An Efficient Realization of Forward Integer Transform in H.264/AVC Intra-frame Encoder

Muhammad Nadeem, Stephan Wong, and Georgi Kuzmanov
Computer Engineering Laboratory
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
E-mail: {M.Nadeem, J.S.S.M.Wong, G.K.Kuzmanov}@tudelft.nl

***Abstract*—The H.264/AVC intra-only frame encoder, for its excellent encoding performance, is well-suited for image/video compression applications such as *Digital Still Camera (DSC), Digital Video Camera (DVC), Television Studio Broadcast* and *Surveillance video.* The *forward integer transform* is an integral part of the H.264/AVC video encoder. In this paper, for image compression applications running on battery-powered electronic devices (such as *DSC*), we propose a low-power, area-efficient realization of the *forward integer transform*. The proposed solution reduces the number of operations by more than 50% (30 vs. 64) and consumes significantly less dynamic power when compared with existing state-of-the-art designs for the *forward integer transform*. For video compression applications such as *Television Studio Broadcast or Surveillance Videos*, where throughput is more important, we propose a low-latency, area-efficient realization of the *forward integer transform* unit in the intra frame processing chain. With the proposed solution, the effective latency for *forward integer transform* is drastically reduced, as the processing unit is no longer on the critical path of the intra-frame processing chain. Moreover, the proposed solution requires half the numbers of operations for its hardware implementation, when compared with existing state-of-the-art designs for forward integer transform.**

Keywords- H.264/AVC; Forwad integer transform; Hadamard transform; Intra frame encoder.

I. INTRODUCTION

The Advanced Video Coding standard H.264/AVC, also known as MPEG-4 part 10, is jointly developed by ITU-T VCEG and ISO/IEC MPEG [1]. It is able to achieve significantly higher compression efficiency over the previous video coding standards, like H.263 and MPEG-2/4. The H.264/AVC provides similar subjective video quality as that of MPEG-2 with at least 50% more reduction in bit-rate [2], [3]. It provides up to 30% better compression when compared with H.263+ and MPEG-4 Advanced Simple Profile (ASP) [4]. This significantly higher compression efficiency is achieved at the cost of additional computational complexity of the video coding algorithms in H.264/AVC, approx. 10 times higher relative to MPEG-4 simple profile [4].

The H.264/AVC supports multiple directional *intra prediction modes* (4 modes for the 16x16 block type and 9 modes for the 4x4 block type) to reduce the spatial redundancy in the video signal. These multiple *intra prediction modes* help to significantly improve the encoding performance of an H.264 *intra-frame encoder*. Several studies [6][17] have shown that H.264/AVC outperforms JPEG2000, a state-of-the-art still image coding standard, in terms of subjective as well as objective image quality. This makes *H.264/AVC intra-frame encoder,* an attractive choice for an image compression engine. Applications like *Digital Still Camera* (*DSC*) can employ *intra-frame encoding technique* to compress high-resolution images.

In a video frame encoded in intra mode, the current macro-block (*MB)* is predicted from the previously encoded neighboring *MBs* from the same video frame. Therefore, an *intra-frame* with all *intra-MB,* does not depend on any other video frame(s) and can be decoded independently. A video encoded with *intra-frames* only is easier to edit than video with predicted frames. Similarly in many surveillance systems, the video is compressed using intra frames encoding mode due to legal reasons. Courts do not accept the predicted image frames as legal evidence in many countries [5]. As a result, a typical video surveillance system compresses video using intra encoded frames only. Consequently, intra-only video coding is widely used coding technique in *Television Studio Broadcast*, *Digital Cinema* and *Surveillance* video applications.

A high-throughput *intra-frame processing chain* is, therefore, an important requirement for *H.264/AVC intra-frame encoder* for real time video processing applications. Similarly a low-power solution is more important for battery-powered devices such as *Digital Still Camera* (*DSC*) applications. .

In this paper, we focus on the realization of a *forward integer transform* unit in an *H.264/AVC intra-frame encoder* and provide solutions for two application domains. The first solution targets a low-power realization of the *forward integer transform* requirement for battery-powered electronic devices. The second solution targets high performance computing, where the main objective is to reduce latency and complexity of the *forward integer transform* unit in the *intra-frame encoder* processing chain. In this context, we provide the following specific contributions:

- A proposal to derive the *forward integer transformed coefficients* directly from that of the *forward Hadamard transform* unit rather than from residual pixel block data. This approach reduces the number of addition operations by more than half (30 vs. 64) and, therefore, the hardware implementation can potentially consumes less dynamic power.

- A proposal for low latency, low complexity *forward integer transform* unit by aggressively re-using the intermediate results from *Hadamard transform* unit. The proposed solution reduces the number of operations by 50% (32 vs 64) and provides zero penalty to the *intra-frame encoder* critical path.

The remainder of this paper is organized as follows. The background and related work for the problem is presented in Section II. Section III describes the proposed solutions in detail whereas Section IV provides comparison results. Finally, Section V concludes this paper.

II. Background and Related Work

This section briefly describes the background knowledge for *H.264/AVC intra-frame encoder* to clearly identify the problem. The related work in the literature is provided further in this section.

