Kvazaar 4K HEVC Intra Encoder on FPGA Accelerated Airframe Server

Panu Sjövall, Vili Viitamäki, Arto Oinonen, Jarno Vanne, Timo D. Hämäläinen
Laboratory of Pervasive Computing
Tampere University of Technology
Tampere, Finland

Ari Kulmala
Accelerator SoC R&D
Nokia
Tampere, Finland

Abstract— **This paper presents a real-time Kvazaar HEVC intra encoder for 4K Ultra HD video streaming. The encoder is implemented on Nokia AirFrame Cloud Server featuring a 2.4 GHz dual 14-core Intel Xeon processor and Arria 10 PCI Express FPGA accelerator card. In our HW/SW partitioning scheme, the data-intensive Kvazaar coding tools including intra prediction, DCT, inverse DCT, quantization, and inverse quantization are offloaded to Arria 10 whereas CABAC coding and other control-intensive coding tools are executed on Xeon processors. Arria 10 has enough capacity for up to two instances of our intra coding accelerator. The results show that the proposed system is able to encode 4K video at 30 fps with a single intra coding accelerator and at 40 fps with two accelerators. The respective speed-up factors are 1.6 and 2.1 over the pure Xeon implementation. To the best of our knowledge, this is the first work dealing with HEVC intra encoder partitioned between CPU and FPGA. It achieves the same coding speed as HEVC intra encoders on ASIC and it is at least 4 times faster than existing HEVC intra encoders on FPGA.**

Keywords— High Efficiency Video Coding (HEVC), Kvazaar, Intra coding, Field-programmable gate array (FPGA), PCI Express (PCIe), Real-time

I. INTRODUCTION

Internet video traffic is forecast to grow threefold in five years from that of 2015 and video is estimated to account for 82% of all global consumer Internet traffic by 2020 [1]. This growth comes from new end users and multimedia applications entering the market but also from higher video dimensions, resolutions, frame rates, and color depths. Despite the fast progress of network capacities, the holistic increase of video volume makes more efficient video compression inevitable.

High Efficiency Video Coding (*HEVC/H.265*) [2], [3] is the latest international video coding standard developed to meet video storage and transmission needs of modern multimedia applications. HEVC is published as twin text by ITU, ISO, and IEC as ITU-T H.265 | ISO/IEC 23008-2. This paper addresses *all-intra* (*AI*) coding configuration [4] of HEVC Main Profile. HEVC is shown to improve intra coding efficiency by 23% over that of the preceding state-of-the-art standard AVC/H.264 [5] for the same objective quality but at a cost of over 3 × encoding complexity [6]. Therefore, implementing a real-time HEVC intra encoder with a reasonable coding efficiency, implementation cost, and power budget requires efficient encoder optimizations and powerful computing platforms.

The complexity of *software* (*SW*) HEVC encoders can be primarily tackled by two techniques: multithreading through data-level parallelism [7], [8] and *single instruction multiple data* (*SIMD*) optimizations [9], [10]. Further speedup and lower power dissipation can be obtained by offloading the compute-intensive coding tools to *hardware (HW)* accelerators or implementing the entire HEVC encoder on HW [11]-[14]. Existing HW encoders include both *application specific integrated circuit (ASIC)* [11], [12] and *field-programmable gate array* (*FPGA*) implementations [12]-[14].

The main motivation of this work was to optimize our Kvazaar HEVC intra encoder [15], [16], for real-time 4K *Ultra High Definition* (*UHD*) coding on Nokia AirFrame Cloud Server. Airframe includes a 2.4 GHz dual 14-core Xeon processor an Arria 10 *PCI Express* (*PCIe*) FPGA accelerator card. Airframe rackmount server is easily expandable to large server farms and an accompanied FPGA brings lots of additional computing power for a single server. Cloud video encoding on servers like AirFrame has gained a lot of traction in the recent years because of the advent of cloud gaming, telco clouds, and edge computation in general.

Our previous works have already investigated parallelization of Kvazaar intra encoder on multi-core processors [8] and SIMD optimizations of Kvazaar [10], so the main emphasis here is on 1) HW/SW partitioning of Kvazaar; and 2) HW acceleration of Kvazaar on FPGA. The HW-oriented C source code of Kvazaar enables more straightforward HW/SW partitioning than other eligible open-source HEVC encoders [17], [18]. Kvazaar code is also written at a suitable abstraction level for *high-level synthesis* (*HLS*) [19] that enables automatic *hardware description language* (*HDL*) generation from C. In this work, our intra coding accelerator is implemented using Catapult C [20] HSL tool. Through HLS, the code is more readable, design and verification times are shorter, and the design reusability is far better than with handwritten HDL equivalents.

