HIGH DATA RATE 8-BIT CRYPTO PROCESSOR

Sheikh M Farhan, Habibullah Jamal, Mohsin Rahmatullah

University of Engineering and Technology,

Taxila, Pakistan

smfarhan@carepvtltd.com, (+92-51-2874794), 19-Ataturk Avenue, G-5/1, Islamabad Pakistan

mohsin@carepvtltd.com, (+92-51-2874794), 19-Ataturk Avenue, G-5/1, Islamabad Pakistan

drhjamal@uettaxila.edu.pk, (+92-596-9314224), UET Taxila, Pakistan

ABSTRACT

This paper describes a high data rate 8-bit Crypto Processor based on Advanced Encryption Standard (Rijndael algorithm). Though the algorithm requires 32-bit wide data path but our novel mix-column architecture makes the algorithm works in a true byte systolic fashion. Initial stages are merged to remove dependency of completion of these stages on mix-column stage. It has resulted in the optimization of data path utilization and bus width thus minimizing control logic, area and power. Most of the commercially available AES crypto processors use different hardware modules to handle Key Expansion and Data Encryption. The paper also presents a novel approach to handle both the key expansion and data encryption phases by re-using the same hardware architecture. The proposed design saves many hardware resources when mapped on FPGA and allows operation at high clock frequencies and data rates.

KEY WORDS

Advanced Encryption Standard (AES), Security, Rijndael, Crypto Processor.

HIGH DATA RATE 8-BIT CRYPTO PROCESSOR

1 INTRODUCTION

The need for a new encryption standard became unquestionable after DES was shown to be vulnerable to brute force attack. In 1997, The National Institute of Standards and Technology (NIST) started an effort towards developing a new encryption standard, called the Advanced Encryption Standard (AES). The development of the new standard was initiated in the form of a contest. In November 2001, NIST accepted a new standard Rijndael [1,2] as Advanced Encryption Standard. AES specifies a FIPS-approved cryptographic algorithm that is used to safely protect electronic data such as secure wireless communications, protected network routers, electronic financial transactions, secure video surveillance system and encrypted data storage.

Many applications, due to their complexity, are at times constrained by the area offered by the FPGAs when mapped to these devices. Some applications do not require high data rate and their speed can be compromised on area requirements. This paper presents such an architecture in which AES algorithm, which is a 32-bit algorithm, is mapped on a byte-systolic 8-bit architecture thus reducing area requirement but still offering moderately high data rates.

The paper is organized as follows: Section 2 describes the AES algorithm briefly. Section 3 presents the proposed Crypto Processor architecture. Results are given in Section 4. Finally, concluding remarks are made in Section 5. Section 6 lists the references.

2 AES ALGORITHM

AES is an iterative symmetric block cipher and works on a fixed block size of 16 bytes (128 bits) as explained in [1,2,5]. The standard has the provision to work with variable key sizes of 128, 192 and 256 bits. AES as well as most encryption algorithms is involutional. This means that almost the same steps are performed to complete both encryption and decryption in reverse order.

Since AES is an iterative block cipher, the same operations are performed many times on a fixed number of bytes. These operations can easily be broken down to the following four functions:

- Add Round Key (ARK)
- Byte Substitution (BS)
- Shift Rows (SR)
- Mix Columns (MC)

Figure 1 shows the basic AES algorithm flow.

