A New Algorithm to Implement Low Complexity DCT for Portable Multimedia Devices

S. Vijay
Dept. of Instrumentation and Control Engineering,
National Institute of Technology, Trichy, India
Email: imvijays@gmail.com

A. P. Vinod
School of Computer Engineering,
Nanyang Technological University
Nanyang Avenue, Singapore
Email: asvinod@ntu.edu.sg

***Abstract*— Low complexity implementation of Discrete Cosine Transform (DCT) requires efficient reduction of multiplier complexity. The number of adders (subtractors) needed to implement the multiplier determines the complexity of the DCT implementation. In this paper, we present an efficient method to reduce the complexity of multiplication in DCT using a binary common subexpression elimination technique. Our algorithm chooses the maximum number of frequently occurring common subexpressions to eliminate redundant computations in the DCT matrix and hence reduces the number of adders required to implement the multiplications. Design example of an 8 x 8 DCT shows that our method offers complexity reduction of 22% over the best known method.**

***Keywords* —** Discrete Cosine Transform, Low Complexity, Common Subexpression Elimination, Multiplier, Adder, Logic Depth.

I. INTRODUCTION

Recently, there is high demand on video and image data transmission and storage. Image compression has become a mandatory requirement, in order to achieve high-speed data transmission, save bandwidth and to save storage space. The Discrete Cosine Transform (DCT) has been widely recognized as the most effective technique among various transform-coding methods for image and video signal compression standards such as JPEG, MPEG, H.261 and H.263 [1]. DCT has been extensively used due to its high energy compaction capability, decorrelation properties and existence of numerous fast DCT algorithms. It has become a necessity to design a low-power and high-speed DCT chip as these standards find applications on portable multimedia devices, personal digital assistants and portable communication equipments. The most power consuming element in a DCT chip is the multiplier. In CMOS technology, there are three sources of power dissipation – switching (dynamic) currents, leakage currents and short-circuit currents. Among these parameters it is found that the switching component of current, which is the function of the effective capacitance, plays the most important role [2]. Transformations such as reductions in the critical path, number of operations, and average transition activity result in architectures that minimize the effective capacitance of the circuit by reducing the power consumption [2].

With increasing real-time video processing requirement, high-speed DCT implementations use efficient dedicated hardware units but suffer from high hardware cost [3]. The multipliers used in the systolic array based designs [4, 5] consume a large silicon area, and hence these designs are not power efficient. An effort to overcome the drawback of the systolic array designs let to the development of ROM based designs of [6, 7] in which the complexity of multiplication is reduced by employing efficient ROM access operations. However, the most crucial component in low-power implementations is not considered in the designs of [3-7] which is the optimization of the computationally intensive multiplication of the input data (image) with the DCT matrix. Based on the concept of Common Subexpression Elimination (CSE) presented in [8] to eliminate multiple constant multiplications, a CSE method using the Canonic Signed Digit (CSD) representation of DCT matrix has been proposed in [9]. The goal of Hartley's CSE [8], which was originally proposed for digital filters, is to identify multiple occurrences of identical bit patterns (called common subexpressions) that are present within each filter coefficient. Since the computation of multiple identical expressions needs to be implemented only once, the resources necessary for these operations can be shared. In [9], the CSE technique was reformulated in the context of DCT. While the conventional low complexity DCT implementation methods focus on reducing the number of inter-structure adders, the CSE method in [9] focused on minimizing the number of intra-structure adders, which is the most power-consuming component in a DCT chip. However, the CSE method in [9] does not maximize the subexpression formation as it groups subexpressions sequentially, i.e., subexpressions are chosen as and when they occur and without employing a look-ahead. Therefore, this approach would produce many unpaired nonzero bits which require additional adders to be implemented.

