Received March 19, 2020, accepted April 24, 2020, date of publication April 29, 2020, date of current version May 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991299

An FPGA-Based Architecture for the Versatile Video Coding Multiple Transform Selection Core

MATÍAS J. GARRIDO, FERNANDO PESCADOR, (Senior Member, IEEE), MIGUEL CHAVARRÍAS, PEDRO J. LOBO, CÉSAR SANZ, (Senior Member, IEEE), AND PEDRO PAZ

Software Technologies and Multimedia Systems for Sustainability (CITSEM) Research Center, Universidad Politécnica de Madrid, 28031 Madrid, Spain

Corresponding author: Matías J. Garrido (matias.garrido@upm.es)

This work was supported by the Spanish Ministerio de Economía y Competitividad under Grant TEC2016–75981–C2–2–R.

ABSTRACT Versatile video coding (VVC) will be released by 2020, and it is expected to be the next-generation video coding standard. One of its enhancements is multiple transform selection (MTS) for core transform. MTS uses three different types of 2D discrete sine/cosine transforms (DCT–II, DCT–VIII and DST–VII) and up to 64×64 transform unit sizes. With this schema, significant enhancements of the compression ratio are obtained at the expense of more computational complexity on both encoders and decoders. In this paper, a deeply pipelined high-performance architecture is proposed that implements the three transforms for sizes from 4×4 to 64×64 according to working draft 4 of the standard. The design has been described in very high-speed integrated circuit hardware description language (VHDL), and it has been prototyped in a system on a programmable chip (SoPC). It is able to process up to 64 fps@3840×2.160 for 4×4 transform sizes. To the best of our knowledge, this is the first implementation of an architecture for VVC MTS supporting the 64×64 size.

INDEX TERMS FPGA, hardware architecture, multiple transform selection, pipeline, SoPC, versatile video coding.

I. INTRODUCTION

The future versatile video coding (VVC) standard, currently in committee draft (CD) status [1], will be released as a new the international standard by 2020. It is expected that this new standard will replace the current state-of-the-art high-efficiency video coding (HEVC) [2] with bit rate reductions of more than 30% at the expense of substantial increments in complexity [3].

The new codec will be based on the same hybrid coding scheme used in the previous ITU–T and ISO/IEC standardized codecs (e.g., HEVC). With this scheme, the prediction error is transformed, quantized and encoded into a bit stream. In VVC, a new multiple transform selection (MTS) algorithm has been proposed. In addition to the bi–dimensional (2D) type II discrete cosine transform (DCT–II) [4] used in HEVC, two additional 2D transforms, based on DCT–VIII and DST–VII [4], may be used during the encoding/decoding process. According to WD 4* [5], the DCT–II transforms may have sizes of up to 64×64, while the other two types may have sizes of up to 32×32. In all cases, the transform sizes may be square or rectangular (i.e., with different widths and heights). Additionally, 2D transforms are implemented by concatenating two 1D transforms of the same or different types.

The implementation of the MTS scheme described above incurs higher computational costs and requires more flexibility than the implementation of the transform cores included in the previous standards. Compared with the current state-of-the-art HEVC standard, the VVC maximum block size increases from 32 to 64, which multiplies the complexity by 4.† Additionally, the aforementioned rectangular transform sizes and mixed types, which are not present in HEVC or in other previous standards, demand more flexible architectures.

In this context, it is useful to consider the design of a dedicated processor to perform the function of the MTS core

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

* The architecture proposed in this paper has been implemented and tested in accordance with WD 4.

† The number of multiplications required by a direct implementation of a 2D N×N point DCT/DST is N^2.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

as part of a VVC full implementation. Thus, such a full implementation could be composed of a multicore general purpose processor (GPP) and one or more dedicated processors that execute parts of the algorithm and act as accelerators [6].

In recent years, several hardware architectures have been proposed for the transform cores of HEVC [7]–[14]; only a few of these are related to VVC [15]–[19]. Moreover, none of these architectures describes a complete processor that is ready to be connected to a GPP.

In this paper, a high-performance architecture for the VVC MTS core implementation is presented. It has been designed to compute 2D transforms of up to 64×64 size in accordance with the WD 4 of VVC. It has been prototyped and tested into a development board based on a Cyclone V Intel–Altera chip and supports HD resolution in real time. To the best of our knowledge, this is the first implementation of a VVC MTS core processor in a system on a programmable chip (SoPC).

The remainder of this paper is organized as follows. Section II summarizes the proposals published in recent years and explains the rationale for this proposal. Section III, introduces some background to enhance the understanding of the next sections. Section IV proposes the architecture, which is shown in detail. Section V, explains the test and prototyping details, and the results are compared with those obtained using related implementations. Finally, section VI concludes the paper.

II. STATE OF THE ART AND RATIONALE

In recent years, much work has been invested in developing hardware architectures that accelerate the computation of transforms in the state-of-the-art HEVC standard [7]–[14]. Additionally, as the standardization process of the future VVC progresses, the first proposals related to this standard are arising [15]–[19].

A. PROPOSALS RELATED TO THE HEVC STANDARD

In HEVC, four transform unit (TU) sizes (4×4, 8×8, 16×16 and 32×32) are defined to encode the prediction errors. A DCT–II type transform is used in all cases with the exception of the intra-coded 4×4 luminance blocks, which use a DST–VII type transform. Tables 1 and 2 summarize several of the most interesting proposals regarding hardware architectures that can be used to implement the HEVC transforms. The proposals shown in Table 1 use ASICs as the target technology, while the proposals listed in Table 2 use FPGA technology. It is worth noting that although video codecs are consumer electronic products, FPGA–based implementations are useful for validating the architectures and establishing comparisons among them in the scientific literature. In fact, it is common practice to conduct the implementations using both ASIC and FPGA technologies, as in [8]–[10]. Moreover, although the throughput of the ASIC–based proposals may be roughly one order of magnitude greater than that of the FPGA–based proposals, most of the latter are still able to operate in real time for both HD and UHD formats.

TABLE 1. Comparison of different ASIC–based DCT–II implementations for HEVC.

proposal and year	tech.	clk (MHz)	#kgates‡	throughput
[8], 2013	45 nm	333	205.5	4 096×2 048@30fps
[9], 2014†	65 nm	742	14.9	7 680×4 320@238fps
[10], 2015	90 nm	400	328	7 680×4 320@60fps
[13], 2019	90 nm	401	303	7 680×4 320@240fps

† only 16×16 size is supported. ‡ kgates are logic gates/1 000.

TABLE 2. Comparison of different FPGA–based DCT–II implementations for HEVC.

prop. and year	technology	clk (MHz)	#kALMs (kslices) ‡	throughput
[7], 2013†	Virtex 7 28 nm	256	(2.0)	3 480×2 160@20fps
[8], 2013	Cyclone IV 60 nm	125	16.2	2 569×1 600@30fps
[9], 2014*	Stratix III 65 nm	27	8.0	3 480×2 160@34fps
[10], 2015	Arria II 40 nm	200	7.3	7 680×4 320@30fps
[11], 2017	Stratix III 65 nm	206	5.2	3 480×2 160@37fps
[12], 2017	Arria II 40 nm	160	6.9	3 480×2 160@30fps
[14], 2019	Stratix III 65 nm	139	2.8	3 480×2 160@45fps

†only 8×8 size is supported. *only 16×16 size is supported.
‡ values in brackets are slices/1 000, otherwise, values are ALMs/1 000.
(1 ALM= 2 ALUTs≈2.5 LEs)

The challenge addressed in the mentioned proposals is the hardware implementation of the HEVC DCT–II transforms for the full set of TU sizes at real–time speed for high-resolution video sequences. In [7] and [9], the implementations support only 8×8 and 16×16 TU sizes, respectively. Although it is claimed that implementations could be designed to support all TU sizes using the proposed architectures, this would occur at the expense of a significant reduction in the throughput and/or a significant increase in the logic resources invested in those implementations. The other proposals in Tables 1 and 2 can work with sizes from 4×4 to 32×32 with good throughputs. In addition, the architectures presented in [11] and [12] support DST–VII transforms for 4×4 blocks, but this feature is unlikely to have a great impact on throughput or on logic resources utilization.

In all cases, the proposed architectures take advantage of the DCT–II separability property to accomplish the transform using two 1D transform processors interconnected through a transposition memory. Additionally, all of the proposals exploit the DCT–II symmetry properties to save logic resources by using different variants of fast DCT algorithms [4]. Several proposals, including [10]–[12] and [14], use multipliers in the 1D transform engines to implement the product matrix. Other proposals attempt to save resources by employing alternative strategies for the multiplication. In [8] and [9], multiplications are implemented by add-and-shift operations. In [7], distributed arithmetic techniques [20] are used to implement the multipliers with simple shift and accumulator circuits. In [13], a n-dimensional Reduced Adder Graph (RAG–n) algorithm [21] is used to minimize the number of adders in add–shift-based multiplication circuits.

One important block in all of the architectures is the transposition memory; proposals [7], [9] and [10] use an array of registers and multiplexers, while [8] and [11]–[14] use memories. Regarding the register–based implementations, a 32 × 32 transposition buffer with 16–bit width would consume 16384 registers, a large amount of logic resources. Proposals [7] and [9] do not have this problem because their implementations only support 8 × 8 and 16 × 16 transforms, respectively. In [10], the blocks to be transformed are divided into small blocks of fixed size, and the processing of individual blocks is reordered in such a way that significant reduction of the size of the transposition buffer is achieved. In [8], [11] and [12], the transposition buffer is implemented with a 32 × 32 buffer based on RAM memories. Compared with the register–based buffer, the memory approach provides a more area–efficient way to store large amounts of data, although it is also less flexible. To parallelize the work of the two 1D transform processors in [9] and [10], a ping–pong buffer is implemented; with this mechanism, the first 1D processor can write the first transform results in one block of the transposition memory while the second 1D processor reads the previous results from the other block. In [11], the results of the first 1D transform are transposed before they are written into the memory; thus, the 1D transform processors use the transposition memory as a simple memory buffer. In [13], the first 1D processor writes the first transform results row–wise in the transposition buffer, and the second 1D processor reads them column–wise; at the same time, the first 1D processor writes the second transform results column–wise, and so on.

B. PROPOSALS RELATED TO VVC

In the VVC WD 4, three different transform types, DCT–II, DCT–VIII and DST–VII, are used. For DCT–II types, the transform size may be up to 64 × 64. In addition, rectangular transform sizes (e.g., 8 × 32) are allowed for all combinations. A simplified algorithm is defined to reduce the complexity of 64–size DCT–II transforms; with this algorithm, the high-frequency coefficients of 64×N or N×64 transforms are*zeroed*, although the final result is still a 64×N or N×64 block. A similar approach is used for 32×N or N×32 transforms of the DCT–VIII and DST–VII types. A more detailed description of these new features is given in section III.

