
Study of MDS Matrix used in Twofish AES (Advanced Encryption Standard) Algorithm and its VHDL Implementation

Aarti Singh
1/EC/97

ACKNOWLEDGEMENT

I carried out this Project at CEERI (Central Electronic Engineering Research Institute), Delhi. I am thankful to my advisor and guide Dr. Amrik Singh, without whose support, this project would not have been possible. He introduced me to this challenging and exciting topic and guided me throughout. I am also thankful to Dr. S.P.S. Saini and Mr. Brijesh Joshi for their useful assistance and comments. Also I take this opportunity to express my regards and sincere thanks to Dr. H.R. Singh who assigned me to this project. I am also indebted to Dr. S.S. Aggarwal, Head CEERI (D) for providing me the opportunity to work in their reputed institute and extend some contribution to it. Last but not the least, I thank all CEERI staff for helping me out with whatever I needed.

Aarti Singh

INTRODUCTION

Data Encryption has assumed a significant role in today's world. With the ever-increasing need to ensure security of data during communication, the Encryption Algorithms need to be strengthened. The advances in crypt-analysis further underscore the importance of having secure Algorithms. With this aim in mind, the National Institute of Standards and Technology (NIST), responsible for maintaining encryption standards in the U.S., gave a call for AES or the Advanced Encryption Standard. Till date there are 5 contenders that have emerged successful in the 3 round-conferences.

The five finalists:

1. Mars
2. RC6
3. Rijndael
4. Serpent
5. Twofish

This Project is an introduction to the AES candidate – **Twofish**. Twofish is named such since it has two main diffusion elements, they are the MDS (Maximum Distance Separable) matrix and the PHT (Pseudo Hadamard Transform). (Moreover it is customary to name ciphers after sea-creatures and Twofish is the successor of Blowfish.) This Project undertakes a deeper analysis of its most significant diffusion element, the MDS matrix. A VHDL model for the hardware implementation of the matrix is also proposed.

AES (Advanced Encryption Standard):

AES is the next-generation of Encryption Standard. It is basically an improvement over the DES (Data Encryption Standard). One of its prerequisite is a 128-bit block cipher to replace the current 64-bit ciphers.

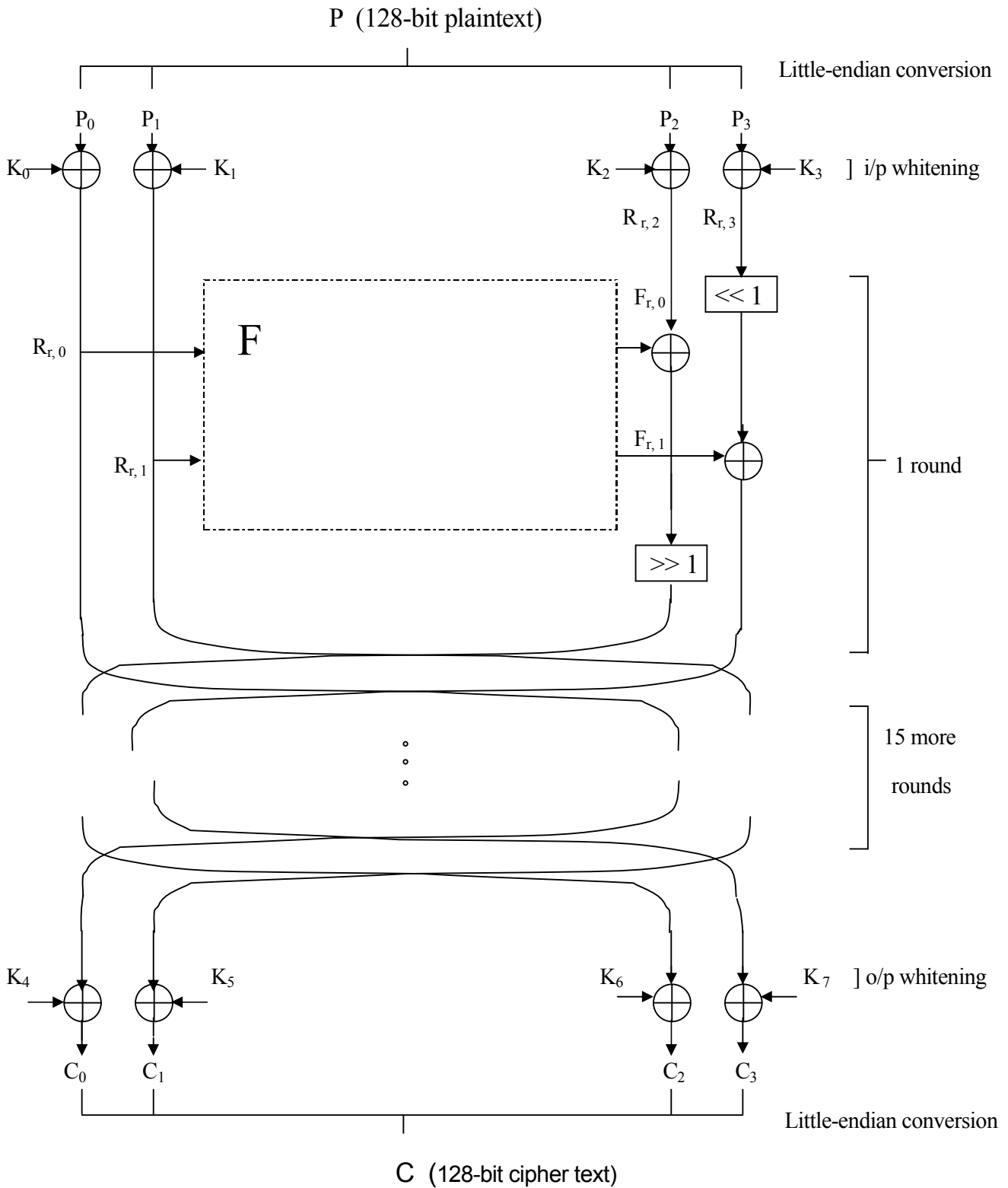
Need for AES/a 128-bit block cipher:

1. The key used in various prevalent 64-bit ciphers like DES is too short for acceptable commercial security requirements of today.
2. Recent advances in distributed key-search technique have made the earlier ciphers prone to cryptanalytic attacks.
3. Ciphers like Triple-DES, which despite being 64-bit offer greater number of rounds to meet the required security, are too slow.
4. Another disadvantage of 64-bit ciphers is that the 64-bit block length is open to attacks when large amount of data are encrypted under the same key.

