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

An Efficient Computer Vision AI Powered Application based Fast Harris Corner Detection Accelerator on Zynq-7000 FPGA

Taoufik Saidani[1],Mohammad H Algarni[2]

Submitted: 29/01/2024 **Revised**: 07/03/2024 **Accepted**: 15/03/2024

Abstract: Tools for rapid prototyping have been crucial in the race to market. A frequently-used computer vision, object detection and Artificial Intelligence tool for image processing, real-time contrast enhancement, is the focus of our study as we investigate the possibility of developing an intellectual property core using a fast prototyping technique. In this research paper, we provide a new method that uses HDL Coder-based high-level synthesis (HLS) rapidly to prototype fast corner detection on FPGAs. Implementing image processing algorithms on FPGAs using traditional RTL-based design approaches may prove a tedious, error-prone procedure. By providing a more abstract level of description, HLS frees up designers to concentrate on algorithmic functionality rather than writing inefficient hardware specifications by hand. We employ this feature by using the Harris corner detection method in MATLAB/Simulink, then using the HDL Coder methodology automatically to transform it into generated VHDL code that can be synthesized on Zynq7000 FPGA. Compared to the conventional RTL method, this design flow drastically reduces the development time and complexity. The suggested method for rapid FPGA prototyping in image processing applications is demonstrated to be successful by our practical findings, which indicate that the HLS-based Harris corner detector achieves the performance of real time video processing on a Xilinx Zynq7000 FPGA platform.

Keywords: *HDL, HDL Coder, Fast Harris corner detection, Computer vision, Artificial Intelligence, Object Detection*

1. Introduction

For various computer vision applications, including object localization, video tracking, motion estimation, image retrieval, and object detection, corner detection is an essential pre-processing step. Accurate, repeatable corner detection is critical for real-time computer vision applications, such as online visual localization, motion estimation, and 3D reconstruction. Although software solutions provide greater leeway, the parallel processing capabilities of FPGAs make them the go-to choice for applications that require a real-time response and have limited resources. It may require huge efforts, a knowledge of hardware description languages, and manual optimization approaches to apply typical RTL-based design processes to FPGAs. A potential solution to this problem is high-level synthesis (HLS), which allows algorithm implementation in well-known environments, like C/C++ or MATLAB/Simulink, thereby speeding up FPGA design [1,2].

Mobile intelligent systems, especially the recent mobility aids for the visually impaired, rely on corner or point of interest recognition as a critical vision operation [1]. Without doubt, when it comes to the safety of the visually impaired, the computational and spatial complexity of embedded and wearable systems in real time are crucial characteristics. Thus, it is important, when building such systems, to tailor their complexity level to suit the desired system's performance. Due to its superior accuracy, high quality identified corners, and invariance to scale and rotation geometric transformations, the Harris-Stephen method is widely employed for corner recognition [2], but the algorithm's temporal complexity is its biggest flaw. Numerous laborious procedures must be applied to each picture pixel (pixel level processing), such as gradient derivatives, Gaussian smoothing, Harris response corner calculation, etc. Since real-time constraints are a significant concern, it may not be feasible to implement software-based sequential solutions [3].

By using field-programmable gate arrays (FPGAs), it is possible to shorten the development cycle of modern designs and decrease the computational complexity of any operation. Additionally, FPGA-based boards facilitate the discovery of the optimal design for the intended application in terms of software and hardware sufficiency. Indeed, various hardware-software partition strategies may be used, resulting in systems with varying processing speeds and lower resource needs, but a new way to create vision systems is to use FPGAs for complicated processing, such as corner detection.

Boards constructed around the latest generation of field-programmable gate array (FPGA) devices allow the implementation of various hardware/software partition strategies, leading to varying degrees of performance [4-5]. Utilizing the accessible parallelism of computational resources, FPGA SoCs (Systems on Chip) enable the implementation of parallel applications with various

[1]*Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha, Saudi Arabia*
** Taoufik.Saidan@nbu.edu.sa*
[2]*Department of Computer Science,Al-Baha University, Saudi Arabia*
** malgarni@bu.edu.sa*

programming models (e.g., grids, trees, and pyramids) and whole processing control modes (e.g., SIMD, true MIMD, and variations thereof). Additionally, BRAMs allow for hard real-time processing, and DDR3 SDRAM has an extremely large memory (up to one gigabyte). Sustainable design is aided by these technologies, which also create new markets for large data processing, image processing, and computer vision in particular [6-7].

