Journal of Advanced Engineering Trends
ISSN : 2682 - 2091

Vol.40, No.2. July 2021

http://jaet.journals.ekb.eg

EFFICIENT IMPLEMENTATION OF LIGHTWEIGHT MDS MATRIX ON FPGA

Ahmed A.H. Abd-Elkader [1]*, Mostafa Rashdan [2], El-Sayed A.M. Hasaneen [3], Hesham F.A. Hamed [4,5]

[1] Qussier Telecom, Telecom Egypt, Red Sea, Egypt
[2] Faculty of Energy Engineering, Aswan University, Aswan, Egypt
[3] Faculty of Engineering, Aswan University, Aswan, Egypt
[4] Faculty of Engineering, Minia University, Minia, Egypt
[5] Faculty of Engineering, Egyptian - Russian University, Cairo, Egypt
* corresponding author E-mail: eng.ahmed_a_abdelkader@yahoo.com

ABSTRACT

Recently, studying of Maximum Distance Separable (MDS) matrix has become a topic of interest. The MDS matrix is the most important component of the diffusion layer in block ciphers. This paper introduces an optimized, low-cost hardware construction of Galois Field $GF(2^8)$ 4×4 MDS matrix. The proposed design is implemented on Field programmable Gate Array (FPGA). The proposed design is synthesized targeting Virtex-7 FPGA using Xilinx ISE Design suite. Xilinx primitives $LUT6$ and $LUT6\_2$ were used to control exactly the component placement in the design to maintain the minimum occupation area. The pipeline and parallel implementation techniques were used to improve the speed performance. The verification of the functionality of the proposed design has been proved using the ModelSim simulation tool. The synthesis result of the proposed design shows that, the new proposed architecture provides very competitive area and throughput trade-offs. In comparison with other related designs, the proposed design occupies the least area with the minimum time delay. The area of the developed MDS matrix design was significantly reduced, 68 LUT, with high throughput of 21.178 Gbps. The proposed design is a suitable candidate for lightweight cryptographic implementations.

Keywords—*MDS, FPGA, Xilinx, Virtex-7, LUT6, Lightweight.*

1. INTRODUCTION

In the information technology era; the private information has to be protected from hackers and third-parties, furthermore, protecting financial transactions, military and industrial secrets from adversaries. Cryptography was used in ancient ages, it was known as encryption, to transfer messages in secret ways by converting it from readable state to unreadable sense. Cryptography supplies mechanisms, techniques, and tools for confidential and authenticated communication, and for accomplishing secure and authenticated transformations over the Internet and other networks. The modern cryptography ciphering and deciphering systems are composed of sets of complicated mathematical algorithms together with key management procedures to make it hard to break by any third-parties [1]. The Substitution-Permutation Networks (SPN) have a well-known structure to construct the Symmetric-key ciphers such as AES [2]. The main components of SPN are diffusion matrices, S-boxes, and key schedule. Maximum Distance Separable (MDS) matrix is used as the diffusion matrices. The MDS is used also in Feistel Ciphers such as Twofish [3]. The increasing of the smaller personal devices in the past decade to manage secure data has encouraged the investigation of modern cryptographic primitives with low-cost area. That urge required a novel lightweight cryptographic hardware implementation. Many studies of MDS codes have been performed, such as classification of MDS codes [4,5], non-Reed-Solomon MDS codes [6], Recursive construction of MDS matrix [7,8], Irregular MDS Array [9] ; and lightweight MDS matrices [10–12].

The low-cost MDS is used also in lightweight block cipher [13,14]. There are two focal approaches towards lightweight MDS matrices. The first approach looks for new matrices that permit a good implementation by design. The second approach, deal with a given MDS matrix and lower its cost by finding a better implementation. The second approach is used to optimize the implementation of a standardized cipher. This paper is based on the second approach in designing of a lightweight MDS matrix. We mostly focus on the cost of a hardware implementation of MDS matrix. The MDS of block cipher Twofish is chosen to be optimized and synthesized as an example of 4-by-4 MDS matrix in $GF(2^8)$ [3].

Received:14July, 2020, Accepted:31July, 2020

Field Programmable Gate Array (FPGA) is an excellent choice for implementation of algorithms which need the benefits of the flexibility of software platforms, and the speed of the Very Large-Scale Integrated Circuits (VLSI), furthermore, the timeless of implementation and development. Due to its reconfigurability and other benefits; Xilinx FPGA was chosen to implement the proposed MDS module. The area of Xilinx FPGA is measured with the number of Look Up Tables (LUT), and the delay is measured in ns . The Xilinx instantiation primitives Lookup Table (LUT) have been used [15], $LUT6$ and $LUT6\_2$,these components are instantiated directly into the design to control the exact placement of the individual blocks.

The remainder of the paper is organized as follows. Section 2 demonstrates in brief the construction of Twofish algorithm and its main blocks. Section 3 presents the MDS matrix and its functions. Section 4 describes the implementation of the proposed design. Section 5 presents the simulation and synthesis results. Finally, section 6 concludes this work.