Twofish Characteristics:

- A 128-bit block cipher.
 - Key lengths of 128 bits, 192 bits and 256 bits.
 - No weak keys.
 - Efficiency and speed on both, software (PIII pro etc.) and hardware (smartcard etc.) platforms.
 - Flexible design e.g.
 - Accepts additional key lengths (all key lengths up to 256 bits with leading zeros filled in).
 - Implementable on variety of platforms.
 - Suitable for stream-ciphering also.
 - Also usable for hash function and MAC (Message Authentication Codes)
-

Twofish Algorithm Block Structure:



Twofish Algorithm steps: (Overview)

(1) Plaintext (128-bit) $\xrightarrow{\text{split using}}$ P_0, \dots, P_3 (32-bits each)

P_0, \dots, P_{15} little-endian conversion

according to:

$$P_i = \sum_{j=0}^3 p_{(4i+j)} \cdot 2^{8j} \quad i = 0, \dots, 3$$

(2) Input Whitening – XORing with sub-keys

$$R_{0,i} = P_i \oplus K_i \quad i = 0, \dots, 3$$

(3) 16 Iterations of round function

A balanced Feistel network constitutes the 16 iterations. A Feistel network transforms any function F into a permutation. In a Feistel cipher, the round function consists of taking one part of the data being encrypted, feeding it into some key-dependent function F , and then XORing the result into another part of the block. If the two parts are each half a block, then the cipher is “balanced”.

For r^{th} iteration,

$$(F_{r,0}, F_{r,1}) = \mathbf{F}(R_{r,0}, R_{r,1}, r) \quad r = 0, \dots, 15$$

r is used to select the appropriate sub-key for the F function.

Finally, after swapping

$$R_{r+1,0} = \text{ROR}(R_{r,2} \oplus F_{r,0}, 1)$$

$$R_{r+1,1} = \text{ROL}(R_{r,3}, 1) \oplus F_{r,1}$$

$$R_{r+1,2} = R_{r,0}$$

$$R_{r+1,3} = R_{r,1}$$

(4) Output Whitening – XORing with sub-key

$$C_i = R_{16, (i+2) \bmod 4} \oplus K_{i+4} \quad i = 0, \dots, 3$$

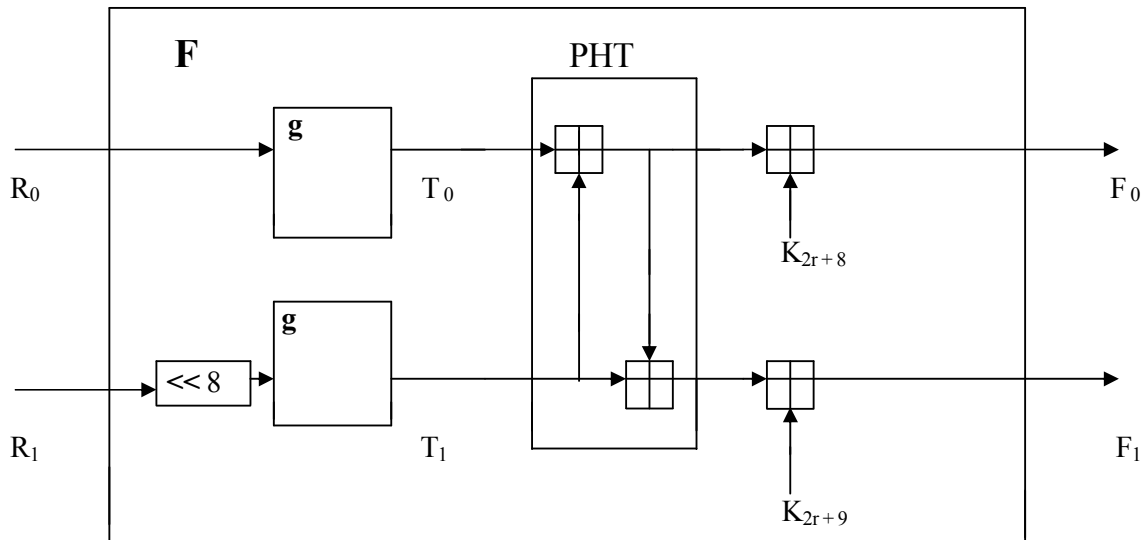
(5) C_0, \dots, C_3 of ciphertext \longrightarrow 128-bit ciphertext, c_i

little-endian conversion

The reverse little-endian proceeds as:

$$c_i = \left[\frac{C_{\lfloor i/4 \rfloor}}{2^{8(i \bmod 4)}} \right] \bmod 2^8 \quad i = 0, \dots, 15$$

Function F – is a key-dependent permutation on 64-bit values. It takes three arguments: two input words R_0 and R_1 , and the round number r used to select the appropriate subkeys and produces F_0 and F_1 as the result.



