2022 8th International Conference on Mechatronics and Robotics Engineering (ICMRE) | 978-1-6654-8377-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICMRE54455.2022.9734102

Open-Source Educational Platform for FPGA Accelerated AI in Robotics

Nicolaj Malle
DIII Group, SDU UAS Center
Mærsk Mc-Kinney Møller Institute
University of Southern Denmark
Odense, Denmark
nhma@mmmi.sdu.dk

Emad Ebeid
DIII Group, SDU UAS Center
Mærsk Mc-Kinney Møller Institute
University of Southern Denmark
Odense, Denmark
esme@mmmi.sdu.dk

***Abstract*—Artificial Intelligence (AI) using neural networks is growing rapidly in the area of robotics and many tools have been developed in the last few years to utilize these networks. However, these tools are very abstract and do not provide deep knowledge on how the neural networks perform their computations. This makes it difficult for roboticists to understand and fully harness the power of AI. In this work, we present an open-source framework for designing and implementing a simple neural network targeting edge computing platforms. The framework goes step-by-step through the training, synthesis, and hardware implementation on a Zynq platform. The final hardware implementation is evaluated against a classical implementation in software. The platform was used in the Embedded Systems Course at the University of Southern Denmark.**

***Keywords*—FPGA acceleration, neural network, high-level synthesis**

I. Introduction

Robotics and advanced autonomous systems are being revolutionized by neural network powered AI. Such algorithms allow roboticists to build problem-solving systems that learn how to solve complex tasks from data.

Nowadays, numerous AI frameworks have been developed, for instance, to detect, track, or locate objects such as YOLO [1], MobileNet [2], and ResNet [3]. These frameworks are mainly built in software and customized to work on specific processing units (CPUs, GPUs, or TPUs). Robotics benefits from AI by utilizing the small-form-factor embedded boards to run the AI algorithms onboard (i.e., Edge AI). Several companies like Google and Nvidia produce different hardware boards to run the AI algorithms. However, these boards are seen as black box devices since they only enable high-level software to configure them. That makes them hard to optimize to fit the desired robotics performance, especially for strict design requirements such as real-time applications.

In this paper, we present a novel technique to develop hardware platforms to run AI on the edge. Our approach allows future engineers to learn how to design, implement, test, and tune their AI algorithms from the high-level software to train the network to the low-level hardware to execute the algorithms. The proposed educational platform gives a clear understanding of what each stage does in building an Edge AI application that can run efficiently on ground and aerial robots. The contributions of this work are:

- An open-source platform to implement AI inference modules on FPGAs using popular tools.
- A flexible framework that allows for modification and implementation of various AI applications in different physical hardware.
- An educational step-by-step implementation of a robotics computer vision demonstration to shed light on the usually behind-the-scenes implementation steps.

All code and instructions for the platform are made freely available here: `github.com/nhma20/FPGA_AI`

AI powered by neural networks is revolutionizing several domains, and inference at the edge is becoming increasingly useful. With their great flexibility and efficiency, FPGAs are well positioned in the race to become the hardware of choice on the edge, and much previous work [4] [5] [6] has been done on the design of such accelerators.

Existing frameworks for implementing AI inference in hardware on FPGAs, such as hls4ml [7], FINN [8], and Xilinx'

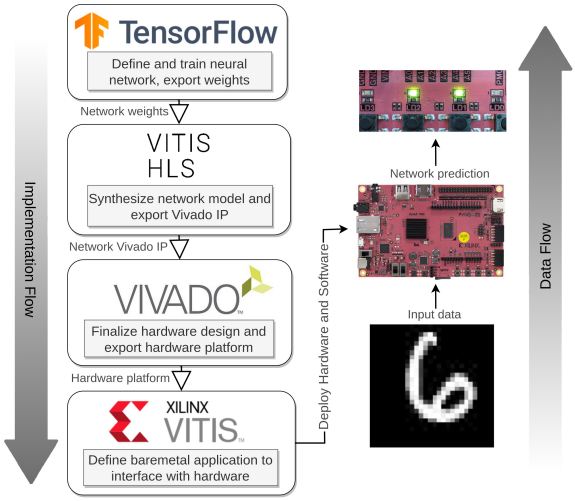

Fig. 1: Workflow and dataflow of proposed platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 861111, Drones4Safety.

DPU [9], all produce black box IPs that try to maximize performance at the inference stage.

Other work [10] [11] [12] has used FPGAs in an educational context, but specifically Huang et al. [13]'s proposed four hour course and Panicker et al. [14]'s proposed multi-lecture course attempt to open up the black box of FPGA-based neural network inference in an educational setting. However, none of these approaches demonstrate the full flow; from network design through hardware/software development to deployment.

