International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 06 | June 2019 www.irjet.net p-ISSN: 2395-0072

Design and Characterization of MAEC IP Core

Shinnu M.S[1], Sujeesh K.V[2], R. Nandakumar[3]

[1]*M.Tech Student, Department of ECE, GEC Idukki, Kerala, India*
[2]*Assistant Professor, Department of ECE, GEC Idukki, Kerala, India*
[3]*Scientist /Engineer 'D' NIELIT Calicut, Kerala, India*

Abstract - *In many applications such as cellular telephony or wireless local area networking, Message Authentication Codes (MACs) that provided error correction capability (ECC) would be ideal. As conventional MACs are vulnerable to any alteration in the message, causing straightaway rejection of messages and retransmission is requested. This retransmission leads to wastage of energy and increase latency. In this paper, introduced a MAC with integrated error correction capability called MAEC. A simple scheme for generation and decoding of single-byte error-correcting codes based on Cellular Automata (CA) is presented and it is implemented on Xilinx Spartan-6 LX16-FPGA (XC6SLX16-CSG324C).*

Key Words: MAC, Byte Error Correcting Code, Cellular Automata, Mersenne Prime, VLSI

1. INTRODUCTION

Message Authentication Code (MAC), also referred to as a keyed hash function, is a cryptographic primitive that verifies the data integrity and the authenticity of its sender. However, as conventional MACs are susceptible to any alteration in the message, even simple channel noises are detrimental to its functionality, causing straightaway rejection of authentic messages. Though this is preferable in some situations, many of the less information sensitive applications, such as image and other multimedia communications, can allow few errors occurred during transmission. The message should be rejected only if large number of errors are present, indicating a probable attempt of forgery. But, classical MAC algorithms do not convey any information about the number or location of the errors. This concludes the need for a MAC construction technique with integrated error correction property that offers some resilience against random channel noises, especially in environments where latency is a concern or resources are limited [1].

Various coding techniques have been used for error detection and correction. Error correcting codes have been used to enhance reliability of the system and data integrity. The key idea is to add checkbytes or redundancy to the information bytes at the encoder so that decoder can retrieve the information bytes from the received block possibly corrupted by the channel noise. Reed-Solomon (RS) codes are widely used codes for error correction, because these codes can detect both random errors as well as burst errors. RS code has found many applications in storage devices (CD, DVD), wireless communications, high speed modems and satellite communications. The complexity of RS encoder and RS decoder increases with the error correcting capability of the code. Hence many researchers have put their effort to minimize the complexity of RS encoder/decoder for communication applications. But VLSI system designer always prefers to have simple, regular and cascadable structure with local interconnection for reliable high speed operations of the circuit. It has been found that these factors are supported by local neighborhood Cellular Automata (CA). Here, proposed a practical and secure integrated ECC-MAC design, named MAEC. In this design CA based byte error correcting code has been proposed. The required hardware for the proposed design is limited compared to the existing techniques used for RS code.

1.1 Cellular Automata (CA)

Cellular Automata (CA) consists of a number of cells arranged in a regular fashion where the state transitions of the cell depend on the state of its neighbours and each cell consists of a D flip-flop and a combinational logic implementing the next-state function [2]. The next-state function for a three- neighbourhood one-dimensional (1D) CA, a cell can be expressed as follows:

$$q_i(t+1) = f[q_{i-1}(t), q_i(t), q_{i+1}(t)]$$

where $q_i(t+1)$ and $q_i(t)$ are the output state of the i^{th} cell at the $(t+1)^{th}$ and t^{th} time step respectively and *f* denotes the local transition function realized with a combinational logic, and is known as a *"rule"* of the CA. For a two state three CA, there are 2^3 that is, 8 distinct neighbourhood configurations and 256 (2^8) distinct next state functions. However, it turns out that out of 2^8 possible boolean functions, only two linear functions are of prime interest viz., rule 90 and 150. The state of i^{th} cell at time instant *t* can be expressed as:

$$s_i^{t+1} = s_{i-1}^t \oplus d_i\, s_i^t \oplus s_{i+1}^t, \quad d_i = \begin{cases} 0, \text{ if } d_i \longrightarrow \text{rule 90} \\ 1, \text{ if } d_i \longrightarrow \text{rule 180} \end{cases}$$

