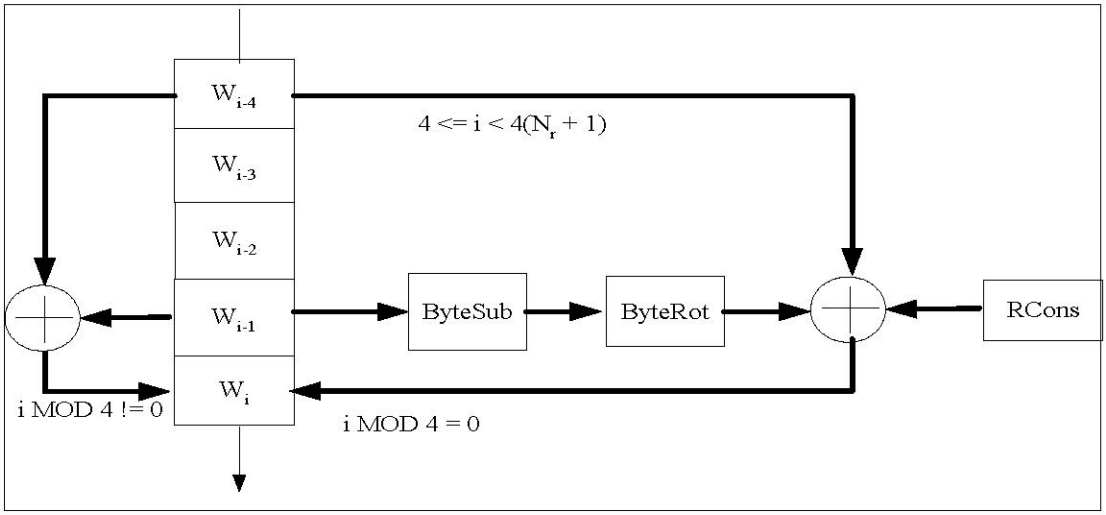

Figure 3: Key Expansion Algorithm

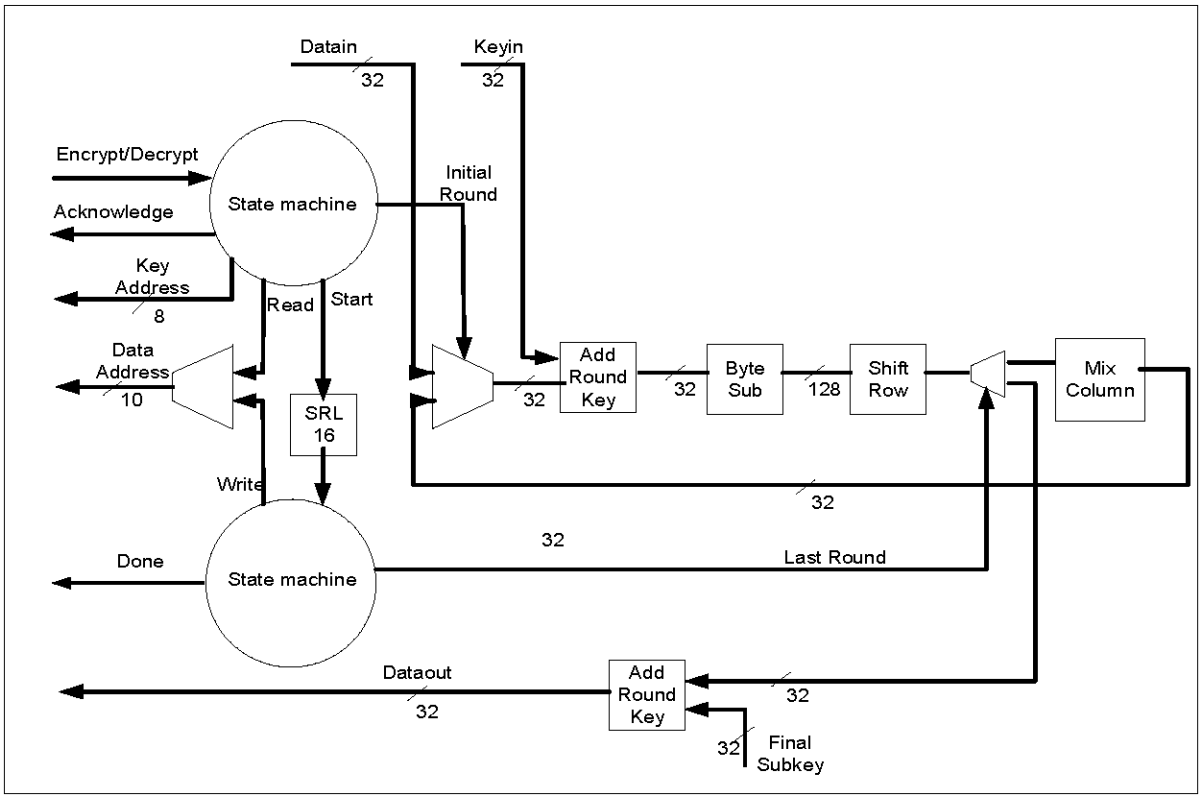

