A TASTE OF BINARISED NEURAL NETWORK INFERENCE FOR ON-BOARD FPGAS

Jannis Wolf[1,3] **and Leonidas Kosmidis**[2,1]

[1] *Universitat Politècnica de Catalunya (UPC), Barcelona, Spain*
[2] *Barcelona Supercomputing Center (BSC), Barcelona, Spain*
[3] *Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany*

ABSTRACT

Recently there is an increased interest for the application of machine learning and artificial intelligence methods in on-board processing. However, space processors cannot provide the required performance for these workloads and COTS processors and accelerators cannot be used either. In this work we present the design and implementation of an FPGA Binary Neural Network accelerator for the acceleration of on-board inference tasks, which has been realised with the help of ESA's TASTE framework. Our simulation results show an estimated performance improvement of 32× compared to LEON3.

Key words: TASTE, AI, ML, BNN, LEON3 .

1. INTRODUCTION

In recent years there has been an increasing interest in artificial intelligence (AI) and machine learning (ML) in space, specifically for on-board processing [1]. For example, current missions from both ESA and NASA include AI processing for autonomy and efficiency. In particular, NASA's latest Mars Rover, Perseverance, uses AI for navigation, however the low processing power of existing space processors permits only offline navigation planning and only over small distances and at low speed. For this reason, COTS accelerators such as Intel Movidius are explored for use in space systems, such as in ESA's Φ-Sat-1 technology demonstration mission, where AI is used for detecting clouds in satellite earth observation images, in order to save bandwidth from transmitting them. However, due to the low radiation tolerance of COTS systems in general, these solutions cannot be currently used beyond low-earth orbit (LEO) or for long term missions such as institutional ones. Moreover, COTS software stacks used in such accelerators, frequently depend on non-space qualified and non-real-time operating systems such as Linux, which creates another challenge to their adoption.

Therefore, ML acceleration features are needed in already qualified FPGAs, so that they comply with space requirements and to reduce their qualification time and cost. In this paper, we describe our on-going work for increasing the AI performance of space processors implementing a state-of-the-art low-cost binarised neural networks (BNN) on FPGAs, using ESA's TASTE framework for a reliable software stack. Our solution is implemented in VHDL and is open-source [2]. Our early experimental results show that our proposal can provide a significant boost to the on-board machine learning processing capabilities of existing space systems.

In the following sections we provide the current design details and implementation status of our hardware proposal as well as preliminary results of our experimental evaluation. Section 2 describes the architecture of the binarised-network accelerator and the workflow than can be used to design an accelerator using the proposed toolchain. Section 3 evaluates the proposal and Section 4 presents the conclusion and our plans for future work.

2. BNN ACCELERATOR

2.1. Architectural Design

Depending on the type and the size of the actual neural network and on other mission requirements, an FPGA accelerator might be a preferred option. Moreover, recently Binary Neural Networks (BNNs) have been proposed [3], which reduce significantly the memory and computational resources required for inference.

While FPGA support for this type of neural networks already exists or can be implemented in various FPGA toolkits such as Xilinx's FINN [4] framework, Xilinx's Versal AI core [5], Microchip's VectorBlox [6] and others including high-level synthesis solutions, there can be certain settings in high criticality space systems which require full control or ownership of the hardware and software stack of the accelerator, such as integration with real-time operating systems (RTOSes), portability across FPGA vendors and other requirements.

In order to satisfy this need, ESA frequently relies on model-based engineering (MBE) approaches. In partic-

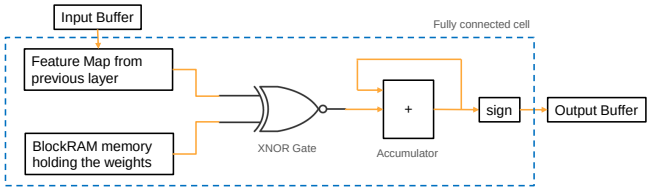

Figure 1. Functioning core of the accelerator architecture: fully connected cell.

ular, ESA has developed the open source TASTE model-based engineering framework [7][8], which supports both software and hardware code generation, while it supports solutions from multiple vendors relevant to the space domain.

