Model-based Design of a High-Throughput Canny Edge Detection Accelerator on Zynq-7000 FPGA

Ahmed Alhomoud
Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Saudi Arabia
aalhomoud@nbu.edu.sa

Refka Ghodhbani
Department of Computer Sciences, Faculty of Computing and information Technology, Northern Border University, Saudi Arabia | Electronics and Micro-Electronics Laboratory, Faculty of Sciences, Monastir University, Tunisia
refka.ghodhbani@nbu.edu.sa

Taoufik Saidani
Department of Computer Sciences, Faculty of Computing and information Technology, Northern Border University, Saudi Arabia | Electronics and Micro-Electronics Laboratory, Faculty of Sciences, Monastir University, Tunisia
taoufik.saidan@nbu.edu.sa (corresponding author)

Hafedh Mahmoud Zayani
Department of Electrical Engineering, College of Engineering, Northern Border University, Saudi Arabia
hafedh.zayani@nbu.edu.sa

Yahia Said
Department of Electrical Engineering, College of Engineering, Northern Border University, Saudi Arabia
yahia.said@nbu.edu.sa

Mohamed Ben Ammar
Department of Information Systems, Faculty of Computing and Information Technology, Northern Border, Saudi Arabia
mohammed.ammar@nbu.edu.sa

Jihane Ben Slimane
Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Saudi Arabia
jehan.saleh@nbu.edu.sa

Received: 16 February 2024 | Revised: 24 February 2024 and 25 February 2024 | Accepted: 27 February 2024

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.7081

ABSTRACT

This paper presents a novel approach for fast FPGA prototyping of the Canny edge detection algorithm using High-Level Synthesis (HLS) based on the HDL Coder. Traditional RTL-based design methodologies for implementing image processing algorithms on FPGAs can be time-consuming and error-prone. HLS offers a higher level of abstraction, enabling designers to focus on algorithmic functionality while the tool automatically generates efficient hardware descriptions. This advantage was exploited by implementing

the Canny edge detection algorithm in MATLAB/Simulink and utilizing the HDL Coder to automatically convert it into synthesizable VHDL code. This design flow significantly reduces development time and complexity compared to the traditional RTL approach. The experimental results showed that the HLS-based Canny edge detector achieved real-time performance on a Xilinx FPGA platform, showcasing the effectiveness of the proposed approach for fast FPGA prototyping in image processing applications.

Keywords-FPGA; high-level synthesis; HDL coder; Canny edge detection; image processing; fast prototyping

I. INTRODUCTION

Canny edge detection is a fundamental image processing technique widely used in computer vision applications. While software implementations offer flexibility, FPGAs are preferred for real-time and resource-constrained applications due to their parallel processing capabilities. However, traditional RTL-based design methodologies for FPGAs can be time-consuming and require expertise in hardware description languages and manual optimization techniques. To address this challenge, High-Level Synthesis (HLS) has emerged as a promising approach to accelerate FPGA design by enabling algorithm implementation in familiar C/C++ or MATLAB/Simulink environments [1-2].

In the realm of computer vision, where raw pixels transform into meaningful insights, edge detection reigns as a fundamental image processing technique [3-4]. It acts as a cartographer, meticulously tracing the boundaries between objects, shapes, and textures, revealing the very essence of a scene. Within this domain, Canny edge detection stands as a beacon of precision and efficiency, with its algorithm meticulously crafting intricate edge maps with unparalleled clarity [3-5]. However, harnessing the power of Canny edge detection for real-time applications often presents a conundrum. Although software implementations offer flexibility, their computational demands can lag behind the lightning-fast pace of real-time processing. FPGAs boast parallel processing prowess, as their ability to crunch numbers at blistering speeds, which makes them ideal candidates for image processing tasks, particularly when latency matters [6]. Unlocking the full potential of FPGAs for Canny edge detection can be a laborious endeavor. Traditional RTL-based design methodologies, while offering fine-grained control, demand expertise in hardware description languages and meticulous hand-coding, often extending development timelines and introducing the risk of human error [4].

