FPGA Implementation of the 2D-DCT/IDCT for the Motion Picture Compression

Rastislav J.R. Struharik[1] and Ivan Mezei[2]

***Abstract* – In this paper architectures for the 2D DCT/IDCT (Discrete Cosine Transform, Inverse Discrete Cosine Transform) are presented. These architectures were developed for the FPGA implementation. First, algorithms for the efficient 2D DCT/IDCT calculation are presented. Using these algorithms micro-architectures for the efficient FPGA implementation are developed. These micro-architectures are then coded in the VHDL and synthesized using Xilinx Foundation ISE development system. Finally, maximum operating frequency and resources needed for the implementation of these cores are reported for the several families of Xilinx's FPGA IC's.**

***Keywords* – Image Compression, JPEG, 2D DCT/IDCT, VHDL, FPGA.**

I. INTRODUCTION

Compression is the process of reducing the size of the data sent, thereby, reducing the bandwidth required for the digital representation of a signal. Many inexpensive video and audio applications are made possible by the compression of signals. Compression technology can result in reduced transmission time due to less data being transmitted. It also decreases the storage requirements because there is less data. However, signal quality, implementation complexity, and the introduction of communication delay are potential negative factors that should be considered when choosing compression technology.

Video and audio signals can be compressed because of the spatial, spectral, and temporal correlation inherent in these signals. Spatial correlation is the correlation between neighboring samples in an image frame. Temporal refers to correlation between samples in different frames but in the same pixel position. Spectral correlation is the correlation between samples of the same source from multiple sensors.

There are two categories of compression: lossy and lossless. In medical system applications, image losses can translate into costly medical mistakes; therefore, lossless compression methods are used. Fortunately, the majority of video and image processing applications do not require the reconstructed data to be identical to the original data. In such applications, lossy compression schemes can be used to achieve higher compression ratios.

Discrete Cosine Transform (DCT) [1] is a lossy compression scheme where an N x N image block is transformed from the spatial domain to the DCT domain. DCT decomposes the signal into spatial frequency components called DCT coefficients. The lower frequency DCT coefficients appear toward the upper left-hand corner of the DCT matrix, and the higher frequency coefficients are in the lower right-hand corner of the DCT matrix. Because the human visual system is less sensitive to errors in high frequency coefficients than it is to lower frequency coefficients, the higher frequency components can be more finely quantized, or even completely discarded. This operation leads to the significant improvements of the compression ratio, thereby reducing the amount of data that needs to be transmitted or stored, with only moderate degradation of the original picture quality.

For most image compression standards, N = 8. An 8 x 8 block size does not have significant memory requirements, and furthermore, a block size greater than 8 x 8 does not offer significantly better compression.

DCT is image independent and can be performed with fast algorithms. Examples of standards using DCT:

- Dolby AC2 & AC3: 1-D DCT (and 1-D Discrete Sine Transform)
- JPEG (still images): 2-D DCT spatial compression
- MPEG1 & MPEG2: 2-D DCT plus motion compensation
- H.261 and H.263: moving image compression for video conferencing and video telephony

Much of the processing required to encode or decode video using these standards is taken up by calculating the DCT and/or IDCT. An efficient hardware block dedicated to these functions will improve the performance of the digital video system considerably.

II. EFFICIENT ALGORITHMS FOR THE 2D DCT/IDCT CALCULATION

A. Algorithm for the Efficient 2D DCT Calculation

The algorithm used for the calculation of the 2D DCT is based on the following equation:

$$Y_{pq} = \frac{c(p)c(q)}{4} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{mn} \cos\left(\frac{\pi(2m+1)p}{2M}\right) \cos\left(\frac{\pi(2n+1)q}{2N}\right) \quad (1)$$

where:

[1]Rastislav J.R. Struharik is with the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia, E-mail: rasti@eunet.yu

[2]Ivan Mezei is with the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia, E-mail: imezei@uns.ns.ac.yu

$$c(p)=\frac{1}{\sqrt{2}}\ for\ p=0, c(p)=1\ otherwise$$
$$c(q)=\frac{1}{\sqrt{2}}\ for\ q=0,\ c(q)=1\ otherwise \quad (2)$$

