HIGH-LEVEL SYNTHESIS IMPLEMENTATION OF HEVC 2-D DCT/DST ON FPGA

Panu Sjövall, Vili Viitamäki, Jarno Vanne, Timo D. Hämäläinen

Laboratory of Pervasive Computing
Tampere University of Technology, Finland
{panu.sjovall, vili.viitamaki, jarno.vanne, timo.d.hamalainen}@tut.fi

ABSTRACT

This paper presents the first known high-level synthesis (HLS) implementation of integer discrete cosine transform (DCT) and discrete sine transform (DST) for High Efficiency Video Coding (HEVC). The proposed approach implements these 2-D transforms by two successive 1-D transforms using a well-known row-column and Even-Odd decomposition techniques. Altogether, the proposed architecture is composed of a 4-point DCT/DST unit for the smallest transform blocks (TBs), an 8/16/32-point DCT unit for the other TBs, and a transpose memory for intermediate results. On Arria II FPGA, the low-cost variant of the proposed architecture is able to support encoding of 1080p format at 60 fps and at the cost of 10.0 kALUTs and 216 DSP blocks. The respective figures for the proposed high-speed variant are 2160p at 30 fps with 13.9 kALUTs and 344 DSP blocks. These cost-performance characteristics outperform respective non-HLS approaches on FPGA.

Index Terms— High Efficiency Video Coding (HEVC), Discrete cosine transform (DCT), Discrete sine transform (DST), High-level synthesis (HLS), Catapult-C, Field-programmable gate array (FPGA)

1. INTRODUCTION

The latest video coding standard, *High Efficiency Video Coding* (*HEVC*) [1], has been developed to meet the transmission and storage needs of modern video applications. Compared with its predecessor standard AVC [2], HEVC is able to halve the bit rate for the same subjective quality, but its encoding complexity tends to be at least doubled in practical encoders.

HEVC adopts the conventional hybrid video coding scheme (inter/intra prediction, transform coding, and entropy coding) [3] from the prior MPEG/ITU-T video coding standards. As a new feature, the coding structure of HEVC has been extended from a traditional macroblock concept to an analogous block partitioning scheme that supports *coding tree units* (*CTUs*) of up to 64 × 64 pixels [4].

This paper focuses on HEVC transform coding for which the sizes of *transform blocks* (*TBs*) and associated core transform matrices can be defined as $N \times N$, where $N \in \{4, 8, 16, 32\}$. Extending the sizes of transform matrices from that of AVC to $N > 8$ improves coding gain by around 5-7% but it also introduces the majority of complexity overhead in HEVC transform coding [5].

HEVC specifies *two-dimensional* (*2-D*) integer *discrete sine transform* (*DST*) for intra coded luminance TBs of size 4 × 4 pixels [6] and 2-D integer *discrete cosine transform* (*DCT*) for all other TBs [7]. Both of these 2-D transforms are separable so they can be computed by applying two N-point 1-D transforms first row-wise and then column-wise [5]. This indirect approach is called a row-column decomposition technique and it is typically utilized by software [8]-[9] and hardware implementations [10]-[16] of HEVC DCT/DST.

This work focuses on HEVC DCT/DST implementations on FPGA. Contrary to previous works [12]-[16], our proposal does not use traditional *hardware* (*HW*) *description languages* (*HDLs*), but *High-Level Synthesis* (*HLS*) [17] which is an emerging approach for raising the abstraction level in HW description. HLS is a way of using well-known programming languages such as C and C++ to describe the designs at behavioral level and automatically generating the HDL from it. This way, the code is more readable, design and verification times are shorter, and the design reusability is far better than with handwritten HDL equivalents.

To the best of our knowledge, this is the first paper to describe an HLS implementation for HEVC DCT/DST. The proposed designs include low-cost and high-speed variants of the 8/16/32-point DCT unit for $N \in \{8, 16, 32\}$ and a separate 4-point DCT/DST unit for $N = 4$. They are all implemented on Arria II FPGA using Catapult C [18] HLS tool.

