International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Time Efficient FPGA Overlay for Compute Intensive JPEG Compression Blocks for Real Time Applications

[1]Minal S. Deshmukh and [2]P. D. Khandekar

Submitted: 17/09/2023 **Revised**: 29/10/2023 **Accepted**: 13/11/2023

Abstract: OpenCV functions are quite intensive in terms of computations, making them inappropriate for real-time embedded processor-based architectures for the reason that they have a fixed clock frequency and inadequate memory. A primary motivation for using FPGA overlays for the compute-intensive part of JPEG compression is that it can lead to fast compile time and OpenCV functions can be used in embedded systems. This study intends to synthesize the compute-intensive block of the standard JPEG compression algorithm on PYNQ-Z2 (Xilinx's Python productivity board), which has ZYNQ 7000 SoC-mixed programmable device that integrates a multi-core processor and an FPGA, providing a compelling platform for IoT, AI, and ML applications and the bit stream generated (overlay) is imported into Python as a hardware library. Application developers who are acquainted with software APIs can take advantage of adaptive computing platforms using the hardware library for the fast development of applications without needing to employ ASIC-style design tools to design hardware. This paper presents the role of the overlay by giving a performance comparison of the OpenCV function on processor-based and FPGA platforms in the form of a custom overlay, empowering general-purpose FPGA applications.

Keywords: *FPGA, synthesize, overlay, empowering, programmable, compression*

1. Introduction

In video processing, machine vision, and the medical imaging field, latency is critical for decision-making. A video at 40 frames per second with a resolution of 1280*1024 pixels would require nearly 943 megabits per second. A minute-long video would require 7 gigabytes of memory [5] [6] [11]. Usually, image processing is typically performed separately from the actual application in non-real time as latency is not a crucial factor. In real-time applications space and latency both are critical factors [9]. Without severely affecting image quality, image compression is a key factor in cutting down on both the storage and transmission times for images. The image can be compressed to the extent that the human visual system (HVS) cannot detect changes [5] [6]. In most of the real-time image processing applications image compression is a must-have feature due to storage space and transmission time limitations. A general-purpose CPU is used to implement standard compression methods, and latency may have an impact on the efficiency of real-time applications.

Implementation of the compression algorithm on a hardware platform can reduce latency [1]. The hardware implementation creates a dedicated block that handles the specified task, which decreases CPU overhead and latency. Hardware implementations increase the overall performance of the algorithms [1] and are typically implemented on an FPGA. Although Verilog code synthesizes efficient hardware but coding is labor intensive and the utilizer has to manually work on pipelining and synchronization [2][4], the alternative to Verilog coding is High-Level Synthesis (HLS) [13] [14]. Vivado HLS compilers offer a programming environment that compiles C++ code into an enhanced RTL microarchitecture, resulting in easiness and effective coding. In this work, we have developed a customized IP core by implementing the color conversion, quantization, and DCT (Discrete Cosine Transform) blocks of the JPEG compression method. The IP created in HLS is integrated with the ZYNQ processing system using Xilinx Vivado Design Suite and the bit-stream produced is brought into Python as a hardware library, i.e., overlay.

Since Python is used in interfacing with hardware on PYNQ (Python productivity for ZYNQ) framework, openCV allows us to compare our custom hardware on Programmable logic (PL) with equivalent optimized C++ functions on Processing System (PS) [8]. This method can also be used for the implementation of OpenCV functions other than those mentioned in this study at both a high performance and low power level, as well as the user's ability to simultaneously target demanding high data rate pixel processing jobs on programmable logic and lower data rate frame-based processing activities on ARM cores [10]. The hardware utilized in this study, which comprises of a dual-core ARM 9 CPU and programmable logic akin to an Artix 7 series FPGA, is interfaced with Python using the PYNQ board

School of Electronics and Telecommunication Engineering
Dr.Vishwanath Karad MIT WPU, Pune 411038,India
Ketan Raut
Vishwakarma Institute of Information Technology Pune
Email:mrsminal@yahoo.com, khandekar.prasad@gmail.com

