Static and Dynamic Reconfigurable Designs for a 2D Shape-Adaptive DCT

Jörn Gause[1], Peter Y. K. Cheung[1], and Wayne Luk[2]

[1] Department of Electrical and Electronic Engineering,
Imperial College of Science, Technology and Medicine,
London SW7 2BT, United Kingdom
{j.gause@ic.ac.uk, p.cheung@ic.ac.uk}
[2] Department of Computing,
Imperial College of Science, Technology and Medicine,
London SW7 2AZ, United Kingdom
wl@doc.ic.ac.uk

Abstract. This paper presents two reconfigurable design approaches for a two dimensional Shape-Adaptive Discrete Cosine Transform (2D SA-DCT). The SA-DCT is an example of a new type of multimedia video processing algorithm where the computations performed are data dependent. A static design, where the configuration does not change during execution of the task, is presented. The use of a data dependence graph (DDG) is proposed which represents the computations and input signals required to calculate a particular output signal depending on a variable input parameter. By re-structuring the DDG and exploiting possible sharing of FPGA resources for different entities within the SA-DCT, it is demonstrated that the area required for an implementation can be significantly reduced. An alternative dynamic approach is also introduced where the FPGA's configuration may change over time. This is well suited to using dynamically reconfigurable logic but suffers from long reconfiguration time if current FPGAs are used.

1 Introduction

Multimedia processing is characterised by very high processing demands. Typical multimedia applications entail combined processing of various data types including video, audio, speech, images, 2D/3D graphics, and text. The video processing tasks are clearly the most computationally intensive. In addition, many novel multimedia processing algorithms involve growing diversity and decreasing predictability in the computation flow. This calls for hardware architectures with increased flexibility at run-time [1].

MPEG-4 has been developed as the new standard for audio-visual coding in multimedia applications [2]. An example of a novel MPEG-4 video processing tool is the Shape-Adaptive Discrete Cosine Transform (SA-DCT) which was introduced by Sikora and Makai in [3]. The algorithm has been included in the MPEG-4 Video Verification Model [4], as an alternative to the standard block-based DCT which is widely used in the MPEG-1, MPEG-2, H.261, and H.263 standards. The SA-DCT is applied for cod-

R.W. Hartenstein and H. Gruenbacher (Eds.): FPL 2000, LNCS 1896, pp. 96-105, 2000.

ing arbitrarily shaped object segments contained within an 8×8 image block, specifically in blocks with at least one transparent pixel.

Due to the arbitrary shape of the object within an 8×8 image block, a hardware implementation of a two dimensional SA-DCT (2D SA-DCT) is not as straightforward as the implementation of the standard 8×8 DCT where the transform is always carried out on eight pixels per row and eight pixels per column. In contrast, for the SA-DCT, the calculations performed, and hence the hardware required to accomplish these calculations, depend on the number of pixels occupied by the object within the 8×8 block. Hence, flexible and efficient architectures and implementations are required to adapt to these constraints.

Reconfigurable logic devices, notably (SRAM based) Field Programmable Gate Arrays (FPGAs), are suitable for dealing with these adaptability requirements as a tradeoff between the speed of ASICs and the flexibility of software [5]. They can be configured for a variety of applications with high processing demands and reconfigured for other applications if necessary. This makes FPGAs very appropriate for the implementation of many MPEG-4 modules in general, and for the SA-DCT in particular.