The functional block diagram for H.264/AVC *intra-frame encoder* is depicted in Fig. 1. The *Intra-mode Prediction, Forward Integer Transform (T), Quantization (Q), Inverse Quantization (IQ)* and *Inverse Integer Transform (IT)* is the encoding loop for the *intra-frame encoder*. All of these processing units constitute the intra-frame processing chain and are on the critical path of the H.264/AVC *intra-frame encoder*.

The H.264/AVC *intra-mode prediction* supports multiple prediction modes (4 modes for the 16x16 block type and 9 modes for the 4x4 block type) to exploit the spatial correlation between the block to be encoded and its already encoded neighboring blocks to significantly reduce the predictive error. When a block is to be encoded in intra mode, a corresponding prediction block is formed using previously encoded and reconstructed blocks. The mode decision unit determines the best mode for the block at hand. The reference software for H.264/AVC encoder utilizes a *Lagrange cost function* to make an intra prediction mode decision. The cost for any given intra mode is computed as follows:

$$J_{Mode}(MB_k,I_k|QP,\lambda_{Mode}) = D(MB_k,I_k|QP) + \lambda_{Mode} * R(MB_k,I_k|QP) \quad (1)$$

Where λ_{Mode}, *D* and *R* represents the *Lagrange parameter*, *distortion* and *rate,* respectively. The rate (*R*) is estimated by the number of bits to encode the mode whereas the distortion (*D*) is measured using the *Sum of Absolute Difference* (*SAD*) or the *Sum of Absolute Transformed Differences* (*SATD*) of the residual block data. The *SATD*, however, is more precise in estimating the distortion and provides better encoding video quality. The H.264/AVC reference software uses the *forward Hadamard transform* to compute the frequency domain coefficients for the candidate *intra-prediction mode* [7]. The *distortion* (D) in Eq. (1) is estimated by using the absolute sum of all these Hadamard transformed coefficients.

The intra prediction process of a block requires the reconstructed pixels of the (Left and/or Top) neighbor blocks. Therefore, the intra prediction process can not be started until

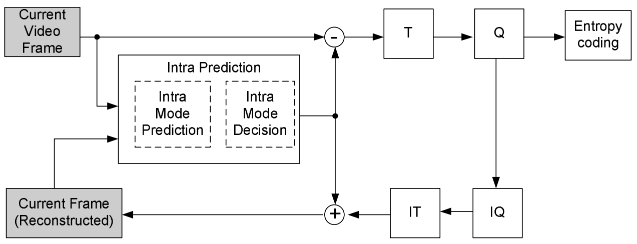

Fig. 1 Functional blocks for H.264 Intra Frame Encoder

these reconstructed pixels of the neighbor blocks are not available for prediction. The reconstruction process involves the *Forward Integer Transform (T)*, *Quantization* (*Q*), *Inverse Quantization* (*IQ*) and *Inverse Integer Transform (IT)* processing units. Consequently, low-latency is an important requirement for all of these processing units for the development of an efficient *intra-frame encoder*.

In the last few years many researchers proposed a number of optimized algorithms and efficient architectures for intra frame processing chain. A number of algorithms [21] – [23] have been proposed in the literature to reduce the complexity and computational load of the intra prediction and mode decision units. These fast algorithms either restrict the number of possible candidate modes by exploiting different characteristics (edge orientation, correlation, etc.) of the video block in the video frame and/or compute only partial cost. Similarly for *forward integer transform* unit, the major focus of research has been to develop fast algorithms with minimal area requirement for its hardware implementation. In [8], Liu, et al., used a parallel register array to realize the transpose operation for the 2-D 4x4 *forward integer transform*. In [9], Cheng proposed high-throughput 4x4 *integer transform* architecture with no transpose memory requirement. Fan, et al., proposed a fast 2-D transform algorithm in [10]. Similarly in [11] – [13], a number of fast algorithms for the *integer transform* have been proposed.

The H.264/AVC encoder utilizes multiple transforms (*Forward/Inverse*) *Integer Transform* and (*Forward/Inverse*) *Hadamard Transform*. Therefore, recently, the research focus has been shifted to design a unified transform block to support multiple transforms in H.264 [14]–[18]. The motivation for such efforts is to reduce the on-chip area requirement for the hardware implementation of these transforms units by sharing area between them. The proposed architectures although support multiple transforms in H.264/AVC, but mostly provide only one type of transform at a time. All of these architectures take residual data as an input and provide the transformed coefficients for the selected type of transform at the output.

III. Proposed Solution

In this section, we first describe the low-power and area-efficient solution for image processing applications along with corresponding hardware implementations. The low-latency, area-efficient solution for video applications is described in the end of this section.

In this paper, we followed a different approach from the related work cited in Section. II. As described earlier, the *forward Hadamrad transformed coefficients* are used to estimate the *distortion* for the candidate *intra prediction modes* [7]. As part of a low-power, low-area solution for the battery-powered electronic devices, we propose to derive the integer transformed coefficients directly from the *Hadamard transformed* coefficients for the best intra prediction mode rather than from the residual block data. The proposed solution requires less than half the number of additions (30 vs 64) when compared with those for state-of-the art-design in literature. Similarly for video processing applications with high-throughput requirement, we propose a low-latency forward integer transform realization by re-using the intermediate results from the *Hadamard transform* unit in the intra-prediction mode decision process**.** This approach helps to reduce the number of addition operations by half (32 vs. 64) and diminishes the effective latency penalty to zero, as the proposed computation for the *integer transform* is removed from the critical path.