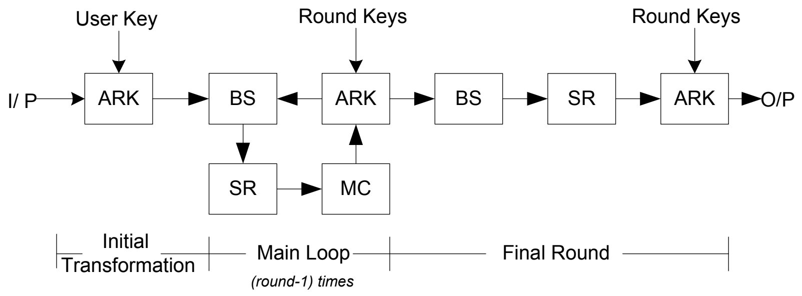

Figure 1. AES Algorithm Flow

An iteration of the above steps is called a 'round'. The number of rounds or iterations of the algorithm depends on the key size. *Table* 1 shows the number of rounds required for different key sizes

Table 1. Number of rounds for different key sizes

Key Size (Bytes)	Block Size (Bytes)	Rounds
16	16	10
24	16	12
32	16	14

The algorithm handles 128-bit input block as a group of 16 bytes organized in a 4×4 matrix called State matrix. The algorithm starts with an initial transformation, followed by a main loop where rounds are executed. Each round of the main loop is composed of a sequence of four transformations (ARK, BS, SR, MC) where as the initial transformation simply contains the ARK transformation. For each round of main loop, a round key is used derived from the original key through a process called Key Expansion. In the last round, three transformations BS, SR and ARK are executed [1, 2].

The next section briefly describes the four AES rounds transformations and cipher key expansion for generating round keys.

2.1 Byte Substitution

In this transformation, each input byte of the state matrix is independently replaced by another byte from a look-up table called Sbox. Sbox is a 256-entry table composed of two transformations: First each input byte is replaced with its multiplicative inverse in GF(2^8) [4] with the element {00} being mapped onto itself; followed by an affine transformation over GF(2^8) [3, 4].

2.2 Shift Rows

A cyclic shift operation is carried out where each row is rotated cyclically to the left using 0, 1, 2 and 3-byte offset for encryption.

2.3 Mix Columns

The Mix Columns transformation operates on the state column-by-column, treating each column as a four-term polynomial. In this transformation each column of the state matrix is multiplied by a constant fixed matrix where each member is represented in GF(2^8) as shown in *Figure* 2 [5]. The members of this matrix are represented in 8-bit wide hex notation.

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

Figure 2. Constant fixed matrix for state matrix multiplication

Two column vectors above represent the *i*-th column of state matrix and *i*-th column of transformed mix column state matrix, for i = 0, 1, 2, 3, respectively.

2.4 Add Round Key

The Add Round Key transformation is self-inverting. It maps a 128-bit input state to a 128-bit output state by XORing the input state with 128-bit round key derived from the cipher key.

2.5 Key Expansion

The cipher key is expanded to round keys using the pseudo code shown in *Figure* 3. Parameter *Nk* is the number of 32 bit words comprising the cipher key. AES Algorithm supports *Nk*=4,6 or 8. Nb is the number of columns (32-bit words) comprising the state. For this standard, Nb = 4. Nr is the number of rounds, which is a function of Nk and Nb (which is fixed). For this standard, Nr=10, 12 or 14. Function *SubWord( )* applies the Sbox to a four byte input word [6]. Function RotWord( ) performs a cyclic permutation on the input word [d0, d1, d2, d3] resulting in the output word [d1, d2, d3, d0]. The Round Constant Rcon[*i*], contains the values given by [x^{i-1}, {00}, {00}, {00}].

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Figure 3. Pseudo code for the AES Algorithm Key Expansion

3 CRYPTO PROCESSOR ARCHITECTURE

The architecture proposed in this article implements AES algorithm for 128-bit cipher block and 256-bit cipher key (*Nk*=8). The architecture is novel in its design as it eliminates the Shift Rows (SR) transformation from all round sequences. Another powerful feature of the proposed design, that makes it different from other architectures is its ability to use the same encryption architecture for key expansion thus completely eliminating the use of a separate hardware unit for key expansion as found in currently available designs. Memories and data path are kept 8-bit wide to keep the design simple and efficient. *Figure* 4 shows the design flow of the proposed architecture without Shift Rows transformation.