Regarding only the transforms and in comparison with HEVC, VVC introduces more complexity in several ways. First, computation complexity is increased due to the 64 × 64 size transforms. Even with the aforementioned simplified algorithm, the computational cost of the 64 × 64 transforms is double the cost of the 32 × 32 transforms. Second, neither DCT–VIII nor DST–VII has the same symmetry properties exhibited by DCT–II. This implies that no fast DCT algorithms may be used in their implementation, thus hindering the optimization of logic. Third, both the simplified transforms and the rectangular transform sizes increase the complexity of the control circuits. Thus, it will be more difficult to reach the high clock frequencies obtained with conceptually simpler proposals for HEVC. Fourth, due to the rectangular sizes and the simplified transform algorithms, the first and second 1D transforms of a given block may have different computational costs. This may be a problem when blocks of different sizes are transformed in sequence because it is difficult to parallelize the operations in an efficient way.

TABLE 3. **Comparison of different proposed hardware architectures for the implementation of VVC transforms.**

prop. and year	tech.	clk (MHz)	#kgates †	#kALMs (kslices) ‡	throughput
[15], 2017	ASIC 90 nm	245	60	n.a.	7 680×4 320@60fps
[15], 2017	FPGA 40 nm	143	n.a.	(5.2)	7 680×4 320@35fps
[16], 2018	FPGA 20 nm	147	n.a.	133	1 920×1 080@35fps
[17], 2019	ASIC 65 nm	250	496	n.a.	7 680×4 320@160fps
[18], 2019	FPGA 20 nm	228	n.a	31.4	3 840×2 160@96fps
[19], 2019	FPGA 28 nm	271	n.a	1.4	1 920×1 080@43fps

† kgates are logic gates/1 000 n.a.: not applicable
‡values in brackets are slices/1 000; otherwise, values are ALMs/1 000

To date, only a reduced set of hardware implementations of the VVC transforms has been proposed [15]–[19]. These implementations are compared in Table 3. In all cases, the proposed architectures are also based on two 1D processors connected through a transposition memory.

In [15], a high-performance 2D transform hardware architecture for future video coding, the previous informal name for VVC, was proposed. The 1D processors are able to implement 5 transform types (DCT–II, DCT–V, DCT–VIII, DST–I and DST–VII) in accordance with the initial algorithm proposal. Inside the 1D processors, each transform is implemented with dedicated hardware; this is rather inefficient, as only one transform is performed at a time. The proposed architecture is implemented with both ASIC and FPGA technology and is able to process 7680 × 4.320 sequences in real time, but it supports only 4 × 4 and 8 × 8 sizes.

In [16], a similar approach is followed to implement the 5 transform types mentioned above; however, 4 × 4, 8 × 8, 16 × 16, and 32 × 32 sizes, as well as all the non–square combinations, are supported. The proposed architecture is carried out with an FPGA. With this implementation, 1920 × 1.080 sequences can be processed in real time, but this is achieved at the expense of a huge amount of logic resources. The data from [16] shown in Table 3 confirm that, as mentioned before, the functional complexity of VVC (e.g., the variety of transform sizes and rectangular blocks) has a negative impact on both the clock frequency and the throughput.

In October 2018, in WD 2 [22], a multiple transform selection (MTS) schema based on DCT–II, DST–VII and DCT–VIII was established, and it has been maintained to date. Based on that schema, in [17], an architecture that supports sizes from 4 × 4 to 32 × 32 but only DST–VII and DCT–VIII types has been proposed. To conserve logic resources, transforms are implemented with add–shift algorithms, and the number of adders is minimized using a RAG–n algorithm [21]. Furthermore, the same logic is reused to implement both DST–VII and DCT–VIII, as the transform matrices actually have the same coefficients (with different signs and positions). Very high throughput is obtained because the logic used to implement the transforms is able to generate a 32–pixel output result every clock cycle. This is achieved at the expense of a large amount of logic resources (Table 3). Additionally, the operation of the second 1D processor is shadowed with the first one in a perfect pipeline, using a 512 × 64 memory to implement a ping–pong buffer. To make it possible to pipeline a mixture of different transform sizes, a TU size of 32 × 32 is considered, and the input data from every TU are read row–wise, while all transforms in the TU are implemented in parallel. The architecture has been implemented with 65 nm ASIC technology; the results are summarized in Table 3. It is worth noting that this implementation does not include DCT–II type transforms, block sizes greater than 32, or the simplified algorithm for block sizes greater than 16.

In [18], an approximation-based approach is proposed to compute both DCT–VIII and DST–VII transforms. It consists of applying low-complexity adjustment stages to the DCT–II to obtain an approximated computation of the other transforms at the expense of a small reduction in the codec performance (e.g., a bit-rate increment for the same quality). The proposed implementation is able to perform both direct and inverse transforms of square sizes up to 32 × 32. The proposal has been implemented with an FPGA. As can be seen from Table 3, the results are very good in terms of throughput and logic resources. Despite its undoubted interest, this type of approximation has not been included in the standard to date.

Finally, in our previous work [19], a 2D multiple transform processor architecture was proposed and implemented with an FPGA. This architecture can process transforms of square sizes up to 32 × 32 and achieves HD real–time operation with a logic consumption significantly smaller than those of the previous proposals.

C. RATIONALE FOR THE PROPOSAL

In this work, an efficient architecture to be used as a hardware accelerator in the MTS core implementation for VVC is proposed. The proposal is based on [19] but incorporates the new features defined in the standard up to WD 4, i.e., DCT–II, DST–VII and DCT–VIII types, rectangular transform sizes up to 64×64, and simplified algorithms for large size blocks.

The same structure used in all the aforementioned proposals, a structure that is based on two 1D processors and a transposition memory, has been chosen. To conserve logic resources while implementing the three transform types, multipliers are used in the 1D processors; in this way, a generic structure can be used to implement any transform type. To further restrict the amount of logic, a single core based on 16 multipliers is recursively used to implement all transform types and sizes. Although this approach limits the speed performance, this is compensated for by a large system clock frequency and an efficient pipeline between the first and the second 1D processors.

Regarding the clock frequency, in the previously mentioned approach the transform computation core has a very regular structure; this results in a more efficient pipelining inside each 1D processor as well as a large maximum frequency of operation. A different issue is the inter–processor pipelining. As has been said, it is difficult to implement because, due to the rectangular blocks and the simplified transform algorithms defined in VVC, a different number of clock cycles may be needed to compute the vertical and horizontal transforms of the same block. One way to solve this problem is to follow the approach mentioned in [17], but this strategy introduces an important restriction to the software that hopefully implements other parts of the encoder/decoder; namely, the entire TU, together with the types and sizes of all the blocks inside, must be available before starting the transform of every TU. In our proposal, large input and output buffers have been included to interface with the GPP. In addition, a large transposition memory working as a circular buffer (instead of the ping–pong buffer implemented in other proposals) has been used. The circular buffer allows several blocks of the same or different sizes to be written while a large block is read. With the aforementioned additional resources, the writing of the input data, the reading of the output data and the computation of transforms have been decoupled. Thus, it is more feasible for the 1D processors to work in a continuous pipeline mode, and, with a variety of input block sizes, the computation time of a 1D processor may be shadowed by the computation time of the other processor.

Finally, there is an issue that has not yet been mentioned. The proposals summarized in this section [7]–[19] claim to support high throughputs (see Table 1, Table 2 and Table 3), but they do not discuss the mechanisms used to move the input data and the output results between the 2D transform processor and a GPP. However, this is a very relevant matter for the implementation of a hardware accelerator. As an example, in [17], a throughput of 7 680 × 4 320@160 fps is claimed. To accomplish that, 32 16–bit pixels should be input to the processor every clock cycle, and the same number of pixels should be output for the results. Thus, a GPP with a 512–bit bus width should be able to transfer more than 497.6 Mwords/s (more than 254.8 Gbps). Even with a throughput of 1 920 × 1 080@30 fps, a 5.8 Mwords/s

TABLE 4. Basis functions for DCT/DST types used in VVC for N–point 1D transforms.

transform type	Basis function $T_i(j)$, i, j = 0, 1,…,N-1
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–VIII	$T_i(j) = \sqrt{\frac{4}{2N+1}} \cdot cos\left(\frac{\pi \cdot (2i+1) \cdot (2j+1)}{4N+2}\right)$
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 4 [23]

(2.98 Gbps) transfer rate should be maintained. The architecture presented in this paper is ready to be connected to a GPP and, in fact, has been tested in an SoPC. In this architecture, two Direct Memory Access (DMA) engines are used to move data between a GPP and the 2D transform processor. To optimize the transfer times, the processor has been provided with a burst mechanism. In this architecture, a set of data blocks of different sizes and types may be sent to the transform processor, which begins processing the first one as soon as the corresponding input block is available. This feature provides flexibility to the GPP, allowing it to order either one or several transforms at a time and, in the latter case, decreasing DMA configuration time as well as shadowing part of the DMA input data transfer with the computing of transforms by the 1D processors.

III. BACKGROUND

A. VVC BLOCK PARTITIONING AND TRANSFORM TYPES

In VVC, pictures are divided into coding tree units (CTUs) with a maximum size of 256 × 256 samples. The CTUs are partitioned into 4 square coding units (CUs), and the CUs may be further divided into square or rectangular CUs with a minimum size of 4 in both dimensions. Leaf CUs are divided into three coding blocks (CBs), one for luminance samples and the other two for chrominance samples. In the encoder, the residuals of these CBs are direct-transformed; in the decoder, the inverse transform outputs are the residuals of the CBs.

In VVC, an MTS scheme is proposed for residual coding for both intra- and inter-CBs [23]. Three transform types are used: DCT–II, DCT–VIII and DST–VII. The basis functions for the 1D transforms of size N are shown in Table 4.

B. COMPUTING THE TRANSFORMS

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

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

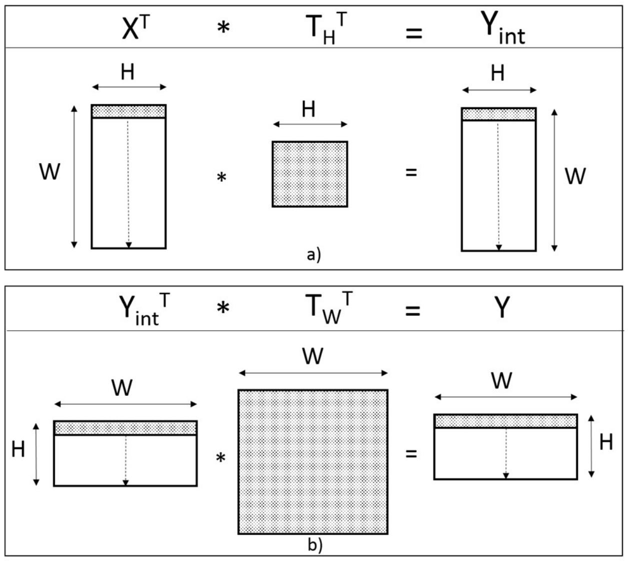


FIGURE 1. Example of the 2D transform of an H×W block: a) horizontal; b) vertical.