The rest of this paper is organized as follows. Section 2 gives an overview of the adopted CPU + FPGA platform and the proposed SW/HW partitioning of Kvazaar on it. Section 3 describes the Kvazaar functionality on CPU, Section 4 the communication between CPU and FPGA, and Section 5 the implemented intra coding accelerator on FPGA. In Section 5, the speedup of HW acceleration is benchmarked against SW only encoding using 2160p (3840 × 2160) and 1080p (1920 × 1080) test videos. Section 6 concludes the paper.

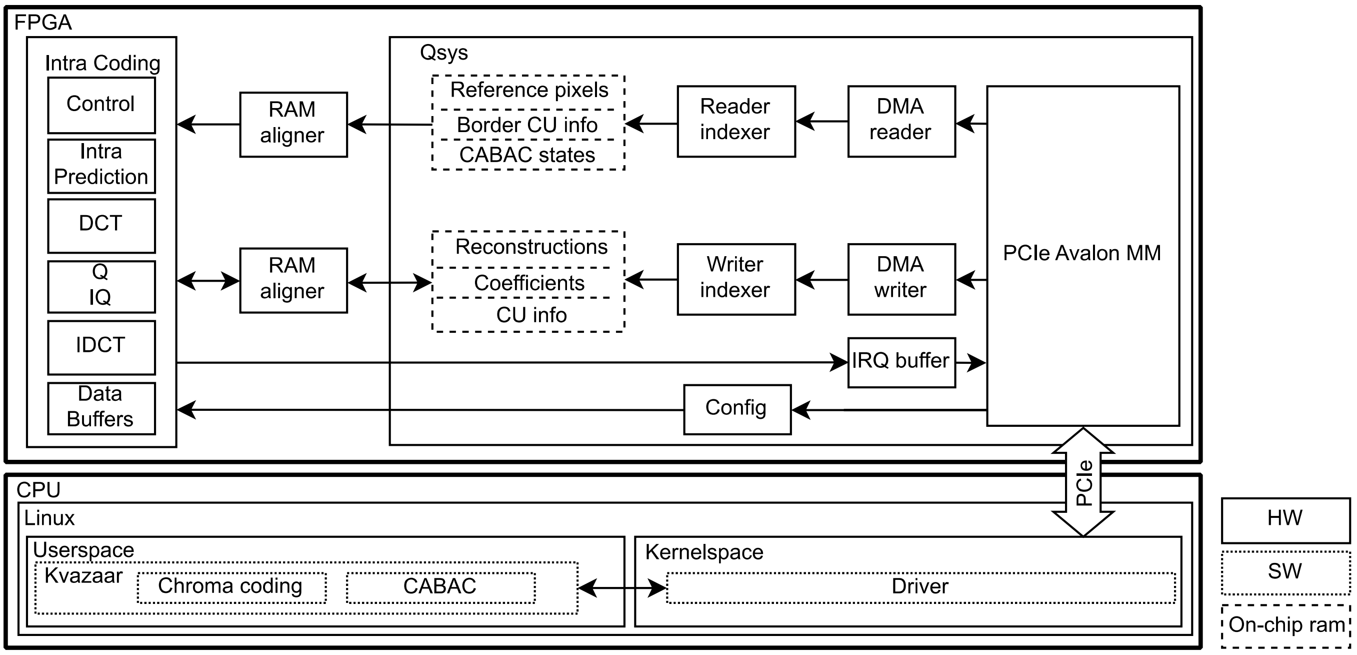

Fig. 1. Block diagram of the proposed encoder system with a single intra coding accelerator.

II. System Overview

Fig. 1 shows the block diagram of the underlying CPU + FPGA platform on which Kvazaar encoder is implemented. The backbone of the system is Nokia AirFrame server [21] with two Xeon E5-2680 v4 processors and 256 GB of memory. Arria 10 FPGA accelerator card is connected to the CPU via a PCIe bus. The operating system is CentOS 6.8.

A. *Kvazaar HEVC Intra Encoder*

Kvazaar [15] is an academic cross-platform open-source HEVC encoder. It contains all integral coding tools of HEVC and its modular code facilitates parallelization on multi and manycore processors as well as algorithm acceleration on HW.

