EFFICIENT REAL TIME ZYNQ 7000 FPGA DEPLOYMENT OF OPTIMIZED YOLOV2 DEEP LEANING MODEL FOR TARGET DETECTION, BASED ON HDL CODER METHODOLOGY

Jihane Ben Slimane *

Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha, Saudi Arabia

* Corresponding Author, e-mail: jehan.saleh@nbu.edu.sa

Abstract: Field Programmable Gate Arrays (FPGAs) have garnered significant attention in the development and enhancement of target identification algorithms that employ YOLOv2 models and FPGAs, owing to their adaptability and user-friendliness. The Simulink HDL compiler was utilized to design, simulate, and implement our proposed design. In an effort to rectify this, this paper presents a comprehensive programming and design proposal. The implementation of the YOLOv2 algorithm for real-time vehicle detection on the Xilinx® Zynq-7000 System-on-a-chip is proposed in this work. Real-time testing of the synthesised hardware revealed that it can process Full HD video at a rate of 16 frames per second. On the Xilinx Zynq-7000 SOC, the estimated dynamic power consumption is less than 90 mW. When comparing the results of the proposed work to those of other simulations, it is observed that resource utilization is enhanced by around 204 k (75%) LUT, 305 (12%) DSP, and 224 k (41%) flip-flops at 200 MHz.

Key words: FPGA accelerator, YOLOv2 accelerator, High level synthesis, HDL coder, Vehicle detection.

1.INTRODUCTION

Target detection is a widely researched topic in the field of computer vision, with practical implications in numerous industries, including intelligent surveillance, industrial inspection, and aerial photography [1]. On the contrary to traditional algorithms [2], deep learning methodologies demonstrate enhanced accuracy and robustness in the detection of targets amidst complex scenarios [3]. When applied to visible and Synthetic Aperture Radar (SAR) images, deep learning detection algorithms have proven to be more accurate and robust than conventional algorithms when detecting targets [4]. The architecture of multithreaded and distributed software processing of graphical data [5], is proposed for improvement of system operation efficiency in real time [6]. Examples of such algorithms include You Only Look Once (YOLO) and Faster Region Convolutional Neural Networks (Faster R-CNN) [7].

Two often referenced tools in this area are Simulink Hardware Description Language (HDL) Coder [8] and Xilinx High-Level Synthesis (HLS) [9]. Due to its capacity to compare with built-in standard algorithms and its thorough functional verification [10], the latter is ideal for the creation of big computer vision systems [11]. Therefore, the HDL coder allows for the functional testing of several image processing methods and fast synthesis of many more, including custom filtering, colorspace conversion, picture statistics gathering, and many more. But there isn't currently any visible support for image segmentation tasks in the toolbox version [12].

CNN-based object identification in real time is unattainable on FPGA systems due to their constrained hardware resources, including diminished memory capacity and weakened CPU performance [13]. CNN accelerators have been suggested by several academics at different design stages, including architecture, transistor, application, and system, with the aim of enhancing performance and decreasing power consumption in situations where hardware and power resources are constrained [14]. A system-level FPGA accelerator and a CNN accelerator that can adapt to architectures of varying sizes have been suggested in recent studies.

The structure of the paper is as follows. A description of high-level synthesis utilizing the HDL compiler is presented in Section 2. In Section 3, the Architecture for YOLOv2 is outlined. In Section 4, a comprehensive account of the design methodology for vehicle detection utilizing YOLOv2 is presented. This includes an in-depth analysis of the hardware development and design methodologies utilized in the proposed design. In Section 5, the Simulation results, Synthesis results, and their outcomes are elaborated. In Section 6, the paper is ultimately concluded.