High-Level Synthesis (HLS) is a revolutionary approach. Bypassing the intricacies of RTL, HLS can express the Canny edge detection algorithm in familiar environments, such as MATLAB/Simulink, leveraging intuitive visual blocks and pre-built image processing libraries. HDL Coder, the interpreter within HLS, transforms this high-level description into synthesizable VHDL or Verilog code, effortlessly bridging the gap between algorithm and hardware [7]. This study showcases the transformative power of HLS for fast FPGA prototyping of Canny edge detection, by proposing a streamlined design flow, charting a course from algorithm modeling in MATLAB/Simulink to real-time edge detection on an FPGA platform. This research aims to evaluate the advantages of real-time performance, resource efficiency, and development time of the proposed approach, contrasting it with the traditional RTL paradigm [7]. More than just entailing technical advancements, this investigation unlocks exciting possibilities for the future of real-time image processing applications. From autonomous vehicles to medical imaging systems that reveal previously unseen details, the swift and precise edges identified by the proposed implementation of HLS-powered Canny edge detection hold the promise of transforming diverse fields with their clarity and insight [4].

II. EMBEDDED IMAGE AND VIDEO SYSTEM DESIGN BASED ON ZYNQ7000

The system used Zedboard, a Zynq-7000 development board, an Avnet FMC-HDMI-CAM module, and a Python 1300-C camera [8]. FMC stands for FPGA Mezzanine Card, a standardized interface that facilitates high-speed connections between FPGAs and peripheral devices [9]. This function enabled seamless integration of all cards equipped with the particular interface. UART communication between the system and a computer allowed for monitoring via the serial port, but the system functioned autonomously, independently of a computer connection. An integrated system consisting of a camera and a display constitutes a comprehensive vision system, facilitating the capture of video data. The system can process the data in real-time, depending on the specific application it is designed for. This study proposes an embedded architecture for the acquisition of video and processing modules in the design of a vision system, using a platform-based design method. This enables the seamless transmission of live video data from the sensor to the monitor using DDR3 memory and FPGA-based hardware processing. Figure 1 shows the design of the proposed system. The system consists of a VITA-2000 image sensor, an FMC module [10], the Xilinx ZedBoard platform, and an HDMI monitor to display the video output. These boards facilitate the software and hardware development of applications that are built on the Xilinx circuit from the Zynq family. The circuit utilizes an All-Programmable System-on-Chip (AP SoC) design, with a dual-core ARM™ Cortex-A9™ processor coupled with a Xilinx FPGA. This platform provides all the necessary resources to design a multitude of high-efficiency applications in the domains of video processing, motor control, software acceleration, and even system creation using embedded Linux [2]. The Zedboard has buttons and switches that allow the developer to select the desired filter and other parameters. Figure 1 displays the overall structure of the system. The heart of the system used in this study is the Zedboard development board, featuring a Zynq-7020 series SoC and various peripherals [10]. High-speed Advanced Expandable Interface (AXI) buses seamlessly bridge the gap between the CPU and FPGA, enabling their synergistic cooperation for dedicated tasks and defying their usual segregation.

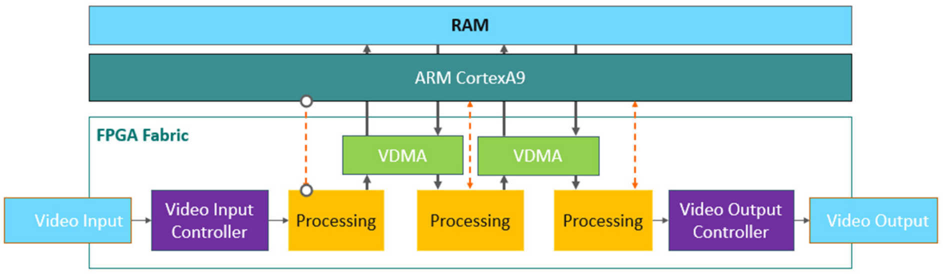


Fig. 1. General schematic of the designed system.