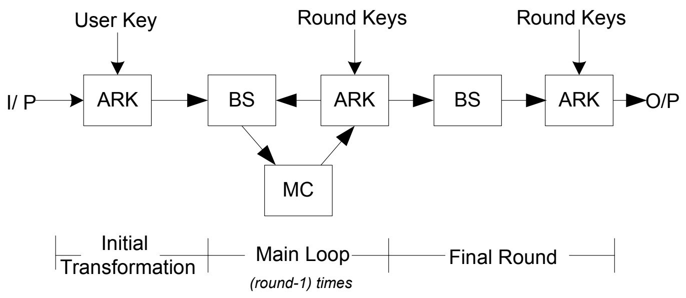

Figure 4. Proposed Crypto Processor design flow

Our Crypto Processor architecture comprises of the following major units: Memory, FSM Controller, Data-Path. The next section describes each hardware section in detail.

3.1 Crypto Processor - Memory Management

No external memory is required for the proposed design. It makes use of the dual port Block SelectRAM^{TM} (BRAM) available inside FPGAs that gives better access time and performance. For input cipher data, one BRAM of 8x32 bits is required which is divided into two pages. The incoming data bits are stored in the first 16-byte location of the memory. Once data is written into the memory, Crypto Processor uses both pages of the data memory in a ping-pong fashion to process the data iteratively.

Cipher and round keys are stored in an 8x240 bits memory. Once the cipher key is written into the memory, Crypto Processor takes over the key memory for key expansion and stores the generated round keys into the same key memory. *Figure* 5 shows the association between memory blocks and Crypto Processor inside FPGA.

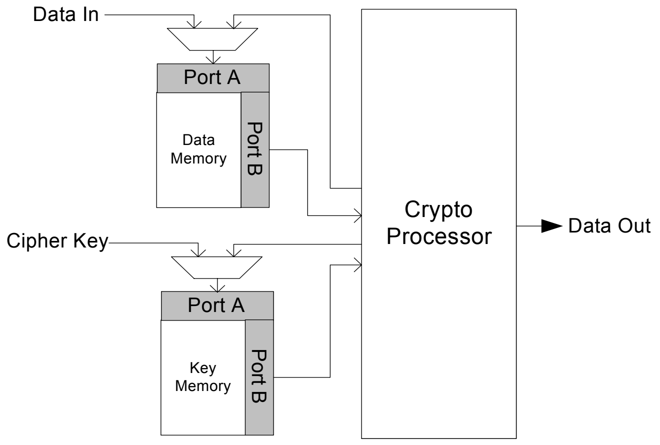

Figure 5. Data flow between memories and Crypto Processor

Elimination of shift rows transformation is achieved through a simple technique of reading the state from the block memory into the Crypto Processor in a manner such that the Crypto Processor receives the input state in an already row shifted format. The same can be achieved when the state is read from the memory in the following sequence of addresses.

```
j=11;
for(i=0; i<16; i=i+1)
{
  j=j+5;
   byte[j];
}
```

Where *j* is a four bit field which wraps around on overflow i.e. $(1111)_b + (0101)_b = (0100\ )_b$. This can be easily implemented in hardware by making a 4-bit counter and adding 5 on every clock pulse. *Figure* 6 shows the state matrix *s* and *s'* before and after shift rows transformation respectively whereas *Figure* 7 shows achieving the same through reading the bytes with an address offset of 5. The numbers in the state box in *Figure* 7 represent the addresses from which bytes are read into the state matrix.

