HLS and FPGA-Powered Streaming Video Encoder Accelerator for IoTs Edge Computing

Cuong Pham-Quoc [1,2]

[1] Department of Computer Engineering, Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology (HCMUT), Ho Chi Minh City, Vietnam
[2] Department of Computer Engineering, Vietnam National University-Ho Chi Minh City (VNU-HCM), Thu Duc, Ho Chi Minh City, Vietnam
Email: cuongpham@hcmut.edu.vn

***Abstract*—In recent years, Internet of Thing (IoT) applications with video processing deployed on edge computing platforms have been widely exploited for many areas, such as surveillance, object monitoring, or check-in/check-out systems. While H.264 video format is widely used for most modern cameras due to its efficiency, the computing power of edge computing platforms usually needs to be higher to process H.264 videos in acceptable intervals. This paper proposes an approach based on the high-level synthesis technique to accelerate video encoding by Field Programmable Gate Array (FPGA) platforms for edge computing applications. The H.264 encoding, chosen as our case study, is profiled to locate the computational-intensive functions to accelerate by FPGA. We use the high-level synthesis technique with optimization approaches, including loop unrolling, loop pipeline, function pipeline, and array partition, to generate the accelerator core. The core is then implemented with the hardware accelerator computing paradigm combining a host processor and the hardware accelerator cores for processing H.264 encoding. The approach is tested with an edge computing FPGA Ultra96-v2 board to validate the proposed approach and evaluate the performance. Experimental results show that we achieve speed-ups by up to 14.9× compared to an Advanced RISC Machine (ARM) quad-core processor. In terms of power consumption, our system requires 4.208 W.**

***Keywords*—Field Programmable Gate Array (FPGA), hardware accelerator, H.264 encoder, edge computing, Internet of Things (IoTs)**

I. Introduction

With the rapid advancement of technology and the internet, videos are being utilized across various domains of the internet, encompassing entertainment as well as supervision and monitoring. Nevertheless, one of the foremost challenges encountered in network streaming video applications pertains to the bandwidth of the communication network and the expenses associated with transferring recorded videos from cameras to cloud storage or data centers. A promising strategy to address this predicament involves the utilization of edge computing platforms to process or pre-process videos, transmitting only essential frames of detected objects from videos to servers or cloud services. Edge computing platforms provide several advantages when compared to conventional cloud computing solutions [1]. By processing data in proximity to its source, edge computing diminishes latency, thereby reducing the time required for data to travel to and from the cloud. This capability proves particularly advantageous for applications that necessitate real-time processing, such as industrial automation, self-driving cars, and object detection.

While this promising strategy holds the potential to address the challenges associated with high-traffic and cost-ineffective video data transfer, there are certain obstacles that necessitate careful consideration, namely, the limited computing performance, energy capacity, and storage of edge computing platforms [2]. Consequently, in this study, we propose an approach to enhance the speed of video encoders on edge computing platforms based on Field-Programmable Gate Arrays (FPGAs), thereby surmounting the performance limitations of edge computing while ensuring energy efficiency. Our proposed approach initially involves profiling the encoders to identify computationally intensive functions that are suitable for acceleration using FPGA fabrics. Subsequently, we employ High-Level Synthesis (HLS) tools to leverage various optimizations, such as loop unrolling or pipelining, to generate the most optimized accelerator core. Finally, the system is constructed based on the hardware accelerator architecture [3], aiming to capitalize on the advantages offered by both general-purpose processors and FPGA-based accelerator cores.

Meanwhile, the H.264 video compression standard, also referred to as MPEG-4 Part 10 (Moving Picture Experts Group), has recently gained widespread recognition and utilization in video streaming and video-based object monitoring due to its exceptional optimization. When compared to other standards, H.264 offers numerous advantages, including high-quality video compression that enables efficient storage and transmission of video content without compromising image quality. Consequently, we have chosen the H.264 video encoder as our focal point for validating the proposed approach. To assess the efficacy of the accelerator, we have developed an experimental

Manuscript received on April 3, 2023, revised May 16, 2023; accepted June 9, 2023; published January 9, 2024.

system utilizing the Ultra96-v2 FPGA edge computing platforms and have compared its performance to that of an embedded Advanced RISC Machine (ARM) processor. Our experimental results, obtained using HD (High Definition) videos from the Derf test media collection [4], demonstrate that our system achieves significant speed-ups of up to 14.9× when compared to a 4-core ARM processor operating at 1.2 GHz.

