FPGA Implementation of AES Encryption and Decryption

*Sounak Samanta  
B.E. III Yr, Electronics & Communication Engg,  
Sardar Vallabhbhai National Institute of Technology, Surat.*

**Abstract**:

This paper presents a high speed, fully pipelined FPGA implementation of AES Encryption and Decryption (acronym for Advance Encryption Standard, also known as Rijndael Algorithm) which has been selected as New Algorithm by the National Institutes of Standards and Technology (NIST) as US FIPS PUB 197 in November 2001 after a 5-year standardization process. The AES encryption & decryption algorithm is implemented on the FPGA. This has to have an interface with the PC. The C source for the encryption and decryption is already provided. The algorithm is implemented to work in software and this is our baseline implementation. The application works in the following manner. The file is to be encrypted in software and transferred to the machine containing the FPGA. The file will be decrypted in hardware in that machine. Also the file is to be encrypted in hardware and decrypted in software. Our design mainly concentrates on the speed up along with silicon area optimization.

Our work-steps includes Paper Designing, Writing VHDL Code (Very high speed integrated circuit Hardware Descriptive Language), Simulating the code on "ModelSim SE PLUS 5.8b", Synthesizing & Implementing (i.e. Translate, Map & Place and Route) the code on "Xilinx - Project Navigator, ISE 7i " with Device XC2V6000 of Vertex II Family & XST Synthesis Tool and finally implemented on ALPHA DATA board with XILINX VERTEX II PMC - 2V6000 FPGA series of ADM-XRC-II. The ADM-XRC-II is a high performance reconfigurable PMC (PCI Mezzanine Card) based on the Xilinx Virtex-II range of Platform FPGAs. This implementation uses: Slices -1051 out of 33792 ( 3% ), Slice Flip Flops - 1399 out of 67584 ( 2% ), 4 input LUTs -1450 out of 67584 ( 2% ), bonded IOBs - 390 out of 684 (57%) ,GCLKs -1 out of 16 ( 6% ), Minimum period - 13.038ns (Maximum Frequency: 76.699MHz). We have interface Host PC with ZBT (Zero Bus Turnaround) RAM on AlphaData Hardware plane through "C" programming language and ZBT RAM with our own AES Module through User Module (which is already provided).

**I. INTRODUCTION**

Software implementation of AES algorithm is slower process (though easy). So the focal approach of our design on hardware platform is to attain speed up (i.e. high throughput No. of block processed per second) at the same time, silicon area optimization.

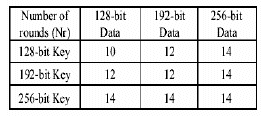
This paper is organized as follows. Section II provides brief overview of AES Algorithm. Section III focuses hardware architecture of our design. Section IV launches salient features. Section V discusses hardwaresoftware co-design. Section VI is endowed with experimental results on FPGA platform and finally section VII winds up with future work and conclusion.

**II. BRIEF OVERVIEW OF AES ALGORITHM**

In cryptography, the Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as an encryption standard by the US government. The cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted to the AES selection process under the name "Rijndael", a portmanteau comprised of the names of the inventors.

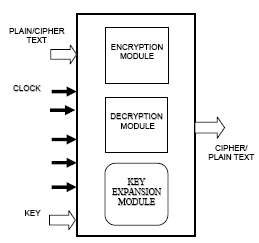
AES has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits, whereas Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits. AES operates on a 4×4 array of bytes, termed the state. For encryption, each round of AES (except the last round) consists of four stages. a) SubBytes - a non-linear substitution step where each byte is replaced with another according to a lookup table (known as S Box). b) ShiftRows - a transposition step where each row of the state is shifted cyclically a certain number of steps. c) MixColumns - a mixing operation which operates on the columns of the state, combining the four bytes in each column using a linear transformation. d) AddRoundKey - each byte of the state is combined with the round key; each round key is derived from the cipher key using a key schedule.

AES algorithm comprises of various rounds depending on the key size and blck size,(Fig No.1).Out of all the rounds the Pre- round comprises only AddRoundKey whereas the final round omits the MixColumns stage.