Thus, a CA rule can be completely specified by an n-tuple $R=[d_1,d_2,.....d_n]$ called *Rule Vector* of CA. The transition function $f: \{0,1\}^n \rightarrow \{0,1\}^n$ of an *n*-cell CA can be represented by a n×n square matrix *T*, referred to as the characteristic matrix of the CA is in the form:

$$T = \begin{bmatrix} d_1 & 1 & 0 & \cdots & \cdots & 0 & 0 \\ 1 & d_2 & 1 & \ddots & & & 0 \\ 0 & 1 & d_3 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & d_{n-2} & 1 & 0 \\ 0 & & & \ddots & 1 & d_{n-1} & 1 \\ 0 & 0 & \cdots & \cdots & 0 & 1 & d_n \end{bmatrix}.$$

A CA characterized by EXOR or EXNOR dependence is called an additive CA. If in a CA, the neighbourhood dependence is EXOR, then it is called non-complemented CA and the corresponding rule is non-complemented rule. For EXNOR dependence, CA is called complemented CA and the such rule is called complemented rule. There exists 16 additive rules which are:

Rules:- 0, 15, 51, 60, 85, 90, 102, 105, 150, 153, 165, 170, 195, 204, 240, 255.

A uniform CA is one in which same rule applies to all cells while in hybrid CA (Fig.1.1) different rules are applied to different cells. In this design, rule 90 and rule 150 are used for cells in CA. Based upon this, designed an efficient error detecting and correcting code.

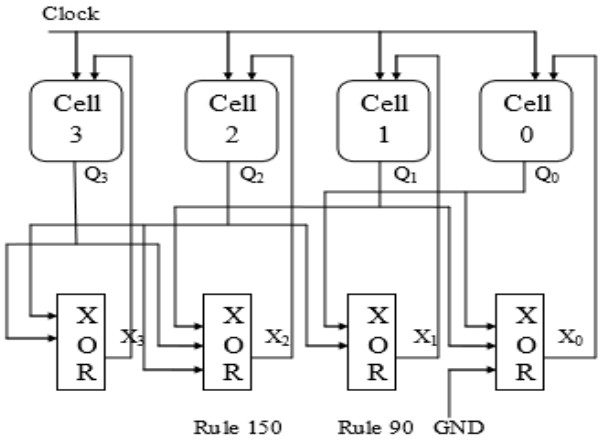

Fig. 1.1 A Hybrid Cellular Automata

2. DESIGN OVERVIEW

This section describes the design of an authentication scheme with integrated error correction capability, suitable for practical applications. Proposed ECC is based on CA, making it highly efficient in hardware. The authentication algorithm works in two different stages, MAC-GENeration stage and the MAC-VERification stage.

2.1 Block Diagram

MAEC IP core comprises 2 modules- Transmitter (Sender) and Receiver side as shown in Fig. 2.1.

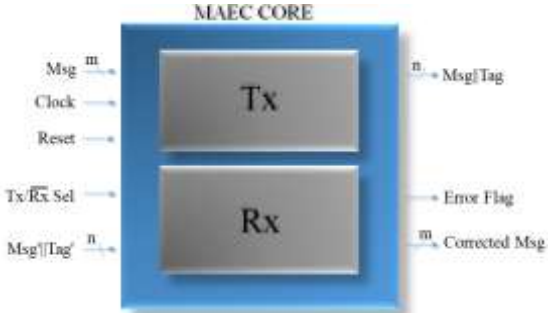

Fig.2.1 Basic I/O Diagram of MAEC IP core

The input of the transmitter side is Msg and output is Msg ||Tag. The input of the Receiver side is Msg' ||Tag' and output is Corrected Msg. There is a Tx/Rx signal to select transmitter side or receiver side and an Error Flag signal to indicate an uncorrectable error. Clock and Reset signals are common to both transmitter and receiver end.

2.2 MAC Generation

The MAC generation process starts with sharing two keys k_1 and k_2 between the sender and the receiver before the initiation of a session. At first, the data goes through a optional padding phase if necessary and partitioned into blocks of size w (w is the number of the cells in the cellular automata, used for error correction). Now, with the help of key k_1, a CA based error correcting code is selected, which serves the purpose of forward error detection/correction. Next, the checkbytes computed by the ECC, are mixed with key k_2 in a non-linear fashion by NMix to get the MAC. This MAC value is then transmitted to the receiver along with the data as (Message, MAC) pair. The MAC generation process is depicted in Fig.2.2 and Algorithm 1 shows the detail.

