International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 09, September - 2017

Hardware Implementation of Modified AES Algorithm for Secure Images

Rabid E
Dept. of Applied Electronics and Instrumentation,
Govt. Engineering College,
Kozhikode, India.

Jithesh C P
Dept. of Applied Electronics and Instrumentation,
Govt. Engineering College,
Kozhikode, India.

***Abstract*—In the era of internet the two main challenges of digital communication are security and storage space. Encryption and compression are the two techniques to address these challenges which are associated with multimedia communication over public network. In this work we implemented an efficient algorithm for encryption of image as a modification of Advanced Encryption Standard (AES) algorithm (MAES) along with an efficient algorithm for compression using Discrete Wavelet Transform (DWT) as modification of DWT (MDWT). The modification to DWT/IDWT is done by calculating a new computationally simple formula from the 9/7 lifting scheme, by which we can avoid usage of floating point arithmetic. The modification to AES is done by adding a key stream generator, such as (A5/1, W7), to the AES image encryption algorithm in order to increase the image security and in turn the encryption performance. Initially all the algorithms are modeled in Eclipse CDT (C/C++ Development Tool). Then hardware part is implemented in DSP starter kit DSK6713 using the integrated development environment (IDE) as Code Composer Studio(CCSv5) and results are verified.**

Keywords - MAES, MDWT, CCS, DSK6713

I. INTRODUCTION

In many areas like medical application, military application, space science application and many more, images are being used as one of the key information source. Special techniques are needed while storing images in online servers and transmitting information over public networks. Compression and encryption address these problems and has now become very important in modern digital communication network. This paper focuses on implementing an efficient algorithm for image encryption as a modification to one of the most popular encryption algorithm for images, the Advanced Encryption Standard (AES). The modification of AES (MAES) is achieved by adding a key stream generator W7 to the AES algorithm. The newly introduced key stream generator brought some computational timing to the algorithm, so prior that we pass the image through a Modified Discrete Wavelet Transform (MDWT) for compression. The compression intern reduces the computational delay and also reduces the required space for storing the images.

There are so many ways to do data encryption, and hence image encryption also. DES (Data Encryption Standard) can also be used for encrypting the image data. However, it is very complicated and computations involved are large. A software DES implementation is not fast enough to process the vast amount of data generated by multimedia applications and hardware for DES implementation (a set-top box) adds extra costs both to broadcasters and to receivers. In order to tackle these problems systems based on AES where proposed. AES is very fast symmetric block algorithm especially by hardware implementation [8]. In [1] M. Zeghid et al. proposes a new method to modify AES algorithm using a key stream generator W7 for highly secure images. In [2] Nagabushanam. M et al. explain how to model a computationally simplified lifting scheme implementation of DWT using a reduced floating point multiplication formula.

In this paper, we focus implementation of MAES algorithm with a key stream generator W7. The well documented 128 bit AES is taken as a base for encryption. The compressed image is split in to a plane of 128 bits and each plane is encrypted using the AES. W7 key stream generator is a random number generator of 128 bits; the random numbers are used as a key for encryption of each plane of image. So using a single 16 byte key as input, W7 generate random 16 byte keys for image encryption process. For achieving compression we use a MDWT. The MDWT is a simplified formulation of 9/7 lifting scheme implementation for finding the DWT, which avoid usage of floating point arithmetic defined in the 9/7 lifting scheme.

For implementation purpose all the four modules (MDWT, MAES, inverse MAES and MIDWT) are initially coded in Eclipse CDT (C/C++ Development Tool). Then the images are visually evaluated at each stage. The hardware implementation is done in DSP starter kit DSK6713 from Texas instruments. CCSv5 is used as the IDE for DSP synthesis and the above implemented C codes are ported to CCS. Then results are analyzed using the debug tool and memory view option available in CCS hardware emulation option.

The rest of the paper is systematized as follows: Section II describes the proposed MAES. Section III describes Implementation Platform. Section IV describes the implementation results. We conclude the paper in section V.

II. PROPOSED MAES

In [1] Zeghid et al. proposed a method to modify AES algorithm for better image encryption using a W7 key stream generator. In [2] M. Nagabushanam et al. explain how a MDWT can be achieved using less computation model. Here we propose a new model for secure images, the encryption is achieved using a more complex MAES and compression is done by using MDWT. Fig.1 shows the block diagram of overall MAES process.

