Efficient Implementations of Mobile Video Computations on Domain-Specific Reconfigurable Arrays

Sami Khawam[1], Sajid Baloch[2], Arjun Pai[2], Imran Ahmed[2], Nizamettin Aydin[1], Tughrul Arslan[1,2], Fred Westall[3]

[1] *School of Electronics and Engineering. University of Edinburgh, King's Buildings, Mayfield Road, Edinburgh, EH9 3JL, UK*

[2] *Institute for System Level Integration The Alba Centre, Alba Campus Livingston, EH54 7EG, UK*

[3] *EPSON Scotland Design Centre Integration House, Alba Campus Livingston, EH54 7EG, UK*

Abstract

Mobile video processing as defined in standards like MPEG-4 and H.263 contains a number of time-consuming computations that cannot be efficiently executed on current hardware architectures. The authors recently introduced a reconfigurable SoC platform that permits a low-power, high-throughput and flexible implementation of the motion estimation and DCT algorithms. The computations are done using domain-specific reconfigurable arrays that have demonstrated up to 75% reduction in power consumption when compared to generic FPGA architecture, which makes them suitable for portable devices. This paper presents and compares different configurations of the arrays to efficiently implementing DCT and motion estimation algorithms. A number of algorithms are mapped into the various reconfigurable fabrics demonstrating the flexibility of the new reconfigurable SoC architecture and its ability to support a number of implementations having different performance characteristics

1. Introduction

Video and multimedia processing on portable devices, such as mobile phones, requires a lot of computing power to run specific computations such as Motion Estimation (ME) and the Discrete Cosine Transform (DCT). These algorithms can be implemented on Digital Signal Processors (DSPs), however, this leads to a high operating frequency and increased power consumption of the system. The other possibility is to implement these complex algorithms on dedicated hardwired logic, which reduces the power consumption considerably at the expense of reducing flexibility in the hardware. The specifications of such algorithms are permanently changing, hence the need for a flexible architecture that could adapt to such changes. A possible solution is a Field Programmable Gate Array (FPGA) which provides a high-flexibility and low-cost at the expense of increased power consumption.

To overcome this problem, the authors recently introduced in [1] and [2] a novel reconfigurable platform that provides a compromise between performance, speed, power-consumption and flexibility when implementing such complex algorithms. The reconfigurable system uses domain-specific programmable arrays that are optimized for the target computations, such as ME or DCT. Domain-specific arrays have less flexibility than generic FPGAs, however, they are more efficient in implementing the target computations. Furthermore, multimedia computations usually have a number of possible implementations each with different drawbacks in terms of quality, power-consumption and processing time; the reconfigurable arrays provide an efficient solution to map a number of these possible implementations and switch between them dynamically.

Both a software flow and interconnects types have been proposed to be able to create reconfigurable arrays specific to any application or computation. Initially this has been used to create arrays targeting ME and DCT. The array presented in [1] targeting ME calculations showed a reduction of around 75% in power consumption when compared to generic FPGAs, while the area is reduced by 45% and timing improved by 23%. In [2] the same structure is used to design an array for Distributed Arithmetic (DA) calculations: the array provides a 38% reduction in power consumption, 14% in area and 54% decrease in the maximum operating frequency.

In this paper we present a number of different ME and DCT implementations targeting the previously described arrays. These implementations prove the flexibility and reconfigurability of the arrays. The implementations have different characteristics in terms of area usage on the array and power consumption which are explored in this paper. This paper shows that the ability of the new reconfigurable

System-on-Chip to support these implementations proves that this domain-specific reconfigurable architecture is well suited to flexibility-demanding applications like MPEG-4.

This paper is organized as follows: In section 2, the reconfigurable system-on-chip architecture is overviewed. Sections 3 describes the DCT implementations while section 4 contains the ME implementations.

2. Reconfigurable Platform and Arrays

The reconfigurable system is composed of DSPs, processors and the domain-specific reconfigurable arrays as shown in Fig. 1 and described in [1] [2]. The communication between these arrays and the processor takes place through a system-on-chip. A controller in the processor is used to integrate and generate the addresses for these array structures.