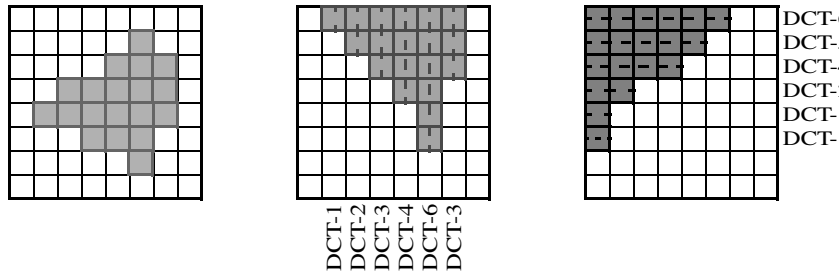
An architecture of a 2D SA-DCT based on time-recursion has been presented in [6]. However, this architecture does not present the best solution in terms of computational requirement and has the disadvantage of numerical inaccuracy due to its second-order recursive structure. An architecture which can perform a DCT-N for variable length N, $2 \le N \le 8$, has recently been proposed in [7]. This design allows efficient sharing of hardware resources for different N but increases the hardware cost compared to an implementation of a single DCT-N for only one particular N.

The purpose of this paper is to investigate suitable designs for an SA-DCT implementation using reconfigurable logic. A generic one dimensional SA-DCT architecture consisting of a static module with a time-constant structure and a dynamic module which can change its structure at run-time is proposed. Employing the proposed 1D SA-DCT architecture, a 2D SA-DCT can be implemented on FPGAs using two different approaches. In this paper, a static design is presented, where the configuration data for all possible computations is loaded once, after which it does not change during execution of the task. The use of a data dependence graph (DDG) is proposed which represents the computations and input signals required to calculate a particular output signal depending on a variable input parameter. By re-structuring the DDG and exploiting possible sharing of FPGA resources for different entities within the SA-DCT, it is demonstrated that the area required for an implementation can be reduced considerably. The results of an implementation based on Distributed Arithmetic on an Altera FLEX10KE FPGA will be presented. A dynamic approach is also introduced where the FPGA's configuration may change over time. It will be shown that this is well suited to using dynamically reconfigurable logic but suffers from long reconfiguration overhead if currently available FPGAs are used.

Section 2 describes the algorithm of the 2D Shape-Adaptive DCT. A generic architecture of the 1D SA-DCT is proposed in Sect. 3 which is used for both, a static and dynamic realisation approach of the 2D SA-DCT. These designs are presented in Sect. 4. In Sect. 5, the implementations of both static and dynamic approach are discussed. Finally, a conclusion follows in Sect. 6.

2 2D Shape-Adaptive DCT

The Shape-Adaptive DCT algorithm is based on predefined orthogonal sets of DCT basis functions. The basic concept of the method for coding an arbitrarily shaped image foreground segment contained within an 8×8 reference block is outlined in Fig. 1 [4]. The required two dimensional SA-DCT is usually separated into two one dimensional SA-DCTs, a vertical SA-DCT followed by a horizontal SA-DCT.



(a) image block with original segment

(b) shifting of opaque pixels and vertical SA-DCT used

(c) shifting of opaque pixels and horizontal SA-DCT used

Fig. 1. 2D SA-DCT method

An example of an image block segmented into foreground (shaded) and background (light) region is shown in Fig. 1(a). To perform the vertical SA-DCT, the number of foreground (or opaque) pixels of each of the eight columns of the image block is calculated, and the columns are shifted and aligned to the top of the 8×8 block (Fig. 1(b)). Depending on the vector size N (number of opaque pixels) for each particular column of the segment, a DCT of size N (DCT-N) is performed on the column data vector $\underline{x} = [x_0\ x_1\ ...\ x_{N-1}]^T$ which results in N vertical DCT-coefficients $\underline{c} = [c_0\ c_1\ ...\ c_{N-1}]^T$ according to [4]:

$$\underline{c} = \sqrt{\frac{2}{N}} \cdot \underline{DCT-N} \cdot \underline{x} \ . \tag{1}$$

The DCT transform matrix $\underline{DCT\text{-}N}$ is defined as:

$$\underline{DCT-N}(p,k) = \alpha(p) \cdot \cos\left[p\left(k+\frac{1}{2}\right)\cdot\frac{\pi}{N}\right] \ , \tag{2}$$

for k, p = 0, 1, ..., N-1, and $\alpha(p) = \begin{cases} \sqrt{\frac{1}{2}} & p = 0 \\ 1 & p \neq 0 \end{cases}$.

