ASIC-FPGA Gap for a RISC-V Core Implementation for DNN Applications

Abdelrahman Hussein and Hassan Mostafa

***Abstract*—Recently, an emerging instruction set called RISC-V has been considered a paradigm in the computer architecture domain, which has easily got the attention of the hardware community. This is because it is fully open-source, which opens wide horizons for development and innovation. As deep learning applications (DL) are becoming more common, it is very important to evaluate the processors' performance handling such tasks. In these lights, we evaluate a RISC-V core performance executing a DL task. We also measure the gap between its ASIC and FPGA implementations in terms of power and area. We find that ASIC achieves better optimization in both aspects more than the FPGA.**

***Index Terms*—RISC-V, Rocket Chip, ASIC, FPGA, DNN**

I. Introduction

Over the computer levels of abstractions, the hardware-software mapping, which is represented by the Instruction-Set Architecture (ISA) concept, plays a significant role in shaping computer technologies around us. This mapping takes so many loads off developers' and researchers' shoulders, who may be concerned with a certain level of abstraction. This is because the ISA interfaces the software layers with the hardware layers. However, the commercial ISAs are proprietary, which means that it cannot be used by other entities other than their original owners, like the famous Intel x86, IBM Power Architecture, and ARM AARCH32/64. Provided that, it has been becoming more challenging for scholars to develop novel techniques that address the emerging computer architectures. Accordingly, the need for such open-source instructions has rocketed for a long time.

To eliminate this need, an emerging ISA called RISC-V [1] developed at the University of California, Berkeley has been solving this issue. It is completely free-to-use, and its source is available for computer architects to contribute further development to the existing paradigms. The RISC-V ISA has been the backbone of many novel processors, such as the Rocket Chip [2] and the PULPino [3]. Moreover, in 2020, Microchip announced its first SoC FPGA development kit that relies on the RISC-V ISA [4].

Concerning implementing any of these cores, designers usually differentiate between implementing using Application-Specific Integrated Circuit (ASIC) and Field Programmable Gate Arrays (FPGA). While the former usually comprise more trade-off optimization, the latter represents a feasible mean to prototype the design. However, the FPGAs have become no longer limited to prototyping purposes and now are being used to accelerate different functions in data centers.

In this paper, we distinguish the gap between the ASIC and FPGA implementations of one of the famous RISC-V Cores, which is the Rocket Chip [2], in terms of power and area. These trade-offs are measured based on implementation using 28-nm Xilinx Zynq-7000 FPGA and a 28-nm CMOS standard-cell ASIC. We use this to highlight the different efficiencies of both approaches. Also, we evaluate the performance of a certain configuration of the Rocket Chip when a Deep Neural Network application runs on it.

This paper is organized as follows. Section II demonstrates the essential background of this work. Section III introduces the methodology and implementations done within this work. The gap between ASIC and FPGA is illustrated in section IV, while section V concludes the work.

II. Background

A. Rocket Chip

Rocket Chip is a RISC-V SoC generator, which generates a full multi-core system with many components like Rocket Core and memory coherent protocols [2]. It is a parameterizable chip generator that can be easily configured to include or exclude different hardware blocks, such as IEEE Floating Point Unit (FPU), types and number of cores, caches, etc. The generation mainly targets producing tiles which may contain different core types (e.g. In-Order Rocket Core, Out-of-Order BOOM Core). Figure 1 depicts the different components that comprise the Rocket Chip generator. The Rocket Chip Generator is coded in Chisel [5], which is based on Scala.

Technically, the Rocket Chip generator supports different types of RISC-V instructions with the capacity to execute both RV32 and RV64 modes. For example, it supports the base mode (I), with existing extensions that include *Multiply and Divide* (M), and *single-precision* (F), etc. It also provides architects with many useful interfaces like AXI4, AHB, and APB. However, the tile generated provides a distinguished interface called RoCC Interface, which stands for Rocket Custom Co-processor Interface. The significant role of this interface is to provide developers with an easy way to couple their own custom accelerators with the core inside the tile.

978-1-6654-2157-7/21/$31.00 ©2021 IEEE

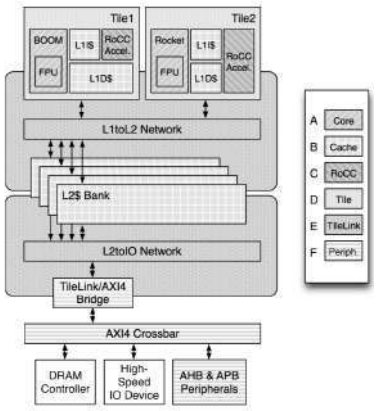

Fig. 1. Different components of which the Rocket Chip generator consists (e.g. cache generator, Tile generator, etc.) [2]

B. Deep Neural Networks

Deep Neural Networks (DNNs) are one example of Deep Learning architectures, which is mainly based on Artificial Neural Networks (ANNs). DNNs have been achieving great progress in numerous tasks and applications such as speech recognition, natural language processing (NLP), and computer vision [6], [7]. Figure 2 shows the general structure of the DNN. Mainly, it consists of input layers (in green), hidden layers (in blue), and output layers (in red).