A new paradigm, High-Level Synthesis (HLS), has arrived, bringing with it a new way of thinking and doing things. Bypassing the complexity of RTL, HLS allows us to write the Harris corner detector approach in MATLAB/Simulink or any familiar Tools environments, utilizing simple visual blocks and prebuilt image processing tools. The translator in HLS, the HDL Coder, efficiently converts this high level picture into synthesizable generated VHDL or Verilog code, thereby connecting the gap between the algorithms and the hardware [9].

This research seeks to demonstrate that HLS can revolutionize fast FPGA detection prototyping. Using MATLAB/Simulink for the algorithm modeling and an FPGA platform for the real-time edge detection, we propose a streamlined design process. By running thorough tests, we wish to demonstrate how our approach outperforms the traditional RTL paradigm in terms of development speed, resource use, and real-time performance [10].

The subsequent sections of this work are structured as follows: the following section. Section 2 introduces a sophisticated synthesis technique for picture and video prototyping using a model-based design with HDL coder. It addresses the challenges encountered during the prototyping process and provides corresponding solutions to these. The proposed method's prototype and experiment results are shown in Section 3. Ultimately, this work is concluded in Section 4.

2. Real Time image and video system design based on the Zyn7000 Platform

According to [11], the Zedboard, which is a Zynq-7000 development board, the Avnet FMC-HDMI-CAM module, and a Python 1300-C camera were all confirmed to be mutually compatible. The "FPGA Mezzanine Card," or FMC, is a kind of standardized interface that enables fast connections between FPGAs and other devices [12]. Since this interface is universal, all of the cards using it can work together seamlessly. It is possible to monitor the system over the serial port using an UART communication with a computer, or it can be run as standalone.

To facilitate the recording of video data, an integrated video system was employed, consisting of a display and a camera, called a comprehensive vision system. The system's capacity to process data in real-time depends on the application domain. In order to develop a vision system, this research suggests a platform-based architecture that incorporates processing modules and video capture. Using DDR3 memory and FPGA-based hardware processing, this method allows the sensor to transmit live video data to the display seamlessly [13]. The video processing system, based on an embedded system architecture, is shown in Figure 1. An HDMI display for viewing the video output, the Xilinx ZedBoard platform, an image sensor from VITA-2000, an FMC module, and the Xilinx ZedBoard platform, comprise the system [14].

Application development for the XilinxTM Zynq family of integrated circuits is facilitated by the use of these boards. Equipped with a XilinxTM FPGA and a dual-core ARMTM Cortex-A9TM processor, the circuit makes use of an "AP SoC" (All Programmable System-on-Chip) architecture. Armed with this equipment, it is possible to create a plethora of efficient applications in fields like embedded Linux, motor control, software acceleration, and video processing [15].

The filter and other settings may be changed using the buttons and switches on the Zedboard. The general layout of the system is shown in Figure 1.

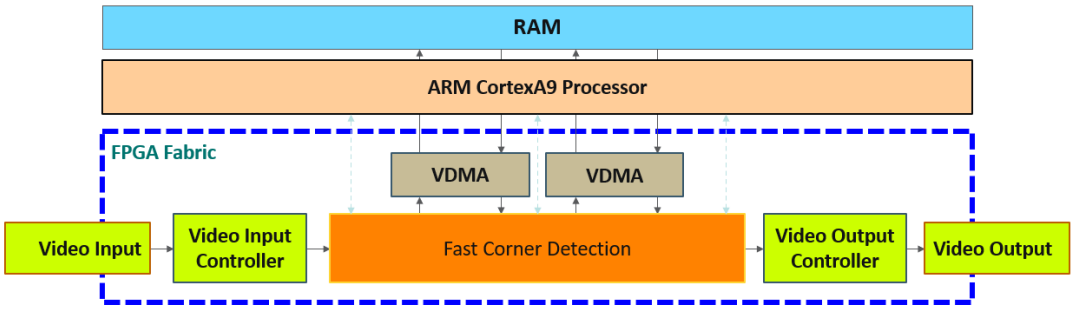

Fig. 1. Full presentation of the video processing system based on the Fast Corner Detectio System

An integrated circuit (ICU) from the Zynq-7020 family and other peripherals constitute the Zedboard experimental board, which is the heart of the experiment [11]. Zynq-7000 is a revolutionary microprocessor that combines a dual-core ARM Cortex-A9 CPU with FPGA fabrics on a single die. Instead of being typically isolated, high-speed Advanced Interface (AXI) buses allow the central processing unit (CPU) and field-programmable gate array (FPGA) to work together seamlessly to complete certain tasks. Figure 2 shows the basic layout of the Zynq architecture [9,11].