Efficient implementation of this equation is possible because 2D DCT can be separated into two 1D DCT [2]. First, the 1D DCT of the rows are calculated and then the 1D DCT of the columns are calculated. The 1D DCT coefficients for the rows and columns can be calculated by separating equation (1) into the row part and the column part. As stated before, for most image compression standards, *M=N=8*. Using vector processing, the output Y of an 8 x 8 DCT for input X is given by the following equation.

$$Y = C \cdot X \cdot C^t \quad (3)$$

C is the cosine coefficients and C^t are the transpose coefficients. This equation can also be written as Y=C·Z, where Z = X·C^t. Coefficients for the C and C^t matrix can be calculated using the following equations.

$$C_{ij}=K\cos\frac{(2i+1)j\pi}{2M}, K=\frac{1}{N}\ for\ j=0, K=\frac{\sqrt{2}}{N}\ else$$
$$C^t_{ij}=K\cos\frac{(2j+1)i\pi}{2N}, K=\frac{1}{M}\ for\ i=0, K=\frac{\sqrt{2}}{M}\ else \quad (4)$$

Values for the scaled and rounded coefficients are presented in the following equation.

$$C=\begin{bmatrix} 23170 & 23170 & 23170 & 23170 & 23170 & 23170 & 23170 & 23170 \\ 32138 & 27246 & 18205 & 6393 & -6393 & -18205 & -27246 & -32138 \\ 30274 & 12540 & -12540 & -30274 & -30274 & -12540 & 12540 & 30274 \\ 27246 & -6393 & -32138 & -18205 & -18205 & 32138 & 6393 & -27246 \\ 23170 & -23170 & -23170 & 23170 & 23170 & -23170 & -23170 & 23170 \\ 18205 & -32138 & 6393 & 27246 & 27246 & -6393 & 32138 & -18205 \\ 12540 & -30274 & 30274 & -12540 & -12540 & 30274 & -30274 & 12540 \\ 6393 & -18205 & 27246 & -32138 & -32138 & -27246 & 18205 & -6393 \end{bmatrix} \quad (5)$$

$$C^t=\begin{bmatrix} 23170 & 32138 & 30274 & 27246 & 23170 & 18205 & 12540 & 6393 \\ 23170 & 27246 & 12540 & -6393 & -23170 & -32138 & -30274 & -18205 \\ 23170 & 18205 & -12540 & -32138 & -23170 & 6393 & 30274 & 27240 \\ 23170 & 6393 & -30274 & -18205 & 23170 & 27246 & -12540 & -32138 \\ 23170 & -6393 & -30274 & 18205 & 23170 & -27246 & -12540 & 32138 \\ 23170 & -18205 & -12540 & 32138 & -23170 & -6393 & 30274 & -27246 \\ 23170 & -27246 & 12540 & 6393 & -23170 & 23138 & -30274 & 18205 \\ 23170 & -32138 & 30247 & -27246 & 23170 & -18205 & 12540 & -6393 \end{bmatrix}$$

Structure of the 2D DCT core using this decomposition is presented on the following figure.

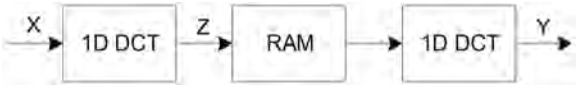

Fig. 1 Efficient 2D DCT Implementation

Let us explain the way the Z matrix is calculated. Each element in the first row of the input matrix X are multiplied by each element in the first column of matrix C^t and added together to get the first value Z_{00} of the intermediate matrix Z. To get Z_{01}, each element of row zero in X is multiplied by each element in the first column of C^t and added and so on. The calculation can be implemented using eight multipliers and storing the coefficients in ROMs. But, after a closer examination of the coefficients from the C^t matrix there is a way to half the number of multipliers. When the equation Z = X·C^t is written in the scalar form we get following equations.