2. FPGA Overlay

Traditionally, the behavior of the FPGA has been specified using text-based hardware description languages (HDL). However, an in-depth understanding of the capabilities of the target technology and core hardware ideas like pipelining and synchronization is still required to obtain the greatest performance out of the HDL implementation [1] [4] [14]. Developing a software application is an entirely different experience than coming up with timing constraints as well as pipelining and optimizing areas, etc., which is error-prone. While developing hardware designs, software programmers are often found burdened due to considerably lower productivity. In this study, we see the ways in which the idea of FPGA overlay can ease some of these burdens related to fast configuration time, portability, and resource abstraction. C Library functions are inbuilt functions in C programming, whereas, in the form of a bit stream, the overlay is a hardware library for programmable logic circuits [1] used to speed up the overall performance of software applications. PYNQ board comes with a number of default overlays, and also we can develop custom overlays using the Xilinx Vivado design suite [4] [8]. High-Level Synthesis and Vivado Design are two components of the basic phases of overlay generation shown in Figure 1 structure of creating overlays. The generated bit stream can be used as a hardware library in Python. A developer can dynamically load overlays into the FPGA that are identical to the standard library functions present in the C/C++ languages [1]

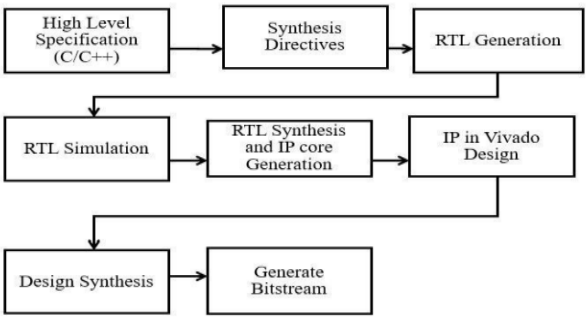

Fig 1. Structure for creating overlays

Vivado HLS and Vivado Integrator have been used in this investigation to develop a custom overlay [4][8]. In this study, a custom overlay is created and loaded from Python as needed for Color Conversion Quantization and DCT (Discrete Cosine Transform) blocks of JPEG compression algorithms. Without having a thorough understanding of CAD tools used to design ASIC-style chips, custom overlay enables users to use custom hardware in FPGAs [10].

3. Need of IP generation with Vivado High Level Synthesis (HLS)

Progressive algorithms in image processing are more sophisticated than ever before. Though the FPGA architectures are massively paralleled and have advantages in power, cost, and performance over traditional processors, many developers turn to high-level languages (C/C++) for implementing image processing algorithms due to the ease of programming on processor-based architectures over Register Transfer Language (RTL) based simulations [1][2][3][9]. Processor-based architectures have limitations in performance as they have fixed clock frequency and inadequate memory. It is also challenging to code these algorithms in RTL as it is time-consuming and error-prone [2]. The user must concentrate on the functionality of the architecture when using the Vivado HLS software tool for IP generation since this leads to effective coding. The HLS C specification is targeted at Xilinx programmable devices for the generation of RTL designs without manual interference [4] [13].

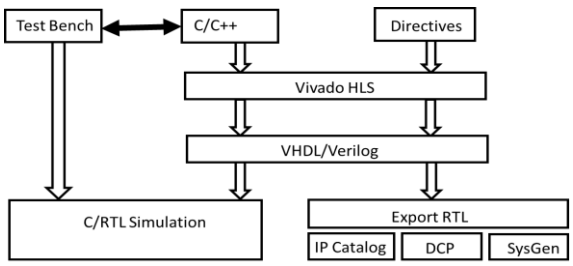

Fig 2. HLS design Flow [4]

The Vivado HLS is a framework for high-level synthesis provided by Xilinx, Inc. The HLS supports most of the libraries in C and C++ for synthesis [3] [4] [13]. Xilinx also provides a library of synthesizable OpenCV functions [10]. All the IPs in the system communicate with the PS using the AXI-4 protocol. In order to create high-performance hardware, HLS enables hardware designers to take the benefit of efficiency while evolving at a high degree of abstraction. The Vivado Design Suite tools and methodologies can be used to design and optimize IP sub-systems, integration automation, and accelerated design closure. Utilizing the Vivado HLS software tool allows exporting the RTL design to create custom IPs. The IPs designed in HLS can be modified according to the target hardware on which they can be implemented. The Vivado Design Suite comes with pre-installed Xilinx target devices and any device not listed can be added to the list manually. A custom IP creation flow utilising the Vivado HLS tool is displayed in the figure 2. The workflow of the Vivado HLS tool to generate a custom IP from a high-level language is as follows: 1. Include the C/C++ code in the source code. 2. Convert the C algorithm into an RTL implementation 3. Check the RTL implementation. 4. Use a test bench to validate IP functionality before synthesis. 5. Add the RTL to your IP catalog. The computationally demanding portions of the algorithm are synthesised in HLS, and the device's latency and resource usage are tracked.