$$T_0 = g(R_0)$$

$$T_1 = g(\text{ROL}(R_1, 8))$$

$$F_0 = (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$$

$$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}$$

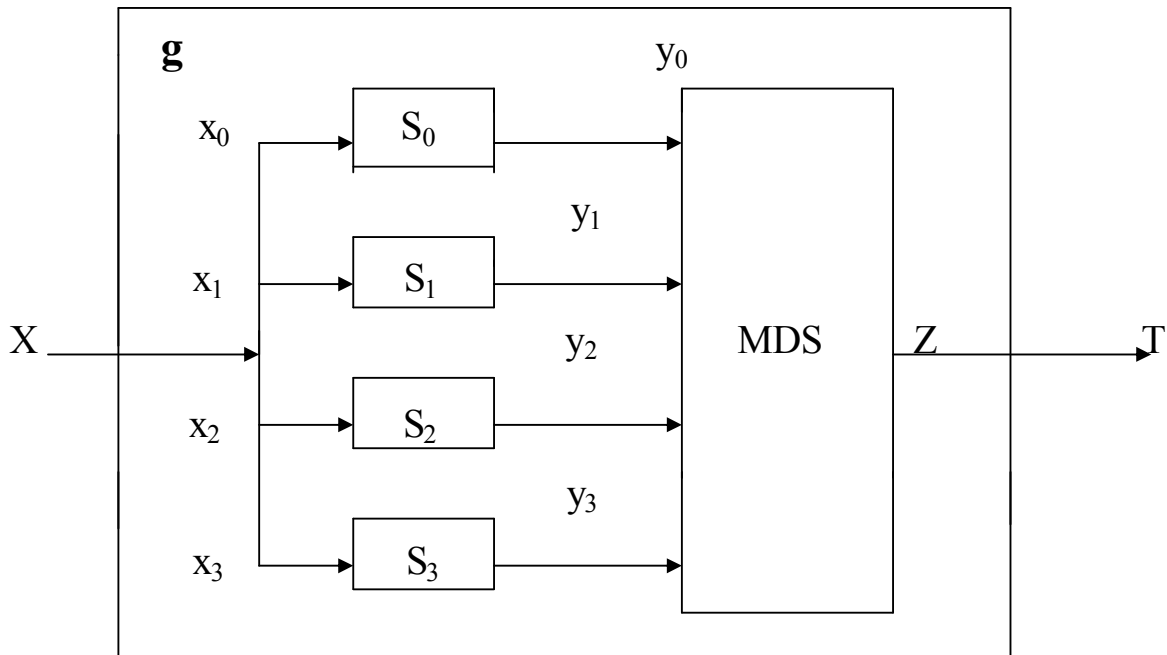
PHT (Pseudo-Hadamard Transform) is a simple mixing operation. This is the second main diffusion element. Twofish uses a 32-bit PHT defined as:

$$a' = (a+b) \bmod 2^{32}$$

$$b' = (a+2b) \bmod 2^{32}$$

where a and b are inputs.

Function g - is the most important part of Twofish Algorithm. It consists of two main elements :- the key-dependent S-boxes and the MDS matrix.



The input word X is split into 4 bytes as

$$X(32\text{-bit}) \longrightarrow x_0, \dots, x_3 \text{ (4 bytes)}$$

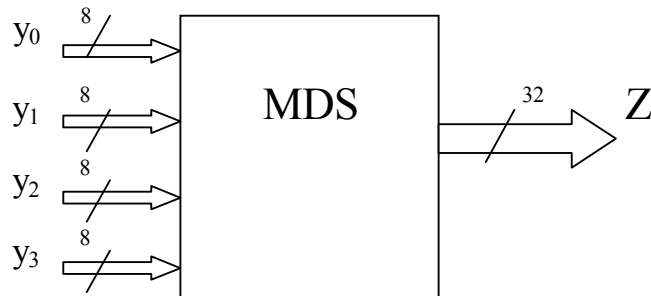
$$x_i = [X/2^{8i}] \bmod 2^8 \quad i = 0, \dots, 3$$

Each byte passes through a key-dependent S-box. An S-box is a non-linear substitution operation. In twofish four S-boxes each constructed of 8-by-8-bit permutations and key material.

$$y_i = S_i[x_i]$$

MDS matrix is the main diffusion element in Twofish. It is described in detail below.

MDS (Maximum Distance Separable) Matrix



$$Z = \sum_{i=0}^3 z_i \cdot 2^{8i}$$

where z_i are the result of multiplication by the MDS matrix:

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad \dots (1)$$

AIM: To understand and implement this 4x4 MDS matrix.

Some Basics from Coding Theory-

Field: A field is a set of elements in which it is possible to add, subtract, multiply and divide (except that division by zero is not defined) and the two operations- addition and multiplication satisfy the commutative, associative and distributive laws.

Galois Field: A field with q elements is called a finite field or Galois field, and is denoted by $GF(q)$. The field with minimum number of elements i.e. two, 0 and 1, is called the binary field, $GF(2)$.

Extended Field: An extended field of degree m over a finite field F has q^m elements if the base field has q elements.
Base field: $GF(q)$ Extended field: $GF(q^m)$

Irreducible Polynomial: A polynomial $p(x)$ that is not divisible by any other polynomial except itself and 1.

Monic Polynomial: A polynomial that has the coefficient of the highest power of x as 1.

Primitive Polynomial: A monic irreducible polynomial.

The field formed by taking polynomials over a finite field F modulo $p(x)$, where $p(x)$ is an irreducible polynomial of degree m , is also an extension field E of degree m over F .

Twofish MDS uses the binary extended field $GF(2^8)[x]/p(x)$ for computations where $p(x)=x^8+x^6+x^5+x^3+1$. ($GF(2^8)[x]$ represents a field where a field element $\sum a_i x^i$, $a_i \in GF(2)$ and $i=0 \dots 7$, is identified with the byte value $\sum a_i 2^i$).