Our paper offers three significant contributions, which can be summarized as follows:

1. We propose an HLS-based approach to enhance the acceleration of streaming video encoders on FPGA devices specifically designed for edge computing platforms.
2. We present a case study utilizing the well-established H.264 video coding format and the Ultra96-v2 FPGA board.
3. We showcase experimental results using HD videos, which can serve as a reference for future studies.

The remainder of the paper is structured as follows: In Section II, we provide background information and discuss related work. Our proposed design approach is introduced in Section III. The implementation of the H.264 encoder as our case study is presented in Section IV. Section V delves into the experiments conducted with HD videos on the Ultra96-v2 FPGA board. Finally, in Section VI, we conclude our paper.

II. Background and Related Work

This section presents the background for our study and related work in the literature.

A. Video Codec & H.264 Encoder

In the video compression process, there are two essential components: the encoder and the decoder. The encoder's role is to encapsulate the original video streams into compressed bitstreams, significantly reducing the data size for efficient transfer and storage. On the other hand, the decoder performs the reverse process by converting the encapsulated bitstreams back into the original videos. The encoder typically consists of three key functions for compression: prediction modeling, spatial modeling, and entropy encoding. Conversely, the decoder utilizes the encoder's coefficients and prediction parameters to execute the entropy decoding function, inverse transform, and reconstruction, thereby converting the compressed videos back to their original form. Fig. 1 provides an overview of the Coder-Decoder (CODEC) process specifically based on the H.264 format [5].

In this study, we utilize the H.264 encoder as a case study to illustrate our proposed approach designed for FPGA edge computing platforms. The H.264 video compression standard incorporates hybrid video coding techniques to efficiently encode and decode video data. The encoding process encompasses multiple steps, commencing with video pre-processing tasks like resizing, color space conversion, and noise reduction. Fig. 2 illustrates the syntax of the H.264 video compression standard. Initially, the sequence of frames from the original video is encoded into Network Adaptation Layer Units (NALUs) packages. Each package comprises parameters and a list of slices that represent the video frames. A slice comprises an encoding information header followed by a set of macroblocks responsible for encoding a 16×16-pixel block within a video frame. Subsequently, compression techniques such as intra-prediction, inter-prediction, and transform coding are applied during the encoding process to minimize redundancies in the video data, thereby enhancing compression efficiency.

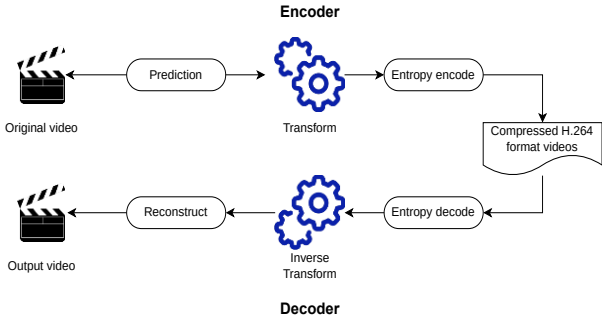

Fig. 1. The generic process of video encoder and decoder based on the H.264 format.

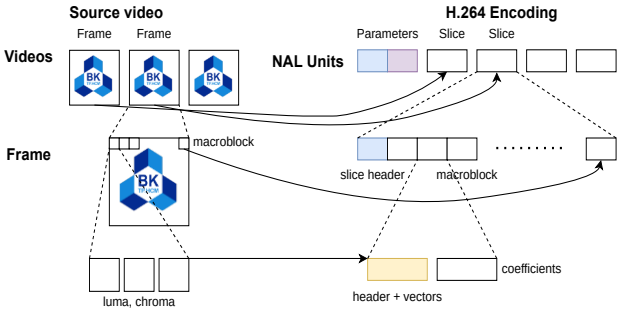

Fig. 2. The H.264 encoding process.

