A High Performance FPGA-Based Architecture for the Future Video Coding Adaptive Multiple Core Transform

Matías J. Garrido, Fernando Pescador, *Senior Member, IEEE*, M. Chavarrías, P. J. Lobo, and César Sanz, *Senior Member, IEEE*

***Abstract*—Future video coding (FVC) will be the next generation video coding standard. Yet in the first stages of the standardization process, it is expected to replace high efficiency video coding beyond 2020. One of the enhancements in discussion is the adaptive multiple core transform, which uses five different types of 2-D discrete sine/cosine transforms (DCT-II, DCT-V, DCT-VIII, DST-I, and DST-VII) and up to 64 × 64 transform unit sizes. This schema, increases the computational complexity of both, encoder and decoder. In this paper, a deeply pipelined high performance architecture to implement the five transforms for 4 × 4, 8 × 8, 16 × 16, and 32 × 32 sizes is proposed. The design has been described in VHDL and it has been synthesized for different Field-programmable gate array chips, being able to process up 182 fps@3840 × 2160 for 4 × 4 transform sizes. Up the best of our knowledge, this is the first implementation of an architecture for the FVC Adaptive Multiple Core Transform supporting 4 × 4, 8 × 8, 16 × 16, and 32 × 32 sizes.**

***Index Terms*—Adaptive multiple core transform, field-programmable gate array (FPGA), future video coding (FVC), hardware architecture, pipeline.**

I. Introduction

VIDEO playing is one of the most demanding applications in actual multimedia terminals, because of the computational complexity of the encoding/decoding process. The new multimedia devices support increasing resolutions, such as high definition (HD) or ultra HD (UHD), with increasing bit rates. But capacity of both, storing devices and communication channels, does not grow at the same speed. Actual video coding standards fill this gap with higher compression ratios at the expense of more computational complexity. As an example, the state of the art high efficiency video coding (HEVC) standard [1] doubles the data compression of its predecessor, H.264 [2] while maintaining the video quality, but its complexity also has been increased several times [3].

In this scenario, the joint video exploration team of ITU-T SG 16 WP3 and ISO/IEC JTC 1/SC19/WG11 has published a call for proposals on video compression with capability beyond HEVC [4]. This is the first milestone in the standardization process that it is expected to finish by 2020 with the publication of a new standard, informally named future video coding (FVC). This new standard is expected to replace HEVC in consumer electronic devices, such as TV sets, mobile phones, tablets, etc. Prior to the CFP, an exploration stage has been carried out [5] with a set of exploration experiments developed by researchers at industry and academia. Also, the promising algorithms have been described in test models [6] and a software version of the codec, implemented on the top of the HEVC reference software [7], is maintained to support the different experiments. Moreover, an optimized version of the decoder is also being developed in a branch of the OpenHEVC decoder [8].

Up to date, both, ITU-T and ISO/IEC standardized codecs have been based on the so called hybrid coding scheme. In this scheme, the prediction error is transformed, quantized, and encoded into a bit stream. Transformation is one of the key tools of any video codec. HEVC standard uses the 2-D discrete cosine transform of type II (DCT-II) [9], approximating the DCT coefficients by integer numbers and computing the 2-D transform with two 1-D transforms in the horizontal and vertical directions [10]. In this case, the size of the 2-D transforms may be 4×4, 8×8, 16×16, or 32×32. DCT-II is the HEVC core transform, although an alternative 4×4 size integer transform, derived from the discrete sine transform, is also applied in some cases [10].

In FVC, several proposals are being evaluated to improve the compression regarding the transform tools. One of them is called adaptive multiple transform (AMT). With AMT, five different 2-D transforms, based on DCT-II, DCT-V, DCT-VIII, DST-I, and DST-VII [9], may be used during the encoding/decoding process. Depending of several encoding parameters, one of the types may be selected for the 1-D horizontal transform while other type (or the same) may be selected for the vertical transform of the same block [6]. The same HEVC transform sizes are used in FVC. Also, a 64×64 size has been proposed for DCT-II only.

Regarding the implementation technologies, in the last years, new advanced field-programmable gate array (FPGA) chips are available allowing the implementation of system on chip designs. These new FPGAs are adequate for low-end [10], [11], middle-end [13], [15], or high-end [14], [15]

Manuscript received January 2, 2018; revised February 12, 2018; accepted February 15, 2018. Date of publication March 7, 2018; date of current version March 29, 2018. This work was supported by the Spanish Ministerio de Economía y Competitividad under Grant TEC2016-75981-C2-2-R. *(Corresponding author: Matías J. Garrido.)*

The authors are with CITSEM, Universidad Politécnica de Madrid, 28031 Madrid, Spain (e-mail: matias.garrido@upm.es; fernando.pescador@upm.es; miguel.chavarrias@upm.es; pedro.lobo@upm.es; cesar.sanz@upm.es).

Digital Object Identifier 10.1109/TCE.2018.2812459

applications, and perform quite well for video processing. In this paper, a fully pipelined high performance FPGA-based architecture for the FVC AMT is presented. The architecture has been designed to implement 4 × 4, 8 × 8, 16 × 16, or 32 × 32 size 1-D horizontal or vertical transforms in 4, 32, 256, and 2048 clocks cycles, respectively. Taking into account the clock frequencies that can be obtained in up to date FPGA devices, the proposed architecture supports both, HD and UHD resolutions while using a small percentage of the FPGA resources. Up the best of our knowledge, this is the first implementation of an architecture for FVC AMT supporting 4 × 4, 8 × 8, 16 × 16, and 32 × 32 sizes.

From this point forward, Section II summarizes the previous work in this subject. In Section III, the proposed architecture is shown in deep. In Section IV, the implementation details are explained and the results are compared with other related implementations. Finally, Section V concludes this paper.

