Adaptive FIR Filter Using Distributed Airthmetic for Area Efficient Design

Manish Kumar[*], Dr. R.Ramesh[**]

[*] Dept.of Electronics and Communication, Saveetha Engineering College, Chennai, India.
[**] Dept.of Electronics and Communication, Saveetha Engineering College, Chennai, India.

Abstract- In this paper we propose an efficient pipelined architecture for low power,high throughput and low area adaptive FIR filter based on distributed airthemetic. The throughput rate is significantly increased by parallel look-up table(LUT) update. Reduction in power consumption is achieved by using a fast bit clock for carry save accumulation. We have shown that sampling period could be sequentially reduced by using carry save accumulation for DA based inner product. It involves half the number of register compared to the existing DA based design to store of input samples. The system is implemented in FPGA that enables rapid prototyping of digital cirtcuits.

Index Terms- Finite Impulse Response (FIR), Look Up Table (LUT), Distributed airthemetic (DA), Field Programmer Gate Array (FPGA).

I. INTRODUCTION

In the recent years, there has been a growing trend to implement digital signal processing functions in Field Programmable Gate Array (FPGA). In this sense, we need to put great effort in designing efficient architectures for digital signal processing functions such as FIR filters, which are widely used in video and audio signal processing, telecommunications and etc.

Many digital signal processing (DSP) applications require linear filters that can adapt to changes in the signals they process. Adaptive filters find extensive use in several DSP applications including acoustic echo cancellation, signal de-noising, sonar signal processing, clutter rejection in radars, And channel equalization for communications and networking systems [1], [2]. In many cases, the sampling frequencies for digital processing of these signals are close to the system clock frequencies. Thus, it is important for the adaptive filters implementedto have a high throughput

Traditionally, direct implementation of a K-tap FIR filter requires K multiply-and-accumulate (MAC) blocks, which are expensive to implement in FPGA due to logic complexity and resource usage. To resolve this issue, we first present DA, which is a multiplier-less architecture.

Implementing multipliers using the logic fabric of the FPGA is costly due to logic complexity and area usage, especially when the filter size is large. Modern FPGAs have dedicated DSP blocks that alleviate this problem, however for very large filter sizes the challenge of reducing area and complexity still remains. Very efficient methods have been developed for the parallel implementation of static digital filters in field programmable logic arrays (FPGAs) or custom ICs [4]. Distributed arithmetic (DA) [5] is one method often preferred since it eliminates the need for hardware multipliers and is capable of implementing large filters with very high throughput .

An alternative to computing the multiplication is to decompose the MAC operations into a series of lookup table (LUT) accesses and summations. This approach is termed distributed arithmetic (DA), a bit serial method of computing the inner product of two vectors with a fixed number of cycles.

The original DA architecture stores all the possible binary combinations of the coefficients w[k] of equation (1) in a memory or lookup table. It is evident that for large values of L, the size of the memory containing the pre computed terms grows exponentially too large to be practical. The memory size can be reduced by dividing the single large memory (2Lwords) into m multiple smaller sized memories each of size 2k where $L = m \times k$. The memory size can be further reduced to $2L-1$ and $2L-2$ by applying offset binary coding and exploiting resultant symmetries found in the contents of the memories.

This technique is based on using 2's complement binary representation of data, and the data can be pre-computed and stored in LUT. As DA is a very efficient solution especially suited for LUT-based FPGA architectures, many researchers put great effort in using DA to implement FIR filters in FPGA. Patrick Longa introduced the structure of the FIR filter using DA algorithm and the functions of each part. Sangyun Hwang analyzed the power consumption of the filter using DA algorithm. Heejong Yoo proposed a modified DA architecture that gradually replaces LUT requirements with multiplexer/adder pairs. But the main problem of DA is that the requirement of LUT capacity increases exponentially with the order of the filter, given that DA implementations need 2Kwords (K is the number of taps of the filter). And if K is a prime, the hardware resource consumption will cost even higher. To overcome these problems, this paper presents a hardware-efficient DA architecture.