In this section, we present an implementation of a BNN accelerator for FPGA systems which is appropriate for use in critical space systems, since its software stack and hardware communication implementation with the host CPU is correct-by-construction, leveraging TASTE.

We use TASTE in order to automatically generate the communication between the host space CPU processor, LEON 3, and our accelerator, based on an Abstract Syntax Notation One (ASN.1) [9] specification. In the ASN.1 specification, we define the amount of data we want to be transferred to the accelerator and their types, which is input data over which we want to perform the inference. Moreover, we describe the data which are sent back to the host CPU to indicate the inference results, which in our case is the short bit-vector of the last BNN layer used for the computation. Then the host processor translates this bit-vector to useful information, eg. to determine the class of the inferred object in case of image recognition.

The TASTE framework generates the host CPU code required for the communication based on the provided specification, which is in our case C code for bare metal execution. However, TASTE supports also code generation for Ada and its high-integrity SPARK subset, as well as integration with the RTEMS real-time operating system, including its SMP variant. Moreover, it generates the hardware implementation of the specified communication protocol in VHDL as well the project configuration targeting specific FPGAs, allowing a large variety of devices from different vendors. As a consequence, we can guarantee that these parts are free of errors, and therefore we can only focus on the implementation and verification of the BNN accelerator functionality.

As opposed to the input data which are streamed to the accelerator, the BNN weights are stored in the FPGA in block ram and they are reused for all inference operations over the different input data. In BNN networks, each activation and weights have two possible values, 1 or -1, which can reduce the memory consumption up to 8-64 times compared to conventional neural networks implemented with integer operations. Since only 1 bit is used for encoding these values, -1 is represented by the value 0. The more important benefit of BNNs is that instead of the expensive Multiply-and-Accumulate (MAC) operation used in the most expensive parts of the conventional neural networks, the implementation of fully connected layers, this is replaced by an exclusive nor (XNOR) operation and bit-count. In a similar way, other operations found in conventional NNs are replaced by bit-wise operations in BNNs.

In our accelerator, we currently implement fully connected networks using interconnected fully connected cells. The architecture of each cell is shown in Figure 1. From an input buffer the feature map gets loaded and flows through an XNOR gate together with the weights loaded from the BlockRAM. An accumulator adds up the values until the full feature vector passed the XNOR gate. The sign function calculates the outcome as follows: $result = 2 * p - n$ where p is the number of set bits and n is the length of the feature vector. If the result is positive, the activation of the neuron is 1, if its negative it is 0, respectively.

The RTL schematic of parts of the accelerator are shown in Figure 2. The fully connected layer receives the data from the FIFO module from the accelerator or the previous layers, as well as the weights from the memory modules. The fully connected layer as well as the memory is clocked and connected with an address counter. In our current implementation the loading of the weights takes twice as many cycles as the number of neurons in the accelerator layers. After the calculation an accumulator module sends the activations to the next layer. We can further optimize the design by using more BRAM modules for storing and loading the weights in parallel, however, the current design can provide enough performance as we show in the next section, so this is left as future work. The full code of our accelerator is released as open source [2].

2.2. Model-Based Engineering BNN Accelerator Design Process

In this project we not only want to present the current details and implementation status of our hardware proposal, but also how the integration of TASTE's model based development can facilitate the design process of custom AI accelerator systems in general.

At the moment, we mainly take advantage TASTE's interfacing capabilities in the form of ASN.1. Apart from a common language for communication, it also structures the implementation of a complex project composed by both hardware and software components. The next step will be to take advantage of the TASTE's deployment capabilities. In the following we show the design process of an neural network accelerator using our method.

1. Starting from the training process, special algorithms for binary weight training must be used. We designed a set of Python classes extending the PyTorch deep learning framework with a Binary Fully

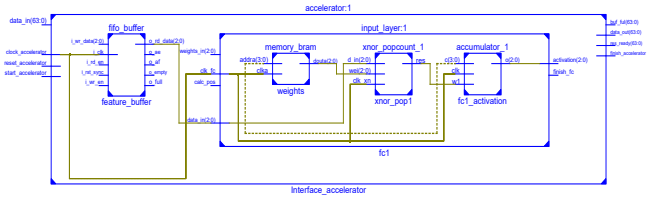

Figure 2. Sample accelerator schematic showing only one fully connected cell.