Kvazaar intra encoder supports HEVC Main profile for 8-bit 4:2:0 video with ten presets out of which *fast* and *medium* presets are used in this work for their favorable cost-performance characteristics. Table I tabulates the settings of these presets. The medium preset is utilized without *rate-distortion optimized quantization* (*RDOQ*). Kvazaar implements a basic HEVC block partitioning structure in which the pictures are partitioned into *coding tree units* (*CTUs*) of size 64 × 64. CTUs can be optionally divided into four equal-sized blocks and the division can be recursively continued until the maximum hierarchical depth of the HEVC quadtree is reached. The leaf nodes of the quadtree are called *coding units* (*CUs*).

The proposed implementation of Kvazaar offers two schemes for parallel CTU coding: 1) *Wavefront Parallel Processing* (*WPP*); and 2) picture-level parallel processing. These schemes can be enabled concurrently.

B. *Kvazaar Partitioning*

Kvazaar is run on the platform under AI coding configuration in which the main coding tools are *intra prediction* (*IP*), *discrete cosine transform* (*DCT*), *quantization* (*Q*), *inverse Q* (*IQ*), *inverse DCT* (*IDCT*), and *context-adaptive binary arithmetic coding* (*CABAC*). In this work, the most computationally intensive coding tools including IP, DCT, Q, IQ, and IDCT are implemented with HLS and synthesized to FPGA. CABAC and other control-intensive coding tools such a control for WPP and for picture-level parallelism are executed on CPU. In addition, CPU takes care of raw input video reading, chrominance coding, and outputting the encoded bit stream.

Arria 10 FPGA has enough resources for two instances of our intra coding accelerator including the needed peripherals and on-chip memories. Mapping a major share of CTU coding to FPGA could be utilized to decrease power dissipation through lower CPU usage. However, we are aiming at the maximum HEVC coding speed, so encoding parallelism is increased by coding additional CTUs entirely in SW with released CPU resources.

TABLE I. Implemented Coding tools of Kvazaar Intra Encoder

Feature	Fast	Medium (wo RDOQ)
Profile	Main	Main
Internal bit depth	8	8
Color format	4:2:0	4:2:0
Coding mode	Intra	Intra
Coding units	16×16, 8×8	64×64, 32×32, 16×16, 8×8
Prediction units	16×16, 8×8	32×32, 16×16, 8×8
Transform units	16×16, 8×8	32×32, 16×16, 8×8
IP modes	35 (DC, planar, 33 angular)	35 (DC, planar, 33 angular)
Intra Search	Full	Full
Transform	Integer DCT	Integer DCT
Mode decision	Sum of absolute difference	Sum of absolute difference
Parallelization	WPP, Picture level	WPP, Picture level
SAO	Enabled	Enabled
Signhide	Disabled	Disabled
Rate Control	Disabled	Disabled
RDO	Disabled	Disabled
RDOQ	Disabled	Disabled

III. Functionality on Xeon

On Xeon processors, Kvazaar is run in the user space and the Linux driver in the kernel space. The Linux driver is used for the CPU-PCIe-FPGA interfacing. It is accessed by Kvazaar via ioctl, write, and read system calls.

A. User Space: Kvazaar

Kvazaar parallelization is implemented using a CPU thread pool with a single CTU as the smallest work unit. The CTUs are put in a queue in the order they would be processed in a single threaded case, and the free worker threads start processing the first CTU with no dependencies. In this work, a CTU search function of Kvazaar is modified to offload a majority of coding tasks to the HW accelerator on FPGA. Offloading is performed through system calls to the kernel driver. A worker thread sends its CTU data to the HW accelerator and sleeps until the accelerator notifies that the CTU coding on FPGA is completed. Then, the worker thread performs chrominance coding and CABAC coding for the CTU according to the results from FPGA. The threads not being able to be served by FPGA are encoded on CPU. Intra coding on FPGA has the highest priority for new CTUs and the CPU is used only when the pipeline of the HW accelerator is full.

