1. Introduction

Nowadays, the H.264/AVC standard is regarded as the de-facto standard for modern video applications, due to its high coding efficiency and an increased flexibility to efficiently support different classes of applications, mainly through the use of multiple combinations of coding profiles and levels [1]. These significantly high coding efficiency levels, i.e., a reduction of at least 50% in the bit-rate when compared with MPEG-2 and up to 30% better compression levels when compared to H.263+ and MPEG-4 [2], are mostly owed to an extensive set of novel coding tools that were, for the first time, included in the classical block-based motion-compensated transform scheme also used on H.264/AVC, such as inter-prediction using multiple reference frames and variable block sizes, motion compensation with quarter-pixel precision, several intra-prediction modes, or improved transform and entropy coding techniques.

Nonetheless, just like the previous ISO/IEC MPEG-x and ITU-T H.26x video standards, transform coding is still one of the key components of H.264/AVC, since it provides one of the fundamental mechanisms for compressing the video data by reducing the spatial redundancies. However, the residual transform coding block of a H.264/AVC codec is much more complex than in previous standards in order to guarantee the desired coding performance levels. In fact, such transform unit, that must be present in both the encoder’s forward and reconstructed processing loops, as well as in the decoder processing loop (see Figure 1), must support several different forward and inverse transform functions and also residual blocks with smaller sizes (4 × 4 pixels) [3]. This poses several challenges for real-time H.264/AVC applications, and even greater ones for video systems implementing the newest SVC extension of H.264/AVC [4], due to the involved computational complexity requirements. Consequently, in the last few years several different specialized and dedicated architectures for transform coding in H.264/AVC have been proposed in the literature.

Almost all of such proposals [5, 6, 7, 8, 9, 10, 11, 12, 13] focus on efficient VLSI architectures that implement fast and optimized algorithms for transform coding, so as to mitigate the involved computational complexity. Among these, several designs adopt a row-column decomposition approach to realize a single H.264/AVC transform and use either a single one-dimensional (1-D) transform kernel with a transposition memory or two 1-D transform kernels with a transposition switch or memory [5, 6, 7]. Nevertheless, other proposals have also been presented that use fast direct two-dimensional (2-D) designs [8, 9] to offer higher performance transform kernels, almost always for a single H.264/AVC transform, but at the expense of higher power consumptions and larger chip areas.

In this trend, other hardware realizations based on unified transform computation kernels, capable of implementing all H.264/AVC transforms, have also been recently proposed [10, 11, 12, 13]. Such architectures quite often impose a significant area overhead and thus present higher power consumption requirements, which reduces their attractiveness when low power applications are considered (e.g.: portable, handheld and other battery supplied devices). For such class of applications, a different category of architectures capable of easily and swiftly scaling its hardware requirements, power consumption and perfor-
performance constraints with the target application requirements is often desirable. In this scope, systolic array architectures [14] have proved to be particularly suitable. However, few systolic designs have been proposed to implement transform kernels for video coding, and most of them concern the classical $8 \times 8$ Discrete Cosine Transform (DCT) adopted in previous ISO MPEG-x and ITU-T H.26x standards [15, 16, 17]. Furthermore, a systolic array architecture for unified transform coding in H.264/AVC has never been proposed yet. In accordance, this paper presents an efficient and programmable Processing Unit (PU) capable of implementing the whole set of H.264/AVC transforms, and that can be used as the basic processor element of a generic systolic array architecture for transform coding in H.264/AVC.

The rest of this paper is organized as follows. In Section 2 the forward and inverse transforms adopted in the most common profiles of H.264/AVC are reviewed. Section 3 introduces the proposed framework to develop efficient systolic architectures for transform coding in H.264/AVC, while section 4 presents the proposed programmable PU that is capable of realizing all H.264/AVC transforms. Experimental results for an implementation of the proposed PU in a Xilinx Virtex4 FPGA device are provided in Section 5 and Section 6 concludes the paper.