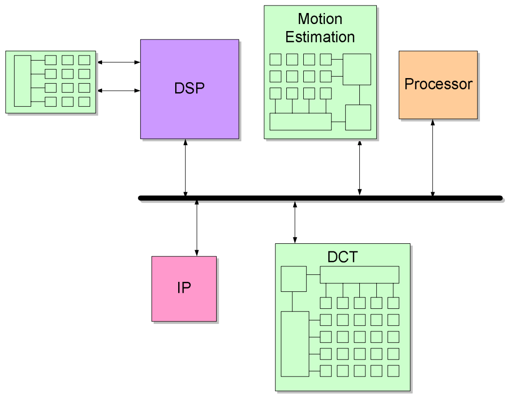

Figure 1: Reconfigurable System-on-Chip

The reconfigurable arrays used can be treated as any soft-core as they are generated as synthesizable netlists which are easily integrated in the software flow of the ASIC. The arrays are heterogeneous and contains clusters specific to one operation. The combination of such clusters makes the array domain specific. The clusters used for DCT and ME are described below.

In the arrays different levels of reconfigurable interconnects are provided to allow flexible mapping of diverse ME and DCT algorithms. Each cluster is composed of a number of elements. High-speed and short interconnects are provided inside the clusters in order to connect the elements together; this allows cascading the elements to permit computations wider than the 4-bits provided by one element. Furthermore, an interconnects mesh similar to the one used in generic FPGA architectures is provided to connect the clusters together. The mesh is composed of a combination of 8-bit and 1-bit tracks, which allows having a reduced number of switches and configuration bits when compared generic fine-grain 1-bit FPGAs.

2.1 Array for Motion Estimation

The computational clusters in the array targeting ME calculations where designed for different ME architectures with varying performance, speed and area requirements. The following clusters were identified as common reusable blocks, and arranged as shown in Fig. 2 :

- Register-Multiplexer (MUX): A 2-to-1 multiplexer with optional register at the output.
- Absolute Difference Calculator (AD): Two inputs addition and subtractions and an optional absolute difference calculator are supported.
- Adder/sub-tractor and accumulator (ADD/ACC): Allows sequential accumulation and simple combinatorial addition and subtraction.
- Min/Max Comparator (COMP): Enables the comparison of two values and a ability to detect the maximum or minimum value of a vector.

2.2 Array for DCT

The array for DCT targets Distributed Arithmetic calculations, which includes computations like filtering, DCT and DWT. Distributed Arithmetic is supported using the following two clusters which are arranged as in Fig. 3:

- Add-Shift clusters that allows addition and subtraction operations as well as shifting. Shift-accumulation is also supported.
- Memory elements can be used to implement Look-Up-Tables and ROMs with configurable geometries.

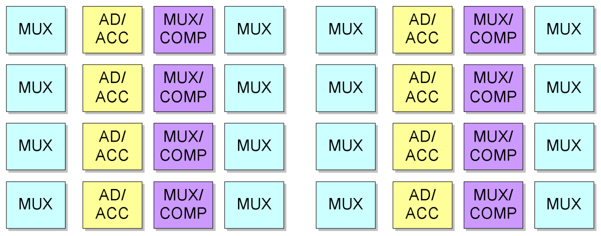

Figure 2: Array with clusters for Motion Estimation [1].

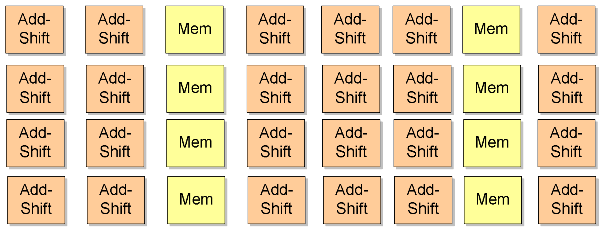

Figure 3: Array with clusters for Distributed Arithmetic and DCT [2].

