# High-level synthesized 2-D IDCT/IDST implementation for HEVC codecs on FPGA

# High-level Synthesized 2-D IDCT/IDST Implementation for HEVC Codecs on FPGA

Vili Viitamäki, Panu Sjövall, Jarno Vanne, Timo D. Hämäläinen
Laboratory of Pervasive Computing
Tampere University of Technology
Tampere, Finland
{vili.viitamaki, panu.sjovall, jarno.vanne, timo.d.hamalainen}@tut.fi

*Abstract*— **This paper presents efficient inverse discrete cosine transform (IDCT) and inverse discrete sine transform (IDST) implementations for High Efficiency Video Coding (HEVC). The proposal makes use of high-level synthesis (HLS) to implement a complete HEVC 2-D IDCT/IDST architecture directly from the C code of a well-known Even-Odd decomposition algorithm. The final architecture includes a 4-point IDCT/IDST unit for the smallest transform blocks (TB), an 8/16/32-point IDCT unit for the other TBs, and a transpose memory for intermediate results. On Arria II FPGA, it supports real-time (60 fps) HEVC decoding of up to 2160p format with 12.4 kALUTs and 344 DSP blocks. Compared with the other existing HLS approach, the proposed solution is almost 5 times faster and is able to utilize available FPGA resources better.**

***Keywords— High Efficiency Video Coding (HEVC); Inverse discrete cosine transform (DCT); Inverse discrete sine transform (DST); High-level synthesis (HLS); Field-programmable gate array (FPGA)***

## I. Introduction

*High Efficiency Video Coding* (*HEVC/H.265*) [1] is the newest international video coding standard published as twin text by ITU, ISO, and IEC as ITU-T H.265 | ISO/IEC 23008-2. It has been developed to address the increasing transmission and storage needs of modern video applications. HEVC is able to reduce the bit rate by almost 40% over the current mainstream standard AVC [2] for the same objective quality, but the respective encoding and decoding complexities tend to be at least 1.5 times higher [3]. The main reason for HEVC coding gain and complexity increase is a new HEVC coding structure that extends a traditional macroblock concept to an analogous block partitioning scheme with *coding tree units* (*CTUs*) of up to 64 × 64 pixels.

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

HEVC standard specifies *two-dimensional* (*2-D*) integer *inverse discrete sine transform* (*IDST*) for intra coded luminance TBs of size 4 × 4 pixels and 2-D integer *inverse discrete cosine transform* (*IDCT*) for all other TBs [1]. Both of these separable 2-D transforms can be computed by two successive $N$-point 1-D transforms, first column-wise and then row-wise [5]. This indirect approach is called row-column decomposition and it is a widely used technique in software [6]-[7] and hardware implementations [8]-[11] of HEVC IDCT/IDST.

This work deals with *field-programmable gate array* (*FPGA*) implementations of HEVC IDCT/IDST architectures. However, our design is not written in traditional *hardware* (*HW*) *description languages* (*HDLs*), but the abstraction level in HW description is raised by *High-Level Synthesis* (*HLS*) [12]. HLS tools support well-known programming languages such as C and C++ in design description from which they can automatically generate the HDL. This approach makes the code more readable, shortens design and verification times, and increases the design reusability over those of handwritten HDL equivalents.

In our recent work [13], we proposed to use HLS for HEVC DCT/DST. This work utilizes the same HLS flow than in [13] and applies it to IDCT and IDST algorithms. The created 2-D IDCT/IDST architecture includes an 8/16/32-point IDCT unit for $N \in \{8, 16, 32\}$, a separate 4-point IDCT/IDST unit for $N = 4$, and a transpose memory for intermediate results. The architecture is implemented on Arria II FPGA using Catapult C [14] HLS tool. For the time being, only a single HLS implementation has been presented for HEVC IDCT [8] and none for HEVC IDST. Thus, this paper presents the first HLS implementation for a complete HEVC 2-D IDCT/IDST.

The remainder of the paper is organized as follows. Section 2 describes the adopted hardware-oriented IDCT and IDST algorithms. Section 3 proposes our HLS implementation for HEVC 2-D IDCT/IDST. In Section 4, performance characteristics of our proposal are reported and compared with the prior-art. Section 5 concludes the paper.

## II. 2-D Integer IDCT/IDST Algorithms in HEVC

In this work, the C implementations of IDCT and IDST algorithms are taken from the open-source Kvazaar HEVC encoder [6]. Basically, Kvazaar implements the same IDCT/IDST functionality than *HEVC reference encoder* (*HM*) [7] but the hardware-oriented C source code of Kvazaar provides a better starting point for HLS.

