Hardware Acceleration of Video Edge Detection with Hight Level Synthesis on the Xilinx Zynq Platform

Taoufik Saidani
Department of Computer Science
Faculty of Computing and Information Technology
Northern Border University
Rafha, Saudi Arabia and
Laboratory of Electronics and Microelectronics
Faculty of Sciences of Monastir
University of Monastir
Monastir, Tunisia
taoufik.saidan@nbu.edu.sa

Refka Ghodhbani
Department of Computer Science
Faculty of Computing and Information Technology
Northern Border University
Rafha, Saudi Arabia and
Laboratory of Electronics and Microelectronics
Faculty of Sciences of Monastir
University of Monastir
Monastir, Tunisia
Refka.Ghodhbani@nbu.edu.sa

Abstract-The study conducted in the current paper consists of validating an original design flow for the rapid prototyping of real-time image and video processing applications on FPGAs. A video application for edge detection with Simulink HDL coder and Vivado High-Level Synthesis (HLS) has been designed as if the code was going to be executed on a conventional processor. The developed tools will automatically translate the code into VHDL hardware language using an advanced compilation technique. This amounts to embedding processors on Xilinx Zynq-7000 System on-Chip (SoC) device in an optimal manner. This automated hardware design flow reduces the time to create a prototype since only the high-level description is required. The design of the video edge detection system is implemented on Xilinx Zynq-7000 platform. The result of the implementation gave effective resource utilization and a good frame rate (95 FPS) under 170MHz frequency.

Keywords-high-level synthesis; automated hardware design; co-design; Xilinx Zynq-7000

I. INTRODUCTION

Over the past ten years, several architectures combining reconfigurable processors and/or circuits (Field Programmable Gate Arrays-FPGAs) have been proposed for the acceleration of the execution of increasingly complex applications [1]. Dedicated signal and image processing systems currently use either processors with general or dedicated use, wired solutions configured for specific circuits of the ASIC type, or a combination of these two. However, the conjugation of the increasing algorithm complexity and the large volume of data to process, usually with strong real-time constraints, requires performance that processors cannot provide [2]. Parallel processing can provide speed and programming flexibility, but at the expense of cost and complexity of implementation. In addition, the use of ASICs which offer more processing speed suffers from rigidity and expensive development. Indeed, adding a new functionality to a wired system will surely require a redesign of one or more ASICs and will increase cost [3]. The reconfigurable architectures represent an appropriate response, they offer better performance than programmable architectures and more flexibility than wired solutions. They use reconfigurable logic components that allow the user to modify the architecture after manufacturing it in a software way, unlike the ASIC whose algorithms are wired in silicon [2, 3].

Video and image processing are now booming and playing an increasingly important role in our everyday life [4, 6]. The integration of embedded systems in the field of modern wireless communications allows us to respond quickly to its ever-increasing demands. The invention of the FPGA made possible the concept of reconfigurable material hardware for video and image processing systems. The traditional design of video and image system architecture around the FPGA does not allow high productivity [3, 4]. The latter can be improved by using a new design technique based on the Model Based Design (MBD) model [5]. Tools based on this technique are intended for embedded systems, the development of signal processing algorithms, the rapid integration of systems, and the analysis of the behavior of complex digital systems for a wide variety of cases. To resolve this constraint, many high level synthesis tools have been developed to take advantage of this technique to achieve FPGA rapid prototyping [3]. The FPGA with its great integration capabilities and reconfiguration is a key component in rapidly developing prototypes. In the objective of encouraging the wide distribution of this type of circuits, it is necessary to improve the development environments to make them more accessible to non-electronics experts[8, 9].

The current study consists in proposing and validating a design flow for the rapid prototyping of real-time image processing applications on FPGAs. It programs a model with HDL coder library as if the code was going to be executed on a Zynq 7000 [8, 14]. The developed tools will automatically generate the code into hardware language (VHDL) using an advanced compilation technique. This amounts to embedding processors in the FPGA in an optimal way.