2.HIGH-LEVEL SYNTHESIS BASED ON HDL CODER

Improving the delivery of complex systems, including video and image processing systems, is the goal of model-based design, which entails the systematic use of models throughout the development process (figure 1) [14]. While delineating design behaviours at high abstraction levels, HLS is swiftly gathering traction among designers in search of a method to guarantee continuous verification throughout the design cycle [15]. Illustrative software for HLS comprises Vivado HLS [16] and MATLAB HDL Coder [17]. Moreover, an extensive array of open-source alternatives can be found. These tools are commonly employed by digital architects and designers to devise and construct algorithms that encompass a wide range of domains, including communications, neural networks, image processing, and deep learning [18]. The implementation of HLS tools can result in a seven to tenfold escalation in code complexity. These tools enable the execution of high-abstraction modelling methods, including transaction-level modelling, and enable verification teams to employ behavioural intellectual property throughout various initiatives [19]. Moreover, processors comprise an overpowering majority of contemporary semiconductor systems [20]. To incorporate custom circuitry, microprocessors, memory, and video processing devices (DSPs) into a single device, supplementary software or firmware must be implemented during the design phase. Architectural and design professionals can assess an extensive range of implementation and algorithmic alternatives using a shared functional specification and an automated

HLS process, with the aim of identifying compromises in power, area, and performance. Consequently, advancements in register transfer level (the RTL) synthesis methods have enabled the widespread adoption of HLS technologies in industrial environments. Additionally, several HLS tools have been commercially marketed by renowned Electronic Designs Automation (EDA) providers [21]. By incorporating subsequent tools and employing behavioural hardware description language (HDL) code, the FPGA functions as an example of a device that facilitates the development of RTL implementations [22].

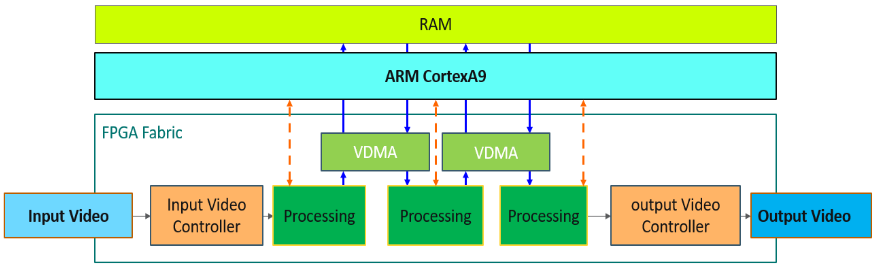

Figure 1: Embedded System for FPGA image processing

For the proposed model-based design presented in figure 2 of executing the whole code on the FPGA inside the framework of a coherent subsystem, the user first defines the algorithms in a standalone Simulink model. When the platform-specific HDL code is ready, the HDL Coder tool employs the Xilinx Vivado development tools to silently synthesis it for the FPGA [20].

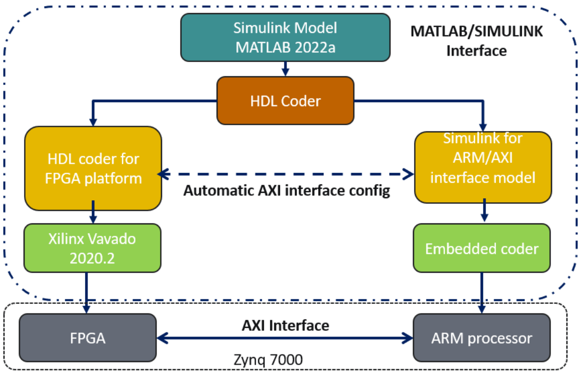
Simulink Model MATLAB 2022a
MATLAB/SIMULINK Interface
HDL Coder
HDL coder for FPGA platform
Simulink for ARM/AXI interface model
Automatic AXI interface config
Xilinx Vavado 2020.2
Embedded coder
FPGA
AXI Interface
ARM processor
Zynq 7000

Fig. 2. High Level Synthesis Flow Based on HDL coder.

Furthermore, the processor may operate independently during programming thanks to an interface model provided by HDL Coder. Instead of using the FPGA subsystem, the relevant AXI interfaces are used in this paradigm for data transmission from and to the FPGA [21]. This paradigm dictates the use of the Embedded Coder tool, which generates the processor code 1. The system may be monitored and controlled in real-time thanks to this feature.