In this paper, we propose a new CSE algorithm using binary representation of the DCT matrix for reducing the number of intra-structure adders in DCT implementation. Recently, we proposed a CSE method in [10, 11] based on binary representation of filter coefficients which produced better adder reductions compared to conventional CSD-based CSE methods in realizing digital filters. Basically, we extend the idea in [10, 11] proposed for digital filters to implement low complexity DCT in this paper. With the combination of Binary Horizontal Subexpression Elimination (BHSE) and Binary Vertical Subexpression Elimination (BVSE), our algorithm eliminates redundant computations to implement the DCT. Our algorithm maximizes the grouping of bits to form subexpressions, which results in fewer ungrouped bits and correspondingly fewer adders to implement the DCT multipliers.

The rest of the paper is organized as follows. In Section II, we analyze the complexity of multiplication in the DCT. In Section III, a review of the conventional subexpression sharing technique is given. In Section IV, the proposed BSE procedure is presented. A design example is shown in Section V. Section VI provides our conclusions.

II. DCT MULTIPLIER COMPLEXITY

The $N \times N$ DCT matrix $C = c(k,n)$ is defined as [1]:

$$c(k,n) = \begin{cases} \dfrac{1}{\sqrt{N}} & \text{for } k = 0 \text{ and } 0 \le n \le N-1 \\ \sqrt{\dfrac{2}{N}}\cos\left(\dfrac{\pi(2n+1)k}{2N}\right) & \text{for } 1 \le k \le N-1 \text{ and } 0 \le n \le N-1 \end{cases} \quad (1)$$

Using matrix notation, the DCT coefficients (*Y*) is obtained from $Y = CU$:

$$Y = \begin{bmatrix} c_{11} & c_{12} & c_{13} \cdot\cdot\cdot & c_{1N} \\ c_{21} & c_{22} & c_{23} \cdot\cdot\cdot & c_{2N} \\ . & . & . \; . \; . \; . \; . & . \\ . & . & . \; . \; . \; . \; . & . \\ c_{N1} & c_{N2} & \cdot\cdot\cdot & c_{NN} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ u_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_N \end{bmatrix} \quad (2)$$

where u_i is the input data matrix and c_{ij} represent the elements of the DCT matrix. The multiplication of the variable (image data, u_i) with the constant (DCT matrix elements, c_{ij}) is implemented using *shifts* and *adds* by representing the constant in CSD.

The adders used for computing the sum of the products, $\sum c_{ii}.u_i$ to obtain the output matrix *Y* are called the inter-structural adders whereas the adders used for computing the products $c_{ii}.u_i$ are called intra-structural adders. For an $N \times N$ DCT, the number of inter-structure adders, $N_{\text{int}er}$, needed to compute $\sum c_{ii}.u_i$ is $N(N-1)$. The objective of the DCT implementation methods [3]-[7] was to reduce the number of inter-structure adders in DCT implementation. However, the actual cost of DCT implementation is dominated by the cost of multipliers required to compute the products, $c_{ii}.u_i$, i.e., the number of intra-structure adders. For example, consider the example of the multiplication to obtain the first term of (2), $y1(1) = c_{11}.u_1$. Assume c_{11} = 0.6458 = 0.101001010101. The product $y1(1)$ can be expressed as

$$y1(1) = 2^{-1}u_1 + 2^{-3}u_1 + 2^{-6}u_1 + 2^{-8}u_1 + 2^{-10}u_1 + 2^{-12}u_1 \quad (3)$$

The adders used to compute (3) are called intra-structure adders. In this case, 5 intra-structure adders are required to obtain $c_{11}.u_1$ as in (3), which is one less than the number of nonzero bits in c_{11}.

If B_{ij} is the number of nonzero bits in the CSD (or binary) representation of c_{ij}, the number of intra-structure adders, $N_{\text{int}ra}$, is given by

$$N_{\text{int}ra} = \sum_{i=1}^{N}\sum_{j=1}^{N}(B_{ij} - 1) \quad (4)$$

The value of $N_{\text{int}ra}$ is substantially larger than $N_{\text{int}er}$. Therefore, a more effective goal of reducing the number of computations (for reducing power) in DCT implementation is to reduce the number of intra-structural adders, i.e., $N_{\text{int}ra}$. The CSD-based CSE method in [9] addressed the problem of reducing the intra-structure adders in realizing DCT. Though [9] achieved good reduction of intra-structure adders, we note that there is still scope to further reduce the hardware complexity.