A. Model Design Based on HDL Coder Methodology

Model-Based Design (MBD) has the power to virtually build and test systems before constructing them in the real world, a concept explored in this study, focusing on the versatile HDL Coder from MathWorks. Although LabVIEW from National Instruments offers another MBD option, it lacks dedicated support for vendor-specific HLTs [11-12]. In contrast to these specialized HLTs optimized for specific hardware, such as filter chains through hardwired blocks, HDL Coder takes a holistic approach by analyzing the entire design and prioritizing functionality over platform limitations. From MATLAB functions, Simulink models, and Stateflow charts, it effortlessly generates portable and synthesizable Verilog and VHDL code, unlocking a world of possibilities from FPGA programming and ASIC prototyping to hardware/software co-design on platforms such as Xilinx Zynq and Intel SoC (complete with generated HDL and C code for embedded CPUs) [1]. HDL Coder goes beyond mere code generation, as its workflow advisor streamlines FPGA programming for platforms such as Xilinx, Microsemi, and Intel. The designer can gain granular control over the HDL architecture and implementation, identifying critical paths and estimating resource utilization. Furthermore, seamless traceability between the Simulink model and the generated code enables rigorous verification, even meeting the strict standards of DO-254 and beyond [7].

Unlike vendor-specific tools that are burdened with a plethora of detailed configurations, often requiring deep hardware knowledge, HDL Coder takes a simpler approach. Its streamlined interface drastically reduces design time and makes it the go-to solution for rapid FPGA prototyping. Whether designing intricate filter chains or complex SoC systems, HDL Coder provides the ability to explore, refine, and deploy designs with unprecedented efficiency and flexibility [2]. In essence, HDL Coder is not just a tool but a design philosophy. Mathworks offers two workflows for Zynq programming: Embedded Coder Hardware Support Package and SoC Blockset. Although both allow communication between Zynq's FPGA and ARM processors, their model construction and capabilities differ [7].

SoC Blockset uses separate Simulink models for each component (FPGA/processor), qualifying fine-grained control. A third model is required for communication, explicitly defining interfaces. This approach allows advanced features, such as direct memory access and interrupts, but can be unreliable for tasks within tight timeframes (<100 µs). Mathworks support confirms this limitation, making it less ideal for power electronics prototyping [13-17]. The Embedded Coder Hardware Support Package implements all code (FPGA and processor) in a single Simulink model, simplifying model management and communication. While basic communication channels are available, the trade-off is less flexibility and control.

Before building the physical components of the system, it is recommended to create a prototype design that can capture camera images [13, 15]. Following the correct acquisition of the camera picture, the filters are created and simulated within the Matlab/Simulink environment. The filters are then enclosed as regular IP blocks with the HDL Coder utility [16]. To incorporate IP blocks into the camera reference design, they need to be integrated into the Vivado environment. The Xilinx SDK environment was eventually used to write the system's software. Subsequently, the required drivers were incorporated into the system, culminating in its final configuration [11]. Building a robust system requires careful planning and execution. Here is how the referenced work tackled it:

1. Reference design: Before stepping into hardware, a blueprint was established, which is a reference design that encompasses camera image recording capabilities [10]. This ensured that all components and functionalities were clearly defined before physical construction.
2. Filter design and simulation: Capturing quality images is crucial. Therefore, the next step involved designing and simulating filters within the Matlab/Simulink environment. These filters, which act as image-processing algorithms, were fine-tuned to enhance the captured footage [5].
3. IP block encapsulation: To seamlessly integrate the designed filters with the hardware, they were packaged as reusable IP blocks using HDL Coder. This standardized format facilitated easy incorporation into the camera reference design within the Vivado environment [7].
4. Software development and integration: The system's software was then developed using the Xilinx SDK. Finally, essential drivers were added, finalizing the system's configuration and making it ready for action [10].
5. Visualization: Although diagrams and figures were not provided in the original text, incorporating visuals related to specific stages of the workflow could further enhance the reader's understanding. For example, an image depicting the reference design or a block diagram illustrating the filter

development and integration process could be helpful additions.