4. JPEG Image Compression blocks with high computational Demand

Image compression is a significant issue in academic, commercial, and industrial applications. Since image processing algorithms need a lot of computation, they are not suitable for real-time applications [11]. To reduce the data storage and transfer overhead on smart devices, data compression is a key feature before transferring real-time created datasets for training and classification.

When processing JPEG images, the RGB color spaces are first changed to the Y, Cb, and Cr color spaces. Some colors are high-frequency colors because they are seen more strongly by the human eyes [11]. Due to the weaker sensitivity of the human eye to some chromium compounds, such as Cb and Cr, these hues can be disregarded. Then, through down sampling, we lower the size of the pixels. Our image is divided into 8x8 pixels, and forward DCT is used (Direct Cosine Transformation) [6][7]. Then, utilizing quantum tables for quantization, further, we use encoding methods, such as run-length encoding and Huffman encoding, to compress our data. This study describes only color transformation, DCT, and quantization blocks that were translated into a custom IP.

4.1 RGB to YCbCr Color Transformation:

The RGB image is converted into the YCbCr, CMY, and YUV, formats for use in applications where color information is crucial. The luminance is easier for the eye to perceive than the chrominance components. [11]. The biggest benefit of converting an image from RGB to YCbCr is the ability to remove the influence of brightness during image processing. As a result, the RGB image is changed to YCbCr during the JPEG compression process. The content of a monochrome image is closely approximated by the luma channel, which is commonly abbreviated Y (or, more accurately, Y', denoting that the channel is gamma encoded) the two Chroma channels, Cb and Cr, are the colour difference channels.

4.2 Discrete Cosine Transform

Many image and video compression techniques, such as the still image standard JPEG in lossy mode and the video compression standards MPEG-1, MPEG-2, and MPEG-4 (DCT), are built on the discrete cosine transform. Given that an image is a two-dimensional signal, the two-dimensional DCT is essential for still picture and video compression. DCT is a separable function, making it possible to compute the two-dimensional DCT by first applying the one-dimensional DCT across the signal horizontally, then vertically. [6] [15].

4.3 Quantization:

The quantization procedure reduces the amount of bits needed to hold an integer value by reducing the precision of the integer [11]. We may typically decrease the precision of the coefficients as we deviate from the DC coefficient given a matrix of DCT coefficients. This is due to the fact that as we get further away from the DC coefficient, an element's contribution to the graphical representation decreases and we are less concerned with preserving strict precision in its value.

5. PYNQ- Device used for prototyping

The key objective of PYNQ, Python Productivity for ZYNQ, is to make it simple for engineers of embedded systems to exploit the distinctive benefits of Xilinx devices in computer vision applications.