*  
FIG NO. 1*

**III. HARDWARE ARCHITECTURE**

**III.A. HARDWARE I/O SPECIFICATIONS**

*  
FIG NO. 2*

The above figure (Fig No.2) shows the hardware I/O specification for the AES Encrypter and Decrypter. It requires a PLAIN / CIPHER TEXT which is of 128 bits length. Also KEY of length 128 bits (or 192 bits or 256 bits) is needed to be given. The control signals are START ENCRYPTION, START DECRYPTION, START KEY GENERATION, ENCRYPTION / DECRYPTION. These signals are used to control the proper sequencing of the modules. This controlling is done through a controller (not shown in the figure) which is interfaced with the Encrypter, Decrypter and the Key Expander.

The input data is processed using the key and the resultant data is available at the output terminal named CIPHER / PLAIN TEXT.

The Key Expansion Module is common for both the Encrypter and Decrypter module.

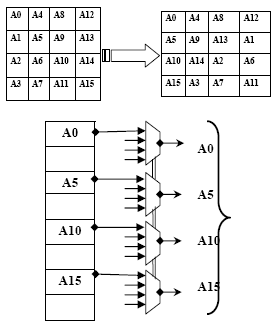
**III. B.TOTAL OVERVIEW OF DESIGN**

In standard AES algorithm, there are four steps like SubByte, ShiftRow, MixColumn and Add Round Key in normal rounds. Our design highlights some following modifications:

1. Exclusion of Shift Row
2. Pipelining for high Throughput
3. Optimizing the design to keep handy balance between Throughput and Silicon Area

**i. Exclusion of Shift Row:**

Exclusion of Shift Row is performed through calling required shifted element from the data matrix,(instead of calling element one by one sequentially orderly from the data matrix); Thus merging of the two steps SUB-BYTE and SHIFT ROW reduces one step.

*  
FIG NO. 3*

The Fig No. 3 shows how exclusion of Shift Row is performed. The 16 elements are stored sequentially after each round in a register file. Using Mux selection required shifted data elements can be called (instead of calling sequentially) from the regier file and put into the S-Box.

**ii. Pipelining for high Throughput**

One of the foremost requirements of AES ENCRYPTION DECRYPTION is high data throughput. For higher throughput, pipeline architecture is the easiest solution. One focal constraint is that pipelining is only possible within each round (Not for whole rounds required for encryption/decryption). Next round can start only when previous round is totally completed as input data of the next round solely depend on the output of the previous rounds. So our design mainly concentrates on Pipelined Architecture Implementation of each round.

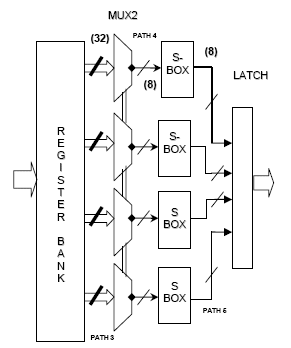
**iii. Optimizing the design to keep handy balance between Throughput and Silicon Area**

Various types of hardware architectures for AES algorithm are possible. The best architecture is one which is having the best trade off between Clock Speed (data throughput) and Silicon Area. Firstly, our design had been tried to complete within 220 cycles. In this design, it is included only one S Box (Look Up Table). So at least 16 cycles are required to process 16 elements through S Box (assuming in each cycle one element is processed through S-Box Transformation). Each AES round is having 4 steps. Using pipelined structures within each round, total cycles required (for 10 round of AES Algorithm for 128 bit data & 128 bit key) is approx 220 cycles. Here silicon area is saved but losing high data throughput. In the extreme case, whole 10 rounds can be completed within 44 cycles using 16 SBox (LUT). In this case, though high throughputs are achieved, silicon area may be too much wasted.

So our design mainly keeps the compromise between data throughput and silicon area in FPGA by introducing 4 SBox (LUT). Here whole encryption / decryption round completes within approx 90 cycles.

**III.C. PIPELINED ARCHITECTURE FOR ENCRYPTER**

The overall pipelined architecture for AES Encryptor looks as shown below. It includes 1. Register Bank, 2. Mux, 3. S-Box, 4. Pipelined Registers(Latch), 5. Mix-Column Module, 6.Add Round Key Module along with the Key Expander, 7. FSM Controller for Encryptor.

*  
FIG NO. 4(a)*

The user is asked to enter the plain text which has to be encrypted and also the key by which he will be able to decrypt this encrypted text back to plain text.