Connected Layer and an optimizer that uses straight through estimation for binary back propagation [3]. This step allows to experiment with various BNN models for early design space exploration and test them in a model-in the loop fashion, before implementing them in hardware. Moreover, the resulting trained BNN will be ready for use in the next steps.

2. The code generation of the communication is done by TASTE. The designer needs to specify the data format sent to the accelerator, which matches the input size of the neural network model and the format of the data send back from the accelerator matching the output size of the model in ASN.1 notation. TASTE returns the implementation of the specified communication protocol in C for the CPU side and in VHDL for the accelerator, respectively.

3. On the software side, the sending and receiving of the data has to be invoked by using the set function for communication generated by TASTE.

4. Currently, the hardware implementation of the binary neural network model in the accelerator has to be performed manually by the designer. However, this process is facilitated by a set of VHDL components, which resemble the PyTorch classes used in the training and inference process with Python in the first step. In the future, changes in the neural network architecture will automatically trigger a reconfiguration of the accelerator design.

5. Verification in simulation can be done by writing testbenches in VHDL and test against the results obtained from Pytorch to find errors earlier before costly deploying. Although not completely automated, this step allows software-in-the-loop testing.

6. The final FPGA deployment is very flexible. TASTE features several different deployment nodes like LEON3 running RTEMS, x86 running Linux and more. On the hardware compatibility, TASTE has been shown to be capable of handling a Spartan3 and is at the moment being developed further to use Xilinx's Zynq based SoCs [10]. Hardware-in-the-loop Verification as a last step concludes the design process.

3. EVALUATION

We have implemented our accelerator using the latest version of the TASTE framework and Xilinx's latest ISE version (14.7), which is the only FPGA flow from Xilinx which is currently integrated with it.

We have trained the BNNs for our test cases in Python using PyTorch in order to obtain the trained binary weights which we use in our accelerator. Currently we have not performed yet the integration of all the system on chip in the FPGA (i.e. including the LEON 3 CPU). Instead, we have validated our design in simulation with ISE, using a testbench for testing the accelerator from end-to-end inference: transfer data to the accelerator, inference and transfer data back from the accelerator. We validate that the inference results using our accelerator match exactly the CPU implementation.

For the functional validation of the accelerator, we have used two test cases: one for the classification of the Iris data set [11] obtained from [12], which consists of 150 4-dimensional feature vectors who group into three classes, and one for the MNIST dataset [13]. The Irist dataset has good characteristics for testing neural network performance with respect to prediction. In terms of computational cost, a 4D feature vector does not require very expensive calculations, but it is a good way to verify our implementation. Similarly, the MNIST dataset is small enough to allow validation by simulation.

For the performance evaluation, we perform synthesis for the Xilinx Spartan3 FPGA and according to ISE's timing estimation, without place and route our accelerator can achieve a maximum frequency of 114.943MHz. We compare our performance against a simulated version of the LEON 3 using Cobham Gaisler's TSIM version 3.0.2, which models a LEON 3 configuration clocked at 50MHz and equipped with a 4 KiB cache, with 16 bytes per cache line.

Due to the memory size and simulated instruction limitations of the evaluation version of TSIM, we cannot use our validation case studies, which despite their small size exceed these limits. For this reason, we model only a 512×512 fully connected layer, which fits in the processor cache, therefore the achieved performance is only limited by the computational capabilities of the CPU. When compiling with -O2, the LEON 3 can perform the inference task in 4.8 Million cycles, while our accelerator needs 65 thousands cycles for the same computation. Given the fact that our accelerator operates with double frequency compared to the CPU, this results in an impressive speedup of 147×.

If we consider the data transfer of these values to the accelerator, the performance benefit is going to be smaller. For example, in the extreme case of such a small network that we are considering which doesn't make much computational demands and in the case that the input data fit in the processor cache, the benefit is about 32×. In case of larger BNNs, which consist of multiple connected lay-

ers and therefore they will require a higher ratio of computations compared to the transferred input data, the benefit is expected to be higher.