3.YOLOV2 ARCHITECTURE

Among the most significant vehicle detection algorithms presented by the architecture in figure 3, YOLO was created by Redmon et al. The main focus of this research is the YOLOv2 network's hardware implementation, which provides better detection accuracy than the tiny YOLO. Cloud deployments of YOLOv2 are rather prevalent. Running on an Internet of Things (IoT) peripheral device requires additional hardware design methodologies due to the device's constrained power consumption and resource availability [13]. Based on the YOLOv2 network study, most layers (except the routing layer) exhibit serial processing. Just by specifying a preset address, the routing layer may be initiated. The minimum requirements for an accelerator are the following operations: data retrieval from memory, data processing, and data writing back to memory. Using loop tiling, the convolution loop R, C, M, N is tiled to Tr, Tc, Tm, and Tn [5]. In light of the substantial volume of input and output data, this method is consistently implemented in order to minimize memory access times and reuse data.
The overall architecture of the accelerator is shown in figure 3:

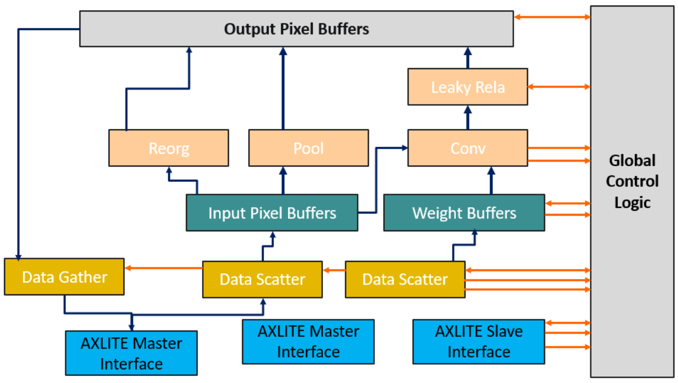

Figure 3: YOLOv2 Architecture

The effective FPGA bandwidth increases in tandem with the duration of the bursts, eventually reaching a plateau beyond a specific threshold [15]. The row-major data architecture in DRAM is typically rendered inaccessible as a result of the data tiling method. By aligning the kernel weights for an entire tile into a contiguous block, the number of memory accesses can be reduced while the utilization of external memory bandwidth is optimized [3].

By implementing input and output parallelism, the convolutional layer accelerates processing in a manner comparable to that described in [16]. Input parallelism (Tn parallelism) and output parallelism (Tm parallelism) can be attained in convolutional calculations through the construction of a multitude of parallel multiplication units and add trees. Simultaneously, the multiplication units of Tm*Tn are calculated. The pipeline produces the partial sums subsequent to the addition of every tree with a depth of Log2 (Tn).

4. DESIGN METHODOLOGY OF VEHICLE DETECTION BASED ON YOLOV2

The present investigation employed video frames with colour dimensions of 240 × 320 x 3 to construct a cascaded model for YOLOv2 detection of vehicle in MATLAB as presented in figure 4. Eight bits were used to represent each hue. The input frames were serialized to a stream of pixels, and each pixel was supplied to the design as an input during each clock cycle. This is in contrast with the HDL implementation, which makes use of frames instead of pixels. By implementing a Simulink library block, the YOLOv2 object detection system was additionally assessed. Following this, a comparison was made between the results obtained from the hardware implementation and the input images used by the Simulink library block [24].

The minimum and maximum values of input and output and intermediate nodes were meticulously documented throughout the simulation period of 100 seconds, which aligns with the duration of the video transmission. Subsequent simulation runs added to this database of lowest and maximum values until every conceivable visual signal input was taken into account. Then, within the given range, the inputs, outputs, and intermediate node values were used to determine the signal widths. The revised constraints were subsequently applied to all data nodes, as well as the main outputs and inputs of the HLS tool [25], in order to generate the RTL.

..