The overwhelming majority of contemporary chip systems use integrated processors. When designing a chip with numerous ICs, it is necessary to install additional software or firmware to handle the concurrent operations of the microprocessors, DSPs, memory, and personalized logic. With the use of a common functional specification, automated HLS allows architects and designers to explore different algorithmic and implementation choices, and evaluate and enhance the tradeoffs associated with space, power, and performance. The implementation of commercial HLS technology is now a more plausible prospect due to advancements in RTL synthesis techniques. Prominent semiconductor design companies, including IBM, Motorola, Philips, and Siemens, have presented exclusive software tools. Leading Electronic Design Automation (EDA) companies support a range of HLS products. An instance of developing RTL implementations using behavioral HDL code and linking to subsequent tools is demonstrated by Synopsys's "Behavioral Compiler" tool, which was introduced in 1995 [16]. Other comparable tools include Mentor Graphics' Catapult HLS [4] and Cadence's Stratus High Level Synthesis [18].

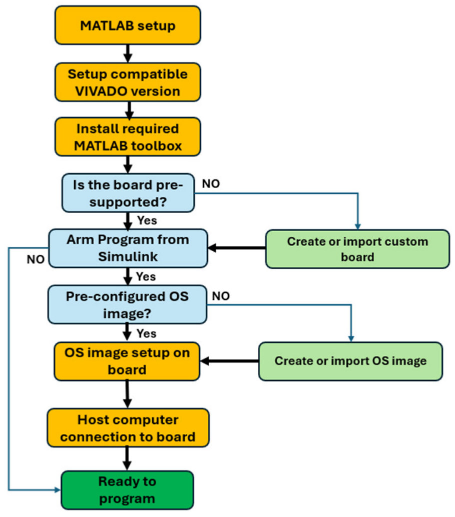
B. *Development Workflow Setup*

Development requires the appropriate configuration of many tools and components. Figure 2 presents the general configuration procedure. The following software is required to transfer control algorithms built in Matlab/Simulink to Zynq-based platforms:

- Simulink/HDL coder and embedded coder tools in Matlab/Simulink.
- Xilinx Vivado, using the free Webpack if the selected board is supported, else System Edition is advised.

This study employed Matlab R2018b and Vivado 2017.4 [7]. The System Generator tool is unable to officially support the necessary version of Matlab for HDL Coder compatibility when utilizing Vivado System Edition. Hence, it is imperative to manually adjust the list of compatible versions to ensure its compatibility with Matlab. As it will immediately establish a connection to Xilinx tools, it is recommended that Matlab be started using the System Generator tool when Vivado System Edition is being utilized. On the other hand, the HDL generation tool necessitates manual configuration every time Matlab is initiated by using the command hdlsetuptoolpath('ToolName', 'XilinxVivado', 'ToolPath', 'C:\Xilinx\Vivado\2017.4\bin\vivado.bat'). The following Matlab Add-Ons must be installed from the Add-On Explorer:

- The HDL Coder Extension packages for the FPGA Zynq-7000 Platform.
- Xilinx Zynq-7000 Platform Embedded Coder Support package.
- A compatible compiler (MinGW-w64 is a recommended option, available for direct installation from the Toolbox Add-On Explorer).



MATLAB setup
Setup compatible VIVADO version
Install required MATLAB toolbox
Is the board pre-supported?
NO
Yes
Arm Program from Simulink
Create or import custom board
NO
Yes
Pre-configured OS image?
NO
Yes
OS image setup on board
Create or import OS image
Host computer connection to board
Ready to program

Fig. 1. System design workflow setup.