Constructing the Galois field $GF(2^8)$ requires a primitive polynomial $p(x)$ of degree 8 and a primitive element β which is a root of $p(x)$. All elements of a field can be expressed in terms of the primitive element.

Code: A block of k message symbols $u = u_1 u_2 \dots u_n$ is encoded into a codeword $x = x_1 x_2 \dots x_n$ ($u_i, x_i \in GF(2^m)$) where $n \geq k$; these codewords form a code.

Linear Code: A code C is called a linear code if $(v + w)$ is a word in C whenever v and w are in C i.e. it is closed under addition.

Length of a code, n : is the length of all its codewords i.e. the number of elements in each codeword where each element may belong to any field q^m .

Hamming weight, wt: or the weight of a codeword is the number of 1's a codeword has.

Hamming distance, d: or the distance between two codewords is the no. of positions in which the two words disagree.
 $d(v, w) = wt(v+w)$

Dimension of a code, k: A code with 2^k codewords has dimension k i.e. k is the number of elements in any basis for the code.

Information rate or efficiency, R: A (n, k) code has rate k/n since each codeword uses n symbols to send k message symbols.

A linear code is characterized by three parameters-

1. Length of the code, n
2. Dimension of the code, k
3. Distance of the code, d – is the minimum distance between its codewords. For a linear code, since zero is always a codeword, distance of the code is the minimum weight of any non-zero codeword.

The following method of encoding produces a linear (n, k, d) code:

To transmit the k message symbols $\mathbf{u} = u_1, u_2, \dots, u_k$, the message \mathbf{u} is encoded as a codeword, \mathbf{x} of length n.

$$\mathbf{x} = \underbrace{x_1 \dots x_k}_{\text{message symbols, } \mathbf{u}} \quad \underbrace{x_{k+1} \dots x_n}_{\text{n-k check symbols}}$$

The check symbols are chosen as:

$$H \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = 0$$

The $(n-k) \times n$ matrix H is a parity check matrix of the code given by:

$$H = [A | I_{n-k}] \quad \dots(2)$$

where A is some fixed $(n-k) \times k$ matrix and I_{n-k} is the $(n-k) \times (n-k)$ unit matrix.

H has n-k linearly independent rows. This guarantees the unique determination of the n-k check symbols x_{k+1} to x_n . If H has lesser no. of linearly independent

rows, then many such check symbols can be found. So, no. of linearly independent rows is less than or equal to n-k.

$$\text{Equation (2)} \Rightarrow [A | I_{n-k}] \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = 0$$

$$\Rightarrow A \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} + I_{n-k} \begin{pmatrix} x_{k+1} \\ \vdots \\ x_n \end{pmatrix} = 0$$

$$\Rightarrow \text{check symbols} \begin{pmatrix} x_{k+1} \\ \vdots \\ x_n \end{pmatrix} = A \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} = A \begin{pmatrix} u_1 \\ \vdots \\ u_k \end{pmatrix} \quad \dots(3)$$

Generator Matrix, G is defined as the matrix that operates on the message symbols to produce the corresponding codeword.

$$\mathbf{x} = \mathbf{uG}$$

$$\Rightarrow G = [I_k | -A^{tr}]$$

Comparing (3) with (1),

Matrix 'A' used in twofish:

$$\begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix}$$

Message symbols, \mathbf{u} : y_0, y_1, y_2, y_3

Check symbols generated: z_0, z_1, z_2, z_3
which are both elements of the field $GF(2^8)$.

Codeword produced, \mathbf{x} : $z_3 z_2 z_1 z_0 y_3 y_2 y_1 y_0$

Length of codeword, $n = 8$

Dimension of code, $k = 4$.

MDS (Maximum Distance Separable) Codes: A linear (n, k, d) code is said to be a maximum distance separable or MDS code if $d = n - k + 1$.

The given twofish MDS code has $n = 8, k = 4 \Rightarrow d = 8 - 4 + 1 = 5$

A necessary and sufficient condition for a matrix to be MDS i.e. for the matrix to generate a MDS code, is that all possible square sub-matrices of matrix 'A' obtained by discarding rows and columns, are non-singular. Since the Twofish matrix satisfies this condition, it is MDS with $d = 5$.

This guarantees that if a single input changes, all four outputs are bound to change. This is very desirable in an Encryption Algorithm to create maximum randomness and renders the algorithm resistant to easy cryptanalysis.

Other criteria used in choice of Twofish MDS matrix:

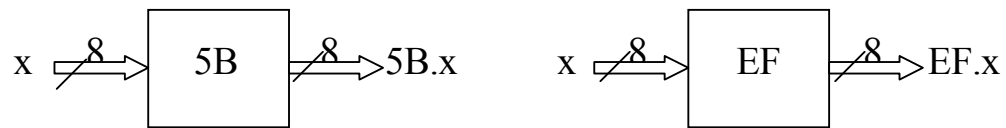
Apart from the non-singular nature of all its sub matrices, which makes it MDS, many other requirements were needed to be fulfilled by this matrix, which helped in the precise determination of its form and elements.

