Towards Reconfigurable Accelerators in HPC: Designing a Multipurpose eFPGA Tile for Heterogeneous SoCs

Tim Hotfilter*, Fabian Kreß, Fabian Kempf, Jürgen Becker
Karlsruhe Institute of Technology (KIT)
{hotfilter, fabian.kress, fabian.kempf, becker}@kit.edu

Juan Miguel de Haro*, Daniel Jiménez-González, Miquel Moretó, Carlos Álvarez, Jesús Labarta
Barcelona Supercomputing Center (BSC)
Universistat Politècnica de Catalunya (UPC)
{juan.deharoruiz, djimenez, miquel.moreto, carlos.alvarez, jesus.labarta}@bsc.es

Imen Baili
Menta S.A.S.
imen.baili@menta-efpga.com

***Abstract*—The goal of modern high performance computing platforms is to combine low power consumption and high throughput. Within the European Processor Initiative (EPI), such an SoC platform to meet the novel exascale requirements is built and investigated. As part of this project, we introduce an embedded Field Programmable Gate Array (eFPGA), adding flexibility to accelerate various workloads. In this article, we show our approach to design the eFPGA tile that supports the EPI SoC. While eFPGAs are inherently reconfigurable, their initial design has to be determined for tape-out. The design space of the eFPGA is explored and evaluated with different configurations of two HPC workloads, covering control and dataflow heavy applications. As a result, we present a well-balanced eFPGA design that can host several use cases and potential future ones by allocating 1% of the total EPI SoC area. Finally, our simulation results of the architectures on the eFPGA show great performance improvements over their software counterparts.**

***Index Terms*—FPGA, HPC, Design space exploration, SoC**

Introduction

Recently, a trend from general purpose High-Performance Computers (HPC) towards platforms with application-specific accelerators due to lower energy consumption and faster execution can be observed [1]. The European Processor Initiative (EPI) builds such a high-performance platform. EPI with its large consortium addresses multiple domains concurrently: A processor cluster, dedicated EPI accelerators and an eFPGA, among others [2]. All platform components are connected to each other via a well-defined interconnect, which allows us to develop different aspects of the platform independently.

In the scope of this article, we present the final design and a performance evaluation of the eFPGA tile, which adds flexibility to the SoC since it allows for reconfiguration during run time. Although the eFPGA can be reconfigured, its initial layout, like the amount of Look-Up Tables (LUTs) has to be defined before chip production. However, during design phase, constraints such as the area or the power must be considered. Hence, a careful exploration of the eFPGA design space along the performance, power and area metrics is required to ensure that it can host the given use cases and provides sufficient eFPGA primitives to meet future applications as well.

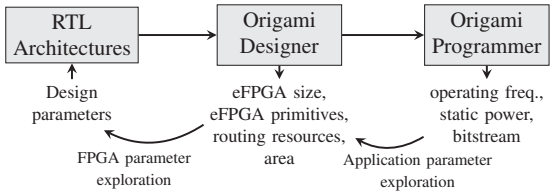

Fig. 1: Overview of the design flow using the Menta tools

The eFPGA initial layout is derived based on two highly customizable architectures to evaluate the design space, which represent common HPC workloads, and thus the market EPI addresses. Recently, open-source frameworks have been published addressing the automated generation of eFPGA layouts [3], [4]. However, these focus on generating the layout by considering only one use case rather than multiple applications. In the scope of EPI, none of these tools could be applied to determine the final eFPGA design. Consequently, to find the best performance to area trade-off, we evaluate multiple configurations for each architecture. Therefore, we utilize the Menta toolflow (Figure 1): With Origami Designer we can adjust the eFPGA layout and derive power and area metrics, and with Origami Programmer we can place and route our architectures to get performance metrics. In particular, our contributions are threefold:

- We describe two highly configurable architectures: Picos, a hardware dependence manager and a convolutional neural network (CNN) accelerator.
- We show how we co-designed the eFPGA tile to maximize the utilization of the eFPGA resources with a 1% area constraint of the total EPI chip area.
- We present and evaluate the final eFPGA layout with our architectures in a simulated EPI GPP environment.