Once the user specifies the key and the input data, a key expander module starts expanding the inputted original key so that it is sufficient for all the rounds of encryption procedure. This expansion of the key is discussed in detailed on Key Expander Details pages.

After the key expansion is over, now encryption process begins. In encryption process , first of all , a pre round has to be performed which XORs the original 128 bits of input data with the original 128 bits of key and the intermediate result is stored in the register bank. Once the pre round is over then the remaining rounds of operation are performed. These rounds processed in the following manner:

*FIG NO. 4(b) [Continuation of Fig No. 4(a)]*

First, the data from the intermediate register bank is read and applied to four 4x1 (32 bits) Mux from where the selected data is feed to the input of S-box. This will transform the data to their corresponding transformed data and pass it to Mix Column stage for further processing. The Mix column stage gets these 32 bits of data and according to the algorithm multiplies the data with a standard matrix to produce a 32 bit output. Now this output is applied to the Add Round Key which also has the 32 bits input from the key memory. These two 32 bits inputs are XORed in this module and are get stored in the intermediate Register Bank again for the next round.

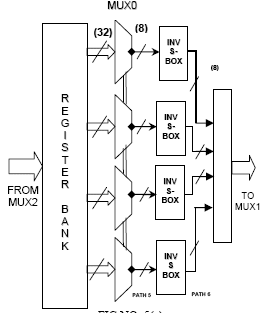
In this way, due to pipeline structure, all the four column of the new intermediate matrix are obtained one after the other keeping all the modules of the design busy all the time. In the last round, Mix Column stage is skipped and the result from the Add Round Key is gets stored in the output cipher text Memory.

The proper selection of the module and data path for a particular round is done by the FSM controller. This Controller also controls the Key Scheduling module so that valid keys are called for the particular round.

This whole procedure of encryption is done for 128 bits plain text and 128 bits key in 88 cycles excluding the 62 cycles for the key expansion module. But the key expansion overhead does not cause degrading of the performance because for a input of larger size, the same expanded keys are again and again.

**III.D. PIPELINED ARCHITECTURE FOR DECRYPTER**

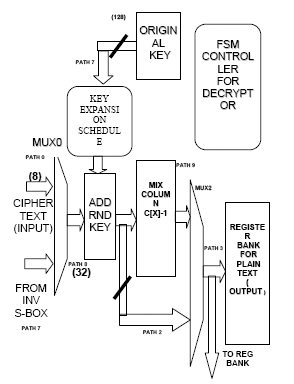
The overall pipelined architeture for AES Decrypter looks as shown below. It includes 1. Register Bank, 2.Mux, 3.Inverse SBox, 4.Pipelined Registers(Latch), 5.Inverse Mix Column Module, 6.Add Round Key Module along with the Key Expander, 7. FSm Controller for Decypter.

*  
FIG NO. 5(a)*

The user is asked to enter the cipher text which has to be decrypted and also the key by which he has encrypted this input data.

Once the user specifies the key and the input data, a key expander module starts expanding the inputted original key so that it is sufficient for all the rounds of encryption procedure. This expansion of the key is discussed in detailed on Key Expander Details pages.

After the key expansion is over, now decryption process begins. In decryption process , first of all , a pre round has to be performed which XORs the original 128 bits of input data with the original 128 bits of key and the intermediate result is stored in the register bank. Once the pre round is over then the remaining rounds of operation are performed. These rounds processed in the following manner

*  
FIG NO. 5(b) [Continuation of Fig No. 5(a)]*

First, the data from the intermediate register bank is read and applied to four 4x1 (32 bits) Mux from where the selected data is feed to the input of Inverse S-box. This will transform the data to their corresponding transformed equivalent and pass it to Add Round stage for further processing. The Add Round Key which also has the 32 bits input from the key memory. These two 32 bits inputs are XORed in this module and are passed to the Inverse Mix Column stage. The Inverse Mix column stage gets these 32 bits of data and according to the algorithm multiplies the data with a standard matrix to produce a 32 bits output which get stored in the intermediate Register Bank again for the next round.

In this way, due to pipeline structure, all the four column of the new intermediate matrix are obtained one after the other keeping all the modules of the design busy all the time. In the last round, Inverse Mix Column stage is skipped and the result from the Add Round Key is gets stored in the output plain text Memory.