The rest of this paper is organized as follows. Section 2 describes the hardware-oriented DCT/DST algorithm implemented in this work. Section 3 proposes our HLS implementations for low-cost and high-speed DCT/DST computation. In Section 4, the proposed HLS implementations are compared with handcrafted prior-art. Section 5 concludes the paper.

2. 2-D INTEGER DCT/DST ALGORITHMS IN HEVC

In this work, the C implementations of DCT and DST algorithms are obtained from the open source Kvazaar HEVC encoder [8]. Basically, Kvazaar implements the same DCT/DST functionality than *HEVC reference encoder* (*HM*) [9] but the hardware-oriented C source code of Kvazaar provides a better starting point for HLS.

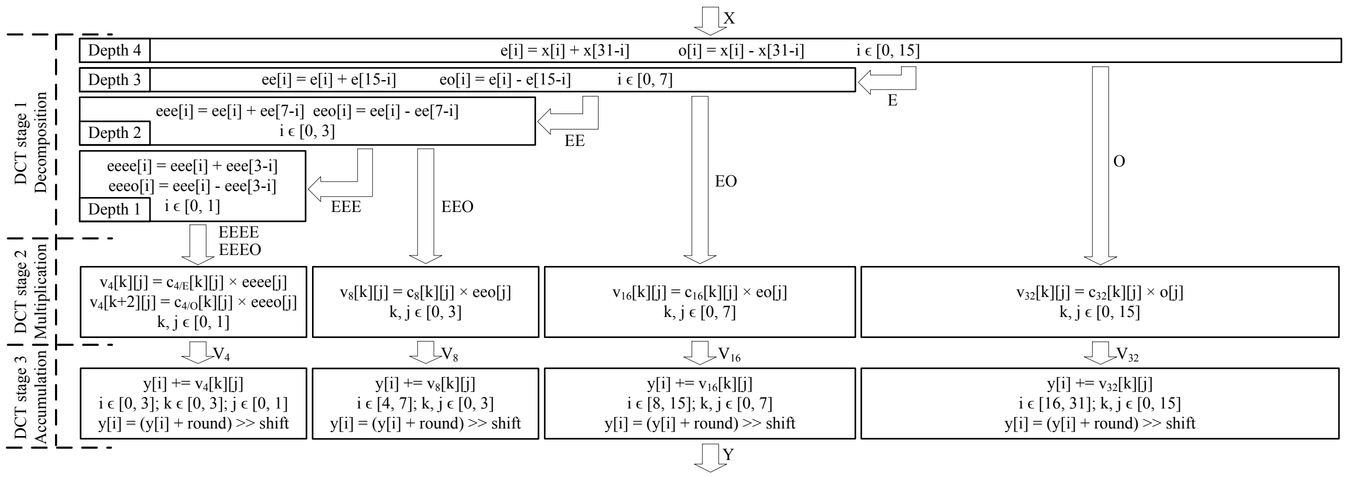
X
DCT stage 1 Decomposition
Depth 4
e[i] = x[i] + x[31-i] o[i] = x[i] - x[31-i] i ϵ [0, 15]
Depth 3
ee[i] = e[i] + e[15-i] eo[i] = e[i] - e[15-i] i ϵ [0, 7]
E
eee[i] = ee[i] + ee[7-i] eeo[i] = ee[i] - ee[7-i]
Depth 2
i ϵ [0, 3]
EE
O
EO
eeee[i] = eee[i] + eee[3-i]
eeeo[i] = eee[i] - eee[3-i]
Depth 1
i ϵ [0, 1]
EEE
EEO
EEEE
EEEO
DCT stage 2 Multiplication
v4[k][j] = c4/E[k][j] × eeee[j]
v4[k+2][j] = c4/O[k][j] × eeeo[j]
k, j ϵ [0, 1]
v8[k][j] = c8[k][j] × eeo[j]
k, j ϵ [0, 3]
v16[k][j] = c16[k][j] × eo[j]
k, j ϵ [0, 7]
v32[k][j] = c32[k][j] × o[j]
k, j ϵ [0, 15]
V4
V8
V16
V32
DCT stage 3 Accumulation
y[i] += v4[k][j]
i ϵ [0, 3]; k ϵ [0, 3]; j ϵ [0, 1]
y[i] = (y[i] + round) >> shift
y[i] += v8[k][j]
i ϵ [4, 7]; k, j ϵ [0, 3]
y[i] = (y[i] + round) >> shift
y[i] += v16[k][j]
i ϵ [8, 15]; k, j ϵ [0, 7]
y[i] = (y[i] + round) >> shift
y[i] += v32[k][j]
i ϵ [16, 31]; k, j ϵ [0, 15]
y[i] = (y[i] + round) >> shift
Y