2.1. FPGA Model Based Design

The capacity digitally to design and test systems prior to physically constructing them is the foundation of the Model-Based Design (MBD) methodology. In this thesis, we explore this concept using several HDLs, with a focus on the adaptable MathWorks HDL Coder.

While these dedicated HLTs are optimized to suit particular hardware, such as filter chains using hardwired blocks, the HDL Coder provides a more comprehensive way, which ensures that functionality takes precedence over platform constraints. It easily produces synthesizable, portable Verilog and VHDL code from MATLAB library functions, Simulink models, and workflow charts. From scripting FPGAs and analog-to-digital (ASIC) prototypes to hardware-software co-design on platforms like an Intel System on a Chip (SoC) and Xilinx Zynq, this code opens up a world of possibilities [1].

The power of HDL Coder extends well beyond the creation of code, however. For Xilinx, Microsemi, and Intel platforms, its workflow adviser simplifies FPGA programming. Pinpointing the essential pathways and calculating resource use make it possible to control the HDL design and implementation, as well as conduct a thorough verification that satisfies even the most stringent requirements, such as DO-254 and beyond, thanks to the code's seamless traceability with the Simulink model [7].

As its name suggests, the HDL Coder is more than simply a tool—it is an approach to design. Its proponents praise its ability to allow experimentation, creation, and changes without leaving the safety of a computer. The HDL Coder allows users quickly to develop strong, adaptive systems, that expedites their FPGA development journey and uncovers the full potential of MBD,. To program with Zynq, Mathworks provides two workflows: SoC Blockset and the Embedded Coder Hardware Support Package. The model architecture and capabilities vary between the two, but they both provide connection between Zynq's FPGA and ARM CPUs [7].

SoC Blockset: For granular control, it employs individual Simulink models for the FPGA and CPU. An additional model, with well-defined interfaces, is necessary for communication. While this method enables powerful capabilities, such as Direct Memory Access and interruptions, it may be unsuitable for urgent tasks (<100 μs). It is less than optimal for power electronics prototyping, as confirmed by Mathworks support [17].

Embedded Coder Hardware Support Package: This creates a unified model in Simulink for both FPGA and processor code, thereby simplifying the model administration and communications. There are several accessible basic communication routes, but these come at the cost of control and flexibility.

It is recommended to develop a prototype design with the potential to collect camera pictures before constructing the actual system components [14, 16]. After the camera image has been correctly acquired, the filters are created and tested in the Matlab/Simulink environment, after which the HDL Coder tool is used to encapsulate the filters as standard IP blocks [17]. The IP blocks must be introduced into the Vivado environment before they can be used in the camera reference design. The system's software was finally written in the Xilinx SDK environment. The final setup of the system was achieved once the necessary drivers had been integrated into it [12].

Building a robust system needs thorough planning and execution. Here is how the referenced work approached this:

1. Reference Design:

An initial plan was drawn up for a reference design with an ability to capture images from the camera [11]. This ensured that, before anything was built, all the parts and functions were specified.

2. Filter Design and Simulation:

To improve the recorded video, the next step was to create and test the filters in the Matlab/Simulink environment. These filters, which functioned as image processing algorithms, were fine-tuned to ensure high-quality picture capture [5].

3. IP Block Encapsulation:

The proposed filters were rendered into reusable Intellectual Property (IP) blocks using the HDL Coder tool to ensure smooth hardware integration. Thanks to this uniform format, it was straightforward to include it in the Vivado environment's camera reference design [7].

4. Software Development and Integration:

The software, or central processing unit, of the system was then built using the Xilinx SDK. After finishing the machine's configuration, the required drivers were installed [11].