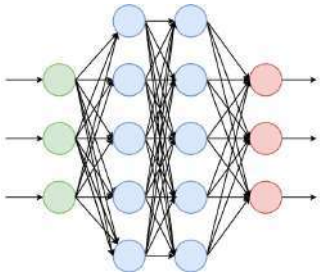

Fig. 2. General representation of a DNN

With regard to computer vision applications, Convolution Neural Network (CNN) has proven a significant success in visual tasks (e.g. object recognition, object detection) [8]. This happens because of its great capacity to learn how to extract features. There are many famous CNN architectures that have achieved state-of-the-art results in various visual tasks like AlexNet [9], ResNet [8], and LeNet-5 [10]. Each of these architectures differs in the number of parameters (i.e. weights), computation cost, and overall performance against different applications. In this work, we use the LeNet-5 to evaluate the Rocket Core performance against a simple deep learning task like classification. Table I depicts the LeNet-5 architecture, where it consists of two convolutional layers that extracts the features of the input image, and fully connected layers to carry out the inference task. It is also important to note that all the activation functions used across the LeNet-5 architecture are all *tanh*, except for the last layer (i.e. output layer), which uses the softmax activation to calculate the likelihood of belonging to each class.

TABLE I
LENET-5 ARCHITECTURE

Layer	Feature Map	Size	Kernel Size
Input	1	32x32	-
CONV Layer	6	28x28	5x5
Average Pooling	6	14x14	2x2
CONV Layer	16	10x10	5x5
Average Pooling	16	5x5	2x2
CONV Layer	120	1x1	5x5
Fully Connected Layer	-	84	-
Fully Connected Layer	-	10	-

C. Hardware Development Flow

Compared to general-purpose processors, which can be used to execute different types of functions via software development, custom hardware development provides dominant performance if it is adopted to implement the same function. This is because it leverages its nature to execute a function without the need to transform the function into specific instructions of the processor, which may leak required parallelism. In [11], the author highlights this significant gap between general-purpose processors, and custom hardware (i.e. ASIC and FPGA). It can be concluded that custom hardware provides a higher level of parallelism, hence, a better overall performance (for an application that reflects sufficient parallelism). This is because the architect in such scenario can duplicate the number of Processing Engines (PEs), which accordingly increases the level of parallelism.

Conventionally, custom hardware implementation can be done using either FPGA or ASIC. FPGA, as its name reflects, provides developers with extensive reconfigurability, unlike ASICs, because of its internal structure. Internally, an FPGA consists of Configurable Logic Blocks (CLBs), memory blocks, Digital Signal Processing (DSP) slices. All these items are distributed over a grid of programmable interconnects that connect all FPGA blocks to each others and to the outer IO cells. All of these building blocks are comprised in figure 3.

On the other hand, the ASIC flow relies on a given CMOS standard-cell library, which is a standard approach to design ASIC [12], [13]. This requires the architects to consider various development phases to ensure that the chip design shall perform the expected task. On the contrary to FPGA, ASIC flow does not reflect any flexibility after development. In return, it achieves way less power consumption and better performance than FPGA [11].

III. METHODOLOGY AND IMPLEMENTATION

With regard to the Rocket Chip used, we generate a Rocket Tile, which includes an in-order Rocket Core coupled with an

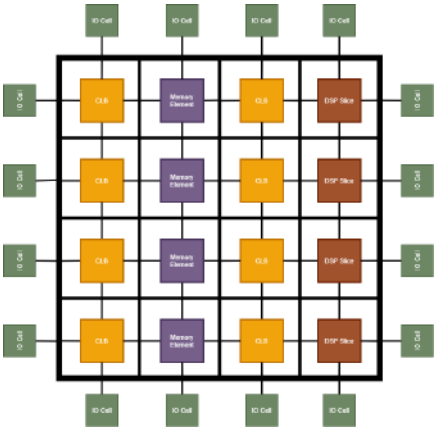

Fig. 3. Internal FPGA structure

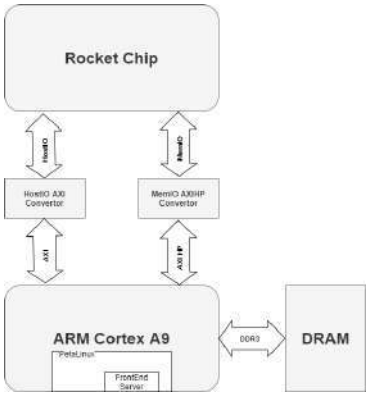

Fig. 4. Tethering Rocket Chip with the programmable system (PS) side

FPU, and L1 Cache. For the ASIC and FPGA resources and power evaluation, caches are excluded.