B. Kernel Space: Driver

Fig. 2 shows the sequence chart of system calls between Kvazaar and the kernel driver. At first, Kvazaar calls the ioctl function to request a free index from the driver, which returns a nonnegative index if the HW accelerator can accept a new CTU for encoding. The driver uses semaphores initialized to the maximum CTU count supported by the accelerator. In the next step, Kvazaar calls the write function to copy all necessary data of the processed CTU to FPGA. The data being sent to FPGA is aligned in consecutive virtual memory addresses in the user space and in consecutive physical memory addresses in the kernel space. A worker thread uses the read function to request intra coding results for the CTU of interest. The thread will sleep in the kernel space until the CTU of interest has finished and the accelerator sends an interrupt signal. Both the write and read system calls return the amount of bytes (*length*) read or written successfully.

IV. Interface between XEON and FPGA

Fig. 1 shows the FPGA interface made of the Avalon-MM Hard IP for PCIe, separate *Direct Memory Access* (*DMA*) blocks for reading and writing, and the on-chip memories of the intra coding accelerator.

A. PCIe Interface

The CPU communicates with the FPGA via the PCIe bus. The PCIe IP is configured to PCIe generation 3 × 4 with 128-bit interface and 250 MHz application clock. The IRQ Buffer block is used for generating the interrupt through the PCIe IP. The IRQ buffer detects the rising edges of the CTU ready signals from the intra coding accelerator and buffers them. The interrupt is delayed until the CPU acknowledges the previous interrupt. This is done in order to prevent two interrupts from happening in consecutive cycles, which is a limitation of the PCIe IP.

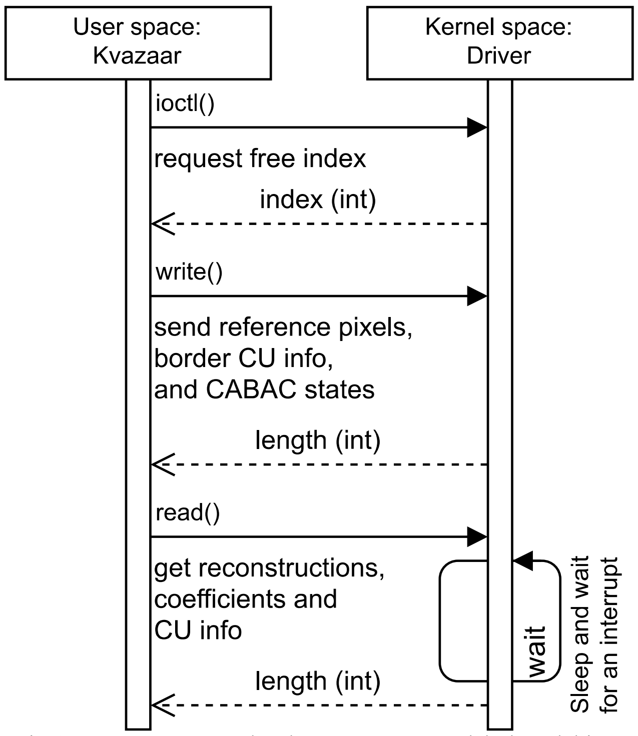
User space: Kvazaar
Kernel space: Driver
ioctl()
request free index
index (int)
write()
send reference pixels, border CU info, and CABAC states
length (int)
read()
get reconstructions, coefficients and CU info
length (int)
wait
Sleep and wait for an interrupt

Fig. 2. Message sequence chart between Kvazaar and the kernel driver.

B. DMAs

A single intra coding accelerator consists of two DMA blocks. One DMA block is used for reading data from the shared memory and the other one is for writing data to the shared memory. This separation allows the DMA blocks to better utilize the bandwidth of the PCIe interface to the CPU memory. Our tests showed that this scheme increases the data transfer speed by 54% compared with sequential reading and writing.

The accelerator utilizes Reader and Writer indexer blocks for address translation. The blocks are configured with the CTU index before the DMA transfers are started. The DMA blocks read and write to consecutive memory addresses, but the memory structure of the on-chip memories on FPGA requires non-consecutive addresses depending on the index of the CTU.

C. On-Chip Memories

For each CTU, the HW accelerator requires the corresponding reference pixels, information about the CU borders (reconstructed pixels and modes), as well as the CTU CABAC states. The reference pixels are used to calculate *Sum of Absolute Difference* (*SAD*) values for intra mode selection and *Sum of Squared Differences* (*SSD*) values for final mode decision. CTU border pixels are used to calculate intra predictions for the CUs on the CTU borders whereas border modes are used as candidate modes when selecting the best intra mode. The CABAC states are used for *mode decision* (*MD*).