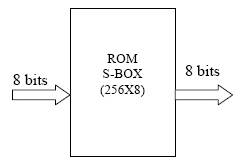
The proper selection of the module and data path for a particular round is done by the FSM controller. This Controller also controls the Key Scheduling module so that valid keys are called for the particular round.

This whole procedure of decryption is done for 128 bits cipher text and 128 bits key in 88 cycles excluding the 62 cycles for the key expansion module. But the key expansion overhead does not cause degrading of the performance because for input of larger size, the same expanded keys are again and again

**III.E. ARCHITECTURE FOR S-BOX (INV S-BOX) AND MIX-COLUMN (INV MIX COLUMN)**

**III.E.1. S-BOX:**

S-Box(INV S-Box) used is derived from the inverse funcion over GF(28), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. But in our design to avoid complexity, SBox( INV S-Box) is simply a Look Up Table (LUT) i.e ROM with capacity 256\*8 bits. The contents of Sbox (INV S-Box) is easily available in website (URL:: [**http://en.wikipedia.org/wiki/**](https://www.design-reuse.com/exit/?urlid=60) ).

*  
FIG NO. 6*

**E.2.MIX-COLUMN:**

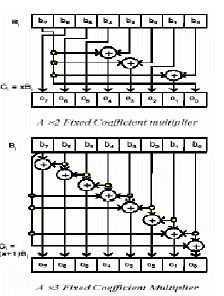
For Mix-Column architecture implemntation, it is preferable to do perform multiplication in Galois Field of Mahematical Computation. According to that, pipelining can also be inroduced. Due to pipelining, one column of the new state matrix can be achieved in one clock cycle only. So the whole of the new state matrix can be achieved in 5 cycles.

Standard Matrix Polynomial for Mix- Column during Encryption is:

|  |  |  |  |
| --- | --- | --- | --- |
| 02 | 03 | 01 | 01 |
| 01 | 02 | 03 | 01 |
| 01 | 01 | 02 | 03 |
| 03 | 01 | 01 | 02 |

Standard Matrix Polynomial for INVMix- Column during Decryption is:

|  |  |  |  |
| --- | --- | --- | --- |
| 0E | 0B | OD | 09 |
| 09 | 0E | 0B | 0D |
| 0D | 09 | 0E | 0B |
| 0B | 0D | 09 | 0E |

  
*FIG. NO. 7  
SOURCE:*[***http://csrc.nist.gov/encryption/aes/***](https://www.design-reuse.com/exit/?urlid=61)

The Fig. No. 7 shows hardware implementation of tha Galois Field Multiplication of {02}\* M(x) and {03}\*M(x) [explained shortly].

Multiplication in Galois Field during Encryption gives simplified form as follows:

{0D}\* M(x)= (m0 + m5 + m6) + (m1 + m5 + m7) x + (m0 + m2 + m6) x2 + (m0 + m1 + m3 + m5 + m6 + m7) x3 + (m1 + m2 + m4 + m5 + m7) x4+ (m2 + m3 + m5 + m6) x5 + (m3 + m4 + m6 + m7) x6 + (m4 + m5 + m7) x7.

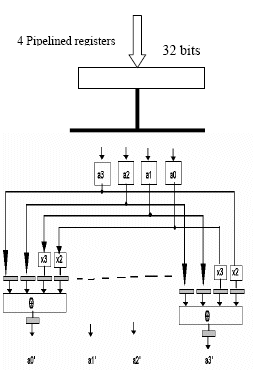
{0E}\* M(x)= (m5 + m6 + m7) + (m0 + m5) x + (m0 + m1 + m6) x2 + (m0 + m1 + m2 + m5 + m6) x3 + (m1 + m2 + m3 + m5) x4+ (m2 + m3 + m4 + m6) x5 + (m3 + m4 + m5 + m7) x6 + (m4 + m5 + m6) x7.

{0B}\* M(x)= (m0 + m5 + m7) + (m0 + m1 + m5 + m6 + m7) x + (m1 + m2 + m6 + m7) x2 + (m0 + m2 + m3 + m5) x3 + (m1 + m3 + m4 + m5 + m6 + m7) x4+ (m2 + m4 + m5 + m6 + m7) x5 + (m3 + m5 + m6 + m7) x6 + (m4 + m6 + m7) x7.