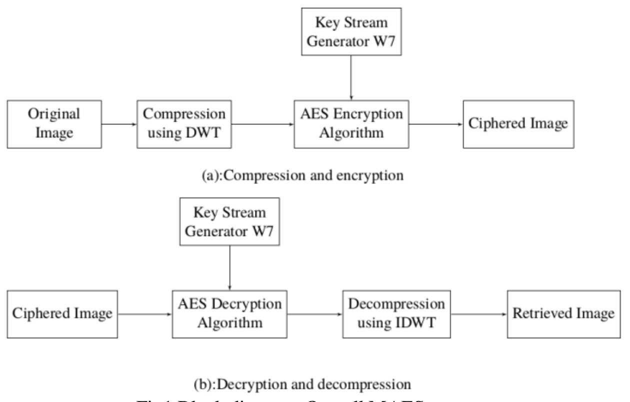


Fig1.Block diagram: Overall MAES process

First grayscale images are given as input to MDWT, resulting a transformed image and this transform is given as input to MAES encryption along with a key to the key-stream generator W7, resulting a ciphered image. In the receiver or decryption side the ciphered image is given as input to MAES decryption module along with the same key for W7, which is used for encryption. Then we get a transformed image of original and we find out an inverse transform using computationally simplified MIDWT algorithm.

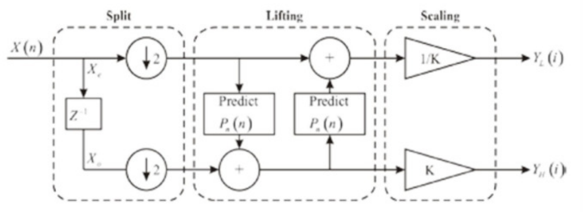


Fig. 2: Lifting Scheme Implementation of 1D-DWT

In high speed or low-power image processing applications, require a computationally simplified and memory efficient algorithm. Recently, a new mathematical formulation for wavelet transformation has been proposed in [3] as a light-weighted computation method for performing wavelet transforms with low power techniques. The lifting-based wavelet transform break-up the high pass and the low pass wavelet filters into a sequence of smaller filters. The lifting scheme requires fewer computations compared to the convolution based DWT. Therefore the computational complexity is reduced to almost a half of that needed with a convolution based approach. The vital steps involved in lifting-based wavelet transform are split, lifting, and scaling, respectively, as shown in Fig. 2

A. *Lifting Based DWT Scheme*

The basic idea of lifting scheme is first to compute a trivial wavelet (or lazy wavelet transform) by splitting the original 1-D signal into odd and even indexed sub sequences, and then modifying these values using alternating prediction and updating steps[2].

(1) Split step
The original signal, $X(n)$, is split into odd and even samples (lazy wavelet transform).
(2) Lifting step
This step is executed as N sub steps (depending on the type of the filter), where the odd and even samples are filtered by the prediction and update filters, Pn(n) and Un(n) .
(3) Normalization or Scaling step
After N lifting steps, scaling coefficients K and 1/K are applied respectively to the odd and even samples in order to obtain the low pass subband LY (i) , and the high-pass subband HY (i).

B. *9/7 Lifting Scheme*

The four stages are in DWT computation are, the split stage, predict and update stage and the scaling stage. When using 9/7 filters, there are two predict and two update stages ([4]), ([5]).

(1) Split step

$$Xe \leftarrow X(2i) \text{Even Samples} \qquad (1)$$
$$Xo \leftarrow X(2i+1) \text{Odd Samples} \qquad (2)$$

In this stage the input sequence X is split into two output sequences of even and odd samples.

(2) Lifting Steps

For (9, 7) filter, N = 2

$$\text{Predict P1 : } D(i) = Xo(i) + a[Xe(i) + Xe(i+1)] \qquad (3)$$
$$\text{Update U1 : } S(i) = Xe(i) + b[D(i-1) + D(i)] \qquad (4)$$
$$\text{Predict P2 : } Y_H(i) = D(i) + c[S(i) + S(i+1)] \qquad (5)$$
$$\text{Update U2 : } Y_L(i) = S(i) + d[Y_H(i-1) + Y_H(i)] \qquad (6)$$

In the predict stage, the even samples are averaged, and added with the odd samples to predict the next even and odd sample. In the update phase, the differences between the predicted sample and the actual sample are updated.

(3) Scaling Step

$$Y\,H(i) = KY_H(i) \qquad (7)$$
$$Y\,L(i) = \frac{1}{K}\, Y_L(i) \qquad (8)$$

In the last stage, the DWT coefficients are scaled to obtain the approximation and detail coefficients. The predict, update and scaling coefficients a, b, c, d and K are derived from 9/7 filter coefficients.

Where,
$a = -1.586134342$, $b = -0.0529801185$,
$c = 0.882911076$, $d = -0.443506852$
and $K = 1.149604398$.

C. *Modified Lifting based Algorithm*

In the lifting equations discussed in the previous section, require fixed point or floating point number representation. So the multipliers and adders need to be designed to operate on fixed or floating point numbers. The throughput and the latency can be improved by a modified lifting scheme algorithm is discussed in this section. The modification of algorithm can be achieved by removing the fractional coefficients and proposes a converted method that uses integer based adders, multipliers and shifting operation[2]. Substituting (3) in (4) to obtain (9).

