Heterogeneous Dataflow Hardware Accelerators for Machine Learning on Reconfigurable Hardware *

Hendrik Woehrle[1], Johannes Teiwes[2], Mario Michael Krell[2], Anett Seeland[1], Elsa Andrea Kirchner[1,2], and Frank Kirchner[1,2]

[1] German Research Center for Artificial Intelligence, DFKI Bremen, Robotics Innovation Center, Robert-Hooke-Str. 1, 28359 Bremen, Germany
hendrik.woehrle@dfki.de,
[2] University of Bremen, Faculty 3 – Mathematics and Computer Science, Robotics Lab, Robert-Hooke-Str.1, 28359 Bremen, Germany,

Abstract. The trend to develop increasingly more intelligent systems leads directly to a considerable demand for more and more computational power. However, in certain application domains, such as robotics, there are several technical limitations, like restrictions regarding power consumption and physical size, that make the use of powerful generic processors unfeasible. One possibility to overcome this problem is the usage of specialized hardware accelerators, which are designed for typical tasks in machine learning. In this paper, we propose an approach for the rapid development of hardware accelerators that are based on the heterogeneous dataflow computing paradigm. The developed techniques are collected in a framework to provide a simple access to them. We discuss different application areas and show first results in the field of biosignal analysis that can be used for rehabilitation robotics.

Keywords: Robotics, Embedded Systems, FPGA, Hardware Acceleration, Dataflow

1 Introduction

Machine learning techniques are widely used nowadays for a broad range of applications. Depending on the specific application, they are employed either on commodity hardware like desktop PCs or powerful high-end systems, like clusters. The primary objective in these settings is to achieve a good accuracy of the methods. Other objectives, like computational efficiency and memory consumption are often regarded as less important or even entirely ignored.

However, with the increasing importance of portable and embedded systems and in the eras of Big Data and wearable computing, these formerly secondary objectives become more and more important. Especially in the case of

* This work was supported by the *German Federal Ministry of Economics and Technology* (BMWi, grants FKZ 50 RA 1012 and FKZ 50 RA 1011).

autonomous systems and robotics, stringent requirements have to be satisfied: 1) the available physical space is limited, which often prohibits the usage of off-the-shelf components like desktop PCs, 2) the power consumption should be small to reduce heat generation, allow the usage of small accumulators, and increase the running time of the system, 3) often real time processing is needed, since the systems have to work in a real world environment and have to keep up with the environment.

1.1 The Problem of Generic Processors

These requirements make the usage of standard *generic* processors suboptimal. The main purpose of generic processors is the execution of arbitrary software, but not the high performance execution of *specific* algorithms. Consequently, they are 1) either bulky and powerful, or small but weak, 2) waste to much energy for the computational power they provide, 3) can not guarantee real time processing themselves, but require real time operating systems. Fortunately, genericness is not always required. Especially many algorithms in machine learning and signal processing depend on a small set of operations like matrix-vector computations (e.g., in neural networks, and support vector machines (SVMs)) or convolution (e.g., in finite impulse response (FIR) filtering or edge detection in image processing).

The above stated facts make it possible to use a dichotomy here: combine a *generic*, but *weak* CPU for software tasks, with *application specific hardware accelerators* for high performance computations. This pattern is widely used. Examples are the usage of specific accelerators, e.g., to reduce the energy consumption in smart phones [1] or to perform machine learning tasks in the Microsoft Kinect [2]. In these devices application specific integrated circuits (ASICs) are used to fulfill a single, specific task. ASICs can not be transferred to other applications - every time the requirements change, a new ASIC has to be constructed. This time consuming and expensive and therefore only reasonable if large quantities are produced. Consequently, ASICs are inflexible, since it is impossible to consider improvements of the underlying algorithm after the ASIC is manufactured. This is not feasible for robotics or machine learning in mobile systems.

Another example is generic computing on graphics processing units (GPUs), which is also often used in the machine learning community [3]. However, GPUs have a high power consumption and are rarely available as individual chips to, e.g., place them on printed circuit boards (PCBs) that have to be built to satisfy the space constraints in robotic systems.