1. Since the primitive polynomial chosen is $p(x) = x^8 + x^6 + x^5 + x^3 + 1$, three elements needed to be chosen which could be expressed in terms of the primitive element β and were at the same time simple enough for multiplication with them to be implemented in hardware with few shifts and XORs. As will be seen later the elements 01, 5B and EF satisfy this. as $5B = 1 + \beta^{-2}$ and $EF = 1 + \beta^{-1} + \beta^{-2}$
 2. During right rotation of the unshifted PHT additions outside the function F, those bytes, which yield a 1 after multiplication with one element, lose this only non-zero bit and the least significant bit of the next higher byte is shifted in. To preserve MDS property even after rotation, the elements are chosen so that for each pair if $a*x=1$, then $b*x$ has the least significant bit set. Obviously, this also holds for any pair a,a . the elements 01, 5B and EF are such elements.
 3. Of all the 4×4 MDS matrices with the elements 01, 5B and EF, only this particular matrix had the property that if a single byte input is changed, not only are all four output bytes changed but also, a maximum Hamming weight difference of 8-bits is observed in the 32-bit output i.e. in all the 32-bits at least 8-bits are sure to be changed. This again guarantees maximum possible randomness produced by a matrix of this size and such simple elements.
-

Hardware Implementation of MDS matrix

In software, the MDS matrix can be implemented easily by a table look-up procedure. However for hardware implementation, simple elements have been chosen which can be implemented using a few shifts and XORs.

The twofish MDS matrix requires just 2 basic blocks - multiplication by 5B and by EF since multiplication by 01 is nothing but the number itself.



The multiplication is carried out mod $p(x)$, where $p(x) = x^8 + x^6 + x^5 + x^3 + 1$ is the primitive element chosen. Now our aim is to simplify these multiplications so that they can be implemented using a few shifts and XORs.

With the earlier defined correspondence, $p(x)$ maps to $2^8 + 2^6 + 2^5 + 2^3 + 2^0$
 $= 1\ 0110\ 1001$
 $= 169\ (H)$

Here + represents addition modulo 2 i.e. XOR.

Also,

$$5B\ (H) = 0101\ 1011 = 2^6 + 2^4 + 2^3 + 2^1 + 2^0$$

$$EF\ (H) = 1110\ 1111 = 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0$$

Therefore,

$$\begin{aligned} 5B.x &= (2^6 + 2^4 + 2^3 + 2^1 + 2^0).x \\ &= 2^0.x + (2^6 + 2^4 + 2^3 + 2^1).x \\ &= x + 2^{-2}.x + (2^6 + 2^4 + 2^3 + 2^1 + 2^{-2}).x \\ &= x + (x \gg 2) + \frac{169(H).x}{4} \\ &= x + (x \gg 2) + \frac{169(H).x_0.2^0}{4} + \frac{169(H).x_1.2^1}{4} + \frac{169(H).x_2.2^2}{4} + \dots + \frac{169(H).x_7.2^7}{4} \end{aligned}$$

Now, 169(H) is a 9-bit number and $\frac{169(H)}{4}$ is a 7-bit number. All multiplications

with 2^2 or more make that term greater than 8-bits. Since multiplication is carried out mod $p(x)$ i.e. mod 169(H), all such terms come out to be zero.

So, finally

$$5B.x = x + (x \gg 2) + \frac{169(H).x_0}{4} + \frac{169(H).x_1}{2} \dots (i)$$

Similarly proceeding for EF.x,

$$\begin{aligned} EF.x &= (2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0).x \\ &= 2^0.x + (2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1).x \\ &= x + 2^{-1}.x + 2^{-2}.x + (2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^{-1} + 2^{-2}).x \\ &= x + (x \gg 1) + (x \gg 2) + \left[\frac{169(H)}{2} + \frac{169(H)}{4} \right].x \\ &\quad \begin{array}{cc} \swarrow & \searrow \\ (2^7 + 2^5 + 2^4 + 2^2 + 2^{-1}) & (2^6 + 2^4 + 2^3 + 2^1 + 2^{-2}) \end{array} \end{aligned}$$

$$= x + (x \gg 1) + (x \gg 2) + \frac{169(H).x_0.2^0}{2} + \frac{169(H).x_0.2^0}{4} + \frac{169(H).x_1.2^1}{4}$$

(retaining only those terms that are ≤ 8 -bits since rest are reduced to zero on taking mod 169(H))

So, finally

$$\begin{aligned} EF.x &= x + (x \gg 1) + (x \gg 2) + \frac{169(H).x_0}{2} + \frac{169(H).x_0}{4} + \frac{169(H).x_1}{2} \dots (ii) \\ &= x + (x \gg 2) + \frac{169(H).x_0}{4} + \frac{169(H).x_1}{2} + (x \gg 1) + \frac{169(H).x_0}{2} \\ &\quad \underbrace{\hspace{10em}}_{5B.x} \end{aligned}$$

Hence the two required multiplications have been reduced to very simple forms. Based on the two equations (i) and (ii), the following VHDL model has been proposed. MDS.vhd is the main file that computes the MDS transformation and it uses the file utils.vhd for declarations. An automatically generated test-bench is also included. The software used is from Aldec: Active-HDL 3.6

```

1:library MDS; use MDS.utils.all;
2:--declaration to include utility package
3:entity MDS is
4:    port(y:in input_data;
5:        Z:out bit_vector(((input_data'length*byte'length)
6:                            -1)downto 0));
7:end entity MDS;
8:
9:architecture behav of MDS is
10:    constant set_up_time:time:=2ns;
11:begin
12:    process is
13:        variable z_temp:input_data; --stores z(0)..z(3)
14:        variable temp:bit_vector(((input_data'length*
15:                                    byte'length)-1)downto 0);
16:        variable op:byte;
17:        variable p,m:integer;
18:    begin
19:        wait for set_up_time;
20:        for i in z_temp'range loop
21:            z_temp(i):=x"00";
22:            for j in y'range loop
23:                case MDS_matrix(i,j) is
24:                    when x"01"=>
25:                        op:=y(j);
26:                    when x"5B"=>
27:                        op:=op1(MDS_matrix(i,j),y(j));
28:                    when x"EF"=>
29:                        op:=op2(MDS_matrix(i,j),y(j));
30:                    when others=>
31:                        report"inconsistency in MDS-matrix"
32:                            severity failure;
33:                end case;
34:                z_temp(i):=op xor z_temp(i);
35:            end loop;
36:        end loop;
37:        p:=Z'high;
38:        for k in z_temp'reverse_range loop
39:            m:=p-(byte'length-1);
40:            temp(p downto m):=z_temp(k);
41:            p:=m-1;
42:        end loop;
43:        Z<=temp; --Z=z(3)z(2)z(1)z(0)
44:        wait on y;
45:    end process;
46:end architecture behav;