Figure 4: AES Encryption Process

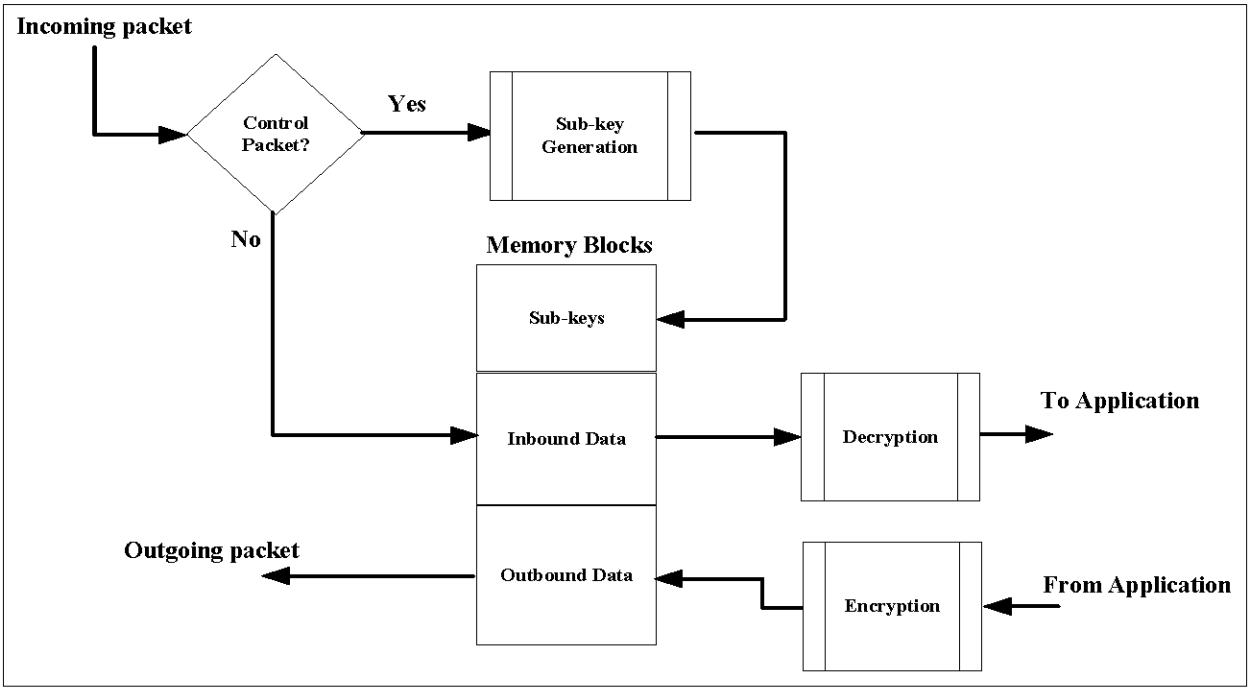

Figure 5: Software Implementation of AES in FPGA