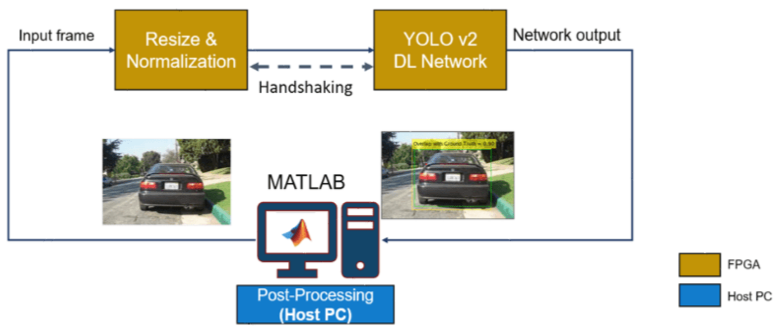

Figure 4. Vehicle detection Design based on YOLOv2

After that, the FPGA was loaded with the optimized RTL for programming purposes. The MATLAB software was utilized to perform a frame format conversion of the pixel-based FPGA output. The desired result was attained through the process of superimposing a delineated image onto the initial one. There are primarily three parts to a YOLO v2 car detecting program presented in figure 4. The supplied picture frame is processed by the first module, preprocessing, which also does image normalization and resizing. Module 2 involves feeding the pre-processed data into the YOLO v2 car detection network, which itself consists of a detection network and a feature extraction network. Thirdly, the input picture is superimposed with the postprocessed bounding box that was determined by finding the strongest bounding boxes from the network output. The following block diagram shows the example's implementation: the first two modules are loaded into the FPGA, and the postprocessing is carried out in MATLAB.

The output frame of the FPGA's HDL implementation, the input frame, and the output frame of the MATLAB model are all illustrated in Figure 4. The figure illustrates that the input picture source, the MATLAB-Simulink behavioural model, and the FPGA HDL model all operated autonomously and processed unique frames from the input video stream.

Back-annotations were applied to the HLS model utilizing the optimal datatype and data widths selected in RTL. The datatypes that have been back annotated for a model intermediate step (gradient computation) are illustrated in Figure 4. In the highlighted case, the result was 36 bits; it was composed of two signed fixed-point values, each containing 18 bits. The minimum breadth allowed in the worst-case data width model was 32 bits.

5.MODELING RESULTS AND DISCUSSION

5.1. Simulink HDL coder modelling

The stimulus-aware bit widths MATLAB HDL version 2022 coder was utilized to generate the RTL code for the HLS model (Figure 5). After being executed on the Xilinx ZedBoard, the RTL code was synthesized through the utilization of the Vivado 2020.2 software.

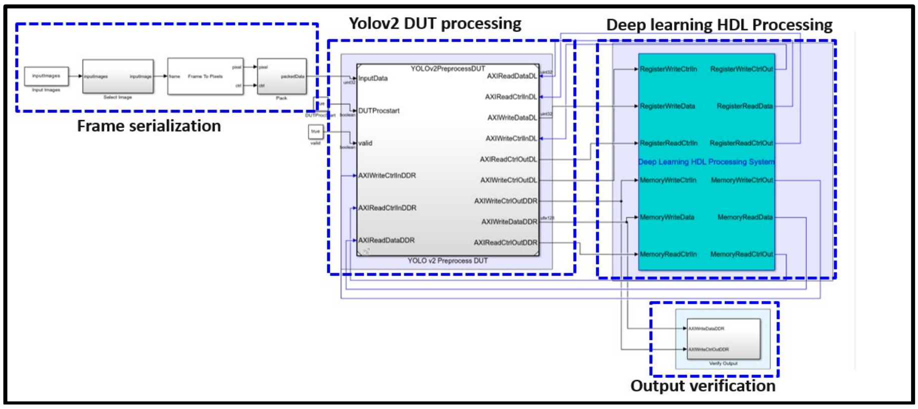

Figure 5. Simulink Top level model for YOLO v2 DUT-Preprocess

The illustration depicts the YOLOv2PreprocessTestbench.slx model at its highest level. The necessary workspace variables for the model are configured in the InitFcn callback via the helperSLYOLOv2PreprocessSetup.m script. The input frame is chosen by the Select Image subsystem from the Input Images block. The input image frame from the Select Image block is converted to a pixel stream and pixel-control bus by a Frame to Pixels block. The Pack subsystem generates uint32 data by concatenating the R, G, and B components of the pixel stream with the five control signals of the pixel control bus. By passing the compressed data through the YOLO v2 Preprocess DUT (figure 6), it is resized and normalized. The DDR is then updated with the pre-processed data via the handshaking signals transmitted by the deep learning IP core. The Deep Learning HDL Processing System block is utilized to simulate the DDR memory and the deep learning processor IP core. Additionally, the model incorporates a Verify Output subsystem that records the necessary signals to verify the pre-processed data as it is written to memory via the preprocessDUTVerify.m script.