B. Related Work

In recent years, considerable research efforts have been dedicated to hardware implementation for accelerating video encoders. For instance, Tao and Gao [6] introduced an FPGA-based entropy encoder tailored for 8K videos. This approach incorporates customized binarization techniques and leverages pipeline methods to achieve substantial parallelism and improve performance. However, it is designed specifically for ultra-HD videos and targeted at high-performance computing applications. Waidyasooriya and Hariyama *et al.* [7] presented an OpenCL-based FPGA-powered acceleration approach for H.266, wherein the transformation and quantization steps are implemented on FPGA to enhance performance. Notably, their experimental system employs high-end FPGA devices rather than edge computing platforms like our work. Mukherjee [8] focused on accelerating the Hexagon-Diamond Search (HEXDS) block matching algorithm on FPGA to enhance the performance of the High-Efficiency Video Coding (HEVC) standard. However, this study only accelerates a specific part of the encoding process and is geared towards a different

compression standard. In Ref. [9], a heterogeneous CPU-FPGA computing architecture is utilized for the HEVC video encoder, targeting the OpenPOWER high-performance computing platform. Similarly, in Ref. [10], an acceleration approach using high-level synthesis is employed to enhance the interpolation filter in the HEVC video standard, with a focus on high-end FPGAs featuring Virtex 6 devices. Pastuszak [11] designed and developed an ASIC accelerator core utilizing the 90-nm TSMC technology to enhance the performance of the HEVC video encoder, achieving notable performance improvements through the ASIC (Application Specific Integrated Circuits) approach. Additionally, Garrido and Pescador *et al.* [12] proposed a method for Versatile Video Coding (VVC) that emphasizes the transformation step, specifically the 64×64 2D discrete sine/cosine transforms, rather than addressing the entire encoding process.

All the above proposals focus on high-performance computing systems where hardware resources available are many and energy capacity is higher than Internet of Thing (IoT) devices. Moreover, these studies target a specific video standard and a particular computing platform instead of providing a generic approach for accelerating video CODEC like our work. Compared to these proposed approaches, our work offers a more flexible design flow with the combination of HLS and FPGA so that various CODEC can be considered for different purposes and platforms.

III. Proposed Design Flow

In this section, we present our design approach for accelerating video streaming encoders on FPGA edge computing platforms, utilizing High-Level Synthesis (HLS). Fig. 3 provides an overview of the proposed design flow, highlighting the steps involved in accelerating the video streaming encoder using FPGA and HLS. The subsequent sections delve into the details of our proposed flow, elucidating each step in a comprehensive manner.

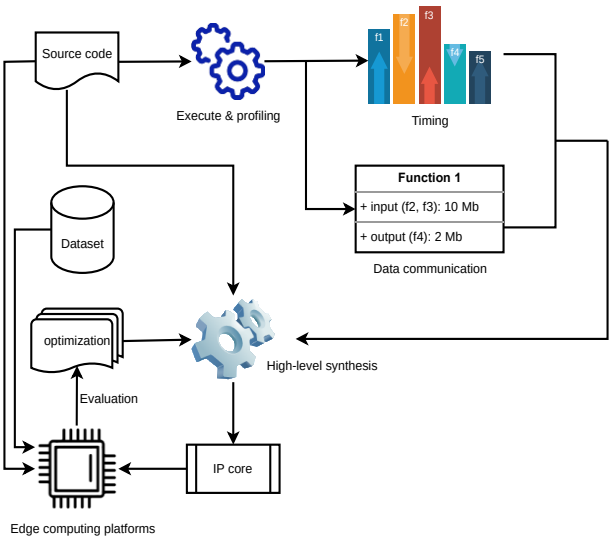

Fig. 3. The proposed design flow for video streaming encoder with HLS and targeting FPGA edge computing platforms.

A. Step 1: Profiling and Selecting Functions

The flow begins with the certified source code of an encoder method. This source code, typically written in C/C++, undergoes execution and profiling using a general-purpose processor. The purpose of this profiling is to identify functions that require significant computational resources and to evaluate data communication patterns. The consideration of data communication is important as the transfer of data between computing cores can also impact the overall system execution time. Based on the profiling results, designers then proceed to partition the original source code into software and hardware accelerated components. The software part is executed by the general-purpose processor, while the hardware dedicated accelerator cores handle the hardware part. By offloading the computationally intensive functions to the accelerator cores, the performance of these accelerated functions, as well as the overall system, can be significantly improved.