In the expression above, X is a 1×N matrix with a row of the input picture block, T is an N×N matrix with the N–point transform coefficients, and Y is an N×1 column matrix with the result of the 1–D transform. The transform coefficients, one set for each transform type and size, are obtained from the basis functions by an integer approximation and will be part of the VVC standard. For convenience, in WD 4 and in the reference software [24], an equivalent approach is used:

$$Y = X^T \cdot T^T \tag{2}$$

For an H×W CB (herein, H and W stand for height and width, respectively), the 2D direct transform computation can be performed by first computing W 1D H–point horizontal transforms:

$$Y_{int} = X^T \cdot T_H^T \tag{3}$$

followed by H 1D W–point vertical transforms:

$$Y = Y_{int}^T \cdot T_W^T = (X^T \cdot T_H^T)^T \cdot T_W^T = T_H \cdot X \cdot T_W^T \tag{4}$$

In expressions (3) and (4), X and Y are H×W matrices with the input and output blocks, and T_H and T_W are the H–point and W–point transform coefficients matrices, respectively. Y_{int} is an intermediate W×H matrix that holds the results of the first W 1D H–point transforms.

As an example, in Fig. 1–a, the W×H transposed input matrix, X^T, is 1D transformed into a W×H intermediate matrix, Y_{int}, and in Fig. 1–b, the transposed H×W intermediate matrix, Y_{int}^T, is 1D transformed into the final H×W output matrix, Y.

The 2D inverse transforms of H×W CBs can be computed in an analogous manner:

$$Y_{int} = X^T \cdot T_H \tag{5}$$

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

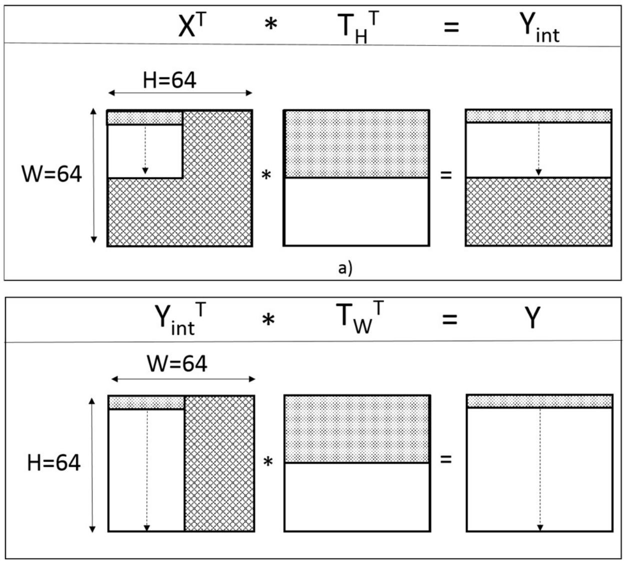


FIGURE 2. Example of the simplified 2D transform of a 64 × 64 block: a) horizontal; b) vertical.

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

To finish, in VVC, the horizontal and vertical types of the transforms are selected by the encoder on a CU basis. The transforms can be of any size from 4 × 4 to 64 × 64. The latter size is only used for DCT–II type transforms.

C. HIGH-FREQUENCY ZEROING FOR LARGE BLOCK-SIZE TRANSFORMS

To reduce the computational cost of large block–size transforms, in WD 4 the effective *H* and *W* of the CBs are reduced depending of the CB size and transform type [5]:

$$nonZeroW = \min(W, (trTypeHor > 0)?16 : 32) \quad (7)$$
$$nonZeroH = \min(H, (trTypeVer > 0)?16 : 32) \quad (8)$$

In expressions (7) and (8), *nonZeroW* and *nonZeroH* are the effective *W* and *H* sizes, *trTypeHor* and *trTypeVer* are the transform types (0: DCT–II, 1: DCT–VIII and 2: DST–VII), and the *min(a,b)* function returns the minimum of *a* and *b*. The sample values beyond the limits of the effective *W* and *H* are considered to be zero, thus reducing the computational cost of 64–size DCT–II and 32–size DCT–VIII and DST–VII transforms.

Fig. 2 shows an example of a 64 × 64.2D direct transform. For the horizontal 1D transform (Fig. 2–a), the 32 high-frequency samples of the first 32 rows, as well as the 32 lower rows, are *zeroed*. As a consequence, the horizontal transform may be computed with only 1/4 of the operations, the lower half of the 64 × 64 transform matrix is not used, and the last 32 output rows are zero. The computation of the subsequent vertical 1D transform (Fig. 2–b) has also been simplified, as only the products involving the upper part of the 64 × 64 transform matrix will produce non–zero results. It is worth noting that in this case the output is still a 64 × 64 matrix, as it would be in a regular 64 × 64 transform.

IV. THE PROPOSED ARCHITECTURE

In this section, an architecture that can compute the 2D inverse transforms of the MTS core for VVC is proposed. The core architecture, consisting of two N–point 1D transform blocks and a transposition memory, is explained in subsections IV.A, IV.B and IV.C. Subsection IV.D explains how a complete SoPC based on this core architecture has been implemented.

A. THE 1D INVERSE TRANSFORM COMPUTATION

The core data path for the inverse transform computation is shown in Fig. 3. It is composed of 4 24–bit multipliers, an adder tree, a 30–bit accumulator, and rounding and saturation logic. The circuit is able to multiply 4 16–bit inputs by 4 8–bit transform coefficients, obtaining a 16–bit output. It has been highly pipelined to allow high clock frequencies and, after an initial latency, it can generate an output every clock cycle. This data path is instantiated 4 times to implement a 1D–MTS processor, as shown in Fig. 4. In the figure, data paths 0…3 are identical except for their ROM content. To compute a 4 × 4 transform, the *start* input is asserted, and the input data are entered (through *din*) on a column basis in 4 clock cycles. After an initial latency, the 4 rows of the output inverse transformed block are generated in *dout*, starting with the top row, in 4 clock cycles.

For sizes other than 4 × 4, the procedure explained previously is iterated. As an example, the computation of an 8 × 16 size inverse transform is illustrated in Fig. 5. It is worth noting that, unlike the examples shown in Fig. 1 and Fig. 2, the 8 × 16 input block has not been explicitly transposed because the processor reads the input data on a column-by-column basis.

In Fig. 5, the 8 × 8 transform matrix has been divided into 4 4 × 4 transform matrices (T1…T4) for convenience. To compute the first output row, R11, the 1D inverse transform circuit reads the first 4 points of the first column from the input, C11, and stores in the accumulator a partial result for R11 using T1. In the next clock cycle, it reads the other 4 points of the first input column, C12, and computes a result using T2, which is accumulated, so that the accumulator now stores the full R11 result. In the next two clock cycles, this process is repeated to compute R12 using T3 and T4. In this way, the entire output row is computed in 4 clock cycles. The whole process is repeated for the 16 columns of the input block to compute the 16 rows of the output transformed block in 16 × 4 clock cycles. Generalizing this procedure, the 1D inverse transform circuit is able to compute square or rectangular transforms of different sizes ranging from 4 × 4 to 64 × 64.

The number of clock cycles needed to compute an H×W 1D inverse transform using the aforementioned algorithm can

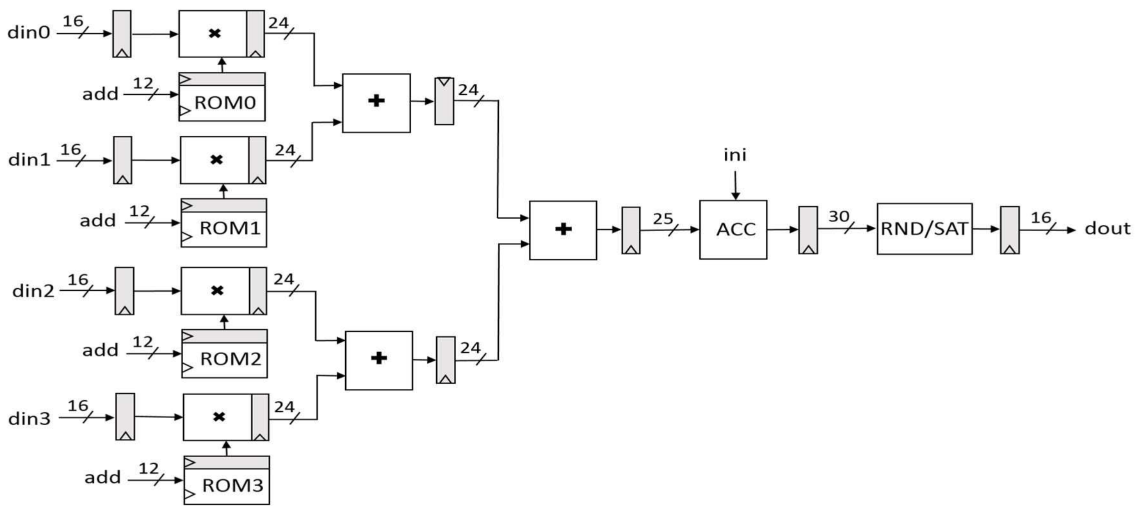


FIGURE 3. Core data path for the inverse transform computation.

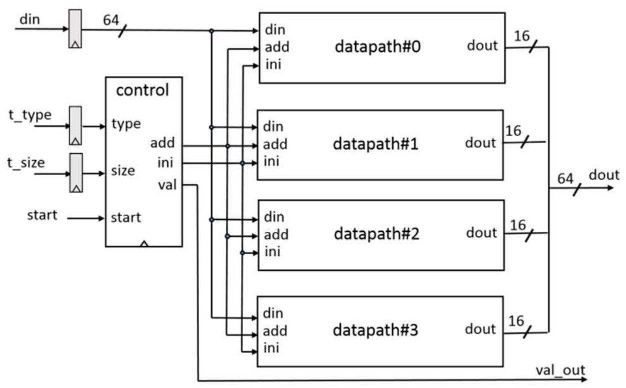


FIGURE 4. Architecture of the 1D inverse transform circuit (1D–MTS processor).

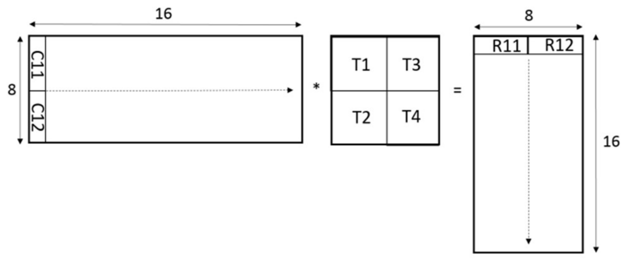


FIGURE 5. Example of computation of an 8 × 16.1D transform.

be obtained from the following expression:

$$N_{cycles\ 1D\ transform} = \frac{1}{a} \cdot \frac{1}{b} \cdot \frac{H^2}{16} \cdot W \quad (9)$$

In the previous expression, variables a and b have been included to take into account the impact of the simplifications explained in subsection III.C. For DCT–II transforms, $a = 2$ if $H = 64$; otherwise, $b = 1$. Additionally, $b = 2$ if $W = 64$; otherwise, $b = 1$. For the other transform types, $a = 2$ if $H = 32$; otherwise, $a = 1$, and $b = 2$ if $W = 32$; otherwise, $b = 1$.

The 1D–MTS processor design is based on the 1D–AMT processor that can be found in [19]. It is worth noting that, unlike 1D–AMT, the 1D–MTS processor proposed in this paper can work with rectangular transforms with up to 64 × 64 sizes, as well as with the simplified transforms mentioned in III.C.

