FPGA Implementation AES for CCM Mode Encryption Using Xilinx Spartan-II

Khoa Vu, David Zier

***Abstract*— This paper discusses a possible FPGA implementation of the AES algorithm specifically for the use in CCM Mode Encryption. CCM Mode encryption is a proposed standard to be used and the security backbone behind the new IEEE Std. 802.11i. CCM currently spends most the computation power performing the AES algorithm. This paper investigate the possibility of creating an off-chip AES system for CCM so that the process can be speed up. The implementation was done on Xilinx Spartan IIE, running on 50MHz platform.**

***Index Terms*— AES, CCM, CCMP, Cryptography, 802.11i, Security, Wireless LAN.**

I. Introduction

AS 802.11i becoms a standard in the near future, it will replace the current 802.11 WEP (wireless equivalent protection) security scheme. Counter mode encryption CBC-MAC protocol (CCMP) will be the basis protocol to protect data transfer across the wireless medium for the 802.11 wireless LAN. CCMP is based on AES in counter mode and CBC-MAC (CCM).

In CCM mode, the majority of the time of the protocol is spent on computing the AES algorithm. AES is used to generate the cipher text from the header of the 802.11 package as well as the package payload. Therefore, it is an incentive to have a hardware assisted ASIC or FPGA for computing the AES cipher text. This will alleviate the computing power from the main processor. One possible implementation of the CCMP protocol is to have the main processor in charge of the MAC layer and ASIC/FPGA device running simultaneously preforming the AES encryption/decryption algorithm.

The goal of the ASIC/FPGA is to compute the AES algorithm to generate the cipher text in the range of at least 11Mbps or higher. However, since the new 802.11a emergence, it is desirable to have the ASIC/FPGA to run at 54 Mbps. This paper will discuss the implementation of the AES algorithm with Xilinx Spartan 2 200K gates FPGA. In addition, there is some background on both AES and CCM.

II. Advanced Encryption Standard

The Advanced Encryption Standard (AES) is a symmetric block cipher that is based upon the Rijndael algorithm. AES can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. A full detailed description of the AES algorithm can be found at in the FIPS-197[1] document.

This paper was written as a final project for ECE 679: Advanced Cryptography in the Spring of 2003

A. Algorithm Specification

For the AES algorithm, the length of the input block, the output block and the State Array is 128 bits. The state array is the internal matrix upon which the data is manipulated and consists of four rows of bytes, each containing Nb bytes, where Nb is the block length divided by 32. Figure 1 illustrates the layout of the State Array.

State array

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

Fig. 1. State Array

The length of the Cipher Key, K, is 128, 192, or 256 bits. The number of rounds executed depend entirely on the length of the Cipher Key. Thus ensuring full permutation of all key bits on the state array. Table I illustrates this connection.

TABLE I
KEY-BLOCK-ROUND

	Key Length *(Nk words)*	Block Size *(Nb words)*	Number of Rounds *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

For both its Cipher and Inverse Cipher, the AES algorithm uses a round function composed of four different byte-oriented transformations: 1) byte substitutions using a substitution table (S-box), 2) shifting rows of the State array by different offsets, 3) mixing the data within each column of the State array, and 4) adding a Round Key to the State.

B. Cipher

The Cipher starts with the addition of the initial Round Key. The State array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first $Nr - 1$ rounds. The round function is parameterized using a key schedule that consists of a one-dimensional array of four-byte words derived using the Key Expansion routine.

All rounds, except the last one, include the individual transformations, *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. The last round does not include the the *MixColumns*[1] transformation.

[1]For more information about the individual round transformations, please refer to FIPS-197[1].

C. Inverse Cipher

The Cipher transformations can be inverted and then implemented in reverse order to produce a straightforward Inverse Cipher for the AES algorithm. The individual transformations used in the Inverse Cipher are *InvSubBytes*, *InvShiftRows*, *InvMixColumns*, and *AddRoundKey*.

D. Key Expansion

The AES algorithm takes the Cipher Key, K, and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total of $Nb(Nr+1)$ words. The algorithm requires an initial set of Nb words, and each of the Nr rounds requires Nb words of key data. The resulting key schedule consists of a linear array of 4-byte words, denoted $[w_i]$, with i in the range $0 \leq i < Nb(Nr+1)$.

The basic idea of the Key Expansion is to preform a *SubWord* on each four-byte word and apply the S-box transformation to each of the four bytes. The four byte word is then rotated in such a way that the input $[a_0, a_1, a_2, a_3]$ becomes $[a_1, a_2, a_3, a_0]$, thus performing a cyclic permutation. To further expand the key, a round constant word array, $Rcon[i]$ is applied every four rounds. This word array contains the values given by $[x^{i-1}, 00, 00, 00]$, with x^{i-1} being powers of x in the field GF(2^8).