III. DESIGN AND SIMULATION OF CANNY EDGE DETECTION IN SIMULINK HDL CODER

A. *Conventional Canny Edge Detection Algorithm*

The Canny edge detection algorithm is highly popular for its robust performance in detecting edges, particularly in the presence of visual noise. Its demanding computing requirements result in the need for high clock frequencies and substantial power consumption on typical microprocessor architectures, especially when real-time constraints must be satisfied. Figure 3 illustrates the stages of the Canny edge detector to identify the texture of edges. The Canny edge detector uses a Gaussian blur mask to effectively smooth and filter out noise. The next step involves convolving the picture utilizing partial derivatives of a 2D Gaussian function, Gx and Gy, to obtain the edge direction edge *Angle* matrix and the strength matrix G. The G and edge *Angle* matrices are employed in the nonmaxima suppression operation. The determination of the local maximum involves a comparison of the pixel with its neighboring pixels in the direction of the gradient. The pixel is removed if it does not have a local maximum gradient magnitude. Along the gradient's direction, a comparison is performed between the real pixel and its neighbors to eliminate any extraneous pixels that are not part of the edge. Hysteresis thresholding is the final stage, which employs a high and a low threshold. Edge pixels with intensities exceeding the high threshold are classified as strong, while edge pixels with intensities below the low threshold are suppressed. Edge pixels falling between the two thresholds are classified as weak. These actions will result in a subtle boundary in the resulting image.

Canny devised a method to obtain an ideal edge detector for handling step edges that are distorted by white Gaussian noise. Figure 3 provides a concise overview of the initial Canny edge detection algorithm [5] for an input image with dimensions $M\times M$ and each pixel represented by n bits. The process consists of the following stages:

1. Using gradient masks, convolve each pixel position to determine the horizontal and vertical gradients Gx and Gy.

2. Calculate the magnitude G and the direction θG of the gradient at every pixel position.
3. NMS: Restore crisp edges in an image with blurred edges and eliminate minima while keeping local maxima in a gradient image.
4. Threshold calculation involves determining potential edges using both high and low thresholds. These thresholds are derived based on the histogram of the gradient magnitude of the entire image.

Hysteresis thresholding generates a smooth edge map by comparing the gradient magnitude values of each pixel with predefined low- and high-threshold values. This process eliminates border pixels resulting from noise and fluctuations in lighting conditions.

B. Simulink HDL Coder Design

This study applied filters within the HDL Coder environment to design the video system. Each HDL Toolbox block was designed individually. Initially, MATLAB was transformed into a subsystem. After performing simulations, the HDL work IP blocks that were compatible with the flux advisor were transformed. Four unique filter designs, all of which can be simultaneously activated in the system, were consolidated into two IP blocks to achieve a size reduction. By doing so, it is possible to simultaneously activate all filters without adding any complication to the design [7].

C. Design Synthesis and Results

Once finished, each filter system was individually transformed into HDL code using the Matlab/Simulink HDL Coder and HDL Workflow Advisor add-on. The Matlab and Simulink systems were transformed into HDL code and the required settings were configured throughout the conversion procedure. The system was transformed into an IP block within the Vivado Suite Camera reference. This IP block was connected to the design. Consequently, filter systems were built sequentially on the Zedboard. In the future, the system will have reduced on-chip components to optimize space utilization and streamline software complexity. The IP blocks that were first built were redesigned and integrated into Simulink [8]. The edge detection algorithm was integrated into a unified IP block. The second IP block embedded sharpening and median filters. The system was reassembled and all necessary connections were made. Figure 4 portrays the schematic representation of the fully implemented system, indicating the created IP blocks.

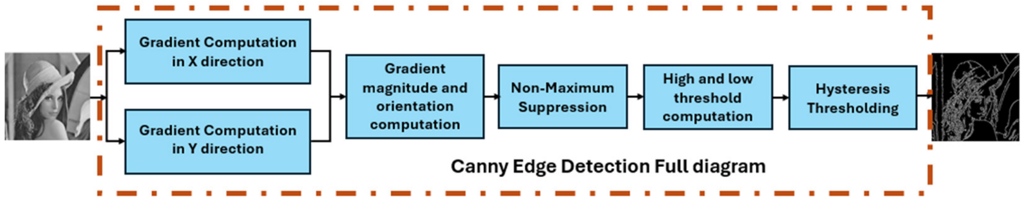

Gradient Computation in X direction
Gradient Computation in Y direction
Gradient magnitude and orientation computation
Non-Maximum Suppression
High and low threshold computation
Hysteresis Thresholding
Canny Edge Detection Full diagram

Fig. 2. Block diagram of the Canny edge detection algorithm.