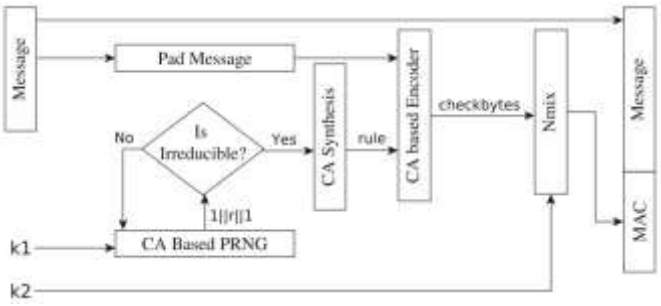

Fig.2.2 MAC generation at the sender

ALGORITHM 1. MAC-GEN(t,m,k_1,k_2) ⊳$|m| \le 2^w$, ⊳ $|k_1| = w-1$, $|k_2| = w$

1.**Procedure** PREPROCESS (m)
2. m′ ⟵ P(m) ⊳ P is the padding function, $|P(m)| = w \times n$, $n \in Z^+$
3. $\{m_1,m_2,\ldots,m_n\}$⟵ B(m′) ⊳ B partitions m′ into n blocks
4. **end procedure**
5. **procedure** SELECT RANDOM CA (k_1)
6. r_1⟵ prefixed linear CA rule ⊳ $|r_1| = w-1$, ⊳ r1 is any maximal length 90,150 CA rule

7. $s \leftarrow Cr_1(k_1)$ ▷ Apply CA with rule r_1, CA initialized with seed k_1
8. $S \leftarrow \{1, s, 1\}$
9. **while** IRREDUCIBLE (S) = FALSE **do** ▷ Rabin's Test [7]
10. $s \leftarrow Cr_1(s)$
11. $S \leftarrow \{1, s, 1\}$
12. **end while**
13. $r_2 \leftarrow$ SYNTHESIZE CA(S)
14. **return** r_2 ▷ r_2 is the randomly selected CA
15. **end procedure**
16. **procedure** COMPUTE MAC(B(m'), r_2, t, k_2)
17. $c_i \leftarrow$ CA ENCODER(B(m'), t, i), $0 \le i \le 2t\text{-}1$
18. $mac_i \leftarrow NMix(c_i, k_2)$
19. **return** MAC $\leftarrow \{mac_0, mac_1, ..., mac_{2t-1}\}$
20. **end procedure**

2.3 MAC Verification

At the receiving end, verification of the MAC is performed to detect the presence of errors in the transmitted data, if any. At first, checkbytes *C* are recomputed from the received data. This step is similar to the checkbyte computation procedure described in the MAC generation process. In parallel to this, the received MAC is subjected to the inverse of the non-linear transformation (INMix) to get back the original checkbytes *C'* transmitted by the sender. Next, the received checkbytes *C'* and the computed checkbytes *C* are passed to the CA based decoder to find out if the transmitted message has arrived correctly, or not. If all of the syndromes are zero, the message is readily accepted as its error-free. However, if any error creeps in, a decision is to be taken depending on the number of words in error. If the number of errors are greater than the error correction capability of the code, it leads to straightaway rejection of the message. Otherwise, its corrected and subsequently accepted. The whole process is depicted in Fig.2.3, and Algorithm 2 shows the detail.

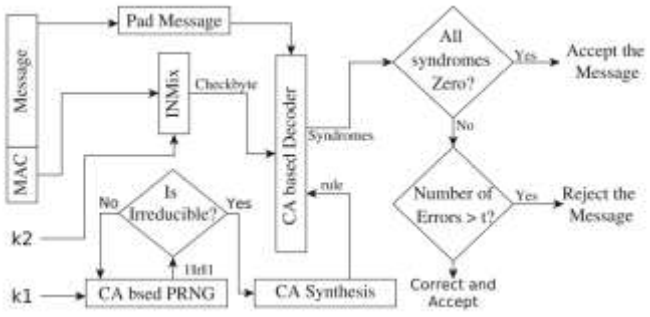

Fig. 2.3 MAC verification at the Receiver

ALGORITHM 2. MAC-VER(m, MAC, t, k_1, k_2)