3. DCT Implementations

3.1 Distributed Arithmetic

Distributed Arithmetic (DA) implementations [4] target sum-of-product calculations with multiplications by fixed-coefficients. The multiplication is replaced by a Look-Up-Table (LUT) and a shift-accumulator. DA provides an efficient implementation for multiple sum-of-product calculations having the same input, which is the case for the DCT [3]. The equation for a 1-D N-point DCT is:

$$X_u = c(u) \cdot \sum_{i=0}^{N-1} x(i) \cdot \cos\left(\frac{(2i+1)\cdot u\pi}{2N}\right)$$

The N-points DCT can be considered as N parallel filters. The basic bit-serial DA implementation of the DCT is shown in Fig. 4. The DCT on the array requires N shift-registers for parallel-to-serial conversion, N LUT memories and N shift-accumulators. All the N memories receive the same address.

One shift-register and a shift-accumulator are each mapped to a *add-shift* cluster, while the LUT is mapped to a part of a *memory* cluster.

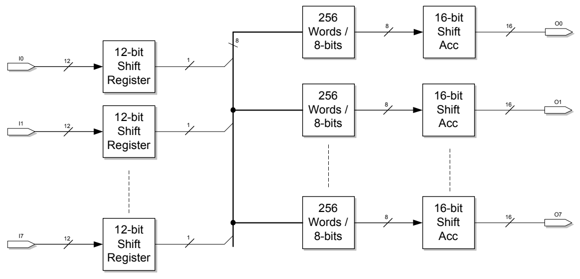

Figure 4: Simple DCT implementation using Distributed Arithmetic

3.2 Mixed ROM

The 8-point 1D-DCT can be expressed as the product of an 8x8 matrix by an eight element column vector [4]. However, through algebraic manipulations [6] [7] this matrix can reduced to 4x4 matrix. Hence, the number of words per ROM is reduced to only 16 which is 16 times less than the previous implementation [5] but some overhead has been incurred in the form of adders to calculate the address of the ROMs. The mapping of the algorithm onto the proposed reconfigurable fabric is illustrated in Fig. 5.

3.3 CORDIC based # 1

The DCT computation using COordinate Rotation DIgital Computer (CORDIC) is used which is described in [3]. Since the memory is an integral part of the DA and ROM size increases exponentially with respect to vector size N. Many techniques have been developed for reducing the size of ROM. The CORDIC algorithm reformulates the 1-D DCT so that the ROM size is reduced to a fix size of 4 words, independent of the bandwidth of the input data.

The DA based CORDIC is calculated through the following expressions [7]:

$$x' = \sum_{ii=0}^{N-1} \left(x_j \cos\phi - y_j \sin\phi\right) 2^{-j} - \left(x_0 \cos\phi - y_0 \sin\phi\right)$$

$$y' = \sum_{ii=0}^{N-1} \left(x_j \cos\phi + y_j \sin\phi\right) 2^{-j} - \left(y_0 \cos\phi + x_0 \sin\phi\right)$$

The DA functionality is implemented by converting parallel data to serial through shift registers and using this data to formulate the address of the memories. This CORDIC based implementation requires 6-CORDIC and 16 butterfly adders for an 8 point 1D DCT. The CORDIC rotators are implemented through ROM and Shift Accumulators as shown in the Figure 6: while butterfly adders are implemented through add-shift clusters.

