Hindawi Publishing Corporation
Advances in Computer Engineering
Volume 2015, Article ID 405856, 10 pages
http://dx.doi.org/10.1155/2015/405856

Research Article

High Performance Discrete Cosine Transform Operator Using Multimedia Oriented Subword Parallelism

Shafqat Khan, Emmanuel Casseau, and Daniel Menard

IRISA Lab-CAIRN, CNRS UMR 6074, Lannion, France

Correspondence should be addressed to Shafqat Khan; shafqat_aero@yahoo.com

Received 27 August 2014; Revised 18 December 2014; Accepted 9 January 2015

Academic Editor: Jenhui Chen

In this paper an efficient two-dimensional discrete cosine transform (DCT) operator is proposed for multimedia applications. It is based on the DCT operator proposed in Kovac and Ranganathan, 1995. Speed-up is obtained by using multimedia oriented subword parallelism (SWP). Rather than operating on a single pixel, the SWP-based DCT operator performs parallel computations on multiple pixels packed in word size input registers so that the performance of the operator is increased. Special emphasis is made to increase the coordination between pixel sizes and subword sizes to maximize resource utilization rate. Rather than using classical subword sizes (8, 16, and 32 bits), multimedia oriented subword sizes (8, 10, 12, and 16 bits) are used in the proposed DCT operator. The proposed SWP DCT operator unit can be used as a coprocessor for multimedia applications.

1. Introduction

In most multimedia applications, image/video data needs to be transmitted. For efficient utilization of channel bandwidth, image compression is done before the transmission. Image compression is the process of reducing the size of the image without degrading the quality of the image to unacceptable limits. The compressed images/videos require less memory to store and less time to transmit. Discrete cosine transform (DCT) is the most widely used operation in video/image compression. Due to the computational efficiency of DCT, different compression standards like JPEG, MPEG, H263.1, and so forth, use cosine function to convert the input pixel values to frequency domain. In the frequency domain, quantization of image is performed to obtain the compression. The quantization is done to such an extent that the image can be reconstructed from compressed data with minimum distortion. Usually the compression ratio depends upon the requirements of quality of the reconstructed image.

Different algorithms and hardware architectures have been proposed to speed up the DCT computation process. Due to the symmetrical nature of computations, these algorithms use different matrix properties to reduce the computational complexity. In [1–3], architectures for two-dimensional DCT for JPEG image compression have been presented. They are based on one algorithm which calculates the DCT in one dimension (1D-DCT) and the 2D-DCT calculation is made using its separability property. This algorithm aims to minimize the number of arithmetic operations to obtain the DCT values. In [4], implementation of DCT and inverse DCT is proposed using Weiping Li algorithm. The resultant design has regular structure and it requires seven constant parallel multipliers. In [5] the modified Loeffler algorithm (11 MUL and 29 ADD) is presented for DCT computations. In [6], fast parallel algorithms for the DCT-kernel-based real-valued discrete Gabor transform (RDGT) and its inverse transform are presented based on multirate signal processing. Similarly in [7], hardware implementations of (8×8) DCT and IDCT on different FPGA technologies are presented.

The performance of DCT implementations can be further improved by introducing the parallelism in arithmetic operations at pixel level. For this purpose, multimedia oriented SWP capability can be used in the implementation of DCT operator. In SWP [8, 9], subwords are packed in word size registers and same computations are performed on packed subwords. In this paper a SWP-based operator is proposed for DCT computations. Along with algorithmic efficiency, the performance of the DCT operator is increased through

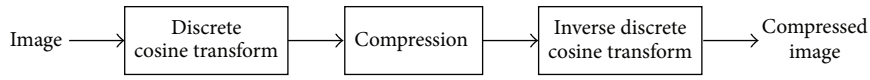

Figure 1: Image compression process.

parallel computations on packed pixel data. In the proposed DCT operator, the coordination between pixel sizes and subword sizes is increased by using multimedia oriented subword sizes (8, 10, 12, and 16 bits) rather than classical subword sizes (8, 16, 32 bits, etc.). This coordination further enhances efficiency through a better resource utilization.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the DCT and the 8-point DCT operator used for computations. Section 3 presents the multimedia oriented SWP technique which is used in the SWP DCT operator. Section 4 explains the architecture of the two-dimensional SWP DCT operator. The functions of the different units used in the SWP DCT operator are also explained in detail. Section 5 presents the control unit for two-dimensional DCT computations. Section 6 describes the synthesis results of the SWP DCT operator. The performance analysis of the SWP DCT operator is also presented in this section. Finally the conclusion is presented in Section 7.

2. Discrete Cosine Transform

In an image, most of the useful information is stored at low frequencies. On the other hand, the information like sharp edges, abrupt transitions, and so forth is stored at higher frequencies. Discarding certain information at higher frequencies has less effect on the overall quality of the image. Therefore, in the compression process, pixels at higher frequencies are discarded up to certain extent without causing substantial degradation in the quality of the image. The pixel values in the image are stored in time or spatial domain. In the first step, the image is converted from time domain to frequency domain using DCT. The process of image compression is done in frequency domain. As per the requirements of compression ratio, certain information at higher frequencies is discarded. This loss of information reduces the quality of image to some extent. This type of compression is called lossy compression because the lost information can not be recovered. The compressed image can be stored in less memory space or it can be transmitted in less time compared to uncompressed image. The compressed image can be converted to the time or spatial domain using inverse DCT. This process is shown in Figure 1.

Instead of applying DCT on the whole image, it is usually applied on smaller blocks of 8×8 size. In each (8×8) block, DCT operation is applied in both horizontal and vertical directions. This is called two-dimensional DCT (2D-DCT). First the DCT operation is applied in row-wise direction and then in column-wise direction. To reduce the complexity, the column-wise DCT operation can also be carried out using row-wise DCT operator. For this purpose, the DCT operation is first applied on the rows of the image block followed by the transpose operation and then the row-wise DCT operation is performed once again. This process is shown in Figure 2.

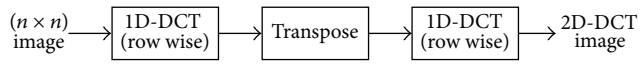

Figure 2: Two-dimensional DCT operation.