B. PIPELINING

To increase the processor performance, the architecture has been fully pipelined. Regarding the functionality, pipelining introduces extra initial latency cycles. This behavior is shown in Fig. 6; in the timeline, five 4 × 4 transforms of different types are launched at 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, the two transforms run in parallel (L1 and L2 latencies in the figure). This means that the first registers of the pipeline are dealing with the second transform while the rest are still with the first transform. In the worst case, when the fifth transform is launched, all five transforms are in progress, each occupying part of the processor pipeline. This behavior allows the processor to maintain its maximum throughput after an initial latency of 13 clocks. It is worth noting that the controller must be able to work simultaneously with up to five different transform requests in which the transforms have the same or different sizes and/or types. This is the worst case; for transform sizes other than 4 × 4, the number of simultaneous transforms is always lower.

C. 2D INVERSE TRANSFORM COMPUTATION

The 2D inverse transform computation is performed by a new version of the 2D–VVC–MTS processor proposed in [19], with the following enhancements:

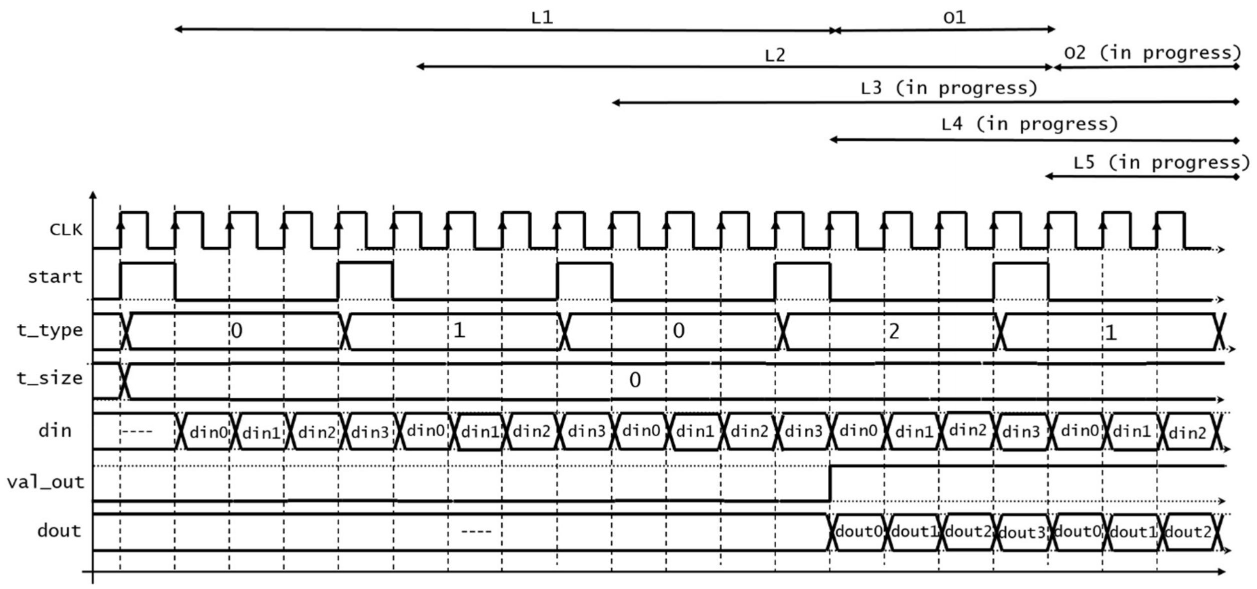


FIGURE 6. Timing chronogram for the fully–pipelined 1D–MTS processor. Five 4 × 4 transforms of different types are launched. Each transform is started with a pulse in *start*, and the results are qualified by *val_out*. When the 5th transform is launched, the other 4 are still in progress.

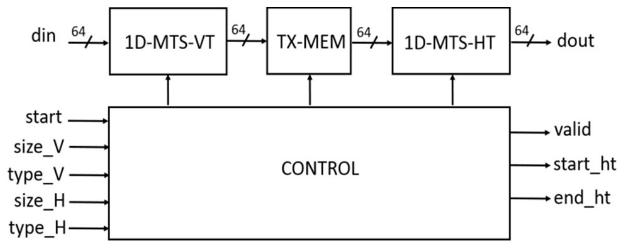


FIGURE 7. Architecture of the 2D inverse transform processor.

- It can process both squared and rectangular blocks.
- It can process blocks up to 64 × 64 in size.
- It can manage *zeroing* for DCT–II, DCT–VIII and DST–VII transform types.

This 2D–VVC–MTS processor uses two 1D–MTS processors to implement a 2D inverse transform with the architecture shown in Fig. 7. The blocks 1D–MTS–VT and 1D–MTS–HT are identical except in their rounding and saturation circuits. The results from the 1D–MTS–VT processor (i.e., the 1D vertical inverse transforms) are stored in a transposition memory, TX–MEM; then, the 1D–MTS–HT processor reads the transposed values and implements the 1D horizontal transform. The transposition memory works as a circular buffer, and it has been designed to store up to 2048 4×4 (or 16 64×64) intermediate results. After 1D–MTS–VT ends a 1D transform, it can proceed with the next transform, while simultaneously, 1D–MTS–HT can start the horizontal transform.

The number of clock cycles needed to compute an H×W 2D inverse transform is obtained by adding the number of cycles needed to compute the corresponding vertical transform (given in expression (9)) and the number of clock cycles needed to compute the horizontal transform:

$$N_{cycles\ 1D\ H\ transform} = \frac{1}{b} \cdot \frac{W^2}{16} \cdot H \qquad (10)$$

In (10), b, H and W have the same meanings as in (9). In Table 5, the numbers of cycles needed to compute the inverse transforms of all of the supported sizes are summarized. In both expression (10) and Table 5, H and W refer to the size of the block to be inverse transformed, which is the input block used in the first 1D (vertical) transform.

It is worth noting that if the 2D processor works in a continuous way, the vertical and horizontal processors will work in parallel, and the numbers stated in Table 5 may be substantially reduced. As an example, one 32 × 32 DCT–II transform will require 2048 + 2048 = 4096 clock cycles, but 8 transforms of the same type and size will be completed in 2048 + 7 × 2048 + 2048 = 18432 clock cycles, as both processors will work in parallel during 7 1D transforms.

D. ARCHITECTURE OF THE SoPC

An SoPC with a 2D–VVC–MTS processor, a flexible input/output interface and a GPP has been designed to test the functionality and performance of the processor.

1) INPUT BUFFER

An input buffer has been provided with a FIFO–based interface to enable it to temporarily store the input blocks and commands corresponding to the inverse transforms to be computed by the 2D–VVC–MTS processor. Actually, this module implements a 64–bit memory mapped input data interface with two addresses. The first address is for writing data blocks (e.g., a 4×4 block) into an input data buffer, while

TABLE 5. Number of clock cycles needed to compute the 1D inverse transform (vertical and horizontal) for different transform sizes.

size H×W	#clock cycles		size H×W	#clock cycles	
	vertical trans.	horizontal trans.		vertical trans.	horizontal trans.
4×4	4	4	32×4	256	32
4×8	8	16	32×8‡	256	128
4×16	16	64	16×32‡	256	512
4×32‡	16	128	32×32‡	512	1 024
8×4	16	8	64×4†	512	64
4×32	32	256	32×8	512	128
4×64†	32	512	16×32	512	1 024
8×8	32	32	32×16‡	512	512
16×4	64	16	16×64†	512	2 048
8×16	64	128	64×8†	1 024	256
8×32‡	64	256	32×16	1 024	512
32×4‡	128	32	32×32	2 048	2 048
16×8	128	64	32×64†	2 048	4 096
8×32	128	512	64×16†	2 048	1 024
8×64†	128	1 024	64×32†	4 096	4 096
16×16	256	256	64×64†	4 096	8 192

† DCT–II type; *zeroing* applied for size > 32
‡ DCT–VIII and DST–VII types; *zeroing* applied for size > 16

the second address is for writing commands (e.g., types and sizes of V and H transforms) into a command buffer.

A simplified block diagram of the input buffer is shown in Fig. 8. It is composed of seven modules:

- GPP–I/F is the input memory mapped interface. The *cmd/data* input is the address (0 to write data and 1 to write commands).
- IBUF–MEM is a circular buffer with the same architecture as the transposition memory mentioned in III.C.
- WR–GEN generates addresses and control information to write the data blocks into IBUF–MEM.
- RD–GEN generates addresses and control information to read the data blocks from IBUF–MEM. It also generates the *start* order to the 2D–VVC–MTS processor.
- COMM–BUF1 is a FIFO buffer that is used to store the commands containing the types and sizes of the transforms to be computed.
- COMM–BUF2 is a FIFO buffer that is used to store the commands containing the types and sizes of the transforms to be sent to the 2D–VVC–MTS processor along with the *start* orders. The types and sizes corresponding to individual 2D transforms are read from COMM–BUF1 once the full data block corresponding to these individual transforms has been written into IBUF–MEM.
- WR–CB is a block that generates the *wr_req* write signal to read commands (types & sizes) from COMM–BUF1 and write them into COMM–BUF2.

As shown in Fig. 8, the input buffer has two clock domains. The *interface clock domain* corresponds to the GPP–I/F,

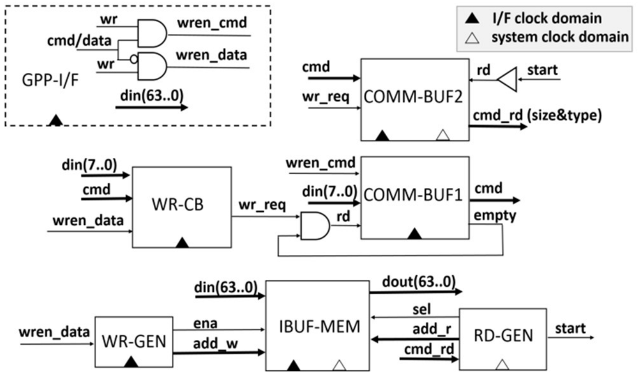


FIGURE 8. Simplified block diagram of the input buffer.

WR–CB, WR–GEN and COMM–BUF1 modules. The *system clock domain* corresponds to the RD–GEN module. The IBUF–MEM and COMM–BUF2 modules work with the *interface clock domain* in their inputs and with the *system clock domain* in their outputs. This separation increases the flexibility in the SoPC physical design phase.

To start a single 2D transform, a command with information on both size and type must be written to address 1. The command is written into COMM–BUF1. The input data block must then be written to address 0 row-by-row as 4 16–bit samples in every interface clock cycle. When the entire input data block has been written into IBUF–MEM, the command is moved into COMM–BUF2. Finally, RD–GEN reads the command from COMM–BUF2, begins reading the input data block from IBUF–MEM, and generates a *start* for the 2D–VVC–MTS processor.