Please note that although the partition process mainly relies on profiling data, not all computationally intensive functions are chosen for accelerating by hardware. There may exist some procedures that cannot be accelerated with accelerator cores, for example interacting with external memory for getting input data. Besides, some functions will also be implemented on FPGA fabrics, although they are relatively inexpensive since they may exchange a large amount of data with other functions accelerated by hardware cores.

B. Step 2: Hardware Accelerator Cores Generation

The time-consuming selected functions are then synthesized with HLS tools provided by FPGA providers or third-party to generate the HDL-based (hardware description language) accelerator cores. This process can repeat several iterations to apply different optimization techniques, such as pipelining, unrolling loop, or array partition, to improve the accelerator cores' performance further. Although HDL-based accelerator cores can be built manually, we propose to use HLS for exploiting different optimization. Besides, the HLS process also provides a report with estimated latency and hardware resource usage results. Based on this report, we can determine if the cores can be implemented on the target platforms. Since IoT edge computing platforms usually suffer from limited resources, if the required resources exceed the capacity of the platforms, the previous step (selecting functions for accelerating) is remade. The output of this step is the HDL-based cores ready for synthesizing and integrating into other parts of the target platforms.

C. Step 3: System Integration

Due to the classification of software and hardware parts, the original source code needs to be modified to invoke the accelerator cores. The HDL-based cores and other parts, such as the I/O interface, interconnect, and Direct Memory Access (DMA), are then synthesized with design tools provided by FPGA vendors. Based on the target platforms, modified source code will be deployed on hardwired embedded (hard) or softcore processors. Finally, the final

system on the target platform executes the dataset for evaluation before deploying.

IV. Case Study with H.264 Encoder

In this section, we provide a case study featuring the implementation of the H.264 encoder on the Ultra96-v2 FPGA edge computing platform.

A. Source Code Analysis

As mentioned earlier, H.264 is a widely recognized video compression standard, with several certified H.264 encoder source codes available, including JM software [13] and ITU-H.264 [14]. In this study, we utilize the x264 open-source software from the VideoLAN Organization [15] as our reference source code for the H.264 encoder. Fig. 4 provides an overview of the key components of the H.264 encoder implemented by the VideoLAN software, namely Filter, Analysis, Encode, and Entropy. These four processes encompass the main computationally intensive functions.

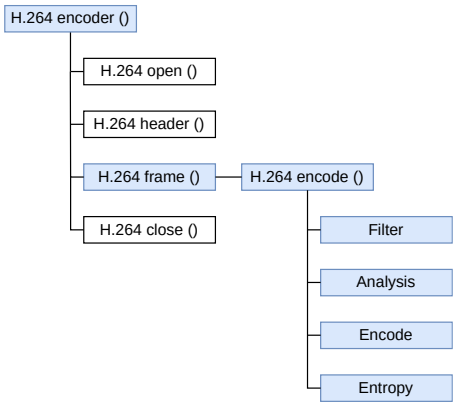

Fig. 4. Overview of H264 encoder main functions.

To extract the timing values of functions, the source code is profiled using the perf tools [16]. While numerous profiling tools could serve this purpose, we opt for perf due to its integration into most Linux kernels. Fig. 5 displays the perf results for the most computationally intensive functions within the source code.

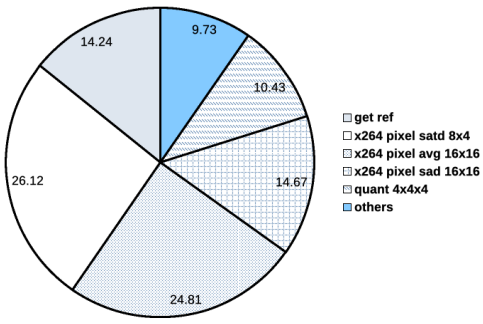

Fig. 5. The percentage of time for main functions of the H.264 encoder.

From Fig. 5, it can be observed that the get_ref function falls within the category of computationally intensive functions. However, it is worth noting that this particular function involves collecting data from external memory, which cannot be accelerated by an FPGA-based accelerator core. Consequently, only the four functions, namely x264_pixel_satd_8×4, x264_pixel_avg_16×16, x264_pixel_sad_16×16, and quant_4×4×4, are suitable for acceleration through FPGA fabrics. In this study, we focus on optimizing these five functions using High-Level Synthesis (HLS) to generate accelerator cores tailored for their efficient execution. The subsequent section provides an in-depth exploration of the optimization techniques employed within the HLS processes.