II. Background

A. Adaptive Multiple Transform

In FVC, the AMT scheme is used for coding both, intra and inter coded blocks [6]. Five transform types are used: 1) DCT-II; 2) DCT-V; 3) DCT-VIII; 4) DST-I; and 5) DST-VII. The basic functions for the 1-D transforms of size N are shown in Table I. For each $N \times N$ picture block a set of N horizontal N-1D transforms are carried out, followed by a set of vertical N-1D transforms. The horizontal and vertical type of transforms are selected by the encoder at a block basis. The blocks can be 4 × 4, 8 × 8, 16 × 16, 32 × 32, or 64 × 64 size. In the reference software [7], the latter size is only used for DCT-II type transforms.

For all transform types, the 1-D direct transform may be computed (using matrix notation) as

$$Y = T \cdot X^T. \tag{1}$$

In the expression above, X is a $1 \times N$ matrix with a row of the input picture block, T is the $N \times N$ transform coefficients matrix and Y is an $N \times 1$ column matrix with the result of the 1-D transform. The transform coefficients are obtained from the basic functions by an integer approximation. The coefficients, one set for each transform type and size, may be obtained from the reference software and will be part of the FVC standard.

B. 2-D Direct Trasnsform

For each $N \times N$ block, the 2-D direct transform computation can be carried out by first applying N 1-D horizontal transforms to its N rows

$$Y_{\text{int}} = T \cdot X^T \tag{2}$$

followed by N 1-D vertical transforms to its N columns

$$Y = T \cdot Y_{\text{int}}^T = T \cdot \left(T \cdot X^T\right)^T = T \cdot X \cdot T^T. \tag{3}$$

In this case, X and Y are $N \times N$ matrices with the input and output blocks, respectively, and T is the same coefficient matrix used in (1). Y_{int} is an intermediate $N \times N$ matrix to hold the result of the first N 1-D horizontal transforms.

TABLE I
Basis Functions for the DCT and DST Types Used in FVC for N Size 1-D Transforms

Symbol	Quantity
DCT-II	$T_i(j) = \omega_0 \cdot \sqrt{\frac{2}{N}} \cdot \cos\left(\frac{\pi \cdot i \cdot (2j+1)}{2N}\right)$ where $\omega_0 = \begin{cases} \sqrt{\frac{2}{N}} & i = 0 \\ 1 & i \neq 0 \end{cases}$
DCT-V	$T_i(j) = \omega_0 \cdot \omega_1 \cdot \sqrt{\frac{2}{2N-1}} \cdot \cos\left(\frac{2\pi \cdot i \cdot j}{2N-1}\right)$, where $\omega_0 = \begin{cases} \sqrt{\frac{2}{N}} & i = 0 \\ 1 & i \neq 0 \end{cases}$, $\omega_1 = \begin{cases} \sqrt{\frac{2}{N}} & j = 0 \\ 1 & j \neq 0 \end{cases}$
DCT-VIII	$T_i(j) = \sqrt{\frac{4}{2N+1}} \cdot \cos\left(\frac{\pi \cdot (2i+1) \cdot (2j+1)}{4N+2}\right)$
DST-I	$T_i(j) = \sqrt{\frac{2}{N+1}} \cdot \sin\left(\frac{\pi \cdot (i+1) \cdot (j+1)}{N+1}\right)$
DST-VII	$T_i(j) = \sqrt{\frac{4}{2N+1}} \cdot \sin\left(\frac{\pi \cdot (2i+1) \cdot (j+1)}{2N+1}\right)$

Source: Test Model [6].

C. 2-D Inverse Transform

The inverse transform can be computed in a similar way

$$Y_{\text{int}} = T^T \cdot X^T \tag{4}$$
$$Y = T^T \cdot Y_{\text{int}}^T = T^T \cdot \left(T^T \cdot X^T\right)^T = T^T \cdot X \cdot T. \tag{5}$$

It is worth noting that the operations involved in both, direct and inverse transforms are the same, being the only difference the order in which the transform coefficients are used to perform the multiplication operations.

D. Hardware Implementations

Since the H-261 video coding standard was published by ITU in 1988, all ITU-T and ISO/IEC, video coding standards have been based on the aforementioned hybrid coding scheme. As a consequence, a large amount of DCT hardware implementations have been proposed in the literature. Most of them are based on fast algorithms to implement the DCT-II [16], [18], as this has been the heart of most standards [9]. Fast algorithms take advantage of reordering operations, reusing operators, and constant multiplication to shift-add conversion to reduce both, computation time and hardware resources. Optimized algorithms may also be used to implement all AMT transforms, but this leads to different implementations for each DCT type, as shown in Mert *et al.* [19]. This approach may be good for a video encoder, if all transform types are computed for each block in order to choose the best option, but it is quite inefficient in a decoder, because only one transform must be computed at a time.

TABLE II
1-D AMCT PROCESSOR INTERFACE

SIGNAL	I/O	#BITS	DESCRIPTION
clk	I	1	System clock
rst_n	I	1	System reset, active low
start	I	1	Positive pulse to start an operation
t_type	I	3	Transform type: 0, DCT-II; 1, DCT-V; 2, DCT-VIII, 3: DST-I; 4, DST-VII.
t_size	I	2	Transform size: 0: 4×4; 1: 8×8; 2: 16×16; 3: 32×32.
din	I	64	Input data (columns, 4 16-bit inputs)
dout	O	64	Output data (rows, 4 16-bit outputs)
val_out	O	1	Qualifies outputs, active high

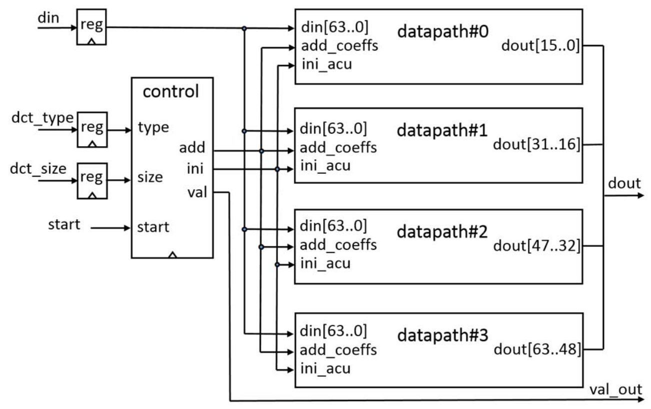

Fig. 1. Architecture of the 1D-AMCT processor.