Corresponding author: Taoufik Saidani

II. ACCELERATION OF THE PROPOSED ARCHITECTURE

Figure 1 shows the functional diagram of the proposed system architecture. The design is applicable to both video and image processing since the input pixels are processed successively. For embedded video processing, the process requires mega pixels. It is built by a video block that can be programmed by adapted algorithms and thus accelerate complex and heavy (in terms of resources and execution time) algorithms on the integrated blocks of the Xilinx SoC platform. The output of the video and image systems is passed directly to a video processing block and a display component [10]. The various video and image processing accelerators in FPGA are implemented in platform systems of the Xilinx Zynq 7000 [5, 8].

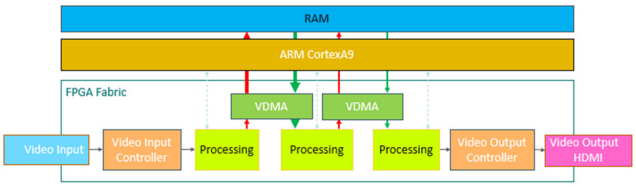

Fig. 1. The proposed global diagram for video and image system architecture.

In the architecture shown in Figure 1, the Advanced Extensible Interface (AXI) interconnects the processor that transmits the input video to the video processing pipeline system. The USB camera shares the interfaces and communicates with the ARM processor. The frame data are stored in the memory controller. Once the video IP core finalizes the process on the frame, it is displayed via the HDMI display. The pipeline is continued for the next iterations [14].

III. MBD MODEL BASED DESIGN FOR VIDEO AND IMAGE PROCESSING

The MBD technique for FPGAs results from the need to design complex DSP systems which require specific arithmetic units such as the addition-compare-select unit for the Viterbi decoder. These specific computing units require a finer level of FPGA-based circuit optimization [11]. This level of optimization is generally associated with traditional digital design video and image processing system. Model-based design is essentially a way of describing how a system will interact with the analog world in real time. This FPGA prototyping technique consists of converting the model of the system in question from its mathematical formulation to an executable specification [12]. The MBD technique provides a common framework that involves different phases of the development process. This reduces the Time to Market needed to create a complete model. The associated design phases to this technique allow the designer to locate and correct errors before prototyping the system [13]. There are many tools available for designing FPGA systems using the MBD technique. Most of these tools take advantage of the standardized Unified Modeling Language (UML). These tools differ in the way they describe a system and define its characteristics. Some implementation techniques used by these tools may be less effective than others. However, they guarantee rapid prototyping of the system ensuring time efficiency. The choice of a tool depends on many factors such as the level of flexibility, the availability of pre-built libraries/blocks, and the overall understanding of these blocks. Some UML-based tools are the Arti-Real time Studio, the I-logix's Rhapsody, MATLAB and Simulink Realtime workshop [9]. These tools are used for the design of an embedded multiprocessor environment.

The tools that allow the generation of HDL code for the FPGA can be classified into two categories, the block-based tools and those based on the C language of blocks that generate HDL code from the block diagram, which is then used by the synthesis tool hardware to implement the system design on FPGA. Most of these tools, such as Synplify DSP, Xilinx System Generator, DSP Builder Altera , and Simulink HDL Coder [5], are based on Simulink and MATLAB environment. These tools guarantee a high level modeling environment of signal processing algorithms. Blocks from the Simulink library are used with IP cores from FPGA suppliers to create HDL code specific to the platform in question. Tools such as the Simulink HDL Coder allow more flexibility to the designer, as they integrate MATLAB functions and m-block files. With these tools, the designer develops a Simulink model, then translates it to the FPGA environment. The second category of MBD tools uses C programming language to create an abstraction for the design of systems. Among these tools are the Mentor Graphics Catapult C and the Celoxica's Handel- C. The main motivation behind these tools is the Simulink HDL coder, which is commonly used for the implementation of embedded systems on FPGA [13].

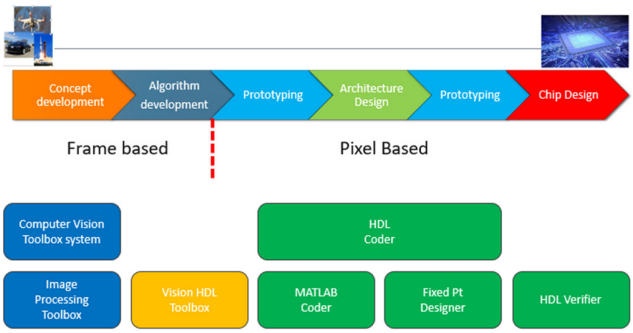

Fig. 2. A complete solution for embedded vision.