5.Visualization:

Pictures related to certain phases in the process were included in order to improve the readers' understanding even

if the original text did not include any graphics or diagrams. For example, pictures of the original plan or schematics of the filters' development and integration might prove helpful.

The use of HLS is increasing because it allows additional abstraction levels to be reached while still ensuring continuous verification throughout the design cycle. The open-source instruments that are used for high-level synthesis (HLS) include Vivado HLS [10] and MATLAB HDL Coder [11]. They are often used by digital architects and designers to create and execute algorithms in fields such as communications, neural networks, deep learning, image processing, and aerospace. The code complexity may be reduced by a factor of seven to ten using HLS technology. They facilitate the reuse of behavioral intellectual property (IP) across several projects and enable verification teams to apply sophisticated modeling approaches, such as transaction-level modeling [12].
In addition, integrated processors are used by the vast majority of modern chip systems. Additional software or firmware must be included when creating a chip with several ICs in order to manage the simultaneous activities of the memory, digital signal processors (DSPs), microprocessors, and customized logic. Architects and designers may experiment with various algorithmic and implementation options using Automated High-Level Synthesis (HLS) based on a shared functional specification. The space, power, and performance tradeoffs may be assessed and improved in this way.
New methods for register transfer level synthesis have made the commercial deployment of HLS technology a more realistic proposal. Some of the best-known names in semiconductor design have developed proprietary software: IBM, Motorola, Philips, and Siemens. At this time, several High-Level Synthesis (HLS) products are supported by the prominent Electronic Design Automation (EDA) companies. Introduced in 1995, Synopsys's "Behavioral Compiler" tool exemplifies the process of creating RTL implementation using behavioral hardware description language (HDL) code and integrating it with later products [17]. The "Catapult HLS" [4] from Mentor Graphics and the "Stratus High Level Synthesis" [19] from Cadence are similar tools. High-Level Synthesis (HLS) in VLSI designs follows a conventional method, as shown in Figure 2.

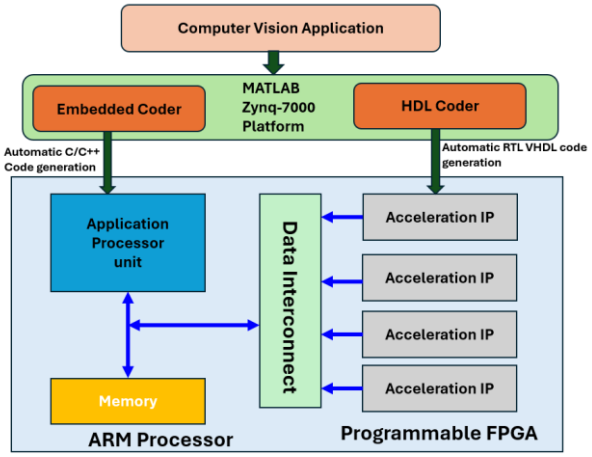

Fig. 2 : System Design Flow based on a Simulink /HDL coder

2.2. Development Workflow Setup

The development technique that has been discussed requires many tools and components in order for it to be set up correctly. Visual representations of the whole setup process are provided in Figure 3 and the following sections.

Transferring control algorithms built in Matlab/Simulink to platforms based on FPGAZynq7000 requires the following software:

- A Simulink/HDL coder toolbox and embedded coder toolbox tools in the environment Matlab/Simulink.
- Xilinx Vivado tools setup based on MATLAB Vivado compatibility

As mentioned in reference, a specific version of the Vivado Designer Suite. The results were obtained using Vivado 2017.4 and Matlab R2018b [7]. Since the "System Generator" tool is not officially compatible with the version of Matlab needed for the HDL Coder in the Vivado System Edition, to ensure that the tool works with Matlab, one must manually change the list of compatible versions. Matlab should be started using the System Generator tool when Vivado System Edition is being used, since it will instantly connect to Xilinx tools. Each time that Matlab is started using the command, however, it will be necessary to configure the HDL creation tool manually:

hdlsetuptoolpath('ToolName','XilinxVivado','ToolPath','C:\Xilinx\Vivado\2017.4\bin\vivado.bat'). The following Matlab Add-Ons must be pre-installed from the Add-On Explorer:

- The HDL Coder Extension toolbox Packages for Fpga Zynq-7000 Platform is available.
- Xilinx Zynq-7000 Platform Embedded Coder toolbox Support Package.