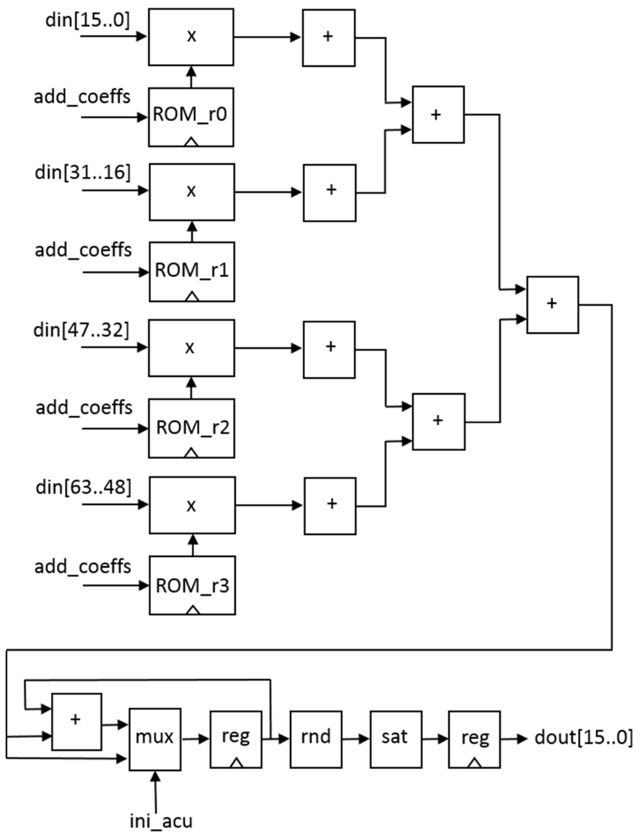

Fig. 2. Architecture of the datapath to compute an element of the 1-D transform output matrix.

III. PROPOSED ARCHITECTURE

As it has been stated before, both, 2-D direct and inverse transforms can be carried out with a combination of 1-D transforms. Also, the 1-D transforms, being direct or inverse, can be carried out with a matrix product. In Section III-A–III-D, 1D-AMT, a fully pipelined architecture to implement the 1D AMT inverse transforms, is proposed. Section III-E explains how this architecture may be used to implement the full 2-D transforms. Finally, Section III-E accounts for the implementation of the direct transforms by introducing only small changes in the design.

A. Inverse Transform Computation

For convenience, in our implementation, the same approach used in [7] and [8] has been employed

$$Y_{\text{int}} = X^T \cdot T \tag{6}$$
$$Y = Y_{\text{int}}^T \cdot T = (X^T \cdot T)^T \cdot T = T^T \cdot X \cdot T \tag{7}$$

where X, Y, Y_{int}, and T are defined in the same way as they were in Section II-B.

B. Logic Model

The 1D-AMT processor interface has been summarized in Table II. A positive pulse in *start* launches a new transform, with size and type defined in the *t_size* and *t_type* inputs. The input data is entered at a column basis, starting in the clock cycle following *start*, through *din* input. Four 16-bit inputs must be inputted at a time. After the processor latency, outputs can be read in *dout* at a row basis, four 16-bit outputs at a time. Finally, the output *val_out* qualifies the valid output data.

Fig. 1 shows the top of the 1D-AMT processor. It is composed of four processing datapaths, each computing one 16-bit output. All datapaths work with the same input data, i.e., the four 16-bits inputs. A control block generates control signals for the internal datapath circuits as well as the *val_out* signal.

The four datapaths have the same internal architecture, shown in Fig. 2. Four synchronous ROMs (ROM_rX) store the transform coefficients. Four 16-bit multipliers and a 3-level add chain compute an output as the sum of products of the four inputs times the four coefficients. Every result is stored in an accumulator, rounded (rnd), saturated (sat) to 16 bits and stored in an output register.

C. Operation

The simplest sequence of operation is carried out when transform size is 4×4. This case is shown in Fig. 3. In the next clock after start, data of the first input column, $X_{00} \cdots X_{30}$ is introduced in *din* input. The processor computes the operation summarized in Fig. 3(a), and, after a latency, asserts *val_out* and outputs the first row of the output matrix in *dout*.