Figure 1. Even-Odd decomposition algorithm ($N = 32$).

2.1 Even-Odd decomposition algorithm

In HEVC encoder, DCT and DST are used to convert spatial-domain residual blocks into transform-domain coefficient matrices. A well-known row-column algorithm [5] executes these 2-D transforms with separable 1-D transforms in two consecutive stages. An N-point transform is first applied 1) to each row of a residual block of size $N \times N$ to generate an intermediate matrix of size $N \times N$; and then 2) to each column of the intermediate matrix to generate a final transform coefficient matrix of size $N \times N$.

The number of arithmetic operations can be further reduced by implementing these 1-D transforms with Even-Odd decomposition algorithm, a.k.a., Partial Butterfly algorithm [5]. It decomposes an input and core transform matrices of size $N \times N$ into two matrices of size $N/2 \times N/2$ according to even and odd rows/columns, respectively. The core transform matrices for each N ($\mathbf{C}_N$) are specified in [7]. Now, an N-point transform can be computed for even and odd cases separately with two $N/2$-point transforms.

For a residual vector $\mathbf{X} = [x(0), x(1), \ldots, x(N\text{-}1)]$, the even and odd vectors, $\mathbf{E} = [e(0), e(1), \ldots, e(N/2\text{-}1)]$ and $\mathbf{O} = [o(0), o(1), \ldots, o(N/2\text{-}1)]$, can be computed as

$$e(i) = x(i) + x(N-1-i) \tag{1}$$
$$o(i) = x(i) - x(N-1-i) \tag{2}$$

where $i = 0, 1, \ldots, N/2 - 1$. The output vector $\mathbf{Y} = [y(0), y(1), \ldots, y(N\text{-}1)]$ of 1-D transform coefficients could be directly obtained by multiplying the vectors $\mathbf{E}$ and $\mathbf{O}$ by the associated transform matrices at this stage. However, the arithmetic operations can be further reduced by applying decomposition recursively. In this approach, the largest transform matrix also embeds the smaller transform matrices.

Fig. 1 depicts the phases of Even-Odd decomposition for $N = 32$. First, the vectors $\mathbf{E}$ and $\mathbf{O}$ of size 16 are computed according to (1) and (2). The latter is an input to $\mathbf{C}_{32} \times \mathbf{O}$ multiplication and the former is recursively decomposed into smaller even and odd vectors as in (1) and (2), i.e., the vector $\mathbf{E}$ is divided into **EE** and **EO** vectors of size 8. The vector **EO** is multiplied by $\mathbf{C}_{16}$ whereas **EE** is decomposed into **EEE** and **EEO** vectors of size 4, and **EEE** to **EEEE** and **EEEO** vectors of size 2. **EEO** is multiplied by $\mathbf{C}_8$, **EEEO** by $\mathbf{C}_{4/O}$, and **EEEE** by $\mathbf{C}_{4/E}$. The corresponding structure can be used for all N by starting at depth ($\log_2 N$) - 1.