2. OVERVIEW OF TWOFISH ALGORITHM

Twofish is a symmetric key block cipher accepts a variable key size up to 256 bits and a block size of 128 bits. At the Advanced Encryption Standard (AES) contest; Twofish was one of the five contestants. The algorithm of the cipher is a 16-round Feistel network. The bijective F function made up of four pre-computed key- dependent 8-by-8- bit S-boxes, a fixed 4-by-4 maximum distance separable matrix (MDS) over $GF(2^8)$, bitwise rotations , a pseudo-Hadamard transform (PHT), and a relatively complex key schedule. In Twofish there are two 1-bit rotations which are non-Feistel elements. The structure of the Twofish algorithm is shown in Fig. 1.

The Feistel network is used for converting any function, called F-function, into a permutation [16]. The 16-bytes input plaintext $p_0, \dots, p_{15}$, 128-bit block size, are split into four 32-bit words $P_0, \dots, P_3$ using little-endian conversion (1), [3].

$$P_i = \sum_{l=0}^{3} p_{l(4i+l)}.2^{8l} \qquad i = 0, \dots ,3 \qquad (1)$$

$P_0, \dots, P_3$, plaintext 32-bit words, are XORed with four 32-bit expanded key words, k_i, in input whitening process (2). R_i is the output of whiting process.

$$R_i = P_i \; XOR \; k_i \qquad i = 0, \dots ,3 \qquad (2)$$

Afterwards, there are sixteen rounds; in each of them the two 32-bit words R_0 and R_1 are used as an input to the F function, R_2 and R_3 are XORed with the output of F function. R_3 are rotated left by one-bit before the XOR procedure, and R_2 are rotated right by one-bit after the XOR procedure (3).

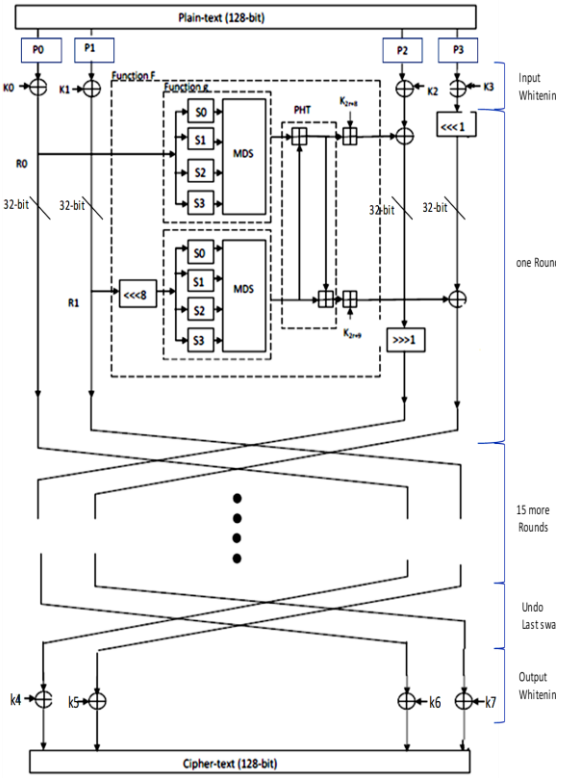

Fig. 1. Twofish algorithm structure

$$R_{0,r+1} = ROR\left(R_{2,r} \oplus F_{0,r}, 1\right)$$
$$R_{1,r+1} = ROL\left(R_{3,r}, 1\right) \oplus F_{1,r} \quad r = 0, \dots, 15 \quad (3)$$

r is denoted for the round turn. Finally, the two halves are swapped for the next round (4).

$$R_{2,r+1} = R_{0,r}$$
$$R_{3,r+1} = R_{1,r} \qquad (4)$$

The swap of round 15 is reversed, the output $R_{0,16}, R_{1,16}$, $R_{2,16}$, and $R_{3,16}$ are XORed with the corresponding 32-bit expanded keys in the output whitening process (5).

$$C_i = R_{(i+2)mod\,4,16} \oplus k_{i+4} \quad i = 0, \dots ,3 \qquad (5)$$

Subsequently, the four 32-bit words are divided into sixteen 8-bit words $c_0, \dots, c_{15}$ via the same Little-endian conversion method used for the plain text (6).

$$c_s = \left\lfloor \frac{C_{\lfloor s/4 \rfloor}}{2^{8(s \bmod 4)}} \right\rfloor mod\; 2^8 \qquad s = 0, \dots, 15 \qquad (6)$$

s is denoted for the byte order of the cipher text. As shown in *Fig. 1*, the F-function is 64-bit key-dependent permutation function, consists of four main blocks; two g-Function, PHT, 8-bit rotate left block for the R, and two 32-bit adders modulo 2^{32}. There are three inputs to the F-function R_0, R_1, and r the round number. The g-function is the most important block in Twofish algorithm, each input byte is processed through its key-dependent bijective S-boxes; subsequently, the outputs are formed as a vector of length 4 over $GF(2^8)$, and multiplied by 4×4 MDS

matrix. The output of MDS matrix is a vector of 32-bit word.