Update U1 : $S(i) = Xe(i) + b[Xo(i-1) + a[Xe(i-1) + Xe(i)] + Xo(i) + a[Xe(i) + Xe(i+1)]]$

Simplified as U1:

$$S(i) = (1 + 2ab)Xe(i) + bXo(i-1) + ab[Xe(i-1) + Xe(i+1)] \quad (9)$$

Equation (10) is the modification of (6) using (5). So the new form of U2 equations can be expressed as:

Update U2:

$$Y_L\ (i) = S(i) + d[D(i-1) + c[S(i-1) + S(i)] + D(i) + c[S(i) + S(i+1)]]$$

Simplified as U2:

$$Y_L\ (i) = (1 + dc)S(i) + dD(i-1) + dc[S(i-1) + S(i+1)] \quad (10)$$

Substituting (9) and (10) into (7) the expression for Y L coefficient is obtained in (11).

$$Y\ L(i) = \frac{1}{K}\ Y_L\ (i) = [(1 + dc)/K]S(i) + [d/K]D(i-1) + [dc/K][S(i-1) + S(i+1)] \quad (11)$$

Equation (11) gives the modified expression for Y L, is obtained by regrouping the samples and coefficients. Scaling the new coefficients by 256 and rounding it, and the integer part gives the final coefficients. The rounded coefficients are used for computation of (11). Table 1 shows the reduced coefficients and their expressions. Numerical values of reduced coefficients scaled coefficients and rounded coefficients.

$$Y\ L(i) = [A1]S(i) + [A2]D(i1) + [A3][S(i1) + S(i+1)]$$

The coefficients are scaled to integer values and are presented in Table 1

Table 1: Rounded and Scaled Coefficients

Coefficients	Values	256*Coefficients/256	Rounded Coefficients
1+2ab	+1.0843358539	+227.512/256	278/256
b	-0.0529801185	-13.56/256	-14/256
ab	+0.08403358539	+21.51/256	+22/256
1+dc	+0.60842288808	155.75/256	156/256
d	-0.443506852	-113.53/256	-114/256
dc	-0.39157711191	-100.24/256	-100/256
A1=((1+dc)/K)	+0.529245442291	+135.48683322/256	135/256
A2=(d/K)	-0.38579084400	-98.762456290/256	-99/256
A3=(dc/K)	-0.34061900990	-87.1984665536/256	-87/256

The computation of YH term is also carried out in a similar manner; the (3) and (9) are used to show the procedure for computing the modified YH term. From the lifting equations, D(i) , S(i) and S(i + 1) are given by:

$$D(i) = Xo(i) + a[Xe(i) + Xe(i+1)]$$

$$S(i) = (1 + 2ab)Xe(i) + bXo(i-1) + ab[Xe(i-1) + Xe(i+1)]$$

$$S(i+1) = (1 + 2ab)Xe(i+1) + bXo(i) + ab[Xe(i) + Xe(i+2)]$$

Substituting, D(i), S(i) and S(i + 1) in Y_H (i) we get,

$$Y_H\ (i) = [Xo(i) + aXe(i) + aXe(i+1)] + c[(1 + 2ab)Xe(i) + bXo(i-1) + ab[Xe(i-1)] + (1 + 2ab)Xe(i+1) + bXo(i) + ab[Xe(i) + Xe(i+2)]$$

Substituting Y_H(i) in YH(i) and simplifying the terms reduces the above equation as in (13)

$$Y\ H(i) = A4[Xe(i) + aXe(i+1)] + A5[Xe(i-1) + Xe(i+2)] + A6Xo(i) + A7Xo(i-1)$$

The terms A4, A5, A6 and A7 are defined in table 2 and are scaled to integer values (M. Nagabushanam,et al., 2013). By scaling coefficients to integers, we can realize the algorithm using integer arithmetic units.

Table 2: Rounded and scaled coefficient in computing detail coefficients

coefficient	Expression	value	256*Coefficient/256	Rounded Coefficient
A4	K(a+c(1+2ab) +abc)	-0.6378406406	-163.28720400/256	-163
A5	K(abc)	0.0852939594	21.8352536/256	22
A6	k(BC+1)	-1.095829659	-280.53239828	-280
A7	Kcb	0.0537747384	13.7663330/256	14

Based on the above equations for YL(i) and YH(i), the integer coefficients form of new equations for YL(i) and YH(i), are presented in (12) and (13).