B. High-Level Synthesis Optimization

High-Level Synthesis (HLS) [17] is a process that generates hardware circuits from high-level programming languages like C or MATLAB. HLS offers several advantages over traditional hardware design methods, such as RTL (register transfer level). It provides higher productivity, faster design iterations, improved design quality, better portability, and reduced time to market compared to the traditional approach. In this study, we leverage the faster design iterations enabled by HLS to explore different optimization approaches for the aforementioned four functions, namely pipeline, unroll, and array partition.

The pipeline technique is utilized to reduce the initiation interval of loops or functions by executing concurrent operations. Unroll approach involves creating multiple copies of RTL logic to perform multiple operations concurrently, instead of executing a single block multiple times. Additionally, array partition optimization is employed to create a description in Hardware Description Language (HDL) for multiple small blocks of memory or registers, rather than using a large memory block [18].

Table I presents the estimation results obtained through HLS for the four functions, showcasing latency and hardware resource usage. The high-level synthesis process in this work is performed using the Xilinx Vitis tool [19]. While various optimization techniques are considered for each function, the table presents the estimation results with the most optimized methods. Notably, the table reveals that the pipeline approach is utilized for all functions, resulting in the utilization of Flip-Flops (FFs) as pipeline registers and Look-Up Tables (LUTs) for computational functions within all four cores.

TABLE I. HLS Estimation Results

Function	Optimized Techniques	Resource Estimates	Latency
x264_pixel_ satd_8×4	pipeline array partition	DSP: 1 FF: 2165 LUT: 2340	11.370μs
x264_pixel_ avg_16×16	unroll pipeline	DSP: 96 FF: 3672 LUT: 6178	75.360μs
x264 pixel sad 16×16	array partition pipeline	DSP: 0 FF: 2075 LUT: 2364	11.050μs
quant 4×4×4	array partition pipeline	DSP: 2 FF: 4088 LUT: 3791	1.900μs

C. System Implementation

Upon optimizing and generating HDL-based hardware descriptions for the four accelerator cores, we proceed to construct our testing system based on the hardware accelerator computing paradigm [3]. In our case study, we employ the Xilinx MPSoC Ultra96v2 edge computing board [20] as the evaluation platform. Our testing system comprises a 4-core ARM Cortex functioning as the host processor, as well as FPGA fabrics that can be utilized to create various components such as local memory (buffer), exchange registers for managing IP cores, and accelerator IP cores responsible for executing the aforementioned computationally intensive functions.

To minimize communication overhead and facilitate pipelining, a DMA block is incorporated into the system for data transfer between the ARM processor and local memory, which is utilized for IP core computation. The overall setup of the platform is depicted in Fig. 6, where all the components within the MPSoC (Multiprocessor System on a Chip) FPGA device are interconnected through a standard Advanced extensible Interface (AXI) bus. It should be noted that the image only displays the primary features of the system, given the constraints of space.

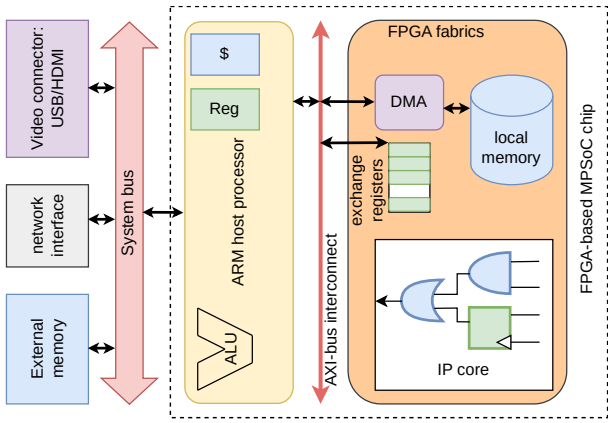

Fig. 6. Overview the FPGA-based edge computing system for H.264 encoder.