In this paper, we propose an efficient LUT-less architecture for high-speed DA-based adaptive filter with very low adaptation-delay. We have shown that the minimum sampling period could be substantially reduced by using carry-save accumulation for DA-based inner-product computation. The proposed design requires less than half the number of registers compared to the existing design for storing the necessary sums of input samples and involves a simple structure for weight updating. This method not only reduces the LUT size, but also modifies the structure of the filter to achieve high speed performance. The proposed filter has been designed and synthesized with ISE 14.1, and implemented with a FPGA

SPARTAN3E device. Our results show that the proposed DA architecture can implement Adaptive FIR filters with high speed.

In the next Section, we present a brief review of the LMS adaptive algorithm, followed by the description of the proposed DA-based technique for adaptive filter in Section III. The structure of proposed adaptive filter is described in Section IV. We provide the synthesis report and X-power Analysis of the proposed structure [8] in Section V, and conclusions are given in Section VI.

II. REVIEW OF LMS ADAPTIVE ALGORITHMS

During each cycle, the LMS algorithm computes a filter output and an error value of magnitude equal to the difference between the current filter output and the desired response. The estimated error is then used to update the filter weights in every training cycle. The weights of LMS adaptive filter during the *n*th iteration are updated according to the following

equations.

$$W_{n+1}=W_n+\mu e(n).X_n \quad (1a)$$

Where

$$e(n) = d(n) - y(n) \quad (1b)$$

and

$$y(n) = W_n^T X_n \quad (1c)$$

X_n and W_n are the input vector and the weight vector, respectively, at the *n*th training iteration, and given by

$$X_n = [\, x(n),x(n-1),\ldots,x(n-N+1)]^T \; W_n =[\, w_n(0),w_n(1),\ldots,w_n(N-1)]^T$$

$d(n)$ is the desired response and $y(n)$ is the filter output of the *n*th iteration. $e(n)$ denotes the error computed during the *n*th iteration which is used to update the weights, and μ is the m convergence-factor. N is the order of the LMS adaptive filter.. The weight-update equation of such delayed LMS adaptive filter is given by

$$W_{n+1} = W_n+\mu e(n-m).X_{n-m} \quad (2)$$

The block diagram of delayed LMS adaptive filter is depicted in Fig. 1. It can be seen that the adaptation-delay *m* is the number of cycles required for the error corresponding to any given sampling instant to become available to the weight adaptation circuit. In the conventional delayed LMS algorithm.

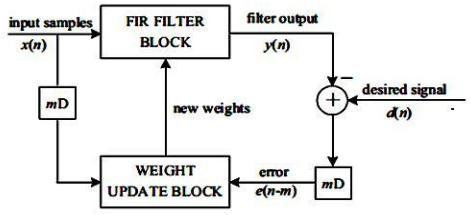

Fig. 1. Pipelined LMS adaptive filter.

(Fig.1), the adaptation-delay of *m* cycles amounts to the delay introduced by the whole of adaptive filter structure consisting of FIR filtering and weight adaptation process. It is shown in [9] that the adaptation-delay of conventional LMS could be decomposed into two parts, where one part is the delay introduced by the pipeline stages in the FIR filtering and the other part is due to the delay involved in pipelining the weight adaptation process. Based on such a decomposition of delay, a pipelined LMS adaptive filter could be implemented by a structure shown in Fig.2. Assuming that the latency of computation of error is $n1$ cycles, the error computed by the structure at the nth cycle is $e(n-n1)$, which is used with the input samples delayed by $n1$ cycles to generate the weight increment term. The weight-update equation of the pipelined LMS algorithm is given by

$$W_{n+1} = w_n+\mu e(n-n1).X_{n-n1} \quad (3a)$$

where

$$e(n-n1) = d(n-n1) - y(n-n1) \quad (3b)$$

and

$$Y(n) = w_n^T - n2.x_n \quad (3c)$$

We can notice that during weight adaptation, the error with $n1$ delays is used while the filtering unit uses the weights delayed by $n2$ cycles.