- You may install MinGW-w64, a compatible compiler, directly from the Toolbox Add-On Explorer. This is highly recommended.

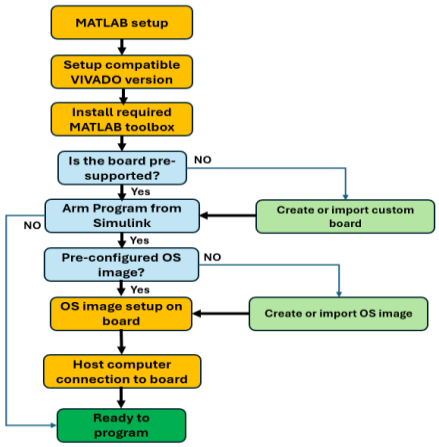

Fig. 3: System Design Wroklow Flow setup

3. HARRIS CORNER DETECTION DESIGN AND SIMULATION

3.1. OVERVIEW OF THE HARRIS CORNER DETECTION ALGORITHM

The term "corners" refers to areas of an image where the level of detail and resistance to distortion differ significantly. To find the wedges, Harris points are created according to the pixels' brightness. The following intensity variations are proposed as the basis for an autocorrelation-based detector:

$$M(x,y) = \sum_{(u,v)} w(u,v) \cdot \begin{pmatrix} I_x^2(x,y) & I_x I_y(x,y) \\ I_x I_y(x,y) & I_x^2(x,y) \end{pmatrix}$$

with Ix and Iy being the local derivatives in x and y, and w (u, v) is a weighting on the window (u, v). The study of the eigenvalues of the matrix M makes it possible to determine whether a point is a corner, a homogeneous region, or an outline. A final criterion is calculated from M, making it possible to define the type of point found.

R=Det(M)- k∗Trace(M)^2

3.2. Simulink HDL coder design

The full system (Figure 4) was designed using filters inside the Simulink/HDL coder environment, according to the paper. Every single HDL Toolbox block is custom-made. First, MATLAB was turned into a subsystem. Our team successfully modified the HDL work IP blocks to make them compatible with the flux advisor, following extensive simulations. To accomplish size reduction, the system's four separate filter designs—all of which may be enabled simultaneously—were combined into two IP blocks. In this way, the design could remain simple while all of the filters were activated at the same time [7].