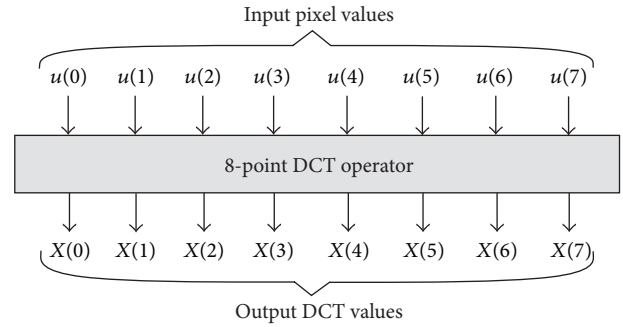

Figure 3: Block diagram of 8-point DCT.

By the use of the transpose operation, the two-dimensional DCT is performed by row-wise DCT operators only. The DCT values for each row of (8×8) image block are calculated using the *8-point DCT* operator. This operator takes 8 pixels as an input and generates eight corresponding DCT values.

DCT operation has high degree of computational complexity. The mathematical expression for performing N-point DCT operation is given in

$$X(k) = \alpha(k) \sum_{n=0}^{N-1} u(n) \frac{\cos(2n+1)}{2N} k\pi, \tag{1}$$

where

$$\alpha(0) = \sqrt{\frac{1}{N}},$$
$$\alpha(k) = \sqrt{\frac{2}{N}}, \tag{2}$$
$$1 \le k \le N-1.$$

For a 8-point DCT operation, N equals 8 in (1). The inputs to the *8-point DCT* operator are eight pixel values $(u(0), \ldots, u(7))$. After performing the DCT computation, eight DCT values $(X(0), \ldots, X(7))$ are obtained at the output of the operator. For computing each DCT value, the values of all the eight input values $(u(0), \ldots, u(7))$ are utilized. The block diagram of the *8-point DCT* operator is shown i n Figure 3.

The operation shown in Figure 3 is performed on each row of the (8×8) image block. Direct implementation of DCT

Table 1: 8-point DCT computations.

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
$b_0 = a_0 + a_7$	$c_0 = b_0 + b_5$	$d_0 = c_0 + c_3$	$e_0 = d_0$	$f_0 = e_0$	$S_0 = f_0$
$b_1 = a_1 + a_6$	$c_1 = b_1 - b_4$	$d_1 = c_0 - c_3$	$e_1 = d_1$	$f_1 = e_1$	$S_1 = f_4 + f_7$
$b_2 = a_2 - a_4$	$c_2 = b_2 + b_6$	$d_2 = c_2$	$e_2 = m_3 \times d_2$	$f_2 = e_5 + e_6$	$S_2 = f_2$
$b_3 = a_1 - a_6$	$c_3 = b_1 + b_4$	$d_3 = c_1 + c_4$	$e_3 = m_1 \times d_7$	$f_3 = e_5 - e_6$	$S_3 = f_5 - f_6$
$b_4 = a_2 + a_5$	$c_4 = b_0 - b_5$	$d_4 = c_2 - c_5$	$e_4 = m_4 \times d_6$	$f_4 = e_3 + e_8$	$S_4 = f_1$
$b_5 = a_3 + a_4$	$c_5 = b_3 + b_7$	$d_5 = c_4$	$e_5 = d_5$	$f_5 = e_8 - e_3$	$S_5 = f_5 + f_6$
$b_6 = a_2 - a_5$	$c_6 = b_3 + b_6$	$d_6 = c_5$	$e_6 = m_1 \times d_3$	$f_6 = e_2 + e_7$	$S_6 = f_3$
$b_7 = a_0 - a_7$	$c_7 = b_7$	$d_7 = c_6$	$e_7 = m_2 \times d_4$	$f_7 = e_4 + e_7$	$S_7 = f_4 - f_7$
		$d_8 = c_7$	$e_8 = d_8$		

using (1) requires lot of hardware resources. Therefore different methods have been proposed to reduce the complexity of 8-point DCT operations [2, 4, 5, 7, 10, 11].

2.1. 8-Point DCT Operator. The architecture of the 8-point DCT unit selected for SWP enhancements was originally proposed in [1, 12]. This architecture implements image compression specific simplifications to provide a high performance 2D-DCT operator. It uses 29 additions and 5 multiplications to compute 1D 8-point DCT operation. Therefore, for two-dimensional DCT operation on (8×8) image block, it requires 40 multiplications and 464 additions. The basic algorithm used in [1, 2] for 8-point DCT operation is shown in Table 1. It consists of six steps of computations. Each step computes the values based on the outputs from the previous step so that the process can be pipelined. The computations are performed using addition, subtraction, and multiplication operators.

The inputs are eight 8-bit pixels $(a_0, \ldots, a_7)$ from an 8×8 image block. In each step, arithmetic operations are performed on the data provided by the previous step. In the first three steps, 21 addition/subtraction operations are performed. In the fourth step, the multiplications with constants m_1, m_2, m_3, and m_4 are done using four multiplier units. The values of these constants are calculated off line using the following equations:

$$m_1 = \cos\left(\frac{4\pi}{16}\right), \qquad m_2 = \cos\left(\frac{6\pi}{16}\right),$$
$$m_3 = \cos\left(\frac{2\pi}{16}\right) - \cos\left(\frac{6\pi}{16}\right), \tag{3}$$
$$m_4 = \cos\left(\frac{2\pi}{16}\right) + \cos\left(\frac{6\pi}{16}\right).$$

The integer and fraction parts of each of these constants are represented by the appropriate number of bits. At the output of the multiplier operators, the products are obtained and contain the integer part as well as the fraction part. When moderate accuracy is targeted, the fraction part at this stage can be ignored to reduce data bit-width. However, for greater accuracy, the fraction part can also be retained to be used in the successive computations. In the fifth and sixth steps, 10 addition/subtraction operations are performed. At the output, eight DCT values $S_0, \ldots, S_7$ are obtained corresponding to eight input pixel values. To obtain 1D-DCT values for an 8×8 image block, the same procedure is repeated for each row in the image block. For 2D-DCT computations, the 1D-DCT computations are used again after the transpose operation.