Hence, a particular element c_p of $\underline{c}$ can be calculated as a sum of N products using

$$c_p = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} \underline{DCT-N}(p,k) \cdot x_k \quad . \tag{3}$$

For example, the right-most column of the object in Fig. 1 is transformed using a DCT-3. To execute the horizontal SA-DCT, the length of each row of the intermediate block (after vertical SA-DCT) is calculated and the rows are shifted to the left border of the 8×8 block as shown in Fig. 1(c). A horizontal DCT adapted to the size of each row is then performed using (1) and (2).

3 Proposed Architecture for 1D SA-DCT

We propose a generic one dimensional (1D) SA-DCT consisting of two main parts, as shown in Fig. 2. Firstly, the opaque pixels in each column or row have to be shifted to the top or left, respectively, and the number of opaque pixels N per column or row has to be counted (module *Shift & Count*). In the second part, module *DCT-N*, a multiplication of an $N \times N$ constant coefficient matrix with an input vector comprising the values of the N shifted pixels, is performed according to (1) and (2). Whereas module *Shift & Count* is *static*, that is exactly the same module is used for all input signals of the SA-DCT, the module *DCT-N* is *dynamic* since its structure can change at run time depending on N.

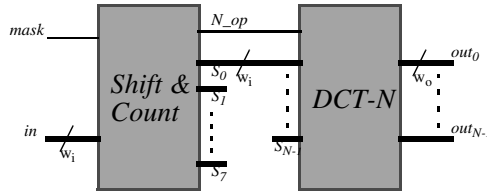


Fig. 2. Proposed generic 1D SA-DCT architecture

The inputs of module *Shift & Count* are the value of one pixel and a mask bit which is 1 if the pixel is opaque and 0 if the pixel is transparent. One pixel value and its respective mask bit are shifted in at a time. The mask bit *mask* is used to count the number of opaque pixels N within each column or row and to shift their values to the first N output registers. Outputs of the module are a) *N_op,* which represents the number of opaque pixels N per column or row, respectively, and b) the eight pixel values of one column or row arranged so that outputs S_0 to S_{N-1} carry the opaque pixel values.

Module *DCT-N* has as inputs *N_op* and the first N outputs of module *Shift & Count*, $S_0 \ldots S_{N-1}$, which are the values of the opaque pixels of the processed column or row (see Fig. 2). Within module *DCT-N* a constant matrix - vector multiplication of size N according to equation (1) is performed. *N_op* can be interpreted as a control signal

which selects the actual DCT-N computation accomplished as sketched in Fig. 3. For $N\_op = 0$, no computations are necessary. Outputs are the N DCT coefficients of the opaque pixels.

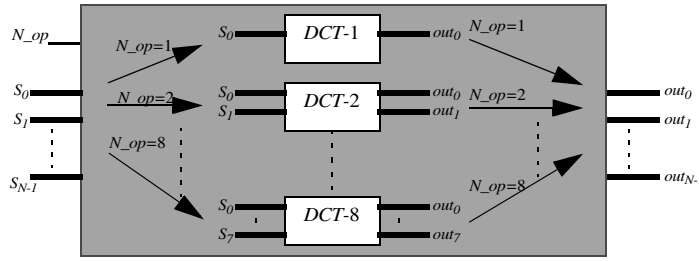


Fig. 3. Generic structure of module *DCT-N*

4 Reconfigurable Design for 2D SA-DCT

The generic architecture for the 1D SA-DCT suggested in Sect. 3 has been used for a reconfigurable design of the 2D SA-DCT. Two general approaches are presented, a *static* design where the configuration is loaded once, after which it does not change during execution of the entire task, and a *dynamic* design where the FPGA's configuration may change over time.