D. Pipeline Processing

As shown above, our final H.264 encoder system includes an ARM processor and four accelerator cores for processing the four computationally intensive functions of the x.264 software program. Therefore, applying the pipeline technique can further exploit the system performance improvement. The encoding processing on our system can be divided into four stages due to the in-balance processing time of cores. First, the ARM general-purpose processor executes the fetch-finalize stage to perform the get_ref and other functions after the four accelerator cores are done. Then, the three following stages are completed by accelerator cores entirely, including the satd stage with the x264_pixel_satd_8×4 core, the avg stage with the x264_pixel_avg_16×16 core, and the sad_quant stage with the two cores x264_pixel_sad_16×16 and quant 4×4×4. Although the pipeline technique can help improve much system's performance, the system needs more memory for storing data of different stages.

V. Experiments

In this section, we present our experiments to validate and estimate the acceleration ability of the above system. At first, the experimental setup is introduced. Then, we summarize the results of hardware resources used for our IoT edge platform. Finally, we introduce our experimental results in terms of execution time, throughput, and energy consumption to illustrate the goals of our work that is efficiency in performance and resources with respect to energy consumption.

A. Experimental Setup

The evaluation of the system described in Section IV is conducted using the Ultra96v2 board, which is equipped with a Xilinx Zynq FPGA. This FPGA offers more than 70 K Look-up Tables, 950 KB Block RAMs, and is accompanied by a 4-core ARM Cortex processor with a maximum clock speed of 1.5 GHz. IP cores generated through HLS, as well as Verilog-based exchange registers and local memory descriptions, are employed to construct the IP core wrappers within the FPGA's programmable logic section of the MPSoC device. These computational cores are connected to the ARM Cortex, serving as the host processor, via an AXI bus.

To evaluate the effectiveness of our proposed computing system, we compare it to the processing capabilities of an ARM processor alone. We examine the following platforms in detail:

1. Our system adopts a hardware accelerator architecture that includes both the ARM processor and hardware cores for the four computationally intensive functions of the H.264 encoder, namely x264_pixel_satd_8×4, x264_pixel_avg_16×16, x264_pixel_sad_16×16, and quant_4×4×4. The hardware cores handle these functions when invoked, while the ARM processor takes care of other parts of the encoder. In this configuration, the ARM processor operates at its maximum frequency (1.5 GHz), while the hardware cores run at 230 MHz.
2. The ARM system utilizes all cores of the 4-core ARM processor to process the entire H.264 encoder in parallel, using the x264 source code. Similar to our system, the ARM processor operates at the maximum frequency of 1.5 GHz to ensure a fair comparison. However, the FPGA fabrics (hardware cores) remain idle as only the ARM processor is utilized.

To test and compare our system with the ARM system, we employ uncompressed videos from the Derf media collection [4]. Additionally, we randomly select three HD videos from the dataset to evaluate the performance and energy efficiency of our proposed system. The subsequent section presents the experimental results obtained from these tests.

B. Experimental Results

Initially, the synthesis of our system is performed using Xilinx tools to configure the Ultra96v2 board. The

synthesis report indicates that our system can operate at 230 MHz with a power consumption of 4.208 W. Table II provides an overview of the hardware resources utilized by our system, showcasing both the absolute amount and the percentage in comparison to the total resources available on the FPGA device. According to the table, there is ample room for expanding the number of hardware cores or replicating existing accelerators to enhance system performance. Currently, we use only 53% Look-up Tables (LUTs), Flip-Flops (FFs), and 10% DSPs for building computational blocks to perform the encoding process. Besides, we use 61% Block-RAMs (BRAM) for storing the parameters of the H.264 encoding.

TABLE II. HARDWARE RESOURCES USAGE FOR BUILDING THE WHOLE SYSTEM

Hardware Resource Type	Amount	Percentage
LUTs	37,396	53%
FFs	74,793	53%
Block RAM	196	61%
DSP	19	10%

As previously mentioned, the pipeline technique employed in our system necessitates a significant portion of block RAM usage (61%). However, there is still over one-third of block RAM available to accommodate additional cores. In terms of Look-up Tables (LUTs), Flip-Flops (FFs), and Digital Signal Processing (DSP), which are the primary types of hardware resources in an FPGA, our system utilizes a maximum of 53% of the available resources. In other words, we can deploy smaller platforms with fewer resources than the Ultra96v2 board for the H.264 encoder. This aligns with our intended target, focusing on IoT edge platforms.