1. PREPROCESS(m)
2. SELECT RANDOM CA (k_1)
3. C ⟵ CA ENCODER(B(m'), t, i), $0 \le i \le 2t\text{-}1$ ▷ C = $\{c_0, c_1, ... c_{2t-1}\}$
4. C' ⟵ INVERSE NMix(mac_i) ▷ C' = $\{c_0', c_1', ... c_{2t-1}'\}$
5. X ⟵ C ⊕ C' ▷ X = $\{x_0, x_1, ... x_t\}$ is the syndrome
6. X' ⟵ $\{ x_i : x_i \neq 0\}$
7. **if** X' = ϕ **then** ▷ All zero syndrome
8. **return** Accept
9. **else**
10. **if** No. of Errors > t **then**
11. **return** Reject ▷ More that t errors, so discard m
12. **else**
13. Correct and Accept
14. **end if**
15. **end if**

2.4 Random Selection of CA with Key k_1

This section describes how to randomly select a CA with the help of key k_1 and still make efficient implementation of it. It is to be noted that if the checkbytes are also used to provide authenticity, it will be vulnerable to simple forgeries if a fixed known ECC is used.

To come up with an efficient solution, we exploit the work of Cattel and Muzio [8], where they presented a detailed method for the synthesis of a one-dimensional linear hybrid CA from a given irreducible polynomial. But, to be able to use such CA based ECC, one must first find a maximum length CA i.e. having a period of $(2^n - 1)$, *n* being the number of cells in the CA. To tackle this issue, exploit another important result that if a given irreducible polynomial is primitive, then the synthesized CA will be unique as well as of maximum-length. This result is interesting, since it maps the problem of randomly selecting a linear code to the problem of finding a primitive polynomial over a finite field. However, finding primitive polynomials in large finite fields is a non-trivial problem. The verification of primitive polynomials and its associated complexity is too costly to implement either in software or in hardware, and hence, not suitable for any practical application. The fastest known algorithms for testing primitivity of polynomials of degree *n* require the factorization of $(2^n - 1)$ to be known beforehand, in addition to find a primitive root over the defining field. To overcome this, use the result given by the following theorem, which establishes a one-to-one correspondence between primitive and irreducible polynomials.

Theorem 1. *If $2^r - 1$ is a prime number (Mersenne Prime) then all degree-r irreducible polynomials are also primitive* [13].

The result furnished in Theorem 1 reduces the hard problem of finding a primitive polynomial to the easier problem of finding an irreducible polynomial. Once such a *w* $(2^w - 1 \in P)$ is fixed, one can choose any irreducible polynomial of degree *w*, and synthesize it into a maximum-length CA using the technique described in [8].

Due to the arguments stated above, random CA based ECC selection turns out to be equivalent to selecting an irreducible polynomial randomly. First, fix GF(2^w) as the extension field in accordance with Theorem 1. A polynomial $f(x) = \sum b_i . x_i$, $0 \le i \le w$ over GF(2^w) is represented as the bit string $\{b_w, b_{w-1}, ...b_0\}$. Now, the key k_1 serves as the seed to a CA based PRNG [5] [6]. A (w−1)-bit CA based PRNG produces random patterns of (w−1)-bits, to which 1 is concatenated at either end to get a pattern of (w+1)-bits. The reason behind prepending 1 is quite straightforward as the polynomials are monic . The appending of 1 at the LSB is also trivial. Otherwise, x will be a non-trivial factor of the polynomial, implying it not irreducible. The polynomial corresponding to the resulting bit-pattern is subjected to Rabin's irreducibility test [7]. This whole process continues until find an irreducible polynomial. Finally, this irreducible polynomial is synthesized into an equivalent maximum-length CA according to [8]. This step corresponds to SELECT RANDOM CA (k_1) in Algorithm 1.

2.5 Calculation of Checkbytes

After a suitable maximum-length CA is selected, it is used for the calculation of checkbytes which helps to keep track of transmission errors. But before any computation begins, the data goes through an optional padding phase to make it a multiple of word-size (w), where $w : 2^w - 1 \in P$ and partitioned into blocks of size equal to w. This corresponds to procedure PREPROCESS(m) of Algorithm 1. After this, the data is passed to the CA based Encoder for the computation of checkbytes. The checkbytes and syndromes for a t-word error correcting code, where each word is n-bit long, can be generated using an n-cell maximum-length CA in the following manner:

$$C_i = \bigoplus_{J=0}^{N-1} T^{i\times(N-1-j)} [B_j], \quad \begin{cases} i = \{0, 1, ..., 2t\}; \\ B = [B_0, B_1, ..., B_{N-1}]; \\ T \longrightarrow \text{characteristic matrix of the CA} \end{cases}$$

$$S_i = C_i \oplus C_i' \ , \quad 0 \le i \le 2t \ , \quad \begin{cases} C_i \text{ is the received checkbyte} \\ C_i' \text{ is the checkbyte computed} \end{cases}$$