4.1 Static Design

A static implementation of the 2D SA-DCT must be able to calculate the right result for every possible shape of the object within the 8×8 image block. The configuration data of the FPGA must therefore contain all possible DCT-N computations for $1 \le N \le 8$. In a straightforward implementation the circuit can perform all eight DCT-N calculations in parallel and select the outputs depending on N. The main disadvantage of this approach is the large amount of hardware necessary to implement all eight DCT-Ns, even though only one DCT-N is required at a time. For an efficient FPGA implementation it is therefore necessary to share hardware resources throughout different DCT-N entities as much as possible. Hence, the relationship between the DCT-size N and the structure of the data flow of the *DCT-N* module has to be investigated to find a representation which allows a more area efficient implementation.

We propose the use of a *data dependence graph* (DDG) which represents the computations and input signals required to calculate a particular output signal depending on a variable input parameter. A DDG consists of coloured nodes and directed edges. The nodes represent independent computations or tasks. If two nodes have the same colour, their respective calculations result in the same output signal. Only one of the computations marked by the same colour can be performed at a time, determined by the variable input signal. The edges of the graph directed towards the nodes show which input sig-

nals are required for the particular operation represented by the node whereas the edges directed away from the nodes point at the output of this operation. The edges are labelled by a value of the variable input signal. For a particular value of the variable parameter, the computation flow uses only the edges and respective nodes labelled by this value. Nodes of the DDG can be grouped into blocks. A block can be thought of as a hardware entity. By re-structuring the DDG and re-grouping nodes, hardware resources can be shared more efficiently.

Figure 4 shows the DDG of a part of the *DCT-N* module (for $N = 1$, 2, and 3), before and after re-structuring and re-grouping. The graph shows which computations and which input signals have to be used to calculate a particular output signal depending on N. The signals s_k ($k = 0 \ldots 7$) stand for the inputs, here the values of the opaque pixels, and the signals c_p ($p = 0 \ldots 7$) represent the outputs. A node of the DDG denoted DCT-N(p) symbolises the computations performed, in this case the multiplication of the pth row vector of matrix $\underline{DCT\text{-}N}$ with input vector $\underline{s} = [s_0\ s_1 \ldots s_{N-1}]^T$, resulting in the pth output c_p of DCT-N according to equation (3). Different dashed line patterns are used for the edges to distinguish between different N. Every computation is to be performed only for one specific N. For instance, DCT-2(1) calculates output c_1 of DCT-2 (that is N=2) using inputs s_0 and s_1. Whilst c_1 does not exist for N=1 (DCT-1 has only one output c_0), it can also be output of DCT-3 (for N=3) where s_0, s_1, and s_2 are used as input signals.

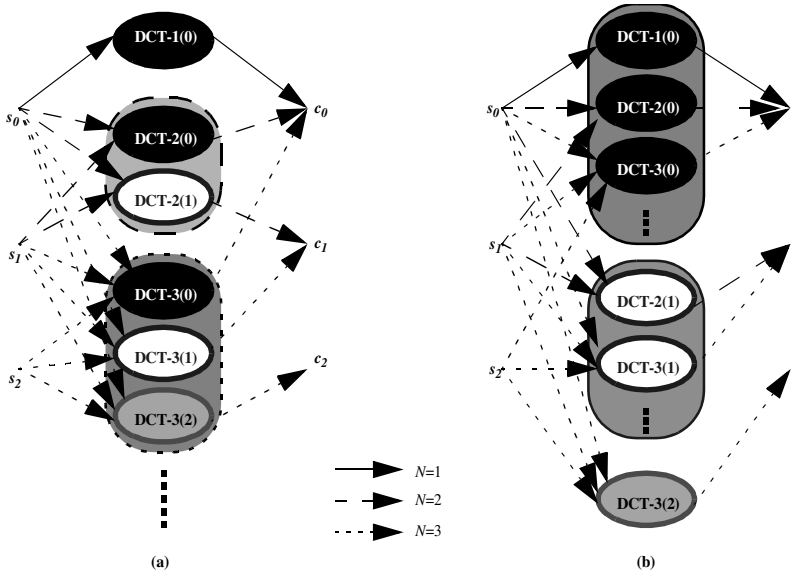


Fig. 4. Restructuring of data dependence graph

In Fig. 4(a) the DDG is arranged in a way so that nodes (operations) marked with the same N are grouped together into blocks. In this case, the computations of every block are necessary only for one particular value of N. For an SA-DCT hardware im-