European Processor Initiative (EPI) project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 826647, from Spanish Government (PID2019-107255GB-C21/AEI /10.13039/501100011033), and from Generalitat de Catalunya (contracts 2017-SGR-1414 and 2017-SGR-1328). M. Moreto is partially supported by the Spanish Ministry of Economy, Industry and Competitiveness under Ramon y Cajal fellowship No. RYC-2016-21104.

*Both authors contributed equally

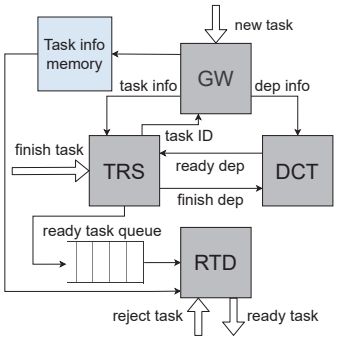

Fig. 2: Overview of the internal design of Picos

Architectures and Use Cases

Although the eFPGA can be reconfigured during runtime, the initial layout has to be defined before the chip is manufactured. Therefore, we perform our architecture evaluation based on two distinct use cases: One controlflow and one dataflow driven. This approach allows us to cover a wide range of potential, yet unknown architectures and future use cases.

Task Dependence Manager: Picos

Task-based programming models such as OpenMP and OmpSs have proven helpful for developing parallel applications due to their simplicity and ease of use. By annotating tasks and their data dependencies with pragmas, a software runtime system dynamically determines which tasks can be executed in parallel.

Picos [5] is a fast hardware implementation of the dependence detection algorithm, suitable for any task-based programming model. Therefore, it can substitute the traditional software runtime and transparently speed up any fine-grained parallel application by significantly reducing the runtime overhead. Figure 2 shows a block diagram of Picos internal design. The Gateway (GW) is the entry point of new tasks. It splits the task properties (e.g. dependence addresses, identifier) between the Task Reservation Station (TRS) and the Dependence Chain Tracker (DCT). Additional information, which is irrelevant to Picos but important to other components, may be stored in the Task info memory. The TRS controls the state of a task (ready/not ready to be executed), and the DCT controls the state of a dependence (whether it depends on other tasks or it is ready). The Ready Task Dispatcher (RTD) communicates with the corresponding external scheduler (hardware or software-based), notifying a processing element to execute a task. If a task cannot be immediately executed due to lack of resources, the `reject task` interface can be used to retry later.

Convolutional Neural Network Accelerator

CNN algorithms have shown tremendous results in image recognition recently and are a key to autonomous driving and moving now into embedded devices. To find a very versatile embedded FPGA configuration, we choose in addition to Picos a data-flow driven CNN accelerator. Our CNN accelerator is optimized towards scalability and a high degree of configurability, which allows for the eFPGA design space exploration. It can work with any prior unknown input data and optimizes the computationally intense convolution operations (CONV), while not neglecting other steps like activation.

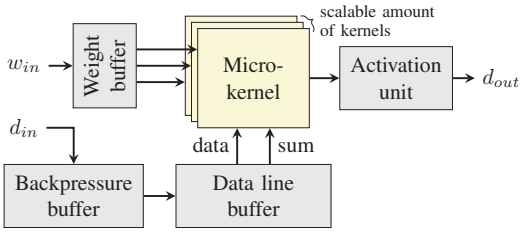

Fig. 3: Overview of the CNN accelerator architecture

An overview of the accelerator is given in Figure 3 with the *microkernel* units at its center, which compute the individual CONVs. The number of microkernel units is scalable, as well as the amount of MACs inside each of them. Besides the number cruncher, the architecture features three buffers: A line buffer to rearrange incoming data, a weight buffers and a backpressure buffer. The microkernels and the buffering scheme are inspired by a work from Qui et al. [6], which we extended with a coarse grained sparsity detection via the sum of an input feature kernel. The reuse of the inputs allows us to perform a convolution operation in a low-latency fashion, especially with low batch sizes, which are common in embedded real-time applications. Besides CONV and activation, the configuration of the accelerator, the setup of memory streams and operations such as pooling are handled by the GPP. Kernel dimensions might be configured dynamically during runtime, while number formats, the amount of microkernels or the buffer sizes have to be synthesized into the accelerator.