3. Subword Parallelism SWP Operators

To increase the performance of processors, SWP technique has been carried out on the basic arithmetic operators (ADD, SUB, MULT, etc.) [13–15]. These basic SWP operators perform parallel operations on subwords which are conveniently compatible with the word size of the processor. By doing this, the processor can achieve more parallelism rather than wasting the word size datapath and register sizes when operating on low-precision data [16–18]. Conventionally the word size of processor is a multiple of subword sizes which helps to reduce the complexity of SWP operators. For instance, some of the conventional subword sizes for 64-bit processors are 8, 16, and 32 bits. As an example, Figure 4 shows four basic addition operations on 16-bit subword in a 64-bit processor.

3.1. Word and Subword Sizes. In multimedia applications, the sizes of the input data (pixels) for computations are 8, 10, 12, or sometimes 16 bits. These multimedia data sizes are not in coordination with existing processor's subword sizes resulting in the under utilization of processor's resources [9, 19]. To increase the performance of multimedia applications, our proposed SWP DCT operator considers multimedia oriented subword sizes rather than conventional subword sizes. This operator can perform operations on word size operands (40 bits) as well as on subwords (five 8-bit subwords or four 10-bit subwords or three 12-bit subwords or two 16-bit subwords) packed in word size registers. Word size of 40 bits is chosen because it gives a good tradeoff between parallelism degree, hardware complexity, and efficient use. It also ensures a better resource utilization rate with different multimedia oriented pixel sizes compared to a usual 32-bit word size as shown in Table 2. The size of the input data is given in the first column. The number of OPs stands for the number of operations performed in parallel with the selected input size. With these multimedia oriented word and subword sizes, the minimum resource wastage (0%) occurs when the selected subword size is either 8 bits or 10 bits and the worst case resource wastage

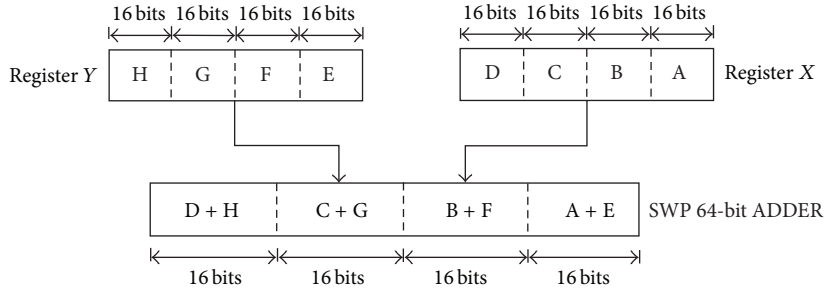

Figure 4: Adder with 16-bit subwords.

Table 2: Resource utilization rate of conventional SWP operator (32 bits) versus our proposed SWP operator (40 bits).

	Conventional (32 bits)		Dedicated (40 bits)	
	Number of OPs	Use ratio %	Number of OPs	Use ratio %
$a(8)$ OP $b(8)$	4	100	5	100
$a(10)$ OP $b(10)$	2	62	4	100
$a(12)$ OP $b(12)$	2	75	3	90
$a(16)$ OP $b(16)$	2	100	2	80

(20%) occurs when selected subword size is 16 bits (i.e., 16-bit pixels) which is less used in practice.

4. SWP DCT Operator

The input to the SWP DCT operator is an 8×8 image block which contains pixel values. The *SWP 2D-DCT* operator performs DCT computations on multimedia oriented pixel sizes of 8, 10, 12, or 16 bits. These sizes are selected based upon the requirements in modern multimedia applications. Parallel computations are performed on multiple pixels which are packed in word size registers. Based on the pixel sizes, the subword size is selected using SWP control signals (SWP_{ctrl}). These control signals direct the SWP arithmetic units about the size of pixels packed in the input registers. Due to the parallel processing of subwords, two-dimensional DCT computations are performed faster. The block diagram of the SWP DCT operator is shown in Figure 5. Each block in Figure 5 performs particular functions to compute two-dimensional DCT of (8×8) image block. The *CLK* signal and synchronous *Reset* signal are provided to each block. In the next few subsections, the functionality of each block is given in detail.

4.1. Input (8×8) Image Block. The SWP DCT operator input consists of an 8×8 image block. We assumed the 8×8 image block is stored in a memory. The size of each pixel can be either 8, 10, 12, or 16 bits. Based upon the pixel sizes, the subword size is selected for SWP arithmetic units used in the SWP DCT operator. These SWP units perform parallel operations on packed pixels.

4.2. Vector Formation for 1D-DCT Unit. In order to perform parallel computations, the pixels from an 8×8 input image block are packed in 40-bit word size registers. The input data from an 8×8 image block is first arranged in eight 40-bit vectors $(a_0, \ldots, a_7)$. Each vector contains p pixels from one particular column of an 8×8 image block. For instance vector a_0 contains p successive pixels from column 0 of an 8×8 image block. Similarly, the other vectors (a_1 to a_7) contain p pixels from corresponding columns. The number of pixels p which are packed in each 40-bit word size vector depends upon the size of the pixels. For 8-bit pixels, each 40-bit vector contains five pixels ($p = 5$). Therefore, the pixels from the first five rows of an 8×8 image block are packed in eight 40-bit vectors. For 10-bit pixels, each 40-bit vector contains four pixels ($p = 4$). Therefore the pixels from the first four rows of an 8×8 image block are packed in eight 40-bit vectors. Similarly for 12- and 16-bit pixel sizes the packing is done accordingly with $p = 3$ and $p = 2$, respectively. This process is shown in Figure 6 for pixel size of 8 and 10 bits.

As shown in Figure 6, for 8-bit subword size, eight 40-bit vectors are formed using pixels from top five rows of input an 8×8 image block. Subsequently, arithmetic computations are performed on eight 40-bit vectors in parallel using the *SWP 8-point DCT* unit. Pixels from the remaining rows are packed in the same fashion for parallel processing. For 10-bit subword size, eight 40-bit vectors are constituted using pixels from four successive rows of an (8×8) input image block. As the subword size increases, the number of pixels which are packed in each 40-bit vector reduces accordingly which ultimately reduces the parallelism.