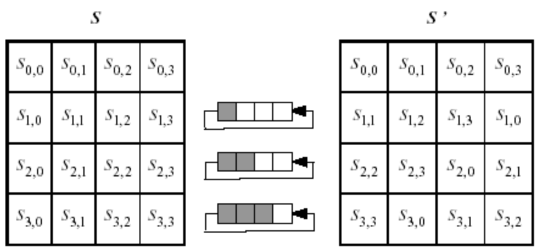

Figure 6. Cyclic Shift of last three rows in state matrix

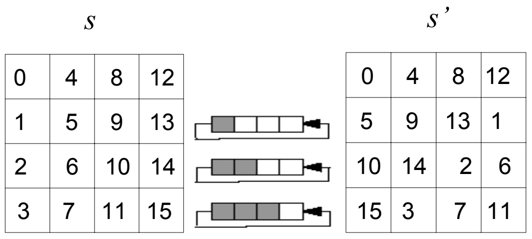

Figure 7. State matrix holding bytes at offset addresses

In the first 16 cycles of encryption, memory is read from page 0 of 8x32 bits wide data memory into the Crypto processor and is written back at corresponding addresses of page 1 after getting processed. In the next 16 cycles, the partially processed data is read into Crypto processor from page 1 and written back to page 0 at the corresponding addresses after processing. This process continues until all iterations are exhausted Keys are also read in the same manner from the key memory, which are then used in the add round key transformation. The key memory is divided into 15 pages, each page holding 16 bytes of round keys. For each iteration, a new page of round keys is read from the memory. To carry out key expansion, the same data memory is used to store round constants (Rcon) and key memory for storing cipher and round keys. Once the key expansion is done, the data memory is available for storing cipher data.

3.2 Crypto Processor - Data Path

Figure 8 and 9 show the data-path section of the proposed Crypto Processor. The whole data path is kept 8-bits wide thus making it cost effective and efficient resulting in a low power design. *Figure* 8 implements the 'add round key' and 'byte substitution' transformation whereas *Figure* 9 implements 'mix columns' transformation. 'Add round key' and 'byte substitution' transformations are pretty straightforward in hardware as shown in *Figure* 8. The mix column data path implements the matrix multiplication in a byte systolic manner. Mix Column stage is a linear transformation based on multiplication in GF(2^8) with a modular polynomial x^4+1. It is easy to be implemented with XOR operations. As shown in *Figure* 9, four partial results are computed by multiplying the resulting data after byte substitution with the first column of the matrix, refer to *Figure* 2. The second, third and fourth column show that the same multiplier coefficients are used as in the first column and can be re-used. This is achieved by circularly shifting the partial results by one in every clock cycle after adding the new product into the corresponding partial sums as shown in *Figure* 9. Multiplexers are inserted at different data feeding points. These multiplexers allow the same hardware to be used for both key expansion and encryption. These multiplexers are controlled by a state machine controller, which can be considered as the brain of the Crypto Processor. The next sections describe how the same data-path is re-used for both key expansion and encryption.