2.2 Proposed hardware-oriented algorithm optimization

In the case of 8-bit video, the residual vector **X** contains 9-bit signed integers for which the original Even-Odd decomposition algorithm produces $9 + (\log_2 N + 6)$ –bit signed results [5] without any truncations. Our motivation is to optimize the algorithm for 18 × 18 multipliers on Arria II FPGA due to which 19-bit ($N = 16$) and 20-bit ($N = 32$) odd and even values are saturated to 18-bit signed values.

The impact of this modification was tested with HM 16.12 using test sequences from HEVC common test conditions (classes A-F) [19] and the average BD-rate overhead is 0.002%. This negligible loss is preferred to using 20 × 20 –bit multipliers that would increment the number of needed DSP blocks fourfold.

3. PROPOSED DCT/DST ARCHITECTURE

The proposed DCT/DST architecture is composed of 1) an 8/16/32-point DCT unit for TBs of size 8 × 8, 16 × 16, and 32 × 32; 2) a separate 4-point DCT/DST unit for TBs of size 4 × 4; and 3) a transpose memory for intermediate results.

3.1 8/16/32-point DCT unit

Fig. 2 shows the block diagram of the 8/16/32-point DCT unit. It contains a control block ($\mathrm{Ctrl}_{8/16/32}$), 3-stage pipeline for DCT computation, and a transpose memory.

A 288-bit input to the $\mathrm{Ctrl}_{8/16/32}$ block is for up to 32 9-bit signed residuals. The $\mathrm{Ctrl}_{8/16/32}$ block sign extends each 9-bit residual to 16 bits and passes them through the 3-stage DCT computation via a 512-bit connection. The mapping of the

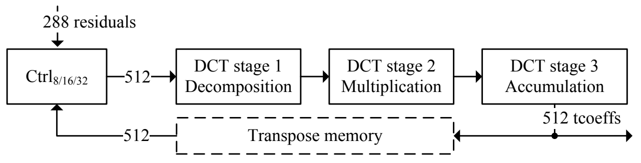

Figure 2. Block diagram of the 8/16/32-point DCT unit.

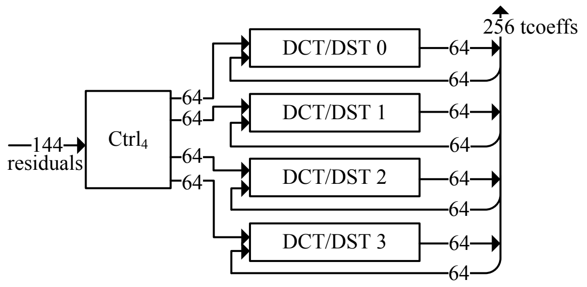

Figure 3. Block diagram of the 4-point DCT/DST unit.

Even-Odd decomposition algorithm to three DCT stages is illustrated in Fig. 1.

The DCT stage 1 performs the recursive Even-Odd decomposition for the 16-bit residuals and computes all even and odd vectors (**E/O, EE/EO, EEE/EEO,** and **EEEE/ EEEO**). It is implemented in C code as a recursive template function which is synthesized by Catapult-C to an adder tree

The DCT stage 2 is for multiplication between transform matrices and odd vectors ($\mathbf{C_{32}} \times \mathbf{O}$, $\mathbf{C_{16}} \times \mathbf{EO}$, $\mathbf{C_8} \times \mathbf{EEO}$, $\mathbf{C_{4/O}} \times \mathbf{EEEO}$, and $\mathbf{C_{4/E}} \times \mathbf{EEEE}$). On FPGA, this functionality is mapped to multipliers of DSP blocks to save logic cells. Catapult-C facilitates instantiation of DSP blocks in C code by providing a library for DSP blocks as C++ templates for different FPGA architectures.

The DCT stage 3 finalizes the 1-D transform by accumulating the individual products of multiplication and scales the coefficients to 16 bits.

The 8/16/32-point DCT unit performs the 2-D DCT in two successive passes and the intermediate data is stored in the transpose memory. The latency for both passes is 3 cycles because of the DCT pipeline. Finally, the 2-D 16-bit *transform coefficients* (*tcoeffs*) are sent via 512-bit output.