The packing process is performed in p main steps. Each step, 8 pixels are sequentially read from the image block to be stored in a serial to parallel shift register of length 8. They enter this register serially so to be available in parallel at the register output at the end of each step for the packing process. As far as the vector formation is concerned, the pixels are read from the memory in a row-wise fashion. In the first step, the pixels of the first row are read from the memory and stored in the eight 40-bit input vectors at appropriate location. Similarly in the second step, the second row of pixels is read from the memory and stored in the input vectors. Based upon the pixel/subword size (8, 10, 12, or 16 bits), this reading and packing process continues in a pipeline fashion until the eight 40-bit vectors are fully packed with p pixel values.

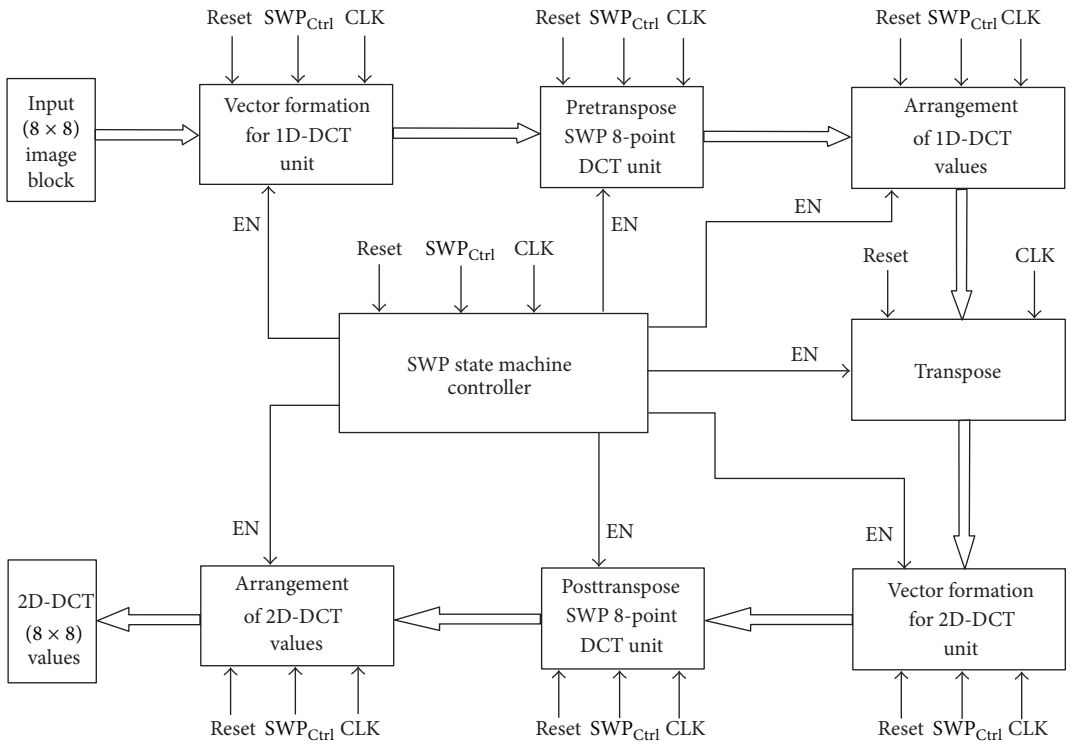

Figure 5: SWP 2D-DCT operator.