There are also on-chip memories for the final reconstructed pixels and quantized coefficients, which are flushed from the intermediate buffer. The CU info contains the resulting modes and depths from the accelerator. The RAM aligners are used as wrappers with the on-chip memories because the PCIe interface and the HW accelerator have different memory access widths.

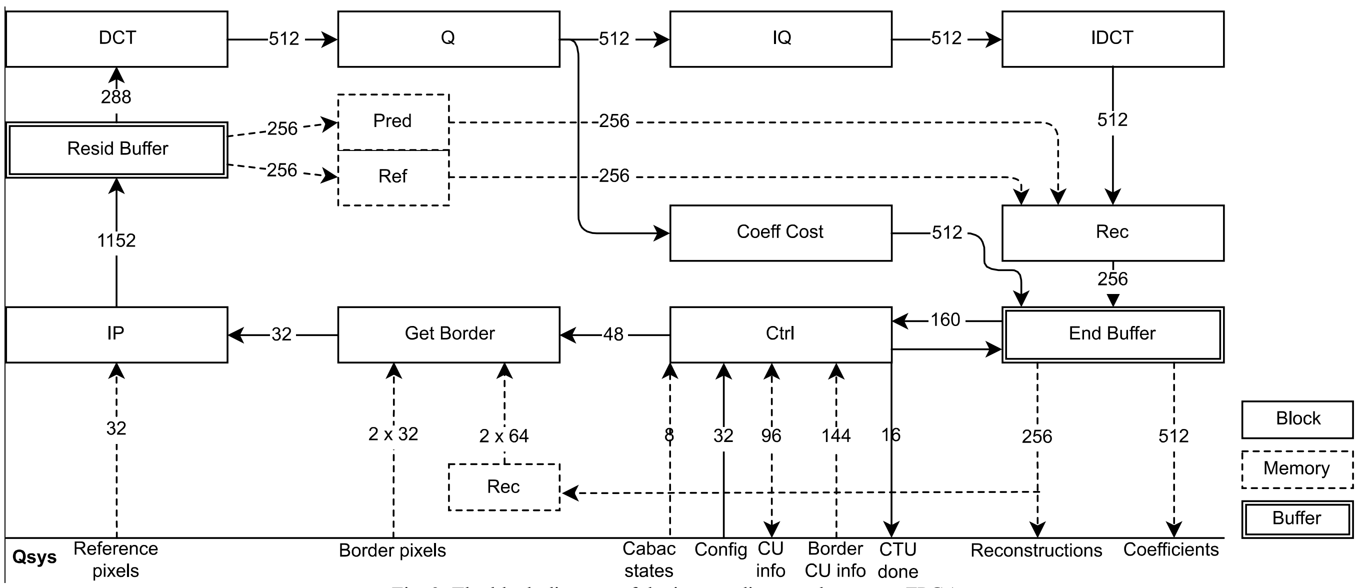

Fig. 3. The block diagram of the intra coding accelerator on FPGA.

V. Intra Coding on FPGA

Fig. 3 shows a block diagram of the intra coding accelerator. It consists of the following units implemented with HLS.

A. Intra Coding Control (Ctrl)

The Ctrl unit receives instructions from the CPU. It is split into Initialization, Scheduler, Start, and End blocks.

The Initialization block generates a full instruction set for processing a CTU. The instruction set contains operations for calculating IPs with different configurations and MD operations for selecting a CTU configuration. The HW generates the instruction set for each CTU individually.

The Scheduler block is responsible for the CTU parallelization in the HW accelerator. It loads the valid instructions for each CTU and selects the ones with the highest priority for processing. The priority for each instruction is determined so that there will be a minimal delay on the intra coding pipeline.

The Start block processes instructions from the Scheduler in order. It initializes the IP configuration for the CU according to the input instruction and sends CU information to the Get Border unit. It also notifies the CPU about finishing the CTU search if it receives the instruction for terminating the search.

The End block is at the end of the intra coding pipeline, from where it receives the search results. The results include the selected intra mode, SSD, and the estimated coding cost of the CU coefficients. The End block uses the results to calculate the MD cost for every CU configuration and stores them to the internal memory. With the MD instructions, the End block determines the best CU partitioning for the CTU according to stored cost values and flushes the pixels and the coefficients for that configuration from the buffer.