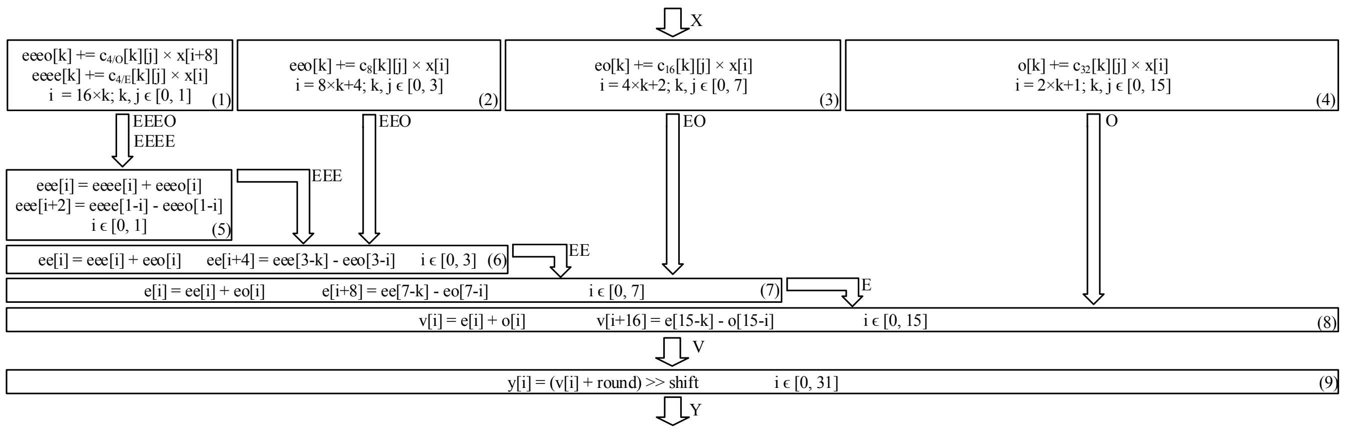
X
eeeo[k] += c4/O[k][j] × x[i+8]
eeee[k] += c4/E[k][j] × x[i]
i = 16×k; k, j ϵ [0, 1] (1)
eeo[k] += c8[k][j] × x[i]
i = 8×k+4; k, j ϵ [0, 3] (2)
eo[k] += c16[k][j] × x[i]
i = 4×k+2; k, j ϵ [0, 7] (3)
o[k] += c32[k][j] × x[i]
i = 2×k+1; k, j ϵ [0, 15] (4)
EEEO
EEEE
EEO
EO
O
eee[i] = eeee[i] + eeeo[i]
eee[i+2] = eeee[1-i] - eeeo[1-i]
i ϵ [0, 1] (5)
EEE
ee[i] = eee[i] + eeo[i] ee[i+4] = eee[3-k] - eeo[3-i] i ϵ [0, 3] (6)
EE
e[i] = ee[i] + eo[i] e[i+8] = ee[7-k] - eo[7-i] i ϵ [0, 7] (7)
E
v[i] = e[i] + o[i] v[i+16] = e[15-k] - o[15-i] i ϵ [0, 15] (8)
V
y[i] = (v[i] + round) >> shift i ϵ [0, 31] (9)
Y

Fig. 1. Even-Odd decomposition algorithm for IDCT ($N = 32$).

## *A. Even-Odd decomposition algorithm*

In HEVC codec, IDCT and IDST are used to convert transform-domain coefficient matrices back into spatial-domain residual blocks. A well-known row-column algorithm [4] executes these 2-D inverse transforms with separable 1-D inverse transforms in two consecutive stages. An $N$-point inverse transform is first applied 1) to each column of a transform-domain coefficient matrix of size $N \times N$ to generate an intermediate matrix of size $N \times N$; and then 2) to each row of the intermediate matrix to generate a final spatial-domain residual block of size $N \times N$.

The number of arithmetic operations can be further reduced by implementing these 1-D inverse 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 columns/rows, respectively. The core transform matrices for each $N$ ($C_N$) are specified in [5]. Now, an $N$-point inverse transform can be computed with two $N/2$-point transforms so that the matrix multiplication is done separately for even and odd cases after which the result is yielded with basic add and subtract operations. The respective decomposition can be applied recursively down to $N = 4$ to reduce arithmetic operations further. In this approach, the largest transform matrix also embeds the smaller transform matrices.