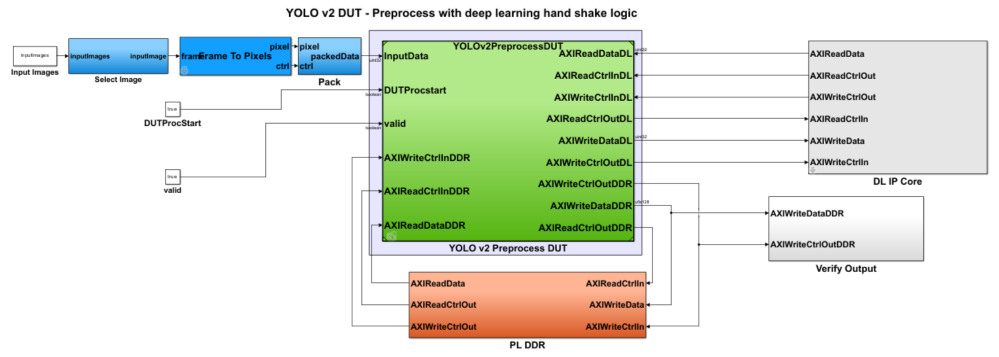

Figure 6. Simulink model for YOLO v2 DUT-Preprocess

Subsystems (figure 7) for unpacking, preprocessing (resizing and normalization), and handshaking logic are included in the YOLO v2 Preprocess DUT. The packed input is returned to the pixel stream and pixel control bus by the Unpack subsystem.

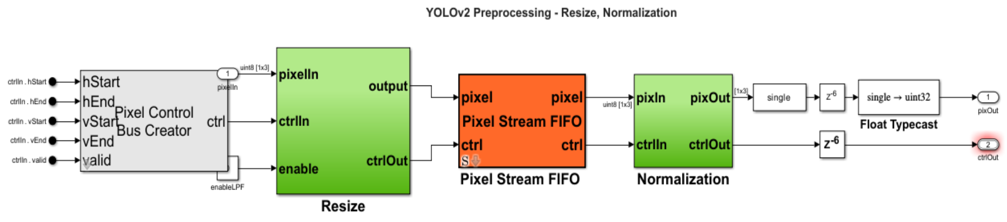
YOLOv2 Preprocessing - Resize, Normalization
Pixel Control Bus Creator
Resize
Pixel Stream FIFO
Normalization
Float Typecast
pixOut
ctrlOut

Figure 7. Simulink model for preprocessing

The input pixel stream is resized and rescaled within the YOLO v2 Preprocess Algorithm subsystem in accordance with the needs of the deep learning network. After being passed to the DL Handshake Logic Ext Mem subsystem, this pre-processed frame is written into the PL DDR. This example simulates two AXI4 Master interfaces, one for writing the pre-processed frame to the DDR memory and the other for reading and writing the deep learning IP Core registers.

The subsystem of the YOLO v2 Preprocess Algorithm consists of operations for resizing and normalization. The pixel stream is forwarded to the Resize subsystem so that it can be resized to the required dimensions by the deep learning network. Assemble the input image and network input dimensions utilizing the assisterSLYOLOv2PreprocessSetup.m script. The Normalization subsystem receives the resized input in order to rescale the pixel values to the range [0, 1]. In the cited source [XX], the resizing and normalization algorithms utilized in this instance are detailed.

5.2. Simulations results

On a non-synthesisable testbed, the generated VHDL RTL code was imitated using Vivado xSim. The operational simulation outcomes of the Vivado xSim simulator are illustrated in Figure 8. As shown in the figure, the pixel output of the proposed method is indistinguishable from the reference pixel values. Furthermore, it was ascertained that they corresponded with the results of the high-level MATLAB simulation for the optimal bit-width model. This was demonstrated by contrasting the output images of the two pathways with the identical input image. By employing a double precision model based on MATLAB, this study determined the quantization error caused by choosing narrower "optimal" signal widths. This was accomplished utilizing the "FPGA in the loop" co-simulation function of MathWorks' HDL Verifier [20]. Less than 1% is the Root Mean Square (RMS) value of the quantization error.