B. Get Reference Border (Get Border)

The *Get Border* unit reads the reconstructed reference pixels and sends them to the IP unit. It operates according to the configuration data consisting of CU block size and coordinates of the CU in the CTU. The coordinates are utilized when reading reconstructed pixels, i.e., either the neighboring column on the left to the CU or the neighboring row above the CU. The reconstructed pixels are read from either the CTU memory or the CTU borders memory, depending on the location of the CU within the CTU.

C. Intra Prediction (IP)

The IP unit is composed of an IP control block, SAD block, and the following IP blocks that predict 35 IP modes in parallel: DC IP (mode 0), Planar IP (mode 1), Positive Angular IP (modes 2-9, 27-34), Negative Angular IP (modes 11-25), and Zero Angular IP (modes 10, 26). All these IP blocks predict four pixels at a time, i.e., 32×32 block is predicted in 256 cycles, 16×16 block in 64 cycles, etc. The IP unit used here is an improved version of our previous IP accelerator presented in [22].

The IP unit operates according to the configuration data consisting of the CU block size and the corresponding reference pixels from the Get Border block. The IP control block filters reference pixels if needed and configures all the IP blocks that perform the prediction algorithm for a proper CU size, and all angular IP blocks for the right angle. This configurability makes the IP blocks more generic and easy to instantiate.

All angular IP blocks calculate the predicted pixels in original order, so additional transposing is not needed. The blocks also have a common control. Furthermore, IP modes with an equal distance to the horizontal (mode 10) and vertical (mode 26) modes are computed by the same IP block. For example, modes 2 and 34 are calculated in the same Positive Angular IP block since 10 - 2 = 34 - 26. This allows a reduced number of intra prediction IPs and saves area.

TABLE II. COMPARISON OF THE PROPOSED AND RELATED HEVC INTRA ENCODERS

Architecture	Technology	Board	HW Lang.	Frequency	Resolution	Coding mode	Cells	DSPs
Proposed	CPU + FPGA	Arria 10	C/C++	125 MHz	2160p@30fps	Intra	308k ALUTs	862
Zhu et al. [11]	ASIC	-	Verilog	357 MHz	1080p@44fps	Intra	2296k gates	-
Pastuszak et al. [12]	ASIC	-	VHDL	200/400 MHz	2160p@30fps	Intra	1086k gates	-
Pastuszak et al. [12]	FPGA	Arria II	VHDL	100/200 MHz	1080p@30fps	Intra	93k ALUTs	481
Atapattu et al. [13]	FPGA	Zyng ZC706	Verilog	140 MHz	1080p@30fps	Intra	84k LUTs	34
Miyazawa et al. [14]	FPGA	Custom 3xFPGA	N.A.	N.A.	1080p@60fps	Intra/Inter	N.A.	-

TABLE III. CODING SPEED WITH 2160P VIDEO (FAST PRESET)

Sequence	**No acceleration**	**1 accelerator**		**2 accelerators**	
(2160p)	**Speed (fps)**	**Speed (fps)**	**Speedup**	**Speed (fps)**	**Speedup**
Beauty	20	31	1.6×	40	2.0×
Bosphorus	21	32	1.6×	42	2.1×
HoneyBee	19	31	1.6×	41	2.1×
Jockey	22	35	1.6×	47	2.1×
ReadySetGo	20	31	1.6×	41	2.0×
ShakeNDry	17	26	1.6×	35	2.1×
YachtRide	19	30	1.6×	40	2.1×
Average	20	31	1.6×	41	2.1×

TABLE IV. CODING SPEED WITH 1080P VIDEO (MEDIUM PRESET)

Sequence	**No acceleration**	**1 accelerator**		**2 accelerators**	
(1080p)	**Speed (fps)**	**Speed (fps)**	**Speedup**	**Speed (fps)**	**Speedup**
Beauty	63	102	1.6×	136	2.2×
Bosphorus	51	82	1.6×	110	2.2×
HoneyBee	46	73	1.6×	98	2.2×
Jockey	52	84	1.6×	113	2.2×
ReadySetGo	51	79	1.6×	107	2.1×
ShakeNDry	44	70	1.6×	94	2.2×
YachtRide	49	78	1.6×	105	2.1×
Average	51	81	1.6×	109	2.2×

The SAD block reads the reference pixels of the processed CU from the corresponding on-chip memory. It also receives predicted pixels from the IP blocks and calculates the SAD in parallel for all modes. The SAD block sends all the predicted pixels and the reference pixels to a buffer, four pixel at a time. After the SAD calculation is done and the best mode is determined, SAD block notifies the buffer. The buffer recalculates the residual vector and reference pixels for the best mode and sends them to the DCT unit.