plementation in this manner only the results of one of eight blocks are used at a time while the others are not required, even though all eight blocks exist and produce outputs. Within one block all computations are required at the same time and each computation produces a different output value, therefore sharing of hardware resources is difficult. In fact, for a 1D SA-DCT design in this manner, all possible 36 multiply and accumulate (MAC) calculations according to (3) have to be implemented (one for DCT-1, two for DCT-2, and so on) separately.

An alternative way of grouping nodes into blocks is shown in Fig. 4(b). Here, all nodes labelled with the same p, that is computations which produce the same output, are placed into the same group. Hence, every output signal is produced only by one particular block, no output selection is necessary. The signal N is used to select the right computation required to calculate a particular output, in contrast to a design according to Fig. 4(a) where N is used to select the right outputs amongst the computation results of all blocks. Each block contains at most one computation for which the result is required at a time. This allows intensive hardware resource sharing within one block while using blocks in parallel. For a 1D SA-DCT implementation in this manner, only eight MAC units, which is the minimum needed to implement a DCT-8 with eight outputs, and a decoder which selects the right constants of the *DCT-N* matrix, depending on N, need to be implemented. Hence, the number of MAC modules, and therefore the hardware cost, can be reduced significantly to approximately 22%.

4.2 Dynamic Design

In a dynamic implementation of the 2D SA-DCT the configuration of the FPGA depends on the input data, that is on the shape of the object to be transformed. While reading in the pixel values of the first column, the values of the opaque pixels are shifted to the top and N is counted using module *Shift & Count* as described in Sect. 3. Depending on N, the part of the FPGA which is used to perform the calculations of module *DCT-N* is reconfigured with the configuration data for the particular DCT-N computations required ($1 \le N \le 8$). Hence, DCT-N is performed using the N opaque pixel values as inputs, and the output values are stored. This process of reading in the data, shifting and counting N, reconfiguring the device for the relevant DCT-N and performing the DCT-N calculation is repeated for all columns and rows, until an entire 2D SA-DCT has been accomplished.

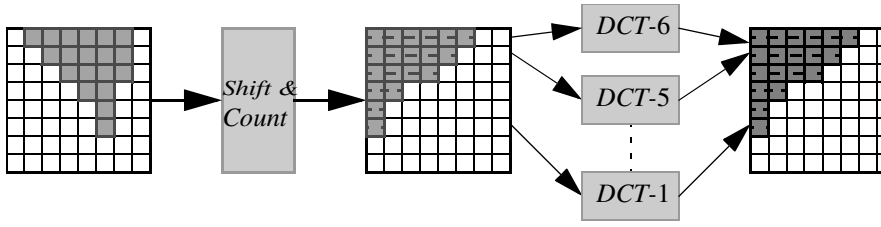


Fig. 5. Example of dynamic 1D (horizontal) SA-DCT

This approach is well suited to using dynamically reconfigurable logic within a custom computing machine (CCM). An example of a dynamic implementation of a horizontal SA-DCT is illustrated in Fig. 5. The top row of the object has six opaque pixels, that is N=6 for this row. Hence, module *DCT-N* is instantiated as *DCT*-6, that is the FPGA is reconfigured to perform a DCT-6. After sending back the computation results, a DCT-5 for the second row of the object has to be performed. Therefore, the FPGA has to be reconfigured with the configuration data for a DCT-5. This is repeated for all rows of the object. Sixteen reconfigurations are necessary to perform a complete 2D SA-DCT in this manner.