$$Y\ L(i) = 135 \times S(i) - 99 \times D(i-1) - 87[S(i-1 + S(i+1)] \quad (12)$$

$$YH(i) = -163 \times [Xe(i) + Xe(i+1)] + 22 \times [Xe(i-1) + Xe(i+2)] - 280 \times Xo(i) + 14 \times Xo(i-1) \quad (13)$$

The coefficients require more than 8 bits for representation; in order to reduce the multiplier size, the coefficients are factored and the equations are described in (14),

$$Y\ L(i) = (64 + 71) \times s(i) - (64 + 35) \times D(i-1) - (64 + 23) \times [S(i-1 + S(i+1)))] \quad (14)$$

Split the coefficients into two parts as (64 + x), where x = 71, 35, 23 the multiplication by coefficient 64 can be realized by right-shifting the samples by 6. The modified equations are given in (15)

$$Y\,L(i) = 71 \times s(i) - \text{right shift } 6(S(i)) - (35) \times D(i-1) - \text{right shift } 6(D(i-1)) - (23) \times [S(i-1) + S(i+1)] - \text{right shift } 6([S(i-1)) \qquad (15)$$

The coefficients 71, 35 and 23 can be represented using six bits, and hence reduces the size of the multiplier architecture. The right-shift operations also eliminate the use of multipliers. The overall architecture designed as above, reduces the computational complexity of YL computations [2].

Similarly the YH term can be realized as in (16),

$$Y\,H(i) = -35\times[Xe(i)+ Xe(i+1)] - \text{right shift } 7([Xe(i)+ Xe(i+1)]) + 22\times[Xe(i-1)+ Xe(i+2)] - 24 \times Xo(i) - \text{right shift } 7(Xo(i)) - \text{right shift } 7(Xo(i) + 14 \times Xo(i-1)) \qquad (16)$$

From (15) and (16), the size of the multiplier is reduced to six bits and right shift7 and Right shift6 operators are used; hence the overall computational complexity has been optimized (M. Nagabushanam,et al., 2013). The 2D-DWT, which consists of two stages of 1D-DWT, can be realized using modified lifting scheme.The developed equations are implemented and simulated in C programming Language using eclipse IDE.

D. AES Algorithm

The block cipher is a type of symmetric-key encryption algorithm that transforms a fixed-length plain text data into cipher text data of the same dimension. This transformation takes place under the action of a user provided secret key. The decryption is performed by applying the reverse transformation to the cipher text block using the same secret key. Advanced encryption standard (AES) is a widely used and well-known block cipher. The AES algorithm was created by the Belgians Vincent Rijmen and Joan Daemen. The overall structure of the AES algorithm can be seen in Fig. 3. State matrix which is a matrix of bytes is processed between many stages, or rounds, and therefore in each stage it is modified.

In the AES algorithm, the matrix size depends on the block size being used, which is composed of 4 lines and Nb columns. Here, Nb is the number of bits in the block that are divided by 32, since 32 bits represent 4 bytes. Thus the state will be composed of 4 lines and 4 columns as AES algorithm uses 128 bit blocks [6]. The number of rounds in AES is represented by Nr and Nk represents the number of columns in the grouping of key as that of the data block. The number of runs in the AES will depend on size of the key, for Nk equals to 4, 6 and 8, Nr will be 10, 12 and 14, respectively [7].

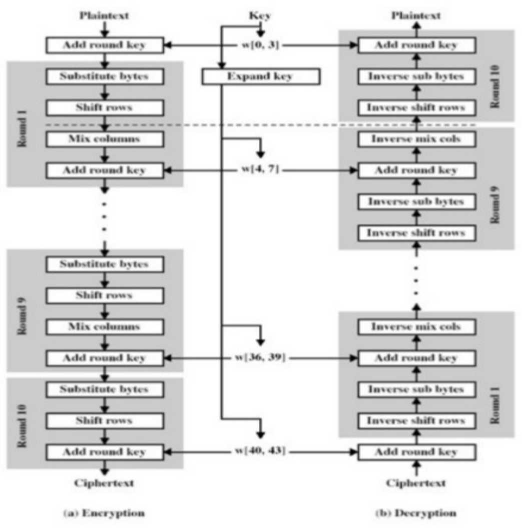


Fig. 3: Overall structure of the AES algorithm.