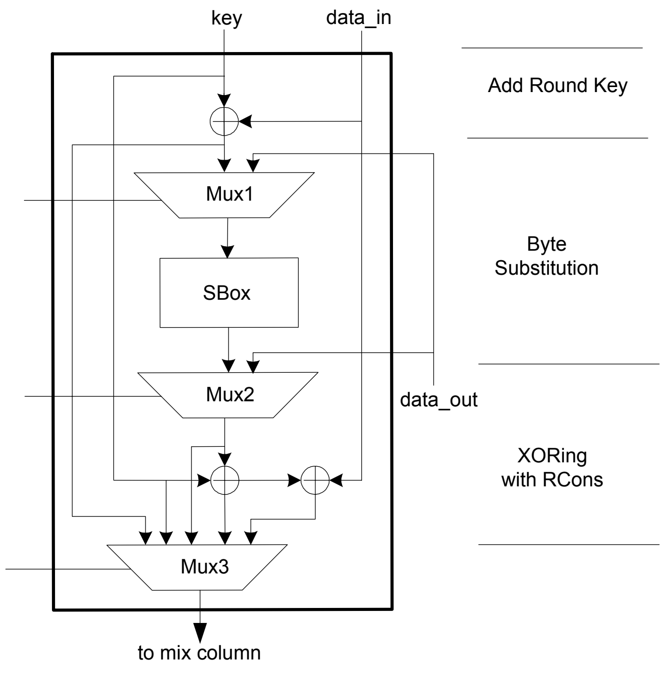

Figure 8. *Add round key and byte substitution*

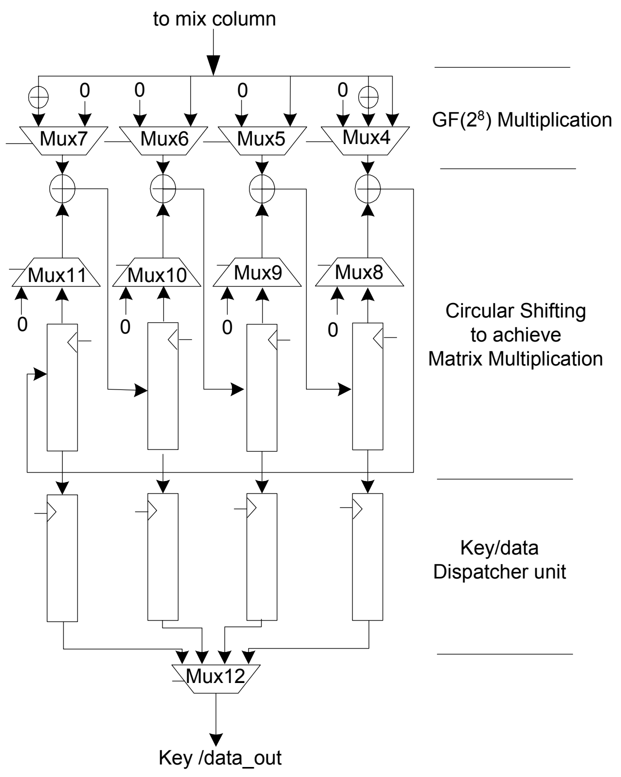

Figure 9. Mix Column

3.2.1 Data Path - Key Expansion

Refer to pseudo code in *Figure 3*. The initial 256-bits (32 bytes) cipher key is written into an 8-bits wide key memory. A counter keeps a record of iterations, which is incremented by one every time after writing four bytes of cipher key into the memory. The counter output will be 7 (starting from 0) after writing 32 bytes of cipher key into the key memory. Key expansion starts from the 8th iteration onwards. *Table* 2 shows different steps performed to generate round keys at different iterations.

*Table 2. Step*s performed at different iterations

Iterations *(i)*		
8, 16, 24, 32, 40, 48, 56	12, 20, 28, 36, 44, 52	Remaining
Rotate word. Byte Subs. XOR with Rcon.	Byte Subs	↓
XOR (key[*i*], key[*i*-8])		

At the start of the 8th iteration, the last four bytes of cipher key are read into the Crypto Processor while bypassing the ‘add round key’ and ‘byte substitution’ module. These last four bytes are directly fed into the mix column module, which rotates these bytes giving them a left circular shift of one byte thus implementing the RotWord( ) transformation. This rotated word is the output from the mix column stage, which is fed back into Mux1 for byte wise substitution thus implementing the ByteSub( ) function. In the same cycle when the byte substitution takes place, key[*i*- *Nk*] is also fetched and the two words are XORed together byte wise in each cycle. The resulting word is then XORed with Rcon[*i*], which is made available from data memory. This will result in four bytes of generated round keys, which are written back into the key memory. The same process continues for other iterations with some steps skipped as shown in *Table* 2. The 60th iteration will give 240 bytes of round keys written into the key memory. At the end of key expansion, the Crypto Processor is ready for encryption.

3.2.2 Data Path - Encryption

On receiving the encryption request, the cipher data and round keys are fetched from the memory byte by byte at every cycle in a manner mentioned in section 3.1. The two data bytes are XORed with each other for add round key transformation. Mux1 directs this data to the Sbox ROM. This data becomes the address of the Sbox ROM and gets substituted with the corresponding data bytes residing at those addresses. After Byte substitution, the date is directed to mix column stage for matrix multiplication as described in section 3.2. This completes one round and the output of the mix column is written back into the data memory. At the end of 15 rounds, the fully encrypted data is available in the data memory.