III. PROPOSED DA APPROACH FOR ADAPTIVE FIR FILTER

As discussed in the previous Section, the LMS adaptive filter consists of two main computational blocks, namely (i) the error-computation block which performs an *N*-point innerproduct computation followed by error calculation and (ii)

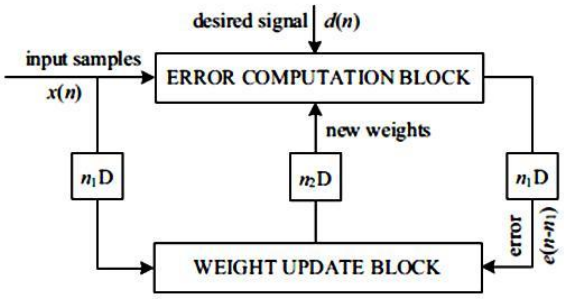

Fig. 2. Modified pipelined LMS adaptive filter.

the weight-update block which performs *N*-multiplication accumulation to obtain the new filter weights. For simplicity of presentation let the inner-product of (1c) be given by

$$Y = \sum_{k=0}^{N-1} wk\, xk \quad (4)$$

where wk and xk for $0 \le k \le N-1$ form the *N*-point vectors corresponding the current weights and recent-most $N-1$ input, respectively. Similarly, the weight-update equation (3a) be expressed in a generalized form

$$(w_k)_{new} = w_k+\mu e.x_k \quad \text{for } 0 \le k \le N-1 \quad (5)$$

A. DA Approach for Inner-Product Computation

Assuming L to be the word-length, each component of input vector **x** may be expressed in two's complement representation:

$$x_k = -x_{ko} + \sum_{l=1}^{L-1} x_{kl} 2^{-l} \qquad (6)$$

where x_{kl} denotes the lth bit of x_k. Substituting (6), we can write (4) in an expanded form:

$$Y = -\sum_{k=0}^{N-1} x_k w_{ko} + \sum_{k=0}^{N-1} x_k \left[\sum_{k=0}^{L-1} w_{kl} x_{kl}\right] 2^{-1l} \qquad (7)$$

To convert the sum-of-products form of inner-product of (4) into a distributed form, the order of summations over the indices k and l in (7) can be interchanged to

$$Y = -\sum_{k=0}^{N-1} x_k w_{ko} + \sum_{l=1}^{L-1} 2^{-l} \left[\sum_{k=0}^{N-1} x_k w_{kl}\right] \qquad (8)$$

and the inner-product given by (8) can be computed as

$$Y = \left[\sum_{l=1}^{L-1} 2^{-l} y_l\right] - y_0 \qquad (9b)$$

Where

$$y_l = \sum_{k=0}^{N-1} x_k w_{kl} \qquad (9a)$$

Since any element of the N-point bit-sequence $\{x_{kl}$ for $0 \le k \le N-1\}$ can either be 0 or 1, the partial sum y_l for $0 \le l \le L-1$, can have $2N$ possible values. If all the $2N$ possible values of y_l are pre-computed and stored in a lookup table (LUT), the partial sums y_l can be read out from the LUT using the bit-sequence $\{x_{kl}$ for $0 \le k \le N-1\}$ as address-bits for computing the inner-product.

The inner-product can, therefore, be calculated according to (9), in L cycles of shift-accumulation followed by LUT-read operations corresponding to L number of bit-slices $\{x_{kl}\}$ for $0 \le l \le L-1$ as shown in Fig.3.

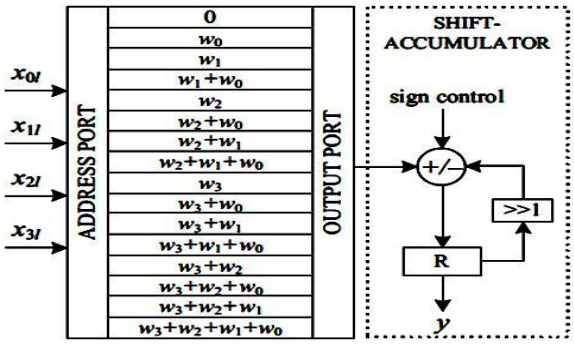

Fig. 3. Block diagram of conventional DA-based implementation of 4-point inner-product. $\{x_{3l}x_{2l}x_{1l}x_{0l}\}$ for $0 \le l \le (L-1)$ which constitute the bit-slices of the input vector are fed as address to the LUT during different bit-cycles.