Now, elaborate how the choice of t i.e. the error correction capability of the code, relates to the security of the scheme. Here, the inherent belief lies on the fact that in wireless transmission systems, burst errors are more common in nature, and are confined to a small part of the data. Thus, it can at most affect a few words of information rather than spanning over the whole data. However, if some intentional tampering or some outright forgery happens, the chances are more that it will be spanned throughout the data. This causes a large number of errors and the message is discarded accordingly. This leads to the careful choice of t by the user.

2.6 Generation of MAC Using ECC

This section describes how the checkbytes computed in the previous section also serves the purpose of authentication. However, due to its linear nature checkbytes alone are not sufficient enough to provide authentication, as it's linear property makes it susceptible to several well known cryptanalysis techniques like linear/differential cryptanalysis. However, a non-linear operation can prevent these type of threats, making it secure. So, for this purpose, proposed algorithm uses NMix as the nonlinear key mixing function in the design. The choice of NMix is driven by the fact that it is a non-linear bent function, i.e. it is as different as possible from all linear or affine functions and thus, naturally hard to approximate.

NMix is applied to the checkbytes $C \leftarrow \{c_0, c_1, ..., c_t\}$ with key k_2 as the random pad r to get the MAC$\leftarrow \{mac_0, mac_1, ..., mac_t\}$ as output. These MAC values along with the message, is then sent to the receiver as the authenticator. It is to be noted that applying non-linear mixing may propagate the error to multiple bits within a single word. However, as CA based decoder deals with byte-errors rather than bit-errors, NMix does not affect the error correction property of the code.

2.7 Forward Non-Linear Mixing (NMix)

NMix function operates on two n-bit variables X= $(x_{n-1}, x_{n-2}, ..., x_0)$, K = $(k_{n-1}, k_{n-2}, ..., k_0)$ and produces a n-bit output variable Y = $(y_{n-1}, y_{n-2}, ..., y_0)$ where each output bit is related to the input bits by the following relationship:

$$y_i = x_i \oplus k_i \oplus c_{i-1}$$

$$c_i = \bigoplus_{j=0}^{i} x_j k_j \oplus x_{i-1} x_i \oplus k_{i-1} k_i$$

where $0 \le i < n$, $c_{-1} = 0$, $x_{-1} = 0$, $k_{-1} = 0$ and c_i is the carry term propagating from i^{th} bit position to $(i+1)^{th}$ bit position. The end carry c_{n-1} is neglected. Each output bit y_i is balanced for all $0 \le i < n$.

2.8 Inverse Non-Linear Mixing (INMix)

In inverse mixing, the mixer takes two n-bit variables Y = $(y_{n-1}, y_{n-2}, ..., y_0)$, K = $(k_{n-1}, k_{n-2}, ..., k_0)$ as inputs and produces an n-bit output X= $(x_{n-1}, x_{n-2}, ..., x_0)$. Inverse mixing operation can be defined as:

$$x_i = y_i \oplus k_i \oplus d_{i-1}$$

$$d_i = \bigoplus_{j=0}^{i} x_j k_j \oplus x_{i-1} x_i \oplus k_{i-1} k_i$$

where $0 \le i < n$, $d_{-1} = 0$, $x_{-1} = 0$, $k_{-1} = 0$ and d_i is the carry term propagating from i^{th} bit position to $(i + 1)^{th}$ bit position. The end carry d_{n-1} is neglected.

2.9 Error Correction

This section presents a scheme for the decoding of the CA based Single Byte Error Correcting (SbEC) codes. The primary objective is to retrieve the correct information bytes. So when an error is detected in the checkbytes only, it can be ignored. This assumption is justified for many applications, where the main criteria is the correct retrieval of information bytes.