III. CTR Mode and CBC-MAC Authentication (CCM)

In 1999, IEEE developed a new standard to handle information over wireless networks known as IEEE Std 802.11-1999. This standard had the ability to handle wireless traffic quite easily, but lacked the strict security standards required by institutions and corporations.

The Specification for Enhanced Security over Wireless Networks, IEEE 802.11i, requires a strong encryption standard, naturally, the use of AES is strongly desired. Therefore, [WHF][2] proposed a combination of counter mode encryption and CBC-MAC authentication. This method proves to be viable for several reasons; there is no known patent encumbrances, the modes have been used and study for a long time and have well-understood cryptographic properties, and they provide for good security and performance, whether implemented in hardware or software.

A. Generic CCM

CCM is a generic authenticate-and-encrypt block cipher mode. CCM is currently only defined for use with block ciphers with a 128-bit block size, such as AES.

For the generic CCM mode there are two parameter choices to be made. The first choice is M, the size of the authentication field. The choice of the value for M involves a trade-off between message expansion and the probability that an attacker can undetectably modify a message. Valid values are 4, 6, 8, 10, 12, 14, and 16 octets. The second choice is L, the size of the length field. This values requires a trade-off between the maximum message size and the size of the Nonce. Different applications require different trade-offs, so L is a parameter. Valid values are 2 to 8 octets (the value $L = 1$ is reserved).

TABLE II
PARAMETERS OF CCM MODE

Name	Description	Field Size	Encoding
M	Number of octets in the authentication field	3 bits	$(M-2)/2$
L	Number of octets the length field	3 bits	$L-1$

B. Inputs

The CCM mode requires the following inputs for authentication and encryption:

- An encryption key K suitable for block cipher (128, 192, or 256-bit key for AES).
- A nonce N of $15-L$ octets that shall be unique within the scope of any encryption key K. This means that the set of nonce values used with any given key shall not contain any duplicate values. Using the same nonce for two different messages encrypted with the same key destroys the security properties of the CCM mode.
- The message m, consisting of a string of $l(m)$ octets where $0 \leq l(m) < 2^{8L}$. The length restriction ensures that $l(m)$ can be encoded in a field of L octets.
- Additional authentication data a, consisting of a string of $l(a)$ octets where $0 \leq l(a) < 2^{64}$. This additional data is authenticated but not encrypted, and is not included in the output of this mode. It can be used to authenticate plaintext headers, or contextual information that affects the interpretation of the message.

C. Authentication

The first step in authenticating is to compute the authentication field T using CBC-MAC. We first define a sequence of blocks $B_0, B_1, \ldots, B_n$ and then apply CBC-MAC to these blocks. Figure 2 illustrates the format of the first block B_0.

Octet no:	0	$1 \ldots 15-L$	$16-L \ldots 15$
Contents:	Flags	Nonce N	$l(m)$

Fig. 2. Format of block B_0

The value $l(m)$ is encoded in most-significant-byte first order. Figure 3 illustrates the format of the Flags field. The *Rsrv* bit is reserved for future expansion and should always be set to zero. The *Adata* bits is set to zero if $l(a) = 0$, and is set to one if $l(a) > 0$. The M field encodes the value M as $(M-2)/2$. The L field encodes the size of the length of the field used to store $l(m)$. The parameter L is encoded as $L-1$ since $L = 1$ is reserved.

Bit no:	7	6	5	4	3	2	1	0
Contents:	Rsrv	Adata	M			L		

Fig. 3. Format of Flag field

The $Adata$ bit in the flag field determines if the authentication field is required, $l(a) > 0$. If a field is required then one or more blocks of authentication data are added. These blocks contain $l(a)$ and a is encoded in a reversible manner.

The string that encodes $l(a)$ is first constructed based on the following rules:

- $0 < l(a) < 2^{16} - 2^8$: The length field is encoded as two octets which contain the value $l(a)$.
- $2^{16} - 2^8 \leq l(a) < 2^{32}$: The length field is encoded as six octets consisting of the octets 0xff, 0xfe, and four octets encoding $l(a)$.
- $2^{32} \leq l(a) < 2^{64}$: The length field is encoded as ten octets consisting of the octets 0xff, 0xff, and eight octets encoding $l(a)$.

The blocks encoding a are formed by concatenating this string that encodes $l(a)$ with a itself, and splitting the result into 16-octet blocks, padding the last block with zeros if necessary. These blocks are appended to the first block B_0.

The message blocks are then added after the additional authentication blocks. The message blocks are formed by splitting the message m into 16-octet blocks and padding the last block with zeros if necessary. If the message string m is empty, then no blocks are added in this step.