Note that the partial inner-product y_l for $1 \le l \le L-1$ are added while y_0 is subtracted. Therefore, sign of the partial sums are required to be modified for $l = 0$. All the output bits of the LUT is thus required to pass through XOR gates for necessary sign-modification. To avoid this sign-modification, we can safely assume the signal samples to be unsigned words of size L, since we can always obtain unsigned input signal by adding fixed offset when the original input signal is signed. The inner-product given by (9) then can be expressed in a simpler form:

$$y = \sum_{l=0}^{L-1} 2^{-1} y_l \qquad (10a)$$

Where

$$y = \sum_{k=0}^{N-1} w_k x_{kl} \qquad (10b)$$

so that no sign-reversal of LUT output is required before the shift-accumulation. Moreover, the shift-accumulation in Fig.3 involves significant critical-path, which could be replaced by carry-save accumulator, shown in Fig.4. The bit-slices of vector x are fed one after the next in the LSB to MSB order to the carry-save accumulator. The sum and carry words obtained after L clock cycles are added together by a final adder.

B. Weight Updating for DA-Based Adaptive Filtering

As suggested in [8], we can update LUT location containing the partial inner-products in Fig.3, instead of updating of individual weights. The content of kth LUT location is updated according to the relation

$$(q_k)_{new} = q_k + \mu e.c_k \qquad (11)$$

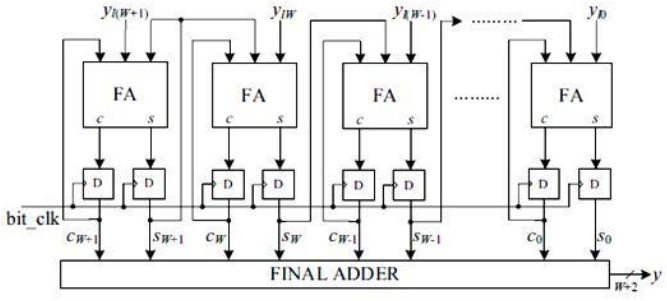

Fig. 4. Carry-save implementation of shift-accumulation.

kj is the (*j*+1)th bit of *N*-bit binary representation of integer *k* for $0 \le k \le 2N - 1$. Note that *ck* for $0 \le k \le 2N - 1$ can be pre computed and stored in RAM LUT of 2*N* words. But instead of storing 2*N* words in LUT we need to store $(2N/2 - 1)$ words in an input register table. The example of such an input table for *N* = 4 is shown in Fig.5. It contains only 7 registers to store the pre-computed sums of input words,

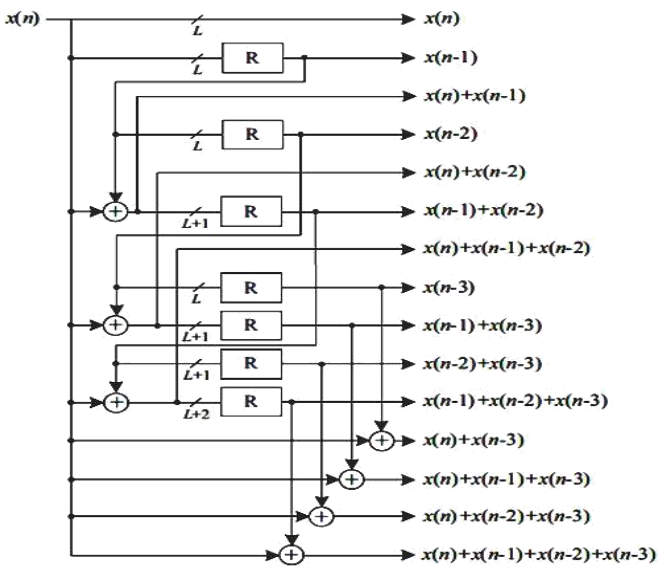

Fig. 5. The input table for generation of possible sums of input samples.

while 7 new sums are computed by seven adders. Also note that $c0 = 0$ and $c1$ is the recent-most input sample which need not be stored in registers

IV. PROPOSED DA-BASED ADAPTIVE FILTER STRUCTURE

A straight-forward DA-based implementation of inner product of long vectors requires LUT of very large size. Therefore, the inner-products of long vectors need to be decomposed [10], and accordingly the computation of adaptive filters of large orders need to be decomposed into small adaptive filtering blocks. Therefore, we describe here the proposed DA-based structures of small order and large order LMS adaptive filters separately in the next two sub-sections.