III. REVIEW OF CSE METHOD

The goal of CSE methods for digital filters is to identify multiple occurrences of identical bit patterns (called common subexpressions) that are present within each filter coefficient. Since the multiplication of Common Subexpressions (CSs) with inputs signal needs to be implemented only once, the resources necessary for these computations can be shared. Each CSE algorithm proposed in literature has its own unique method of subexpression elimination process so as to maximize the reduction of adders used to compute the sum of partial products. The

CSE approach proposed for digital filters has been reformulated for DCT implementation in [9] – This is summarized below.

The expressions for the output Y in (2) can be written as

$$\begin{aligned} y_1 &= c_{11}.u_1 + c_{12}.u_2 + \ldots + c_{1N}.u_N \\ y_2 &= c_{21}.u_1 + c_{22}.u_2 + \ldots + c_{2N}.u_N \\ &. \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ y_N &= c_{N1}.u_1 + c_{N2}.u_2 + \ldots + c_{NN}.u_N \end{aligned} \tag{5}$$

The objective of CSE method is to minimize $N_{\text{int}ra}$ required for computing the products, $c_{ij}.u_i$. From (5), it can be noted that each data needs to be multiplied with several constants. For example, the data u_1 is multiplied with $c_{11}, c_{21}, \ldots . c_{N1}$, which can be written as

$$u_1 . \left[c_{11} \;\; c_{21} \ldots\ldots c_{N1}\right]^T \tag{6}$$

By sharing the CSs that exist in the matrix elements c_{ij} in (6), $N_{\text{int}ra}$ can be reduced. Thus, in order to obtain the products $c_{ij}.u_i$, the DCT computation is reformulated as [9]:

$$\begin{aligned} &u_1 . \left[c_{11} \;\; c_{21} \ldots\ldots c_{N1}\right]^T \\ &u_2 . \left[c_{12} \;\; c_{22} \ldots\ldots c_{N2}\right]^T \\ &. \quad . \quad . \quad . \quad . \quad . \quad . \\ &u_N . \left[c_{1N} \;\; c_{2N} \ldots\ldots c_{NN}\right]^T \end{aligned} \tag{7}$$

The basic idea in [9] was that, using the CSs in c_{ij}, (7) can be computed efficiently, i.e., with fewer intra-structure adders. The CSE method [9] was based on CSD representation of c_{ij}. In this paper, we show that the intra-structure adders can be further reduced using a CSE method based on binary representation of c_{ij}.

IV. Proposed CSE Method

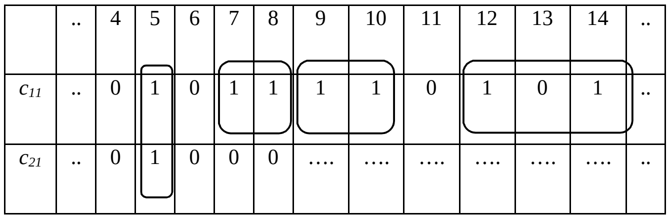
In this section, we explain the procedure of proposed binary representation based CSE method. We call our method, Binary Subexpression Elimination (BSE). First, a program is run to identify the most commonly occurring subexpressions in binary representation of c_{ij}. We use two types of CSs: (1) Horizontal Common Subexpressions (HCSs), i.e., bit patterns that occur within each element of the DCT matrix and (2) Vertical Common Subexpressions (VCSs), i.e., bit patterns that occur across adjacent elements of the DCT matrix. Based on statistical analysis of c_{ij} s for various DCT sizes, we found that the most frequently occurring HCSs are [1 1], [1 0 1], [1 1 1], and VCS is [1 1]. Therefore, we use these HCSs and VCSs in our proposed BSE algorithm. The proposed BSE procedure is as follows.

Step 1: Obtain the DCT matrix elements c_{ij} for each data element using (1).
Step 2: Obtain the binary representation of the decimal values of the c_{ij} for the desired word length.
Step 3: Let *z* represent the DCT matrix element and *w*, the bit (the shift) analyzed. Set *z = w = 1* so as to begin with the Most Significant Bit (MSB) of the first coefficient (c_{11}).