This work proposes two alternate 8/16/32-point DCT units with different parallelization strategies:

1) A low-cost unit processes N residuals (one row/column of a TB) in parallel. In this unit, the residuals enter the DCT stage 1 at depth ($\log_2 N$) - 1. In addition, the DCT stages 2 and 3 operate at double clock frequency to be able to compute the largest TB in two phases with the reduced number of DSP blocks. This approach halves the width of the largest multiplier array, without increasing latency.
2) A high-speed unit processes 32 residuals ($32/N$ rows/columns of a TB) in parallel so that a constant data rate with full hardware utilization is achieved. In this unit, the residuals enter the DCT stage 1 at depth 4. In addition, all DCT stages operate at the same frequency and the DCT stage 2 contains a full-width multiplier array.

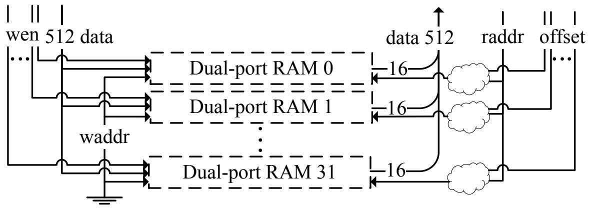

Figure 4. Block diagram of the transpose memory.

3.2 4-point DCT/DST unit

Fig. 3 depicts a 4-point DCT/DST unit that operates in parallel with the 8/16/32-point DCT unit. A 144-bit input to the Ctrl_4 block accepts a single 4 × 4 residual block at a time. The 9-bit residuals are sign extended to 16-bits and passed row-wise to the respective four DCT/DST blocks. The intermediate matrix is ready in one cycle after which it is sent back to the same DCT/DST blocks by picking the intermediate values from the registers in a transposed order. After these two passes, the unit outputs 16 16-bit coeffs.

A separate 4-point DCT/DST unit increases the occupied resources on FPGA. However, this overhead is compensated by better load balancing since the share of 4 × 4 TBs is relatively high compared to the other TBs.

3.3 Transpose memory

Fig. 4 depicts the structure of the transpose memory used in the 8/16/32-point DCT unit. On FPGA, it is made of 32 dual-port on-chip memory modules without registers. Each memory module has a 512-bit write (N coefficients) and a 16-bit (1 coefficient) read port. The structure supports block transpose for $N \in \{8, 16, 32\}$.

The memory utilization of the low-cost 8/16/32-point DCT unit depends on N. The intermediate matrix is written to the memory modules row by row and the module number is incremented from 0 to N accordingly. The right module is identified by a one-hot write enable (wen) signal. A matrix is read from the memory column by column by accessing a single coefficient per each module and incrementing the *read address* (*raddr*) by one after each read (from 0 to N).

The high-speed 8/16/32-point DCT unit utilizes the whole memory for each N. To enable simultaneous reading of $32/N$ columns of the matrix without any access conflicts, the same rows are written to $(32/N)^2$ modules. Let us use $N = 8$ as an example. The first four rows are written in the modules 0-3, 8-11, 16-19, 24-27 after which the last four rows are written to the remaining modules respectively. Eight columns can now be read in two cycles by using raddr and offset.

4. PERFORMANCE ANALYSIS

Table 1 reports the cost-performance characteristics of the proposed and the most competitive prior-art FPGA implementations. The comparison is simplified by deriving

Table 1. Comparison of the proposed and related work.