## *B. Example: 32-point IDCT*

Fig. 1 depicts the Even-Odd decomposition of the 1-D inverse transform for $N = 32$. The transform coefficients of the input vector $\mathbf{Y} = [y(0), y(1), \ldots, y(31)]$ are recursively decomposed into five parts which can be multiplied in parallel by transform matrices as shown by (1), (2), (3), and (4) in Fig. 1. That is, $\{y(8), y(24)\}$ are multiplied by $\mathbf{C_{4/O}}$, $\{y(0), y(16)\}$ by $\mathbf{C_{4/E}}$, $\{y(4), y(12), y(20), y(28)\}$ by $\mathbf{C_8}$, $\{y(2), y(6), \ldots y(30)\}$ by $\mathbf{C_{16}}$, and $\{y(1), y(3), \ldots y(31)\}$ by $\mathbf{C_{32}}$ to yield 2-point vectors **EEEO** and **EEEE,** a 4-point vector **EEO**, an 8-point vector **EO**, and a 16-point vector **O**, respectively.

In the next stage, a 4-point vector **EEE** is computed from the vectors **EEEE** and **EEEO** with add and subtract operations as in (5). Correspondingly, an 8-point vector **EE** is derived from the vectors **EEE** and **EEO** as in (6), a 16-point vector **E** from the vectors **EE** and **EO** as in (7), and finally a 32-point vector **V** from the vectors **E** and **O** as in (8). The residual output vector **X** = $[x(0), x(1), \ldots, x(31)]$ is then formed by scaling the vector **V**. The scaling factor depends on the transform stage (column or row) and on the video bit depth (8 bits in our case) [5].

## III. Proposed IDCT/IDST Architecture

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

## *A. 8/16/32-point IDCT unit*

Fig. 2 depicts a block diagram of the 8/16/32-point IDCT unit. It can be divided into three parts: 1) a control block ($\mathrm{Ctrl}_{8/16/32}$); 2) a 3-state IDCT computation pipeline; and 3) a transpose memory.

The $\mathrm{Ctrl}_{8/16/32}$ block has a 512-bit input for 32 16-bit signed *transform coefficients* (tcoeffs). It extends the coefficients with configuration bits and passes them to the IDCT computation. The configuration bits are used to define the block size and the scaling factor. After the first pass, the $\mathrm{Ctrl}_{8/16/32}$ block extends the transposed results with new configuration bits and sends them back to the IDCT computation.

The IDCT stage 1 performs multiplications between the transform matrices ($\mathbf{C}_{32}$, $\mathbf{C}_{16}$, $\mathbf{C}_{8}$, $\mathbf{C}_{4/O}$, and $\mathbf{C}_{4/E}$) and input ($\mathbf{Y}$). To save logic cells on an FPGA, multiplications are mapped to DSP blocks. 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 IDCT stage 2 includes three addition/subtraction levels to compose the final even vector (**E**) from the decomposed even and odd vectors (**EEEO**, **EEEE**, **EEO**, **EEE**, **EO**, and **EE**). Fig. 1 depicts this for $N = 32$ in (5) – (7). For $N < 32$, all these levels are used to calculate $32/N$ rows or columns at a time. For example, for $N = 8$, four rows or columns are decomposed into the **EEEO**, **EEEE**, **EEO**, **EO**, and **O** vectors in parallel to take full advantage of the available adding and subtracting levels.

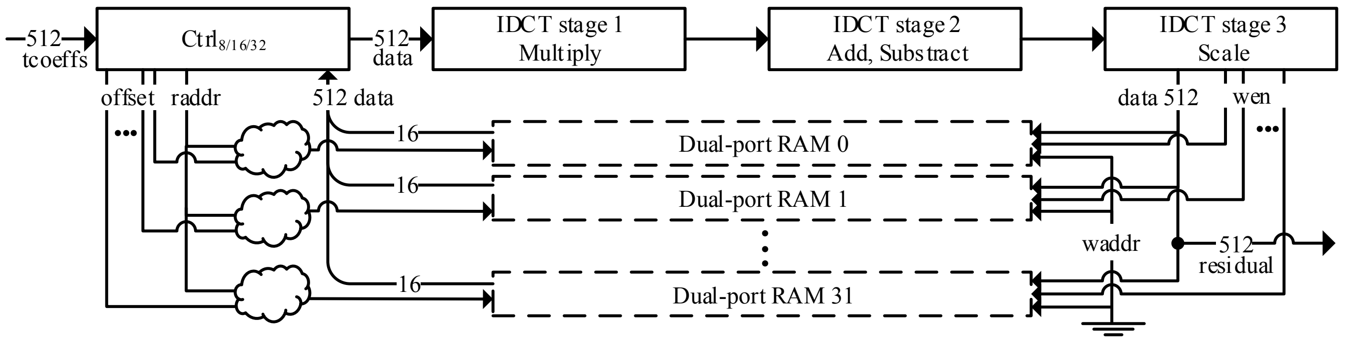

Fig. 2. Block diagram of the pipelined 8/16/32-point IDCT unit and a transpose memory.