A. Cascaded Processing Unit for Image Processing Applications

The H.264/AVC uses a 2D-DCT-based *integer transform* "C" for all 4x4 block of residual pixel data "X" and is given as follows

$$Z = CXC^{T} \tag{2}$$

where

$$C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \tag{3}$$

Similarly, the 2-D *Hadamard transformed* coefficients "Y" for a 4x4 input block "X" is computed using the *Hadamard transform* "H" as follows.

$$Y = HXH^{T} \tag{4}$$

where

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \tag{5}$$

If "S" is such a transform that it can be used to convert the *forward Hadamard transform* coefficients "Y" into *forward integer transform* coefficients "Z", then for 1-D case, we can write

$$C = SH \Rightarrow S = CH^{-1} \tag{6}$$

and "S" can be represented by the following:

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & -0.5 & 0 & 1.5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0.5 \end{pmatrix} \tag{7}$$

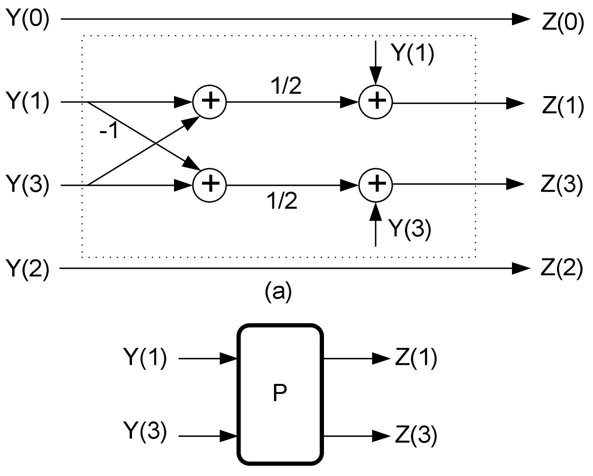

Fig. 2 (a) Signal flow diagram for 1-D S transform, (b) Basic processing block for 1-D S transform

The proposed 1-D transform requires 4 additions and 2 shift operations to generate the *integer transform* coefficients from the *Hadamard transform* coefficients data. The corresponding signal flow diagram for 1-D case is depicted in Fig. 2. The fast 1-D *forward integer transform* algorithm in [19] requires 8 additions along with 2 shift operations to compute the same *forward integer transform* results from intra prediction residual data.

The 2-D transform can be realized by row-column processing using two 1-D "S" transform units and a single transpose unit. This organization requires a total of 32 additions and 16 shift operations to generate *integer transform* coefficients from *Hadamard transformed* coefficients data. In case the *forward integer transform* coefficients are generated from the intra prediction residual data, the fast algorithm using row-column computation method in [13], [14] requires 64 additions and 16 shifts operations. Therefore, our proposal reduces the number of additions by half when compared with the current state-of-the-art method in literature.

The major disadvantage of the row-column method for 2-D transform is that it requires a transpose unit between successive 1-D transform operations. The transpose unit is on the critical path; therefore, it increases the latency of the processing chain and also requires significant area for its implementation. The transpose unit can be removed from the processing chain by using a direct 2-D transform method.

For 1-D processing, only two *Hadamard transformed* coefficients (Odd numbered coefficients) are processed for any given column or row in "Y" (Fig 2a). The input coefficients with even numbered rows and even numbered columns are passed unaltered. The coefficients at the cross-section of odd rows and odd columns are processed in both directions. The coefficients with either odd numbered row or odd numbered column are processed only once as part of 1-D column or row-transform operation respectively. The 2-D transform can be realized using the processing blocks “P”

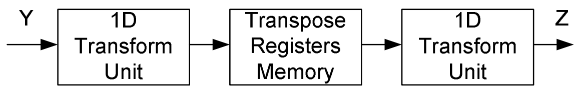

Fig. 3 Row-column processing for 2-D transform

identified in Fig. 2b. The functional block diagram for 2-D transform is depicted in Fig. 4.

The 2-D transform no longer requires a transpose unit anymore. The number of adders remains the same, i.e., 32, as for the row-column processing depicted in Fig. 3. This number can be further reduced by removing redundant operations between cascaded P processing units in Fig. 4 using the direct 2-D transform algorithm.

The direct 2-D algorithm for transform "S" in (6), can be derive as follows:

$$vec(Z) = (S \otimes S) \cdot vec(Y) \qquad (8)$$

In (8), "$\otimes$" is the Kronecker product. $vec(Y)$ is the column vector, consisting of *Hadamard transform* coefficients as input and $vec(Z)$ represents the corresponding column vector with *forward Integer transform* coefficients values as output of the direct 2-D computation approach.