The g function is composed of four key dependent S-boxes followed by MDS. T_0 and T_1 are the outputs of g-functions (7), and the input to PHT; the two modulo 2^{32} adders.

$$T_0 = g(R_0)$$
$$T_1 = g\left(ROL\left(R_0, 8\right)\right) \quad (7)$$

where ROL is the function that rotates their first argument (a 32-bit word) left by the number of bits indicated by their second argument. The results of F-function are 32-bit F_0 and F_1 (8).

$$F_0 = (T_0 + T_1 + k_{2r+8}) \, mod \, 2^{32}$$
$$F_1 = (T_0 + 2T_1 + k_{2r+9}) \, mod \, 2^{32} \quad (8)$$

The Twofish key schedule provides 40 expanded keys $k_0, \ldots, k_{39}$. Eight expanded keys for whitening process and 32 expanded keys for 16 rounds, in addition to two subkeys S_0 and S_1 as the inputs to the g-function. The Twofish accepts keys of length 128-bit, 192-bit, and 256-bit.

3. MAXIMUM DISTANCE SEPERABLE (MDS)

An MDS matrix is a matrix that describes a function with certain diffusion properties that have useful cryptographic applications. The codes are provided specifically over finite fields with powerful algorithms for encoding and decoding.[3,16]. MDS represents the main diffusion procedure for the S-box output. The 32-bit word input to the g-function is divided into four bytes; each byte represents the input to its corresponding bijective S-box. The output of the S-box is a vector of four bytes $Y_0, \ldots, Y_3$, the output is multiplied by 4 × 4 MDS matrix over $GF(2^8)$; as shown in (9); [17].

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} \cdot & \cdots & \cdot \\ \vdots & MDS & \vdots \\ \cdot & \cdots & \cdot \end{pmatrix} \begin{pmatrix} Y_0 \\ Y1 \\ Y_2 \\ Y_3 \end{pmatrix} \quad (9)$$

A primitive polynomial with degree 8 over $GF(2)$ is used; the polynomial and the hexadecimal representation are shown in (10).

$$p(x) = x^8 + x^6 + x^5 + x^3 + 1 = (169)_{16} \quad (10)$$

The MDS matrix is:

$$MDS = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \quad (11)$$

The diffusion mechanism of MDS guaranteed that; any change in input byte will followed by changing the output bytes.

4. IMPLEMENTATION OF MDS MODULE

This section presents the multiplication of MDS matrix in $GF(2^8)$, converting its polynomial form into binary form, and minimizing the cost of the hardware implementation of the proposed design.

4.1 CONVERTING MDS MATRIX INTO BINARY FORM

The MDS module is repeated four times in the same round, twice for the g-function and twice for the h-function in key schedule; as shown in Fig. 2, [3]. The h function generates the subkeys in Twofish, which can be presented as four key-dependent S-boxes followed by an MDS matrix [3]. Therefore, the main purpose of this article is to provide an improved hardware design of the MDS module for Twofish algorithm without increasing its time delay. Xilinx FPGA used to implement the proposed design. The following equation (12) of the MDS matrix output can be deduced from (9) and (11).

$$z_0 = Y_0 + Y_1.EF + Y_2.5B + Y_3.5B$$
$$z_1 = Y_0.5B + Y_1.EF + Y_2.EF + Y_3$$
$$z_2 = Y_0.EF + Y_1.5B + Y_2 + Y_3.EF$$
$$z_3 = Y_0.EF + Y_1 + Y_2.EF + Y_3.5B \quad (12)$$

Every single multiplication operation in (12) is performed using polynomial mathematics over $GF(2^8)$, with the primitive polynomial $p(x) = 169_{16}$, as shown in (10). In the polynomial arithmetic the addition is just XOR operation. Referring to (10) and (12), there are mainly two modular multiplication operations (13).

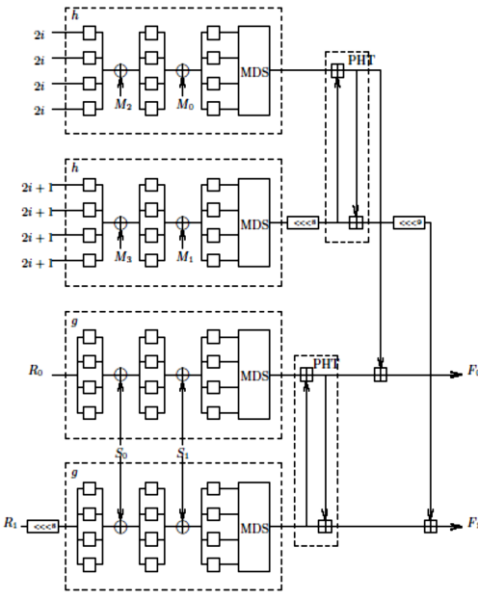

Fig. 2 MDS module in one round

$$A = (Y_j . 5B_{16}) mod\ 169_{16}$$

$$B = (Y_j . EF_{16}) mod\ 169_{16} \quad j = 0, \dots, 3 \quad (13)$$

Y_j is one input byte and the only unknown in (13). Therefore, MATLAB was used to deduce the lookup tables of 256-bytes for each modular multiplication operation of A and B. The outputs of the two lookup tables are minimized, and the resulted outputs are $A = a_7, a_6, \dots, a_0$, and $B = b_7, \dots, b_0$. The algorithms for modules A and B are demonstrated in Algorithm-1 and Algorithm-2 respectively. The four-input bytes of the MDS module are represented by: $Y_j = y_{7,j}, y_{6,j}, \dots, y_{0,j}$.