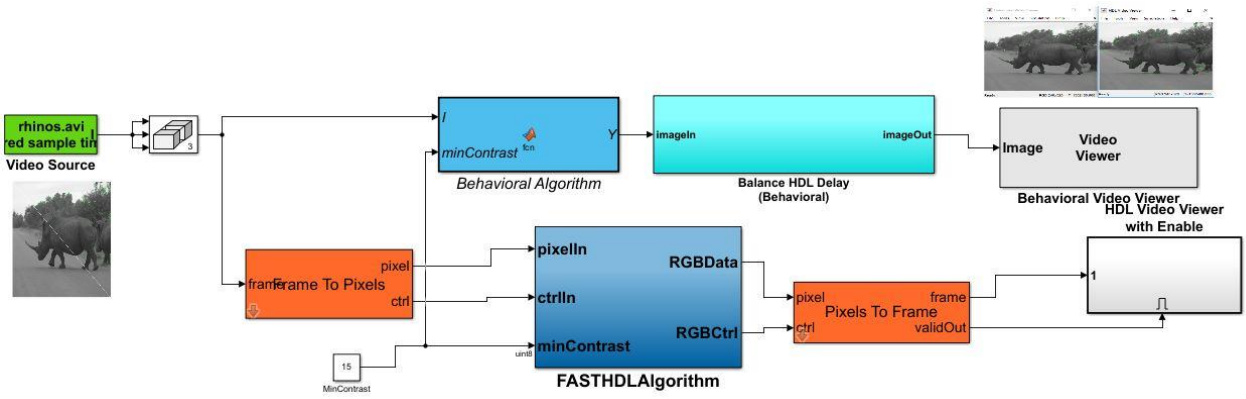

Fig. 4 :Full video processing Design based on FAST Harris Corner Detection

3.3. DESIGN SYNTHESIS AND RESULTS

Using the Simulink program, we can simulate the designs. At the end, the code for each filter system is converted into HDL code separately. To test this process, we used the HDL Workflow Advisor and Matlab/Simulink HDL Coder add-ons. A specialist in HDL workflow provided expert consulting services. During the conversion process, we set up the necessary parameters and transformed the Matlab and Simulink systems into HDL code to provide a graphical user interface. Next, according to the Vivado Suite Camera reference, the system was converted into an IP block. It is impossible to separate this IP block from the design, so the filter systems were constructed on the Zedboard in a sequential fashion. In addition, future studies might focus on optimizing the space consumption and simplifying the software complexity by reducing the number of on-chip components in the system. The originally developed IP blocks were redesigned and integrated using Simulink.

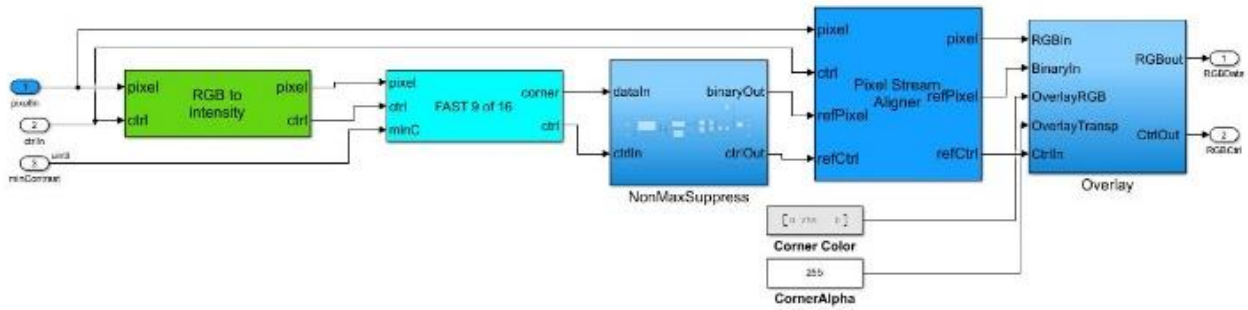

Fig. 5 :Simulink Design of FAST Harris Corner Detection

Consequently, a single IP block incorporates the smart edge detection technologies. The additional IP blocks included median and sharpening filters. All the required connections were completed and the system was reassembled. Figure 16 shows a schematic of the system that was developed using Vivado Suite. The IP blocks made specifically for this research are shown in the figure.

3.3.1 Simulation results

To model the non-synthesisable testbed, we imported the VHDL RTL code that we had created into the Vivado xSim program. Figure 6 displays the results for the functional simulations conducted using the Vivado xSim simulator. The figure shows that both the reference and suggested methods produce identical pixel values. It was also found that they were identical to the optimum bit-width model's high-level MATLAB simulation results. By comparing the output images produced by the two pathways with the same input picture, we were able to prove this. We also compared the quantization error of the MATLAB-based double precision model with that of the reduced "optimal" signal widths. The mathWorks HDL Verifier's "FPGA in the Loop" cosimulation tool was used to achieve this [7].

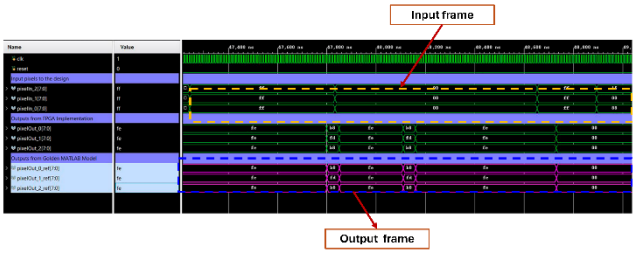

Fig. 5 :Fast Harris Corner Detection RTL Simulation

3.3.2 Synthesis Results

Table 1 displays the resource allocation following system synthesis and chip integration. Clearly, we have only used up to 12% of the resources that were available. Additionally, it should be noted that no optimization attempts were made in this particular situation. The main clock speed of the system is 150 MHz, while the image processing and camera blocks run at 110 MHz for each pixel. With the help of Formula 8, we can calculate the system's operating speed. The screen refresh rate (EYH) and total horizontal and vertical pixel values (including the active, gap, and synchronization pixels) all form part of the equation [12].