E. Modified AES algorithm

We can enhance the performance of AES by randomizing the key used for encryption. The new image encryption scheme is a modified AES algorithm. The modification can be done using a key stream generator W7 as shown in Fig. 1. W7 key stream generator contains eight similar models; C1, C2, ..., C8. Three LFSR's and one majority function included in each model [8]. There is a control unit and a function unit. The key stream is generated by the function unit. The proposed architecture can see in Fig. 4. For one cell there are two inputs and one output. The one input is the key and it is the same for all the cells. The other input consists of control signals. Produces the output is of 1-bit long. Combining outputs of all cell give us one key stream byte

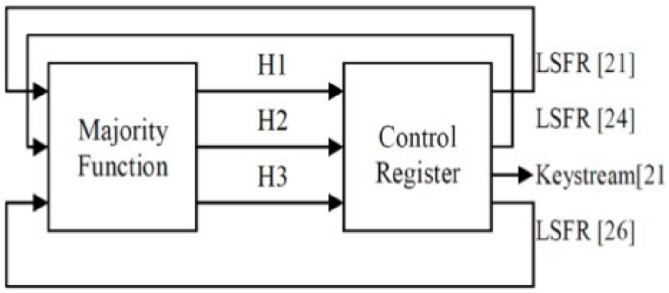


Fig. 4: proposed architecture for W7 key stream generator

The proposed architecture for implementation of one cell is presented in Fig. 5. Here each cell consists of three LFSRs 38-, 43- and 47-bit long and a majority function. Their initial state is mapped from input key of 128bit long, which is the same for all modules.

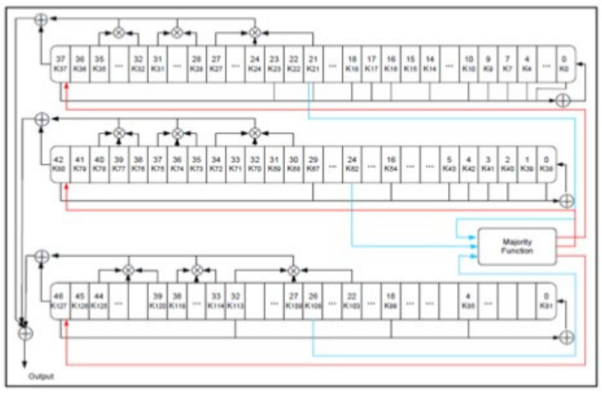


Fig. 5: Proposed architecture for W7 cell C2

The 128-bit of the key map to the LFSRs initial state as:
LFSRa(38 − bit) : LFSR0 = K0, LFSR1 = K1, ..., LFSR36 = K36, LFSR37 = K37
LFSRb(43 − bit) : LFSR0 = K38, LFSR1 = K39, ..., LFSR41 = K79, LFSR42 = K80
LFSRc(47 − bit) : LFSR0 = K81, LFSR1 = K82, , LFSR45 = K126, LFSR46 = K127

The remaining Cells mapping, tapping for majority function and output bits are discussed in detail on [9].

III. IMPLEMENTATION PLATFORM

Initially all functional blocks (modified DWT, AES Encryption, key stream generator W7, AES Decryption, and modified IDWT) are individually developed and tested. For the testing process first we implemented these functional