```

```

1:--Utility package for MDS
2:package utils is
3:  constant n:natural:=4;
4:  constant GF_base:bit_vector:=x"169";
5:  constant GF_base_2:bit_vector(7 downto 0):=x"B4";
6:    --to make the GF base 169(H)/2, a 8-bit number
7:  subtype byte is bit_vector(7 downto 0);
8:    --little-endian format
9:  type Matrix_nxn is array(0 to n-1,0 to n-1) of byte;
10: type input_data is array(0 to n-1) of byte;
11: constant MDS_matrix :Matrix_nxn:=
12: (0=>(0=>x"01",1=>x"EF",2=>x"5B",3=>x"5B"),
13:  1=>(0=>x"5B",1=>x"EF",2=>x"EF",3=>x"01"),
14:  2=>(0=>x"EF",1=>x"5B",2=>x"01",3=>x"EF"),
15:  3=>(0=>x"EF",1=>x"01",2=>x"EF",3=>x"5B"));
16:    --MDS matrix for Twofish
17:  function op1(element,x:byte) return byte;
18:  function op2(element,x:byte) return byte;
19:end package utils;
20:
21:package body utils is
22:  function op1 (element,x:byte) return byte is  --5B.x
23:    variable result:byte;
24:  begin
25:    result:=(x xor (x srl 2));
26:    if x(0)='1' then
27:      result:=result xor (GF_base_2 srl 1);
28:      --eqv. to GF_base srl 2
29:    end if;
30:    if x(1)='1' then
31:      result:=result xor (GF_base_2);
32:      --eqv. to GF base srl 1
33:    end if;
34:    return result;
35:  end function op1;
36:  function op2(element,x:byte) return byte is  --EF.x
37:    variable result:byte;
38:  begin
39:    result:=(op1(element,x) xor (x srl 1));
40:    if x(0)='1' then
41:      result:=result xor (GF_base_2);
42:      --eqv. to GF_base srl 1
43:    end if;
44:    return result;
45:  end function op2;
46:end package body utils;

```

```

1:--*****
2:--* This file is automatically generated test bench template *
3:--* By ACTIVE-VHDL <TBgen v1.10>. Copyright (C) ALDEC Inc. *
4:--* *
5:--* This file was generated on: 12:11 AM, 7/30/00 *
6:--* Tested entity name: MDS *
7:--* File name contains tested entity: $DSN\src\MDS.vhd *
8:--*****
9:library MDS;
10:use MDS.utils.all;
11: -- Add your library and packages declaration here ...
12:entity mds_tb is
13:end mds_tb;
14:
15:architecture TB_ARCHITECTURE of mds_tb is
16: -- Component declaration of the tested unit
17: component MDS
18: port( y : in input_data;
19: Z : out BIT_VECTOR(((input_data'length*
20: byte'length)-1) downto 0) );
21: end component;
22:
23:-- Stimulus signals- signals mapped to the input and inout
24:-- ports of tested entity
25: signal y : input_data;
26:-- Observed signals-signals mapped to the output ports of
27:--tested entity
28: signal Z : BIT_VECTOR(((input_data'length*
29: byte'length)-1) downto 0);
30: -- Add your code here ...
31:begin
32: -- Unit Under Test port map
33: UUT : MDS
34: port map(y => y,
35: Z => Z );
36: process is
37: begin
38: -- Add your stimulus here ...
39: y<=(0=>(x"01"),1=>(x"23"),2=>(x"45"),3=>(x"67")),
40: (0=>(x"67"),1=>(x"45"),2=>(x"23"),3=>(x"01"))after 200ns;
41: wait;
42: end process;
43:end TB_ARCHITECTURE;
44:
45:configuration TESTBENCH_FOR_MDS of mds_tb is
46: for TB_ARCHITECTURE
47: for UUT : MDS
48: use entity work.MDS (behav);
49: end for;
50: end for;
51:end TESTBENCH_FOR_MDS;

```

An Alternative Circuit Design Approach to Hardware Implementation

From equation (1),

$$z_0 = 01.y_0 + EF.y_1 + 5B.y_2 + 5B.y_3$$

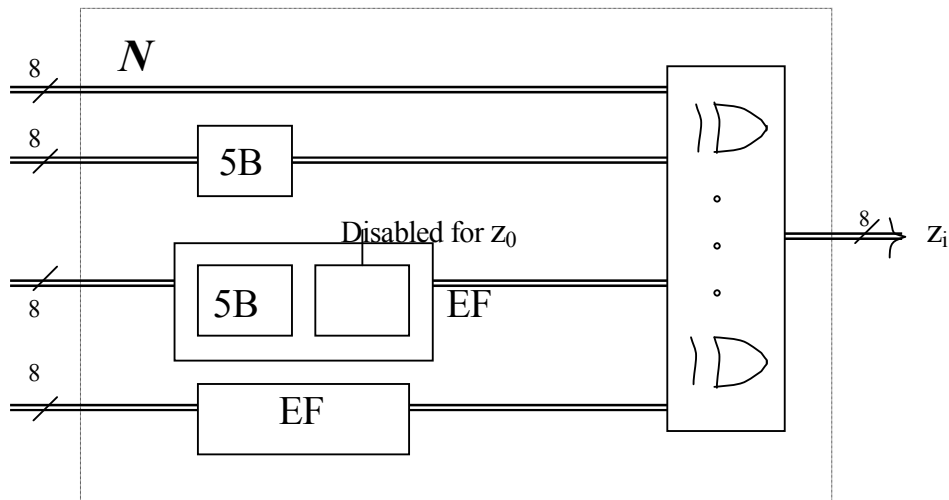
$$z_1 = 5B.y_0 + EF.y_1 + EF.y_2 + 01.y_3$$