A. Proposed Structure for Small Filter Orders

The proposed structure of DA-based adaptive filter of order $N = 4$ is shown in Fig.6. The filtering unit of this structure consists of an array of 15 registers R containing the partial inner-products *yl* for $0 < l \le 15$ and a 16 : 1 MUX1 to select the content of one of the those registers. Bit-slices of input word $A = \{x3l\ x2l\ x1l\ x0l\}$ for $0 \le l \le L - 1$ are fed to the MUX as control in LSB to MSB order, and the output of the MUX is fed to the carry-save accumulator (Fig. 4) for successive accumulation in each cycle. After *L* bit-cycles (The duration of each bit-cycle is equal to sum of one full-adder time *TFA* and a delay of the 16 : 1 MUX.) the carry-save accumulator shift-accumulates all the partial inner-products and generates a sum word and a carry word of size (*W*+2)-bit each.

The weight-update unit for $N = 4$ consists of an input table consisting of 15 words (shown in Fig.5). The 15 words of input table are fed to 15 barrel-shifters. The barrel-shifter output yields the desired increments to be added with or subtracted from the partial inner-product available in the weight registers. The sign-bit of the error is used as the control for add/subtract cell such that when sign-bit is 0 or 1 the barrel-shifter output is added with or subtracted from the content of the corresponding weight register, respectively.

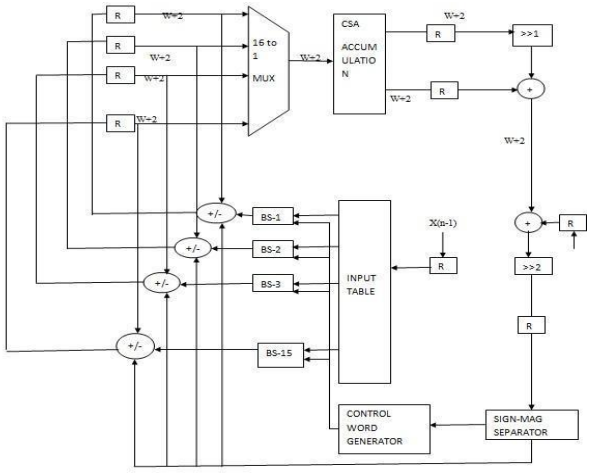

Fig. 6. Proposed structure of DA-based LMS adaptive filter of order N = 4.

if $r_6 = 1$ then $t = 0$;
else if $r_5 = 1$ then $t = 1$;
else if $r_4 = 1$ then $t = 2$;
else if $r_3 = 1$ then $t = 3$;
else if $r_2 = 1$ then $t = 4$;
else if $r_1 = 1$ then $t = 5$;
else if $r_0 = 1$ then $t = 6$;
else then t = 7;

r=mag($\mu e(n-2)$)
r_i : ith bit of 7-bit word r

Fig. 7. The logic used for generation of control word t for the barrel-shifter.

Fig. 8. Proposed structure of DA-based LMS adaptive filter of order N = 16 and P = 4.

V. SYNTHESIS REPORT, RESULTS AND DISCUSSIONS

We discuss here the area and power consumption of existing DA based design of [8]. A filter with larger order replicates most part of the circuit for $N = 4$ and involves some additional adders as shown in Fig. 8. The proposed design involves 16 adders/subtractors (15 for weight-update and 3 in CSA accumulator, final adder and error calculation of Fig. 6, and 7 in the input table of Fig. 5), 15 logarithmic barrel shifters of 3-stage each and 32 registers which is given in fig 9. In X-power analysis in fig 10 we get power consumption of 0.010W. the utilization is 66 out of 7 which provide utilization factor of 10.6. In thermal properties effective TJA is given 40.0C/w. Maximum ambient temperature is given 81.7C. Due to this junction temperature is 28.3C. We have coded the each architecture in verilog and simulate with Xilinx 14.1.

```
Macro Statistics
# Adders/Subtractors                   : 16
 1-bit adder                          : 7
 1-bit adder carry in                 : 2
 4-bit adder carry out                : 4
 6-bit adder                          : 1
 6-bit subtractor                     : 1
 7-bit adder                          : 1
# Registers                            : 106
 Flip-Flops                           : 106
# Latches                              : 32
 1-bit latch                          : 32
```