A. FPGA Flow

To implement the Rocket Chip on the FPGA, we used Xilinx Zynq-7000 Zedboard FPGA, which has an ARM dual-core Cortex A9 MPCore. For this purpose, we adopt the official Github repository [14]. The Rocket Tile is tethered with the ARM processor as shown in figure 4. The CPU runs a Xilinx PetaLinux operating system, on which a Front-End server runs. This server is responsible for executing RISC-V instructions on the Rocket Tile. The Rocket Chip in this implementation operates on 25 MHz.

We use this implementation for two purposes; to identify resource utilization for the Rocket Tile on FPGA, and to evaluate the number of execution clock cycles required to perform a DL task. For the Rocket Tile, the post-implementation resources utilization and dynamic power on FPGA are concluded in tables II and III, respectively. The post-routing layout is highlighted in red in figure 5. Since the FPGA contains DSP slices that perform mathematical operations, we constrained the design so that it would not infer a DSP slice. This is for the purpose of making a fair comparison with the ASIC flow.

TABLE II
SINGLE-CORE ROCKET TILE RESOURCES UTILIZATION ON FPGA

Module	Slice LUTs	Slice Registers
Rocket Core	5.9K	2K
TLMasterXBar	44	15
PTW	0.6K	0.5K
FrontEnd (For Branching Purposes)	3.2K	3.4K
FPU	16.5K	3.8K
TLBuffer	0.5K	15
Rocket Tile (Total)	**26.7K (50.2%)**	**9.7K (9.17%)**

TABLE III
SINGLE-CORE ROCKET TILE RESOURCES UTILIZATION ON FPGA

Module	Dynamic Power (in mW)
Rocket Core	14.36
TLMasterXBar	0.018
PTW	2.733
FrontEnd (For Branching Purposes)	9.101
FPU	9.152
TLBuffer	1.434
Rocket Tile (Total)	**36.8**

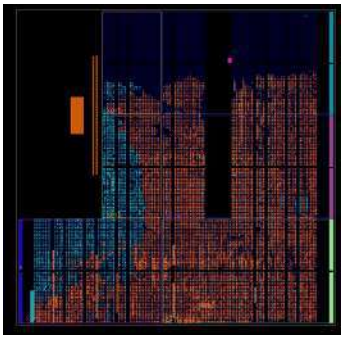

Fig. 5. Post-Routing layout of Rocket Tile (highlighted in red)

B. Performance Evaluation

In order to evaluate the performance of the Rocket Chip executing a Deep Learning task, we use LeNet-5 architecture coded in C [15]. The network is trained to infer two different datasets; MNIST hand-written digits (figure 6), and MNIST Fashion (figure 7). The model is trained for each of the two datasets and achieved test accuracy 96% for the hand-written digits, and 91% for the fashion dataset. The model is then cross-compiled using the RISC-V tool-chain into the compatible instructions. Since this task comprises floating-point tasks, it is essential to configure the Rocket Tile to include an FPU block.

The performance is evaluated using the scheme shown in figure 4 in order to determine the number of clock cycles spent to successfully perform the inference task. For accuracy considerations, the number of clock cycles is fetched from the Control/Status Register (CSR). The inference latency was 2.2 seconds/image, operating at the same frequency 25 MHz. The inference latency is the same for images from any of the two datasets because the execution relies on the compiled instructions, which does not change by changing the dataset.

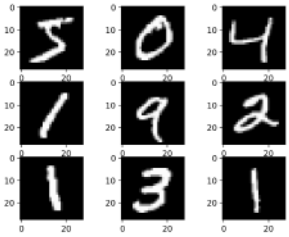

Fig. 6. MNIST hand-written digits dataset

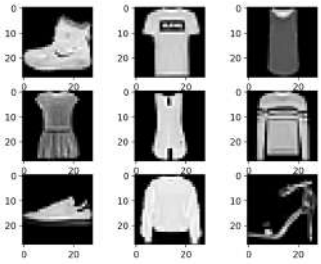

Fig. 7. MNIST fashion dataset

C. ASIC Flow

To get the required insights about the ASIC-FPGA gap, we synthesize the same Rocket Tile to generate the equivalent netlist. The synthesis is done using Synopsys Design Compiler. The Rocket Tile operates at a frequency of 25 MHz, which is identical to its counterpart on FPGA. The chip post-synthesis layout is shown in figure 8. All memory blocks (e.g. caches) are excluded during this process. Area and power consumption of the design are presented in table IV, based on UMC 28-nm standard-cell technology.