We further define permutation matrices P_r and P_c, such that

$$I_{16} = P_r \cdot (P_r)^T = (P_r)^T \cdot P_r = P_c \cdot (P_c)^T = (P_c)^T \cdot P_c \qquad (9)$$

then from (8)

$$vec(Z) = P_r^T \cdot M \cdot P_c^T \cdot vec(Y) \qquad (10)$$

where $$M = P_r \cdot (S \otimes S) \cdot P_c \qquad (11)$$

$$M = \begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1.5&0.5&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&-0.5&1.5&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1.5&0.5&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&-0.5&1.5&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1.5&0.5&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&-0.5&1.5&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1.5&0.5&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&-0.5&1.5&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&2.25&0.25&0.75&0.75\\
0&0&0&0&0&0&0&0&0&0&0&0&0.25&2.25&-0.75&-0.75\\
0&0&0&0&0&0&0&0&0&0&0&0&-0.75&0.75&2.25&-0.25\\
0&0&0&0&0&0&0&0&0&0&0&0&-0.75&0.75&-0.25&2.25
\end{pmatrix}$$

$$P_r = \begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0
\end{pmatrix}
P_c = \begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0
\end{pmatrix} \qquad (12)$$

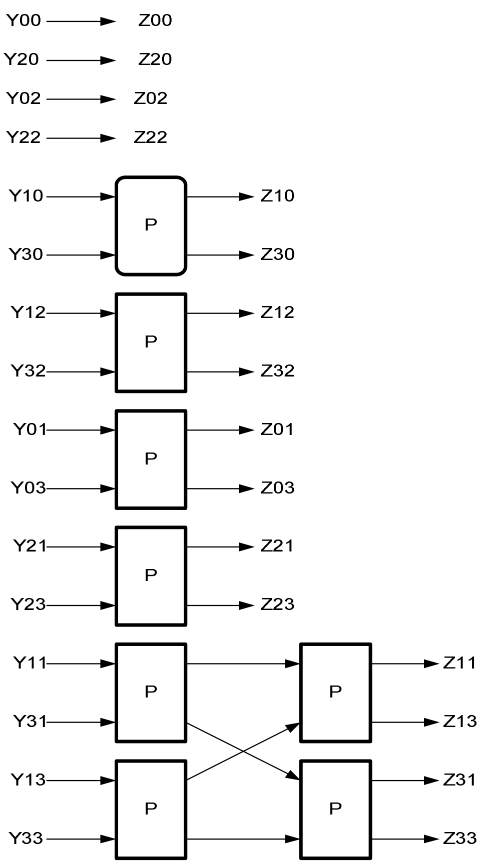

Fig. 4 Functional block diagram for 2-D transform

From (11), we can further decompose "M" as follows

$$M = \begin{pmatrix} I_4 & O_4 & O_4 & O_4 \\ O_4 & M_1 & O_4 & O_4 \\ O_4 & O_4 & M_1 & O_4 \\ O_4 & O_4 & O_4 & M_2 \end{pmatrix} \qquad (13)$$

Where

$$M_1 = \begin{pmatrix} 1.5 & 0.5 & 0 & 0 \\ -0.5 & 1.5 & 0 & 0 \\ 0 & 0 & 1.5 & 0.5 \\ 0 & 0 & -0.5 & 1.5 \end{pmatrix}; M_2 = (1/4) \bullet \begin{pmatrix} 9 & 1 & 3 & 3 \\ 1 & 9 & -3 & -3 \\ -3 & 3 & 9 & -1 \\ -3 & 3 & -1 & 9 \end{pmatrix}$$

The functional block diagram for (10) is depicted in Fig. 5(a). The signal flow graph for M_1 and M_2 is depicted in Fig 5(b) and Fig 5(c), respectively. The hardware implementation for (10) requires 30 adders in all. The number of adders are reduced to a value less than half (64 vs. 30), when compared with existing state-of the art fast 2-D *integer transform* hardware architectures in the literature. Therefore, the proposed solution not only requires less area for its hardware implementation at one hand, but additionaly consumes low dynamic power on the other, because of reduced number of adder units.

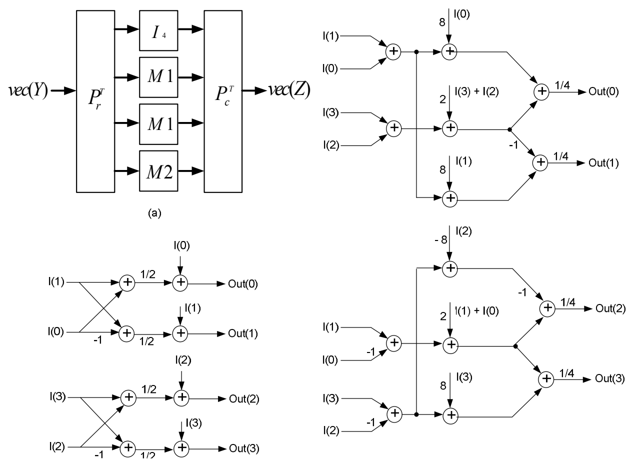

Fig. 5 (a) Functional block diagram for 2-D transform, (b) Signal flow graph for M_1, (c) Signal flow graph for M_2