$$\begin{aligned}
Z_{k0} &= 23170(x_{k0}+x_{k1}+x_{k2}+x_{k3}+x_{k4}+x_{k5}+x_{k6}+x_{k7})\\
Z_{k1} &= 32138(x_{k0}-x_{k7})+27246(x_{k1}-x_{k6})+18205(x_{k2}-x_{k5})+6393(x_{k3}-x_{k4})\\
Z_{k2} &= 30274(x_{k0}+x_{k7})+12540(x_{k1}+x_{k6})-12540(x_{k2}+x_{k5})-30274(x_{k3}+x_{k4})\\
Z_{k3} &= 27246(x_{k0}-x_{k7})-6393(x_{k1}-x_{k6})-32138(x_{k2}-x_{k5})-18205(x_{k3}-x_{k4})\\
Z_{k4} &= 23170(x_{k0}+x_{k7})-23170(x_{k1}+x_{k6})-23170(x_{k2}+x_{k5})+23170(x_{k3}+x_{k4})\\
Z_{k5} &= 18205(x_{k0}-x_{k7})-32138(x_{k1}-x_{k6})+6393(x_{k2}-x_{k5})+27246(x_{k3}-x_{k4})\\
Z_{k6} &= 12540(x_{k0}+x_{k7})-30274(x_{k1}+x_{k6})+30274(x_{k2}+x_{k5})-12540(x_{k3}+x_{k4})\\
Z_{k7} &= 6393(x_{k0}-x_{k7})-18205(x_{k1}-x_{k6})+27246(x_{k2}-x_{k5})-32138(x_{k3}-x_{k4})\\
k &= 0, 1, ..., 7
\end{aligned} \quad (7)$$

We can see that for example, input values x_{k0} and x_{k7} are always multiplied by the same coefficient, only the sign can change. This can be efficiently explored to reduce the number of multipliers as shown on the following figure.

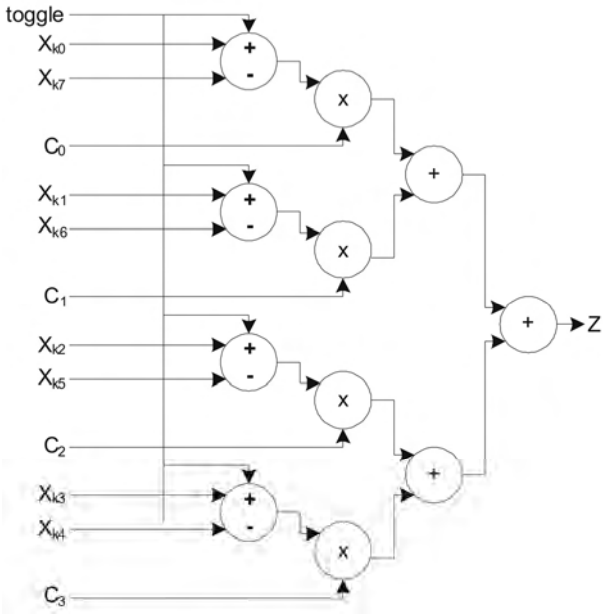

Fig. 2 Efficient 1D DCT Implementation

Using the *toggle* signal the adder/subtractor modules can be configured to operate as adder or as subtractor depending on the current need.

All 64 values for the matrix Z can be calculated in 64 clock cycles. These values are stored in the RAM memory shown between two 1D DCT block on Fig. 1. Using these stored values as input, second 1D DCT is performed resulting in the matrix Y. Structure of this second 1D DCT block is similar to the structure shown on the Fig. 2. Matrix Y holds the values for the 2D DCT transform of the input matrix X.

B. Algorithm for the Efficient 2D IDCT Calculation

Now let us examine the problem of calculating the 2D IDCT. Using the 2D DCT matrix Y, original input matrix X can be calculated in the following way [3].

$$X = C^t \cdot Y \cdot C \tag{7}$$

Matrix C and C^t are identical to those from 2D DCT. We can see that the only difference between Eq. (3) and (7) is in the order by which the matrix C and C^t are applied. Although this seems to be only a minor difference, it turns out to be a significant one, because now we cannot explore the coefficient symmetries like in the case of the 2D DCT. Once more the Eq. (7) can be split into two simpler equations, $X=C^t \cdot Z$, and $Z = Y \cdot C$. Because the different order of matrix multiplications, we cannot find the similar symmetry between the coefficients during the 1D IDCT operations. This means that every 1D IDCT block will now use eight multipliers. Structure of the 1D IDCT block is presented on the Fig. 3.