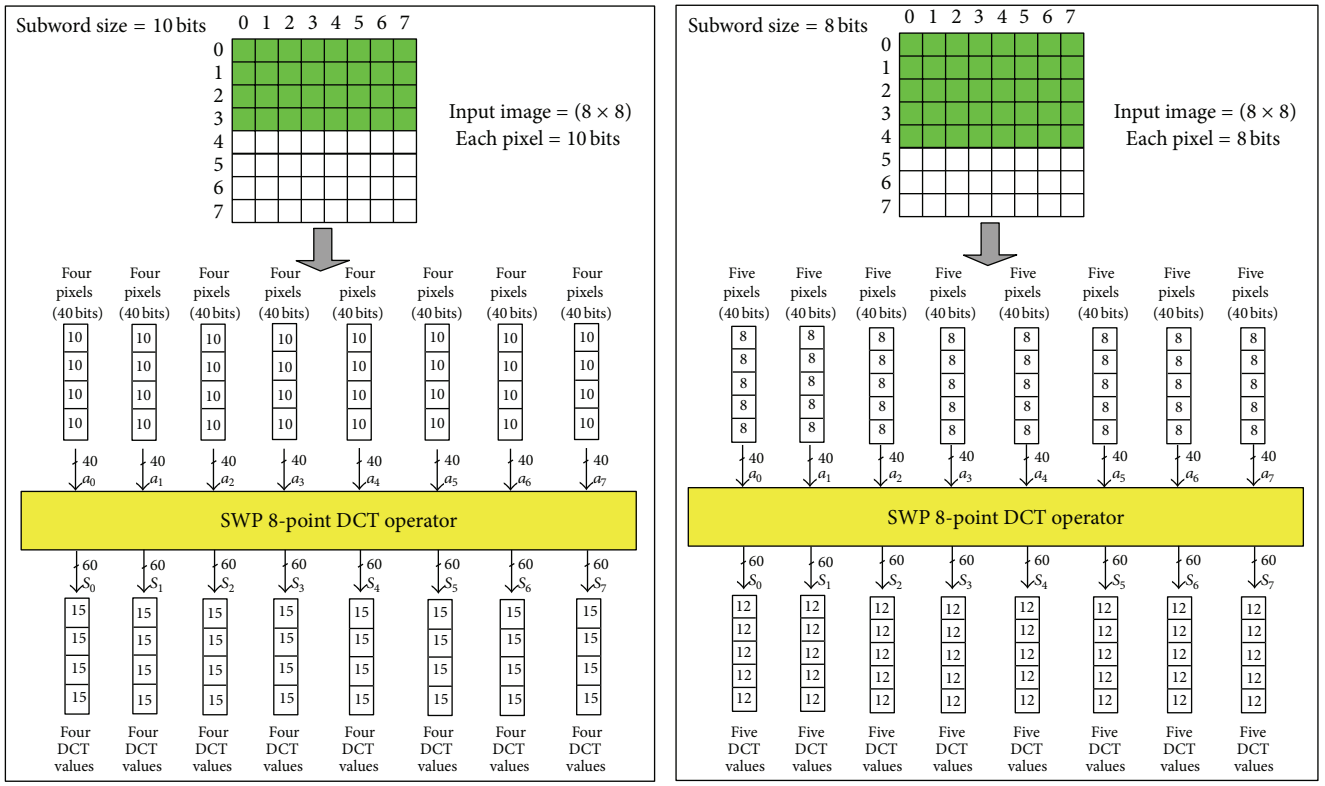

Figure 6: Formation of vectors for different subword sizes.

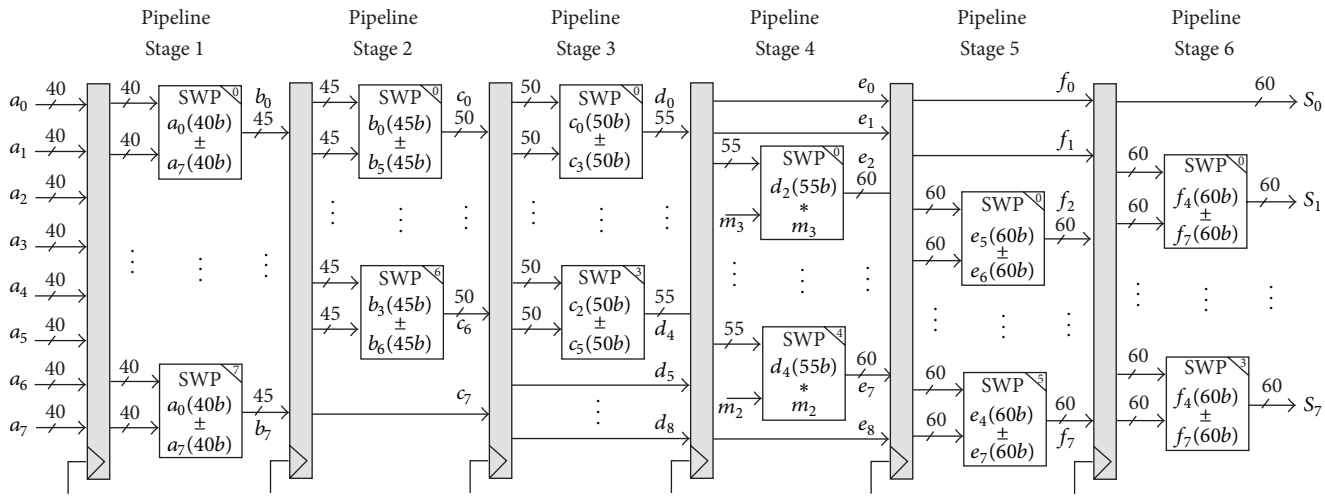

Figure 7: Pretranspose SWP 8-point DCT.

4.3. Pretranspose SWP 8-Point DCT Unit. The *pretranspose SWP 8-point DCT* unit is used to perform 1D-DCT computations on eight 40-bit vectors. Instead of operating on one single row of an 8 × 8 image block, the *pretranspose SWP 8-point DCT* unit performs operations on multiple rows in each instantiation. The number of rows on which the DCT operations are performed in parallel is equal to p; that is, the smaller the subword size the higher the number of rows that can be processed simultaneously. The architecture and bit-width requirements at each pipelined stage of the *pretranspose SWP 8-point DCT* unit are shown in Figure 7.

The *pretranspose SWP DCT* unit inputs are eight 40-bit vectors $(a_0, \ldots, a_7)$. Depending on the selected subword size, each input vector contains either five 8-bit pixels or four 10-bit pixels or three 12-bit pixels or two 16-bit pixels. So the pixels packed in each vector correspond to different rows in an 8×8 input image block. Therefore multiple rows are processed in parallel using the *pretranspose SWP DCT* unit. The details of SWP operations performed in each pipelined stage are given below.

Pipelined Stage 1. These stage inputs are eight 40-bit vectors $(a_0, \ldots, a_7)$ which contain packed pixels. Addition/subtraction operations are performed on these data vectors using eight SWP arithmetic units and generate eight 45-bit vectors $(b_0, \ldots, b_7)$. Five guard bits are allocated to each output vector to avoid any overflow. The number of these guard bits is selected on the basis of minimum subword size (8-bit) supported by the operator and largest parallelism degree (5 subwords are handled in parallel in this case). For 8-bit subword size, one extra bit is allocated to each of the five packed subwords, which results in an output of 45 bits. Five guard bits are sufficient to avoid overflow for other larger subword sizes (10, 12, and 16 bits) with less parallelism degree.

Pipelined Stage 2. In this stage, seven SWP arithmetic units are used to perform addition/subtraction operations on eight 45-bit vectors $(b_0, \ldots, b_7)$ and generate eight 50-bit vectors $(c_0, \ldots, c_7)$. Five guard bits are allocated to each vector to prevent any overflow.

Pipelined Stage 3. These stage inputs are eight 50-bit vectors $(c_0, \ldots, c_7)$. Four SWP addition/subtraction units are used to perform arithmetic operations on subword data. The output at this stage are nine 55-bit vectors $(d_0, \ldots, d_8)$. Five guard bits are allocated to each vector to prevent any overflow.

Pipelined Stage 4. These pipeline stage inputs are nine 55-bit vectors $(d_0, \ldots, d_8)$. Five multiplier units are used to perform multiplications of subwords with constants. This stage output consists of nine 60-bit vectors $(e_0, \ldots, e_8)$. Due to the small values of the constants (m_i), 60 bits are sufficient in practice for moderate accuracy requirements (see next subsection).