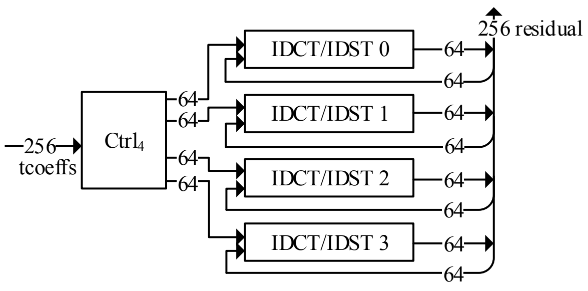

Fig. 3. Block diagram of the separate 4-point 2-D IDCT/IDST unit.

The IDCT stage 3 finalizes the 1-D transform by combining the **E** and **O** vectors and by scaling the final result (**V**) to 16-bit signed residuals (**X**). Before outputting the final residual vector, it is permuted back to its original order.

The 8/16/32-point IDCT unit performs the 2-D IDCT in two successive passes and the intermediate data is stored in the transpose memory. 32 coefficients (32/*N* rows or columns of a TB) are processed in parallel to ensure a more constant hardware utilization. The latency for both passes is three clock cycles. Finally, the 32 16-bit residuals are sent via the 512-bit output. The same output is connected to the transpose memory.

## B. *Transpose memory*

Fig. 2 also shows the structure of the transpose memory used with the 8/16/32-point IDCT unit. On FPGA, it is made of 32 dual-port on-chip memory modules without registers. Each memory module has a 512-bit write (32 coefficients) and a 16-bit (1 coefficient) read port. The structure supports a block transpose for $N \in \{8, 16, 32\}$.

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

## C. *4-point IDCT/IDST unit*

Fig. 3 depicts a 4-point IDCT/IDST unit that can operate in parallel with the 8/16/32-point IDCT unit. The 256-bit input to the $\text{Ctrl}_4$ block accepts one 4 × 4 coefficient block at a time. The 16 16-bit coefficients are passed column-wise to the respective four IDCT/IDST blocks. The intermediate matrix is ready in one clock cycle after which it is sent back to the same IDCT/IDST blocks by picking the intermediate values from the registers in a transposed order. After these two passes, the unit outputs 16 16-bit residuals.

A separate 4-point IDCT/IDST 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.

## IV. Performance Analysis

Table 1 reports the cost-performance characteristics of the proposed and the most competitive prior-art FPGA implementations.

## A. *Proposed architecture*

Table 1 tabulates results for the proposed 8/16/32-point IDCT unit and for the 4-point IDCT/IDST unit separately. Altogether, the combined resource usage of our proposal is (6.9 + 5.6) kALUTs = 12.4 kALUTs and 344 DSP blocks. The total cell count rises to 49.9 kALUTs if the DSP blocks are not used. It is assumed here that one DSP block (DSP18×18) equals to 109 ALUTs. This relationship was obtained by synthesizing an equivalent DSP functionality using only logic cells.

The proposed design is capable of supporting 4:2:0 Ultra HD (3840 × 2160) video decoding at 68 *frames per second* (*fps*) and 4:2:0 Ultra HD video encoding at 35 fps. The reported speeds are for the worst case scenarios: a decoded bit stream is assumed to contain TBs of size 8 × 8 pixels only and an encoder is assumed to encode all TBs in a CTU when searching for the best block partitioning.

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

## B. *Comparison with prior-art*

Kalali et al. [8] present the first HSL implementations for HEVC IDCT. Altogether, they proposed three implementations