$$z_2 = EF.y_0 + 5B.y_1 + 01.y_2 + EF.y_3$$

$$z_3 = EF.y_0 + 01.y_1 + EF.y_2 + 5B.y_3$$

Assuming data-bus to be of 8 bits, we'll generate z_i sequentially and enable 8 bits of the 32-bit final register in turn.

Now, for each z_i four multiplication blocks are required i.e. 01, EF, EF(5B for z_0) and 5B. This can be represented as:-

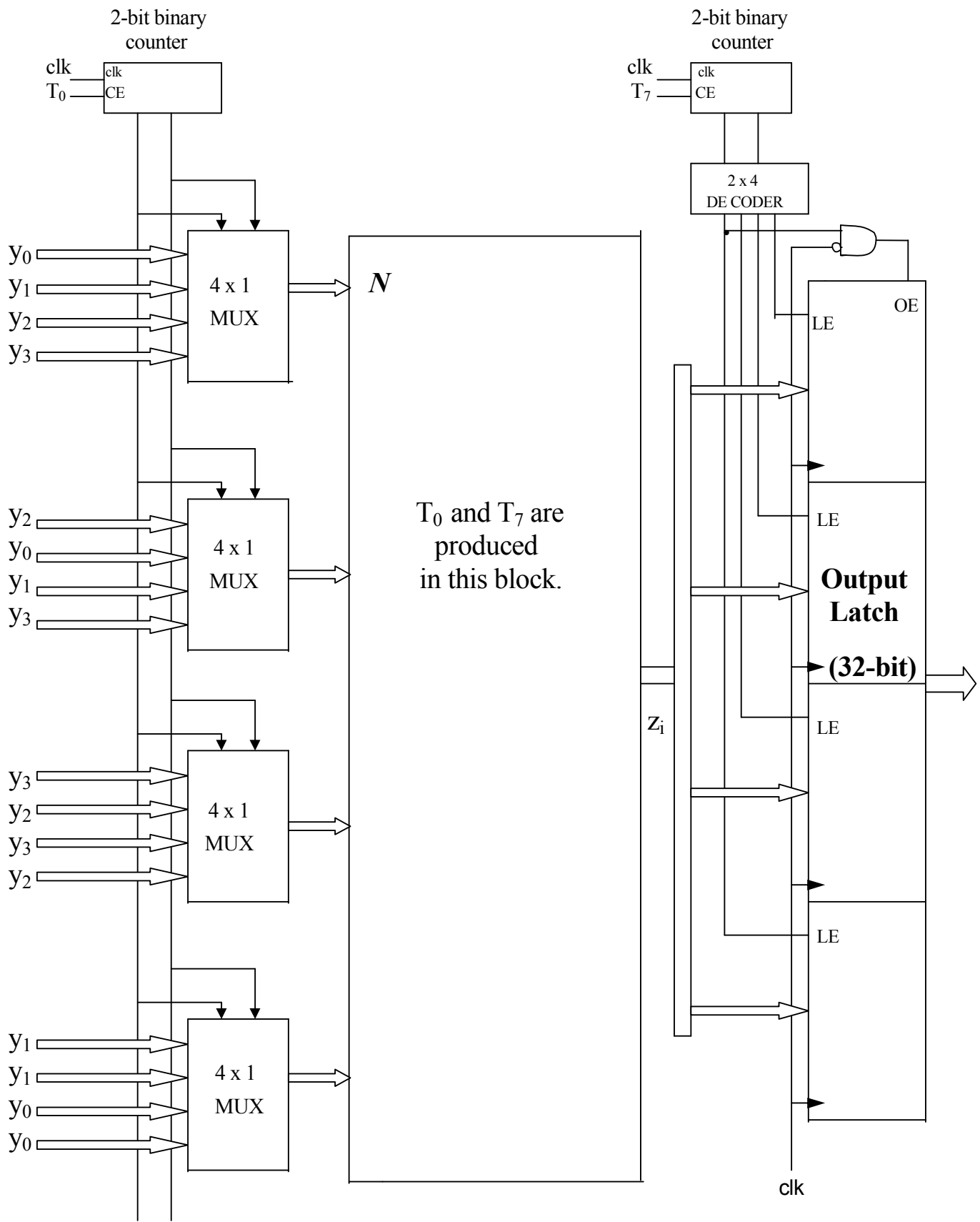


since as shown by eq.(ii), $EF.x$ includes the terms for $5B.x$.

To generate z_i for $i=0..3$, the inputs required for the 4 blocks are as follows:

Inputs for z_i	01	5B	5B/EF	EF
z_0	y_0	y_2	y_3	y_1
z_1	y_3	y_0	y_2	y_1
z_2	y_2	y_1	y_3	y_0
z_3	y_1	y_3	y_2	y_0

This can be achieved using multiplexers to choose the inputs to each block. Let T_0 be the control signal used to load data into the blocks. The overall circuit diagram can now be drawn. (Note the use of the above N block). T_7 is the control signal that outputs each z_i and clocks it into an output 32-bit latch. Thus after all four z_i are latched, the final output Z is available.