B. Parallel Processing Unit for Video Processing Applications

As mentioned previously in Section I, *the intra prediction* process for the next block can not be started until the reconstructed pixels for the current block are not available for prediction. The *intra-frame* only video applications, such as *Television Studio Broadcast,* require efficient processing chain in terms of throughput for real-time processing of high-resolution videos. The *forward integer transform* is on the critical path for *intra frame encoder*. Therefore, a low-latency and area-efficient realization of *forward integer transform* is an important requirement for an efficient intra frame processing chain. The straightforward solution for low-latency can be achieved by implementing the *forward integer transform* unit in parallel with the *intra prediction mode* decision unit. Depending on the outcome of the *intra prediction mode* decision unit, the *forward integer transformed* block for the selected mode can be passed on to the next processing unit in the chain with zero latency. This approach provides an efficient solution for latency issue. The area and power consumption for such a solution can be reduced by overlapping the parallel processing units, i.e., the *forward integer transform* unit and *Hadamard transform* unit in *intra mode* decision functional block in Fig. 1. The area is reduced by re-using the common intermediate results from the *Hadamard transform* unit. The more the common intermediate results are, the more shall be the area reduction. The remainder of this section describes our approach to identify the common intermediate results and the architecture for its hardware implementation.

From (2) and (4), the 2-D 4x4 *forward integer transform* and 2-D *forward Hadamard transform* can be expressed as follows:

$$vec(Z) = (C \otimes C) \cdot vec(X) \qquad (14)$$

$$vec(Y) = (H \otimes H) \cdot vec(X) \qquad (15)$$

where "$\otimes$" is the Kronecker product, $vec(X)$ is the input column vector containing residual pixel data, $vec(Z)$ and $vec(Y)$ are the corresponding output column vectors for the *forward integer transform* coefficients and the *Hadamard transform* coefficients results, respectively.
We can write as follows:

$$vec(Z) = vec(Y) + (vec(Z) - vec(Y))$$

$$\Rightarrow \quad vec(Z) = vec(Y) + O_{offset} \qquad (16)$$

Thus the *forward integer transformed* coefficients can be derived from *Hadamard transformed* coefficients by adding an appropriate value for corresponding offset for each coefficient. Using (14), (15), and (16), the "O_{offset}" can be computed as follows:

$$O_{offset} = Q \cdot vec(X) \qquad (17)$$

where

$$Q = (C \otimes C) - (H \otimes H)) \qquad (18)$$

"Q" is the difference of Kronecker product of the *forward integer transform* and the *forward Hadamard transform,* where "H" and "C" are defined in (3) and (5), respectively.

$$Q = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\
3 & 1 & -1 & -3 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -3 & -1 & 1 & 3 \\
1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 \\
1 & -3 & 3 & -1 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 & -1 & 3 & -3 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & -3 & -1 & 1 & 3 & 3 & 1 & -1 & -3 & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & -1 & 3 & -3 & 1 & 1 & -3 & 3 & -1 & 0 & 1 & -1 & 0
\end{pmatrix}$$

We further define permutation matrices P_{r1}, P_{r2} and P_{r3} such that:

$I_{16} = P_{r1} \cdot (P_{r1})^T = (P_{r1})^T \cdot P_{r1}$, $I_{16} = P_{r2} \cdot (P_{r2})^T = (P_{r2})^T \cdot P_{r2}$ and $I_{16} = P_{r3} \cdot (P_{r3})^T = (P_{r3})^T \cdot P_{r3}$

While considering the intermediate results generated in the 2-D *Hadamard transform* processing and using permutation matrices defined above, we split "Q" in (18) into 3 matrices $\hat{Q}_1$, $\hat{Q}_2$, and $\hat{Q}_3$, such that :

$$Q_1 = P_{r1} . \hat{Q}_1 \ , \quad Q_2 = P_{r2} . \hat{Q}_2 \quad \text{and} \quad Q_3 = P_{r3} . \hat{Q}_3$$

From (17) and (18), we can write:

$$O_{offset} = (P_{r1}{}^{T} \cdot Q_1 + P_{r2}{}^{T} \cdot Q_2 + P_{r3}{}^{T} \cdot Q_3) \cdot vec(X) \qquad (19)$$

$$Q_1 = \begin{pmatrix}
1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 \\
2 & 0 & 0 & -2 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -2 & 0 & 0 & 2 \\
1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & -1 \\
1 & 0 & 0 & -1 & -2 & 0 & 0 & 2 & 2 & 0 & 0 & -2 & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \qquad (20)$$

$$Q_2 = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & 0 \\
0 & 2 & -2 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & -2 & 2 & 0 \\
0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 \\
0 & 1 & -1 & 0 & 0 & -2 & 2 & 0 & 0 & 2 & -2 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \qquad (21)$$

$$Q_3 = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 \\
1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 \\
1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0
\end{pmatrix} = \begin{pmatrix}
O_4 & O_4 & O_4 & O_4 \\
O_4 & O_4 & O_4 & O_4 \\
H & O_4 & O_4 & -H \\
O_4 & -H & H & O_4
\end{pmatrix} \qquad (22)$$

and

$$P_{r1} = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

$$P_{r2} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}; P_{r3} = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