2. H.264/AVC transforms

In H.264/AVC six different 2-D transforms (i.e., $8 \times 8$ and $4 \times 4$ forward and inverse integer DCT transforms, $4 \times 4$ and $2 \times 2$ Hadamard transforms) can be used to encode each residual block of data, although the $8 \times 8$ transform is only used in the High Profile levels [2]. To avoid inverse transform mismatch problems, all the considered transforms operate in the integer domain. Such important characteristic allows them to be implemented using only addition and shift operations, which significantly reduces the computational complexity requirements.

All these transforms process the pixel data by grouping the pixels in $(N \times N)$ block structures. These blocks are then transformed and mapped into the frequency domain. Hence, by considering a 2-D image block $x$, defined in the spatial (pixel) domain, the corresponding frequency domain representation $X$ is obtained by applying the $(N \times N)$ linear transform process

$$X = C \cdot C^T$$

where the matrix $C$ is usually referred to as the transform kernel or basis function.

The $4 \times 4$ integer DCT transform is applied to all $4 \times 4$ blocks of residual data resulting from both the motion-compensated prediction and the intra-prediction stages (blocks 0 to 15 and 18 to 25 in Figure 2). Likewise the conventional DCT, this integer transform is also orthogonal, separable, and presents a strong decorrelating performance. In practice, it consists of a simplified $4 \times 4$ DCT, where the scaling factor component has been transferred to the quantization stage in the encoding algorithm [18]. Hence, the forward transform contains only four different coefficient values (i.e., 1, -1, 2 and -2), in order to allow its

![Generalized block diagrams of a H.264/AVC encoder and decoder.](image-url)
are depicted in Equations 2 to 5, respectively.

The 2-D transform operations adopted by video coding standards, and in particular the ones described in section 2 for transform coding in H.264/AVC, typically consist of regular algorithms implementing reduced complexity operations, but with high data throughput and computational rates, especially when real-time operation is considered. Hence, efficient implementations of such algorithms usually require parallel and pipelined processing in order to comply with the desired performance goals.

Although current general purpose processors with SIMD extensions might be able to fulfill such requisites, its hardware efficiency and power consumption requirements greatly compromise its usage for the most common video coding applications being executed in portable and battery supplied devices. Conversely, for other classes of applications not imposing such constraints, efficient and optimized parallel implementations of the transform algorithms should still be required, so as to avoid performance degradation due to communication bottlenecks on memory access. Furthermore, the static architectural nature of general purpose processors prevents the adaptation of such processing structures in terms of power consumption and hardware efficiency to the specific characteristics of a given video coding application or operation scenario (i.e., different video resolutions, different QoS requisites, status of the system battery, etc.). In this scope, systolic array processors have proved to provide rather convenient solutions to overcome all of the above mentioned problems [14].

Systolic arrays consist of locally connected Processor Elements (PEs), each one computing a restricted and very well known set of operations, that exchange the data to be processed with each other using a pipelined dataflow. This design approach, which is depicted in Figure 3 for a generic 2-D array, is usually regarded as requiring low control and interconnection circuitry and offering high clock frequencies. Consequently, such structures are able to provide high computational rates and high data throughput with very high hardware usage efficiency.

The structure of a systolic array targeting a given algorithm can be derived using a well known methodology that is described in [14]. Using such methodology it is possible to derive dependence graphs for the considered algorithms, in which the computation nodes represent the basic operations that need to be performed and the dependence arcs define the order by which the operations need to be executed. Hence, hardware implementations of such graphs
will include two different types of units: i) processor elements, that are designed to realize the operations specified by the computational nodes; and ii) communication links, which implement the dependence arcs.

To improve the hardware utilization rate and achieve viable hardware implementations, where the graphs should be defined in index spaces with no more than two dimensions, more elaborate techniques must be used, like projection, algorithm decomposition into subparts or time scheduling. In fact, this is the case for the H.264/AVC transform algorithms, which are defined over a four-dimensional index space owing to the involved operations, as it can be seen in Equation 6.

\[ Y_{ij} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} C_{ik} C_{jl} x_{kl}, \quad i, j = 0, \ldots, N - 1 \]  

Equations 7 and 8 illustrate this decomposition in two subparts for a generic H.264/AVC transform algorithm, where \( X, Y, M \) and \( C \) denote the input data, the output data, intermediate data and the transform coefficients, respectively.