Definitions:-

1. Error syndrome:- Let C_i denote the i^{th} check byte for a given information block (in absence of errors), and C_i' is the i^{th} checkbyte recomputed from the received information bytes (with possibility of errors present). Then the error syndrome corresponding to the i^{th} checkbyte is defined as:

$$S_i = C_i \oplus C_i' \quad , \quad 0 \le i \le 2t$$

2. Error Byte:- Let B_{N-1-j} and B'_{N-1-j}, respectively, denote the j^{th} information byte in absence of errors and the j^{th} information byte as received. As noted in Fig. 2.4, byte B_j counted from left is same as byte B_{N-1-j} counted from right. Then the error byte for B_j is defined as:

$$E_j = B_{N-1-j} \oplus B'_{N-1-j} \quad , \quad j \le 0$$

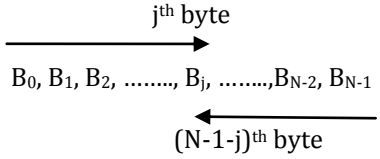

Fig. 2.4 Byte position counted from left and right

Clearly, an all-zero error byte is indicative of the fact that the corresponding information byte is fault-free, and a non-zero value indicates the byte positions in error. The objective of error correction is to deduce the error vector $[E] = [E_0, E_1, \ldots, E_{N-1}]$ from the received information bytes $[B'] = [B'_{N-1}, B'_{N-2}, \ldots, B'_0]$ and the received checkbytes $[C] = [C'_0, C'_1, \ldots, C'_{2t}]$. The correct information bytes can then be obtained as:

$$[B] = [B'] \oplus [E]$$

On receiving a block of bytes, the checkbytes are recomputed using the same logic as in encoding. The error syndromes S_0, S_1 and S_2 are computed by EXOR-ing the newly generated checkbytes with the received checkbytes. Assume that the errors have occurred in the i^{th} byte from the left, i.e., the corresponding error byte is E_i. The checkbytes generated on the receiving side can be expressed as:

$$C'_0 = B_{N-1} \oplus B_{N-2} \oplus \ldots\ldots B_0 \oplus E_i$$
$$C'_1 = B_{N-1} \oplus T\,[B_{N-2}] \oplus \ldots\ldots T^{N-1}\,[B_0] \oplus T^i\,[E_i]$$
$$C'_2 = B_{N-1} \oplus T^2\,[B_{N-2}] \oplus \ldots\ldots T^{2(N-1)}\,[B_0] \oplus T^{2i}\,[E_i]$$

Hence, the syndrome generated on the receiving side can be expressed as

$$S_0 = C_0 \oplus C'_0 = E_i$$
$$S_1 = C_i \oplus C'_1 = T^i\,[E_i]$$
$$S_2 = C_i \oplus C'_2 = T^{2i}\,[E_i]$$

Decoding Algorithm:-

1. If all the syndrome bytes S_0, S_1 and S_2 are zeros then there is no error in the received information block

2. If any one of the syndrome bytes S_k (k= 0, 1, 2) is non zero but the other two are zeros, then only the checkbyte C_k is in error

3. If more than one of the syndrome bytes are non zero, try to find an i such that
$$T^i[S_0] = S_1 \quad \text{and} \quad T^{2i}[S_0] = S_2$$

4. If such an i exists, then a single error has occurred in the $(N-1-i)^{th}$ byte, and the error byte is S_0

5. If no such i can be found, it indicates the presence of an uncorrectable error (double or more)

3. EXPERIMENTAL RESULTS

The design has been implemented and verified in Xilinx Spartan-6 LX16 FPGA (XC6SLX16-CSG324C). Xilinx ISE 14.1 is used as an FPGA development environment during the implementation process (i.e., synthesis, map, place & route). The Table 3.1 shows the overall resource consumption.

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	1,533	18,224	8%
Number of Slice LUTs	1,016	9,112	11%
Number used as Memory	118	2176	5%
Number used as Logic	824	9112	9%
Number of occupied Slices	691	2278	30%
Number of LUT Flip Flop pairs used	1598		
Number of bonded IOBs	3	232	1%
Number of BSCANs	1	4	25%
Number of STARTUPs	0	1	0%
Average Fanout of Non Clock Nets	3.11		

Table 3.1 Resourse Utilization Summary

Hardware implementation of MAEC core has been done using FPGA Spartan 6 (XC6SLX163-CSG324C). After configuring the device, analyzed the design using Xilinx ChipScope Pro Analyzer as shown in Fig 3.2. It mainly consists of 3 cores- VIO, ILA and ICON.