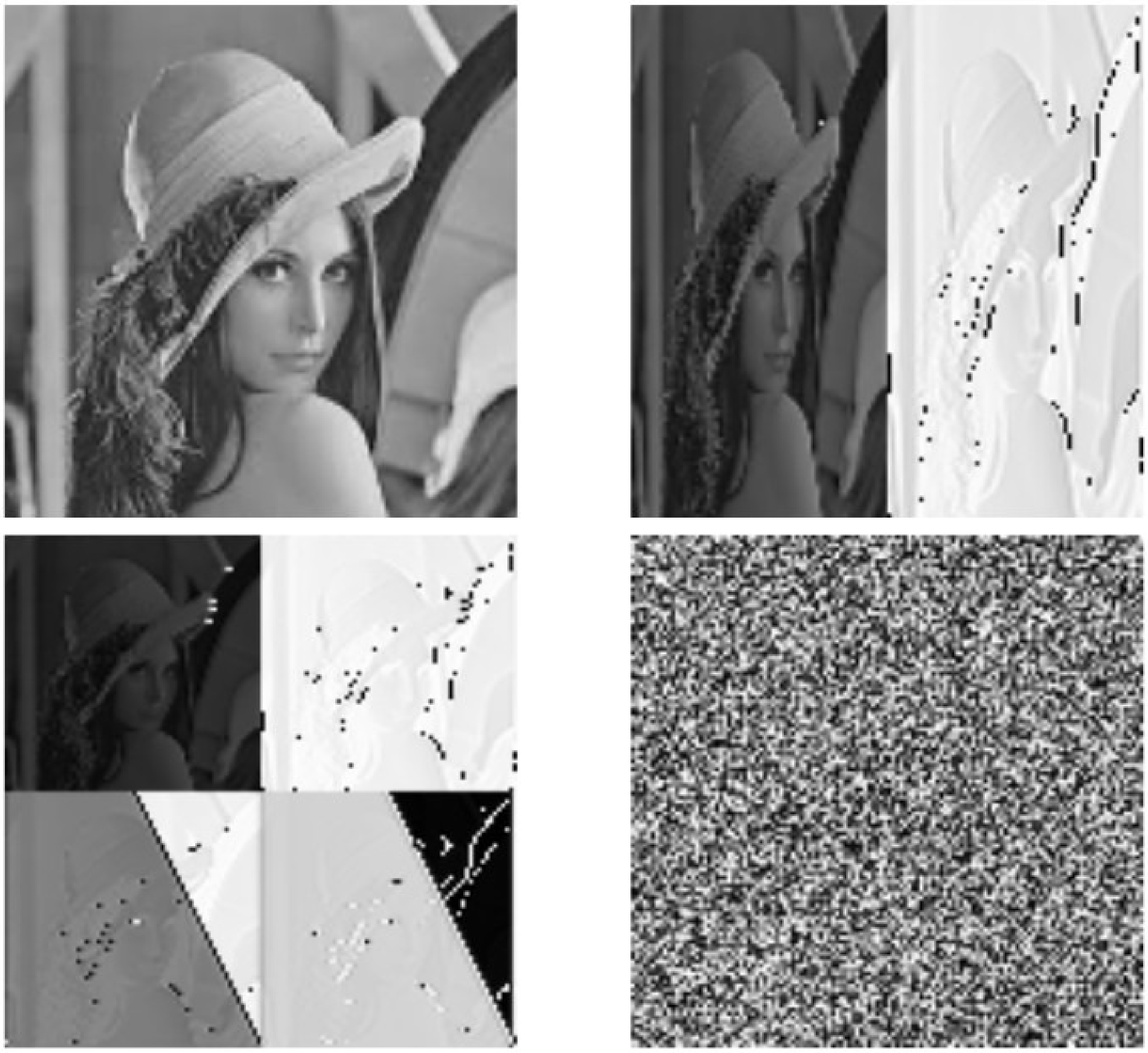


Fig. 6: Images of various stages on LenaGS image

blocks using programming language C in Eclipse IDE. Then the hardware implementation is done in DSP starter kit DSK6713 from Texas instruments. Code Composer Studio.v5

(CCSv5) is used as the IDE for DSP synthesis and the above implemented C codes are ported to CCS. Then results are analyzed using the debug tool and memory view option available in CCS hardware emulation option. The physical view and Layout of C6713 DSK can be seen in Fig. 7

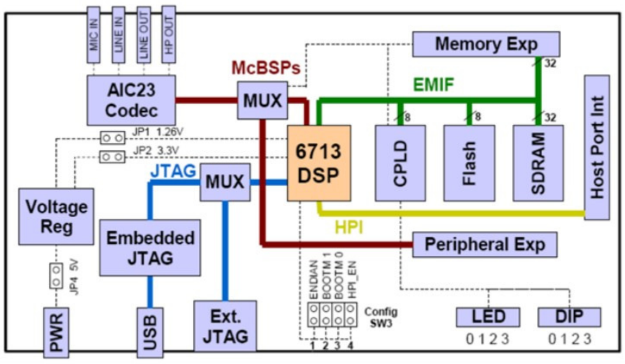


Fig. 7: Functional block diagram of DSK6713.

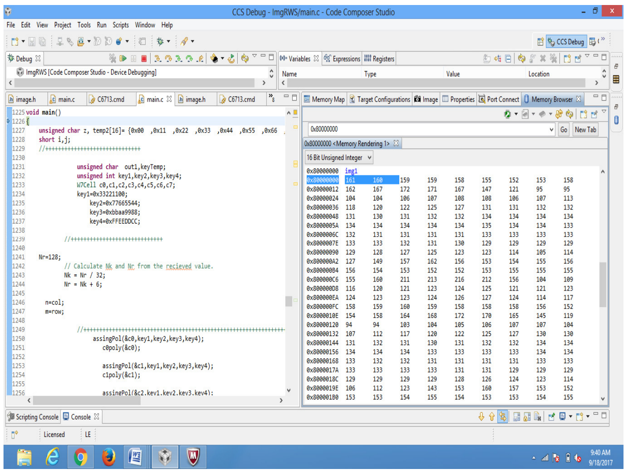
IV MPLEMENTATION RESULTS

The results opted after applying the algorithm has been shown below. Four different sample images (widely used) are taken which has different intensity variations. 8 bit 128x128 gray scale images are used for experiment. For getting 2D-WT first we apply 1D-DWT along columns and then we apply
Fig. 7: Functional block diagram of DSK6713.same 1D-DWT along rows of the image. The encryption methodology works on a state of 128 bits, so each image has been divided into its corresponding pixel blocks and then the algorithm applied on it. Fig.6(a) to (d) shows orginal image, 1D-DWT, 2D-DWT, and modified AES encryption done on the sample image 'LenaGS.png'

For hardware implementation, the TMS320C6713 DSP has only 192 K-bytes of internal RAM. So applications such as image processing, which are using large memory, we want to utilize the additional external RAM available in DSK6713. To utilize this memory we want to set the variables used in programming to the external RAM. This can be done by editing the .far field to .far > EMIFCE0, in the C6713.cmd file associated with the CCS project, which ensure global variables used in C programming should be in external RAM. Then we can numerically verify the emulation results using the memory view option available in the CCS. Fig.8 and Fig.9 shows the memory view of image before and after execution. The results show that there is little variation in the final reconstructed signal from original image gray level values.