The input buffer has been designed to efficiently support the pipelined work of the 2D–VVC–MTS processor. It has been dimensioned to store up to 8 32 × 32 input blocks (i.e., 512 4 × 4 blocks). This allows it to store enough data to implement up to 8 64 × 64 DCT–II transforms because the *zeroed* high-frequency points are not actually written into the input buffer. To start a burst of #N transforms, #N commands with information about the sizes and types of the individual 2D transforms can be written to address 1 (in COMM–BUF1). Next, the input data blocks corresponding to these transforms must be written to address 0 (into IBUF–MEM) in sequence. After each single input data block has been written, the corresponding command is moved into COMM–BUF2. When the first command is available in COMM–BUF2, the first transform is started by RD–GEN. Thus, while the 2D–VVC–MTS processor is computing the first transform, the other input data blocks are being written into IBUF–MEM, and the corresponding commands are being moved from COMM–BUF1 to COMM–BUF2. In this way, when the 2D–VVC–MTS processor ends with a 1D vertical transform, a new transform will be started automatically.

2) OUTPUT BUFFER

The output buffer is also a FIFO–based interface that is used to store the output data blocks corresponding to the inverse

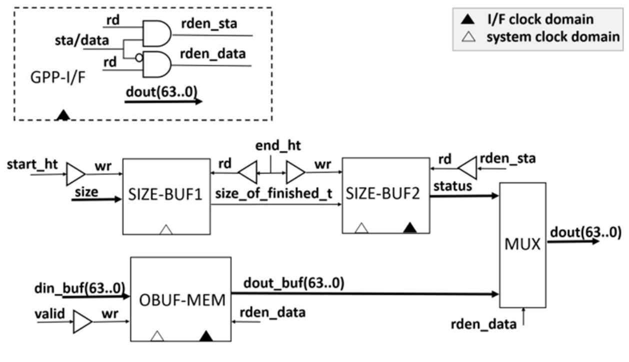


FIGURE 9. Simplified block diagram of the output buffer.

2D transform results computed by the 2D–VVC–MTS processor. The first address is used to read data blocks from a buffer, while the second is used to read status information.

A simplified block diagram of the output buffer is shown in Fig. 9. It is composed of five modules:

- GPP–I/F is the output memory mapped interface. The *sta/data* input is the address (0 to read data and 1 to read status).
- OBUF–MEM is a FIFO buffer. It stores the 64–bit output data from the 2D–VVC–MTS processor until it can be read through the GPP–I/F.
- SIZE–BUF1 is a FIFO that stores the sizes of every individual transform started by the 2D–VVC–MTS processor.
- SIZE–BUF2 is a FIFO that stores the sizes of the individual transforms actually computed by the 2D–VVC–MTS processor.
- MUX is a module that allows reading of either the output data blocks stored in OBUF–MEM or the status information through the GPP–I/F.

As shown in Fig. 9, the SIZE–BUF1 module belongs to the *system clock domain*. The GPP I/F belongs to the *interface clock domain*. The other modules have the *system clock domain* for their inputs and the *interface clock domain* for their outputs.

The OBUF–MEM has been dimensioned to store up to 8 64 × 64 DCT–II output blocks (i.e., 2048 4 × 4 blocks). The output transformed blocks stored into the OBUF–MEM can be read in sequence. By reading the processor status, information about the number of blocks stored in the OBUF–MEM, the block size of the first block, and the *empty* and *full* flags from the SIZE–BUF2 FIFO can be obtained.

3) ARCHITECTURE OF THE SOPC

The SoPC consists of a 2D–VVC–MTS processor that is connected to the input and output buffers described in the previous subsections. In addition, two DMA engines are used to move data from the GPP to the INPUT BUFFER and from the OUTPUT BUFFER to the GPP. The system architecture is shown in Fig. 10.

In the SoPC, the GPP writes one or more input data blocks to be inverse-transformed into an internal On Chip

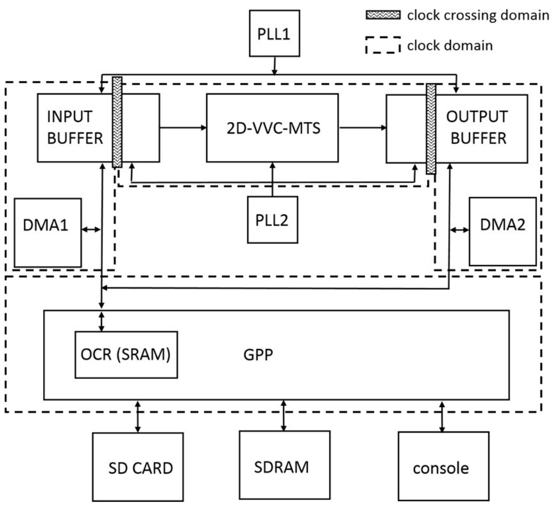


FIGURE 10. Architecture of the SoPC.

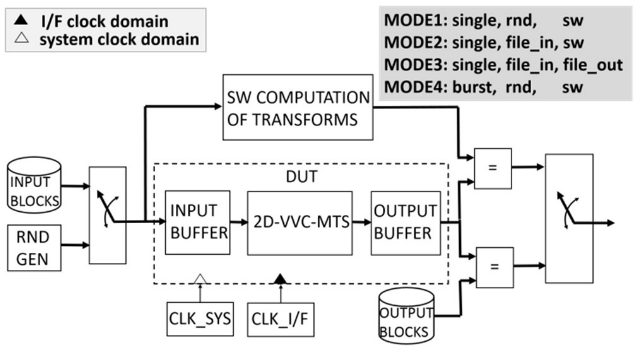


FIGURE 11. Simplified block diagram of the VHDL testbench.

RAM (OCR). After that, it writes one or more commands to the COMM–BUF1 FIFO (inside INPUT BUFFER), each with the types and sizes of the individual transforms. It then configures DMA1 to transfer the data blocks from the OCR into the IBUF–MEM (also inside INPUT BUFFER). The GPP has time to perform other tasks while the input blocks are transferred to the INPUT BUFFER and the transforms are computed and stored in the OUTPUT BUFFER; alternatively, it can continuously poll the status in the OUTPUT BUFFER interface to determine how many transforms have been completed. Finally, the GPP may configure the DMA2 to transfer the results from the OUTPUT BUFFER into the OCR.

V. TESTS AND RESULTS

A. VHDL TESTBENCH

A VHDL testbench has been designed and used to fully test the 2D–VVC–MTS processor and the input and output buffers using ModelSim simulator [25]. The testbench (see Fig. 11) may be configured to run tests in 4 modes.

In mode 1, the testbench uses pseudorandom input data blocks and commands with all the supported types and sizes. A uniform distribution was selected for the random generator. For each block, the 2D inverse transform is performed by the

2D–VVC–MTS processor and computed in software at the same time, and the results are automatically compared for equality.

In mode 2, the testbench reads the input data blocks and commands from a file. The 2D–VVC–MTS processor performs the inverse transforms, and the results are automatically compared with the transforms computed by the software. Mode 3 works in the same way as mode 2, but the results are compared with those stored in a file instead of with results computed by the software.

Finally, mode 4 works in the same way as mode 1 except that the input data, types and sizes are written and transformed in bursts of fixed size.

In all cases, a 150-MHz clock was used for the *interface clock domain*, and a 200-MHz clock was used for the *system clock domain*. These values are directly related to the logic synthesis results summarized in subsection V.C. With this testbench, the processor has been verified using mode 1 and mode 4 tests. In each case, a set of 10^6 2D inverse transforms of random sizes and types was tested and verified. Additionally, mode 2 and mode 3 were used to conduct tests with *Cactus*, *BasketballDrive* and *BQTerrace* sequences with *Random Access 8–bit* decoding configuration and quantization parameter (QP) 32; these are 1920 × 1.080 sequences defined according to common test conditions [26]. In this case, the input data files (needed for both mode 2 and mode 3 tests) and the output data files (needed only for mode 3 tests) were obtained by *instrumenting* the decoder in the VVC reference software [24]. Version 4.2 was used for this purpose.

B. 2D–VVC–MTS PERFORMANCE RESULTS

A first set of performance results was obtained with the VHDL testbench working in mode 3. The results are shown in Table 6 for the aforementioned *Cactus* (CA), *BasketballDrive* (BD) and *BQTerrace* (BQ) sequences. For each sequence, the total decoding time and the average decoding time per picture were obtained (see *Full seq.* column in Table 6). Additionally, for both I–type and P–type pictures, the decoding time of the slowest and fastest picture and the average decoding time per picture are shown. The obtained full sequence average decoding times per picture lead to performances ranging from 282 to 1052 fps depending on the sequence. With these numbers, real–time is supported for both HD (1920 × 1.080) and UHD (3840 × 2.160) resolutions. These quite good results benefit from the fact that most B pictures contain many skipped blocks. Thus, the average time required to decode I pictures ranges from 5 to 23 times the average time required to decode B pictures depending on the sequence. If only I–type pictures are taken into account, performance ranges from 28 to 48 fps.

Although the results given previously are close to those that could be obtained in a real scenario, they depend greatly on the type of video sequences used in the tests. To obtain objective results that can be compared with those obtained using other implementations, it is usual to compute the performance in number of fps when all picture blocks are transformed (i.e., with no skipped pictures). Table 7 shows the theoretical number of frames per second that can be processed by the VVC–MTS–PROC for both HD and UHD resolutions in a number of scenarios. In all cases, it has been assumed that the processor works with a 200 MHz system clock (subsection V.C) in a fully pipelined mode, i.e., that it is always computing transforms one after another. In this way, the 1D vertical and horizontal transforms of different blocks can run in parallel, and the time needed to complete a transform can be computed as the time needed to complete the horizontal 1D transform. It has also been assumed that the data motion to and from the processor runs in parallel with the transforms processing. The last 6 columns of Table 7 correspond to simple scenarios in which all blocks are square and of a fixed size. In the last column, *Mix*, the frames are supposed to contain blocks with a uniform distribution of random sizes and types.

With these numbers, real time is supported for HD resolution. It is worth noting that the processor performance is closely related to the system clock frequency. The FPGA used to prototype the SoPC is a low–end 28-nm chip [27]. The use of medium–end [31], [32] or high–end [32], [33] FPGAs or of an ASIC would greatly increase the maximum system clock and hence the performance in terms of number of frames per second.