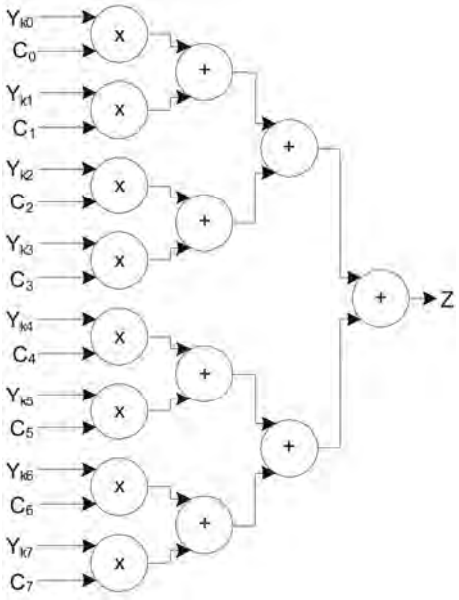

Fig. 3 Structure of the 1D IDCT Module

Basic structure and operation of the 2D IDCT core is identical to that of 2D DCT core presented on Fig. 1.

C. Algorithm for the Quantization/Dequantization Operations

As stated before, to improve the compression ratio it is common practice to perform the quantization of DCT components. Quantization is the process of selectively discarding visual information without a significant loss in the visual effect. Quantization reduces the number of bits needed to store an integer value by reducing the precision of the integer. Each DCT component is divided by a separate quantization coefficient, and rounded to the nearest integer. The larger the quantization coefficient (i.e., coefficient weighting), the smaller the resulting answer and associated bits needed to express the DCT component. In the reverse process, the fractional bits are "rounded" and are recovered as zeros, constituting a precision loss from the original number.

There are several different recommended procedures to perform the quantization. We have opted for the procedure used in the MPEG-2 compression standard [4]. Since we used a 12-bit representation of the DCT components, DC value was not quantized. For the AC components following formula is used to determine the value of the quantization factor.

$$QDCT_{ij} = \frac{\left\lfloor \frac{32 \cdot DCT_{ij}}{Qmatrix_{ij} \cdot Qscale} \right\rfloor}{2} \tag{8}$$

Value of the *Qmatrix* is the matrix of the quantization coefficients with the following values depending whether we are quantizing luminance or chrominance components.

$$Qmatrix_{luma} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

$$Qmatrix_{chroma} = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix} \tag{9}$$

Value of the *Qscale* parameter enables easy modifications of the quantization factors that will be used during the quantization. Dequantization is performed using the inverse expression of the expression in Eq. (8).

III. ARCHITECTURE OF THE DEVELOPED 2D DCT/IDCT CORES

After reviewing the efficient algorithms for the 2D DCT/IDCT calculations we can now present the basic architecture for the two developed cores, together with their interfaces.

Fig. 4 presents the interfaces for the 2D DCT and 2D IDCT cores.

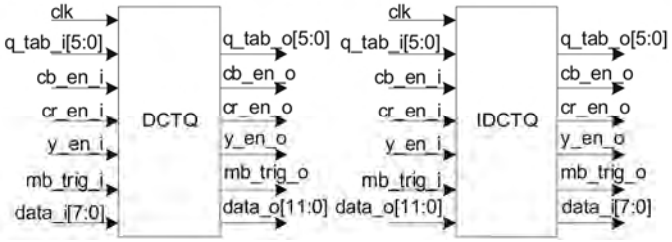

Fig. 4 Interface of the 2D DCT and 2D IDCT Cores

As can be seen from the Fig. 4 interface of both cores is the same. Input signal *q_tab_i[5:0]* is used to define the required

value of the *Qscale* parameter in the quantization and dequantization operations. Input signals, *cb_en_i*, *cr_en_i* and *y_en_i* are used to specify the type of the current 8x8 block. Both cores assume that the picture is represented in the YCbCr format. Signal *mb_trig_i* is a global synchronizing signal used to indicate the start of the next 8x8 block. Input signal *data_i[7:0]* (*data_i[11:0]* in the case of 2D IDCT core) holds the pixel values (DCT component values in case of 2D IDCT core). Meaning of every output signal is identical to the input signal with the same name. Fig. 5 presents the basic architecture for the 2D DCT/IDCT cores.