Algorithm-1 MDS A-Module

Input: $Y_j = y_{7,j}, y_{6,j}, \dots, y_{0,j}$.[**one − Byte**].
Output: **A** $= (Y_j . 5B_{16}) mod\ 169_{16}$.[**one − Byte**].
--LUT $a_0 - a_2$
$a_0 = y_{0,j} \oplus y_{2,j}$;
$a_2 = y_{1,j} \oplus y_{2,j} \oplus y_{4,j}$;
--LUT $a_1 - a_3$
$a_1 = y_{0,j} \oplus y_{1,j} \oplus y_{3,j}$;
$a_3 = y_{0,j} \oplus y_{3,j} \oplus y_{5,j}$;
--LUT $a_4 - a_6$
$a_4 = y_{0,j} \oplus y_{1,j} \oplus y_{4,j} \oplus y_{6,j}$;
$a_6 = y_{0,j} \oplus y_{6,j}$;
--LUT $a_5 - a_7$
$a_5 = y_{1,j} \oplus y_{5,j} \oplus y_{7,j}$;
$a_7 = y_{1,j} \oplus y_{7,j}$;
Return ($A = a_7$ & a_6& a_5 & a_4 & a_3 & a_2& a_1 & a_0**)**

Algorithm-2 MDS B-Module

Input: $Y_i = y_{7,j}, y_{6,j}, \dots, y_{0,j}$.[**one − Byte**]
Output: **B** $= (Y_j . EF_{16}) mod\ 169_{16}$.[**one − Byte**]
--LUT $b_0 - b_7$
$b_0 = y_{0,j} \oplus y_{1,j} \oplus y_{2,j}$;
$b_7 = y_{0,j} \oplus y_{1,j} \oplus y_{7,j}$;
--LUT $b_1 - b_2$
$b_1 = y_{0,j} \oplus y_{1,j} \oplus y_{2,j} \oplus y_{3,j}$;
$b_2 = y_{0,j} \oplus y_{1,j} \oplus y_{2,j} \oplus y_{3,j} \oplus y_{4,j}$;
--LUT b_3
$b_3 = y_{0,j} \oplus y_{3,j} \oplus y_{4,j} \oplus y_{5,j}$;
--LUT b_4
$b_4 = y_{1,j} \oplus y_{4,j} \oplus y_{5,j} \oplus y_{6,j}$;
--LUT $b_5 - b_6$
$b_5 = y_{0,j} \oplus y_{1,j} \oplus y_{5,j} \oplus y_{6,j} \oplus y_{7,j}$;
$b_6 = y_{0,j} \oplus y_{6,j} \oplus y_{7,j}$;
Return (B= b_7 & b_6& b_5 & b_4 & b_3 & b_2& b_1 & b_0**)**

4.2 Optimization of MDS sub-modules

Some techniques have been applied to optimize the area of the proposed design. Referring to Algorithm-1 and Algorithm-2, there are 14 XOR operations are required in A module, and there are 23 XOR operations are required in B module. Each XOR operation needs one LUT, then, there are 37 LUTs are required. Therefore, more optimization is needed. The cost of the implementation can be decreased by decreasing the number of LUTs, the available primitives of the FPGA can be utilized. Fig. 3 shows the LUT6 and $LUT6-2$ block diagrams, it is available in Xilinx 7-series and also in Spartan-6 FPGA [18]. Thus, the available primitives of the Xilinx FPGA have been used to control the exact placement of individual LUTs.

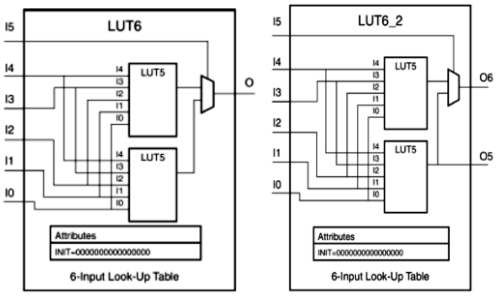

Fig. 3 LUT6 and LUT6_2

$LUT6-2$ can produce two bits of sub-module output instead of one in regular LUT, each two semi-similar output bits of sub-module have been utilized to be the two outputs of single $LUT6-2$. Therefore, the A-module has become only 4 LUTs; instead of 14 LUTs. The B-module has become only 5 LUTs; instead of 23 LUTs. The output function for each LUT of A-module and B-module is shown in Algorithm-1 and Algorithm-2 respectively.