TABLE 6. Performance results obtained in mode 3 tests for the Cactus (CA), BasketBallDrive (BD) and BQTerrace (BQ) sequences.

seq. (#pictures)	Full seq.		I pictures			B pictures		
	total dec. time (s)	avg. per pic. (ms)	slowest (ms)	fastest (ms)	avg. (ms)	slowest (ms)	fastest (ms)	avg. (ms)
CA (500)	1.34	2.7	14.2	15.3	14.5	9.3	0.002	2.3
BD (500)	1.77	3.5	19.4	15.3	17.4	17.8	0.142	3.1
BQ (600)	0.57	1.0	11.2	14.7	12.4	16.1	0.000	0.6

TABLE 7. Theoretical number of frames per second obtained with the VVC-MTS-PROC implementation for HD (1920 × 1.080) and UHD (3840 × 2.160) resolutions. 4:2:0 sampling was considered for all cases.

resolution	transform size						
	4×4	8×8	16×16	32×32	32×32‡	64×64†	Mix
1 920×1 080	257	128	64	32	64	32	40
3 840×2 160	64	32	16	8	16	8	10

† DCT–II type; *zeroing* applied for size > 32
‡ DCT–VIII and DST–VII types; *zeroing* applied for size > 16

C. IMPLEMENTATION AND LOGIC SYNTHESIS RESULTS

The SoPC has been implemented in a Cyclone V 5CSXFC6D6F31C6 FPGA [27] and prototyped with a

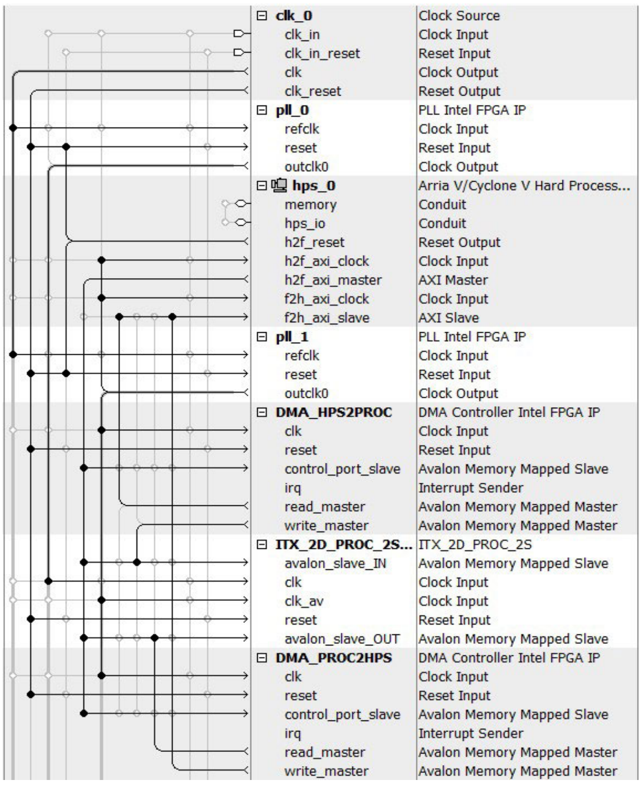
TABLE 8. Physical design results for the SoPC design.

max. i/f clock (MHz)	max. sys clock (MHz)	#reg	#mul	#ALMs	#RAM blocks (×10Kb)
151	204	9 104	32	5 179	391

SoCkit development board [28]. It has been built using *Platform Designer* [29]. Fig. 12 shows the *Platform Designer* view of the SoPC. The VVC–MTS–PROC block groups the 2D–VVC–MTS processor and the input and output buffers. The other blocks are Intel–Altera IPs: a CLKIN module to input a 50-MHz clock, two PLLs to generate the clocks of the two domains from the 50-MHz clock, the GPP (an ARM Cortex–A9 MPCore), and two DMA controller engines to move data between the VVC–MTS–PROC and the GPP.

The GPP has two 64–bit AXI interfaces, a master and a slave. The master interface (*h2f_axi_master* in Fig. 12) is used to write the commands to the input buffer, read the status from the output buffer, and configure the DMA controllers. The slave interface (*f2h_axi_slave*) is used by the DMA1 controller to move data from the GPP OCR to the VVC–MTS–PROC input buffer. It is also used by the DMA2 controller to move data from the VVC–MTS–PROC output buffer to the GPP OCR. It is worth noting that the VVC–MTS–PROC input and output buffers have *Avalon Memory Mapped* interfaces; the compatibility between the Avalon and AXI interfaces is solved by the interconnection logic automatically generated by *Platform Designer* [29].

The logic synthesis and physical design of the system were performed using *Quartus Prime* [30]. Experimentally, we chose clocks of 150 MHz for the *interface clock domain* and 200 MHz for the *system clock domain*. To support these clock speeds, the logic synthesis tool has been constrained to prioritize speed versus area (i.e., resource consumption). Additionally, retiming techniques have been used to increase the clock speed. The physical design results are summarized in Table 8.



⊟ **clk_0**	Clock Source
clk_in	Clock Input
clk_in_reset	Reset Input
clk	Clock Output
clk_reset	Reset Output
⊟ **pll_0**	PLL Intel FPGA IP
refclk	Clock Input
reset	Reset Input
outclk0	Clock Output
⊟ **hps_0**	Arria V/Cyclone V Hard Process...
memory	Conduit
hps_io	Conduit
h2f_reset	Reset Output
h2f_axi_clock	Clock Input
h2f_axi_master	AXI Master
f2h_axi_clock	Clock Input
f2h_axi_slave	AXI Slave
⊟ **pll_1**	PLL Intel FPGA IP
refclk	Clock Input
reset	Reset Input
outclk0	Clock Output
⊟ **DMA_HPS2PROC**	DMA Controller Intel FPGA IP
clk	Clock Input
reset	Reset Input
control_port_slave	Avalon Memory Mapped Slave
irq	Interrupt Sender
read_master	Avalon Memory Mapped Master
write_master	Avalon Memory Mapped Master
⊟ **ITX_2D_PROC_2S...**	ITX_2D_PROC_2S
avalon_slave_IN	Avalon Memory Mapped Slave
clk	Clock Input
clk_av	Clock Input
reset	Reset Input
avalon_slave_OUT	Avalon Memory Mapped Slave
⊟ **DMA_PROC2HPS**	DMA Controller Intel FPGA IP
clk	Clock Input
reset	Reset Input
control_port_slave	Avalon Memory Mapped Slave
irq	Interrupt Sender
read_master	Avalon Memory Mapped Master
write_master	Avalon Memory Mapped Master

FIGURE 12. Platform Designer view of the SoPC.

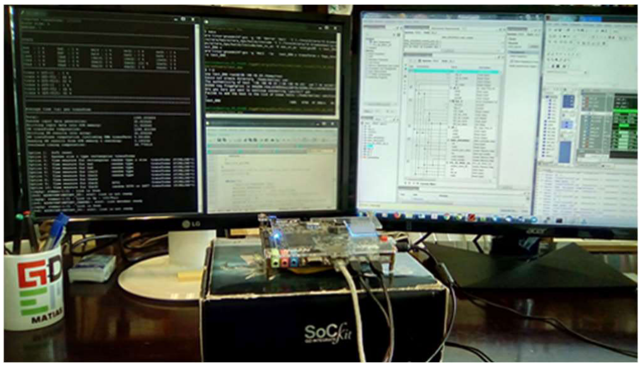


FIGURE 13. Running the test with the prototyping board.

D. TESTS CONDUCTED WITH THE PROTOTYPE

Tests were conducted with the prototype by computing a wide set of transforms with random input blocks, transform types and sizes.

In this testbench, the GPP boots a Linux OS from an SD card that also contains the FPGA configuration file generated by *Quartus Prime*; this file is transferred to the FPGA during the OS booting. Finally, a UART interface is used to implement a PC terminal to run applications from the GPP OS. A test application was written in C language and compiled to be run in the GPP. When executing the application from the console, a menu is displayed with the following options:

- Option 1: A reset command is sent to the processor. The effect is the same as that of a hardware reset.
- Option 2: A set of 10^6transforms, each with random input data blocks, type and size, is performed.
- Option 3: A set of 10^6 transforms, grouped in bursts of 8, are performed. Each transform has random input data blocks, types and sizes.
- Options 4, 5, 6, 7 and 8: These are the same as option 3 except that the block sizes are fixed to N×N, where $N = 4 \times 2^{(\#option-4)}$.
- Option 9: This option works similar to option 3 but with block sizes fixed to 32 × 32 and transform types fixed to DCT2.
- Option 10: This option works similar to option 9, but the block types are randomly selected from DST–VII and DCT–VIII.

In all of the above options, the GPP also computes the same transforms by software and compares the results with those

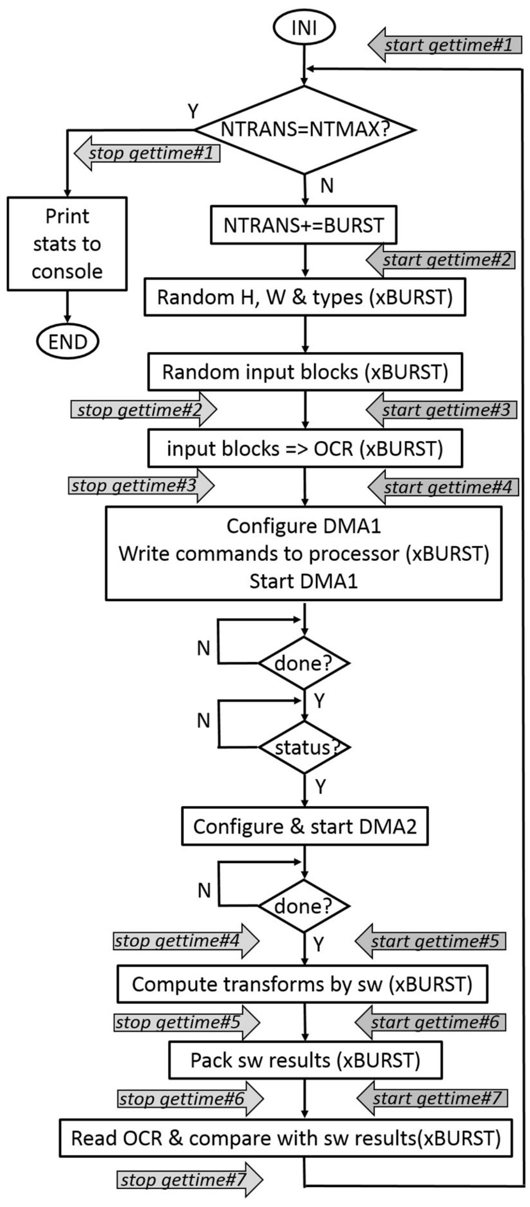


FIGURE 14. Simplified flow diagram of test option 3.

obtained by the processor on the fly. The software implementation is based on V 4.2 of VVC Reference Software. Fig. 13 shows the prototyping board connected to a PC host using an USB–UART interface. The console on the left side shows the test menu.

Test option 2 is focused on validating the processor's functionality. After the execution of this test, the console shows the percentage of blocks of different sizes and transform types used in the test and indicates whether errors were found when the hardware- and software-computed transforms were compared.

The remaining test options focus on measuring the time needed to perform transforms in the burst mode. In Fig. 14, a simplified flow diagram for test option 3 is shown (the flows in options 4…10 are similar; the flow in option 2 is simpler). Initially, *NTMAX* is fixed to 10^6, and *BURST* is set to 8. For each loop, a random set of 8 input blocks with random types and sizes is generated. After the input blocks are written into the OCR, 8 commands (one per transform) are written to the processor input buffer (into COMM–BUF1), and DMA1 is configured and begins to move the input blocks from the OCR to the processor input buffer (into IBUF–MEM). After DMA1 transfer is completed, the processor status is polled until the transforms are completed. It is worth noting that, in an actual application, the GPP could perform other tasks instead of continuously polling the DMA and the processor. When the transforms have been computed, DMA2 is configured and begins to move the results from the processor output buffer (from OBUF–MEM) into the OCR. Finally, when DMA2 transfer has been completed, the 8 transforms are computed in software, and the results are compared with the results stored in the OCR. When the *NTMAX* transforms have been performed, the console shows statistics for the number of transformed blocks of each size and type and the average time used to run various parts of the test.