Simulink HDL coder is an MBD tool that enables modeling, analysis, and simulation of systems. It gives a well-structured graphical environment for the designer that allows him/her to create system designs of a high level of complexity. In addition, this tool allows the user to create flexible custom blocks from MATLAB functions. The Simulink HDL coder allows the designer to create HDL code bit-precise and synthesizable from the model developed using Simulink blocks. The obtained HDL code can be synthesized and mapped to the target FPGA board using tools such as Altera Quartus II, Synplify, and Xilinx Vivado. The Simulink HDL coder has many built libraries [15]. Some of these predefined libraries include adders, multipliers, accumulators, integrators, multi-port switches, lookup tables, etc.

IV. MODEL BASED MDB DESIGN FOR FPGA PROTOTYPING

A typical MBD design flow for implementing FPGA-based video and image processing system is shown in Figure 3.

A. System Specifications

The design of any DSP block always begins with defining the design specifications. This definition represents an abstract way to describe the main function of the block. At the end of this step, different parameters are obtained, such as the type of input/output and the basic mathematical function of the block. We can always determine whether the design of the block can be reused or not [12].

B. Analysis of Design Needs

As mentioned above, the basic mathematical function of the block is determined in the specification phase. The next step in the design process is to identify the algorithm required for the implementation practice. This algorithm is chosen based on many factors such as complexity, duration of the calculation, and the resources used. The algorithm is then subdivided into functions [12].

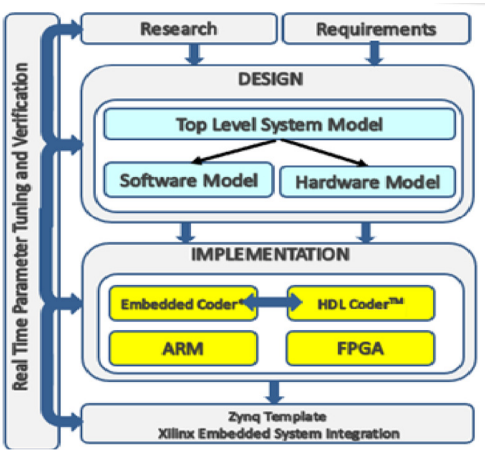

Fig. 3. Model-based design for embedded vision.

C. Simulation and Modeling

In this step we convert the design specifications into a high-level model using MBD tools such as the Simulink HDL Coder and XSG. This step consists in representing the mathematic equations in the form of models. Libraries and blocks provided by the tool facilitate this representation. The resulting model is then tested by observing the output for the various test input signals. Despite the wide variety of features and blocks provided by Simulink, only certain blocks can be converted using its HDL conversion environment which means that the other blocks must be replaced or built using the primitive libraries of the HDL converter tools such as VHDL [12].

D. System Implementation

This is the most important step. It consists of an automatic generation mechanism which is provided by the MBD tools. The flow advisor tool provided by Simulink HDL Coder, helps checking the model compatibility for code generation, converting the floating point model to a fixed point model, setting the clock, types of input/output data into the fixed point model, and data scaling. When the HDL code is generated, it can be synthesized using the Simulink HDL Coder tool. The latter supports simulation tools. It also supports the synthesis tools of Xilinx and Altera [14].

V. EXPERIMENTAL APPLICATIONS

A. Sobel Edge Detector

The Sobel operator allows to locally evaluate the norm of the two-dimensional spatial gradient of a grayscale image. The regions of strong local variations in intensity corresponding to the contours are amplified. Along the Ox and Oy axes, the Sobel operator approximates the directional derivatives using a convolution of the image $f(x,y)$ with 3×3 masks. We notice that the mask of the Sobel operator corresponds in fact to the application of a smoothing operation by the (1 0 1) operator followed by the application of a derivation operation by the (1 0 -1) operator in the orthogonal direction. The 3×3 matrix is convolved with the image to calculate the approximated horizontal and vertical gradients Gx and Gy as follows:

$$G'x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad (1)$$

$$G'y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (2)$$

The approximate absolute gradient magnitude and the gradient angle of each pixel are shown by:

$$|G'| = \sqrt{G'x^2 + G'y^2} \quad (3)$$

$$\theta = \arctan\left(\frac{G'y}{G'x}\right) \quad (4)$$

B. Edge Detection Hardware Design Using Simulink

The proposed video edge detection hardware prototyping uses the Sobel filter algorithm, and it has been designed with the Simulink HDL coder toolbox, so this design can be inserted as an FPGA IP core within any video processing pipeline flexibly. This is followed by a detailed description of the proposed hardware architecture. Figure 4 presents the top-level module for video edge detection based on the Sobel filter. The "video From File" block obtains the input video for the video edge detection system from the directory. This input video is converted to frames. The frames are serialized to pixels. Since the pixel values are of double type, the "Convert" block is used. Pixel-streaming processing is performed by HDL implementations of image and video processing algorithms. Therefore, a pixel stream is created with the Simulink "serialize" blocks [12]. The inverse process at the output of video processing system is performed by a "deserialize" block to verify the output processed in image format. These two blocks are depicted in Figures 6 and figure 7 respectively.

C. Synthesis and FPGA Implementation

Finally, in the third step (Figure 8), the Simulink HDL Coder converts the video edge detection Software-Hardware model into AXI4 streaming bus compliant IP core in the form of HDL (VHDL) source code. To verify in real time the

functionality of this IP core in a practical environment, a Hardware–Software co-design (HW-SW) has been implemented directly on a Xilinx Zynq-7000 AP SoC XC7Z020-CLG484 FPGA running at 170MHz. Simulink HDL coder generates a Full Vivado project for the HW-SW. This project can be implemented in the Xilinx Vivado tool (version 2019.1), with all the software and hardware peripherals. Also, the Color transform IP core and the Sobel core are connected across a single bus.

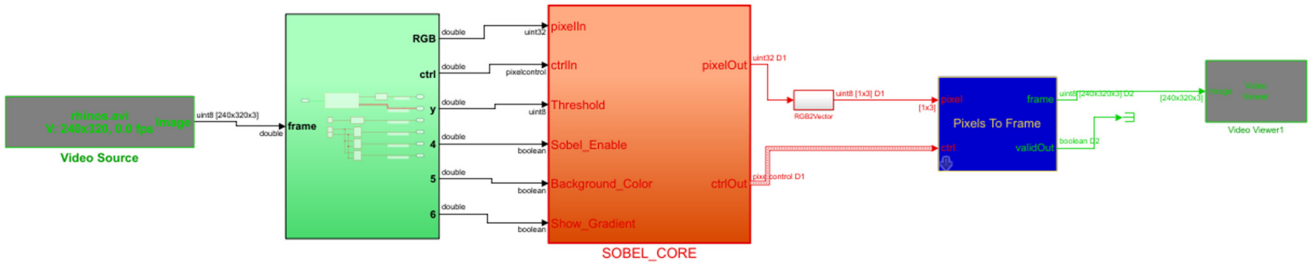

Fig. 4. Simulink HDL coder model for video edge detection.