Step 4: The program checks for nonzero bits at *(z, w), (z+1, w), (z, w+2) and (z+1, w+2).*

Case 1: When *(z, w), (z, w+1)* or *(z, w+2)* are *(z+1, w)* are nonzero, i.e., when there is a horizontal and a vertical pattern at *(z, w)*, then:

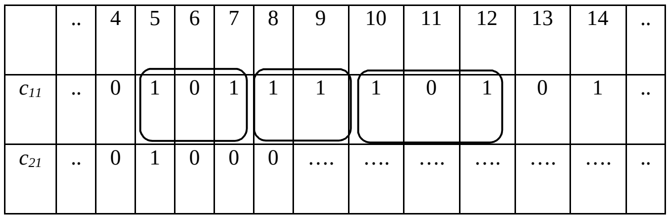
(a) First the VCS at *(z, w)* is considered. The number of CSs and the bits that cannot form CSs are determined for the remaining bits in the *z* coefficient as shown in Fig. 1 (a).



	..	4	5	6	7	8	9	10	11	12	13	14	..
c_{11}	..	0	1	0	1	1	1	1	0	1	0	1	..
c_{21}	..	0	1	0	0	0	….	….	….	….	….	….	..

Fig. 1(a). Grouping of subexpressions with VCS given preference (case 1).

(b) Then the HCS at *(z, w)* is considered, and the same procedure as (a) is followed as shown in Fig. 1 (b).



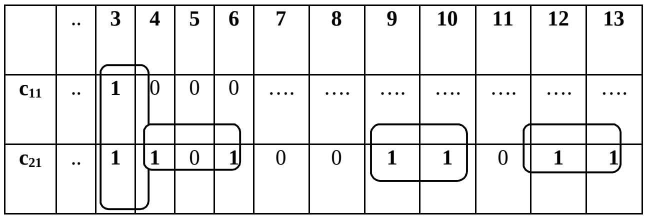
	..	4	5	6	7	8	9	10	11	12	13	14	..
c_{11}	..	0	1	0	1	1	1	1	0	1	0	1	..
c_{21}	..	0	1	0	0	0	….	….	….	….	….	….	..

Fig. 1(b). Grouping of subexpressions with HCS given preference (case 1).

The number of CSs and ungrouped bits are compared for both these procedures and the one with the largest number of CSs is chosen to pair up the rest of the bits in *(z)* coefficient. For example, in the case shown in Fig. 1 (a), we get four CSs with no bits ungrouped. But in the case of Fig. 1(b), we get three subexpressions and two ungrouped bits. Hence the VCS method is used to pair in this case. If the number of subexpressions and ungrouped bits are the same, then the procedure which considers the HCS is implemented. Depending on whether the VCS or HCS at *(z, w)* is chosen to group and form subexpressions for the *(z)* coefficient, increment w correspondingly. If $w \le N-1$, go to step 4. Otherwise go to step 5.

Case 2: A similar procedure as illustrated above is used when *(z, w), (z+1, w)* are *(z+1, w+1)* or *(z+1, w+2)* are nonzero, i.e., when there is a HCS at *(z+1, w)* and a VCS at *(z, w).*

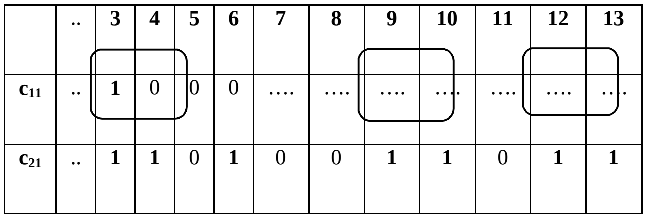
(a) First, the VCS at *(z, w)* is considered. The number of subexpressions and the bits that cannot form CSs are determined for the remaining bits in the *(z+1)* coefficient as shown in Fig. 2 (a).