D. Discrete Cosine Transform (DCT)

The DCT unit equals the high-speed variant of our 8/16/32-point DCT unit presented in-depth in [23]. The unit performs the 2-D DCT in two successive passes and the intermediate data is stored in a transpose memory. It can process 32 residuals in parallel so that a constant data rate with full HW utilization is achieved. The latency for both passes is three cycles because of the DCT pipeline. After the 2-D transform, the 16-bit *transform coefficients* (*tcoeffs*) are passed to the Q unit.

E. Quantization (Q)

The Q unit operates according to the configuration data consisting of the block size and the quantization parameter. The unit receives one or several columns of tcoeffs from the DCT unit per write, depending on the block size. Then it performs the quantization to all tcoeffs in parallel and outputs the quantized tcoeffs to the IQ unit and the Coeff Cost unit.

F. Inverse Quantization (IQ)

The IQ unit operates according to the configuration data consisting of the block size and the quantization parameter. The unit receives one or several columns of quantized tcoeffs from the Q unit per write, depending on the block size. Then it performs the inverse quantization to all quantized tcoeffs in parallel and outputs them to the IDCT unit.

G. Inverse Discrete Cosine Transform (IDCT)

The IDCT unit equals the 8/16/32-point IDCT unit presented by us in-depth in [24]. The unit performs the 2-D IDCT in two successive passes and the intermediate data is stored in a transpose memory. The IDCT unit can process 32 tcoeffs in parallel to ensure a more constant HW utilization. The latency for both passes is three clock cycles. After the 2-D inverse transform, the 16-bit residuals are passed to the Rec unit.

H. Coefficient Cost (Coeff Cost)

The Coeff Cost unit operates according to the configuration data consisting of the block size. The unit reads all the columns of the quantized tcoeffs, which are transposed back to the original order. After the transpose, the block calculates the approximate coding cost for the CU coefficients, processing 32 coefficients in parallel.

I. Reconstruction (Rec)

The Rec module reads the reconstructed residuals from the IDCT unit and the original and predicted pixels from the memory in parallel. It generates the final reconstructed pixels and calculates the SSD for the processed CU. The reconstructed pixels are stored to memory through a buffer in order to store the right CU in the CTU sized memory.

VI. EXPERIMENTAL RESULTS

Table II tabulates the characteristics of the proposed and other existing HEVC intra encoders on ASIC and FPGA. The real-time coding speed of the ASIC-based HEVC intra encoder in [11] is limited to 1080p video. The HEVC intra encoder in [12] supports real-time 2160p video encoding on ASIC but the respective FPGA implementation is limited to 1080p resolution. Similarly, the FPGA-based HEVC intra encoder in [13] is restricted to 1080p video coding. The intra/inter HEVC encoder in [14] is able to encode 1080p at 60 fps with a custom board of three separate FPGA chips. Higher resolutions are also supported but not without increasing the number of boards. To sum up, our proposal is the only FPGA-based implementation that supports real-time HEVC encoding up to 2160p resolution with a single board.

Table III and Table IV report HEVC coding speed of the proposed system with fast and medium presets (Table I) using 2160p and 1080p test videos, respectively. The 8-bit 4:2:0 2160p test sequences were taken from Ultra Video Group test sequence set [25] and scaled down to 1080p resolution for our tests. In both cases, the results are given for our system with 0, 1, and 2 intra coding accelerators.

The average results with the fast preset show that our implementation is able to encode 2160p video at 20 fps without HW acceleration, at 30 fps with a single accelerator, and at 40 fps with two accelerators. The speedups obtained with one and two accelerators are 1.6× and 2.1× over the pure SW implementation, respectively. In 2160p case, real-time coding speed (30 fps) requires at least one accelerator. The coding speeds of 1080p test videos are approximately 2.6 times as high as those of 2160p sequences even though a more complex medium preset (without RDOQ) is used. Hence, real-time coding speed is attainable without any HW acceleration in 1080p case. Our implementation would also be able to encode three separate real-time 1080p sequences in parallel.