Fig 9 Design synthesis

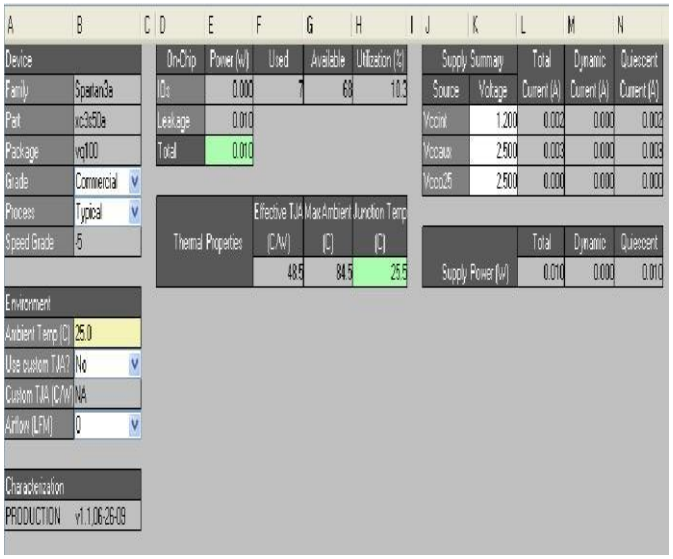

Fig 10 X-Power Analysis

VI. CONCLUSION AND FUTURE WORK

We have suggested an efficient pipelined architecture for low-power, high-throughput, and low-area implementation of DA based adaptive filter. Throughput rate is significantly enhanced by parallel LUT update and concurrent processing of filtering operation and weight-update operation. We have also proposed a carry-save accumulation scheme of signed partial inner products for the computation of filter output. Offset binary coding is popularly used to reduce the LUT size to half for area-efficient implementation of DA [2], [5], which can be applied to our design as well.

REFERENCES

[1] B. Widrow and S. D. Stearns, Adaptive signal processing. Prentice Hall, Englewood Cliffs, NJ, 1985.

[2] S. Haykin and B. Widrow, Least-mean-square adaptive filters. Wiley-Interscience, Hoboken, NJ, 2003.

[3] M. Meyer and P. Agrawal, ―A modular pipelined implementation of a delayed LMS transversal adaptive filter,‖ in Proc. IEEE International Symposium on Circuits and

[4] Systems, ISCAS ’09, May 1990, pp. 1943– 1946.

[5] H. Herzberg, R. Haimi-Cohen, and Y. Be’ery, ―A systolic array realization of an LMS adaptive filter and the effects of delayed adaptation,‖ IEEE Transactions on Signal Processing, vol. 40, no. 11, pp. 2799–2803 , Nov. 1992.

[6] M. Meyer and D. Agrawal, ―A high sampling rate delayed LMS filter architecture,‖ IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 40, no. 11, pp. 727–729, Nov. 1993.

[7] S. A. White, ―Applications of the distributed arithmetic to digital signal processing: A tutorial review,‖ IEEE ASSP Magazine, vol. 6, no. 3, pp. 5–19, Jul. 1989.

[8] D. Allred, H. Yoo, V. Krishnan, W. Huang, and D.

[9] Anderson, ―A novel high performance distributed arithmetic adaptive filter implementation on an FPGA,‖ in IEEE International Conference on Acoustics, Speech, and Signal

[10] Processing, 2004. Proceedings.(ICASSP’04), vol. 5, 2004, pp. 161–164.

[11] D. J. Allred, H. Yoo, V. Krishnan, W. Huang, and D. V. Anderson, ―LMS adaptive filters using distributed arithmetic for high throughput,‖ IEEE Trans. Circuits and Systems I: Regular Papers, vol. 52, no. 7, pp. 1327–1337, Jul. 2005.

[12] P. K. Meher and M. Maheshwari, ―A high-speed FIR adaptive filter architecture using a modified delayed LMS algorithm,‖ in Proc. IEEE International Symposium on

[13] Circuits and Systems (ISCAS'2011), Rio de Janeiro, Brazil, May 2011, pp. 121–124.

[14] P. K. Meher, S. Chandrasekaran, and A. Amira, ―FPGA realization of FIR filters by efficient and flexible systolization using distributed arithmetic,‖ IEEE Trans. Signal Processing, vol. 56, no. 7, pp. 3009– 3017, Jul. 2008.

AUTHORS

First Author – Manish Kumar, Dept.of Electronics and Communication, Saveetha Engineering College, Chennai, India., Manish.manu2025@gmail.com

Second Author – Dr. R.Ramesh, Dept. of Electronics and Communication, Saveetha Engineering College, Chennai, India. ramesh@saveetha.ac.in