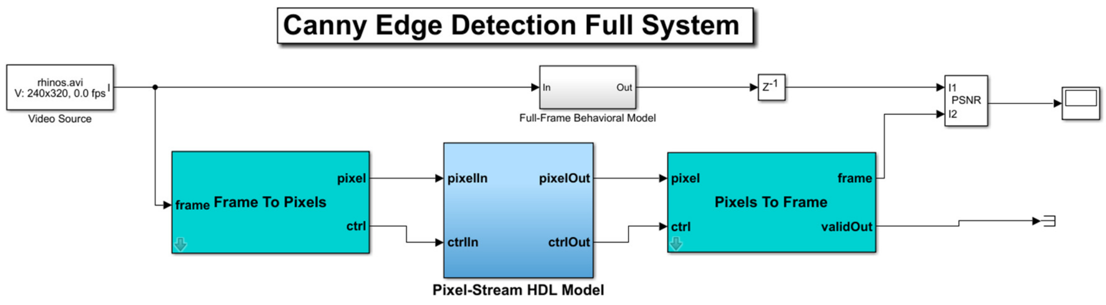


Fig. 3. HDL coder for Canny edge detector.

D. Simulation Results

The Vivado xSim tool was utilized to simulate the non-synthesisable testbench employing the generated VHDL RTL code. The reference and recommended approaches yield identical pixel values. Additionally, it was discovered that they were indistinguishable from the high-level simulation results achieved using MATLAB on the optimal bit-width model. This was demonstrated by performing a comparison of the resulting photographs generated by both paths utilizing the identical input image. Additionally, the quantization error that resulted from selecting the narrowed "optimal" signal widths was contrasted with that of the MATLAB-based double precision model. This was achieved using the "FPGA in the Loop" co-simulation feature of the MathWorks' HDL Verifier [7].

E. Synthesis Results

Table I presents the allocation of system resources. A maximum of 12% of the available resources were utilized. Furthermore, it is important to mention that no optimization efforts were undertaken in this context. The system's primary clock speed was set at 150 MHz, while the camera and image processing blocks operated at a pixel clock speed of 110 MHz. These findings demonstrate that the proposed image processing system is capable of functioning at an image resolution of 1280×1024 with a refresh rate of 60 Hz, allowing it to process

60 images per second [11]. This shows that the system is capable of satisfying the operational requirements in real-time.

1) *Implementation of the System Software*

Once the hardware design was completed, the required software was developed in parallel with the hardware. The design was executed in the C programming language using the Vivado Suite SDK tool. The software prepares and runs the IP blocks and modifies the filter and input-output unit parameters. This technology allows for real-time adjustment of filters, and numerous filters can be simultaneously active. The Zedboard's system functions can be accessed via its built-in buttons and switches. Moreover, the LEDs on the Zedboard can be engaged to monitor the functioning status of the filters. Multiplexer logic was put into service to configure the system to select the appropriate filter depending on the Zedboard switches' states. The system was programmed to automatically activate the appropriate filter according to the condition of the keys.

TABLE I. PROPOSED CANNY EDGE DETECTION SYSTEM'S IMPLEMENTATION RESULTS

	Bit Depth	**LUT-FF Pairs**	**LUTs as Logic**	**LUTs as Memory**	**Slice Registers**	**fmax**	**FPS at 1920×1080**
Number	3x8	4740	2380	1930	1920	190MHz	89