Table 1. Proposed Fast Harris corner detection Implementation Results

Resource	***Utilization***	***Available***	***Utilization (%)***
LUT RAM	502	17400	11.05 %
LUT	5900	53200	2.86%
FF	10975	106400	10.30%
BRAM	36	140	26.76%
DSP	20	220	9.50%
IOs	102	200	50%
Frame Rate	**125 FPS**		

According to the results, the image processing system can handle 125 pictures per second at a resolution of 1280 x 1024, and a refresh rate of 120 Hz. This demonstrates that the system can fulfil the operational needs in real-time.

3.3.3 Implementation of the System Software

The system's software was advanced in tandem with the hardware after the hardware design had been completed. The idea was put into action by using the C programming language and the Vivado Suite SDK tool. The software does more than simply set up the filter and I/O device; it also runs the IP blocks and prepares them for use. It is now possible to adjust the filters in real time and run several filters simultaneously, all thanks to the newest innovation. One may access the system features via the buttons and switches incorporated into the Zedboard. Furthermore, the Zedboard's LEDs make it possible to test the filters' operation. Figure 6 displays the software flow diagram for the system. The system continuously selects the principal application cycle filter, and may be set up to choose the appropriate filter depending on the zedboard switches' states using multiplexer logic. The keypad of the Zedboard is shown in Figure 18. The technology automatically activates the appropriate filter, based on the key condition. In terms of bits, the least significant bit is DS0 and the most significant bit is DS7.

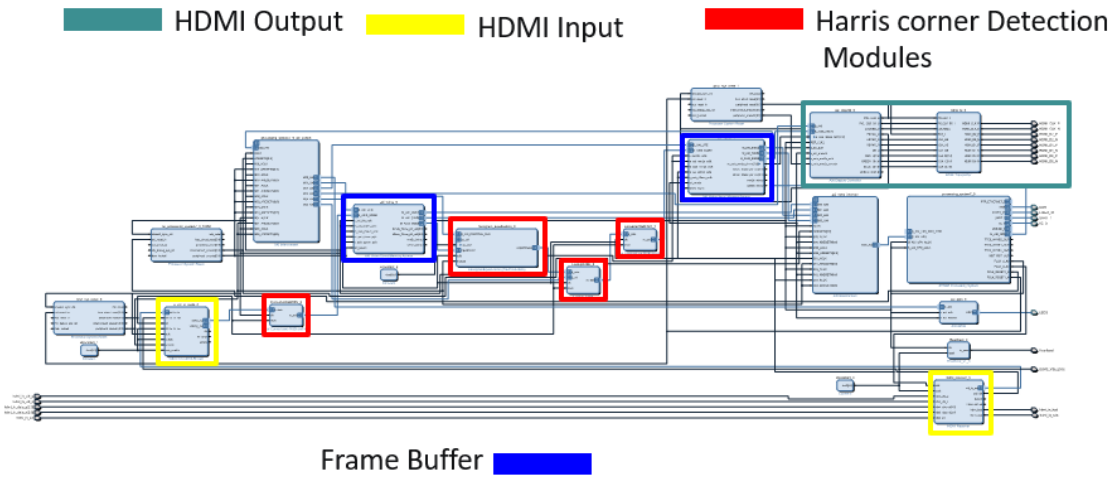

Fig. 6 :RTL Design for the proposed System based Zynq7000

4. Conclusion

This research presents the first complete approach to programming Zynq-based boards for power electronics prototyping using Matlab/Simulink, the HDL Coder, and the Embedded Coder tools. By employing the proposed method, users are able to program Zynq processors via the Simulink graphical user interface, which allows the development and monitoring of power electronics systems. Even those unfamiliar with HDL or C may use Zynq boards to construct working prototypes of power electronics systems, if they are familiar with Simulink.

The Zynq7000 development board is used in this work to power a real-time image processing system that is model-based. Due to the high processing power needs of the research, the Zynq7000 architecture was used to construct the hardware and software. The system's hardware and software were designed using Xilinx's Vivado Suite and associated tools, in compliance with Zynq. The image processing systems were designed using Matlab/Simulink and Mathworks' HDL Coder and Vision HDL Toolbox. This method shortens the design cycle, as it eliminates the need to create HDL code manually. For this research, a system that could operate at 60 Hz and have a resolution of 1280 x 1024 pixels was designed using a Zedboard development board. The system's input and output modules on the card facilitate detection for video processing. The end result is a flexible system that can be easily configured. After reviewing the designs, it was found that the system can meet the criteria and work in real-time. It is thought that the resource consumption is rather low but, if the HDL Coder tool receives the required improvements, the consumption may drop considerably. We have successfully executed the idea and ensured its complete reusability. This approach will make it possible to reuse parts of or the whole IP system in future studies, regardless of the system type. Future studies may use this technique as a building block for more complex real-time item identification and tracking applications.