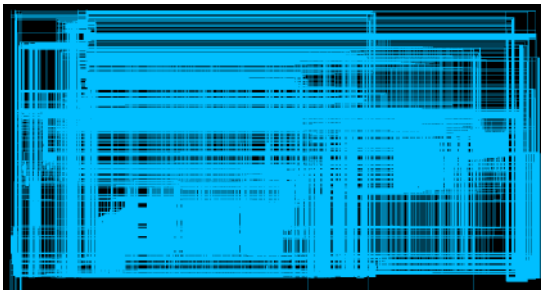

Fig. 8. Post-Synthesis layout of Rocket Tile

TABLE IV
ROCKET TILE POST-SYNTHESIS AREA AND POWER CONSUMPTION USING

Operational Voltage (in V)	Area (in μm^2)	Power (in mW)
0.7	52.7K	11.24
1.05	52.7K	25.30

IV. ASIC-FPGA GAP REALIZATION

A. Power Gap

As stated in tables III and IV, the FPGA consumes at least 1.5x more power for the same Rocket Tile than the power consumed by the ASIC when it operates on 1.05 operational voltage. The ASIC power consumption drops to lower power consumption (11.24 mW), which makes the FPGA consumes 3.2x more power than the ASIC. This raises the ASIC as an efficient choice for applications that need significant power optimization.

B. Area Gap

As demonstrated in section II, there are many significant differences between ASIC and FPGA in terms of area and power. For this purpose, Rocket Tile's implementation on ASIC and FPGA is compared in terms of these two factors. Both flows can estimate the power consumption directly with a few commands to execute after synthesis or routing. However, in terms of the area, it is much easier to determine the chip area in ASIC than it is in FPGA. This is because the FPGA flow relies on reporting resources utilization not the actual total area of resources that are consumed by the design.

However, this can be possible if the area of each of the different individual slices is known. We estimate the area of the LUT slice and the register slice using the method proposed in [16], which can be summed up in table V.

TABLE V
FPGA CELL AREA AND ROCKET TILE TOTAL AREA

Cell Type	Cell Area (in μm^2)	Count	Total Area
Register	9.03952	9.7K	87.683K
LUT	9.05388	26.7K	241.738K
Total Area			329.421K

As noticed, the FPGA achieves more than 6.2x larger area than its counterpart on ASIC. This does not take into consideration the routing for both, which will likely affect the individual area for each of them.
This can be justified because the FPGA would need more logic elements to perform a specific function, due to its limited resources, unlike ASIC, which relies on a larger library that contains a variety of cells that differ in all trade-offs. This makes the optimization task much easier in ASIC flow than it is in FPGA flow.

V. CONCLUSION

In this article, we generated a Rocket Tile that contained a single-core in-order Rocket Core with an FPU. We evaluated its performance to handle a DL inference task by measuring the inference latency determined from the number of clock cycles required to achieve this task. Furthermore, we distinguished the power and area gaps between ASIC and FPGA flows. We found that ASIC offered a more optimized implementation than its counterpart in FPGA. This should be considered for our future attempt to design an accelerator for the Rocket Chip.

REFERENCES

[1] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi, "The risc-v instruction set manual. volume 1: User-level isa, version 2.0," CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, Tech. Rep., 2014.

[2] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[3] A. Traber, F. Zaruba, S. Stucki, A. Pullini, G. Haugou, E. Flamand, F. K. Gurkaynak, and L. Benini, "Pulpino: A small single-core risc-v soc," in *3rd RISCV Workshop*, 2016.

[4] A. Chandler. The industry's first soc fpga development kit based on the risc-v instruction set architecture is now available. [Online]. Available: https://www.microchip.com/pressreleasepage/industry-s-first-soc-fpga-development-kit-for-risc-v

[5] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 1212–1221.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] K. Noda, Y. Yamaguchi, K. Nakadai, H. G. Okuno, and T. Ogata, "Audio-visual speech recognition using deep learning," *Applied Intelligence*, vol. 42, no. 4, pp. 722–737, 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[10] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[11] E. Vansteenkiste, "New fpga design tools and architectures," Ph.D. dissertation, Ghent University, 2016.

[12] D. Chinnery and K. Keutzer, *Closing the gap between ASIC & custom: tools and techniques for high-performance ASIC design*. Springer Science & Business Media, 2002.

[13] M. J. S. Smith, *Application-specific integrated circuits*. Addison-Wesley Reading, MA, 1997, vol. 7.

[14] Rocket chip on zynq fpgas. [Online]. Available: https://github.com/ucb-bar/fpga-zynq

[15] Lenet-5. [Online]. Available: https://github.com/fan-wenjie/LeNet-5

[16] N. Gamal, H. Fahmy, Y. Ismail, and H. Mostafa, "Design guidelines for embeded nocs on fpgas," in *2016 17th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2016, pp. 69–74.