4.3 Implementation of the proposed design on FPGA

The proposed hardware design of MDS matrix has been described using Schematic design tool in Project Navigator of Xilinx ISE-14.4; utilizing the primitives $LUT6$ and $LUT6\_2$ of the Virtex-7 FPGA. Referring to (12), twelve A and B sub-modules are needed to construct the MDS module, nonetheless, only four A sub-modules and four B sub-modules are required in the proposed design. The algorithm of the proposed MDS module is shown in Algorithm-3. The implementations of A-module and B-module are shown in Fig. 4 and Fig. 5 respectively.

Algorithm-3 Modified MDS Module

Input: Y_0, Y_1, Y_2, Y_3 .[**Four − Bytes**]
Output: MDS_OUT [**Four − Bytes**]

1- $\boldsymbol{A_0 = Y_0 \rightarrow A;}$
$\boldsymbol{A_1 = Y_1 \rightarrow A;}$
$\boldsymbol{A_2 = Y_2 \rightarrow A;}$
$\boldsymbol{A_3 = Y_3 \rightarrow A;}$
$\boldsymbol{B_0 = Y_0 \rightarrow B;}$
$\boldsymbol{B_1 = Y_1 \rightarrow B;}$
$\boldsymbol{B_2 = Y_2 \rightarrow B;}$
$\boldsymbol{B_3 = Y_3 \rightarrow B;}$

2- $\boldsymbol{Z_0} = Y_0 \oplus B_1 \oplus A_2 \oplus A_3;$
$\boldsymbol{Z_1} = Y_3 \oplus B_1 \oplus A_0 \oplus B_2;$
$\boldsymbol{Z_2} = Y_2 \oplus B_0 \oplus A_1 \oplus B_3;$
$\boldsymbol{Z_3} = Y_1 \oplus B_0 \oplus A_3 \oplus B_2;$

Return ($MDS\_OUT = \boldsymbol{Z_3}\,\&\boldsymbol{Z_2}\&\boldsymbol{Z_1}\&\boldsymbol{Z_0}$)

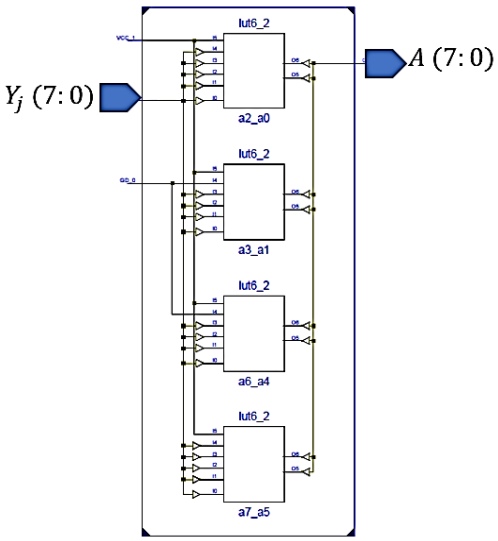

Fig. 4 RTL schematic of A- module

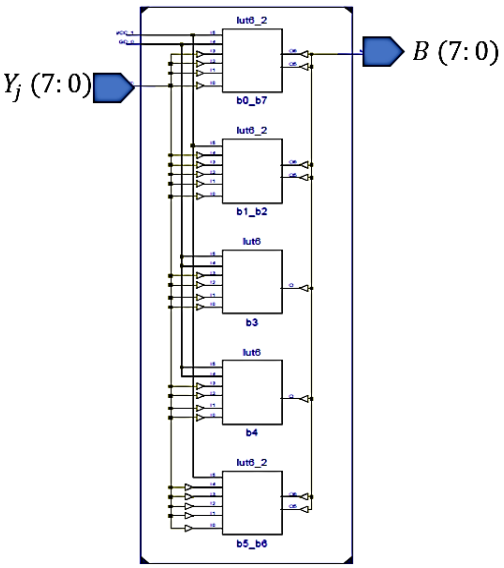

Fig. 5 RTL schematic of B-module

The pipeline and parallel techniques are used to improve the time delay and the speed performance of the proposed design with the minimum number of gate levels. Referring to Fig. 4 and Fig. 5 of A-module and B-module, the internal LUT of each sub-module is processed in parallel to decrease the delay time of the proposed design. Similarly, all sub-modules A and B are processed in parallel in the main MDS module. The pipeline technique is utilized in dividing the hole process into two stages, the first stage is calculating the result of all sub-modules, the second stage is the Addition process. The pipeline is utilized also internally for each sub-module in dividing the input byte, Y_j, into bits, $y_{7,j}, \ldots, y_{0,j}$, and processing it simultaneously to decrease the output delay time of each sub-module. In order to add the result of the sub-modules; 32 LUTs as XOR logics have been used, Algorith-3. Fig. 6 shows the top-level block of the MDS module proposed design, and the RTL schematic is shown in Fig. 7.