From (20) and (21), we can observe that "Q_1" and "Q_2" compute the difference (or difference scaled by a factor of 2) between the 1st and 4th rows, or 2nd and 3rd rows of the input block"X", respectively. The difference between these rows is also computed in the 2-D *Hadamard transform* computations. Similarly, in (22), "Q_3" computes the difference between rows of 1-D *Hadamard transformed* results. Again, these difference values are also computed as part of the 2-D *Hadamard transform* computation. We can reuse these intermediate results from the 2-D *Hadamard transform* and therefore replace the input $vec(X)$ by input $vec(D)$ with difference values. Consequently, from (19) we can derive:

$$O_{offset} = (P_{r1} \cdot W_1 + P_{r2} \cdot W_2 + P_{r3} \cdot W_3) \cdot vec(D) \qquad (23)$$

where "W_1", "W_2" and "W_3" are given as follows:

$$W_1 = [C\ O_4\ O_4\ O_4]^T;\ W_2 = [O_4\ C\ O_4\ O_4]^T$$

$$W_3 = [O_4\ O_4\ I_4\ I_4]^T \qquad (24)$$

and

$$vec(D) = [D(0)\quad D(1)\quad D(2)\ \ldots\ D(15)]^T \qquad (25)$$

where

$D(0) = x_{00} - x_{30}$; $D(1) = x_{01} - x_{31}$;
$D(2) = x_{02} - x_{32}$; $D(3) = x_{03} - x_{33}$;
$D(4) = x_{10} - x_{20}$; $D(5) = x_{11} - x_{21}$;
$D(6) = x_{12} - x_{22}$; $D(7) = x_{13} - x_{23}$;
$D(8) = X_{00} - X_{03}$; $D(9) = X_{10} - X_{13}$;
$D(10) = X_{20} - X_{23}$; $D(11) = X_{30} - X_{33}$;
$D(12) = X_{01} - X_{02}$; $D(13) = X_{11} - X_{12}$;
$D(14) = X_{21} - X_{22}$; $D(15) = X_{31} - X_{32}$;

x_{ij} is a data item from the 4x4 input block containing residual pixel data and X_{ij} represents the 1-D *Hadamard transformed* coefficient for the 4x4 input block. Whereas i, j = 0, 1, 2 and 3; denotes the row/column respectively in the 4x4 input block. The signal flow diagram for (23) is depicted in Fig. 6. Consequently, the proposed design requires 20 adders for computation of O_{offset} in hardware and another 12 adders to compute the *forward integer transform* coefficients by adding the offset values to the *Hadamard transformed* coefficients. The computation unit for O_{offset} operates in parallel with the *forward Hadamard transform* unit and, therefore, it does not contribute to the overall latency of the intra frame processing chain.

IV. Design Evaluation

The two solutions proposed in this paper were implemented in VHDL and synthesized by Synopsys Design Compiler (v2002 rev05), for a maximum clock frequency of 200 MHz with 0.18 CMOS standard cell library (v1.5). Both implementations were simulated using ModelSim™ and the results were verified with those of H.264/AVC Reference Software [7]. After logic synthesis, the dynamic power consumption was estimated using Synopsys PrimePower™.

TABLE I. COMPARISON TABLE FOR SINGLE TRANSFORM

	Cheng [9]	Roman[13]	Lin[18]	Proposal 1
DPR[a] (pixels/cycle)	8	16	8	16
Transpose / Permutation	Permutation	Permutation	Permutation	Permutation
Transform Type	4x4	4x4	4x4	4x4
Technology (µm)	0.35	0.18	0.35	0.18
Latency (ns)	10.93	6.30	30.66	4.82
Speed (MHz)	91	159	33	200
Throughput (pixels/sec)	732M	2552M	261M	3200M
Area (gates)	2539	11727	15327	2638
Throughput/Area (pixels/s)	288K	218K	17K	1213K
Power (mW)	NA	NA	NA	3.7[c] @ 31.25MHz
Throughput/Power (Mpixels/s/mW)	NA	NA	NA	135

TABLE II. COMPARISON TABLE FOR MULTIPLE TRANSFORM

	Huang[16]	Fan[14]	Woong[15]	Proposal 2
DPR[a] (pixels/cycle)	8	4	16	16
Transpose / Permutation	Transpose	Transpose	Permutation	Permutation
Transform Type	Multiple	Multiple	Multiple	Multiple
Technology (µm)	0.18	0.18	0.18	0.18
Latency (ns)	20.00	16.00	5.00	4.92
Speed (MHz)	50	125	200	200
Throughput (pixels/sec)	400M	500M	3200M	3200M
Area (gates)	39800	6458	63618	9926
Throughput/Area (pixels/s)	10K	77K	50K	323K
Power (mW)	38.7 @ 50MHz	9.1 [b] @ 62.5MHz	86.9 @ 200MHz	78.52 @ 200MHz, 4.08 @ 16MHz
Throughput/Power (Mpixels/s/mW)	10.34	54.9	36.8	40.75

[a] Data Processing Rate

[b] Power consumption by the transpose register, estimated by PrimePower is 4.102 mW [15]

[c] The result computed for same throughput as that of [14]

Our first solution, for the *forward integer transform*, targets image processing applications running on battery-powered electronic devices, such as *DSC*. The comparison with the existing single 4x4 *forward integer transform* solutions from the literature is provided in Table 1. From the comparison with other solutions, we suggest that the proposed solution provides the minimum latency penalty (4.82ns) among all solutions in Table 1. and also requires significantly less area (2.6K gates) in terms of equivalent gate count for its hardware implementation. Therefore, it provides up to 5 times better performance in terms of throughput/area ratio for the same process technology. This is achieved by significantly reducing the number of addition operations (30 vs. 64) and higher data processing rate (16 vs. 4) for the realization of the *forward integer transform*. The power consumption for the other single 4x4 *forward integer transform* solutions is not available, therefore, we cannot compare the dynamic power consumption results.