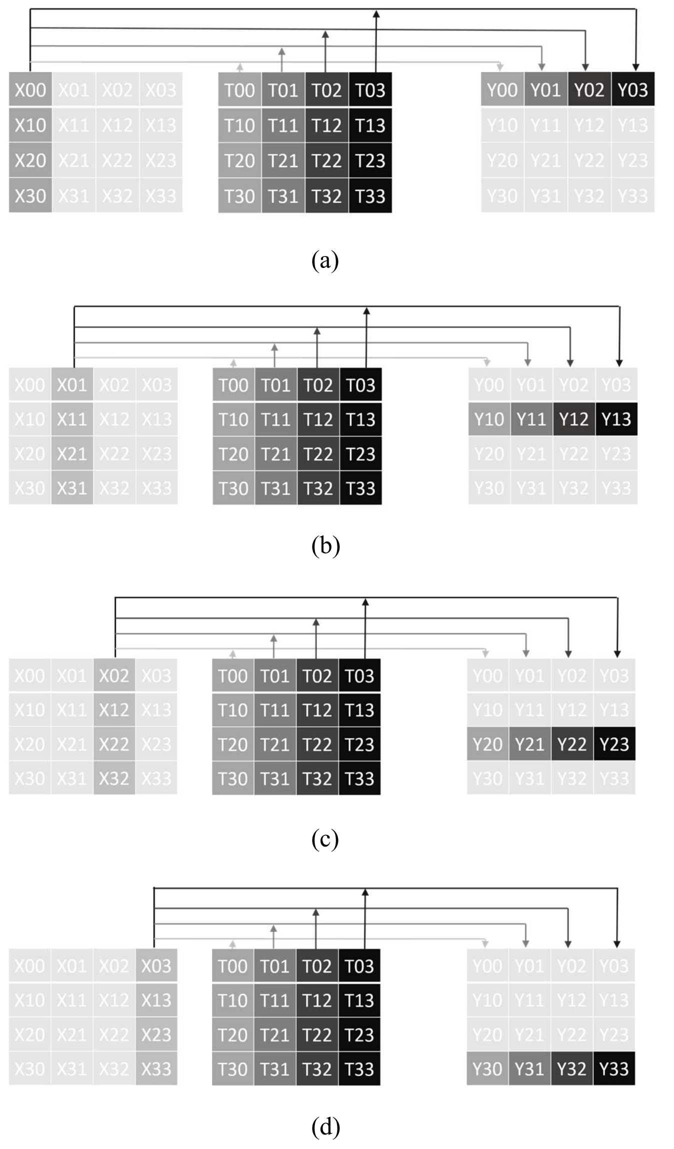

Fig. 3. Computing of the 1-D transforms for a 4 × 4 input block. Four 1-D transforms are needed to transform a 4 × 4 input block. (a) Output line 0 is computed as: $Y_{0j} = \sum_{i=0}^{3} X_{i0} \times T_{ij}, j = 0 \cdots 3$. (b) Output line 1 is computed as: $Y_{1j} = \sum_{i=0}^{3} X_{i1} \times T_{ij}, j = 0 \cdots 3$. (c) Output line 2 is computed as: $Y_{2j} = \sum_{i=0}^{3} X_{ij} \times T_{i2}, j = 0 \cdots 3$. (d) Output line 3 is computed as: $Y_{3j} = \sum_{i=0}^{3} X_{ij} \times T_{i3}, j = 0 \cdots 3$.

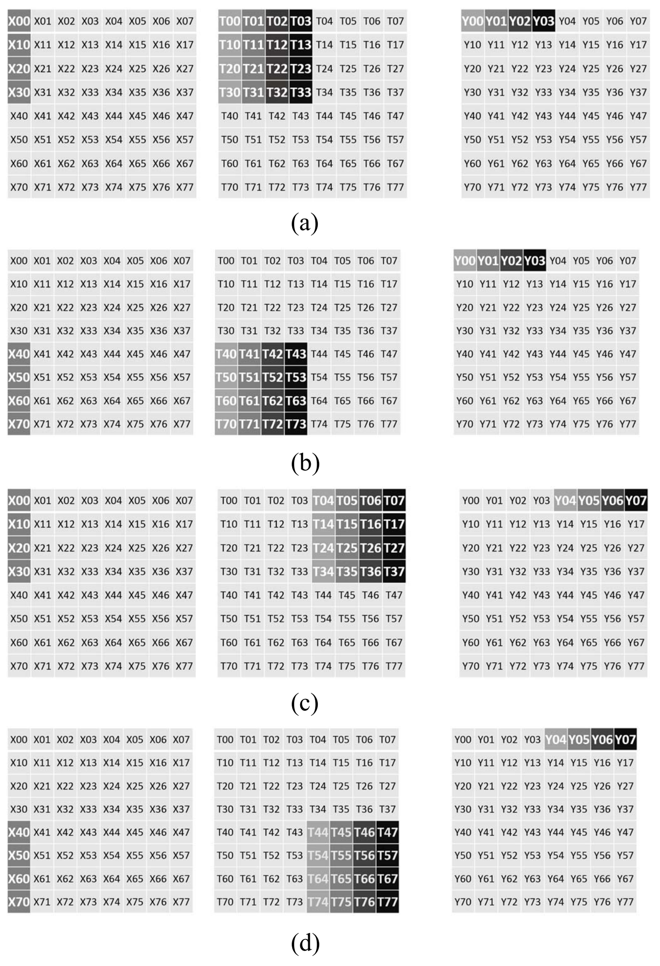

Fig. 4. Computing of the 1-D transforms for an 8 × 8 input block is carried out with 4-size 1-D transforms. Four 4-size 1D-transforms are needed to compute a line. The transformation of the 8 × 8 input block requires 32 4-size transforms. (a) First four outputs are partially computed. (b) Computation of the first four outputs is completed. (c) Other four outputs of the same line are partially computed. (d) Computation of the full line is completed.

In the next three clock cycles, the other three rows of the output matrix are computed, as it is outlined in Fig. 3(b)–(d). This way, and after an initial latency, the processor is able to compute an entire transform every 4 clock cycles. Each datapath is responsible for computing an output (e.g., datapath 0 computes Y_{00}). Inside every datapath, each ROM contains one transform coefficient per type (e.g., ROM_r0 of datapath #r contains 5 T_{00} coefficients, one for each transform type). For the rest of the transform sizes, the output matrices are computed by iterating on the basic procedure explained before.

When transform size is set to 8 × 8, the sequence of operation is outlined in Fig. 4. In this case, every internal ROM stores four coefficients per transform type (e.g., ROM_r0 of datapath #r contains 5 T_{00}, T_{40}, T_{04}, and T_{44} coefficients, a set for each transform type). During the sequence, the controller (see Fig. 1) generates the different address patterns, taking into account the transform type and size. After *start* is asserted, the first four inputs of the first input column must be introduced at *din* and, after a latency, the first four outputs of the first row are partially computed as the product of the inputs times the transform coefficients [Fig. 4(a)]. This partial results are stored in the accumulator when *ini_acu* is asserted. In the next clock, *ini_acu* is negated and the computation is completed by accumulating the product of the remaining inputs of the first column times the transform coefficients [Fig. 4(b)]. The computation of the first row is completed in the next two clock cycles in a similar way [see Fig. 4(c) and (d)]. As the output block has eight rows, the processor needs 4 × 8 = 32 clock cycles to complete the entire transform. During the described process, *ini_acu* is asserted every two clocks. It is worth noting that input data of every column must be introduced twice in the processor; this is clearly shown in Fig. 4.