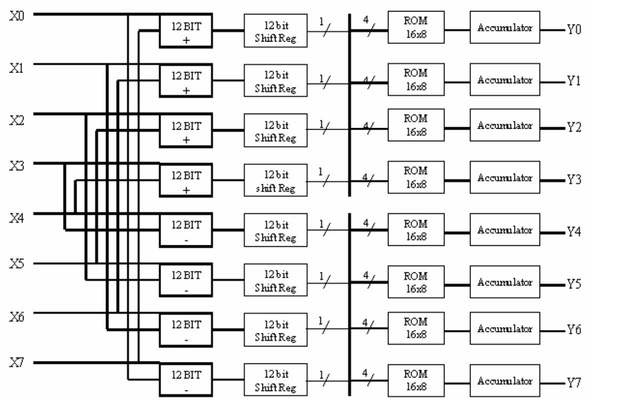

Figure 5: DCT Implementation using 4x4 matrix for memory reduction

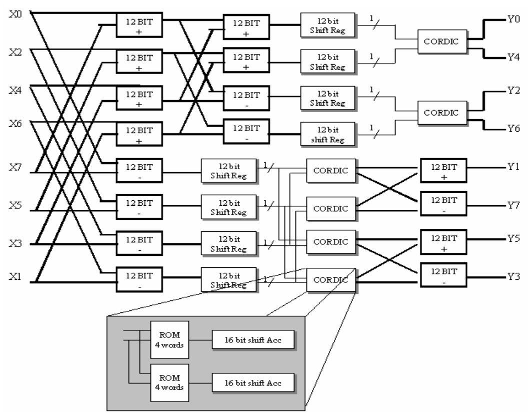

Figure 6: CORDIC Rotator Based 8-Point DCT Implementation

3.4 CORDIC based # 2

Since CORDIC algorithm results in a very regular structure suitable for VLSI implementation, there has been great interest in developing the CORDIC based DCT architecture. The implementation shown in Figure 7 is a scaled DCT architecture based upon the CORDIC algorithm, which is described in [8]. This implementation is different to first CORDIC based implementation in the following respects:

- Uses 20 butterfly adders instead of 16
- Uses 3 CORDIC rotators instead of 6

The constant scale factor is not considered in this implementation as that can be combined with the quantization constants without requiring any extra hardware.

The CORDIC rotators are based upon the same elements which are discussed in previous CORDIC based implementation.

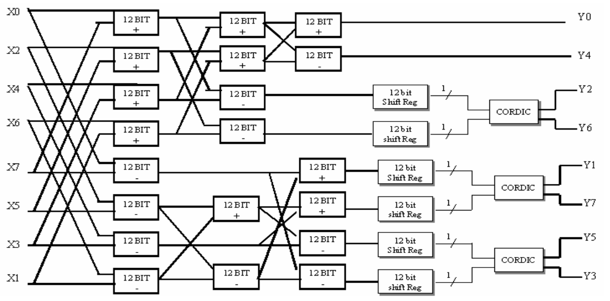

Figure 7: A Scaled DCT Architecture Based Upon CORDIC Algorithm

3.5 Skew circular convolution based

The algorithm is proposed by Li [10] which is well suited for DA based VLSI implementations. This technique starts with re-ordering the input sequences. Then Skew circular convolutions are performed on the reordered inputs, which gives odd-indexed transformed sequence. The transformed sequences are re-ordered for the proper output sequences. The Mapping of Li's Algorithm onto re-configurable array for odd-indexed DCT is shown in figure 8.

The odd-indexed DCT components are calculated through following equation under skew circular convolution [10]:

$$X'_j = \sum_{i=0}^{N/2-1} [X'_i - X'_{N/2+i}] \cos\left(\left(\frac{2\Pi}{4N}\right) 3^{i+j}\right)$$

In the same way, even indexed DCT components are calculated using:

$$X'_j = \sum_{i=0}^{N/2-1} [X'_i + X'_{N-1-n}] \cos\left(\frac{\Pi(2n+1)j}{2\frac{N}{2}}\right)$$

These equations can be presented in matrix form which is explained in [11] and these can be implemented in VLSI. Only a 16 words ROM is required as DCT components are separated into odd and even. The different elements which are required for Skew Circular Convolution are illustrated in Figure 8.

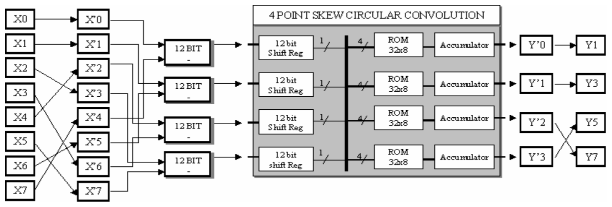

Figure 8: Implementation of Li's Algorithms to Compute Odd-indexed DCT coefficients.