3.3 Crypto Processor - FSM Controller

The FSM controller is responsible for driving the Crypto Processor either for key expansion or for data encryption. The controller is also responsible for generating status/control signals for data transportation, key expansion, encryption and addresses for BRAMs for reading and writing the data from/into the memory. The controller directs the data through appropriate channels inside the engine to perform key expansion and data encryption. The controller has the intelligence to skip or perform transformations at different iterations through different multiplexers embedded in the design. For example in key expansion, in all those iterations not divisible by integer 8, all the transformations are bypassed using mux3. It is the controller that generates that address sequence for data and key memory to eliminate shift rows transformation. Initial transformation, main loop rounds and final round are also sequenced through the same controller.

4 RESULTS

The synthesizable AES Crypto Processor core was described in Verilog HDL using ModelSim 5.7G simulator and synthesized using Xilinx ISE6.1i. The target device selected was Xilinx XC2V1000 [12]. The synthesis results show that the synthesized device uses only 336 slices. This is far lesser than those devices that are designed for 32-bit wide data path. The device uses only two BRAMS each 8-bit wide and can be operated at a clock frequency of 110 MHz. The data rate achieved for this byte systolic architecture is in the range of 53 Mbits/s. However, the same design if mapped to an ASIC is guaranteed to give even better data rates for the 8-bit Crypto Processor. The FPGA device utilization for the Crypto Processor is shown in *Table 3* whereas a comparison of number of slices between different 32 bit AES architectures and our 8-bit Crypto Processor is shown in *Table 4*. *Figure* 10 shows a snapshot of the Crypto Processor testing and verification phase in a simulation environment.

Table 3. FPGA Device Utilization for Crypto Processor

XC2V1000 Device Utilization			
Resources	***Used***	***Available***	***%***
Number of Slices	336	5120	6
Number of Slice Flip Flops	287	10240	2
Number of 4 input LUTs	615	10240	6
Number of bonded IOBs	98	172	56

Table 4. Performance analysis measurements

Architecture	**Process**	**FPGA Device**	**CLB Slices**	**Frequency (MHz)**
1	Encr	XCV1000 BG560	5302/10992	14.1/31.8
2	Encr	XILINX (Not Specified)	5673	---
3	Decr	Altera EPF10K 250AG	2885	41.5
4	Encr/Decr	Altera APEX1K4001	845LE	--
5	Encr/Decr	Xilinx Virtex	2902	25.9
8-bit Crypto Processor	KeyExp /Encr	Xilinx XC2V1000	**336**	**110**