For 16 × 16 transform sizes, each internal ROM stores 16 coefficients per transform type. To compute the first four outputs of the first 16-size row, the basic procedure outlined in

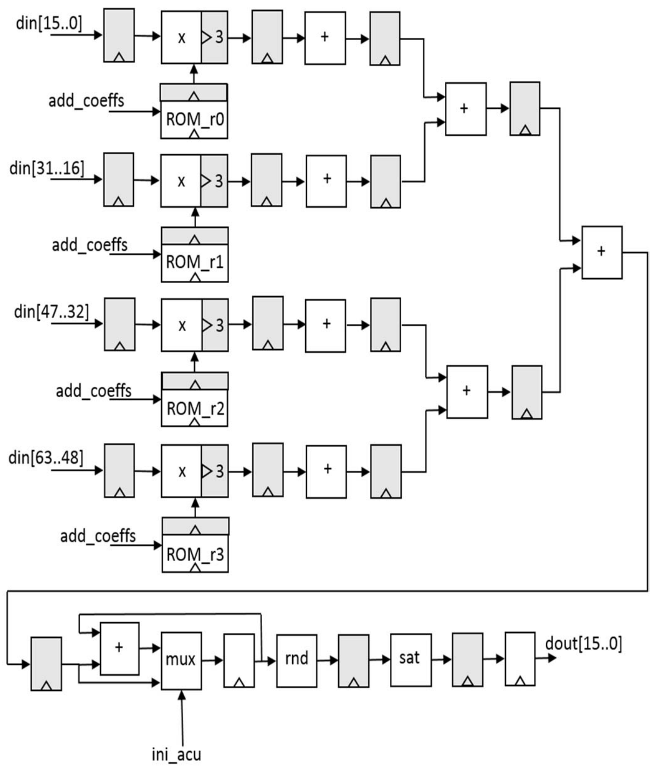

Fig. 5. Architecture of the fully pipelined datapath to compute an element of the 1-D transform output matrix.

Fig. 3 is iterated four times, one for each of the four data sets of the 16-size first column of input data. A total of four clocks are needed to compute this first four outputs. Signal *ini_acu* is asserted in the first cycle, and it is negated in the other three. This way, the computation of a whole output row needs $4\times4 = 16$ clock cycles, while the entire transform is computed in $16 \times 16 = 256$ clock cycles.

Finally, for 32×32 transform sizes, every internal ROM stores 64 coefficients per transform type, signal *ini_acu* is asserted 1 out of 8 times and the entire transform needs $8 \times 8 \times 32 = 2048$ clock cycles to be computed.

D. Pipelining

In order to increase the processor performance the architecture has been fully pipelined. As it can be shown in Fig. 5, pipeline registers have been added in the din input, in the output of the ROMs, inside the multipliers (3 registers), at the output of the multipliers, at the output of all adders belonging to the adder tree and at the output of the round and saturation logic. In Fig. 5, the pipelined registers have been highlighted with a gray shadow. The processor performance will be discussed in the next section.

Regarding the functionality, pipelining introduces extra initial latency cycles. This behavior can be shown in Fig. 6; in the timeline, five 4×4 transforms of types 0, 1, 2, 3, and 4 (see Table II) are launched on intervals of 4 clock cycles. The results of the first transform can be read at *dout* after 13 clocks (initial latency L1 in the figure) along 4 clocks (O1 in the figure). After the second transform is launched both. L1 and L2 latencies *run* in parallel. This means that the first registers of the pipe are dealing with the type 1 transform while the rest are with the type 0 transform. In the worst case, when transform type 4 is launched, the five transforms are in progress, each taking up a part of the processor pipe. This behavior allows the processor to maintain its maximum throughput after an initial latency of 13 clocks. It is worth noting that, for this performance to succeed; the controller must be able to work with up to five different transform requests simultaneously, being of the same or different size or type.

E. 2-D Inverse Transform

Two 1D-AMT processors can be used to implement a 2-D inverse transform. If the results from the first processor (i.e., the horizontal transform) are stored in a transposition memory, then, the second processor can read the transposed values and implement the vertical transform. With a circular buffer both processors can work in parallel and, after an initial latency, the throughput can be the same as the one obtained with the 1-D inverse transform carried out with a single processor.

Actually, as the final rounding process is not exactly the same, the two processors are slightly different. Two variants, 1D-AMT-HT (for the horizontal transform) and 1D-AMT-VT (for the vertical transform) have been implemented and evaluated, with the results explained in the next section.

F. 2-D Direct Transform

The 2-D direct transform can be computed using the same approach explained in Section III-A

$$Y_{\text{int}} = X^T \cdot T^T \tag{8}$$

$$Y = Y_{\text{int}}^T \cdot T^T = \left(X^T \cdot T^T\right)^T \cdot T^T = T \cdot X \cdot T^T. \tag{9}$$

It is clear that the basic 1-D operation stated in (8) is identical to the basic operation for the 1-D inverse transform stated in (6), being the only difference the transposition of the coefficient matrix, *T*. Thus, the same architecture explained in Section II-B could be used for implementing the direct 1-D transform, just modifying the control logic to change the addressing pattern of the coefficient ROMs, and also the rounding block (*rnd* block in Fig. 5).

Also, the 2-D direct transform could be carried out using two processors as explained in Section III-E.