Pipelined Stage 5. These stage inputs are nine 60-bit vectors $(e_0, \ldots, e_8)$. For 8-bit selected subword size, each 60-bit vector contains five 12-bit subwords. Six SWP addition/subtraction units are used to perform arithmetic operations on subword data. The output at this stage is eight 60-bit vectors $(f_0, \ldots, f_7)$.

Pipelined Stage 6. This is the last stage of the *pretranspose 8-point SWP DCT* operator. The input to this stage is eight 60-bit vectors $(f_0, \ldots, f_7)$. Four SWP addition/subtraction units are used to perform arithmetic operations on subword data. This stage generates final output which consists of eight 60-bit vectors $(S_0, \ldots, S_7)$. Each output vector contains multiple DCT values.

By using the SWP DCT operator, the DCT computations can be performed much faster compared to a simple DCT operator. This increase in speed occurs because of the parallel processing of pixels using multimedia oriented SWP arithmetic units.

Data Bit-Width. For the proposed SWP DCT operator, data width is chosen with regard to the more constrained subword size, that is, 8-bit subwords, with 100% resource utilization

ratio and that implements the largest parallelism degree (5 subwords are handled in parallel). For 10-, 12-, and 16-bit subwords, data widths larger than required will thus be available. For the first 1D-DCT (pretranspose *SWP 8-point DCT* unit), in the first three stages, only additions or subtractions are performed, so one extra bit per subword and per pipeline stage is required to avoid any overflow. Internal word sizes are then 45, 50, and 55 bits for pipeline stages 1, 2, and 3, respectively (see Figure 7). In pipeline stage 4, multiplications with constant values less than 1.3 (m_4) are performed. We assume a moderate accuracy is needed for the DCT so that the fraction part at the output of the multiplier can be discarded to reduce data-width increase (e.g., JPEG). One extra bit per subword only is thus required for the output of pipeline stage 4 so that internal word size is then 60 bits (for applications which have greater accuracy requirements, data-width can be increased accordingly). The two next pipeline stages perform additions and subtractions. However, thanks to the 12-bit internal subword size at the output of stage 4, data-width increase is not required in practice.

4.4. Arrangement of 1D-DCT Values. The 1D-DCT values obtained from the *pretranspose 8-point SWP DCT* unit are in the form of packed subwords. To perform further operations these values need to be arranged in 8×8 matrix format. In each clock cycle, the incoming 1D-DCT values are placed in the proper location for subsequent transpose operation.

4.5. Transpose Unit. In the *transpose* unit, 8×8 matrix which contains row-wise 1D-DCT values is transposed. In this operation rows are converted into columns. The transpose operation is done so that the column-wise 1D-DCT computations can also be performed using row-wise operator. Performing two row-wise operations plus the transpose operation is much simpler compared to row-wise DCT operation followed by column-wise operation.

4.6. Vector Formation for 2D-DCT Unit. The transposed one-dimensional DCT values are arranged in the form of eight 60-bit vectors which are used as input to the *posttranspose SWP 8-point DCT* unit. This formation of vectors is done in the same manner as explained in Section 4.2. However, instead of 40 bits, the length of each vector is 60 bits in this case. Each 60-bit vector contains p packed DCT values subwords.

4.7. Posttranspose SWP 8-Point DCT Unit. The size of 1D-DCT values obtained from the *pretranspose SWP DCT* operator consists of input pixel size plus the guard bits. For 8-bit pixels, each 1D-DCT value consists of 12 bits. Similarly for 10-, 12-, and 16-bit pixel sizes, the corresponding DCT values consist of 14, 16, and 20 bits, respectively. These 1D-DCT values are packed in 60-bit vectors and are given to the *posttranspose SWP 8-point DCT* unit for the computation of the two-dimensional DCT. This unit is similar to the *pretranspose SWP 8-point DCT* unit (Section 4.3). Data sizing for the *posttranspose SWP 8-point DCT* unit is also similar to the *pretranspose SWP 8-point DCT* unit data sizing: for 8-bit input image pixels, internal subword sizes are, respectively, 13, 14, 15, and 16 bits for the first four pipeline stages. Data-width increase is not required in practice for the next pipelines stages. After performing computations in the different pipeline stages, the output is obtained. The output of the *posttranspose SWP 8-point DCT* unit consists of eight 80-bit vectors ($S_0, \ldots, S_7$). Based upon the selected subword sizes, each of these 80-bit vector contains different number of 2D-DCT values. For 8-bit selected subword size (i.e., 8-bit pixels), each 80-bit output vector contains five 16-bit 2D-DCT values. Similarly for 10-, 12-, and 16-bit subword sizes, each output vector contains four 18-bit, three 20-bit, and two 24-bit 2D-DCT values, respectively. These packed values are final two-dimensional DCT values of an input image block.

4.8. Arrangement of 2D-DCT Values. The 2D-DCT values obtained from the *posttranspose SWP 8-point DCT* unit are arranged in 8×8 block format. These values can be converted into the integer format for any further use. The same computations are performed on each 8×8 block to compute the overall 2D-DCT of an image.

5. State Machine Controller

The state machine controller is used to control the sequence of operations performed by the SWP arithmetic units. It activates the unit at the required times to perform any particular task. The sequence of operations performed by the SWP DCT operator is shown in Figure 8.

The control signals generated by the *state machine controller* are shown in Figure 9.

Vector Formation for 1D-DCT Enable. When this signal is activated it enables the formation of vector by packing p pixels in word size registers.

SWP 1D-DCT Operator Enable. When this signal is activated, the *pretranspose SWP 8-point DCT* unit performs parallel DCT computations on packed pixels.

Arrangement of 1D-DCT Values Enable. When this signal is activated, the 1D-DCT values computed by the *pretranspose SWP 8-point DCT* operator are arranged in 8×8 matrix format. This matrix contains 1D-DCT values.

Transpose Enable. The activation of this control signal starts the process of transposing the 1D-DCT values matrix. As a result the row values are converted to column values.

Vector Formation for 2D-DCT Enable. When this signal is activated, the 1D-DCT values are packed in input registers for 2D-DCT computations. The size of each value is equal to subword size plus the guard bits to avoid overflow.

SWP 2D-DCT Operator Enable. This control signal enables the *posttranspose SWP 8-point DCT* unit. This unit performs parallel 2D-DCT computations on input values.