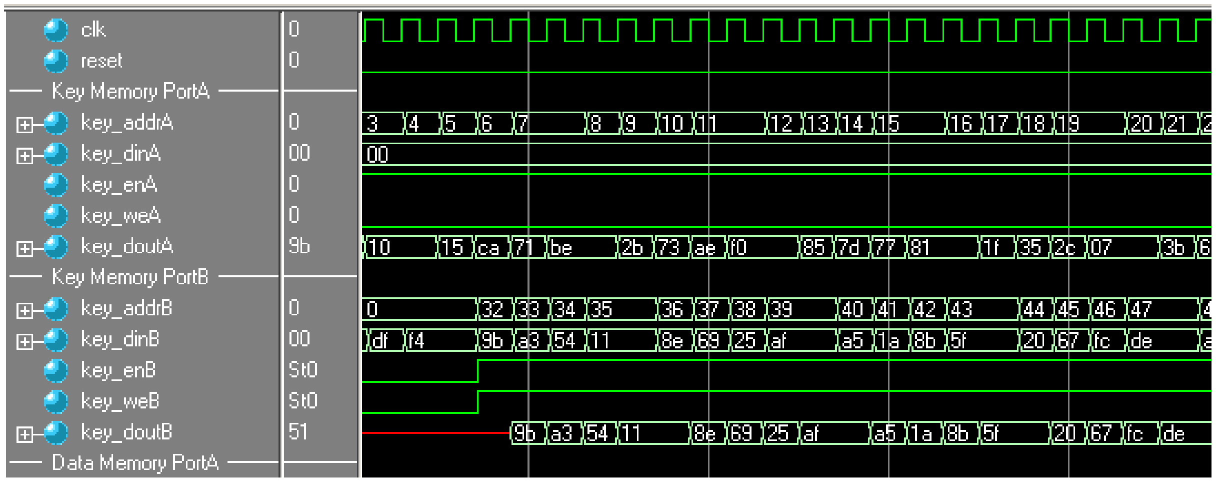

Figure 10. Crypto Processor simulation during key expansion

5 CONCLUSION

The paper presented a novel and unique design for an 8-bit Crypto Processor. The 32-bit AES algorithm is modified for 8-bit systolic implementation for low power and area efficient design. The proposed architecture is capable of reusing the same architecture for key expansion thus eliminating the use of a separate hardware module to generate round keys. A high level of resource sharing is done between key expansion and encryption. The technique used in the mix column transformation reduced dependency of completion of initial stages on the multiplication operations in mix column. The proposed design optimizes hardware reuse and makes the architecture to run in a true systolic fashion. The Crypto Processor is best suited for applications with moderate high data rates in the range of 30 ~70 Mbits/s.

Future work includes the realization of full encryptor/decryptor core mapped for an 8-bit architecture as well as possible design improvements.

6 REFERENCES

[1] J. Daemen, V.Rijmen *"The Rijndael Block Cipher" AES proposal, First Candidate conference (AESI)*, August 20-22, 1998.
[2] Joan Daemen, Vincent Rijmen " *The Design of Rijndael, AES-The Advanced Encryption Standard"* Springer-Verlag Berling Heidelberg New York 2002.

[3] Wade Trappe, Lawrence C. Washington *"Introduction to Cryptography with Coding Theory"*, Prentice Hall, Upper Saddle River, NJ 07458.

[4] *"Announcing the Advanced Encryption Standard (AES)"*, Federal Information Processing Standards Publication 197 November 26, 2001

[5] Nazar A. Saqib, Fransciso Rodrigues-Henriquez, Arturo Diaz-Perez. *"AES Algorithm Implementation- An efficient approach for Sequential and Pipeline Architectures",* Fourth Mexican International Conference on Computer Science (ENC'03)

[6] Anderson Cattelan Zigiotto, Roberto d'Amore, *"A Low-cost FPGA Implementation of the Advanced Encryption Standard Algortihm",* 15th symposium on Integrated Circuits and Systems Design (SBCC1'02)

[7] A.J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, *"An FPGA Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists*", Proc. Third Advanced Encryption Standard (AES) Candidate Conf., Apr. 2000.

[8] A. Dandalis, V.K. Prasanna, and J.D.P. Rolim, *"A Comparative Study of Performance of AES Final Candidates Using FPGAs*", Proc. Third Advanced Encryption Standard (AES) Candidate Conf., Apr. 2000. (This work has also been published in the Proc. CHES 2000, Aug. 2000).

[9] P. Mroczkowski, *"Implementation of the Block Cipher Rijndael Using Altera FPGA*", http://csrc.nist.gov/encryption/aes/round2/ pubcmnts.htm, 2001.

[10] V. Fischer and M. Drutarovsky, *"Two Methods of Rijndael Implementation in Reconfigurable Hardware",* Proc. CHES 2001, May 2001.

[11] K. Gaj and P. Chodowiec, *"Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware*", Proc. Third Advanced Encryption Standard (AES) Candidate Conf., Apr. 2000.

[12] Xilinx Virtex^{TM} II Platform FPGAs. URL: www.xilinx.com, October 14, 2003