5 FPGA Implementation and Results

We first analyse how to efficiently implement a general constant matrix - vector multiplication as required for each DCT-N calculation and then how to incorporate those computations into an SA-DCT architecture. A very efficient method of computing this multiply and accumulate (MAC) operation, especially for FPGAs, is to use Distributed Arithmetic [8], where the MAC calculations are performed in a bit-serial manner. The technique is based on storing pre-calculated scalar products for all possible bit patterns of the input signals in a ROM. By exploiting symmetries within the *DCT-N* matrices the number of ROM address bits can be reduced to $\lceil N/2 \rceil$ and the number of required ROM words can be decreased to $2^{\lceil N/2 \rceil}$ [9]. Distributed Arithmetic can make extensive use of look-up tables (LUTs) and/or embedded ROMs which are part of modern SRAM based FPGAs, such as Altera FLEX 10K [11] or Xilinx Virtex [12], and hence make them ideal for this type of computations. No multipliers are necessary. This is especially important for FPGAs since they are very efficient for shift and add operations but are generally inefficient for implementing parallel multipliers [10].

For a static SA-DCT implementation using Distributed Arithmetic, effective hardware resource sharing according to Fig. 4(b) can be achieved by using the same ROM for more than one DCT-N, with N forming part of the ROM address instead of selecting the outputs at the end. The signal N is three bits wide since it can have eight different values in the range 1 through 8. If N is used as part of the ROM address, the ROM size will become $2^3 = 8$ times as large. Since $\lceil N/2 \rceil$ address bits are required to select the right matrix coefficients depending on the input signals, no more than seven bits are required for the entire 1D SA-DCT to address the ROMs. Because the minimum address width of embedded ROMs within many modern FPGAs such as Altera FLEX10K [11] and Xilinx Virtex [12] is at least eight bits, the number of ROMs required is not increased. In fact, the embedded ROM blocks can be utilised more efficiently.

The static design of the 2D SA-DCT for MPEG-4 has been implemented on an Altera FLEX 10K130E device [11], in compliance to the MPEG-4 Verification Model [4]. 3721 (55%) of the Logic Cells (LCs) and all 16 EABs have been employed. A schematic of the implemented circuit is shown in Fig. 6. The module *Transpose* is used for the necessary matrix transposition between vertical and horizontal SA-DCT. The circuit runs at 47 MHz with a throughput of one value per two clock cycles. Using this clock frequency, a complete 2D SA-DCT can be performed in 4.47 µs.

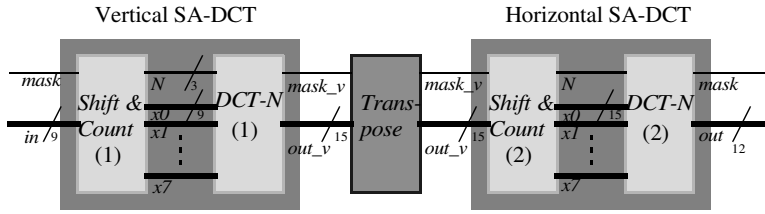


Fig. 6. Implementation of 2D SA-DCT

For the dynamic design, all eight possible instances of the DCT-N module have been implemented for Altera FLEX 10KE devices. The problems of the dynamic approach are the long configuration time of current FPGAs and the high number of reconfigurations necessary for a considerably small amount of processing time of a DCT-N computation. Provided that all possible values of *N* have the same frequency of occurrence, it takes on average approximately 21 ms to reconfigure an appropriate Altera FLEX 10K FPGA [11] compared to only 167 ns to compute one DCT-N. In this case the reconfiguration overhead would be approximately 125,000. Even with (partially) dynamically reconfigurable devices such as Xilinx XC6200 or Virtex FPGAs [12] the overhead is still large. For a Virtex FPGA, the average partial reconfiguration time for a DCT-N is about 420 µs, still 2,500 times longer than the time to compute a DCT-N. It takes 1.51 µs to compute a complete 2D SA-DCT if all parts of the dynamic design can work at their highest clock frequency and the reconfiguration time is not taken into account. Hence, to make the dynamic design quicker than the static design, the time for all 16 reconfigurations needs to be smaller than 2.96 µs, that is, 185 ns for one reconfiguration.