The rest of the paper is structured as follows; Sec. II covers the different parts of the platform; Sec. III describes the testing performed to validate the deployed hardware; Sec. IV concludes on the findings of the paper.

II. Platform Description

The workflow of this paper's proposed platform is outlined in Fig. 1. First, a neural network is defined and trained in Python with Tensorflow [15] and a dataset, in this case the MNIST [16] dataset of handwritten digits, after which the weights of the network are extracted.

Next, the neural network is reassembled in C++ in the Vitis HLS tool [17]. The tool then synthesizes the code into a VHDL/Verilog IP module. This IP, along with some peripherals, is assembled into a block diagram in Vivado [18] to complete the hardware platform.

The hardware platform is connected with a software application in Vitis [19] to interface with the hardware. Finally, the hardware/software application is ready to be deployed on the Pynq-Z2 [20] embedded development board.

Inference can now be performed by sending data to the device from a host PC. The dataflow of the system during inference is shown in Fig. 2.

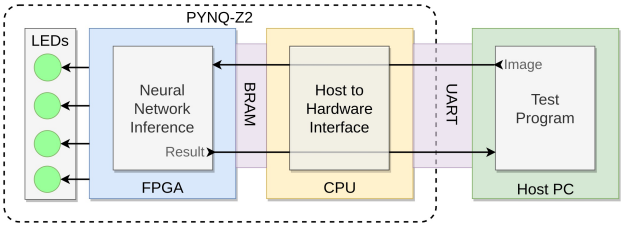

Fig. 2: Data flow in implemented system.

The host PC communicates with the PS (Processing System) side of the Zynq [21] chip and sends data via a serial connection. The PS side acts as a gateway to the PL (Programmable Logic) side of the Zynq and moves the received data to the PL side via BRAM (Block RAM). Inference takes place in PL, and the result is displayed on the user LEDs and sent back through the PS side to the host PC.

A. Neural Network Design and Training

As an example, the neural network in this demonstration will be trained on the MNIST dataset to recognize handwritten digits. The network is designed and trained using Tensorflow with which it takes only a couple of lines of Python script to set up a network and train it.

The specific network that is implemented is a multilayer perceptron (MLP) characterized by fully connected layers. The network has 100 nodes in the input layer, 32 neurons in the first hidden layer, 16 neurons in the second hidden layer, and 10 outputs in the output layer. The network's structure can be seen in Fig. 3

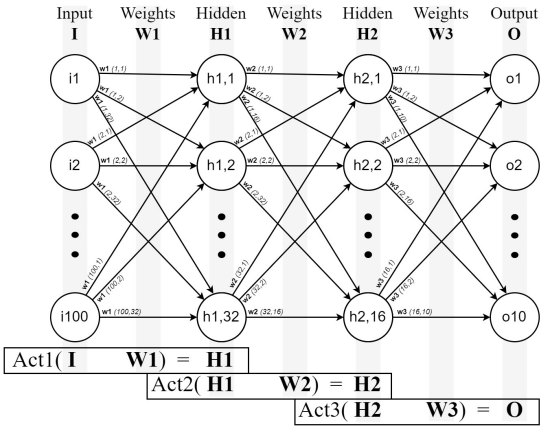

Fig. 3: Implemented neural network structure and math.

All the math needed to do inference with a network like this is also shown in the figure - purely matrix multiplications and non-linear activation functions. The first two activation functions are both ReLU (Rectified Linear Unit) while the output activation is softmax.

The network's 3872 trainable weights ($100 \cdot 32 + 32 \cdot 16 + 16 \cdot 10$) are trained with a gradient descent optimization algorithm and back-propagation. During training, the values of all weights get fine-tuned to best relate a given input to the desired output. After a few seconds of training on a non-GPU machine, the network reaches an accuracy of about 94% and the trained weights are extracted.

B. High-Level Synthesis

After obtaining the weights of the trained network, high-level synthesis is used to produce an IP of the network that can be integrated into a Vivado block design. In Vitis HLS, the network is reconstructed with the weights, basic matrix multiplication, and simple activation functions - all that is needed for inference.

Vitis HLS has the option to validate the functionality of the design with C Simulation. Here, a testbench is created which calls the functions to be synthesized by HLS to make sure the behavior is as expected. Once verified, C Synthesis synthesizes the design after which the IP can be exported.

HLS has the option to use various directives to guide the C Synthesis to meet certain requirements like I/O or timings. This demonstration generates the IP mostly without any guiding directives, leaving room for performance improvements.

C. Hardware Implementation

Vivado is used to design the hardware platform that includes the generated IP module. By default, HLS infers an

`ap_ctrl_hs` interface when synthesizing a module, and the surrounding hardware must accommodate for this protocol to use the module.