V. CONCLUSION

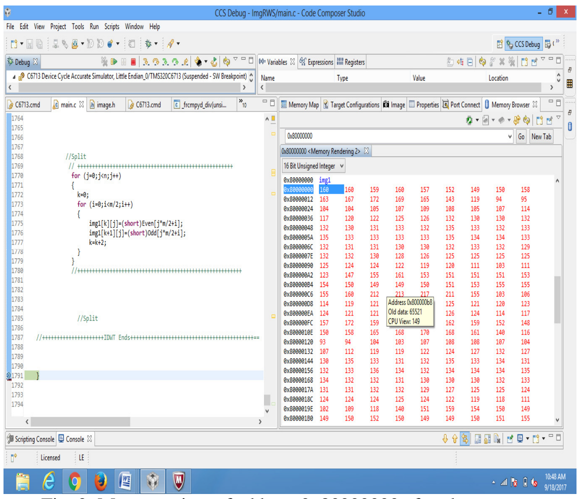
This work mainly focused on developing an algorithm for modified AES (MAES) for secure images. The modification is done using a key stream generator W7. Before encryption, the
image is processed through a 2D-DWT algorithm, which is actually modified 1D-DWT taken along rows and columns of the images. Here the modification is done in such a way that the computation of lifting schemes can be achieved within less complexity by avoiding the floating point multiplication. Software modeling is done in eclipse and simulation results have been shown. Hardware implementation of the algorithm was done using Code Composer Studio CCSv5 for the DSK6713 kit platform. By introducing the key stream generator, the security features are enhanced at the cost of computational complexity. Computational complexity may be further improved by exploring suitable encryption techniques to reduce the image size.



```
void main()
{
    unsigned char z, temp2[16]= {0x00  ,0x11  ,0x22  ,0x33  ,0x44  ,0x55  ,0x66
    short i,j;
    //++++++++++++++++++++++++++++

                unsigned char  out1,keyTemp;
                unsigned int key1,key2,key3,key4;
                W7Cell c0,c1,c2,c3,c4,c5,c6,c7;
                key1=0x33221100;
                    key2=0x77665544;
                    key3=0xbbaa9988;
                    key4=0xFFEEDDCC;

            //++++++++++++++++++++++++++++

    Nr=128;
            // Calculate Nk and Nr from the recieved value.
            Nk = Nr / 32;
            Nr = Nk + 6;

      n=col;
      m=row;

                //+++++++++++++++++++++++++++++++++++++++++++++++++++++++
                     assingPol(&c0,key1,key2,key3,key4);
                        c0poly(&c0);

                        assingPol(&c1,key1,key2,key3,key4);
                        c1poly(&c1);

                        assingPol(&c2,key1,key2,key3,key4);
```

```
0x80000000  img1
0x80000000  161     160     159     159     158     155     152     153     158
0x80000012  162     167     172     171     167     147     121     95      95
0x80000024  104     104     106     107     108     108     106     107     113
0x80000036  118     120     122     125     127     131     131     132     132
0x80000048  131     130     131     132     132     134     134     134     134
0x8000005A  134     134     134     134     134     135     134     134     133
0x8000006C  132     131     131     131     131     133     133     133     133
0x8000007E  133     133     132     131     130     129     129     129     129
0x80000090  129     128     127     125     123     123     114     105     114
0x800000A2  127     149     157     162     156     153     154     155     156
0x800000B4  156     154     153     152     152     153     155     155     155
0x800000C6  155     160     211     213     216     212     156     104     109
0x800000D8  116     120     121     123     124     125     121     121     123
0x800000EA  124     123     123     124     126     127     124     114     117
0x800000FC  158     159     160     159     158     158     158     156     152
0x8000010E  154     158     164     168     172     170     165     145     119
0x80000120  94      94      103     104     105     106     107     107     104
0x80000132  107     112     117     120     122     125     127     130     130
0x80000144  131     132     131     130     131     132     132     134     134
0x80000156  134     134     134     133     133     133     133     134     134
0x80000168  133     132     132     131     131     131     131     133     133
0x8000017A  133     133     133     133     131     131     129     129     129
0x8000018C  129     129     129     129     128     126     124     123     114
0x8000019E  106     112     123     143     153     160     157     153     152
0x800001B0  153     153     154     155     154     153     153     154     155
```