To measure time, the Linux *clock_gettime* function was used. As shown in Fig. 14, the execution time between each sequential pair of 7 test points labeled from 1 to 7 was measured. The time measured for test point #N is the GPP execution time between the *start gettime #N* point and the *stop gettime #N* point in Fig. 14. These test points correspond to the following time measurements:

- Test point #1: whole loop.
- Test point #2: generation of random inputs, types and sizes.
- Test point #3: copying of input blocks into the OCR.
- Test point #4: implementation of the transforms, including data motion into/from the processor.
- Test point #5: software computation of the transforms.
- Test point #6: packaging of software results to make them ready for the comparison.
- Test point #7: time spent comparing processor and software-computed results.

Table 9 outlines the main results obtained with the aforementioned tests. All of the measurements shown in this table are given in microseconds. The first row, *Th*, was included for reference. It corresponds to the theoretical time needed by the 2D–VVC–MTS processor to complete transforms of different sizes and types assuming fully pipelined work. The times in this row are obtained by multiplying the system clock period (5 ns) by the number of clocks needed to implement the horizontal 1D transform, which is given in Table 5.

The times in the second row, *Hw*, come from the measurements obtained with test options 3, 4, 5, 6, 8, 9 and 10. They include the DMAs configuration, the time to move data from the GPP OCR to the 2D–VVC–MTS processor, the time to complete the transforms, and the time to move the results back to the OCR. In all cases, the transforms are computed in bursts of 8. Thus, the fastest 1D transforms are shadowed by the slowest ones; this is true for all but the first

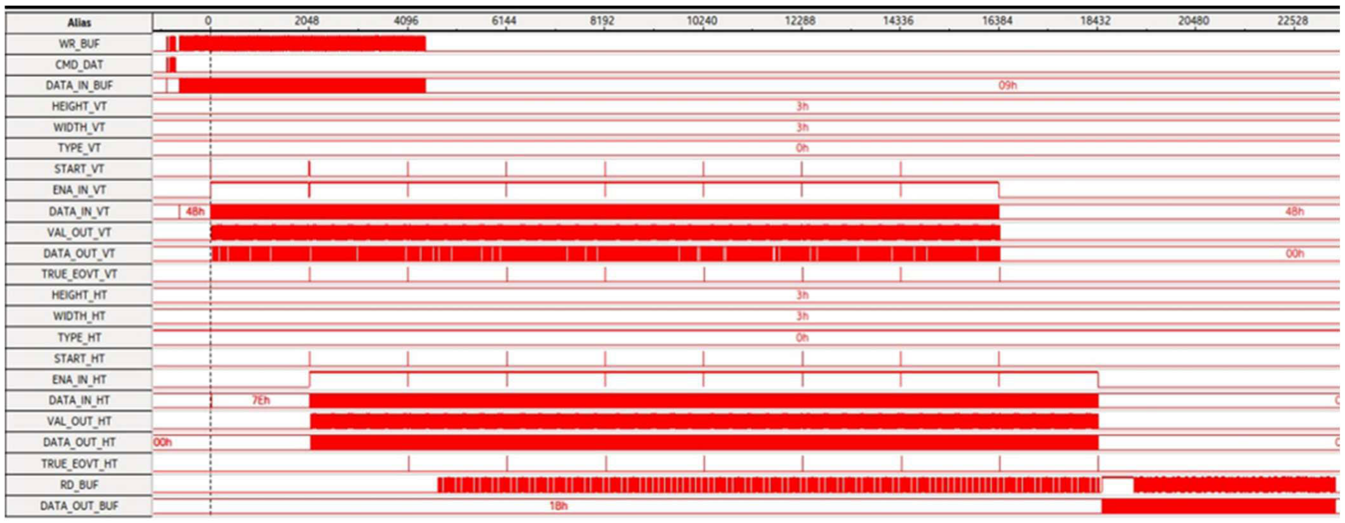


FIGURE 15. Snapshot of the Logic Analyzer timing chronogram obtained during test 8.

TABLE 9. Time to complete 2D transforms of different sizes and types (μs).

source	mix*	4×4	8×8	16×16	64×64	32×32	32×32
		random types only			DCT–II		DCT–VIII DST–VII
Th†	4.53	0.02	0.16	1.28	40.96	10.24	5.12
Hw ‡	9.52	0.77	1.09	2.99	58.4	15.98	9.58
Sw**	440.5	3.5	43.6	107.1	2 652.9	891.8	1 268.4

† Theoretical. 200 MHz system clock. Fully pipelined work (the operation of one 1–D processor is shadowed with the other one).
‡ Measured (hardware). Includes the time to write the input data and to read the results. 8 transform bursts; the time measured for a burst is divided by 8.
* Mixture of random selected types and rectangular sizes.
** Measured (software). Transforms are computed by the GPP in software.

and the last transforms in each burst. Moreover, part of the time to move data from the OCR to the processor is also shadowed with the computation of the first transforms. The times shown were obtained by dividing the average time to complete 125000 bursts by 8.

Finally, in the third row, *Sw*, the time employed by the GPP to compute the same transforms by software, is given for reference. The software implementation is based on the VVC reference software; in particular, no optimizations were performed to take advantage of any GPP special instructions.

The measurements in Table 9 show that the 2D–VVC–MTS processor's actual computation times are close to the theoretical times only for large block sizes (e.g., 32 $\times$ 32 or 64 $\times$ 64). For small sizes, the time required to program the DMA configuration and move data to/from the processor is dominant. However, for all block sizes, the computation times are significantly shorter than those obtained with the software implementation.

E. LOGIC ANALYZER SNAPSHOTS

To obtain further insight into the SoPC operation, the Quartus Prime Signal Tap Logic Analyzer was used to monitor several key internal signals in real time. As an example, Fig. 15 shows the snapshot obtained during the occurrence of a 32×32 burst in test option 8. In the upper left corner, the simultaneous activity in the WR_BUF and CMD_DAT signals shows the writing of the 8 commands into the INPUT BUFFER. The activity in WR_BUF that occurs while CMD_DAT = '0' identifies the DMA1 transfers of the input data blocks from the GPP OCR to the INPUT BUFFER. When enough data has been moved into the INPUT BUFFER, the first vertical transform is started; this can be identified by the pulse in the internal START_VT signal. When a vertical transform is completed, a new one is started, and a horizontal transform (START_HT) is started at the same time. It is worth noting that the DMA1 data motion runs in parallel with the first 1D vertical transforms. Additionally, with the exception of the first vertical transform and the last horizontal transform, the vertical and horizontal transforms run in parallel. Finally, when the last horizontal transform is completed, the DMA2 transfers the results from the OUTPUT BUFFER to the GPP OCR. This can be identified by the activity on the RD_BUF and DATA_OUT_BUF signals.

F. COMPARISON WITH OTHER PROPOSALS

Table 10 summarizes the key parameters used to compare the performance of the 2D–VVC–MTS processor with that of other implementations. All proposals in Table 10 implement 2D transforms using two 1D transform processors plus a transposition memory. The type of transform implemented in the proposals (direct or inverse) is not relevant for the purpose of the comparison. To ensure a fair comparison, the data included in Table 7 were used to characterize the performance of our proposed implementation (SoPC, in the first row).

TABLE 10. Comparison of FPGA–based implementations considering UHD (3840 × 2160) resolution and 4:2:0 sampling.

im pl.	fps@trans_size					clk MHz	#kreg *	#mul	#kALM (kslice) [kgates] * *
	4	8	16	32	64†				
SoPC, 28 nm	64	32	16	8	8	200	9.1	32	5.1
[15], 40 nm	93	93	n.a.	n.a.	n.a.	143	4.5	0	(5.2)
[16], 20 nm	7	12	24	13	n.a.	147	274	1561	133
[17], 65 nm	643 ×64	643 ×16	643 ×4	643	n.a.	250	n.a.	n.a.	[496]
[18], 20 nm	11‡	19‡	41	96	n.a.	228	75.5	738	36.7
[19], 28 nm	87	43	21	10	n.a.	271	4.6	32	1.4

† DCT–II type, *zeroing* applied as required
n.a. not applicable (not supported)
* kreg are registers/1000
**values in round brackets are slices/1000; values in square brackets are gates/1000; others are ALMs/1000.
‡ This result is not given in [18]. It was assumed that both 4 × 4 and 8 × 8 transforms are implemented as in [16] but with the (higher) clock frequency in [18].

At this point, it is important to say that the implementation described in this paper has a set of unique features:

- First, it is the only implementation that supports rectangular block sizes up to 64 × 64. The proposal in 17] only supports DCT–VIII and DST–VII transform types (with sizes up to 32×32, as defined in the standard). The proposal in [15] supports 4 × 4 and 8 × 8 sizes only. The other proposals ([16], 18] and [19]) support sizes up to 32 × 32. Rectangular blocks are only supported in [16] and in our proposal.
- Second, it is the only implementation that supports the simplified computation for the higher size transforms that were defined in WD 4 (and that have been maintained in the actual CD).
- Third, our proposal is the only proposal that has been implemented as a peripheral ready to be connected to a GPP. Moreover, it has been implemented and tested in an SoPC.

It is worth noting that all of these characteristics introduce a complexity in the design that is not present in the other proposals, posing a challenge in achieving good results with respect to both consumed logic resources and throughput.

Regarding logic resources consumption, our proposal is clearly better than [16] and [18]. Even considering that the implementation in [18] includes both direct and inverse transforms, the differences are very large. Compared with [15], our implementation has 32 multipliers and twice the number of registers, but it supports rectangular blocks up to 64 × 64 size. A fair comparison with proposal in [17] is difficult as it is an ASIC implementation. In any case, its functionality is very limited compared with that of our proposal. Finally, it must be taken into account that the architecture proposed in this paper is an enhancement of our previous work in [19]; this enhanced system supports 64 × 64 block sizes, rectangular blocks and simplified computation of large transforms and includes the input and output buffers and the system architecture described in subsection IV.D. All of these improvements are responsible for the increment in logic resource usage compared with [19].

As would be expected, the best throughputs are obtained by the less complete implementations [15] and [17]. Our implementation yields results similar to those of [16] but uses far less logic resources. The implementation in [18] exhibits very good performance for 32 × 32 block sizes, but the throughput decreases as the block sizes go down. Our implementation has the opposite behavior. In an actual application, a mixture of blocks of different sizes can be expected; in this case, the performances of the two proposals approach similarity. As an example, for a mix of equally probable blocks of sizes from 4 × 4 to 32 × 32, our proposal would have a throughput of 30 fps, while the proposal in [18] would reach 41 fps. It is worth noting that, with the same mixture of blocks, the proposal in [19] has a throughput of over 40 fps. From this, it follows that the difference (40 − 30 = 10) is the cost of supporting 64 × 64 block sizes, rectangular blocks, simplified computations for large transforms and a complete interface for a GPP. Additionally, as mentioned previously, the approximated approach in [18] is non–standard and has a slight cost in terms of bit rate.

Last but not least, it should be noted that our proposal was implemented with a low–end FPGA, while proposals [16] and [18] were implemented with medium–end FPGAs, and proposal [17] was implemented with an ASIC. Our estimation is that with a medium or high–end FPGA or an ASIC, the architecture proposed in this paper could support real–time UHD formats.