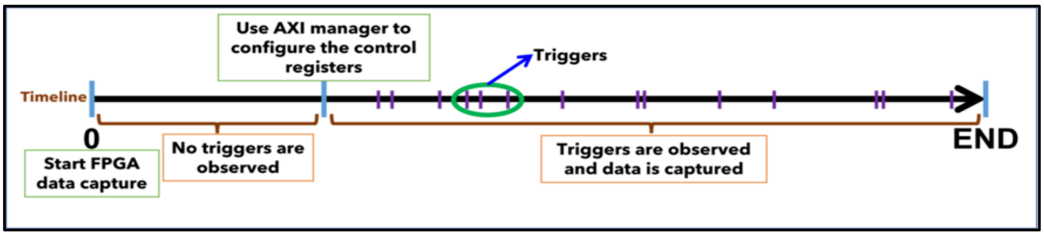
Use AXI manager to configure the control registers
Triggers
Timeline
0
Start FPGA data capture
No triggers are observed
Triggers are observed and data is captured
END

Figure 8. YOLOv2 Architecture simulation results

5.3. FPGA implementation and Synthesis results

Opti The proposed YOLOv2 design was developed utilizing various bit-width data types, as detailed in Section 3. For simulation, XSim was employed, as specified in Section 5.2.

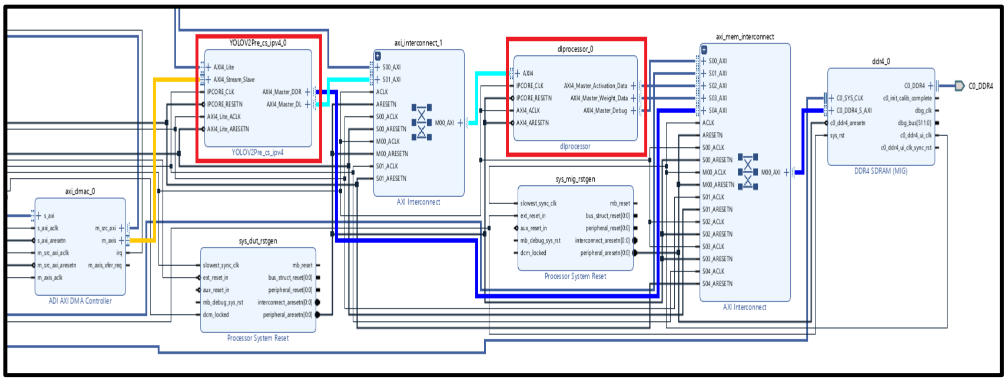

Figure 9: Vivado Design for proposed Yolov2 architecture

Target of a Vivado 2020.2 synthesized version of the exact Verilog design was the Xilinx ZedBoard. The results obtained from the synthesis and implementation of the proposed design are presented in Table 1. In addition, high-level synthesis tools provide access to an abundance of additional optimization strategies, such as pipelining and resource coordination. While these can be employed to improve the previously mentioned results, optimizing them is beyond the scope of this study. Figure 9 present a vivado project for the proposed vehicle detection design.

Table 1. Proposed YOLOv2 implementation results

Resource	Utilization	Resource	Utilization
LUT	204395(75%)	IO	52(15%)
LUTRAM	26211(18%)	BUFG	21(5%)
FF	224939(41%)	MMCM	2(50%)
BRAM	565 (61%)	PLL	1(13%)
DSP	305(12%)	FREQUENCY	200MHZ

Vivado 2020.2 was used to synthesize both the suggested IP core for vehicle identification and the whole HW-SW co-design. The core was built using the yolov2 algorithm. Table 1 replicates the findings of the synthesis. The overall resource utilization, as well as that of various layer types on the FPGA, is detailed in Table 1. In contrast, the 191 convolutional layer, max-pooling layer, and scale layer do not occupy substantial amounts of resources. It offers several crucial benefits, such as the ability to expedite time to market by simplifying the design process. In addition to the convenience of testing and verification. Many applications have used the possibility of converting not just MATLAB scripts but also Simulink models into HDL code.

6. CONCLUSION

This paper presents a fully-fledged technique for prototyping vehicle detection based on Yolov2 using Zynq-based boards. The methodology is based on Matlab/Simulink and the tools HDL Coder and Embedded Coder. An novel HLS design methodology was proposed to implement the suggested design for vehicle detection based on YOLOv2 model. The using of the model-based design based on HDL coder can reduce Time of prototyping by 60 %. We used Xilinx's Vivado xSim simulator to validate the RTL design's functionality. Following the implementation of the model-based design process. Consequently, this method may be seriously explored for use in FPGA applications that need real-time image processing. The method was tested using the Xilinx Zedboard and the MATLAB HDL coder flow. The area utilization was for design 1, 189 slice registers, 2303 slice LUTs (lookup tables), 204 k (75%) LUT, 305(12%) DSP blocks and 224 k (41%) flip-flops at 200 MHz. The next future work in this paper is to include optimization methodologies from well-known synthesis tool suppliers like MathWorks, Xilinx, Mentor, and Cadence into the suggested implementation approach. The goal is to increase the speed, surface area, and power consumption. In addition, there are intentions to broaden this inquiry to include ASIC designs in the future.