Architecture	Transform	N	FPGA	Logic cells	DSPs	Freq.	Mtcoeffs/s	Cells/(Mtcoeffs/s)
Proposed (low-cost)	2-D DCT	8/16/32	Arria II	4 263 ALUTs	216	100 MHz	515	8.3
Proposed (high-speed)	2-D DCT	8/16/32	Arria II	8 114 ALUTs	344	160 MHz	1 224	6.6
Proposed (4×4)	2-D DCT/DST	4	Arria II	5 775 ALUTs	0	160 MHz	1 280	4.5
Jeske et al. [12]	1-D DCT	16	Stratix III	5 168 ALUTs	0	88 MHz	*701	7.4
Darji et al. [13]	1-D DCT	16	Spartan 3E	3 419 LEs	0	48 MHz	*384	**7.1
Zhao et al. [14]	2-D DCT	4/8/16/32	Cyclone IV	40 541 LEs	0	125 MHz	238	**136.3
Arayacheeppreecha et al. [15]	1-D DCT	4/8/16/32	Spartan 3A	15 677 LEs	77	205 MHz	*820	**15.3
Pastuszak et al. [16]	2-D DCT	8/16/32	Arria II	6 928 ALUTs	256	100 MHz	512	13.5
Pastuszak et al. [16]	2-D DCT/DST	4	Arria II	4 256 ALUTs	0	100 MHz	400	10.6

*Scaled sample rate (divided by two) **1.25 × LE = ALUT

Table 2. Logic cells (DSP blocks replaced by logic).

Architecture	Cells w/o DSPs	Cells/(Mtcoeffs/s)
Proposed (low-cost)	25 698	36.3
Proposed (high-speed)	38 829	23.0
Arayacheeppreecha et al. [15]	18 124	22.1
Pastuszak et al. [16]	29 744	42.3

normalized performance and cost figures for the architectures: sample rate as million tcoeffs processed per second (Mtcoeffs/s) and performance-cost ratio as logic cells per sample rate (cells/(Mtcoeffs/s)). The works in [12], [13], [15] only implement the 1-D transform. For fair comparison, their sample rates have been scaled (divided by two) to correspond to those of the 2-D transform architectures.

4.1 Proposed architecture

Table 1 tabulates the results for the proposed low-cost and high-speed variants of 8/16/32-point DCT units and for the 4-point DCT/DST unit separately. Altogether, the combined resource usage of our proposal is (4.2 + 5.8) kALUTs = 10.0 kALUTs and 216 DSP blocks in the low-cost case and (8.1 + 5.8) kALUTs = 13.9 kALUTs and 344 DSP blocks in the high-speed case. The low-cost approach uses 28% less ALUTs and 37% less DSP blocks than the high-speed one which has, on the other hand, almost 2.4× better sample rate.

The sample rate of our low-cost solution is adequate for transform coding of 4:2:0 1080p (1920 × 1080) video at 60 fps. The speed is computed for the worst case where the DCT/DST is needed once for all TBs in a CTU. It is also assumed here that there are always residual blocks available for the architecture. A practical HEVC intra/inter encoder can meet these conditions by coding successive CTUs in parallel without rate-distortion optimization. Respectively, the high-speed case is for 4:2:0 2160p (3840 × 2160) video at 30 fps.

On FPGA, the functionality of the proposed design was validated as a part of Kvazaar HEVC intra encoder.

4.2 Comparison with prior-art

The architecture proposed by Jeske et al. [12] is limited to $N = 16$ whereas the work of Darji et al. [13] supports all TBs but results are given for $N = 16$ only. Hence, the features of these two works are not directly comparable with our proposal. Furthermore, Zhao et al. [14] support all TB sizes but with non-competitive cost-performance figures.

The remaining approaches make also use of DSP blocks whose impact on the overall logic cell count is taken into account in Table 2. For the proposed designs, the total cell count is obtained by synthesizing them without the DSP blocks. The same cost per DSP block (72.5 ALUTs) is used when estimating the respective count for the related works.

The fastest prior-art solution is presented by Arayacheeppreecha et al. [15] whose overall cell count is also the smallest. However, including the missing DST unit and transpose memory would add overhead in their cost-performance figures. Furthermore, the cell counts of our both architectures are smaller if DSP blocks are available. Our high-speed architecture is also almost 1.5× faster.