IV. Tests and Results

A. Testbench

The proposed architecture has been tested in simulation with a state of the art simulator engine [20]. The testbench uses an input file with a set of random generated inputs corresponding to blocks of different sizes, from 4×4 to 32×32. Every input has been generated as a random signed 16-bit number. Taking these blocks as input data, transforms of all five types (DCT-II, DCT-V, DCT-VIII, DST-I, and DST-VII) have being launched. Besides, a fork of the open source OpenHEVC decoder [8] has been used to generate the same transforms with the same input data and the results have been stored in a file. Using this file, the testbench can self-check the results of all the tests to be the

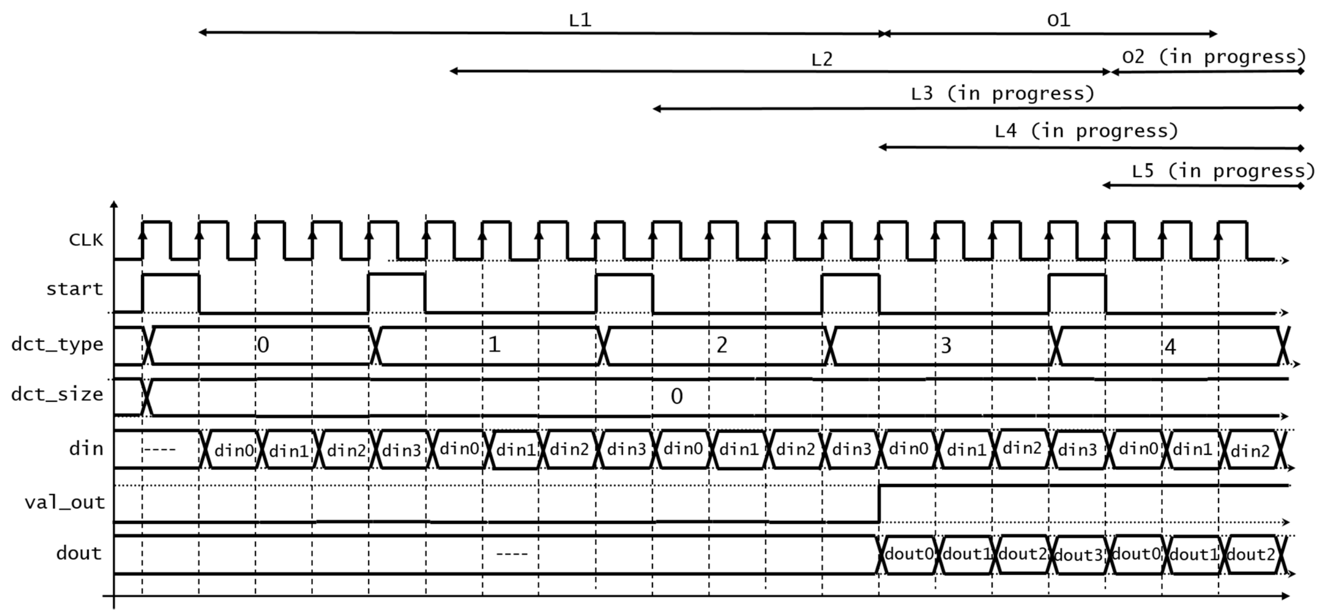

Fig. 6. Timing chronogram for the fully pipelined 1D-AMCT processor. Five 4 × 4 transforms of different types are launched. Each transform is started with a pulse in start and the results are qualified by val_out. When the fifth transform is launched, the other 4 are still in progress.

same as the ones obtained with the aforementioned software implementation.

B. Implementation and Logic Synthesis Results

Several state of the art FPGA architectures have been used as the target technology for the 1D-AMT processor implementation. Representatives of a 28 nm low-end FPGA family (from now on LE_FPGA [11]), a 20 nm mid-end FPGA family (ME_FPGA [13]) and a 14 nm high-end FPGA family (HE_FPGA [14]) have been selected. Finally, a state of the art tool has been used in the logic synthesis process [21].

In all cases, the logic synthesis tool has been configured in a high performance mode that prioritizes speed versus area (i.e., resource consumption). Also, retiming techniques have been used, meaning that the tool may move the pipeline registers through the datapath in order to optimize the timing, without changing the functionality (e.g., the pipeline register at the output of the saturation logic in Fig. 5 will be automatically moved to the left in order to segment this saturation logic). Finally, different seeds have been used in the fitting process and the best results have been chosen for each case [22].

The obtained results have been summarized in Table III for both, 1D-AMT-HT and 1D-AMT-VT variants. Actually, results are very similar, when not identical, for both processors; when there is a difference, results for 1D-AMT-VT design have been shadowed. Clock frequencies range from 300 MHz for low-end devices to 570 MHz for high-end ones. The amount of consumed resources is very similar and, in all cases, represents a small percentage of the total available resources within the FPGAs. As an example, the low-end FPGA-based design spends 1% of the ALMs, 6% of the memory, and 16% of the multipliers available in the target FPGA.

TABLE III
LOGIC SYNTHESIS RESULTS FOR DIFFERENT FPGAS

FPGA TYPE	max. clk freq. (MHz)	#reg	#mul	#ALMs	#RAM blocks
LE_FPGA (28 nm)	304.14	1567	16	585	32 (×10KBITS)
	301.84	1581		572	
ME_FPGA (20 nm)	458.72	1356	16	501	16 (×20KBITS)
	458.72	1364		498	
HE_FPGA (14 nm)	568.83	1352	16	618	16 (×20KBITS)
	573.07	1351		632	

Moreover, Table IV summarizes the number of frames per second that can be processed with the low-end, middle-end, and high-end FPGA-based implementations of the 1D-AMT processor. In all implementations, the smallest clock frequency has been used to calculate the number of frames per second (e.g., 301.84 MHz has been used for the LE_FPGA implementation) so the results can be applied to both, 1-D and 2-D transforms. As can be seen, the performance is fair good for 4×4 and 8×8 transform sized even with the low-end FPGA-based implementation. Good performance can be obtained for 16×16 sizes and the middle-end FPGA-based implementation. For 32×32 sizes, the high-end implementation approaches 24 fps for 4K UHD resolutions.