In order for the system to be able to exchange information between hardware and software, the Zynq processing system must be instantiated in the Vivado block design alongside BRAM functionality. The Zynq block also provides the rest of the hardware design with its main 100MHz clock. Lastly, the hardware directly controls some I/O like the user LEDs.

Once the block design is complete, Vivado can generate a bitstream needed to deploy the design in actual hardware. Exporting the platform makes it possible to build a software application on top of the hardware.

D. Software Application

The software application is built in Xilinx Vitis (formerly SDK) on a platform project derived from the exported hardware and bitstream. The application runs bare-metal (i.e. no operating system) on the APU of the Pynq-Z2 board although Xilinx provides Petalinux for embedded development.

In this demonstration, the software application acts as an interface between the implemented inference hardware and a connected computer sending data to be analyzed by the network. This works by initializing the UART connection to the host PC and BRAM connection to the FPGA. In addition, it facilitates the flow of data between the two. During benchmarking, the software application performs network inference as well as timing.

III. Testing and Validation

This section takes a closer look at the implemented inference hardware throughout the toolchain and compares it with a purely software implemented inference application.

A. Generated Hardware Inference Module

The Vitis HLS tool reports an estimate of the synthesized IP's resource utilization when C Synthesis completes. Similarly, Vivado reports the actual resources used in the design when finishing implementation. Tab. I shows the HLS and Vivado reported utilized resources.

TABLE I: Estimated and actual FPGA ressource utilization.

	LUT	FF	BRAM	DSP
Vitis HLS	33332	26256	34	165
Vivado	15898	22309	19	165
Available Resources	53200	106400	140	220

The Vivado values represent the actual hardware implementation and also include all other modules and inference the IP in the block design. Despite this, all Vivado values are significantly lower or the same as the HLS values and within the number of available resources. Only the DSP values are the same and also relatively high compared to the available number. This is likely because the hardware inference implementation uses single-precision floating point arithmetic which is quite expensive on FPGAs and gets mapped to dedicated DSP (Digital Signal Processing) slices of which there are only 220 for the Pynq-Z2 board.

B. Software vs. Hardware Implementations

In software, computationally heavy applications come with a trade-off in execution time. For FPGAs however, the tradeoff is usually the additional resources consumed by the larger design while maintaining low execution time. Therefore it is important to keep in mind, when comparing the two computing platforms, that the software will slow down with more features added, while the FPGA's limited silicon can keep execution time low but only accommodate a limited number of features.

Before benchmarking the inference hardware it is worth noting that the HLS tool gives an estimated latency of the IP. When synthesizing the demonstration application, HLS reports an estimated latency of 16755 clock cycles. At 100MHz this corresponds to 166.8 μs.

The actual timing of the inference hardware is done on the FPGA itself for maximum accuracy. The HLS-implemented `ap_ctrl_hs` interface of the generated IP comes with an `ap_done` signal to notifiy when the IP's output is valid. Since the IP is in continuous operation, the timing measures the number of clock cycles between each `ap_done` rising edge.

It seems fitting to compare the performance of the inference hardware with the same functionality in software - on the same chip. To achieve this, the same code that was used with HLS to synthesize the IP is implemented in bare-metal on the Zynq processor. No other tasks will be running while timing the software inference, illustrating the best-case scenario for the Zynq processor. No manual optimization or tuning has been performed, similar to the FPGA (HLS) implementation.

Fig 4 shows the results of running the TCF profiler in Vitis on the software inference.

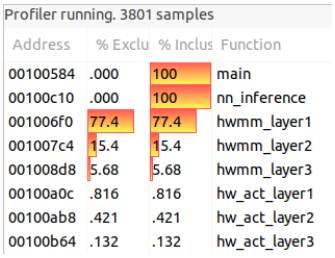

Fig. 4: Xilinx Vitis TCF profiler analysis of inference functions in software.

As expected, the heaviest part of the inference, the first of the three hardware multiplication layers covering 3200 weights, takes up the most execution time. Then comes the second layer multiplications and then third layer multiplications. Together, they account for over 98% of the execution time. Interestingly, the first layer's 3200 weights accounts for 82.6% of all weights but only 77.4% of execution time. The second layer has 13.2% of all weights and 15.4% of execution time, while the third layer is at 4.1% and 5.7% respectively.

The timings of the software and hardware benchmark can be seen in Tab. II.

Without any specific optimizations, the inference times in hardware and software for the same network are extremely

TABLE II: Inference timing results

	Clk (MHz)	Cycles	Time (μs)
PS	650	107662	165.6
PL	100	16680	166.8

alike. However, it is worth noting that adding any extra tasks, even just communication jobs, to the software would inevitably make it slower, whereas there is still plenty of room on the FPGA for additional functionality, as seen in the previous section.