\[ Y_{ij} = \sum_{k=0}^{N-1} C_{ik} M_{jk}, \quad j, k = 0, \ldots, N - 1 \]  

\[ M_{jk} = \sum_{l=0}^{N-1} C_{jl} x_{kl}, \quad i, j = 0, \ldots, N - 1 \]  

Actually, the matrix-product form of the 2-D H.264/AVC transform functions is typically derived using the corresponding representations for Equations 7 and 8, as it can be seen in Equation 9.

\[ [Y]_{ij} = Y = CM^T = C (CX^T)^T = CXC^T \]  

As a result of the previous discussion, it is clear that several different systolic arrays implementing the H.264/AVC transform algorithms can be derived using the methodology presented in [14] and by considering different optimization techniques. As an example, in Figure 4 it is depicted the block diagram of one of such structures (Transform Kernel) that results from decomposing the 2-D transform algorithm in two parts. The first part deals with indices \( l \) and \( j \) in Equation 6 and thus computes a 1-D row-wise transform function (Equation 8), while the second part is spawned across indices \( k \) and \( i \) in Equation 6 and computes a 1-D column-wise transform function using the transposed results obtained from the first part of the algorithm (Equations 7).

This row-column decomposition approach not only reduces the amount of processor elements required to implement the 2-D transform algorithm, but also significantly increases its utilization rate. Nonetheless, such hardware efficiency can still be further improved if the basic processor element of the systolic array is able to compute operations for multiple transforms. In section 4 it is proposed a programmable PU that is capable of realizing the fundamental operations of all the H.264/AVC transform algorithms described in section 2, and that can also be used as the base processor element of any given systolic array structure to obtain a unified transform engine.

As a final remark, it should be mentioned that independent systolic transform engines can only be obtained when all the four functional blocks depicted in Figure 4 are considered. The input and output buffers are required to guarantee a regular dataflow within the systolic array and minimize the delays in accessing the external data memories where the data is located. On the other hand, the control unit is responsible for controlling the systolic array and the operation of the I/O buffers, as well as for implementing the necessary synchronization mechanisms between the transform engine and the video coding system.

4. Proposed Programmable Processing Unit

In order to efficiently support the realization of the H.264/AVC transforms presented in section 2, the proposed PU adopts a hybrid programmable and dedicated architecture with a simple and reduced signal interface, as it can be seen in Figure 5. Moreover, to minimize the delays resulting from control operation and thus to maximize the PU data processing speed, all the control logic required for the correct circuit operation is also contained within the PU. By using such approach it is therefore possible not only to obtain a low delay transform computational unit, owing to the dedicated hardwired computational logic circuits, but also to optimize the low capacitance local connections between the several PUs that may compose the systolic processing array of the transform engine. Consequently, systolic architectures based on the proposed PU are able to provide high throughput rates due to being able to operate with high clock frequencies in a pipeline organization.

As it can be seen in Figure 5, the proposed PU is com-
posed by two main blocks: the arithmetic module and the control module. Transform coefficient values are computed in the arithmetic module using an accumulator and a specialized multiplier that has its multiplicand programmed according to the type of transform being implemented. In this scope, the residue values (or the intermediate transform coefficient values, when the considered transform engine adopts the row-column decomposition approach) are first loaded into an internal standing-data register. Moreover, the partial value of the transform coefficient being computed, which can have been calculated in either the same or in a different PU at a previous clock cycle, is also loaded into another internal standing-data register. Then, depending on the transform to be realized (specified by the TYPE_T signal) and on the coordinates of the transform coefficient to be computed (identified by the COORD_X and COORD_Y signals), the partial value of the transform coefficient is updated with the result of the multiplication of the residue value (or the intermediate transform coefficient value) by a specific multiplier value (i.e., 1, -1, 2, -2, $\frac{1}{2}$, or $-\frac{1}{2}$).