{09}\* M(x) = (m0 + m5) + (m1 + m5 + m6) x + (m2 + m6 + m7) x2 + (m0 + m3 + m5 + m7) x3 + (m1 + m4 + m5 + m6) x4+ (m2 + m5 + m6 + m7) x5 + (m3 + m6 + m7) x6 + (m4 + m7) x7.

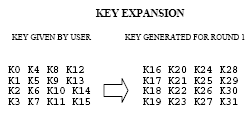
The above Galois Field Standard Derivation is taken from website. It is to be noted that + signs enclosed by the braces are denoting EX-OR operations and \* denoting multiplication in Galois Field.

Using this algorithm, overall Mix- Column step can also be pipelined for achieving high throughput. Due to pipelining, one column of the new state matrix can be achieved in one clock cycle only. So the whole of the new state matrix can be achieved in 5 cycles. The Top level hardware architecture for the Mix-Column is shown in Fig No.8

  
*FIG NO. 8 SOURCE: unknown*

**III.F. PIPELINED ARCHITECTURE FOR KEY EXPANSION SCHEDULE**

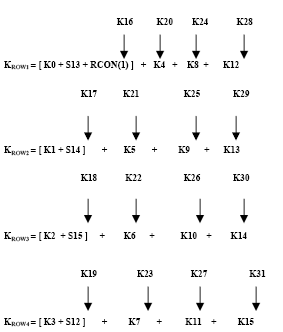
Apart from Encryption and Decryption Module, another main component is Key Expansion Schedule. The security factor of the AES Encryption / Decryption Standard mainly depends on this part. For better security, AES Algorithm says that in first round user key is XORed with the original Plain / Cipher Text. And next round onwards Expanded Key from Expanded Key Schedule is XORed with data. The expansion algorithm of the AES is fixed. To speed up the process of Key Generation, it is opt for pipeline architecture.



Likewise for 10 rounds (except pre rounds) total number of keys generated is 160 byte. In other words, for 128 bit data and 128 bit key with 10 rounds total number of key utilized in AES is 176 bytes.

If User Key is of 192 bit (or 256 bit) keeping data bit constant of 128bit, then number of rounds increases to 12 (or 14).

The KEY EXPANSION ALGORITHM as follows.

*  
FIG NO. 9*

Where:

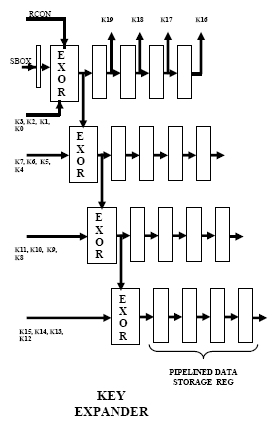
1. Sij : means S-box transformation of the corresponding key byte Kij

2. RCON=ROUND CONSTANT

RCON(1) = 01H,    RCON(2) = 02H,  
RCON(3) = 04H,    RCON(4) = 08H,  
RCON(5) =10H,     RCON(6) = 20H,  
RCON(7) = 40H,    RCON(8) = 80H,  
RCON(9) = 1BH,    RCON(10) = 36H  
RCON(11) =6CH,   RCON(12) = D8H,  
RCON(13) = ABH, RCON(14) =4DH,  
RCON(15) = 9AH,

**III.F.1. PIPELINED ARCHITECTURE FOR KEY EXPANSION MODULE**

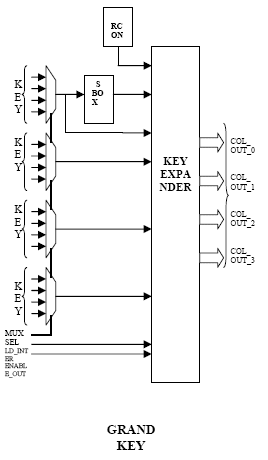
The figure No.10(a) shown presents a hardware architecture for Key Expansion Module which is one of the main components of Grand Key.

*  
FIG NO. 10(a)*

Key Expander comprises of EX-OR, Pipelined Data Registers. Grand Key is the top module of Key Expander for proper sequential access of key input to the Key Expander. Basically Grand Key comprises FSM controller to control the input keys and control signals required for Key Expander.