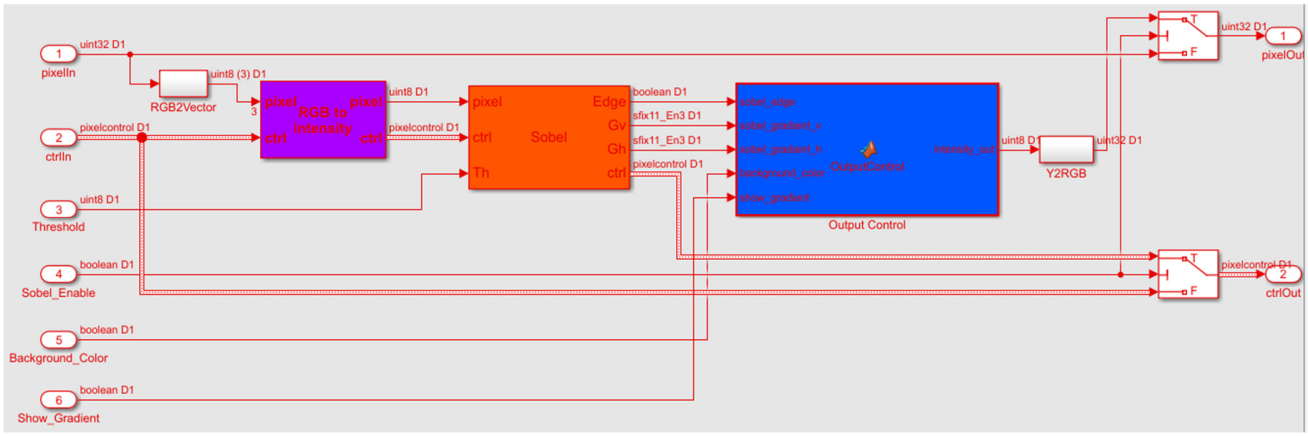

Fig. 5. Simulink HDL coder model for video edge detection core.

Fig. 6. Frame pixel serialization.

Fig. 7. Frame pixel deserialization.

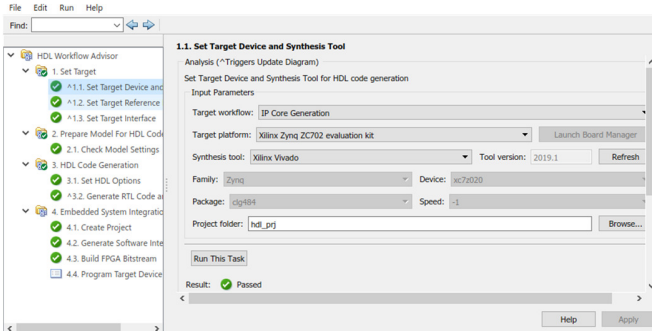

Fig. 8. HDL workflow for the video edge detection system.

The HLS tool allows the generation of stream interfaces incorporating the hardware accelerator (IP). Once the high level synthesis is completed, a compressed file (.zip) containing all hardware components is generated. This file is exported to the media Vivado packages to generate our SW/HW design. At this level, the Vivado tool constitutes a complete environment for the design of the finalized SW/HW architecture. The latter allows the installation of an on-board processor connected to one or several hardware accelerators through AXI interconnection buses specific to the selected platform. For example, in order to reset the IP core, one has to write 0x1 to the bit 0 of the IPCore_Reset register. To enable or disable the IP core, 0x1 or 0x0 must be written to the IPCore_Enable register.

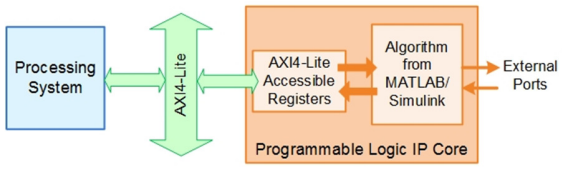

Fig. 9. Software-Hardware interface via AXI4-Lite.

To access the data ports of the MATLAB/Simulink algorithm, read or write to the associated data registers. The AXI4 Slave port to pipeline register ratio is selected as 35 in task 3.2 for this model. The default delay to read the AXI4 register is one clock cycle. Depending on the selected ratio and the IO connected to the AXI4 interface, register pipelining is introduced in the read logic of the AXI4 registers. For this model the AXI4 pipeline register ratio setting 35 is larger than all the readable AXI4 slave registers. The total readable AXI4 slave registers are 1, so no pipelining is added to the AXI4 register read back logic. Figure 11 shows the design created using the Xilinx's Vivado CAD tool.