TABLE 1. COMPARISON OF THE PROPOSED AND RELATED IDCT/IDST ARCHITECTURES ON FPGA

| Architecture | HLS | Transform | N | FPGA | Logic cells | DSPs | Cells w/o DSPs | Freq. | Speed (worst case) |
|---|---|---|---|---|---|---|---|---|---|
| Proposed | YES | 2-D IDCT | 8/16/32 | Arria II | 6 859 ALUTs | 344 | 44355 ALUTs | 150 | 2160@68fps |
| Proposed (4x4) | YES | 2-D IDCT/IDST | 4 | Arria II | 5 559 ALUTs | 0 | 5559 ALUTs | 150 | 2160@96fps |
| Kalali et al. [8] | YES | 2-D IDCT | 4/8/16/32 | Virtex 6 | 13 669 4-LUTs | 0 | *13669 ALUTs | 110 | 1080@55fps |
| Pastuszak et al. [9] | NO | 2-D IDCT | 8/16/32 | Arria II | 3 079 ALUTs | 512 | 58887 ALUTs | 200 | 2160@64fps |
| Pastuszak et al. [9] | NO | 2-D IDCT/IDST | 4 | Arria II | 3 554 ALUTs | 0 | 3554 ALUTs | 100 | 2160@32fps |
| Kalali et al. [10] | NO | 2-D IDCT/IDST | 4/8/16/32 | Virtex 6 | 38 790 4-LUTs | 0 | *38790 ALUTs | 150 | 2160@48fps |
| Conceição et al. [11] | NO | 2-D IDCT | 4/8/16/32 | Stratix V | 17 340 ALMs | 0 | **34680 ALUTs | 63 | 2160@20fps |

*4-LUT = ALUT **ALM = 2×ALUT

done with three different HLS tools. The best performance characteristics were obtained with MATLAB Simulink HDL Coder. It supports IDCT for all TB sizes but is missing the IDST unit. In addition, it is only capable of decoding 1080p video at 55 fps. Our solution is 3.7 times larger when DSPs are normalized to logic cells, but also 5.0 times faster.

Pastuszak et al. [9] introduce an approach similar to ours by implementing separate units for $N = 4$ and $N \in \{8, 16, 32\}$. However, our approach is slightly faster and consumes around 25% less resources when DSPs are normalized to logic cells. Our 4-point IDCT/IDST unit, compared with their counterpart, consumes 1.6 times more resources, but is three times faster.

Kalali et al. [10] also propose a handwritten Verilog RTL implementation, that is notably faster but also larger than their HLS implementation. Compared with our solution, the cell count is much higher as it does not utilize DSP blocks. With DSPs normalized to logic cells, our architecture is 1.3 times larger, but also 1.4 times faster.

Conceição et al. [11] present an implementation that supports IDCT for all TB sizes but not IDST. Their implementation does not utilize DSP blocks. When DSPs are normalized to logic cells, our proposal requires 1.4 times more logic cells, but is also 3.4 times faster.

## V. CONCLUSION

This paper presented the first complete HLS design for HEVC 2-D IDCT/IDST on FPGA. The proposed design implements a hardware-oriented Even-Odd decomposition algorithm whose C code is directly synthesized to HDL with HLS. The architecture supports 2160p video decoding up to 68 fps at a cost of 12.4 kALUTs and 344 DSP blocks. It is almost 5 times faster than the existing HLS implementation for IDCT and consumes less logic cells on an FPGA due to efficient mapping of computation to the available DSPs.

Compared with traditional approaches, HLS design techniques are known to increase design productivity, code readability, and design reusability. The presented results also show that our HLS solution is very competitive with the existing non-HLS IDCT/IDST solutions in terms of performance and cost. Hence, the main conclusion of this paper is that the manifold benefits of HLS do not come at the cost of implementation overhead.

## ACKNOWLEDGMENT

This work was supported in part by the European Celtic-Plus Project 4KREPROSYS and Academy of Finland (decision number 301820).

## 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] J. Vanne, M. Viitanen, T. D. Hämäläinen, and A. Hallapuro, "Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12 pp. 1885-1898, Dec. 2012.

[4] 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.

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

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

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

[8] E. Kalali and I. Hamzaoglu, "FPGA implementations of HEVC inverse DCT using high-level synthesis," *in Proc. Conf. Design and Architectures for Signal and Image Processing*, Krakow, Poland, Sep. 2015.

[9] 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.

[10] E. Kalali, E. Ozcan, O. M. Yalcinkaya, and I. Hamzaoglu, "A low energy HEVC inverse transform hardware," *IEEE Trans. Consumer Electron.*, vol. 60, no. 4, pp. 754-761, Nov. 2014.

[11] R. Conceição, J. C. de Souza, R. Jeske, M. Porto, B. Zatt, and L. Agostini, "Power efficient and high troughtput multi-size IDCT targeting UHD HEVC decoders," *in Proc. IEEE Int. Symp. Circuits Syst.*, Melbourne, Australia, Jun. 2014.

[12] 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.

[13] P. Sjövall, V. Viitamäki, J. Vanne, and T. D. Hämäläinen, "High-level synthesis implementation of HEVC 2-D DCT/DST on FPGA," *in Proc. IEEE Int. Conf. Acoustics, Speech, Signal Process.*, New Orleans, Louisiana, USA, Mar. 2017.

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