	..	3	4	5	6	7	8	9	10	11	12	13
c_{11}	..	1	0	0	0	….	….	….	….	….	….	….
c_{21}	..	1	1	0	1	0	0	1	1	0	1	1

Fig. 2(a). Grouping of subexpressions with VCS given preference (case 2).

(b) Then the HCS at *(z+1, w)* is considered, and the same procedure as (a) is followed as shown in Fig. 2 (b)



	..	3	4	5	6	7	8	9	10	11	12	13
c_{11}	..	1	0	0	0	….	….	….	….	….	….	….
c_{21}	..	1	1	0	1	0	0	1	1	0	1	1

Fig. 2 (b). Grouping of subexpressions with HCS given preference (case 2).

The number of CSs and ungrouped bits are compared for both these procedures and the one with the largest number of patterns is chosen as the method to pair up the rest of that (z+1) coefficient. For example, in the case shown in Fig. 2 (a), we get three subexpressions with no bits ungrouped. But in the case of Fig. 2 (b), we get three subexpressions and two ungrouped bits. Hence the VCS method is used to pair in this case. If the number of subexpressions and ungrouped bits are the same, then the procedure which considers the HCS is implemented. Depending on whether the VCS at *(z, w)* or the HCS at *(z+1, w)* is chosen to group and form CSs for the *(z+1)* coefficient, increment w correspondingly. If $w \le N-1$, go to step 4. Otherwise go to step 5.

Case 3: When only *(z, w)* and *(z, w+1)/(z, w+2)* are nonzero, i.e., when there is a HCS only and no VCS at *(z, w)*, then select the HCS. Increment w correspondingly. If $w \le N-1$, go to step 4. Otherwise go to step 5.

Case 4: When only *(z, w)* and *(z+1, w)* are nonzero, i.e., when there is a VCS and no HCS at *(z, w)*, then select the VCS. Increment w correspondingly. If $w \le N-1$, go to step 4. Otherwise go to step 5.

Case 5: For any other combinations, w is just incremented by one. If $w \le N-1$, go to step 4. Otherwise go to step 5.

Step 5: When $w > N-k$, where k is the length of the pattern that is checked, set $w = 1$ and increment z by one, go to step 4. When $w > N-k$ and $z = N$, go to step 6.

Step 6: Once the HCSs and the VCSs are grouped, the coefficients are now checked for Super-Subexpressions (SSs), i.e., patterns that have more bits like [1 1 1 1], [1 0 1 0 1] , [1 1 0 1] , [1 0 1 1]. The SSs are implemented only if they occur at least twice in the DCT coefficient matrix. As and when these SSs are implemented, depending on the pattern, increment w accordingly, and go to step 5. When $w > N-k$, set $w = 1$ and increment z by one, and go to step 7. When $w > N-k$, and $z = N$, go to step 6.

Step 7: When the SSs are grouped, another look-ahead method is implemented wherein a check is made as to whether there is a better way to group the nonzero bits. This is because the SSs tend to increase the logic depth (number of adder-steps in a maximal path of decomposed multiplications) of the multiplier which in turn would increase the multiplier delay. Thus the logic depth (LD) is kept under check as and when the patterns are selected. After eliminating a pattern, increment w according to the size of the pattern selected, and go to step 7. When $w > N-k$, set $w = 1$ and increment z by one, go to step 7. When $w > N-k$, and $z = N$, terminate the program.

V. DESIGN EXAMPLE

In this section, we present the design of a 8 X 8 DCT using our BSE procedure. Due to space constraints, we only provide the detailed implementation of $u_1.c_{i1}$ for $i = 1$ to 8 given by (6), and the same procedure is adopted to implement other terms of (7). The first column of the DCT matrix is [c_{11} c_{21} c_{31} ….. c_{71} c_{81}]^T . The decimal values of the elements in first column are shown in Fig. 3.

c_{11}	0.3536	c_{51}	0.3536
c_{21}	0.4904	c_{61}	0.2778
C_{31}	0.4619	c_{71}	0.1993
c_{41}	0.4157	c_{81}	0.0975

Fig. 3. DCT matrix elements for data element u_1.