Our second solution targets real-time video processing applications, such as *Television Studio Broadcast or Surveillance videos*, where throughput is more important. We proposed a low-latency, area-efficient realization of *forward integer transform* unit in the intra frame processing chain. With this proposed solution, the effective latency penalty for the *forward integer transform* unit is reduced to zero, as the computation for the *forward integer transform* is no longer on the critical path. For comparison with multiple transform solutions, we merged the 4x4 *forward Hadamard transform* unit with our proposal. The comparison results for multiple transform solution are provided in Table 2. In additions to zero latency penalty in the intra-frame processing chain, the proposed solution provides up to 30 times better performance in terms of throughput/area ratio. This is achieved by aggressively reusing the intermediate results from the *Hadamard transform* unit to reduce the number of addition operations by 50% (32 vs. 64) for realization of the *forward integer transform*, higher maximum operating frequency (200 MHz) and data processing rate (DPR : 16 pixels/cycle). The comparison of dynamic power consumption with [14] needs some clarifications. The solution in [14] consumes 9.1 mW for a throughput of 250M pixels/s whereas the proposed solution in this paper consumes 78.52 mW for a throughput of 3200M pixels/s (approximately 13x higher throughput). However, the dynamic power consumption for the proposed solution for the same throughput as that of [14], is only 4.08mW.

As [13], [14], [15], and [16] have been synthesized for the same technology as our proposals, the performance and power efficiency benefits can be derived straight from Table 1 and Table 2. Regarding [9] and [18] in Table 1, further investigations suggest that our design is more performance and power efficient. Compared to [9], which is the more efficient design of the two, we identify that our proposal provides 2x higher data processing rate yet do not require a set of (8, 13-bit) multiplexer units to implement data paths with and without shift operations to realize multiplications. Therefore, the proposed solution shall not only require less area for its hardware implementation at one hand for 0.35 µm process technology, but also potentially consumes low dynamic power on the other, because of reduced number of processing units.

V. CONCLUSION

In this paper, we proposed two solutions for realization of the *forward integer transform* in the processing chain of intra-only frame encoder applications. The first solution targeted image compression applications running on battery-powered

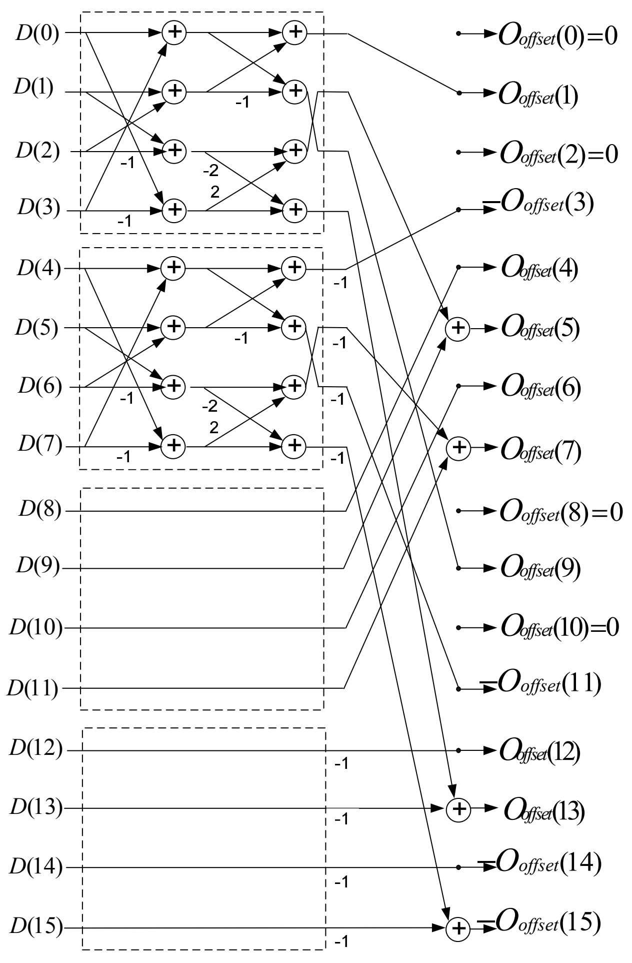

Fig. 6 Data flow for computation of O_{offset}

electronic devices, such as *Digital Still Camera (DSC)*. The proposed solution utilizes a novel 2-D transform to derive the *forward integer transform* coefficients directly from that of *Hadamard transform* with significantly reduced number of operations and, therefore, area and power consumption. The second solution targeted video compression applications processing high-resolution video frames in real-time, such as *Digital Video Camera (DVC), Television Studio Broadcast* and *Surveillance video*. The proposed solution reduced the effective latency penalty of the *forward integer transform* to zero. Moreover, it aggressively re-used the intermediate results from *Hadamard transform* and, therefore, requires significantly reduced number of operations and thus less area for its hardware implementation.