Arrangement of 2D-DCT Values Enable. When this signal is activated, the 2D-DCT values computed by the *posttranspose*

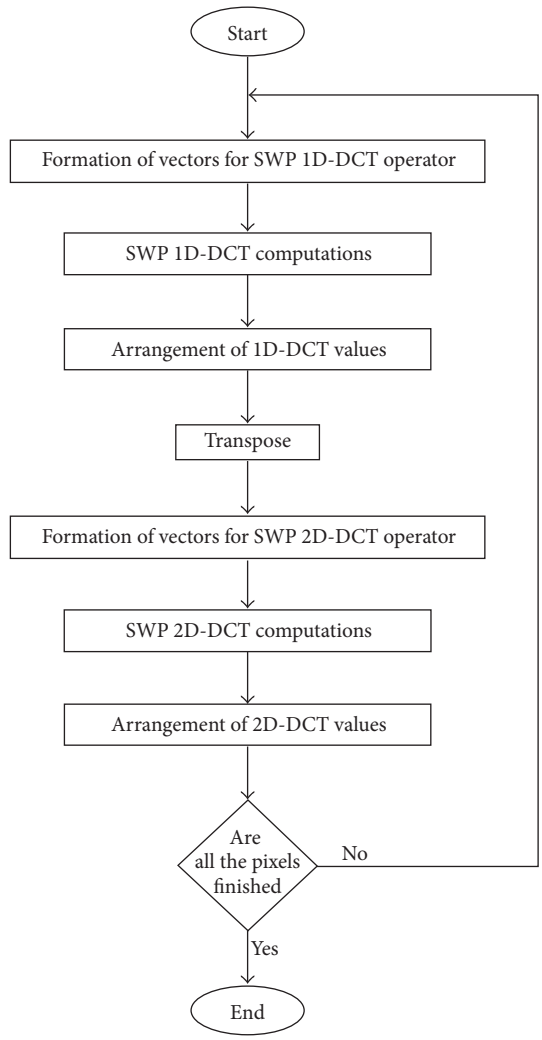

Figure 8: Sequence of operations.

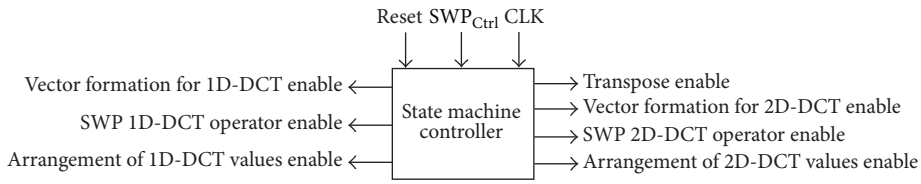

Figure 9: Block diagram of the controller.

SWP DCT operator are arranged in 8×8 matrix format. As a result, this matrix contains required 2D-DCT values.

6. Synthesis Results

To analyze the area, speed, and power, the proposed SWP DCT operator has been synthesized to the ASIC standard cell HCMOS9GP 130 nm (CORE9GPLL 4.1 low leakage standard cell library from ST Microelectronics) and 90 nm (fsd0t-a standard performance low voltage threshold cell library from UMC) technology using Synopsys Design Vision and to Xilinx Virtex II FPGA platform using Mentor Graphics Precision RTL tool. Table 3 shows the synthesis results.

Due to the large number of computations they implement, the units which consumes maximum resources are the *pretranspose SWP 8-point DCT* and the *posttranspose SWP 8-point DCT* units. These units almost consume 85% to 90% of area on different target technologies. The rest of the units like *formation of vectors*, *transpose*, *arrangement of DCT values, and so forth* consumes almost 10% to 15% of total area. Due to the available parallelism of the *SWP 2D-DCT* operator, computations are performed at high speed. The actual speed-up depends upon the selected subword sizes

Table 3: Synthesis results of SWP 2D-DCT.

90 nm CMOS ASIC			130 nm CMOS ASIC			FPGA VirtexII	
Nand gates	CP (ns)	Power (mW)	Nand gates	CP (ns)	Power (mW)	CLB	CP (ns)
Pretranspose SWP 8-point DCT							
195650	9.8	7.6	160689	21.8	5.7	10790	19.8
Posttranspose SWP 8-point DCT							
293476	11.1	11.4	241034	25.7	8.5	16185	24.9
Other units							
54347	5.7	2.1	60028	10.4	2.2	4760	12.3
Complete SWP DCT operator							
543474	11.1	21.2	461752	25.7	16.4	31735	24.9

Table 4: Performance of the SWP 2D-DCT operator.

Block size	Pixel (bits)	Computational CLKs	SWP overheads CLKs	Total CLKs
(8 × 8)	8	32	7	39
	10	32	7	39
	12	48	9	57
	16	64	11	75
(16 × 16)	8	119	19	138
	10	119	19	138
	12	180	27	207
	16	244	35	279

(8, 10, 12, or 16 bits). Table 4 shows the number of clock cycles required to compute the 2D-DCT on different block sizes while considering multimedia oriented subword/pixel sizes. To perform the 2D-DCT operation on (8 × 8) image blocks, our proposed SWP DCT operator requires 39, 39, 57, and 75 clock cycles for subword/pixel sizes of 8, 10, 12, and 16 bits, respectively. These cycles include both computational clock cycles and SWP overhead clock cycles like packing/unpacking and so forth. Compared to [1, 2], the proposed SWP operator almost requires one-fourth of the number of clock cycles for 8-bit subword size. This reduction in the number of clock cycles occurs because the SWP operator performs parallel computations on packed subwords (five 8-bit pixels are processed in parallel).

To compute two-dimensional DCT of (16 × 16) image blocks, the *SWP 2D-DCT* operator requires 138, 138, 207, and 279 clock cycles while considering pixel sizes of 8, 10, 12, and 16 bits, respectively. Compared to a simple DCT operator without SWP capability, the SWP DCT operator performs the computations in 73% less time when the pixel size is 8 bits. Similarly the SWP DCT operator requires 68%, 60%, and 43% less time when subword sizes are 10, 12, and 16 bits, respectively.