We then obtain the binary representation of the DCT coefficients [c_{11} c_{21}….. c_{71} c_{81}]^T which is shown in Fig. 4.

The numerals in the first row of Fig. 4 represent the number of bit-wise right shifts whereas the first column is the DCT coefficients corresponding to data element u_1.

	1	2	3	4	5	6	7	8	9	10	11	12
c_{11}	0	1	0	1	1	0	1	0	1	0	0	0
c_{21}	0	1	1	1	1	1	0	1	1	0	0	0
c_{31}	0	1	1	1	0	1	1	0	0	1	0	0
c_{41}	0	1	1	0	1	0	1	0	0	1	1	0
c_{51}	0	1	0	1	1	0	1	0	1	0	0	0
c_{61}	0	1	0	0	0	1	1	1	0	0	0	1
c_{71}	0	0	1	1	0	0	0	0	1	1	1	1
c_{81}	0	0	0	1	1	0	0	0	1	1	1	1

Fig. 4. Binary representation of c_{ij}.

The HCSs $u_{2,1}$ = [1 0 1] , $u_{4,1}$ = [1 1] , $u_{5,1}$ = [1 1 1] and the VCS $u_{3,1}$ = [1 1] are indicated inside rectangles in Fig. 4. Fig. 5 is obtained from Fig. 4 by substituting the respective pattern numbers in the respective positions where $u_{2,1}$ = 2, $u_{3,1}$ = 3, $u_{4,1}$ = 4, and $u_{5,1}$ =5. In Fig. 5, patterns like [2 0 0 2], [4 0 0 2] etc. are grouped to form the Horizontal Super-Subexpressions (HSSs). Similarly, patterns like $[5\ 5]^T$, $[4\ 4]^T$ etc. are grouped to form the vertical supersubexpressions (VSSs). Fig. 5 is simplified and represented in Fig. 6 with pattern numbers assigned to the supersubexpressions.

	1	2	3	4	5	6	7	8	9	10	11	12
c_{11}	0	2	0	0	2	0	0	0	3	0	0	0
c_{21}	0	5	0	0	4	0	0	1	0	0	0	0
c_{31}	0	5	0	0	0	4	0	0	0	3	0	0
c_{41}	0	4	0	0	2	0	0	0	0	0	1	0
c_{51}	0	3	0	4	0	0	2	0	0	0	0	0
c_{61}	0	0	0	0	0	5	0	0	0	0	0	3
c_{71}	0	0	4	0	0	0	0	0	5	0	0	0
c_{81}	0	0	0	4	0	0	0	0	5	0	0	1

Fig. 5. Representation of c_{ij} of after taking HCSs and VCSs.

	1	2	3	4	5	6	7	8	9	10	11	12
c_{11}	0	7	0	0	0	0	0	0	3	0	0	0
c_{21}	0	9	0	0	8	0	0	1	0	0	0	0
c_{31}	0	0	0	0	0	0	0	0	0	3	0	0
c_{41}	0	6	0	0	0	0	0	0	0	0	1	0
c_{51}	0	3	0	6	0	0	0	0	0	0	0	0
c_{61}	0	0	0	0	0	5	0	0	0	0	0	3
c_{71}	0	0	8	0	0	0	0	0	9	0	0	0
c_{81}	0	0	0	0	0	0	0	0	0	0	0	1

Fig. 6. Final representation of c_{ij} after subexpression formation.

Fig. 6 shows the final implementation of the DCT matrix for the input data element u_1. Note that $u_{6,1}$ = [4 0 0 2] = 6, $u_{7,1}$ = [2 0 0 2] = 7, $u_{8,1} = \begin{bmatrix}40\\04\end{bmatrix} = 8$ and $u_{9,1} = \begin{bmatrix}5\\5\end{bmatrix} = 9$. It can be determined from Fig. 4 that a total of 31 intra-structural adders are required to obtain (6) using the direct method (direct method means implementation using shifts and adds and without any subexpressions). Using the CSD-based CSE method [9], 16 adders are required, which is a reduction of 48%. From Fig. 6, using our BSE method, the output y_1 corresponding to the first data element u_1 is

$$\begin{aligned} y_1 &= 2^{-2}u_7 + 2^{-9}u_3 + 2^{-2}u_{10} + 2^{-5}u_8 + 2^{-8}u_1 \\ &+ 2^{-10}u_3 + 2^{-2}u_6 + 2^{-11}u_1 + 2^{-2}u_3 + 2^{-4}u_6 \\ &+ 2^{-6}u_5 + 2^{-12}u_3 + 2^{-3}u_8 + 2^{-9}u_{10} + 2^{-12}u_1 \end{aligned} \tag{8}$$