On the other hand, the control signals of the proposed programmable PU allow to command the circuit operation and to clear the accumulated values. The PU operation is controlled by the system global enable signal (EN) and the local enable signal generated by the system control module (CALC). This provides the necessary mechanisms to guarantee the desired dataflow within the systolic processing array of any given transform engine. Likewise, the accumulator is cleared whenever the system global reset signal (RST) is activated, or on demand by the system control unit (CLR). Such flexibility is especially important for transform engines implementing systolic array structures that can be run-time reprogrammed to have either all of its PUs computing a single transform or multiple transforms in parallel. For example, a systolic array with four PUs can be used to compute the coefficients for a $4 \times 4$ transform or for two $2 \times 2$ transforms. To guarantee a full systolic operation, all the control signals are registered with internal registers within the PU control module, before they are made available to subsequent PUs in the systolic array.

5. Implementation and Experimental Results

To validate the functionality and to assess both the hardware cost and the computation performance levels provided by the proposed programmable PU, a parameterizable behavioral description of such processing structure was realized using the IEEE-VHDL language. This description was used to synthesize the proposed PU for a general purpose Virtex4 XC4VLX100 FPGA device using the synthesis tool from Xilinx ISE 10.0i under two different scenarios: area and speed optimized implementations. Table 1 presents the results that were obtained for such implementations.

<table>
<thead>
<tr>
<th>Optimization Goal</th>
<th>Slices</th>
<th>LUTs</th>
<th>Max. F.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>34</td>
<td>62</td>
<td>261 MHz</td>
</tr>
<tr>
<td>Speed</td>
<td>38</td>
<td>69</td>
<td>285 MHz</td>
</tr>
</tbody>
</table>

Table 1. Implementation results of the proposed programmable PU in a Xilinx Virtex4 XC4VLX100 FPGA.
Table 2. Maximum allowed processing rates for two systolic transform engines based on the proposed PU.

<table>
<thead>
<tr>
<th>Kernel Design</th>
<th>Serial</th>
<th>Direct 2-D</th>
</tr>
</thead>
<tbody>
<tr>
<td># PUs</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>Issue Rate [CC]</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>Latency [CC]</td>
<td>128</td>
<td>8</td>
</tr>
<tr>
<td>Throughput [Coefs/CC]</td>
<td>1/16</td>
<td>4</td>
</tr>
<tr>
<td>Maximum frame-rate [fps]</td>
<td>CIF, 4CIF, HD720, HD1080, UHDV</td>
<td>175.7, 43.9, 19.3, 8.6, 0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11245.3, 2811.3, 1237.0, 549.8, 34.4</td>
</tr>
</tbody>
</table>

The results shown in Table 1 demonstrate the reduced hardware cost of the proposed programmable PU, since its implementation only requires less than 50 Virtex slices, despite allowing to realize all the H.264/AVC 4×4 and 2×2 transforms. Moreover, the maximum allowed clock frequency values depicted in Table 1 clearly evidence the high performance level of this adaptable processing structure. In fact, such results allow to conclude that the proposed PU can be used to forward/inverse transform in real-time video sequences with resolutions of 4320×7680 pixels (UHDV). Such conclusion can be easily extrapolated from Table 2, which depicts the maximum processing rates that are possible to attain for two generic unified transform systolic engines: i) a serial implementation of a unified transform kernel; and ii) a direct 2-D implementation of a unified transform kernel. In addition, from Table 2 it is also possible to verify the quite different performance nature of the two considered implementations. While the serial structure uses an array with only one PU to compute a 2-D transform function, thus requiring 16 clock cycles (CCs) to process a single transform coefficient, the direct 2-D structure consists of an array with 4×8 PUs and is able to process four different coefficients on every CC. Consequently, it can be concluded that the proposed PU can be used to develop efficient unified transform kernels for both low power and high performance systems.

Finally, the comparative results in terms of the hardware efficiency and performance metrics between the proposed PU and other existing structures with similar functionality are also very promising. Such conclusions are presented in Table 3 for the subset of the transform architectures with FPGA implementations that were reviewed. For each of the considered architectures, this table shows the corresponding hardware cost in terms of slice-count, the maximum clock rate, the latency in CCs, the data throughput rate in CCs (computed as the number of transformed coefficients processed in one CC), and the Data Throughput per Unit of Area (DTUA), defined as the ratio of the data throughput rate over the hardware cost in terms of units of area. Due to the different functionalities offered by these circuits and the diverse FPGA devices that were considered for their implementation, we consider a Virtex slice as a unit of area and take as figure of merit for comparisons the DTUA. Additionally, the presented data throughput values consider only blocks of 4×4 residue values.