LIST OF ABBREVIATIONS

YOLO:	You Only Look Once	**RTL**:	Register Transfer-Register
FPGA:	Field Programmable Gate Array	**DDR**:	Double Date Rete
VHDL:	Very High Description language	**DUT**:	Data Under Test
MBD:	Model Based Design		

ACKNOWLEDGEMENT

The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA for funding this research work through the project number “NBU-FFR-2024-2099-01”.

REFERENCES

[1]. Diwan, Tausif, G. Anirudh, and Jitendra V. Tembhurne. Object detection using YOLO: Challenges, architectural successors, datasets, and applications. *Multimedia Tools and Applications*, vol.82, no.6, 2023, pp.9243-9275. https://doi.org/10.1007/s11042-022-13644-y

[2]. Sirisha, U., Praveen, S.P., Srinivasu, P.N. et al. Statistical analysis of design aspects of various YOLO-based deep learning models for object detection. International Journal of Computational Intelligent l Systems, vol.16, art.no.126, 2023. https://doi.org/10.1007/s44196-023-00302-w

[3]. Sun B, Wang X, Oad A, Pervez A, Dong F. Automatic ship object detection model based on YOLOv4 with transformer mechanism in remote sensing images. *Applied Science,* vol.13, no.4, 2023, art.2488. https://doi.org/10.3390/app13042488.

[4]. Qiu Z, Bai H, Chen T. Special vehicle detection from UAV perspective via YOLO-GNS based deep learning network. *Drones*. 2023; vol.7, no.2, p.117. https://doi.org/10.3390/drones7020117

[5]. Z. Wang, K. Xu, S. Wu, L. Liu, L. Liu and D. Wang, Sparse-YOLO: Hardware/Software co-design of an FPGA accelerator for YOLOv2. *IEEE Access*, vol. 8, 2020, pp. 116569-116585. doi: 10.1109/ACCESS.2020.3004198.

[6]. O. Ja. Kravets, I. A. Aksenov, I. V. Atlasov, P. A. Rahman, Yu. V. Redkin, O. Yu. Frantsisko, L. M. Bozhko. Implementation and evaluation of the effectiveness of dynamic object detection tools for multithreaded and distributed processing of graphical data. *International Journal on Information Technologies and Security*, vol.15, no.1, 2023, pp. 49-58. https://doi.org/10.59035/CMXQ2840

[7]. Al-batat R, Angelopoulou A, Premkumar S, Hemanth J, Kapetanios E. An end-to-end automated license plate recognition system using YOLO based vehicle and license plate detection with vehicle classification. *Sensor*s, vol.22, no. 23, 2022, p.9477. https://doi.org/10.3390/s22239477.

[8]. Li Y, Wang J, Huang J, Li Y. Research on deep learning automatic vehicle recognition algorithm based on RES-YOLO model. *Sensors*. vol.22, no.10, 2022, p.783. https://doi.org/10.3390/s22103783

[9]. Rani, Esther. LittleYOLO-SPP: A delicate real-time vehicle detection algorithm. *Optik*, vol.225, 2021, 165818.

[10]. S. Zhang, J. Cao, Q. Zhang, Q. Zhang, Y. Zhang and Y. Wang, "An FPGA-based reconfigurable CNN accelerator for YOLO. *2020 IEEE 3rd International Conference on Electronics Technology (ICET)*, Chengdu, China, 2020, pp. 74-78, doi: 10.1109/ICET49382.2020.9119500.

[11]. Babu, P., Parthasarathy, E. Reconfigurable FPGA architectures: A survey and applications. *Journal of the Institution Engineers (India)*. Series B, 102, 2021, pp.143–156 (2021). https://doi.org/10.1007/s40031-020-00508-y.