However, he have to highlight that both these results are obtained with simulation so they have to be considered with care. Once we will perform the integration of the LEON 3 and our accelerator in an FPGA, we will be able to obtain performance results with higher confidence. Moreover, we will be able to compare their area and frequency trade-offs in a realistic way, using synthesis results after placement and routing. Last but not least, we will use a relevant case study from a safety-critical domain similar to [14] or [15].

4. CONCLUSION AND FUTURE WORK

In this paper we introduced a work-in progress approach aiming to the improvement of artificial intelligence computation in space computing using a model-based approach.

Our Binary Neural Network (BNN) FPGA accelerator design confirmed once again the benefits of the TASTE framework, which allows developers to concentrate on the design of their hardware without being concerned about the software stack for communication with the accelerator. This accelerator has demonstrated incredible results with simulation, showing the advantages of BNNs. Although we take them with care, we are optimistic about the actual performance benefit that our accelerator can have on a real FPGA.

As a future work we are planning to create a VHDL code generator for the implementation of the AI accelerator directly from Python and integrate it with the TASTE framework. Furthermore, different architectural choices in neural network design could be integrated.

ACKNOWLEDGMENTS

This work is partially supported by ESA's GPU4S (GPU for Space) project (ITT AO/1-9010/17/NL/AF), by the Spanish Ministry of Economy and Competitiveness (MINECO) under grants PID2019-107255GB and FJCI-2017-34095. It received also support by the European Commission's Horizon 2020 programme under the UP2DATE project (grant agreement 871465), the HiPEAC Network of Excellence, the Xilinx University Program (XUP) and XUP Board Partner Red Pitaya.

REFERENCES

[1] Jan-Gerd Meß, Frank Dannemann, and Fabian Greif. Techniques of Artificial Intelligence for Space Applications - A Survey. In *European Workshop on On-Board Data Processing (OBDP)*, 2019.

[2] Jannis Wolf. FPGA BNN Accelerator. `http://www.github.com/JannisWolf/fpga_bnn_accelerator`, 2021.

[3] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe. Binary Neural Networks: A Survey. *Pattern Recognition, Special Issue: Explainable Deep Learning for Efficient and Robust Pattern Recognition*, 105, 2020.

[4] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees 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*, FPGA '17, page 65–74, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, and Trevor Bauer. Xilinx adaptive compute acceleration platform: Versaltm architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, page 84–93, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Joseph James Edwards. *Real-time Computer Vision in Software using Custom Vector Overlays*. PhD thesis, University of British Columbia, 2018.

[7] European Space Agency (ESA). TASTE. https://essr.esa.int/project/taste, 2021.

[8] Maxime Perrotin, Eric Conquet, Julien Delange, André Schiele, and Thanassis Tsiodras. TASTE: A Real-Time Software Engineering Tool-Chain Overview, Status, and Future. In *SDL 2011: Integrating System and Software Modeling, LNCS, Vol. 7083*, pages 26–37, 01 2011.

[9] International Standards Organization (ISO). Information technology - abstract syntax notation one (asn.1): Specification of basic notation. (ISO/IEC 8824-1:2015), 2015.

[10] Tiago da Silva Jorge and Rubén Domingo Torrijos. CORA-MBAD for Zynq 7000. In *ESA TEC-ED & TEC-SW Final Presentation Days*, May 2020.

[11] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[12] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[14] I. Rodriguez et al. GPU4S Bench: Design and Implementation of an Open GPU Benchmarking Suite for Space On-board Processing. Technical Report UPC-DAC-RR-CAP-2019-1, Universitat Politecnica de Catalunya. `https://www.ac.upc.edu/app/research-reports/public/html/research_center_index-CAP-2019,en.html`.

[15] David Steenari, Leonidas Kosmidis, Ivan Rodriquez, Alvaro Jover, and Kyra Förster. OBPMark (On-Board Processing Benchmarks) - Open Source Computational Performance Benchmarks for Space Applications. In *European Workshop on On-Board Data Processing (OBDP)*, 2021.