Reducing the reconfiguration time could be possible by using context switching FPGAs where a number of different configurations, which can be selected very rapidly, are stored on-chip [13]. An approach to reduce the number of reconfigurations could be realised by consecutively passing through all contour blocks of the object for a given *N* and performing the DCT-N operation only for this particular value of *N*. However, this approach introduces irregular data processing and memory usage.

6 Conclusion

We have presented two reconfigurable design approaches for a 2D Shape-Adaptive DCT, an example of a new type of multimedia video processing algorithm where the computations performed are data dependent. A static design, where the configuration data does not change during execution of the task, has been presented. The proposed use of a data dependence graph (DDG) allows a structured method for optimising sharable resources. By re-structuring the DDG and exploiting possible sharing of FPGA resources for different entities within the SA-DCT, it has been demonstrated that the area required for an implementation can be significantly reduced. An alternative dynamic approach has also been presented where the FPGA's configuration may change over

time. This is well suited to using dynamically reconfigurable logic but suffers from long reconfiguration overhead if currently available FPGAs are used.

Current and future work includes the development of a generic reconfigurability model in order to determine the conditions under which dynamic reconfiguration would be more attractive than static reconfiguration. Possible model parameters include the number and size of independently reconfigurable units, and the number, size, computation time and reconfiguration time of independent computations and their probability of occurrence, as well as the probability that one computation follows another.

Acknowledgements. This work was supported by the Department of Electrical and Electronic Engineering, Imperial College, and Sony Broadcast & Professional Europe.

References

1. Pirsch, P., Stolberg, H.-J.: VLSI Implementations of Image and Video Multimedia Processing Systems. IEEE Trans. Circuits Syst. Video Technol. **8** (1998) 878-891
2. MPEG Group: Overview of the MPEG-4 Standard. ISO/IEC JTC1/SC29/WG11 N2725 (1999)
3. Sikora, T., Makai, B.: Low Complexity Shape-Adaptive DCT for Generic Coding of Video. Proc. Workshop on Image Analysis and Image Coding (1994)
4. MPEG Group: MPEG-4 Video Verification Model Version 15.0. ISO/IEC JTC1/SC29/WG11 N3093 (1999)
5. Haynes, S.D., Stone, J., Cheung, P.Y.K., Luk, W.: Video Image Processing with the SONIC Architecture. IEEE Computer **33** (2000) 50-57
6. Le, T., Wendt, M., Glesner, M.: VLSI-Architecture of a Time-Recursive 2-D Shape-Adaptive DCT Processor for Generic Coding of Video. Proc. Intern. Conf. on Signal Processing Applications and Technology (1997) 1238-1242
7. Le, T., Glesner, M.: A New Flexible Architecture for Variable Length DCT Targeting Shape-Adaptive Transform. Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing **4** (1999) 1949-1952
8. Peled, A., Liu, B.: A New Hardware Realization of Digital Filters. IEEE Trans. Acoust., Speech, Signal Process. **22** (1974) 456-462
9. Sun, M.T., Wu, L., Liou, M.L.: A Concurrent Architecture for VLSI Implementation of Discrete Cosine Transform. IEEE Trans. Circuits Syst. **34** (1987) 992-994
10. Haynes, S.D., Cheung, P.Y.K.: A Reconfigurable Multiplier Array For Video Image Processing Tasks, Suitable For Embedding In An FPGA Structure. Proc. IEEE Symposium on FPGAs for Custom Computing Machines (1998) 226-234
11. Altera Inc.: FLEX 10KE Embedded Programmable Logic Family Data Sheet (1999)
12. Xilinx Inc.: Virtex^{TM} 2.5 V Field Programmable Gate Arrays (2000)
13. Chang, D., Marek-Sadowska, M.: Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs. IEEE Trans. on Computers **48** (1999) 565-578