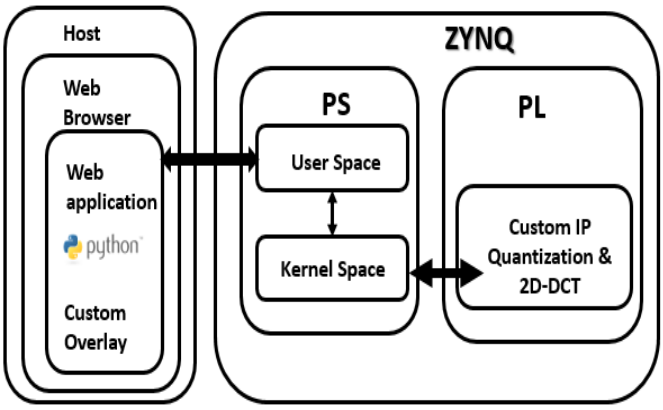

Fig 3. PYNQ Frame work

PYNQ frame work shown in figure 3 comprises the following features: A Client-side Jupyter notebook interface having web apps, open-source preinstalled packages, overlays, APIs, and IPs, and a Linux-based public library to regulate programmable logic [8]. Without prior knowledge of ASIC-style design, engineers and programmers who design embedded systems can use PYNQ, an open-source framework, to make use of ZYNQ devices (programmable logic from the Artix-7 family). ZYNQ-SOC integrates an ARM Cortex A9 dual-core processor and a ZyNQ XC7Z020 FPGA. The TUL PYNQ-Z2 board, which holds the XilinxZYNQ-7020 SOC, was employed in this investigation. The SOC contains 4 high-performance AXI ports, 13000 logic slices of PL (programmable logic) with 4 6 input LUTs on each, 8 FFs, 220 specialised DSP slices, and 630 KB of rapid block RAM. A bootable Linux image called PYNQ contains a number of open-source Python packages [8]. The PYNQ boards provide a Python interface to the hardware on the ZYNQ board, so it can be effortlessly programmed in the Jupyter Notebook using Python. The Processing unit and the Programmable Logic unit can communicate with one another using interfaces built on ARMs in AMBA AXI4. The PYNQ framework is based on the Jupiter notebook, and the implementation is almost similar to using the Jupiter notebook on any other system. An Ethernet cable connects the board to the host computer and the power to the board can be applied with a USB cable or a 12V power supply, and the operating system is loaded onto the board. Copy the bit stream file to the board in order to use the IP in the PL part [8][9] [13].

6. HSL Implementation

A programming environment that compiles C++ code into an improved RTL microarchitecture is provided by the VIVADO HLS compiler. The Vivado High-level Synthesis translates C/C++ code into Register Transfer Language. High-level RGB to YCbCr and Grey, DCT, quantization routines are written in C++. The RTL block must pass the test bench before exporting IP into the VIVADO design suite to generate the bit stream [13].

Algorithm: Color Transformation Function

Require: I/O Streaming Interface

Ensure: Data Transmission on Input Stream

while Input Buffer Not Empty do

Multiply I/P data with YCbCr matrix

Transfer Result on output stream

if last pixel then

BREAK

else

CONTINUE

end if

end while

The 256*256 original RGB image is converted to a YCbCr image using the colour transformation function described in the preceding technique, and the resulting YCbCr image shown in figure 4 is then used as input for the quantization and DCT block.

Fig 4.RGB to YCbCr Simulation result

Applying the aforementioned 2D DCT and Quantization base line JPEG compression algorithm produced a compressed image with a dimension of 256 * 256 and a compression ratio of 1.97 when using HLS.

Algorithm: 2D DCT and Quantitation

Require: I/O Streaming Interface

Ensure: Data Transmission on Input Stream

while Input Buffer Not Empty do

Feed input data to 2d-dct function

Multiply the result with quantization matrix

if last pixel then

BREAK

else

CONTINUE

end if

end while

HLS offers C libraries to support greater efficiency and high performance RTL design. This technique can enable numerous computer vision algorithm to be developed with high performance. This design we can put it to the Vivado Design suit to create IP core.

7. Developing Custom IP Core for Color Transformation Function quantization and DCT using HLS

The Vivado Design includes a ZYNQ processing system (PS), AXI interconnect, and a custom IP AXI_DMA controller. Design for color conversion shown in figure 5. and DCT, quantization block is in Figure 6. This configuration, enables very quick communication between the processing system and the programmable logic as well as the sharing of external memory by the CPU and FPGA over a relatively quick memory interface in comparison to off-chip memory. This kind of approach combines the adaptability of a CPU with the speed of specialized hardware by allowing ordinary software to interleave with hardware-accelerated function calls.