We finally get the resulting sequence of blocks B_0, B_1, ..., B_n and compute the CBC-MAC by:

$$X_1 := E(K, B_0) \quad (1)$$

$$X_{i+1} := E(K, X_i \oplus B_i) \quad \forall i = 1, \ldots, n \quad (2)$$

$$T := first - M - bytes(X_{n+1}) \quad (3)$$

where $E()$ is the block cipher encryption function and T is the MAC value.

D. Encryption

To encrypt the message data, CCM uses CTR mode. This is done by first defining the key stream blocks by

$$S_i := E(K, A_i) \quad \forall i = 0, 1, 2, \ldots. \quad (4)$$

Figure 4 illustrates the formatting of the A_i field where i is encoded in most-significant-byte order.

Octect no:	0	$1 \ldots 15 - L$	$16 - L \ldots 15$
Contents:	Flags	Nonce N	Counter i

Fig. 4. Format of A_i

The *Flags* field is formatted as illustrated in Figure 5. The *Reserved* bits are set to zero and Bits 3, 4, and 5 are set to 0 as well. This technique is used to ensure that all A blocks are distinct from the B_0 block. Bits 0, 1, and 2 contain L, using the same encoding as B_0.

Bit no:	7	6	5	4	3	2	1	0
Contents:	Rsrv	Rsrv	0			L		

Fig. 5. Format of Flag field

The message is then encrypted by XORing the octets of the message m with the first $L(m)$ octets of the concatenation of $S_1, S_2, \ldots$. Note that S_0 is not used to encrypt the message. The authentication value U is computed by encrypting T with the key stream block S_0 and truncating it to the desired length.

$$U := T \oplus first - M - bytes(S_0) \quad (5)$$

E. Output

The final result c consists of the encrypted message m, followed by the encrypted authentication value U.

F. Decryption

The following input information is required for the decryption of the message:

- The encryption key K.
- The nonce N.
- The additional authentication data a.
- The encrypted and authenticated message c.

Decryption starts by recomputing the key stream to recover the message m and the MAC value T. The message and additional authentication data is then used to recompute the CBC-MAC value and check T. If the value T is not correct, the reciever shall not reveal any information except for the fact that T is incorrect. In particular, the receiver shall not reveal the decrypted message, the value T, or any other information.

IV. AES Implementation on a Xilinx Spartan IIE

Since CCM spends a lot of computational time on AES decryption/encryption, we would like to free the main processor by implementing the AES off chip on a separate FPGA. We choose a Xilinx Spartan II for its 200K gates. The implementation is partitioned into three modules: Input Interface, Output Interface, and the AES block cipher engine[2].

A. Input Interface

The Input Interface module is a parallel shift register of sorts. It takes the 128-bit plaintext message, M, one byte or 8-bits at a time. The module will continue to load each byte until the entire 128-bit plaintext message has been received. Once the entire message has been received, then the Input Interface module signals the AES engine to begin encryption. This module also has the ability to encrypt partial messages, meaning, messages that are not exactly 128-bits long.

B. Output Interface

The Output Interface module is very similar to the Input Interface module. Once a ciphertext, C, is ready, the Output Interface module is then triggered. It will then output the 128-bit ciphertext message one byte at a time for 16 iterations. Once the ciphertext message has been successfully outputted, then the system will be ready for another plaintext message.

C. AES Cipher Engine

The AES Cipher Engine module performs the AES encryption algorithm as described in Section II. This module is made up of three components; the Key Scheduling component, the Round Hardware component, and the control component.

[2]Due to time constraints and chip size, the authors were only able to implement the AES engine for encryption only. Future versions will contain decryption as well so that the whole AES process is off chip

1) The Round Hardware Component: is the basic data path and circuitry the computes a single AES round encryption. It contains the hardware for the ByteSub, ShiftRow, MixColumn, and AddRoundKey transformations. The ByteSub transformation requires the use of an S-Box, where each S-Box has an input of 8-bits or one byte. Since we are encrypting 128-bits in parallel on every round, the ByteSub requires 16 S-Boxes to perform the transformation. This is a huge restriction since each S-Box is 128 bits and since we need 16 of them, then we will need to find space for 2048 bits or 256 bytes just for the S-Boxes.

The Round Hardware Component was designed to allow for future scalability issues. It allows the designer the option to cascade the Round Hardware Component to create a pipelined architecture. This ability depends on the device's platform capabilities.

2) The Key Scheduling Component: performs the Round Key generation. This round key is generated dynamically each round based on the previous rounds key. It follows the same rules and algorithm as described in Section II-D.

3) The Control Component: controls the number of rounds of execution as well as the managing the Key Scheduling component. For our implementation, we used the standard 128-bit key and 128-bit data block size. Therefore, the Control Component allows for only 10 rounds of AES encryption. Due to the limitation of the Spartan II, only one round can be fit on the chip at a time.