[12]. Babu, P., Parthasarathy, E. Hardware acceleration for object detection using YOLOv4 algorithm on Xilinx Zynq platform. *Journal of Real-Time Image Processing*, vol.19, 2020, pp.931–940. https://doi.org/10.1007/s11554-022-01234-y

[13]. Z. Li and J. Wang. An improved algorithm for deep learning YOLO network based on Xilinx ZYNQ FPGA. *2020 International Conference on Culture-oriented Science & Technology (ICCST)*, Beijing, China, 2020, pp. 447-451, doi: 10.1109/ICCST50977.2020.00092.

[14]. Badawi A, Bilal M. High-Level Synthesis of Online K-Means Clustering Hardware for a Real-Time Image Processing Pipeline. Journal of Imaging. 2019; 5(3):38. https://doi.org/10.3390/jimaging5030038

[15]. A. Shawahna, S. M. Sait, A. El-Maleh. FPGA-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, vol. 7, 2019, pp. 7823-7859. doi: 10.1109/ACCESS.2018.2890150.

[16]. Yang X, Zhuang C, Feng W, Yang Z, Wang Q. FPGA implementation of a deep learning acceleration core architecture for image target detection. *Applied Sciences*. vol.13, no.7, 2023, p.4144. https://doi.org/10.3390/app13074144

[17]. J. Pandey, A. R. Asati, M. V. Shenoy and P. Sikka. Verification of hardware resource utilization through high level synthesis for FPGA implementation. *2023 4th IEEE Global Conference for Advancement in Technology (GCAT)*, Bangalore, India, 2023, pp. 1-6, doi: 10.1109/GCAT59970.2023.10353312

[18]. Ayachi, R., Said, Y. & Ben Abdelali, A. Optimizing neural networks for efficient FPGA implementation: A survey. *Archives of Computational Methods in Engineering*, vol.28, no.7, 2021, pp.4537–4547). https://doi.org/10.1007/s11831-021-09530-9

[19]. Ahmed Alhomoud, Real time FPGA implementation of a high speed for video encryption and decryption system with high level synthesis tools. *International Journal of Advanced Computer Science and Applications (IJACSA),* vol.15, no.1, 2024. http://dx.doi.org/10.14569/IJACSA.2024.0150172

[20]. T. Saidani, R. Ghodhbani, Hardware acceleration of video edge detection with hight level synthesis on the Xilinx Zynq platform. *Engineering Technology and Applied Science Research*, vol. 12, no. 1, Feb. 2022, pp. 8007–8012, Feb. 2022 (Visited on 20-02-2024).

[21]. M. Versen, S. Kipfelsberger, F. Soekmen. Model-based reference design projects with MathWorks' HDL workflow advisor for custom-specific Electronics with the Zedboard. *ANALOG 2016;15. ITG/GMM-Symposium*, Bremen, Germany, 2016, pp. 1-4. (Visited on 08.02.2024).

[22]. Babu, Praveenkumar, Eswaran Parthasarathy. Hardware acceleration of image and video processing on Xilinx zynq platform. *Intelligent Automation & Soft Computing*, vol.30, no.3, 2021 (Visited on 20-02-2024)

[23]. Kayed, S., Elsayed, G. An optimizing technique for using MATLAB HDL coder. *Bulletin of the National Research Centr*e, vol.47, no.1, 2023, p.94. https://doi.org/10.1186/s42269-023-01066-1

[24]. Sankar, Deepa, et al. Rapid prototyping of predictive direct current control in a low-cost fpga using hdl coder. *International Journal of Power Energy Systems*, vol.43, no.10 (2023), pp.1-9. https://doi.org/10.2316/J.2023.203-0437

[25]. Ghada Elsayed; Somaya Ismail Kayed. A comparative study between MATLAB HDL coder and VHDL for FPGAs design and implementation. *Journal of International Society for Science and Engineering*, vol.4, no.4, 2022, pp.92-98. doi: 10.21608/jisse.2022.136645.1056.

Information about the authors:

Dr. Jihane Ben Slimane is currently, an Assistant Professor in the Department of Computer Sciences, Faculty of Computing and Information Technology, Northern Border University, Rafha, Kingdom of Saudi Arabia. Her areas of expertise and research interests include Artificial Intelligence, Machine Learning, Neural Networks, Fault Diagnosis, and Cybersecurity.

Manuscript received on 17 March 2024