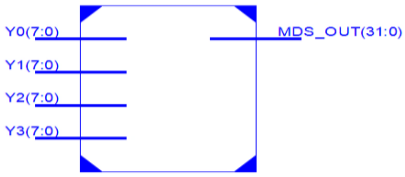

Fig. 6 Top-level block of MDS module

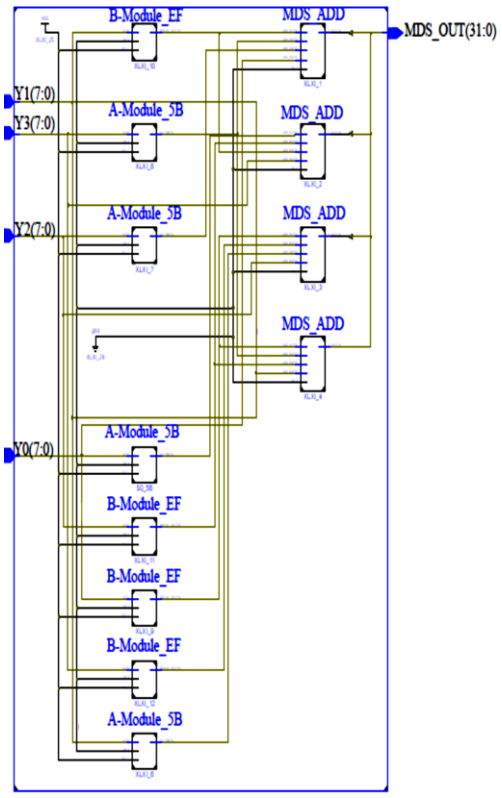

Fig. 7 RTL schematic of Proposed MDS module

5. TESTS AND RESULTS

5.1 SIMULATION RESULTS

The verification of the proposed MDS design was proved by testing the g-function, Fig. 8. The g-function is the heart of Twofish. The input R_0 is divided into four bytes, each byte runs through its own key-dependent S-box. The S-boxes are described using VHDL and appended to the MDS proposed design to construct the g-function. Each S- box is bijective; takes one byte of input and produces one byte of output. The four bytes,$Y_1 \dots Y_4$, are multiplied by the proposed design of MDS matrix. The resulting vector T_0 is interpreted as a 32-bit word which is the result of g-function.

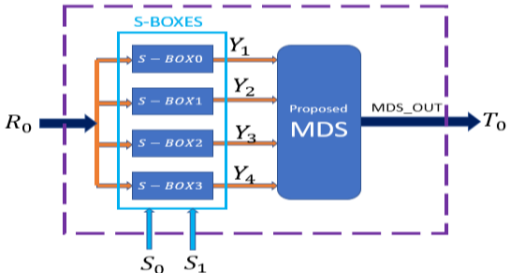

Fig. 8 g-Function of the Twofish Algorithm

The simulation tool is ModelSim-SE-10.5. Table I introduces four different test vectors used in proving the validity of the proposed design [3]. S_0 and S_1 are forced to "00000000_{16}". As shown in Fig. 9; the output results T_0 are typical as its corresponding input R_0 in Table I.

The simulation results in Fig. 9 presents the output result of inputting four different inputs to the g - function. There is a simultaneous output R_0 for each input T_0. As a result, the proposed design can accept any change in input and computes its correct corresponding output, that proves the correctness of the functionality of the proposed design of the MDS matrix.

TABLE I. TEST VECTORS OF G-FUNCTION

32-bit Input R_0	32-bit Output T_0
$52C54DDE_{16}$	$C06D4949_{16}$
$C38DCAA4_{16}$	$7C4536B9_{16}$
$55A538DE_{16}$	$60DAC1A4_{16}$
$899063BD_{16}$	$607AAEAD_{16}$

5.2 SYNTHESIS AND COMPARISON RESULTS

The hardware implementation of Twofish algorithm was demonstrated in [16,19]. The implementation of the Twofish encryption algorithm on FPGA has been treated in a number of articles in order to increase the speed of their proposed designs; decrease the time delay needed to process the input signals, or to decrease the area of its modules [13,14,20–24]. There is a tradeoff in the hardware implementation between the speed and area. The aim of the proposed design is not only to minimize the occupied area of the hardware implementation; but also, to increase the speed, and to enforce the maximum combinational path delay to be stable. The speed performance has been improved by using the pipeline and parallel techniques for the proposed design of the Twofish-MDS module. To minimize the area of the proposed module, the $LUT6$ and $LUT6-2$ primitives of the 25 nm Xilinx FPGA, the 7-Series, have been utilized. The targeted device for the implementation is Virtex-7 xc7vx330t-3ffg1157. Table II shows the comparison results of the hardware implementation of the proposed design and the other related works of MDS matrix over $GF(2^8)$. The comparison in terms of the occupied area measured in number of LUTs, the delay in ns, the maximum frequency in MHz, and the throughput in Gbps. For fairness in comparison, it has been established among the related works with same 4×4 MDS matrix, the same elements, and over the same Finite Field.