To evaluate the proposed system, three HD videos from the dataset were selected for testing. The performance of the system is presented in Table III, which highlights the execution evaluation. The results demonstrate significant speed-ups, reaching up to 14.9× compared to the ARM system. Specifically, for the HD video "in_to_tree 420 720p50" with a resolution of 1280×720, our system achieves a processing rate of up to 47 frames per second, meeting real-time requirements. Although we cannot achieve real-time ability with other videos ("riverbed 1080p25" and "ducks take off 420 720p50"), performance of the system can be improved with newer and better FPGA-based IoT platforms.

TABLE III. EXPERIMENTAL RESULTS AND COMPARISON OF OUR SYSTEM AND ARM PROCESSOR

Video Input	#frames	Our System (ms/frame)	ARM System (ms/frame)	Total Output Size
riverbed 1080p25	250	470.12	4231.12	11 MB
in to tree 420 720p50	500	21.06	313.76	648 KB
ducks take off 420 720p50	500	44.60	453.86	2.6 MB

Fig. 7 illustrates a comparison of the speed-ups achieved by the proposed system and the ARM processor. The figure clearly shows speed-ups ranging from 9× to 14.9×. For all the testing videos, our system with the proposed approach outperforms the ARM processor with 4 cores running at 1.5 GHz (our system works at 230 MHz only). Additionally, the figure showcases a comparison of energy consumption between the two systems, where energy is calculated as the product of power and time. Remarkably, our system achieves up to 6.24× greater energy efficiency compared to the ARM processor. This energy reduction allows our system works in more than 6× longer duration than the ARM-based system with the same amount of energy capacity. This is also the main purpose of our proposed approach that target IoT devices with limit energy capacity.

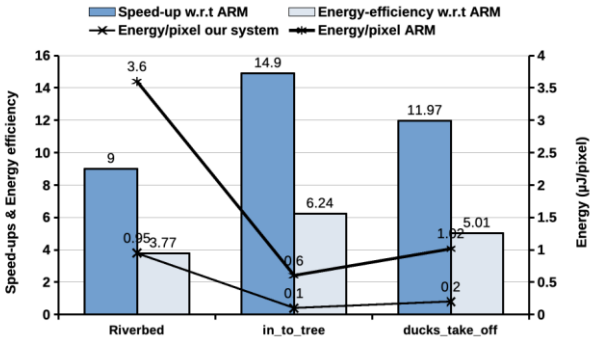

Fig. 7. Speed-ups and energy-efficiency with respect to the ARM system and energy consumption (µJ/pixel) of the two system.

Fig. 7 also presents energy consumption per pixel of the two systems. These values allow designers and researchers to determine the most appropriate approach for their application in the future. As shown in the figure, our system always requires less than 1 µJ for a pixel. Meanwhile, the ARM system energy consumption per pixel is variant from 0.6 to 3.6 µJ/pixel. These values prove that our system is energy efficiency and stability for edge computing systems.

VI. CONCLUSIONS

Over the past few years, video-processing IoT applications deployed on edge computing platforms have been widely utilized in various areas, such as object monitoring, surveillance, and check-in/check-out systems. However, despite the efficiency of the H.264 video format, it is challenging to process H.264 videos on edge computing platforms due to their low computing power. To address this issue, this paper proposes an approach that employs high-level synthesis techniques to accelerate video encoding using FPGA platforms for edge computing applications. The proposed approach profiles the H.264 encoding to lo- cate the computational-intensive functions to accelerate by FPGA. Optimization approaches such as loop unroll, loop pipeline, function pipeline, and array partition are employed with high-level synthesis techniques to generate the accelerator core. The hardware accelerator computing paradigm, which combines a host processor and the accelerator core for processing H.264

encoding, is utilized to implement the core. The proposed approach is tested on an edge computing FPGA Ultra96-v2 board to validate its effectiveness and performance. Experimental results indicate speed-ups of up to 14.9× compared to an ARM quad-core processor. The proposed approach also requires low power consumption, with a power requirement of 4.208 W.

In the future, we continue improve the proposed design flow and system architecture with more optimization techniques such as high-performance interconnect for data communication or multiple accelerator cores for parallel processing. Targeting modern platforms is also future work.

CONFLICT OF INTEREST

The author declares no conflict of interest.

ACKNOWLEDGMENT

The author acknowledges Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for supporting this study.