REFERENCES

[1] ITU-T Rec. H.264 and ISO/IEC 14496-10:2005 (E) (MPEG-4 AVC), "Advanced Video Coding for Generic Audiovisual Services", 2005.

[2] ISO/IEC JTC1/SC29/WG11, "Report of the Formal Verification Tests on H.264/AVC," doc. N6231, Waikoloa, USA, 2003.

[3] G. Sullivan, P. Topiwala and A. Luthra, "The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions", SPIE Conf. On Apps. Of digital Image Processing, vol.5558, pp. 454-474, 2004.

[4] J. Ostermann, J. Bormans, P. List, D. Maroe, M. Narroschke, F. Pereira, T. Stockhammer and T. Wedi, "Video Coding with H.264/AVC: Tools, Performance and Complexity", IEEE Circuit and Systems Magazine, vol 4, no. 1, pp. 7-28, 2004.

[5] G.L. Foresti, P. Mahonen and C.S. Regazzoni, "Multimedia Video-Based Surveillance Systems: Requirements, Issues and Solutions", Kluwer Academic Publishers, ISBN 0-7923-7927-6, 2000.

[6] Y. W. Huang, B. Y. Hsieh, T. C. Chen, L. G. Chen, "Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder", IEEE Transaction on Circuit and Systems for Video Technology (CSVT), vol. 15, no. 3, pp. 378-401, 2005.

[7] Reference Software for H.264 codec JM 7.5b: http://iphome.hhi.de/suehring/tml/index.htm

[8] L. Z. Liu, Q. Lin, M. T. Rong, and J. Li,"A 2-D Forward/Inverse Integer Transform Processor for H.264 based on Highly-Parallel Architecture", Proc. 4th IEEE Int. workshop on system-on-Chip for Real-Time Applications , pp. 158-161, 2004

[9] Z. Y. Cheng, C. Chen, B.D. Liu, and J. F. Yang, "High Throughput 2-D Transform Architectures for H.264 Advanced Video Coders", Proc. IEEE Asia-Pacific Conf. Circuits and Systems. Pp 1141-1144, 2004.

[10] C. P. Fan, "Fast 2-Dimensional 4x4 Forward Integer Transform Implementation for H.264/AVC", IEEE Transaction on Circuit and Systems II , vol. 53, no. 3, pp. 174-177, 2006.

[11] T. C. Wang, Y. W. Huang, H. C. Fang, and L. G. Chen. "Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC/H.264.", Proc. IEEE Int. Symp. Circuits and Systems, pp. 800-803, 2003.

[12] K. H. Chen, J. I. Guo, and J. S. Wang, "A High-Performance Direct 2-D Transform IP Design for MPEG-4 AVC/H.264", IEEE Transaction on Circuits and Systems for Video Technology. Vol 16, no. pp. 472-483, 2006.

[13] Roman. C. Kordasiewics, S. Shirani "ASIC and FPGA Implementations of H.264 DCT and Quantization Blocks.", IEEE Int. Conf. on Image Processing (ICIP), vol. 3, pp. 1020-1023, 2005.

[14] C. P. Fan, "Cost-effective Hardware Sharing Architectures of Fast 8x8 and 4x4 Integer Transforms for H.264/AVC", IEEE Asia Pacific Conf. on Circuits and Systems, pp. 776-779, 2006.

[15] W. Hwangbo, and C. M. Kyung, "A Multitransform Architecture for H.264/AVC High-Profile Coders", IEEE Transactions on Multimedia, vol. 12, no. 3, 2010.

[16] C. Y. Huang, L. F. Chen, and Y. K. Lai, "A High-speed 2-D Transform Architecture with Unique Kernel for Multi-standard Video Applications", IEEE Int. Symposium on Circuits and Systems(ISCAS), pp. 21-24, 2008.

[17] Y. Li, Y. He, and S. Mei, "A Highly Parallel Joint VLSI Architecture for Transforms in H.264/AVC", Journal of Signal Processing Systems, vol. 50, no. 1, pp. 19-32, 2008

[18] H. Y. Lin, Y.C. Chao, C. H. Chen, B. D. Liu, and J F. Yang, " Combined 2-D transform and Quantization Architectures for H.264 Video Coders", ISCAS, pp. 1802-1805, 2005

[19] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity Transform and Quantization in H.264/AVC", IEEE Transaction on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 598-603, 2003

[20] D. Marpe, V. George, H. L. cycon, K. U. Barthel, "Performance Evaluation of Motion-JPEG2000 in Comparison with H.264/AVC Operated in Pure Intra Coding Mode", SPEI Conference on Wavelet Applications in Industrial Processing, pp. 129-137, 2003.

[21] G. jin and H. J. Lee, "A Parallel and Pipelined Execution of H.264/AVC Intra Prediction", IEEE International Conference on Computer and Information Technology (CIT), pp. 246-251, 2006.

[22] E. Sahin and I. Hamzaoglu, "An Efficient Hardware Architecture for H.264 Intra Prediction Algorithm", DATE, pp. 44-49, 2007.

[23] M. Shafique, L. Bauer, and J. Henkel, "A Parallel Approach for High Performance Hardware Design of Intra Prediction in H.264/AVC Video Codec", DATE, pp 1434-1439, 2009