Fig. 8: Memory view of address 0x80000000 before the process.



```
                //Split
                 // +++++++++++++++++++++++++++++++++++++++++++++++++
                  for (j=0;j<n;j++)
                  {
                    k=0;
                    for (i=0;i<m/2;i++)
                    {
                        img1[k][j]=(short)Even[j*m/2+i];
                        img1[k+1][j]=(short)Odd[j*m/2+i];
                        k=k+2;
                    }
                  }
                  //++++++++++++++++++++++++++++++++++++++++++++++++++++




                    //Split

    //++++++++++++++++++++IDWT Ends++++++++++++++++++++++++++++++++++++++++++==



    }
```

```
0x80000000  img1
0x80000000  160     160     159     160     157     152     149     150     158
0x80000012  163     167     172     169     165     143     119     94      95
0x80000024  104     104     105     107     109     108     105     107     114
0x80000036  117     120     122     125     126     132     130     130     132
0x80000048  132     130     131     133     132     135     133     132     133
0x8000005A  135     133     133     133     133     135     134     134     133
0x8000006C  132     131     131     130     130     132     133     132     129
0x8000007E  132     132     130     128     126     125     125     125     125
0x80000090  125     124     124     122     119     120     111     103     111
0x800000A2  123     147     155     161     153     151     151     151     153
0x800000B4  154     150     149     149     150     151     153     155     155
0x800000C6  155     160     212     213     217     211     155     103     106
0x800000D8  114     119     121     [illegible] [illegible] 125  121     120     123
0x800000EA  124     121     121     [illegible] [illegible] 126  124     114     117
0x800000FC  157     172     159     [illegible] [illegible] 162  159     152     148
0x8000010E  150     158     165     168     170     168     161     140     116
0x80000120  93      94      104     103     107     108     108     107     104
0x80000132  107     112     119     119     122     124     127     132     127
0x80000144  130     135     133     131     132     135     133     134     131
0x80000156  132     133     136     134     132     134     134     134     135
0x80000168  134     132     132     131     130     130     130     132     133
0x8000017A  131     131     132     132     129     127     125     125     124
0x8000018C  124     124     124     125     124     122     119     118     111
0x8000019E  102     109     118     140     151     159     154     150     149
0x800001B0  149     150     152     150     149     149     150     151     155

Address 0x800000b8
Old data: 65521
CPU View: 149
```

Fig. 9: Memory view of address 0x80000000 after the process.

REFERENCES

[1] M. Zeghid, M. Machhout, L. Khriji, A. Baganne, and R. Tourki. (2007). A Modified AES Based Algorithm for Image Encryption. International Journal of Computer, Information, Systems and Control Engineering, 1(3): 745 -750.

[2] Nagabushanam. M, P. Kumar and Ramachandran .S. (2013). FPGA Implementation of 1D and 2D DWT Architecture using Modified Lifting Scheme. WSEAS TRANSACTIONS on SIGNAL PROCESSING 4(9): 227-236 .

[3] Tze-Yun Sung,Hsi-Chin Hsin, Yaw-Shih Shieh and Chun-Wang Yu. (2006). Low-Power Multiplierless 2-D DWT and IDWT Architectures Using 4-tap DaubechiesFilters. Seventh International Conference on Parallel and Distributed Computing, (pp): 185-190.

[4]. Anirban Das, Anindya Hazara, and Swapna Banerjee. (2010). An Efficient Architecture for 3-D Discrete Wavelet Transform. IEEE Transactinos on circuit and systems for video technology , 20(2):286-296.

[5]. Tinku Acharya and Chaitali Chakrabarti. (2006). Survey on Lifting-based Discrete Wavelet Transform Architectures. Springer Science, Journal of VLSI Signal Processing , 42:321-339

[6] DAEMEN, J. AND RUMEN, V. A.(2010). Specification for the AES Algorithm NIST(http://csrc.nist. gov/archive/aes/rijndael/wsd /index. Html)

[7] FIPS dards F1PS-197.(2001).Publication FIPS-197,Federaln Information Advanced Processing Encryption Standard Stan-(AES). http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[8] K. Janvinen, M. Tominisko, J. Skytta. (2003)VA fully pipelined memorlyess 17.8 Gpbs AES-128 encryptor. International symposium of Field programmable Gate arrays : 207-215

[9] S. Thomas, D. Anthony, T. Berson and G. Gong, "The W7 Stream Cipher Algorithm", Internet Draft, April 2002. *https://tools.ietf.org/html/draft-thomas-w7cipher-01*