This numbers have been obtained supposing same size for all transforms. But, in actual applications, each frame is encoded with a mix of transforms of all sizes. As an example, the *mix* column in Table IV includes the number of frames per second processed with the different implementations supposing 25% of transforms of each size. In this case, even the low-end implementation exhibits a reasonable performance for 4K UHD resolution.

TABLE IV
NUMBER OF FRAMES PER SECOND OBTAINED WITH LOW-END, MIDDLE-END, AND HIGH-END IMPLEMENTATIONS OF THE 1D-AMCT PROCESSOR FOR HD (1920 × 1080) AND UHD (3840 × 2160) RESOLUTIONS. 4:2:0 SAMPLING CONSIDERED FOR ALL CASES

implementation	resolution	transform size				Mix
		4×4	8×8	16×16	32×32	
LE_FPGA	1920×1080	388	194	97	48	181
	3840×2160	97	48	24	12	45
ME_FPGA	1920×1080	585	294	146	72	273
	3840×2160	146	73	36	18	68
HE_FPGA	1920×1080	729	364	182	90	340
	3840×2160	182	90	45	22	85

TABLE V
PERFORMANCE OF DIFFERENT FPGA-BASED TRANSFORM IMPLEMENTATIONS

implementation	fps@trans_size				clk	#Kreg	#mul	#ALMs (slices)*	RAM Kbits
	4	8	16	32					
LE,28 nm	97	48	24	12	301	3.1	32	1157	640
ME,20 nm	146	73	36	18	458	2.7	32	999	640
HE,14 nm	182	90	45	22	568	2.7	32	1250	640
[17],28 nm	‡	20	‡	‡	256	1.1	0	(2021)	0
[18],65 nm	†	†	†	34	206	†	32	5200	†
[19],40 nm	93	93	‡	‡	143	4.5	0	(5292)	0

†not available information
‡not applicableº
*values in brackets are slices, otherwise ALMs

C. Comparison

A fair comparison with other FPGA-based architectures is quite difficult. On the one hand, up the best of our knowledge, this is the first proposal implementing the AMT with up to 32 × 32 size. The proposal by Chen *et al.* [18] is focused on HEVC and implements DCT-II only while Mert *et al.* implementation [19] works for the five transforms but only up to 8 × 8 size. On the other hand, different proposals use different FPGA technologies, from 28 to 65 nm, and also use FPGAs from different programmable logic manufacturers.

Table V summarizes the key parameters to compare the performance of the 1D-AMT processor with other FPGA-based implementations. All proposals in Table V implement 2-D transforms using two 1-D transform processors plus a transposition memory, as it has been mentioned in Section III-E. In our case, as the 2-D transform implementation would require two 1D-AMT processors (actually one 1D-AMT-HT and one 1D-AMT-VT), the consumed resources showed in the first three rows in Table V are the sum of the ones included in Table III for the two variants of 1D-AMT, and the clock frequency is always the lowest. The kind of transform implemented in the proposals (direct or inverse) is not relevant for the purpose of the comparison. It is worth noting that the resources used to implement the transposition memory have not been taken into account in Table V, with the exception of Kitsos *et al.* [17] implementation.

Regarding the working clock frequency, our implementations obtain values in the range from 300 to 568 MHz, far away from the other implementations. On the one hand, this is because we are using the most recent technologies available. On the other hand, if we compare the first and fourth rows, in which 28 nm technology is used, we found that our implementation could operate with a 20% higher clock frequency. The high pipelining of the 1D-AMT architecture may be responsible for this improvement.

Regarding the logic resources, only the comparison with Chen *et al.* [18] seems to make sense, because the other proposals only work with up to 8 × 8 sizes. As it is shown in Table V, Chen's proposal consumes more than four times the ALMs used in our proposal. This is because, in this architecture, parallelization of the transform computation is deeper. As a result, the number of frames per second it can process is higher with a lower clock, even when it is compared with the throughput of the high-end 14 nm-based implementation. In any case, as it is shown in Table IV, the 1D-AMT implementation can process real-time 4K UHD formats, considering the fact that not all transforms will be implemented with a 32×32 size, with significantly less resources. Finally, it is worth noting that Chen's proposal only implements DCT-II transform.

V. CONCLUSION

In this paper, 1D-AMT, an FPGA-based architecture for the implementation of the FVC AMT has been proposed. Actually two variants, 1D-AMT-HT and 1D-AMT-VT, which are identical except in the final rounding stage, can be used to implement 2-D AMTs with up to 32 × 32 size. 1D-AMT has been designed to implement the five AMT inverse transforms, but direct transforms could be also implemented with minor changes. Up the best of our knowledge, this is the first proposal to implement the FVC AMT with sizes up to 32×32.

The architecture has been implemented on three state of the art FPGAs: 1) a low-end; 2) a middle-end; and 3) a high-end chip. It has been fully pipelined and synthesized for maximum speed. The synthesis results show that the architecture supports real-time 4K UHD formats with low resource consumption.

In the future work, we plan to implement a more parallelized version of the architecture that will support 8K UHD formats while maintaining a reduced resource consumption. Also, we plan to support 64 × 64 transform sizes as defined in the standard proposal.

REFERENCES

[1] *High Efficiency Video Coding*, ITU, Geneva, Switzerland, Recommendation ITU-T H.265, 2013.
[2] *Advanced Video Coding for Generic Audiovisual Services*, ITU, Geneva, Switzerland, Recommendation ITU-T H.264, 2003.
[3] F. Pescador, M. Chavarrias, M. J. Garrido, E. Juarez, and C. Sanz, "Complexity analysis of an HEVC decoder based on a digital signal processor," *IEEE Trans. Consum. Electron.*, vol. 59, no. 2, pp. 391–399, May 2013.
[4] *Joint Call for Proposals on Video Compression With Capability Beyond HEVC*, MPEG document N17195, Joint Video Exploration Team (JVET) of ITU-T VCEG (Q6/16) and ISO/IEC MPEG (JTC 1/SC 29/WG 11), Geneva, Switzerland, Oct. 2017.