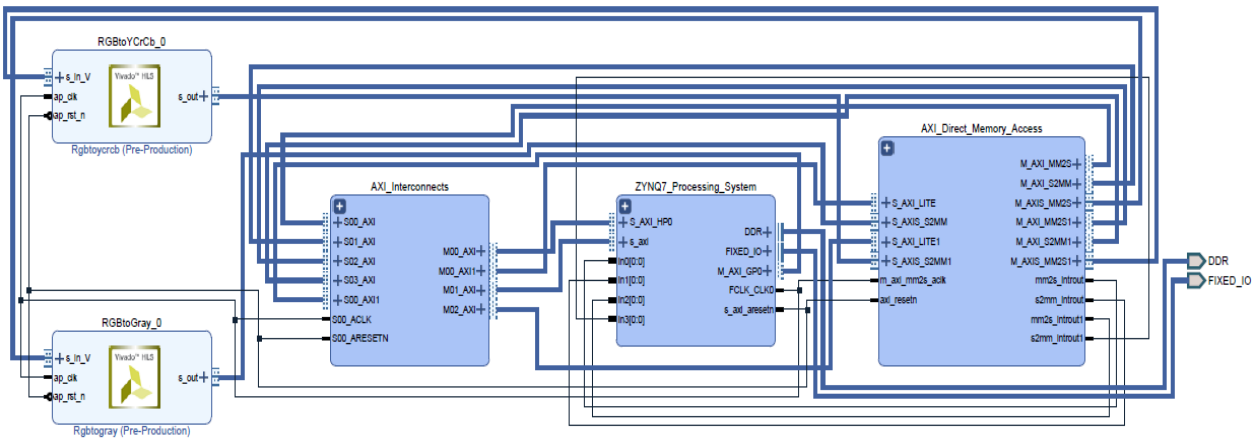

Fig 5. Vivado Design for YCbCr Color conversion blocks

The operating system peripheral interface of the ARMA9 processor is part of the ZYNQ processing system. Using the AXI interface, the ZYNQ processing system interfaces with customized Intellectual Property (IP; predefined logic functions that can be incorporated into your design). Xilinx video processing components typically connect with pixel data using the AXI14 streaming protocol. The AXI memory interface and AXI peripheral interconnect allow any CPU-based software to incorporate the AXI14 connector and utilize it to connect to an FPGA device. Direct memory access (DMA), which is used to access data from IP, minimizes CPU overhead. The PYNQ structure's custom IP core was created in VIVADO HLS and then imported into the Python interface of the framework. In order to leverage the PYNQ framework for direct performance comparison of software and hardware implementation, the PYNQ-Z2 board comprises the ZYNQ-7020 SOC (system on chip) [7][8][10].

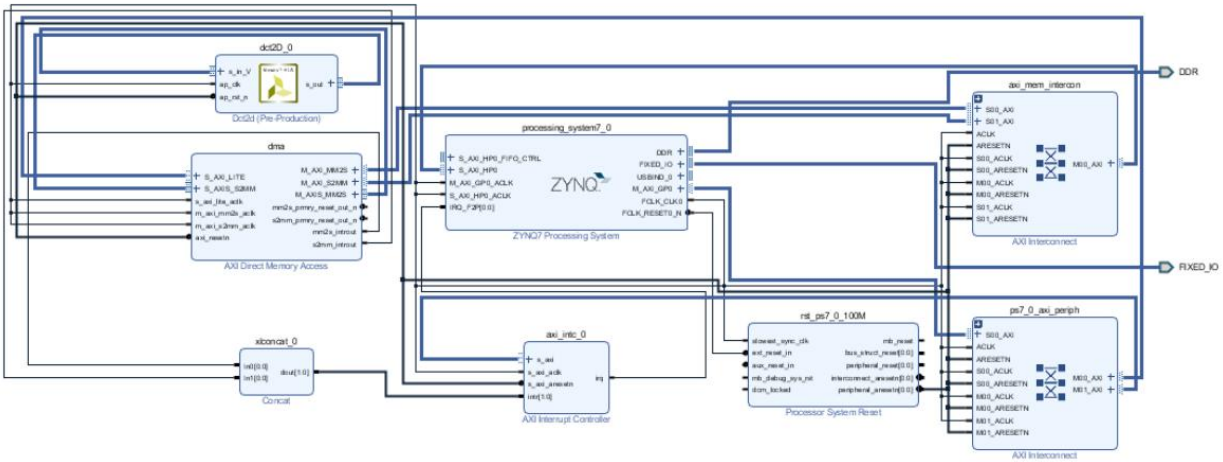

Fig 6. Vivado Design for DCT & Quantization block

8. Performance Comparison

The PYNQ Z-2 board's related IPs correspond to the computationally intensive color transformation, DCT, and quantization functions that are built and synthesized in Vivado HLS. We have measured the execution time on the FPGA, and on the CPU with and without the OpenCV function. For real-time image processing applications, the FPGA execution time is shorter in color transformations, DCT, and quantization, according to Tables I and II, which also provide details on an essential study attribute used for custom overlay design. PYNQ overlays are created using the Vivado IP integrator, which simplifies embedded design. Without having to create a color transformation overlay, software developers can import bespoke overlays using the Python environment. This makes it possible for Python running on the Processing system to control overlays made using programmable logic. This is akin to software libraries created by competent programmers that are accessible to programmers working at the application level [3]. Because Python is used in the PYNQ structure to interface with hardware, OpenCV can compare custom hardware implementation on programmable logic (PL) with equivalent optimized C++ functions on the processing system (PS) [8].