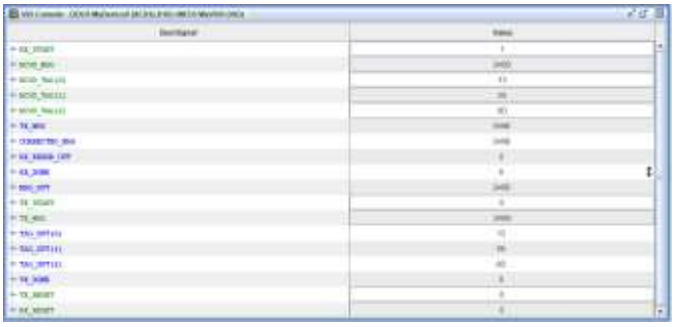

Fig 3.2 ChipScope Pro Analyzer Output

After implementing the design on a Xilinx Spartan-6 LX16-FPGA (XC6SLX16CSG324C), the maximum clock frequency is found out as 143.225 MHz and the total power consumption is obtained as 0.055W.

4. CONCLUSIONS

This paper presented a MAC algorithm for message authentication with single byte error correction capability using cellular automata based approach. It also introduced an efficient technique for random selection of an error correcting code making it secure. The CA based encoding/decoding scheme for correcting and detecting byte errors of such a code is suitable from VLSI design view point and attractive for its simplicity and regularity.

The behavioral description of the proposed design is written in VHDL and simulated using Xilinx isim 14.1 platform. Then the design is successfully implemented on a Xilinx Spartan-6 LX16 FPGA (XC6SLX16-CSG324C). Design verification is performed with the help of ChipScope ILA tool of Xilinx. The maximum frequency of operation of core is obtained from Post-PAR static timing report and is found as 143.225 MHz. The synthesis result shows that the circuit requires much less hardware. So CA based implementation of the proposed scheme provides a simple cost effective solution.

ACKNOWLEDGEMENT

This work is supported by National Institute of Electronics and Information Technology (NIELIT), Calicut.

REFERENCES

[1] Sengupta, Abhrajit, et al. "AEC: A Practical Scheme for Authentication with Error Correction.", International Conference on Security, Privacy, and Applied Cryptography Engineering., Springer, Cham,2014.

[2] Bhaumik, Jaydeb, Dipanwita Roy Chowdhury, and Indrajit Chakrabarti. "An improved double byte error correcting code using cellular automata." International Conference on Cellular Automata. Springer, Berlin, Heidelberg, 2008.

[3] D. Roy Chowdhury, I. Sen Gupta, P. P. Chaudhuri, " CA-Based Byte Error- Correcting code," IEEE Transaction on Computers, vol.44, no.3, pp. 371-382 Mar. 1995.

[4] Bhaumik,J., RoyChowdhury,D.: "An integrated ecc-mac based on rscode.", Transactions on Computational Science 4, 117–135 2009.

[5] Comer, J.M., Cerda, J.C., Martinez, C.D., Hoe, D.H.K.: "Random number generators using cellular automata implemented on fpgas." In: 2012 44th Southeastern Symposium on System Theory (SSST), pp. 67–72 , March 2012

[6] Wolfram, S.: "Random sequence generation by cellular automata". Advances in Applied Mathematics 7(2), 123–169 ,1986

[7] Gao, Shuhong, and Daniel Panario. "Tests and constructions of irreducible polynomials over finite fields.", Foundations of computational mathematics. Springer, Berlin, Heidelberg, 346-361, 1997.

[8] Cattell, Kevin, and Jon C. Muzio. "Synthesis of one-dimensional linear hybrid cellular automata." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15.3: 325-335, 1996.

[9] Fournaris, Apostolos P., and O. Koufopavlou. "A systolic inversion architecture based on modified extended euclidean algorithm for GF (2k) fields." , 2006 13th IEEE International Conference on Electronics, Circuits and Systems. IEEE, 2006.

[10] Bhaumik, Jaydeb, and Dipanwita Roy Chowdhury. "Nmix: An Ideal Candidate for Key Mixing.", SECRYPT. 2009.

[11] Bhaumik, Jaydeb, Balaji Janakiram, and Dipanwita Roy Chowdhury. "Architectural design of CA-based double byte error correcting codec." 2008 IEEE Region 10 and the Third international Conference on Industrial and Information Systems. IEEE, 2008.

[12] Chowdhury, Dipanwita Roy, and Parimal Pal Chaudhuri. "Architecture for VLSI design of CA based byte error correcting code decoders.", Proceedings of 7th International Conference on VLSI Design. IEEE, 1994.

[13] Golomb, S.W.: Shift register sequences (1967)