The Control Component feeds the Key Generation the round constant. This allows for the Control Component to keep track of the round keys that are needed and generated.

D. Constraints

This project was implemented on Xilinx's Webpack version of the ISE development environment, therefore there were many feature that were lacking. These features could have enhanced the performance of the project. One example is that the Webpack does not have the "CoreGen" function that allows you to generate memory elements that are very compact. In addition, this project was based on Xilinx's Spartan 2, 200K gates. It is not feasible to have a pipelining architecture implemented on Spartan 2 chip. There are 16 duplicated S-boxes for 16 bytes. Additionally there are 4 duplicated S-boxes that are used in the key scheduling module for a total of 20 S-boxes per round. Beside the S-box, additional circuitry is required for mix column operation, which involves 128-bits XORing and Shifting operations. Needless to say, a chip with a much larger capacity would be better suited for this task.

E. Future Improvements

If the AES algorithm was implemented on an ASIC or a much larger FPGA, we could pipeline the round hardware component to greatly improve the overall performance. One possible scenario is to have ten pipeline stages for each round. Each stage could share S-Boxes, thus requiring a maximum of 16 S-Boxes for all ten rounds. The S-box data could be available during the clock transition to low.

F. Implementation Results

Table III gives the final implementation results using the Xilinx Webpack to layout the design. It describes the device utilization of AES and the timing reports.

TABLE III
RESULTS OF AES IMPLEMENTATION ON AN FPGA

Device Utilization Summary		
Selected Device	2s200pq208-5	
Number of Slices	2035 out of 2352	86%
Number of Slice Flip Flops	787 out of 4704	16%
Number of 4 input LUTs	3921 out of 4704	8%
Number of bonded IOBs	24 out of 144	16%
Number of GCLKs	1 out of 4	25%
Timing Report		
Speed Grade	-5	
Minimum Period	23.075ns	
Maximum Frequency	43.337MHz	
Minimum input arrival time before clock	16.783ns	
Maximum output required time after clock	16.756ns	
Maximum combinational path delay	No path found	

V. OVERVIEW OF IMPLEMENTATION

Now that we have a hardware interface module for AES, we need to discuss how this will interface with the rest of the system. Network protocols, particularly IEEE Std. 802.11i, are composed of several different layers. When a user wants to send information over the network, the information is separated into smaller, individual packets. These packets are then sent down through the layers and additional information is added to them to ensure proper communication, such as source address and destination address. The packet is then sent through the ethernet and arrives at the destination where it travels up the layers and is reassembled for the receiver.

One of the layers that the packet must go through in 802.11i, is the MAC layer. It is responsible for packing the packet and ensuring proper security. The security scheme that is being proposed is the CCM Mode Encryption as described in Section III-A. CCM Mode Encryption relies heavily on software block ciphers such as AES. This is where our implementation of AES on an FPGA comes into play. It is explicitly designed to interface directly with the MAC layer and perform all block cipher operations requested by the CCM Mode Encryption.

By including the use of a hardware block cipher, the processor can spend some of the compuational time on other computations. The ideal goal would be to have this AES hardware implementation be embedded on a Wireless Network Adapters for PCs and Laptops. Ultimately, this could relieve enough time from the processor to enable faster communication rates and generally both speed up and secure wireless communications.

VI. CONCLUSION

The AES implementation on the FPGA is a viable solution for improving the speed and processing power of CCM Mode Encryption. A much larger FPGA or ASIC would be preferred, since both encryption and decryption could be implemented

as well as some pipelining of processes. Although the design was implemented and intended to be interfaced with a micro-controller/microprocessor running the CCM Mode Encryption, our sources and tools prevented us from performing any 'real' life experiments. The simulation results preformed quite well and the authors are confident that the solution would work.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Çetin Koç and the ECE 679 Advanced Cryptography course for allowing us the opportunity to research this area of cryptography. The authors would also like to thank Dr. Alexandre Tenca for the use of the Spartan IIE board for the duration of this project.

REFERENCES

[1] *Announcing the Advanced Encryption Standard AES.* Federal Infromation Processing Standards FIPS, November 2001, FIPS Publication 197.
[2] D. Whiting, R. Housley, and N. Ferguson, "IEEE P802.11 wireless LANs: AES encryption & authentication using CTR mode & CBC-MAC," IEEE, Tech. Rep. IEEE 802.11-02/001r2, May 2002.
[3] J. R. Walker, "IEEE P802.11 wireless LANs: Unsafe at any key size; an analysis of the wep encapsulation," IEEE, Tech. Rep. IEEE 802.11-00/362, October 2000.
[4] P. A. Lambert, R. Housley, O. Letanche, and D. Stanley, "IEEE P802.11 wireless LANs: Alternate text for TGi 8.3.4," IEEE, Tech. Rep. IEEE 802.11-03-118r0, November 2002.