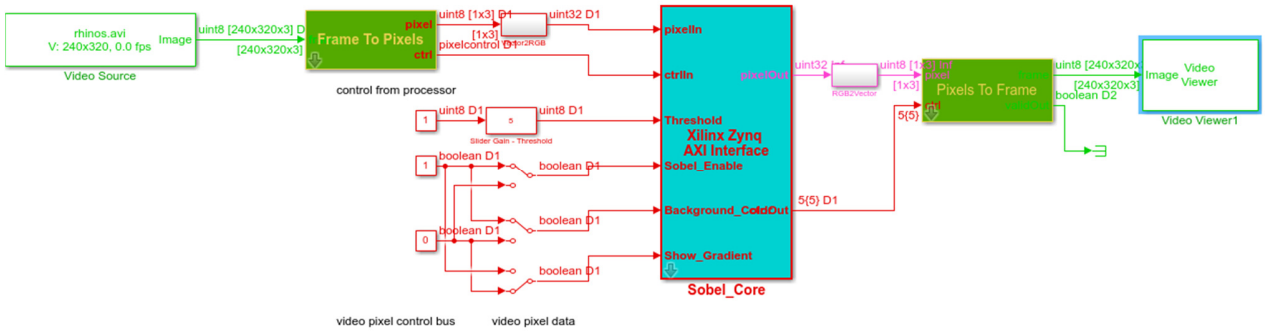

Fig. 10. The AXI Zynq interface.

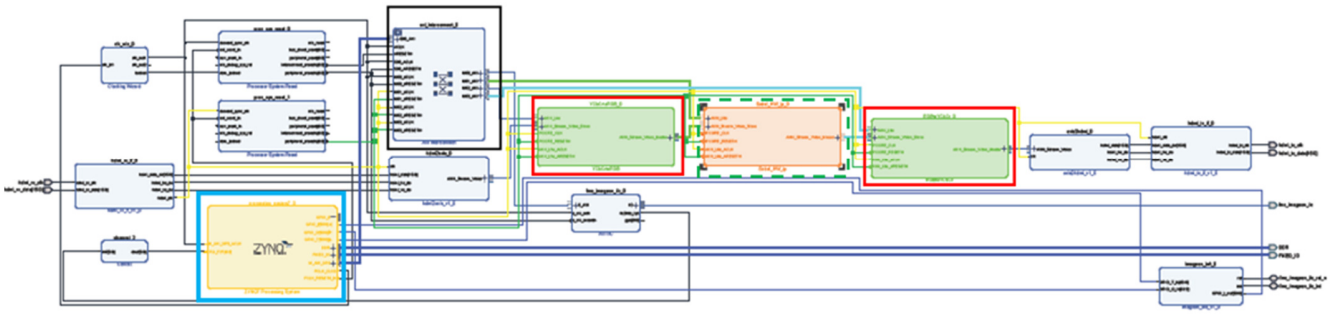

Fig. 11. HW/SW video edge detection architecture.

In Figure 11, the blue rectangle encircles the IP ZYNQ7 Processing System which represents the processor part of the Xilinx Zynq-7000 SoC. This IP is used to make the system configurations, such as configuring clocks and enabling the propagation of security status from ARM cores (the security state of the processor part) to the reconfigurable part. The rest of the blocks constitute the elements that are implemented in the reconfigurable part of the SoC. The black rectangle encircles the AXI Interconnect which allows the IPs to be connected to each other. The green rectangle surrounds the edge detection IPs with Sobel filters. The red rectangle surrounds the direct and reverse color transformation IP.

D. Resource Utilization

At the next step, we implemented our video processing system cores on Zynq-7020 SOC. The resources used are presented in Table I. Data values, instruction memories, and the firmware are stored in BRAM. The consumed LUTs and DFFs define the architecture of the processor including control signals, internal registers, and microcode. The optimum frequency for our proposed system core is approximately 170MHz and 95FPS for throughput. The complexity of the implemented design affects the resources of the utilized Zynq7000 SOC. The proposed design is performed for input resolution of 1080-1920 frames. The reconfigurable SOC platform using Vivado is approved by these results.

TABLE I. RESOURCE UTILIZATION AND MAXIMUM FREQUENCY FOR THE VIDEO SOBEL EDGE DETECTION MODULE

Maximum frequency	170 Mhz	
LUT-FF Pairs	3816.667	7.38%
LUTs as Logic	2727.667	5.12%
LUTs as Memory	648.6667	3.73%
Slice Registers	2465	2.32%
RAM 36/18	0	0
DSP48	0	0

VI. CONCLUSION

In this paper, a high level model-based hardware design flow using MBD tools was presented. The proposed design flow was put by a video edge detection prototyping into a Zynq 7000 SoC board. Each step of the adopted high level synthesis

prototyping design, from the top level description to the hardware implementation, was described. The global design process can reduce the time of processing by 65%. The experimental results of the proposed video design based on Sobel edge detection reimburse the hardware constraints reducing the complexity for embedded video processing in FPGAs.