**III.F.2. PIPELINED ARCHITECTURE FOR GRAND KEY**

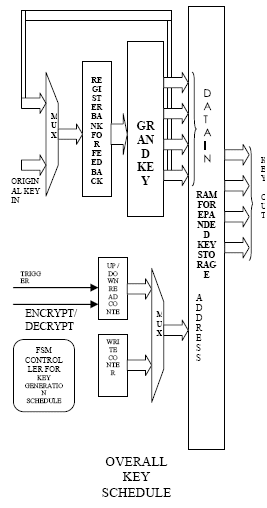
The figure No. 10(b) shown presents a hardware architecture for Grand Key, which is oe of the main component of Key Expansion Schedule.

*  
FIG NO. 10(b)*

Grand Key and Key Expander is basically a pipelined implemented architecture of the Key Expansion Algorithm shown. Key Expander uses approx 5 cycles to generate four column i.e. in 5 cycle expanded key is generated for one round. Total cycles required for completion of key expansion/ generation is approx 60 cycles.

**III.F.3. PIPELINED ARCHITECTURE FOR OVERALL KEY SCHEDULE**

The figure 10(c)shown presents a hardware architecture for overall Key schedule.

*  
FIG NO. 10(c)*

Overall Key Schedule is the top module of the Key Generation Module. Overall Key Schedule comprises of Key Expander (shown in figure) and Grand Key.

Overall Key Schedule is basically a storage of expanded keys. For storing the expanded key, RAM is used where for reading and writing, binary counter is used for addressing purpose. To re-use the previous expanded keys, one feedback register is used to store the previous round key. Now trigger pin and Encrypt/ Decrypt are of high importance. As trigger pin is connected to Up/Down Counter, whenever trigger is given from AES Module, new expanded key are come out from Overall Key Schedule which is utilized i.e. XORed with the data text. Encrypt / Decrypt option is to decide whether user wants to encrypt or decrypt. As during Encryption, expanded key should be called from the storage RAM in forward direction but during Decryption, expanded key should be called from the RAM storage in backward direction ( As Decryption is the reverse procedure of Encryption one ).

**IV. SALIENT FEATURES OF AES ENCRYPTION & DECRYPTION**

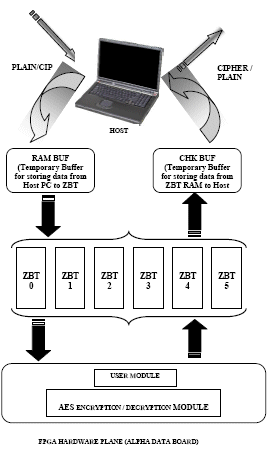
The salient features of the AES ENCRYPTION/DECRYPTION are summarized in the following manner:

* **HIGH THROUGHPUT :**
  + For fully Pipelined implementation, area requirements increase with larger Key Size but throughput ( No. of blocks processed per second ) is unaffected.
* **PARAMETER FLEXIBILTY:**
  + Any combination of Key sizes and Block sizes those are multiples of 32 bits can be accommodated. As a result, number of rounds can be modified.
* **IMPLENTATION FLEXIBILTY:**
  + Decryption can be implemented in same structure as Encryption. (Though with different components)
* **NO KNOWN SECURITY ATTACK:**
  + Although it has received criticism due to its simple mathematical structure.

**V. HARDWARE AND SOFTWARE CO-DESIGN**

The coding of the DSP architecture is done in VHDL language. AES architecture is implemented on the FPGA of device family Virtex-2.VHDL Code is synthesized using ISE. It is simulated using Model Sim.

* Device Family:Virtex2
* Tools used:
  + Xilinx ISE 7.1i
  + ModelSim SE PLUS 5.8b
* Device: xc2v3000
* Board:` ADM-XRC
* SSRAM: 4 banks 256k\*36bits (ZBT)

*  
FIG NO.11 FPGA HARDWARE PLANE (ALPHA DATA BOARD)*

The above figure No.11 shows an overall view of what a interface looks like. The host PC is the user s computer which will be connected to this AlphaData Board (FPGA Board) for the Encryption and Decryption purpose.