VII. Conclusion

This paper presented the first known 4K HEVC intra encoder partitioned between a processor and a PCIe-connected FPGA. The encoder functionality is based on C source code of Kvazaar HEVC intra encoder and HLS was used to implement the most compute-intensive Kvazaar coding tools on FPGA. For the first time, HLS was applied to the whole intra coding chain from intra prediction to block reconstruction. HLS is generally known to reduce design and verification times over a traditional HDL workflow. This work shows that these benefits do not come at a cost of coding performance.

The proposed encoder implementation was prototyped on Nokia AirFrame Cloud Server composed of dual 14-core Intel Xeon processor and Arria 10 FPGA. On AirFrame, our solution is able to encode one 2160p video in real-time. The introduced HW acceleration roughly doubles coding speed over that of a pure SW encoder. Further performance boost could be easily obtained by inserting another FPGA card in the available slot in the server and replacing the current FPGAs with larger ones. This way, up to four times as high coding speed is anticipated.

Acknowledgment

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

References

[1] Cisco, *Cisco Visual Networking Index: Forecast and Methodology*, 2015-2020, Jun. 2016.

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

[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, Dec. 2012, pp. 1649-1668.

[4] J. Lainema, F. Bossen, W. J. Han, J. Min, and K. Ugur, "Intra coding of the HEVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1792-1801.

[5] *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.

[6] 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, Dec. 2012, pp. 1885-1898.

[7] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel scalability and efficiency of HEVC parallelization approaches," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1827-1838.

[8] A. Koivula, M. Viitanen, J. Vanne, T. D. Hämäläinen, and L. Fasnacht, "Parallelization of Kvazaar HEVC intra encoder for multi-core processors," *in Proc. IEEE Workshop Signal Process. Syst.*, Hangzhou, China, Oct. 2015, pp. 1-6.

[9] Y. J. Ahn, T. J. Hwang, D. G. Sim, and W. J. Han, "Implementation of fast HEVC encoder based on SIMD and data-level parallelism," *EURASIP J. Image Video Process.*, vol. 16, Dec. 2014, pp. 1-19.

[10] A. Lemmetti, A. Koivula, M. Viitanen, J. Vanne, and T. D. Hämäläinen, "AVX2-optimized Kvazaar HEVC intra encoder," *in Proc. IEEE Int. Conf. Image Processing*, Phoenix, Arizona, USA, Sep. 2016, pp. 549-553.

[11] J. Zhu, Z. Liu, D. Wang, Q. Han, and Y. Song, "HDTV1080p HEVC Intra encoder with source texture based CU/PU mode pre-decision," *2014 19th Asia and South Pacific Design Automation Conf.*, Singapore, 2014.

[12] 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, Jan. 2016, pp. 210-222.

[13] S. Atapattu, N. Liyanage, N. Menuka, I. Perera, and A. Pasqual, "Real time all intra HEVC HD encoder on FPGA," *in Proc. IEEE Int. Conf. on Application-specific Syst., Architectures and Processors*, London, Jul. 2016, pp. 191-195.

[14] K. Miyazawa, H. Sakate, S. Sekiguchi, N. Motoyama, Y. Sugito, K. Iguchi, A. Ichigaya, and S. Sakaida, "Real-time hardware implementation of HEVC video encoder for 1080p HD video," *in Proc. Picture Coding Symposium*, San Jose, California, USA, Dec. 2013, pp. 225-228.

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

[16] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: open-source HEVC/H.265 encoder," *in Proc. ACM Int. Conf. Multimedia*, Amsterdam, The Netherlands, Oct. 2016, pp. 1179-1182.

[17] *x265* [Online]. Available: http://x265.org/

[18] *Turing codec* [Online]. Available: https://github.com/bbc/turingcodec

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

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

[21] *AirFrame data center solution* [Online]. Available: https://networks.nokia.com/solutions/airframe-data-center-solution

[22] P. Sjövall, J. Virtanen, J. Vanne, and T. D. Hämäläinen, "High-level synthesis design flow for HEVC intra encoder on SoC-FPGA," *in Proc. Euromicro Symp. Digit. Syst. Des.,* Funchal, Madeira, Portugal, Aug. 2015, pp. 49-56.

[23] 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, pp. 1547-1551.

[24] V. Viitamäki, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "High-level synthesized 2-D IDCT/IDST implementation for HEVC codecs on FPGA," *in Proc. IEEE Int. Symp. Circuits Syst.*, Baltimore, Maryland, USA, May 2017.

[25] *Test Sequences* [Online]. Available: http://ultravideo.cs.tut.fi/#testsequences