Design Space Exploration

Finding a viable eFPGA configuration poses the challenge to host both presented use cases and future user architectures in an eFPGA. Even they do not run at the same time, each use case requires different FPGA primitives and IPs. Since both are designed highly configurable, multiple configurations of each use case are mapped on different eFPGA configurations while meeting the project requirements like a total area budget of 1% and the application requirements.

Picos Parameter Exploration

For Picos, we study different configurations to find the best performance for a variety of applications. The following parameters can be configured in Picos:

- Task Memory (TM) size. The TM stores task data such as the identifier and the number of free dependencies.
- Dependence Memory (DM) size. The DM stores dependence addresses for address matching.
- Version Memory (VM) size. The VM is used to find dependence relationships between tasks with the same addresses, and establish an execution order.

TABLE I: Overview of the dimensions and static power of the eFPGA, the number of macro blocks and the utilization as well as operation frequency of the two architectures

Parameter	Value	Utilization / Perf. NN Acc	Picos
Technology	7 nm	-	-
Static Power	1.7393 mW	-	-
LUT6	9632	78%	51%
Flip-Flops	12086	72%	29%
DSPs	52 × I16_32P	92%	0%
Memories	80 × 2Kx16	5%	46%
Operating freq. [MHz]		159	100
Matrix shape	38 × 40	-	-

A task with n dependencies uses a single slot in the TM and n slots in the VM. On the DM side, depending on if previous tasks reference the same addresses, the number of slots may range from 0 to n. When any of the three memories are full, Picos is not able to accept more tasks. Hence, the DM and VM should be bigger than the TM to maximize memory usage. The size of TM should be maximized, however, we have to maintain a good ratio with the other memories. Based on the benchmarks introduced in the performance evaluation, we found that four dependencies per task are sufficient.

Microkernel Parameter Exploration

First we evaluate common neural network structures that are used as benchmarks, such as SqueezeNet [7], which is the basis for the face recognition use case in EPI [8], because of its small memory footprint. The size of the input buffers is determined by the size of the input features. For 3x3 CONVs, we have to store at least two lines of 244x244 pixel input features. Thus, the input buffer size is set to 488 words.

The DSP configuration depends on the width of the operands and accumulators. With linear quantization, we can reduce the precision of weights and input features. Although this has an impact on the prediction accuracy, little quantization as only small impact, while the computational effort can be reduced massively. We achieve a SqueezeNet baseline accuracy of 86 % while training from scratch with CASIA-Web faces dataset [9] and found 8 bit an optimal quantization level with 81 % accuracy. Hence, the DSP operands can be set to 8 bit. For the accumulator we choose 16 bit, which is sufficient, as the intermediate values do not exceed 16 bit.

The performance of the CNN accelerator heavily depends on the number of available DSPs. Each microkernel has three MACs and SqueezeNet has an even amount of CONVs, thus we have to set the amount of DSP to a multiple of six.

Exploration Results

Our eFPGA configuration has to meet the requirements of the two presented use cases, with enough spare resources to be capable of hosting future, yet unknown architectures. Picos has to optimize towards a large memory, while the CNN accelerator optimizes towards a high number of MACs. Since the interface to the processor cluster was not established at the time of the eFPGA layout, we had to leave margin for the communication protocol with the NoC.