As it can be seen in Table 3, the DTUA of a direct 2-D implementation of a unified transform engine based on the proposed programmable PU is about 0.3 times smaller than that of the most efficient structure ([19]), which only implements the forward 4×4 integer DCT transform. Nonetheless, when compared with other structures for unified transform coding, the DTUA of the proposed PU is in fact over 1.5 times higher than that of [13]. Such results are mainly owed both to the very low hardware cost and the high clock frequency of the proposed PU, which results in very high throughput rates. However, it should be noted that this discussion is only indicative in what concerns the DTUA values that can be obtained by using the proposed PU, since it considers not only different implementation technologies but also different circuits with distinct characteristics to implement multiple transform functions.

6. Conclusion

A programmable PU for the implementation of the forward and inverse 4×4 and 2×2 H.264/AVC transforms was presented. The main advantages of such flexible structure are a very simple and reduced control logic and the fact that it only requires addition and multiplexer blocks to implement all the six transforms. As a consequence, the proposed PU can be highly efficient in terms of performance, hardware cost and power consumption. Moreover, due to its reduced set of interface signals, this programmable processing structure can be used to realize very efficient H.264/AVC unified transform systolic engines based on either 1-D or 2-D transform kernels. Such class of architectures is highly relevant for applications that require a fine scaling and tuning of the hardware requirements, power consumption and performance levels with the constraints of the implemented video coding algorithm. Experimental results concerning the implementation of the proposed programmable PU in a Xilinx Virtex-4 FPGA device demonstrated its reduced hardware cost and high performance. In addition, they also revealed that unified transform systolic engines based on the proposed PU can realize all six H.264/AVC transforms in real-time for video sequences with resolutions up to UHDV.

Acknowledgments

This work was supported by the Portuguese Foundation for Science and for Technology (INESC-ID multiannual funding) through the PIDDAC Program funds, by the PROTEC Program funds under the research grant SFRH / PROTEC / 50152 / 2009, by the Spanish Government under project DR.SIMON : Dynamic Reconfigurability for Scalability In Multimedia Oriented Networks (TEC2008-06846-C02-02), and by the 7th Framework Program of the HiPEAC European Network of Excellence.
Table 3. Comparison of FPGA implementations.

<table>
<thead>
<tr>
<th>Design</th>
<th>Transform Function</th>
<th>FPGA</th>
<th>Slices ($\times 10^3$)</th>
<th>Max. F. (MHz)</th>
<th>Latency (CC)</th>
<th>Throughput (Coefs/CC)</th>
<th>DTUA (Coefs/CC.Slices $\times 10^3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[19]</td>
<td>Forward 4 x 4</td>
<td>Virtex-2 Pro</td>
<td>0.6</td>
<td>107</td>
<td>1</td>
<td>16</td>
<td>2867.4</td>
</tr>
<tr>
<td>[20]</td>
<td>Inverse 4 x 4</td>
<td>Virtex-2 Pro</td>
<td>4.2</td>
<td>132</td>
<td>64</td>
<td>1</td>
<td>31.4</td>
</tr>
<tr>
<td>[21]</td>
<td>Inverse 4 x 4</td>
<td>Virtex-4</td>
<td>–</td>
<td>108</td>
<td>8</td>
<td>4</td>
<td>–</td>
</tr>
<tr>
<td>[13]</td>
<td>Unified 4 x 4</td>
<td>Virtex-4</td>
<td>2.1</td>
<td>167</td>
<td>8</td>
<td>8</td>
<td>636.2</td>
</tr>
<tr>
<td>[12]</td>
<td>Forward 4 x 4</td>
<td>Virtex-4</td>
<td>1.6</td>
<td>225</td>
<td>8</td>
<td>8</td>
<td>562.5</td>
</tr>
<tr>
<td>This Work</td>
<td>Unified 4 x 4</td>
<td>Virtex-4</td>
<td>1.2</td>
<td>285</td>
<td>1</td>
<td>4</td>
<td>937.5</td>
</tr>
</tbody>
</table>

References