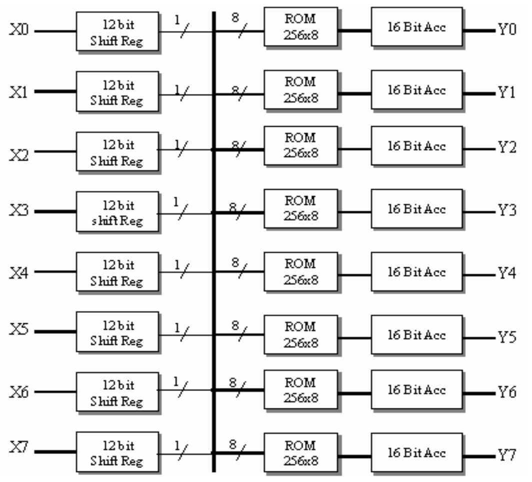

Figure 9: Implementation of Li's Algorithms with larger LUTs but not adders and subtracters at the input.

DCT can be calculated through skew circular convolution without splitting into odd and even indexed components and reordering stages at input and output, through following set of equations [10].

$$X'_j = \sum_{i=0}^{N-1} X'_i . \cos\left(\frac{2\Pi . 3^{i+j}}{4N}\right).$$

The above equation can be expressed as:

$$\begin{bmatrix} Y'_0 \\ Y'_1 \\ Y'_2 \\ Y'_3 \\ Y'_4 \\ Y'_5 \\ Y'_6 \\ Y'_7 \end{bmatrix} = \begin{bmatrix} C_0 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 \\ C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_0 \\ C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_0 & C_1 \\ C_3 & C_4 & C_5 & C_6 & C_7 & C_0 & C_1 & C_2 \\ C_4 & C_5 & C_6 & C_7 & C_0 & C_1 & C_2 & C_3 \\ C_5 & C_6 & C_7 & C_0 & C_1 & C_2 & C_3 & C_4 \\ C_6 & C_7 & C_0 & C_1 & C_2 & C_3 & C_4 & C_5 \\ C_7 & C_0 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \end{bmatrix} x \begin{bmatrix} X'_0 \\ X'_1 \\ X'_2 \\ X'_3 \\ X'_4 \\ X'_5 \\ X'_6 \\ X'_7 \end{bmatrix}$$

Where $C_i = \cos\left(\left(2\Pi / 4N\right)3^i\right)$

The implementation requires 256 words ROM which is 16 times more than the previous implementation but does not require adder/subtracters. DA based implementation of the above algorithm is shown in Figure 9.

3.6 Comparison of implementations

The implementations have been compared in terms of area consumed on the array and the result is shown in Table 1. Since all the clusters have a similar area on the chip, the total number of clusters used defines the total area usage. At these initial stages no power estimation was performed; the implementations can have different power consumption due to the different area usage and different signal activities in the design. Hence, some implementations can be more suited to low-power applications.

	MIX ROM	COR-DIC 1	COR-DIC 2	SCC EVEN/ ODD	SCC
Add-Shift Clusters					
a) Adders	4	08	10	04	0
b) Subtracters	4	08	10	04	0
c) Shift Reg	8	08	06	08	8
d) Acc	8	12	06	08	8
Total	24	36	32	24	16
Mem- Cluster	08	12	06	08	08
Total clusters	32	48	38	32	24

Table 1: Area usage of the DCT implementations

4. Motion Estimation Implementations

Motion estimation is based largely on a search scheme, which tries to find the best matching position of a 16x16 macro-block of the current frame with all the candidate blocks within a predetermined or adaptive search range in the previous frame. The matching position relative to the original position is described by a displacement vector called the motion vector. The motion vector gives the displacement of the macro-block with the minimum distortion from the reference macro-block. The matching criterion usually used is the Sum of Absolute Differences (SAD) and this is the one supported by the array:

$$\text{SADN (dx,dy)} = \sum_{m=x}^{N}\sum_{n=y}^{N}\left|I_K(m,n)-I_{K-1}(m+dx,n+dy)\right|$$