The proposed BSE method requires only 14 intra-structural adders (7 adders for the actual realization and 7 adders for the subexpressions) which is 54% reduction over the direct CSD method and 12.5% reduction over the CSE method in [9]. Using the proposed BSE technique, the total number of intra-structure adders required to obtain all the products of the 8 x 8 DCT is 102, whereas the adder requirement is 130 for the previously proposed CSE method [9] and 210 in the direct CSD implementation. Thus, the adder reduction achieved using our BSE is 22% over the CSE method [9] and 51% over the direct CSD method. The critical path length and the transition activity also need to be minimized apart from reducing the number of additions. Moreover, it can also be seen that in the proposed method, with the increase in the order of the DCT matrix, the scope for subexpression formation also increases proportionately, thereby increasing the reductions in hardware required to implement it. The logic depth of the multiplier implemented using our method is 4 adder-steps, which is same as that in [9]. Thus our BSE achieves adder reduction without increasing the delay.

VI. CONCLUSIONS

We have proposed an efficient look-ahead binary representation based CSE algorithm low complexity implementation of a DCT chip. Our implementation method gives more importance to reduce the number of intra-structural adders, which is the most power-consuming component in a DCT chip. As binary representation has a higher frequency of occurrence of common subexpressions, this concept has been efficiently used to produce better reduction of adders and shifts without affecting the speed of the multiplier.

References

[1] K. R. Rao and J. J. Hwang, *Techniques and Standards for Image, Video and Audio Coding,* Englewood Cliffs, NJ: Prentice-Hall, 1996.

[2] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Brodersen, "Optimizing power using transformations," *IEEE Trans. On CAD,* vol. 14, no. 1, pp. 12-31, Jan. 1995.

[3] P. Pirsch, N. Ranganathan, "VLSI architectures for video compression-A survey," *Proc. IEEE,* vol. 83, pp. 220-246, Feb. 1995.

[4] L. W. Chang and M. C. Wu, "A unified systolic array for discrete cosine and sine transforms," *IEEE Trans. Signal Processing,* vol. 39, pp. 192-194, Jan. 1991.

[5] J. I. Guo, C. M. Liu, and C. W. Jen, "A new array architecture for prime length discrete cosine transform," *IEEE Trans. Signal Processing,* vol. 41, no. 1, pp. 436-442, 1993.

[6] D. Slawecki and W. Li, "DCT/IDCT processor design for high data rate image coding," *IEEE Trans. Circuits Syst. Video Technology,* vol. 2, pp. 135-146, June 1992.

[7] J. I. Guo, C. M. Liu, and C. W. Jen, "The efficient memory-based VLSI arrays for DFT and DCT," *IEEE Trans. Circuits Syst. II,* vol. 39, no. 10, pp. 723-733, 1992.

[8] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II,* vol. 43, pp. 677-688, Oct. 1996.

[9] A. P. Vinod and E. M-K. Lai, "Hardware efficient DCT implementation for portable multimedia terminals using subexpression sharing," "in *Proc. of IEEE TENCON,* Thailand, November 2004.

[10] R. Mahesh and A. P. Vinod, "A new common subexpression elimination algorithm for implementing low complexity FIR Filters in software defined radio receivers," *Proc. of IEEE International Symposium on Circuits and Systems,* vol. 4, pp. 4515-4518, Island of Kos, Greece, May 2006.

[11] R. Mahesh and A. P. Vinod, "A new common subexpression elimination algorithm for realizing low complexity higher order digital filters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits & Systems,* (In Press).