[5] *MPEG Future Video Coding Web Page*. Accessed: Mar. 14, 2018. [Online]. Available: https://mpeg.chiariglione.org/standards/exploration/future-video-coding

[6] *Algorithm Description of Joint Exploration Test Model 7(JEM7)*, MPEG document N17055, Joint Video Exploration Team (JVET) of ITU-T VCEG (Q6/16) and ISO/IEC MPEG (JTC 1/SC 29/WG 11), Geneva, Switzerland, Jul. 2017.

[7] *JEM Reference Software*. Accessed: Mar. 14, 2018. [Online]. Available: https://jvet.hhi.fraunhofer.de/svn/svn_HMJEMSoftware/

[8] *OpenHEVC Decoder*. Accessed: Mar. 14, 2018. [Online]. Available: https://github.com/OpenHEVC/openHEVC

[9] V. Britanak, P. Yip, and K. R. Rao, *Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations*, Feb. 2016. [Online]. Available: http://dsp-book.narod.ru/BYPRDCT.pdf

[10] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[11] *Cyclone V Device Overview*, Intel, Santa Clara, CA, USA, 2016. [Online]. Available: https://www.altera.com/documentation/sam1403480548153.html

[12] *Zynq-7000 All Programmable SoC Data Sheet: Overview*, Xilinx, San Jose, CA, USA, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[13] *Intel Arria 10 Device Overview*, Intel, Santa Clara, CA, USA, 2017. [Online]. Available: https://www.altera.com/documentation/sam1403480274650.html

[14] *Intel Stratix 10 GX/SX Device Overview*, Intel, Santa Clara, CA, USA, 2017. [Online]. Available: https://www.altera.com/documentation/joc1442261161666.html

[15] *Ultrascale Architecture and Product Data Sheet Data Sheet: Overview*, Xilinx, San Jose, CA, USA, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf

[16] H. El-Banna, A. A. El-Fattah, and W. Fakhr, "An efficient implementation of the 1D DCT using FPGA technology," in *Proc. 15th Int. Conf. Microelectron. (ICM)*, Cairo, Egypt, 2003, pp. 278–281.

[17] P. Kitsos, N. S. Voros, T. Dagiuklas, and A. N. Skodras, "A high speed FPGA implementation of the 2D DCT for ultra high definition video coding," in *Proc. 18th Int. Conf. Digit. Signal Process. (DSP)*, 2013, pp. 1–5.

[18] M. Chen, Y. Zhang, and C. Lu, "Efficient architecture of variable size HEVC 2D-DCT for FPGA platforms," *AEU Int. J. Electron. Commun.*, vol. 73, pp. 1–8, Mar. 2017.

[19] A. C. Mert, E. Kalali, and I. Hamzaoglu, "High performance 2D transform hardware for future video coding," *IEEE Trans. Consum. Electron.*, vol. 63, no. 2, pp. 117–125, May 2017.

[20] Mentor. *ModelSim Functional Verification Tool Web*. Accessed: Mar. 14, 2018. [Online]. Available: https://www.mentor.com/products/fv/modelsim/

[21] *Intel FPGA Download Center*. Accessed: Mar. 14, 2018. [Online]. Available: https://www.altera.com/downloads/software.html

[22] R. Scoville. *Fitter Algorithms, Seeds and Variation. The SPICE of Life*. Accessed: Mar. 14, 2018. [Online]. Available: http://www.alterawiki.com/uploads/e/e6/FittingAlgorithms_and_SeedSweeps.pdf

Matías J. Garrido received the Ph.D. degree from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2004.

Since 1987, he has been an Associate Professor with UPM. He has been with Electronic and Microelectronic Design Group at UPM since 1997 and has been with CITSEM since 2013. His current research interests include electronic digital design, video coding, and digital video broadcasting.

Fernando Pescador (M'07–SM'13) received the Ph.D. degree from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2011.

He has been an Associate Professor with UPM since 1995 and has been the Head of the Telematic and Electronic Engineering Department since 2012. He has been with the Electronic and Microelectronic Design Group at UPM since 1999 and has been with CITSEM since 2013. His current research interests include real-time video coding, multiprocessor architectures for video coding, and digital video broadcasting.

Dr. Pescador has been the Editor-in-Chief of the IEEE TRANSACTION ON CONSUMER ELECTRONICS since 2017.

M. Chavarrías received the Ph.D. degree in systems and services engineering for the information society from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 2017.

Since 2016, he has been an Assistant Professor of the Telematic and Electronic Engineering Department, UPM. He has been with Electronic and Microelectronic Design Group at UPM since 2011, integrated in CITSEM since 2012. His current research interests include HEVC, power efficiency techniques for multiprocessor SoCs, multiprocessor architectures for video coding, and reconfigurable video coding.

P. J. Lobo received the B.Sc. and M.Sc. degrees in telecommunication engineering from the Universidad Politécnica de Madrid, Madrid, Spain, in 1993 and 2004, respectively, where he is currently pursuing the Ph.D. degree.

He has been a member of Electronic and Microelectronic Design Group at UPM since 2001, integrated in CITSEM since 2012. His current research interests include multiprocessor architectures and multiprocessor programming methodologies.

César Sanz (M'88–SM'05) received the Ph.D. degree from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 1998.

Since 1984, he has been a Faculty Member of the ETSIS de Telecomunicación, UPM, and since 1999, he has been an Associate Professor with the SEC Department (currently DTE). In addition, he leads the Electronic and Microelectronic Design Group at UPM, involved in Research and Development Projects. Since 2013, he has been a Researcher with CITSEM. His current research interests include microelectronic design applied to image coding, digital TV, and digital video broadcasting.