VI. CONCLUSION

In this paper, 2D–VVC–MTS, an efficient FPGA–based architecture for the computation of the future versatile video coding (VVC) multiple transform selection (MTS) core, has been proposed. The architecture supports rectangular transform sizes ranging from 4 × 4 to 64 × 64 and simplified algorithms for large bocks according to WD 4 of the standard.

Moreover, 2D–VVC–MTS has been provided with a flexible input/output interface, and a system on a programmable chip (SoPC) has been designed to demonstrate its performance. To the best of our knowledge, this is the first proposal to include block sizes of up to 64×64 and the simplified algorithms for large-size blocks. These two key characteristics introduce a complexity that is not seen in previous proposals, including those for the current HEVC standard and those for the new VVC. In comparison with the latter, the architecture proposed in this paper consumes fewer logic resources and is able to work at higher or similar clock frequencies. It supports HD resolution in real time and UHD@10 fps. These numbers are achieved with a low–end FPGA-based implementation, but with an ASIC or even a high–end FPGA, UHD formats could be processed in real time. In addition, the small footprint of the core 1D transform processor leaves a margin for future parallelization of the transform computation by doubling or quadrupling the number of cores to increase the speed (e.g., $\times 2$ or $\times 4$).

REFERENCES

[1] *ISO/IEC CD 23090-3 Versatile Video Coding*, document N10692, Joint Video Experts Team (JVET) of ITU-T SG 16 WP3 and ISO/IEC JTC 1/SC 29/WG 11, 127th Meeting: Gothenburg, Jul. 2019.

[2] *High Efficiency Video Coding*, document Rec. ITU-T H.265, 2013.

[3] *Requirements for a Future Video Coding Standard v5*, document N17074, MPEG, Joint Video Exploration Team (JVET) of ITU-T VCEG (Q6/16) and ISO/IEC MPEG (JTC 1/SC 29/WG 11), Jul. 2017.

[4] 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

[5] *Versatile Video Coding (Draft 4)*, document JVET-M1001-v7, Joint Video Experts Team (JVET) of ITU-T SG 16 WP3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, Jan. 2019.

[6] M. J. Garrido, C. Sanz, M. Jiménez, and J. M. Menesses, "An FPGA implementation of a flexible architecture for H.263 video coding," *IEEE Trans. Consum. Electron.*, vol. 48, no. 4, pp. 1056–1066, Nov. 2003.

[7] 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)*, Jul. 2013, pp. 1–5.

[8] W. Zhao, T. Onoye, and T. Song, "High-performance multiplierless transform architecture for HEVC," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2013, pp. 1668–1671.

[9] R. Conceição, J. C. de Souza, Jr., R. Jeske, B. Zatt, M. Porto, and L. Agostini, "Low-cost and high-throughput hardware design for the HEVC 16×16 2-D DCT transform," *J. Integr. Circuits Syst.*, vol. 9, no. 1, pp. 25–35, Dec. 2014.

[10] G. Pastuszak, "Hardware architectures for the H.265/HEVC discrete cosine transform," *IET Image Process.*, vol. 9, no. 6, pp. 468–477, Jun. 2015.

[11] 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.

[12] P. Sjovall, V. Viitamaki, J. Vanne, and T. D. Hamalainen, "High-level synthesis implementation of HEVC 2-D DCT/DST on FPGA," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 1547–1551.

[13] M. Masera, G. Masera, and M. Martina, "An area-efficient variable-size fixed-point DCT architecture for HEVC encoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 1, pp. 232–242, Jan. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8576601, doi: 10.1109/TCSVT.2018.2886736.

[14] M. Zheng, J. Zheng, Z. Chen, L. Wu, X. Yang, and N. Ling, "A reconfigurable architecture for discrete cosine transform in video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 3, pp. 810–821, Mar. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8630665, doi: 10.1109/TCSVT.2019.2896294.

[15] 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.

[16] A. Kammoun, W. Hamidouche, F. Belghith, J.-F. Nezan, and N. Masmoudi, "Hardware design and implementation of adaptive multiple transforms for the versatile video coding standard," *IEEE Trans. Consum. Electron.*, vol. 64, no. 4, pp. 424–432, Nov. 2018.

[17] Y. Fan, Y. Zeng, H. Sun, J. Katto, and X. Zeng, "A pipelined 2D transform architecture supporting mixed block sizes for the VVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, early access, Aug. 12, 2020, doi: 10.1109/TCSVT.2019.2934752. [Online]. Available: https://ieeexplore.ieee.org/document/8794833

[18] A. Kammoun, W. Hamidouche, P. Philippe, O. Deforges, F. Belghith, N. Masmoudi, and J.-F. Nezan, "Forward-inverse 2D hardware implementation of approximate transform core for the VVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, early access, Nov. 20, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8907817, doi: 10.1109/TCSVT.2019.2954749.

[19] M. J. Garrido, F. Pescador, M. Chavarrías, P. J. Lobo, and C. Sanz, "A 2-D multiple transform processor for the versatile video coding standard," *IEEE Trans. Consum. Electron.*, vol. 65, no. 3, pp. 274–283, Aug. 2019.

[20] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Mag.*, vol. 6, no. 3, pp. 4–19, Jul. 1989.

[21] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, p. 11, May 2007.

[22] *Versatile Video Coding (Draft 2)*, document JVET-K1001-v7, Joint Video Experts Team (JVET) of ITU-T SG 16 WP3 and ISO/IEC JTC 1/SC 29/WG 11, 11th Meeting, Ljubljana, Jul. 2018.

[23] *Algorithm Description for Versatile Video Coding Test Model 4 (VTM 4)*, document JVET-M1002-v2, Joint Video Experts Team (JVET) of ITU-T SG 16 WP3 and ISO/IEC JTC 1/SC 29/WG 11, 13th Meeting, Marrakech, Jan. 2019.

[24] *VVC VTM Reference Software*. [Online]. Available: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM

[25] Mentor. *ModelSim Functional Verification Tool Web*. Accessed: May 2, 2020. [Online]. Available: https://www.mentor.com/products/fv/modelsim/

[26] *JVET Common Test Conditions and Software Reference Configurations for SDR Video*, document JVET-K1010-v2, Joint Video Experts Team (JVET) of ITU-T SG 16 WP3 and ISO/IEC JTC 1/SC 29/WG 11, 11th Meeting, Ljubljana, Jul. 2018.

[27] Intel. (2016). *Cyclone V Device Overview*. [Online]. Available: https://www.altera.com/documentation/sam1403480548153.html

[28] *SoCkit Development Kit Product Description*. Accessed: May 2, 2020. [Online]. Available: https://www.arrow.com

[29] Platform Designer. (2019). *Intel Quartus Prime Standard Edition User Guide*. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qps-platform-designer.pdf

[30] *Intel FPGA Download Center*. Accessed: May 2, 2020. [Online]. Available: https://www.altera.com/downloads/software.html

[31] Intel. (2017). *Intel Arria 10 Device Overview*. [Online]. Available: https://www.altera.com/documentation/sam1403480274650.html

[32] Xilinx. (2017). *Ultrascale Architecture and Product Data Sheet: Overview*. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf

[33] Intel. (2017). *Intel Stratix 10 GX/SX Device Overview*. [Online]. Available: https://www.altera.com/documentation/joc1442261161666.html

MATÍAS J. GARRIDO was born in Valencia, Spain, in 1965. He received the B.S. and M.S. degrees in telecommunications engineering and the Ph.D. degree from the Universidad Politécnica de Madrid (UPM), Spain, in 1986, 1996, and 2004, respectively.

Since 1987, he has been an Associate Professor with the UPM. He has been with the Electronics and Microelectronics Design Group (GDEM) since its creation, in 1997, and with the Software Technologies and Multimedia Systems for Sustainability (CITSEM) Research Center, since 2013. He has authored 14 journal articles and 38 conference papers. He holds a patent. He has participated in more than 40 Research and Development industry projects. His research interests include electronics digital design, video coding, and digital video broadcasting. He received the Best Paper Award from the 2011 IEEE International Symposium on Consumer Electronics and the IEEE Consumer Electronics Society Chester Sall Award for the third place best paper in the IEEE Transactions on Consumer Electronics, in 2014.

FERNANDO PESCADOR (Senior Member, IEEE) received the Ph.D. degree from the Universidad Politécnica de Madrid (UPM), Spain, in 2011.

He has been an Associate Professor with the UPM, since 1995, and the Head of the Department of Telematics and Electronics Engineering (DTE), since 2012. He has been with the Electronics and Microelectronics Design Group (GDEM), since 1999, and with the Software Technologies and Multimedia Systems for Sustainability (CITSEM) Research Center, since 2013. His research interests are real-time video coding, multiprocessor architectures for video coding, and digital video broadcasting. He received the Best Paper Award from the 2011 IEEE International Symposium on Consumer Electronics and the IEEE Consumer Electronics Society Chester Sall Award for the third place best paper in the IEEE Transactions on Consumer Electronics, in 2014. Since 2017, he has been the Editor-in-Chief of the IEEE Transactions on Consumer Electronics.

MIGUEL CHAVARRÍAS received the Ph.D. degree from the Universidad Politécnica de Madrid (UPM), Spain, in 2017. He has been an Assistant Professor with the Department of Telematics and Electronics Engineering, UPM, since 2019. He has been with the Electronics and Microelectronics Design Group (GDEM), since 2011, and with the Software Technologies and Multimedia Systems for Sustainability (CITSEM) Research Center, since 2012. His current research interests include high-performance embedded systems, machine learning, and new video coding techniques.

PEDRO J. LOBO received the B.S. and M.S. degrees in telecommunications engineering from the Universidad Politécnica de Madrid (UPM), Spain, in 1993 and 2004, respectively, where he is currently pursuing the Ph.D. degree. He has been a member of the Electronics and Microelectronics Design Group (GDEM), since 2001, which was integrated into the Software Technologies and Multimedia Systems for Sustainability (CITSEM) Research Center, in 2012. His research interests are multiprocessor architectures and multiprocessor programming methodologies.

CÉSAR SANZ (Senior Member, IEEE) received the Ph.D. degree in telecommunications engineering from the Universidad Politécnica de Madrid (UPM), in 1998. He has led the Electronics and Microelectronics Design Group (GDEM). He has been involved in Research and Development projects with private companies and public institutions, since 1996. Since 2013, he has been a Researcher with the Software Technologies and Multimedia Systems for Sustainability (CITSEM) Research Center. He was the Dean of the School of Telecommunications Systems and Engineering (UPM), from 2008 to 2017. Since 1985, he has been a Faculty Member of the UPM, where he is currently a Full Professor with the Department of Telematics and Electronics Engineering (DTE). He has more than 100 publications in international journals and conferences, and has participated in more than 80 Research and Development projects. His research interests include electronics digital design applied to video coding and hyperspectral imaging.

PEDRO PAZ was born in A Coruña, Spain, in 1997. He received the B.S. degree in telecommunications engineering from the Universidad Politécnica de Madrid (UPM), Spain, in 2019, where he is currently pursuing the M.S. degree in telecommunications engineering. He has been with the Software Technologies and Multimedia Systems for Sustainability (CITSEM) Research Center, since 2018. His research interests are electronics digital design and video coding.