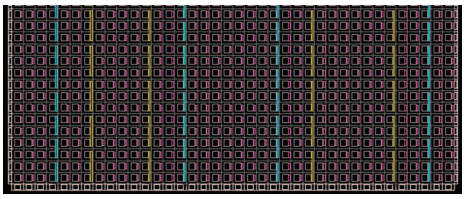

Fig. 4: Architecture segment of the eFPGA configuration, showing DSPs, I/Os and memories highlighted in teal, white and yellow, respectively

With respect to the 1% share of total area, in a square shape, we can fit an overall of 38 columns and 40 rows. The final eFPGA configuration is detailed in Table I and the layout is shown in Figure 4. All parameters are derived from Origami Designer and the frequency from Origami Programmer after the place and route of each individual architecture. The design features four columns of DSPs and four columns of memories. Memory cells are evenly distributed over the eFPGA and DSPs are located close to the memories with some logic resources in between to enable quick access to the data.

The total of 52 DSPs and 80 memories allow for a wide range of applications. However, while more DSPs or memories might favor the CNN accelerator or Picos, respectively, we aimed for a relatively high amount of LUTs and FFs to keep the eFPGA tile versatile towards future architectures that might run on the EPI chip. Therefore, we also choose 16-bit inputs and 32-bit accumulators for the DSPs, especially for applications that require higher precision. The memory cells are configured with a fixed word count and word bit-width of 2048 and 16 respectively. In order to maximize memory usage of Picos, we set the DM and VM memory size to the 2048 word count and to maintain the 4:1 ratio with the TM, we set its size to 512 accordingly. This configuration allows an average of four unique dependencies per task, and a total of 512 in-flight tasks. Regarding the CNN accelerator, with the total of four DSP columns we can place 16 microkernels, which accounts for an overall utilization of 48 of the 52 total available DSPs. Given the available memories of the architecture, we are able to place all required buffers in the memory cells.

Performance Evaluation

The eFPGA tile can be developed independently. Hence, to evaluate the performance of the architecture configurations, we carry out functional verification in simulation and on FPGA devices. The communication overhead and latency is added to the simulation to generate numbers that fit the later platform.

For Picos, we measure the performance of a selected set of benchmarks in a simulated SMP environment on a Xilinx Zynq UltraScale+ FPGA@100MHz that represents a similar setup to the EPI chip. Our setup simulates the execution time of an application with an arbitrary number of cores. These cores are controlled directly by Picos and a simple hardware scheduler, which receives tasks extracted from a real execution trace and schedules them on the first free simulated core. The

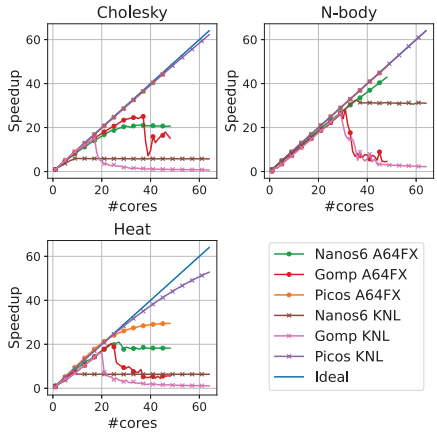

Fig. 5: Speedup against sequential execution of different runtimes

benchmarks used are the following: The N-body simulation of the force interaction between astronomical objects, their movement, and speed; Cholesky factorization of a matrix and heat propagation on a 2-dimensional surface using a Gauss-Seidel method.

We execute the benchmarks in two different clusters to generate simulator traces and measure the performance with Gomp (OpenMP) and Nanos6 (OmpSs-2) as software runtimes: An Intel Xeon Phi CPU 7230 64-core processor (KNL) and a 48-core Fujitsu A64FX ARM CPU node. We run each benchmark with the same task granularity and problem size of 500k-1M tasks, only changing the number of cores. Since the artificially introduced communication overhead heavily depends on the final processor architecture, we use a very pessimistic latency estimation, according to the *lat_mem_rd* benchmark of the *lmbench* suite on the A64FX core. We add 40 extra cycles or 400ns to each task execution.