Where $I_K(m,n)$ and $I_{K-1}(m+dx\ ,\ n+dy)$ are luminance values of the macro block and N is the size of the block (could be 8, 16 or 32)

Full search algorithms provide optimal solutions with low control overhead and are based on matching criterion. The FSBMA are computationally complex and are accompanied by significant power consumptions. The 1-D array architectures proposed among which are [12]-[14] require high operating frequencies in order to fulfill the data-flow requirements of these demanding complex algorithms for ME. Several ME algorithms where reviewed with an intension to create a feasible reconfigurable cluster array. Various low-power 2D systolic array implementations for block matching algorithms have been proposed in [18] and [19].

A systolic array was mapped on the reconfigurable array. The basic structure of the PE is depicted in the Fig. 10. The PE architecture shows the use of the clusters from the array.

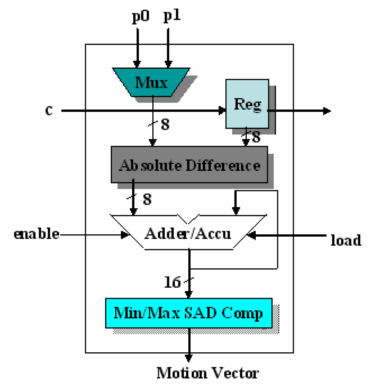

Figure 10: PE Module for the Motion Estimation architecture

The PE array is composed of a 4 x 16 PEs (64 PE 2-D array) as shown in Fig. 11. The search area pixels are broadcasted through all the 16 PE array (PE module 0) and the current pixel data is propagated through using a shift register. The PE array is segregated into 4 modules with each row having a 16 PE single array. Each PE module is responsible for calculating the SAD for a particular candidate block during block matching. The first round of SAD calculations would take 16 clock cycles. The previous data is fed to the PEs through a reconfigurable Register-Multiplexer module which helps in reducing the memory bandwidth. The current pixel data value is shifted through a register array. Each PE cell structures helps in computation of the absolute difference between the previous and current block luminance values of these pixels.

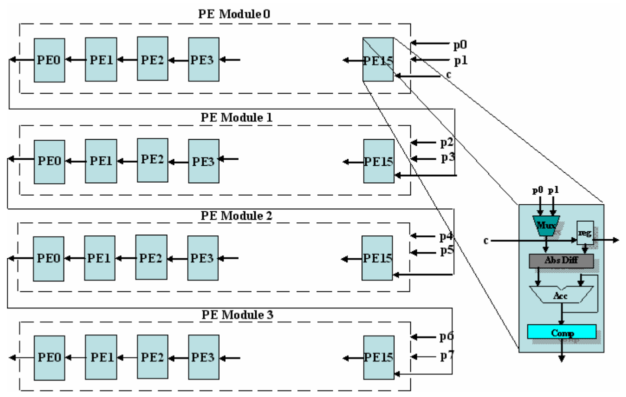

Figure 11: Low-Power 2-D systolic array architecture

5. Conclusion

This paper has described a number of DCT and Motion Estimation implementations that have been successfully mapped to domain-specific reconfigurable arrays designed for these computations. The implementations have different advantages in terms of area usage on the array and time needed to finish the computation as well as the quality and precision of the output result. This demonstrates the flexibility provided by the arrays in allowing the mapping of a range of implementations of the same calculation.

Hence, the flexible arrays provide a better alternative to hardwired ASIC solutions for complex algorithms that are in constant update such as MPEG-4. Furthermore, the arrays have the ability to be dynamically reconfigured to support different implementations of the same algorithms for different run-time constraints, such as low-battery conditions and noisy channels in mobile devices.