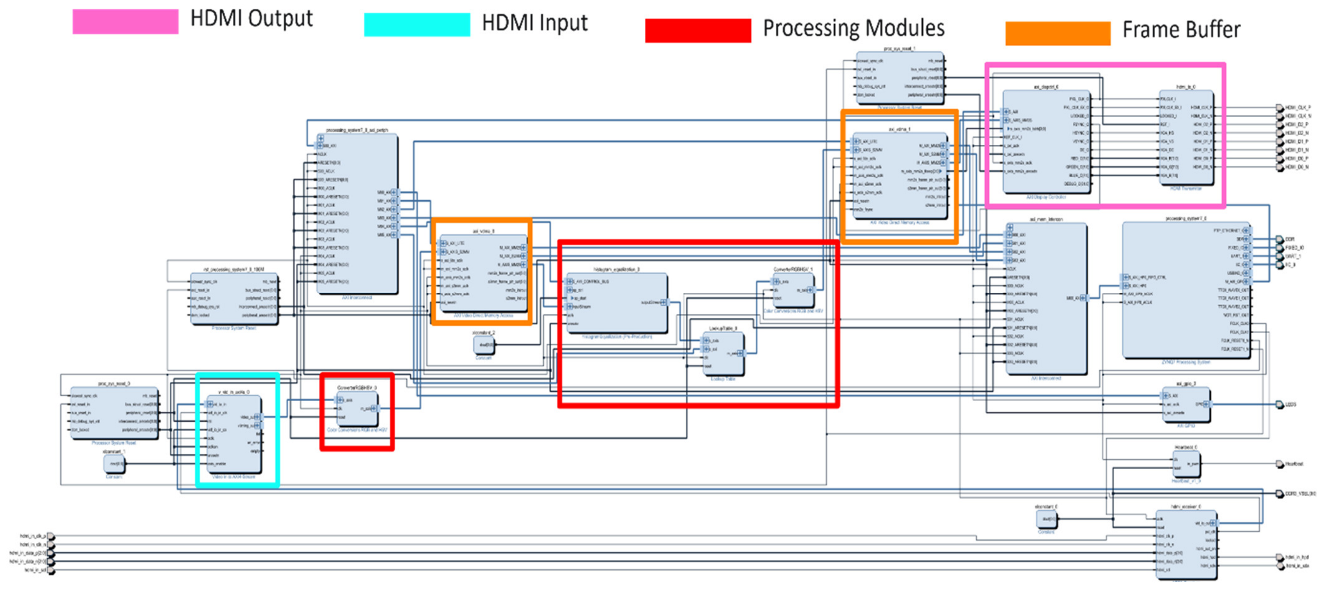


Fig. 4. RTL design of the proposed system based on Zynq 7000.

IV. CONCLUSION

This study uses a model-based real-time image processing system running on the Zynq 7000 development board. This study's hardware and software were developed applying the Zynq 7000 architecture due to the high processing power requirements. The Vivado Suite and related tools were employed for the system's hardware and software design, while Matlab/Simulink and Mathworks' HDL Coder and Vision HDL Toolbox were utilized for the design of the image processing systems. By adopting this approach, there is no requirement to develop HDL code manually, resulting in a reduction in design time. A Zedboard development board was used to design a system capable of operating at a frequency of 60 Hz and a resolution of 1280×1024 pixels. The system permits controlling the Canny edge detection algorithm for video processing through the input and output units on the card. This results in a versatile system that can easily adjust the settings. The designed system was capable of satisfying the requirements and operating in real-time. Nevertheless, resource utilization is believed to be rather low. The usage might be even more reduced, though, if the necessary optimizations are implemented in the HDL Coder tool. The proposed concept was successfully implemented with full reusability. By adopting this approach, it will be feasible to reuse IP pieces or the entire system developed in subsequent research and across diverse systems. This technology can be utilized as a preprocessing unit to achieve more sophisticated real-time object detection and tracking applications in future designs. The implementation of hardware acceleration for corner edge detection was carried out on the Xilinx Zynq-7000 SoC hardware platform, specifically for image resolutions of 1920×1080. The simulation and synthesis were acquired employing Vivado 2017.4.

ACKNOWLEDGMENT

The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA for funding this research work through project number "NBU-FFR-2024-2225-01".

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 779–788, https://doi.org/10.1109/CVPR.2016.91.

[2] R. Ghodhbani, T. Saidani, A. Alhomoud, A. Alshammari, and R. Ahmed, "Real Time FPGA Implementation of an Efficient High Speed Harris Corner Detection Algorithm Based on High-Level Synthesis," *Engineering, Technology & Applied Science Research*, vol. 13, no. 6, pp. 12169–12174, Dec. 2023, https://doi.org/10.48084/etasr.6406.