Pastuszak et al. [16] present an approach similar to ours by implementing separate units for $N = 4$ and $N \in \{8, 16, 32\}$. However, our low-cost architecture is slightly faster and consumes still 14% less resources. Our high speed approach needs around 31% more resources but is around 2.4× faster.

5. CONCLUSIONS

This paper presented the first known HLS implementation for HEVC 2-D DCT/DST on FPGA. The presented architecture implements a hardware-oriented even-odd decomposition algorithm whose C code is synthesized to HDL with HLS. A low-cost variant of the architecture is able to support 1080p video up to 60 fps and a high-speed variant 2160p video up to 30 fps. HLS reduces design and verification times over traditional handwritten approaches. This work shows that these benefits do not come at the cost of implementation overhead but our HLS solution outperforms the prior-art approaches in terms of performance and cost.

6. ACKNOWLEDGMENT

This work was supported in part by the European Celtic-Plus Project 4KREPROSYS and the Academy of Finland (decision no. 301820).

7. REFERENCES

[1] *High Efficiency Video Coding*, document ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), ITU-T and ISO/IEC, Apr. 2013.

[2] *Advanced Video Coding for Generic Audiovisual Services*, document ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), ITU-T and ISO/IEC, Mar. 2009.

[3] G. J. Sullivan, J. R. 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.

[4] I. K. Kim, J. Min, T. Lee, W. J. Han, and J. Park, "Block partitioning structure in the HEVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1697-1706, Dec. 2012.

[5] M. Budagavi, A. Fuldseth, G. Bjøntegaard, V. Sze, and M. Sadafale, "Core transform design in the High Efficiency Video Coding (HEVC) standard," *IEEE J. Select. Topics Signal Process.*, vol. 7, no. 6, pp. 1029-1041, Dec. 2013.

[6] A. Saxena and F. C. Fernandes, "CE7: Mode-dependent DCT/DST without 4×4 full matrix multiplication for intra prediction," *Document JCTVC-E125*, Geneva, Switzerland, Mar. 2011.

[7] A. Fuldseth, G. Bjøntegaard, M. Budagavi, and V. Sze "Core transform design for HEVC," *Document JCTVC-G495*, Geneva, Switzerland, Nov. 2011.

[8] *Kvazaar HEVC encoder* [Online]. Available: https://github.com/ultravideo/kvazaar

[9] *Joint Collaborative Team on Video Coding Reference Software, ver. HM 16.3* [Online]. Available: http://hevc.hhi.fraunhofer.de/

[10] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim, and C. Yeo, "Efficient integer DCT architectures for HEVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 1, pp. 168-178, Jan. 2014.

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

[12] R. Jeske, J. C. de Souza, G. Wrege, R. Conceição, M. Grellert, J. Mattos, and L. Agostini, "Low cost and high throughput multiplierless design of a 16 point 1-D DCT of the new HEVC video coding standard," *in Proc. Southern Conf. Programmable Logic*, Bento Goncalves, Spain, Mar. 2012.

[13] A. D. Darji and R. P. Makwana, "High-performance multiplierless DCT architecture for HEVC," *in Proc. Int. Symp. VLSI Design and Test*, Ahmedabad, India, Jun. 2015.

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

[15] P. Arayacheeppreecha, S. Pumrin, and B. Supmonchai, "Flexible input transform architecture for HEVC encoder on FPGA," *in Proc. Int. Conf. Electrical Engineering/ Electronics, Computer, Telecommunications and Information Tech.*, Hua Hin, Thailand, Jun. 2015.

[16] G. Pastuszak and A. Abramowski, "Algorithm and architecture design of the H.265/HEVC intra encoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 210-222, Jan. 2016.

[17] P. Coussy, D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Des. Test. Comput.*, vol. 26, no. 4, pp. 8-17, Jul.-Aug. 2009.

[18] *Catapult: Product Family Overview* [Online]. Available: http://calypto.com/en/products/catapult/overview

[19] F. Bossen, "Common test conditions and software reference configurations," *Document JCTVC-J1100*, Stockholm, Sweden, Jul. 2012