Table I. performance estimate of the RGB to YCbCr

Computing System	Process Time (S)
IP on Programmable Logic of FPGA(custom overlay)	0.00366
Processing System without using OpenCV function	0.1011
Processing System using OpenCV function	0.00768

Table II. Performance estimate of 2D-DCT and Quantization

Computing System	Process Time (S)
IP on Programmable Logic of FPGA(custom overlay)	**0.8531**
Processing System without using OpenCV function	**120.44**
Processing System using OpenCV function	**0.5114**

9. Conclusion

This study presents cutting-edge technology, i.e., FPGA custom overlay for color transformation, DCT, and quantization blocks to accelerate real-time processing applications. HLS successfully converts C++ code to HDL code while utilizing the fewest resources possible and getting a custom IP on FPGA to execute at the fastest possible speed. The approach used to employ specific design in programmable logic without the need for application-specific integrated circuits gives designers greater options when balancing precision, speed, and power in portable systems. Only executable codes are reloaded when switching applications, ensuring excellent portability without the need for FPGA re-synthesis, in a manner similar to loading a software default library. Overlays for color conversion and DCT can be loaded into the FPGA as needed for any image processing application, not limited to JPEG compression. This approach allows for the development of complex real time algorithms on FPGA.

References

[1] Abhishek Kumar Jain, Douglas L. Maskell ,and Suhaib A. Fahmy , Senior Member, IEEE "Coarse Grained FPGA Overlay for Rapid Just-In-Time Accelerator Compilation", IEEE Transactions on Parallel and Distributed Systems (Volume: 33, Issue: 6, 01 June 2022).

[2] Yunxuan Yu, Chen Wu, Xiao Shi, Lei He, "Overview of a FPGA-Based Overlay Processor", 2019 China Semiconductor Technology International Conference (CSTIC), Added to IEEE Xplore: 08 July 2019.

[3] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, Zhiru Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment", Ieee Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 30, No. 4, April 2011

[4] Vivado Design Suite User Guide, UG910 (v2021.2) October 27, 2021.

[5] Weidong Xiao,Nianbin Wan, Alan Hong, Xiaoyan Chen, "A Fast JPEG Image Compression Algorithm Based on DCT". 2020 IEEE International Conference on Smart Cloud (SmartCloud), Added to IEEE Xplore: 27 November 2020.

[6] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform", IEEE Trans. Computers, vol. C-23, pp. 90-93, Jan. 1974.

[7] G. K. Wallace, "The JPEG still picture compression standard", in Communications of the ACM, vol. 34, pp. 31-44, April 1991

[8] Python productivity for Zynq (Pynq) Documentation, Release 2.7, Xilinx, November ,2021.

[9] Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS

Video Libraries, XAPP1167 (v3.0) April 30, 2015

[10] John C. Russ, F. Brent Neal, “The Image Processing Handbook, 7th Edition”, CRC Press ISBN: 9781498740289, September 2018.

[11] Yahia Said, Taoufik Saidani, Mohamed Atri, “FPGA-based Architectures for Image Processing using High-Level Design”, WSEAS TRANSACTIONS on SIGNAL PROCESSING Volume 11, 2015.

[12] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera,Kees Vissers, and Zhiru Zhang, “High-Level Synthesis for FPGAs: From Prototyping to Deployment”, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 30, NO. 4, APRIL 2011.

[13] Dimitris Tsiktsiris, Dimitris Ziouzios, Minas Dasygenis, “A High-Level Synthesis Implementation and Evaluation of an Image Processing Accelerator”, Special Issue "Modern Circuits and Systems Technologies on Electronics 2018"

[14] Brant and G. G. F. Lemieux. ZUMA: an open FPGA overlay architecture. In 2012 IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012, 29 April - 1 May 2012, Toronto, Ontario, Canada.

[15] Muzhir Shaban AL-Ani, Fouad Hammadi Awad, “THE JPEG IMAGE COMPRESSION ALGORITHM”, International Journal of Advances in Engineering & Technology ISSN 2231-1963 2013

[16] María, K., Järvinen, M., Dijk, A. van, Huber, K., & Weber, S. Machine Learning Approaches for Curriculum Design in Engineering Education. Kuwait Journal of Machine Learning, 1(1). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/111

[17] Tonk, A., Dhabliya, D., Sheril, S., Abbas, A.H.R., Dilsora, A. Intelligent Robotics: Navigation, Planning, and Human-Robot Interaction (2023) E3S Web of Conferences, 399, art. no. 04044, .