In interfacing part, there are two parts:

**1. Host Access** (interfacing between Host PC and ZBT RAM in ALPHADATA Board )

**2. User Module Access** (interfacing between ZBTRAM and User Module, AES ENCRYPTION / DECRYPTION MODULE).

In Host Access, everything is done through C program. Whenever User wants to do Encryption/Decryption, File of Plain/Cipher text is read from Host PC to a temporary buffer RamBuf. From RamBuf it is copied to required ZBT. After Copying, C program provides User Module Access.

In User Module Access, AES ENCRYPTION / DECRYPTION Module written in VHDL Code accesses its required data from required ZBT. After reading all required input data from ZBT, it starts its computation and writes its output data to required ZBT. After doing so, again Host Access is provided. This can be possible through interrupt facility or any default sleep in C program. To avoid complexity, it is preferred to opt for the later one.

In Host Access, again output data from required ZBT is transferred to another Temporary Buffer ChkBuf. From ChkBuf, output data is written to any file on the Host PC.

* Platform Used :
  + Windows 2000 Proffesional,Fedora 2
* Compiler:
  + GCC compiler

**VI. RESULTS & ACHIEVEMENTS**

The simulation and synthesis results are presented below. It is acknowledged that synthesis results are with some (approx. 41) warnings which are allowable.

= = = = = = = = = = = = = = = = = = = = = = = =  
**SIMULATION RESULTS: (Using MicroSIM)**= = = = = = = = = = = = = = = = = = = = = = = =  
===================================

* Key Expansion : 60 cycles (approx)
* Encryption : 90 cycles (approx)
* Decryption : 80 cycles (approx)

===================================

= = = = = = = = = = = = = = = = = = = = = = = =  
SYNTHESIS RESULTS: (Using Xilinx ISE 7.1i)  
= = = = = = = = = = = = = = = = = = = = = = = =

**o Device utilization summary:**  
=======================

Selected Device: 2v6000bf957-4

* Number of Slices:

1051 out of 33792 ( 03% )

* Number of Slice Flip Flops:

1399 out of 67584 (02% )

* Number of 4 input LUTs:

1450 out of 67584 (02% )

* Number of bonded IOBs:

390 out of 684 (57%)

* Number of BRAMs:

11 out of 144 (07% )

* Number of GCLKs:

1 out of 16 (06%)

**o Timing Summary:**  
===============

Speed Grade: -4

* Minimum period: 13.038ns (Maximum Frequency: 76.699MHz)
* Minimum input arrival time before clock: 6.933ns
* Maximum output required time after clock: 6.611ns
* Maximum combinational path delay: 7.790ns

The results shows that the hardware implementation of AES Algorithm is no doubt faster than the software implemented AES algorithm.

**VII. FUTURE WORK AND CONCLUSION**

For future work,

* This AES algorithm can be parameterised by selection of cipher key bits (128,192 or 256).
* For higher throughput, 16 S-Box can be used completing whole processes around 44-50 cycles(at the same time,compromising the silicon area)
* Graphical Use Interface(GUI) can also be made which may be interacive wih the user.

**VIII. ACKNOWLEDGEMENTS:**

I hereby take the opportunity to express deep sense of gratitude to Dr. Kolin Paul, Computer Science and Engineering Department, IIT Delhi, for the motivation, guidance and supervision. He has helped us in all respects for finishing this project and hence gave a good working environment for doing the project.

I like to thank Prof. M.Balakrishnan, Computer Science and Engineering Department, IIT Delhi, who has directed us in time to time.

I also heartily thank Prof (Mrs.) N.Y.Desai, Head, Electronics Engineering Dept, SVNIT, Surat for recommending me to do the project work at CSE, IIT, Delhi.

I unreservedly thank Mr. Anil Aggarwal, my group partner for helping me to debug and complete the project in all respect. I also like to thank Mr. Vikas Upmanue, Mr. Chandrasekhar, Mr. Nagarjuna and Mr. A. Sahu for providing resources and guidance.

Finally, I wholeheartedly thank Department of Computer Science and Engineering, IIT Delhi, and XILINX and CMC Ltd. for organizing and sponsoring the workshop, which not only gave me a good and challenging work-experience on Digital System Design but also motivated me to take it as a research area in our coming days.