REFERENCES

[1] H. M. Abdelgawad, M. Safar, and A. M. Wahba, "High Level Synthesis of Canny Edge Detection Algorithm on Zynq Platform," *International Journal of Computer and Information Engineering*, vol. 9, no. 1, pp. 148–152, Jan. 2015.

[2] T. T. Duong, J. H. Seo, T. D. Tran, B. J. Young, and J. W. Jeon, "Evaluation of Embedded Systems for Automotive Image Processing," in *2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Busan, Korea (South), Jun. 2018, pp. 123–128, https://doi.org/10.1109/SNPD.2018.8441073.

[3] C. Li, Y. Bi, F. Marzani, and F. Yang, "Fast FPGA prototyping for real-time image processing with very high-level synthesis," *Journal of Real-Time Image Processing*, vol. 16, no. 5, pp. 1795–1812, Oct. 2019, https://doi.org/10.1007/s11554-017-0688-1.

[4] M. B. Ayed, S. Elkosantini, and M. Abid, "An Automated Surveillance System Based on Multi-Processor and GPU Architecture," *Engineering, Technology & Applied Science Research*, vol. 7, no. 6, pp. 2319–2323, Dec. 2017, https://doi.org/10.48084/etasr.1645.

[5] Arjona, R., Baturone, I., 2020. Using Simulink HDL Coder to implement a Fingerprint Recognition Algorithm into an FPGA, in: 2020 XIV Technologies Applied to Electronics Teaching Conference (TAEE). Presented at the 2020 XIV Technologies Applied to Electronics Teaching Conference (TAEE), Porto, Portugal, https://doi.org/10.1109/TAEE46915.2020.9163790.

[6] H. Mestiri, I. Barraj, and M. Machhout, "AES High-Level SystemC Modeling using Aspect Oriented Programming Approach," *Engineering, Technology & Applied Science Research*, vol. 11, no. 1, pp. 6719–6723, Feb. 2021, https://doi.org/10.48084/etasr.3971.

[7] L. Zouari, S. Chtourou, M. B. Ayed, and S. A. Alshaya, "A Comparative Study of Computer-Aided Engineering Techniques for Robot Arm Applications," *Engineering, Technology & Applied Science Research*, vol. 10, no. 6, pp. 6526–6532, Dec. 2020, https://doi.org/10.48084/etasr.3885.

[8] T. Han, G. W. Liu, H. Cai, and B. Wang, "The face detection and location system based on Zynq," in *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, Xiamen, China, Aug. 2014, pp. 835–839, https://doi.org/10.1109/FSKD.2014.6980946.

[9] A. Alsheikhy and Y. F. Said, "Design of Embedded Vision System based on FPGA-SoC," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 10, 2019, https://doi.org/10.14569/IJACSA.2019.0101013.

[10] J. Jiang, C. Liu, and S. Ling, "An FPGA implementation for real-time edge detection," *Journal of Real-Time Image Processing*, vol. 15, no. 4, pp. 787–797, Dec. 2018, https://doi.org/10.1007/s11554-015-0521-7.

[11] R. Ghodhbani, L. Horrigue, T. Saidani, and M. Atri, "Fast FPGA Prototyping based Real-Time Image and Video Processing with High-Level Synthesis," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 2, 2020, https://doi.org/10.14569/IJACSA.2020.0110215.

[12] "HDL Coder Evaluation Reference Guide," *Mathworks*. https://nl.mathworks.com/matlabcentral/fileexchange/58941-hdl-coder-evaluation-reference-guide (accessed Dec. 08, 2021).

[13] "Accelerate Design Space Exploration Using HDL Coder Optimizations - Video," *Mathworks*. https://nl.mathworks.com/videos/accelerate-design-space-exploration-using-hdl-coder-optimizations-81998.html (accessed Dec. 08, 2021).

[14] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Glasgow, UK: Strathclyde Academic Media, 2014.

[15] *Introduction to FPGA Design with Vivado High-Level Synthesis (UG998)*. Xilinx, 2019.