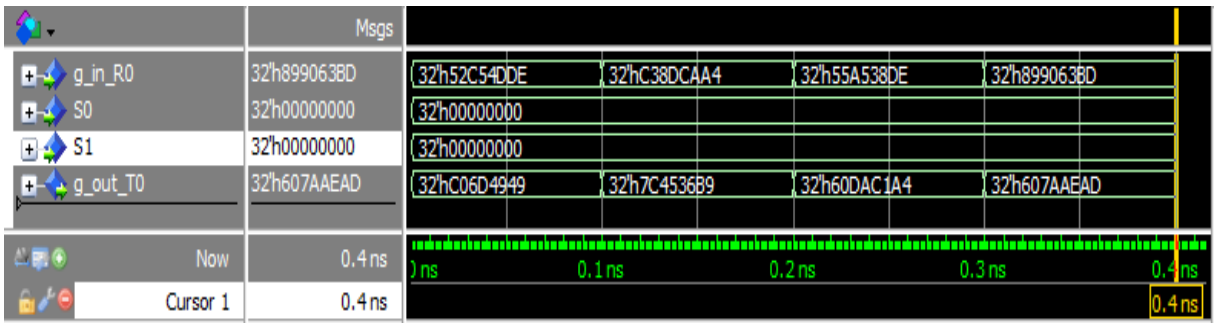

Fig. 9 simulation results of g-function block inputs and outputs

TABLE II. COMPARISON OF THE SYNTHESIS RESULTS

MDS Design	Finite Field	Area LUT	Delay ns	Max. Frequency MHz	Throughput Gbps	FPGA
[22]	$GF(2^8)/169_{16}$	102	9.615	104	3.328	Xilinx
[25]	$GF(2^8)/169_{16}$	112	9.5	105.163	3.365	Xilinx
[26]	$GF(2^8)/169_{16}$	185	4	250	8	Altera
[21]	$GF(2^8)/169_{16}$	112	-	-	-	Xilinx
[20]	$GF(2^8)/169_{16}$	-	13.706	72.960	2.334	Xilinx
Proposed Design	$GF(2^8)/169_{16}$	68	1.511	661.813	21.178	Xilinx

Referring to Table II, the hardware area of the proposed design on FPGA is the lightest occupied area among other related works. The area of the proposed design is only 68 LUTs, this number of LUTs is exactly as targeted to be in the proposed design. The cost of the area implementation of the proposed design is less than 66% of the area of the other comparative works, and the maximum path time delay is also the least among others. The proposed design Maximum frequency is 661.318 MHz with throughput of 21.178 Gbps.

6. CONCLUSION

This paper presents an efficient implementation of the MDS matrix. The MDS matrix was divided into modules internally constructed in parallel utilizing the pipeline technique to decrease the delay of the proposed design and enhance its maximum frequency. Xilinx primitives $LUT6$ and $LUT6-2$ were utilized in the proposed design to control exactly the placement of components and to guarantee the minimum occupation area. The verification of the proposed design was proved using ModelSim tool. The proposed design was implemented on Xilinx Virtex-7 FPGA. The area of the proposed design is only 68 LUTs. The synthesizes results comparing with other related designs show that the proposed design has the least area, furthermore the smallest maximum path time delay; with maximum frequency of 661.318 MHz, and high throughput of 21.178 Gbps. The proposed design is a suitable candidate for lightweight cryptographic implementation.

REFERENCES

[1] C. Paar, J. Pelzl, Understanding Cryptography, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. https://doi.org/10.1007/978-3-642-04101-3.

[2] J. Daemen, V. Rijmen, The Design of Rijndael, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. https://doi.org/10.1007/978-3-662-04722-4.

[3] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, Twofish : A 128-Bit Block Cipher, NIST AES Propos. 15 (1998).

[4] J.P. Pedersen, C. Dahl, Classification of pseudo-cyclic MDS codes, IEEE Trans. Inf. Theory. 37 (1991) 365–370. https://doi.org/10.1109/18.75254.

[5] J.I. Kokkala, D.S. Krotov, P.R.J. Ostergard, On the Classification of MDS Codes, IEEE Trans. Inf. Theory. 61 (2015) 6485–6492. https://doi.org/10.1109/TIT.2015.2488659.

[6] R.M. Roth, A. Lempel, A Construction of Non-Reed-Solomon Type MDS Codes, IEEE Trans. Inf. Theory. 35 (1989) 655–657. https://doi.org/10.1109/18.30988.

[7] M. Sajadieh, M. Dakhilalian, H. Mala, P. Sepehrdad, Recursive diffusion layers for block ciphers and hash functions, Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 7549 LNCS (2012) 385–401. https://doi.org/10.1007/978-3-642-34047-5_22.

[8] K.C. Gupta, S.K. Pandey, A. Venkateswarlu, Towards a general construction of recursive MDS diffusion layers, Des. Codes, Cryptogr. 82 (2017) 179–195. https://doi.org/10.1007/s10623-016-0261-0.

[9] C. Chen, S.J. Lin, N. Yu, Irregular MDS Array Codes with Fewer Parity Symbols, IEEE Commun. Lett. 23 (2019) 1909–1912. https://doi.org/10.1109/LCOMM.2019.2937778.

[10] D. Yin, Y. Gao, A new construction of lightweight MDS matrices, 2017 3rd IEEE Int. Conf. Comput. Commun. ICCC 2017. 2018-Janua (2018) 2560–2563. https://doi.org/10.1109/CompComm.2017.8322997.