[3] T. Saidani and R. Ghodhbani, "Hardware Acceleration of Video Edge Detection with Hight Level Synthesis on the Xilinx Zynq Platform," *Engineering, Technology & Applied Science Research*, vol. 12, no. 1, pp. 8007–8012, Feb. 2022, https://doi.org/10.48084/etasr.4615.

[4] T. Saidani, R. Ghodhbani, A. Alhomoud, A. Alshammari, H. Zayani, and M. B. Ammar, "Hardware Acceleration for Object Detection using YOLOv5 Deep Learning Algorithm on Xilinx Zynq FPGA Platform," *Engineering, Technology & Applied Science Research*, vol. 14, no. 1, pp. 13066–13071, Feb. 2024, https://doi.org/10.48084/etasr.6761.

[5] CongJason *et al.*, "FPGA HLS Today: Successes, Challenges, and Opportunities," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, Aug. 2022, https://doi.org/10.1145/3530775.

[6] Z. Tan and J. S. Smith, "Real-time Canny Edge Detection on FPGAs using High-level Synthesis," in *2020 7th International Conference on Information Science and Control Engineering (ICISCE)*, Sep. 2020, pp. 1068–1071, https://doi.org/10.1109/ICISCE50968.2020.00217.

[7] A. Fuentes-Alventosa, J. Gómez-Luna, and R. Medina-Carnicer, "GUD-Canny: a real-time GPU-based unsupervised and distributed Canny edge detector," *Journal of Real-Time Image Processing*, vol. 19, no. 3, pp. 591–605, Jun. 2022, https://doi.org/10.1007/s11554-022-01208-0.

[8] F. Siddiqui *et al.*, "FPGA-Based Processor Acceleration for Image Processing Applications," *Journal of Imaging*, vol. 5, no. 1, Jan. 2019, Art. no. 16, https://doi.org/10.3390/jimaging5010016.

[9] P. Babu and E. Parthasarathy, "Hardware acceleration for object detection using YOLOv4 algorithm on Xilinx Zynq platform," *Journal of Real-Time Image Processing*, vol. 19, no. 5, pp. 931–940, Oct. 2022, https://doi.org/10.1007/s11554-022-01234-y.

[10] F. N. Taher, M. Kishani, and B. C. Schafer, "Design and Optimization of Reliable Hardware Accelerators: Leveraging the Advantages of High-Level Synthesis," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, Platja d'Aro, Spain, Jul. 2018, pp. 232–235, https://doi.org/10.1109/IOLTS.2018.8474222.

[11] "AXI4-Stream Video IP and System Design Guide," Xilinx, UG934, Oct. 2019.

[12] Stephen Neuendorffer, Thomas Li, and Devin Wang, "Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries," Xilinx, Aug. 2013.

[13] K. Kintali and Y. Gu, "Model-Based Design with Simulink, HDL Coder, and Xilinx System Generator for DSP," Mathworks, 2015.

[14] S. Liu *et al.*, "Real-time implementation of harris corner detection system based on FPGA," in *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, Jul. 2017, pp. 339–343, https://doi.org/10.1109/RCAR.2017.8311884.

[15] S. Chumpol, P. Solod, K. Thongnoo, and N. Jindapetch, "Model-Based Design Optimization using CDFG for Image Processing on FPGA," *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, vol. 17, no. 4, pp. 479–487, Oct. 2023, https://doi.org/10.37936/ecti-cit.2023174.252417.

[16] S. Titri, C. Larbes, and K. Y. Toumi, "Rapid prototyping of PVS into FPGA: From model based design to FPGA/ASICs implementation," in *2014 9th International Design and Test Symposium (IDT)*, Algeries, Algeria, Dec. 2014, pp. 162–167, https://doi.org/10.1109/IDT.2014.7038606.

[17] I. El Hajjouji, S. Mars, Z. Asrih, and A. El Mourabit, "A novel FPGA implementation of Hough Transform for straight lane detection," *Engineering Science and Technology, an International Journal*, vol. 23, no. 2, pp. 274–280, Apr. 2020, https://doi.org/10.1016/j.jestch.2019.05.008.