Figure 5 displays speedup plots scaling with the number of cores. Except for the Heat benchmark, the results show that Picos can scale almost perfectly. This is due to the limited memory of Picos, mainly the TM, which prevents it from finding enough parallelism for further scaling. All results show that Picos outperforms Gomp and Nanos6 with a significant difference, in short tasks. Nanos6 shows a stable speedup when it reaches the core limit, whereas Gomp tends to scale down with high variability. However, In most cases, both runtimes cannot scale with more cores due to the overhead they add to the execution time. While long tasks can benefit from both software and hardware runtimes, fine-grain applications can be handled much faster by Picos.

The CNN accelerator is evaluated with SqueezeNet for face recognition, which requires about 962 million 8 bit MAC operations. This translates into 306 million microkernel operations. Since the neural network computation is dataflow driven and most of its operations are independent, the simulation shows that we can reach full utilization if the bandwidth is sufficient. Running the eFPGA at the maximum operation frequency this yields a bandwidth of 159 MB/s for reading inputs and writing results back. Different microkernels can operate on the same input data with a different set of weights, which might be reused and hence not be reloaded from the memory. Considering a worst-case scenario in the given SqueezeNet topology, the smallest convolution layer performs 12,544 CONVs with the same set of weights. After this amount of operations the weights have to be exchanged, which adds on average 1 MB/s over the course of the processing. Due to the availability of a high-performance NoC, this bandwidth can be reached. With full utilization, we can compute all operations of the optimized SqueezeNet in $\sim 150ms$.

Conclusion

An embedded FPGA in a SoC like the EPI platform allows for a fast deployment of specific hardware accelerators. In the scope of this work, we elaborated on a tightly-coupled setup of an eFPGA in a System on Chip and guided the design process for the initial eFPGA structure. Therefore, we defined two state-of-the-art use cases that represent a broad range of HPC workloads, to evaluate the design possibilities. Our found eFPGA configuration can fit both in a 1% share of the total chip area, without suffering in performance. With 38 rows and 40 columns, we are able to embed 9632 LUTs, 12086 FFs, 52 DSPs and 80 memories. The configuration maintains the flexibility of the eFPGA by budgeting enough spare resources, which enables the eFPGA to host further yet unknown use cases. The final eFPGA design supports the EPI platform in accelerating various applications in an adaptive way. In the phase of the project, it can be easily integrated into the silicon to demonstrate the performance of the overall platform.

References

[1] Iris Walter et al. Embedded face recognition for personalized services in the assistive robotics. In *ECML PKDD Workshops*. Springer, 2021.

[2] Mario Kovač, Philippe Notton, Daniel Hofman, and Josip Knezović. How europe is preparing its core solution for exascale machines and a global, sovereign, advanced computing platform. *Mathematical and Computational Applications*, 25(33):46, Sep 2020.

[3] Dirk Koch, Nguyen Dao, Bea Healy, Jing Yu, and Andrew Attwood. Fabulous: An embedded fpga framework. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, page 45–56, New York, NY, USA, 2021.

[4] Ang Li and David Wentzlaff. Prga: An open-source fpga research and prototyping framework. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, page 127–137, New York, NY, USA, 2021. Association for Computing Machinery.

[5] Xubin Tan, Jaume Bosch, Carlos Álvarez, Daniel Jiménez-González, Eduard Ayguadé, and Mateo Valero. A hardware runtime for task-based programming models. *IEEE Transactions on Parallel and Distributed Systems*, 30(9):1932–1946, 2019.

[6] Jiantao Qiu et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, page 26–35, New York, NY, USA, 2016.

[7] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016.

[8] Tim Hotfilter, Fabian Kempf, Jürgen Becker, Dominik Reinhardt, and Imen Baili. Embedded image processing the european way: A new platform for the future automotive market. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6, 2020.

[9] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. Learning face representation from scratch. *arXiv:1411.7923 [cs]*, Nov 2014.