By introducing a small amount of optimization, specifically by using arbitrary precision fixed point datatypes (`ap_fixed<32,24>`) in HLS instead of single precision float, the estimated inference time drops 8x, BRAM usage drops 1.5x, DSP usage drops 4x, FF usage drops 1.35x, and LUT usage drops 3x. Another way to increase performance would be to quantize and prune the network.

Another aspect to consider when comparing the software and hardware approach is the power consumption. Typically, FPGAs consume less power than most other computing units, perhaps except ASICs, when doing similar work. Fig. 5 shows Vivado's estimation of on-chip power consumption.

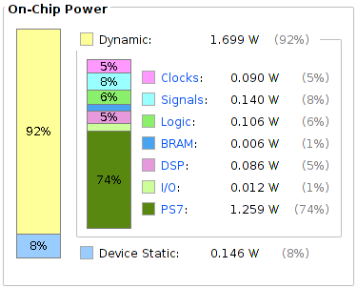

Fig. 5: Estimated on-chip power consumption of design.

This assumes a 50% load on the CPU (PS7) which is realistic given that the software inference benchmark runs as fast as possible on one of its two cores. From the figure it is apparent that the PS alone consumes almost ¾ of the total on-chip power, giving the PL the clear edge when it comes to performance/watt.

IV. Conclusion

This paper presented an educational platform to introduce and deploy AI in hardware on FPGAs. A step-by-step approach through a toolchain generates a heterogeneous hardware/software system ready to deploy on a physical device to showcase the strengths of hardware empowered AI. A comparison between software and hardware inference of an identical neural network shows how an FPGA has the potential to match or exceed the inference frequency of the software approach while maintaining a much lower power consumption.

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)*, 2016, pp. 779–788.

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2015.

[4] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A Survey of FPGA-Based Neural Network Inference Accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, mar 2019. [Online]. Available: https://doi.org/10.1145/3289185

[5] Z. Li, Y. Zhang, J. Wang, and J. Lai, "A Survey of FPGA Design for AI Era," *Journal of Semiconductors*, vol. 41, no. 19090017, p. 021402, Jan 2020. [Online]. Available: http://www.jos.ac.cn/article/id/a3f00d44-ccf7-43b9-8f53-241ddea6cbf7

[6] R. Wu, X. Guo, J. Du, and J. Li, "Accelerating Neural Network Inference on FPGA-Based Platforms—A Survey," *Electronics*, vol. 10, no. 9, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/9/1025

[7] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. P. Carloni, G. Di Guglielmo, P. Harris, J. Krupa *et al.*, "hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices," *arXiv preprint arXiv:2103.05579*, 2021.

[8] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[9] "DPU for Convolutional Neural Network." [Online]. Available: https://www.xilinx.com/products/intellectual-property/dpu.html

[10] J. M. Ramirez-Cortes, P. Gomez-Gil, V. Alarcon-Aquino, J. Martinez-Carballido, and E. Morales-Flores, "FPGA-based Educational Platform For Real-Time Image Processing Experiments," *Computer Applications in Engineering Education*, vol. 21, no. 1, pp. 193–201, 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.20461

[11] T. Sansaloni, A. Perez-Pascual, V. Torres, V. Almenar, J. F. Toledo, and J. Valls, "FFT Spectrum Analyzer Project for Teaching Digital Signal Processing With FPGA Devices," *IEEE Transactions on Education*, vol. 50, no. 3, pp. 229–235, 2007.

[12] W. Balid and M. Abdulwahed, "A novel FPGA educational paradigm using the next generation programming languages case of an embedded FPGA system course," in *2013 IEEE Global Engineering Education Conference (EDUCON)*, 2013, pp. 23–31.

[13] N.-S. Huang, J.-M. Braun, J. C. Larsen, and P. Manoonpong, "Teaching Hardware Implementation of Neural Networks using High-Level Synthesis in Less Than Four Hours for Engineering Education of Intelligent Embedded Computing," in *2019 20th International Carpathian Control Conference (ICCC)*, 2019, pp. 1–7.

[14] R. C. Panicker, A. Kumar, and D. John, "Introducing FPGA-based Machine Learning on the Edge to Undergraduate Students," in *2020 IEEE Frontiers in Education Conference (FIE)*, 2020, pp. 1–5.

[15] M. Abadi *et al.*, " TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[16] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[17] Xilinx, "Introduction to vitis hls," 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS

[18] ——, "Vivado ml overview," 2021. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[19] ——, "Vitis software platform," 2021. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[20] TUL, "Tul pynq™-z2 board," 2021. [Online]. Available: https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html

[21] Xilinx, "Zynq-7000 soc," 2021. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html