REFERENCES

[1] M. Caprolu, R. D. Pietro, F. Lombardi, and S. Raponi, "Edge computing perspectives: Architectures, technologies, and open security issues," in *Proc. 2019 IEEE International Conference on Edge Computing (EDGE)*, 2019, pp. 116–123. https://doi.org/10.1109/EDGE.2019.00035

[2] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog *et al.*: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018. https://doi.org/https://doi.org/10.1016/j.future.2016.11.009

[3] C. Pham-Quoc, J. Heisswolf, S. Werner, Z. Al-Ars, J. Becker, and K. Bertels, "Hybrid interconnect design for heterogeneous hardware accelerators," in *Proc. 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 843–846. https://doi.org/10.7873/DATE.2013.178

[4] Xiph.org: Video test media. DERF's collection. (2016). [Online]. Available: https://media.xiph.org/video/derf/

[5] I. E. Richardson, *The H.264 Advanced Video Compression Standard*, 2nd ed. Wiley Publishing, 2010.

[6] L. Tao and W. Gao, "A hardware implementation of entropy encoder for 8k video coding," in *Proc. 2022 IEEE International Conference on Multimedia and Expo (ICME)*, 2022, pp. 1–6. https://doi.org/10.1109/ICME52920.2022.9859988

[7] H. M. Waidyasooriya, M. Hariyama, H. Iwasaki, D. Kobayashi, Y. Omori, K. Nakamura, K. Nitta, and K. Sano, "Opencl-based design of an FPGA accelerator for h.266/vvc transform and quantization," in *Proc. 2022 IEEE 65th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2022, pp. 1–4. https://doi.org/10.1109/MWSCAS54063.2022.9859281

[8] A. Mukherjee, "VLSI architecture design of motion estimation block with hexagon-diamond search pattern for real-time video processing," in *Proc. 2021 IEEE 18th India Council International Conference (INDICON)*, 2021, pp. 1–6. https://doi.org/10.1109/INDICON52576.2021.9691531

[9] Y. Qiu, J. Jiao, Y. Tang, Y. Liu, J. Ren, X. Zeng, and Y. Fan, "A heterogeneous HEVC video encoder system based on two-level CPU-FPGA computing architecture," in *Proc. 2021 IEEE 14th International Conference on ASIC (ASICON)*, 2021, pp. 1–4. https://doi.org/10.1109/ASICON52560.2021.9620382

[10] P. Sjovall, M. Rasinen, A. Lemmetti, and J. Vanne, "High-level synthesis implementation of an accurate HEVC interpolation filter on an FPGA," in *Proc. 2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*, 2021, pp. 1–7. https://doi.org/10.1109/NorCAS53631.2021.9599653

[11] G. Pastuszak, "Multisymbol architecture of the entropy coder for h.265/HEVC video encoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2573–2583, 2020. https://doi.org/10.1109/TVLSI.2020

[12] M. J. Garrido, F. Pescador, M. Chavarrıas, P. J. Lobo, C. Sanz, and P. Paz, "An FPGA-based architecture for the versatile video coding multiple transform selection core," *IEEE Access*, vol. 8, pp. 81887–81903, 2020. https://doi.org/10.1109/ACCESS.2020.2991299

[13] Joint Video Team. (Jan. 2009). H.264/14496-10 AVC reference software manual. [Online]. Available: https://iphome.hhi.de/suehring/tml/JM%20Reference%20Software%20Manual%20(JVT-AE010).pdf

[14] International Telecommunication Union. (Feb. 2016). H.264.2: Reference software for itu-th.264 advanced video coding. [Online]. Available: https://www.itu.int/rec/T-REC-H.264.2

[15] VideoLAN. (Aug. 2013). x264, the best h.264/avc encoder. [Online]. Available: http://www.videolan.org/developers/x264.html

[16] Perf Wiki. Linux profiling with performance counters. [Online]. Available: https://perf.wiki.kernel.org/

[17] R. Gupta and F. Brewer, "High-level synthesis: A retrospective," in *High-Level Synthesis*, Netherlands: Springer Dordrecht, 2008, pp. 13–28. https://doi.org/10.1007/978-1-4020-8588-8_2

[18] Vitis High-Level Synthesis User Guide (UG1399). [Online]. Available: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls

[19] AMD Xilinx. Vitis unified software platform overview. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[20] AMD Xilinx. Zynq™ UltraScale+™ MPSoC. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html