The Xilinx Zynq-7000 SoC hardware platform was used to provide acceleration in hardware for corner edge detection, with a focus on a 1920 × 1080 picture definition. Vivado 2017.4 was used to obtain the results of the synthesis and simulation.

References and Footnotes

Author contributions

Taoufik Saidani: Conceptualization, Methodology, Modeling,Simulation, Paper writing, Peper formatting

Mohammad H Algarni: Conceptualization, Methodology, Modeling,Simulation, Paper writing, Peper formatting

Conflicts of interest

The authors declare no conflicts of interest.

References

[1] Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.

[2] V.H. Schulz, F.G. Bombardelli, E. Todt, A Harris corner detector implementation in SoC-FPGA for visual SLAM, in: F. Santos Osorio, ´ R. Sales Gonçalves (Eds.), Robotics. SBR 2016, LARS 2016. Communications in Computer and Information Science 619, Springer, Cham., 2016.

[3] Sui Xuyang, Chen Zhuo, Liu Yicong, et al. Real-time Video Edge Detection System Based on FPGA. Ordnance Industry Automation, 2021, 40(2): 58-60

[4] Jason Cong, Jason Lau, Gai Liu, Stephen Neuendorffer, Peichen Pan, Kees Vissers, and Zhiru Zhang. 2022. FPGA HLS today: Successes, challenges, and opportunities. ACM Trans. Reconfigurable Technol. Syst. 15, 4, Article 51 (Aug. 2022), 42 pages. https://doi.org/10.1145/3530775

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

[6] Fuentes-Alventosa, A., Gómez-Luna, J. & Medina-Carnicer, R. GUD-Canny: a real-time GPU- based unsupervised and distributed detector. J Real-Time Image Proc 19, 591–605(2022). https://doi.org/10.1007/s11554-022-01208-0

[7] 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. DOI: https://doi.org/10.48084/etasr.4615

[8] Fahad Siddiqui, Sam Amiri, Umar Ibrahim Minhas, Tiantai Deng, Roger Woods, Karen Rafferty, and Daniel Crookes. 2019. FPGA-based processor acceleration for image processing applications. Journal of Imaging 5, 1 (2019). https: //doi.org/10.3390/jimaging5010016.

[9] Babu, P.; Parthasarathy, E. Hardware acceleration for object detection using YOLOv4 algorithm on Xilinx Zynq platform. J. Real-Time Image Process. 2022, 19, 931–940.

[10] Farah Naz Taher, Mostafa Kishani, and Benjamin Carrion 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). 2018, pp. 232–235. doi: 10.1109/IOLTS.2018.8474222

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

[12] S Neuendorffer, T. Li and D. Wang, "Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries", Xilinx Inc. Application Note, June 2015.Vivado Design Suite User Guide: High-Level Synthesis," Xilinx, UG902 (v2018.3), 2018.

[13] Kintail, K., Gu, Y.: Model-based Design with Simulink, HDL Coder, and Xilinx System Generator for DSP, pp. 1–15. MathWorks, Inc (2012)

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

[15] MMathWorks, “Limitations for MATLAB Loop Optimization,” HDL coder user guide, pp. 8-30, 2023.

[16] S. Titri, C. Larbes and K. Y. Toumi, “Rapid prototyping of PVS into FPGA: From Model-Based Design to FPGA/ASICs Implementation,” 2014 9th International Design and Test Symposium (IDT), Algeries, Algeria, pp. 162-167, 2014.

[17] El Hajjouji, I., Mars, S., Asrih, Z., & El Mourabit, A. (2020). A novel FPGA implementation of hough transform for straight lane detection. Engineering Science and Technology, an International Journal, 23(2), 274-280. https://doi.org/10.1016/j.jestch.2019.05.0