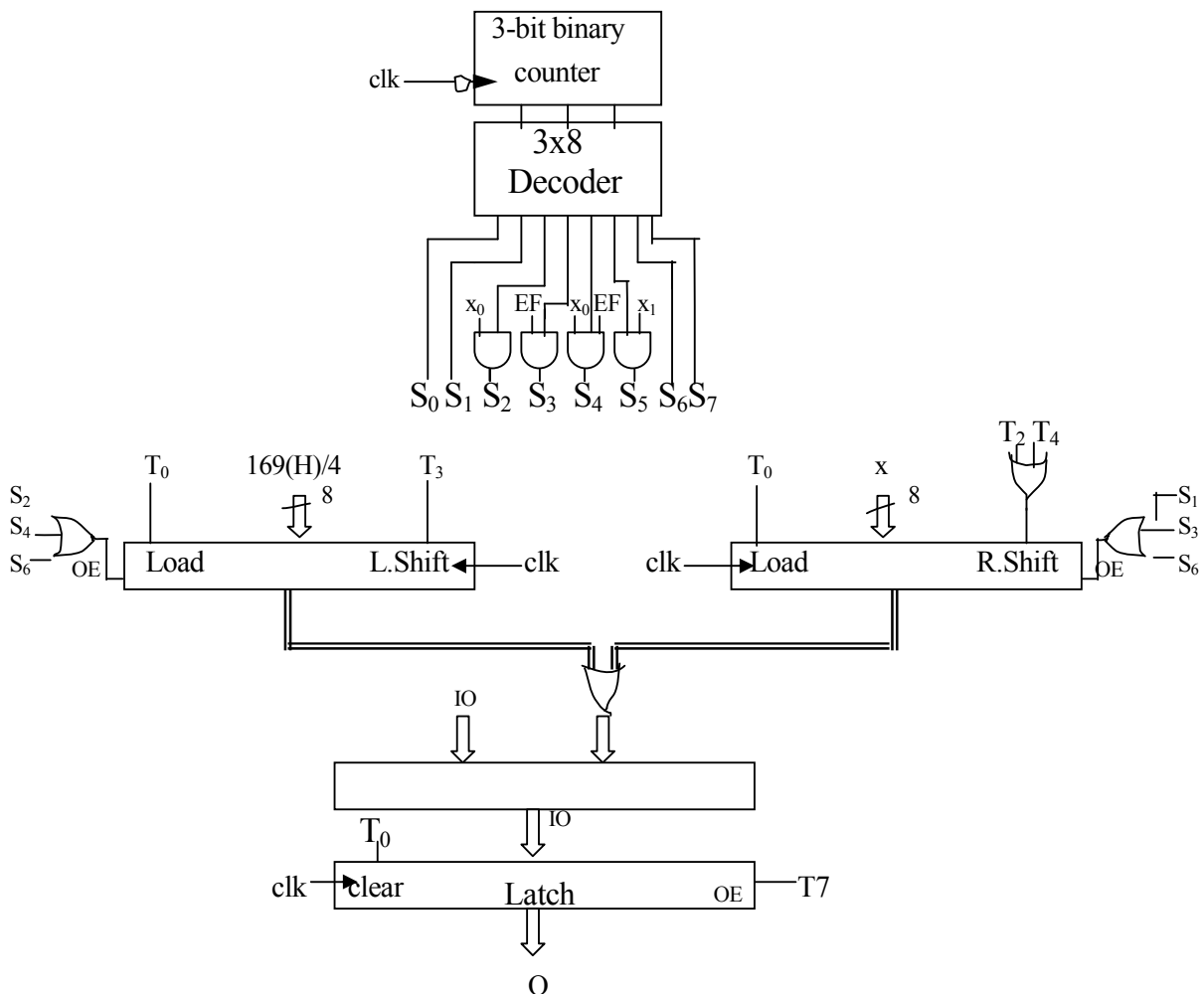
Now for the N block, we design a 5B/EF box i.e. a multiplier that multiplies the input by 5B as well as EF (provided signal 'EF'=1). From eqⁿ. (ii),

$$EF.x = x + \underbrace{(x \gg 2) + \frac{169(H).x_0}{4} + \frac{169(H).x_1}{2}}_{5B.x} + \frac{169(H).x_0}{2}$$

Therefore the algorithm to be followed is:

<u>T- state</u>	<u>Operation</u>
T ₀	Clear
T ₁	XOR x
T ₂	If x ₀ =1, XOR 169(H)/4
T ₃	If EF=1, XOR x/2
T ₄	If x ₀ =1 & EF=1, XOR 169(H)/2
T ₅	If x ₁ =1, XOR 169(H)/2
T ₆	XOR x/4
T ₇	Output

Hence the following circuit is proposed :-



CONCLUSION

The study of the AES candidate - Twofish, helped to gain some insight into the present-day requirements of commercial security. With special emphasis on the MDS (Maximum Distance Separable) matrix, the application of coding theory elements in Encryption Algorithm was seen. The importance of MDS codes in producing diffusion was analysed and this revealed the significance of maximum diffusion in preventing crypt-analytic attacks.

Finally, a hardware implementable VHDL model was proposed which could be used to observe outputs that conformed the expected results from theory.

References:

1. Twofish Encryption Algorithm - A 128-bit Block Cipher
By Bruce Schneier, John Kelsey, Doug Whiting, David Wagner,
Chris Hall, Niels Ferguson
2. The Theory of Error-Correcting Codes
By N.J.A. Sloane and F.J. Mac Williams
3. Coding Theory-The Essentials
By D.G.Hoffman, D.A.Leonard, C.C.Lindner, K.T.Phelps,
C.A.Rodger, J.R.Wall
4. The Designer's Guide to VHDL
By Peter J. Ashenden
5. www.nist.gov/aes
6. www.counterpane.com/twofish