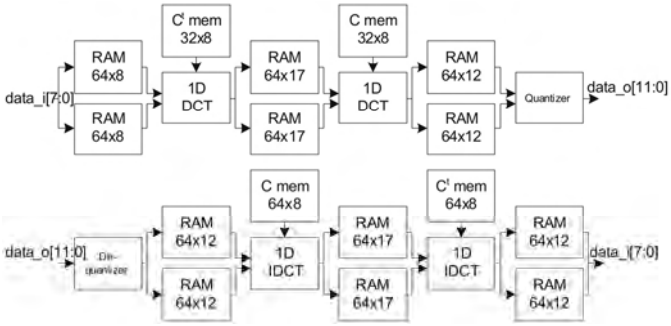

Fig. 5 Interface of the 2D DCT (up) and 2D IDCT (down) Cores

In every core there are three pipeline stages. This enables efficient calculation of the DCT or IDCT values that requires only 64 clock cycles per one 8x8 block. This is the maximum speed at which both cores can operate, but if needed they can work at slower speed. Fig. 6 illustrates the typical waveforms for the characteristic signals.

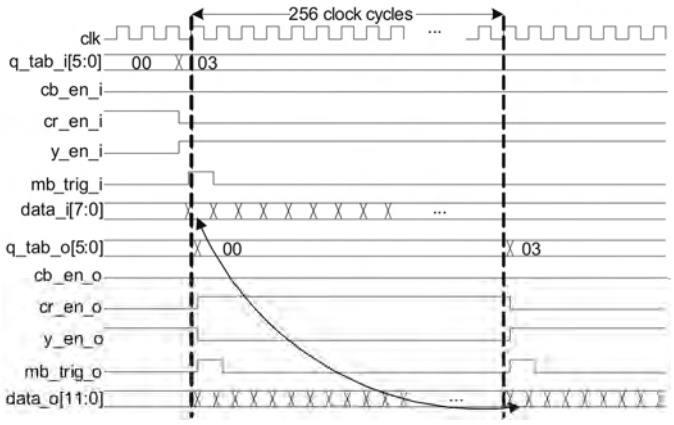

Fig. 6 Typical waveforms of the interface signals for the 2D DCT/IDCT Cores

Previous figure illustrates the interface signal waveforms in case of 256 clock cycle duration of one 8x8 block.

IV. Synthesis results

Both cores were coded in VHDL and synthesized using Xilinx Foundation ISE 6.2i software. Following table presents the obtained results in terms or core size (number of slices required to implement the core) and maximum operating frequency for several Xilinx FPGA families.

TABLE I
SYNTHESIS RESULTS (OPTIMIZATION GOAL: SIZE)

FPGA Family	2D DCT Core		2D IDCT Core	
	Core Size (# slices)	Core Speed	Core Size (# slices)	Core Speed
SpartanIIE	3777	24.9 MHz	4525	22.7 MHz
SpartanIII	1214	47.8 MHz	1222	43.4 MHz
Virtex	3776	18.8 MHz	4525	17.3 MHz
Virtex2Pro	977	43.7 MHz	971	40.1 MHz

TABLE II
SYNTHESIS RESULTS (OPTIMIZATION GOAL: SPEED)

FPGA Family	2D DCT Core		2D IDCT Core	
	Core Size (# slices)	Core Speed	Core Size (# slices)	Core Speed
SpartanIIE	3794	28.4 MHz	4657	26.0 MHz
SpartanIII	1268	51.1 MHz	1264	47.4 MHz
Virtex	3850	22.3 MHz	4657	21.0 MHz
Virtex2Pro	1036	46.2 MHz	1059	43.5 MHz

Significantly smaller cores sizes in case of the SpartanIII and Virtex2Pro FPGA families are due to the fact that these families have dedicated multipliers that can be used to implement all the multiplications required in the DCT/IDCT calculations. In contrast, SpartanIIE and Virtex families don't have dedicated multipliers embedded on the chip, so every multiplier has to be implement using the general purpose logic resources, resulting in larger core sizes.

V. Conclusion

In this paper hardware implementation of the 2D DCT/IDCT cores was investigated. Efficient algorithms for the calculation of the 2D DCT/IDCT values were proposed. These algorithms were implemented in hardware using the FPGA technology. Using the Xilinx Foundation ISE software synthesis results for several available FPGA families were reported.

References

[1] M. Popović, "Digitalna Obrada Signala", Beograd, Nauka, 1997.
[2] Xilinx Application Note XAPP610, "Video Compression Using DCT"
[3] Xilinx Application Note XAPP611, "Video Decompression Using IDCT"
[4] Xilinx Application Note XAPP615, "Quantization"