Compared to an architecture using five 8-bit simple DCT operators in parallel, that is, processing five pixels in parallel to assess the same parallelism degree as the SWP operator when pixel size is 8 bits, the SWP DCT operator requires 14% more area. This small increase in area occurs because the SWP DCT operator can perform operations not only on 8-bit pixel size but also on 10-, 12-, or 16-bit pixel sizes and also because of SWP overheads. On the other hand a simple DCT operator is designed for one particular pixel size only which is 8 bits in this case. Thus, even in the presence of these overheads, the SWP DCT operator provides a good area/delay tradeoff.

7. Conclusions

This paper presents the implementation of the 2D-DCT algorithm using a SWP operator as coprocessor to increase performance. The performance of the original simple DCT operator [1] has been enhanced using the SWP technique. The proposed SWP DCT operator operates on multimedia oriented pixel sizes using different selections of subword sizes. The SWP DCT operator utilizes the data-level parallelism and increases the performance through parallel processing of pixels. Coordination between pixel sizes and subword sizes provides high efficiency through a high resource utilization rate. Compared to a simple DCT operator, the SWP DCT operator provides efficiency as well as flexibility by operating on different pixel sizes of multimedia nature. In the future, the same technique can be used to enhance the performance of other multimedia oriented operators so that the overall performance of multimedia processors can be increased.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work was supported by the French *Architectures du Futur* ANR program ANR-06-ARFU-004.

References

[1] M. Kovac and N. Ranganathan, "JAGUAR: a fully pipelined VLSI architecture for JPEG image compression standard," *Proceedings of the IEEE*, vol. 83, no. 2, pp. 247–258, 1995.

[2] L. V. Agostini, I. S. Silva, and S. Bampi, "Pipelined fast 2D DCT architecture for JPEG image compression," in *Proceedings of the 14th Symposium on Integrated Circuits and Systems Design*, pp. 226–231, Brasilia, Brazil, September 2001.

[3] N. I. Cho and S. U. Lee, "DCT algorithms for VLSI parallel implementations," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 1, pp. 121–127, 1990.

[4] M. Bousselmi, M. S. Bouhlel, N. Masmoudi, and L. Kamoun, "New parallel architecture of the DCT and its inverse for image compression," in *Proceedings of the 7th IEEE International Conference on Electronics, Circuits and Systems (ICECS '00)*, vol. 1, pp. 345–348, Jounieh, Lebanon, December 2000.

[5] A. B. Attitalah, P. Kadionik, F. Ghozzi, P. Nouel, N. Masmoudi, and P. Marchegay, "Implementation of Loeffler algorithm on Stratix DSP compared to classical FPGA solutions," in *Proceedings of the International Symposium on Communications, Control and Signal Processing (SCCSP '06)*, pp. 1–4, Marrakech, Morocco, March 2006.

[6] L. Tao and H. K. Kwan, "Multirate-based fast parallel algorithms for DCT-kernel-based real-valued discrete Gabor transform," *Signal Processing*, vol. 92, no. 3, pp. 679–684, 2012.

[7] K. Z. Bukhari, G. K. Kuzmanov, and S. Vassiliadis, "DCT and IDCT implementations on different FPGA technologies," in *Proceedings of the 13th Annual Workshop on Circuits, Systems and Signal Processing*, pp. 232–235, Computer Engineering Lab, Delft University of Technology, Veldhoven, The Netherlands, November 2002, http://ce-publications.et.tudelft.nl.

[8] D. Menard, E. Casseau, S. Khan et al., "Reconfigurable operator based multimedia embedded processor," in *Reconfigurable Computing: Architectures, Tools and Applications: 5th International Workshop, ARC 2009, Karlsruhe, Germany, March 16–18, 2009. Proceedings*, vol. 5453 of *Lecture Notes in Computer Science*, pp. 39–49, Springer, Berlin, Germany, 2009.

[9] S. Khan, E. Casseau, and D. Menard, "Reconfigurable SWP operator for multimedia processing," in *Proceedings of the 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '09)*, pp. 199–202, IEEE, Boston, Mass, USA, July 2009.

[10] Z. Li, S. Peng, H. Ma, and Q. Wang, "A reconfigurable DCT architecture for multimedia applications," in *Proceedings of the Congress on Image and Signal Processing*, vol. 1, pp. 360–364, Hainan, China, May 2008.

[11] A. Sanyal and S. K. Samaddar, "The performance analysis of fast DCT algorithms on parallel cluster architecture," *International Journal of Information and Electronics Engineering*, vol. 2, no. 3, pp. 369–373, 2012.

[12] M. Kovac and N. Ranganathan, "VLSI circuit structure for implementing JPEG image compression standard," *US patent US5659362*, 1997.

[13] A. A. Farooqui, V. G. Oklobdzija, and F. Chechrazi, "Multiplexer based adder for media signal processing," in *Proceedings of the IEEE International Symposium on VLSI Technology, Systems, and Applications*, pp. 100–103, Taipei, Taiwan, June 1999.

[14] S. Krithivasan and M. J. Schulte, "Multiplier architectures for media processing," in *Proceedings of the of the 37th Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 2193–2197, Pacific Grove, Calif, USA, November 2003.

[15] A. Danysh and D. Tan, "Architecture and implementation of a vector/SIMD multiply-accumulate unit," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 284–293, 2005.

[16] J. Fridman, "Sub-word parallelism in digital signal processing," *IEEE Signal Processing Magazine*, vol. 17, no. 2, pp. 27–35, 2000.

[17] A. Wang and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 1, pp. 310–319, 2005.

[18] M. O. Cheema and O. Hammami, "Customized SIMD unit synthesis for system on programmable chip—a foundation for HW/SW partitioning with vectorization," in *Proceedings of the IEEE Design Automation Conference*, pp. 54–60, San Francisco, Calif, USA, January 2006.

[19] S. Khan, E. Casseau, and D. Menard, "High speed reconfigurable SWP operator for multimedia processing using redundant data representation," *International Journal of Information Science and Computer Engineering*, vol. 1, no. 1, pp. 45–52, 2010.