[11] C. Beierle, T. Kranz, G. Leander, Lightweight Multiplication in GF(2^n) with Applications to MDS Matrices, in: Springer Berlin Heidelberg, 2016: pp. 625–653. https://doi.org/10.1007/978-3-662-53018-4_23.

[12] S.M. Sim, K. Khoo, F. Oggier, T. Peyrin, Lightweight MDS involution matrices, Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 9054 (2015) 471–493. https://doi.org/10.1007/978-3-662-48116-5_23.

[13] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, L. Wingers, The SIMON and SPECK lightweight block cIPhers, Proc. - Des. Autom. Conf. 2015-July (2015). https://doi.org/10.1145/2744769.2747946.

[14] J. Guo, T. Peyrin, A. Poschmann, M. Robshaw, The LED block cipher, Lect. Notes Comput. Sci.

(Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 6917 LNCS (2011) 326–341. https://doi.org/10.1007/978-3-642-23951-9_22.

[15] Xilinx, Virtex-II Libraries Guide for Schematic Designs, UG616(v14. (2013) 631. www.Xilinx.com.

[16] J. Solomon, I. V Be, A Study of Twofish Algorithm, Int. J. Eng. Dev. Res. 4 (2016) 2321–9939. https://www.ijedr.org/papers/IJEDR1602023.pdf.

[17] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, Twofish: A 128-bit block cipher, NIST AES Propos. 15 (1998).

[18] X. Fpgas, Xilinx 7 Series FPGAs :, 405 (2012) 1–9.

[19] Yeong-Kang Lai, Liang-Gee Chen, Jian-Yi Lai, Tai-Ming Parng, VLSI architecture design and implementation for TWOFISH block cipher, 2002 IEEE Int. Symp. Circuits Syst. Proc. (Cat. No.02CH37353). (2002) II-356-II–359. https://doi.org/10.1109/ISCAS.2002.1010998.

[20] P. Gehlot, S. R. Biradar, B. P. Singh, Implementation of Modified Twofish Algorithm using 128 and 192-bit keys on VHDL, Int. J. Comput. Appl. 70 (2013) 36–42. https://doi.org/10.5120/12024-8087.

[21] A. Singh, Study of MDS Matrix used in Twofish AES (Advanced Encryption Standard) Algorithm and its VHDL Implementation, 1997.

[22] A. Singh, FPGA Implementation and Analysis of DES and TWOFISH Encryption Algorithms, THAPAR, 2010.

[23] D. Smekal, J. Hajny, Z. Martinasek, Hardware-Accelerated Twofish Core for FPGA, 2018 41st Int. Conf. Telecommun. Signal Process. TSP 2018. (2018) 1–5. https://doi.org/10.1109/TSP.2018.8441386.

[24] C. De Cannière, O. Dunkelman, M. Knežević, KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers, Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 5747 LNCS (2009) 272–288. https://doi.org/10.1007/978-3-642-04138-9_20.

[25] M. De Clercq, V. Levesque, A VHDL Implementation of the Twofish Block Cipher, (1999).

[26] O. De Souza Martins Gomes, R.L. Moreno, A compact 128-bits symmetric cryptography hardware module, Proc. 2016 8th Int. Conf. Inf. Technol. Electr. Eng. Empower. Technol. Better Futur. ICITEE 2016. (2017). https://doi.org/10.1109/ICITEED.2016.7863244.

تنفيذ فعال لمصفوفة المسافة الفاصلة القصوى على شريحة المصفوفات القابلة للبرمجة حقليا

الملخص

في الآونة الحديثة، فإن دراسة "مصفوفة المسافة الفاصلة القصوى" MDS أصبح موضوع ذا اهتمام. MDS هي أهم مكون في طبقة الانتشار في التشفير الكتلي. هذا المقال يقدم مصفوفة MDS 4×4 $GF(2^8)$ مثالية ذات حجم صغير. تم تطبيق التصميم المقترح على مصفوفات البوابات المبرمجة حقليا. التنفيذ للتصميم المقترح يستهدف Virtex-7باستخدام برامج شركة Xilinx. تم استخدام تصميمات LUT6 و LUT6-2 الخاصة بشركة Xilinx للتحكم بشكل دقيق في موقع كل مكون من التصميم المقترح للحصول على أصغر مساحة ممكنة. تم استخدام طريقة التنفيذ بالتجزئة والتصميم المتوازي لتحسين السرعة. تم التحقق من الأداء الخاص بالتصميم المقترح باستخدام برنامج Modelsim. إن نتائج التوليف وضحت أن المساحة المشغولة للتصميم المقترح يقدم نتائج تنافسية في المساحة والإنتاجية. بالمقارنة بالتصميمات المماثلة السابقة، التصميم المقترح يشغل أقل مساحة وأقل زمن للإنتاجية. المساحة المشغولة للتصميم المقترح قلت بشكل ملحوظ، LUT ٦٨ ، مع معدل إنتاجية 21.178جيجا بت في الثانية الواحدة. التصميم المقترح مرشح مناسب لتطبيقات التشفير ذات الحجم الصغير.