6. REFERENCES

[1] Khawam S., Arslan T., Westall F., "Embedded reconfigurable array targeting motion estimation applications", Proceedings of the 2003 IEEE International Symposium on Circuits and Systems, May 2003, Vol. 2, pp. 760-763

[2] Khawam S., Arslan T., F. Westall F., "Domain-specific reconfigurable array for Distributed Arithmetic", Proceedings of the 13th International Conference on Field Programmable Logic and Applications, FPL 2003, Sept 2003, pp. 1139-1144

[3] Ahmed N., Natrajan T., Rao K.R., "Discrete Cosine Transform" IEEE Tans. On Computers, Vol. C-23, No. 1, p90-93, Dec. 1984

[4] White, S.A, "Applications of distributed arithmetic to digital signal processing: a tutorial review", ASSP Magazine, IEEE, Volume: 6 Issue: 3, Jul 1989, Page(s): 4-19

[5] Sungwook Yu, Swartzlander, E. E., Jr., "DCT implementation with distributed arithmetic", IEEE transactions on Computers, Vol. 50 Is. 9, Sept. 2001.

[6] B. G. Lee. "A new algorithm to compute the discrete cosine transform" IEEE Transactions on Accoustics, Speech and Signal Processing, ASSP-32:1243-1245, Dec. 1984.

[7] P. Pirsch, N. Demassieux and W. Gehrke, "VLSI architectures for video compression-a survey", Proceedings of the IEEE, 83:220-246, Feb. 1995.

[8] Yi Yang, Chunyan Wang, Omair Ahmed, M., Swamy M. N. S., "An online CORDIC based 2-D IDCT implementation using distributed arithmetic", 6th Inter. Symp. on Signal Processing and its applications,. 2001, Vol. 1

[9] Sungwook Yu, Swartzlander, E. E., Jr., "A scaled DCT architecture with the CORDIC algorithm" IEEE Transactions on Signal Processing, Vol. 50, Jan. 2002

[10] W. S. Wong, A. Berno, Hussein M. A., "A fast VLSI chip for computing the two-dimensional discrete cosine transform", IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, 1993.,Vol.: 2 , pp : 662 -665

[11] W. Li, "A new algorithm to compute the DCT and its inverse", IEEE Transactions on Signal Processing, Vol. 39 no. 6, pp. 1305-1313, Jun. 1991.

[12] Komarek, T.; Pirsch, P., Array architectures for block matching algorithms, IEEE Transactions on Circuits and Systems, Vol. 36 Issue: 10 , Oct. 1989

[13] Yang, K.-M.; Sun, M.-T.; Wu, L. , A family of VLSI designs for the motion compensation block-matching algorithm, IEEE Transactions on Circuits and Systems, Vol. 36 Issue: 10 , Oct. 1989

[14] De Vos, L.; Stegherr, M.; Noll, T.G., VLSI architectures for the full-search blockmatching algorithm, International Conference on Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989

[15] Stechele, W., Algorithmic complexity, motion estimation and a VLSI architecture for MPEG-4 core profile video codecs, International Symposium on VLSI Technology, Systems, and Applications, 2001

[16] Xiao-Dong Zhang; Chi-Ying Tsui; An efficient and reconfigurable VLSI architecture for different block matching motion estimation algorithms Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE.

[17] Kin-Hung Lam; Chi-Ying Tsui; Low power 2-D array VLSI architecture for block matching motion estimation using computation suspension, Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on , 11-13 Oct. 2000

[18] Elgamel, M.A.; Shams, A.M.; Xi Xueling; Bayoumi, M.A.; Enhanced low power motion estimation VLSI architectures for video compression, Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE Inte. Symp. on , Vol.: 4

[19] Elgamel, M.A.; Shams, A.M.; Bayoumi, M.A.; A comparative analysis for low power motion estimation VLSI architectures ,Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on , 11-13 Oct. 2000

[20] Sousa, L.; Roma, N.; Low-power array architectures for motion estimation, Multimedia Signal Processing, 1999 IEEE 3rd Workshop on , 13-15 Sept. 1999