1.2 Field Programmable Gate Arrays

One solution approach to overcome these problems is based on Field Programmable Gate Arrays (FPGAs). FPGAs consist of generic logic elements that can be configured to form specific circuits to implement an algorithm *in hardware*. This has several advantages for machine learning and robotics. First, the hardware implementation of an algorithm can provide significant performance improvements

while keeping the power consumption low. Second, since the configuration process is very flexible, the disadvantages of ASICs regarding development costs do not apply here. The algorithm can be modified if needed, since the circuit can be changed by a reconfiguration of the FPGA.

Traditionally, FPGAs were used as simple, but flexible logic elements or to provide other simple functionalities in electronic devices. However, in the last couple of years the application areas were extended to other fields, such as digital signal processing. Vendors integrate components into FPGAs to further improve the usability in these application areas. Examples are DSP slices such as the DSP48 slice in Xilinx FPGAs [4] to efficiently perform multiply-accumulate operations, or memory elements such as block RAMs [5] to buffer data. A further advantage is the inherent real time capability of the FPGA. It is possible to design the circuit in such a way that it executes an algorithm in an exact number of clock cycles to meet time constraints.

However, these advantages are not for free: a major problem of FPGAs is the design complexity that requires careful attention of the FPGA designer. For example, the designer has to decide for every arithmetic operation, if it should be performed as a single or double precision floating or even fixed point operation. To save resources, the latter is preferred, but this can result in numerical problems. Furthermore, the exact timing of all operations has to be specified and the circuit must be made accessible for the software side. Up to now, this design complexity has prevented FPGAs from being widely used in the machine learning community.

1.3 FPGAs for Machine Learning

Often, FPGAs are still used as *classical* electronic components: to provide glue logic or otherwise simple functionality like low-level communication. However, there are various approaches that use FPGAs for increasingly more complex tasks that range from simple signal processing to complex control architectures.

Furthermore, FPGA implementations for different popular machine learning applications are presented in the literature, e.g. neural networks [6] or support vector machines [7]. However, most approaches are *singular*, i.e., they conduct just a *single* functionality, without any possibility of generalization or transferability to other applications. A generic framework for machine learning and robotics has been proposed in [8]. However, due to its design, it is only suitable for stationary high performance systems, but not for robotics. Furthermore, the framework is only usable in a concrete hardware setup. In order to facilitate the usage of FPGAs for the development of innovative machine learning-based applications in future robotic and mobile systems, the typical *engineering* problems have to be *hidden* or *simplified*. In order to achieve this, the following points are of importance:

1. The possibility to rapidly implement typical algorithms in the field of machine learning as hardware accelerators.
2. It should be possible to easily integrate third party implementations, so called intellectual property (IP) cores, into the framework.

3. One should be able to easily verify the functionality of the hardware accelerator.
4. Mechanisms to simplify the accessibility to the hardware accelerator from the software side have to be provided.
5. Mechanisms to automatically optimize the design regarding required FPGA-resources, given a set of defined constraints, are required.

In this paper, we discuss an approach to meet these requirements by proposing the *re*configurable *S*ignal *P*rocessing *A*nd *C*lassification *E*nvironment (*reSPACE*). It is designed to reduce the complexity of development while retaining the most important advantages especially in the field of machine learning and signal processing, the later is especially relevant since often an appropriate feature extraction is of high importance [9]). We illustrate the properties of reSPACE on an example of biomedical signal processing, namely the realtime prediction of movements based on single trial analysis of the human electroencephalogram. In future, this can for example be used in the field of rehabilitation robotics embedded in a wearable assistive robotic device.

2 Accelerator Hardware Architecture

The proposed framework is based on the *static heterogeneous synchronous dataflow* computing paradigm. A dataflow-like concept is used in frameworks that are popular for machine learning such as MDP [10], scikits-learn [11] or pySPACE [12]. Using reSPACE, it is easy to implement FPGA-based application specific dataflow accelerators (DFAs) that speed up machine learning operations.

2.1 Heterogeneous Synchronous Dataflow Computing Paradigm

In the dataflow computing paradigm, data is streamed through a sequence of algorithms and transformed on its way through them [13]. In the following, we call this sequence a *flow*, and the implementations of the specific algorithms *nodes*. Since it is possible to combine different nodes, we use a *heterogeneous* dataflow paradigm. Since the structure is pre-defined for a specific application, it is a *static* dataflow. However, in contrast to software dataflow concepts, the data is shifted by one step through the flow on each clock tick of the system, resulting in a *synchronous* design.

In reSPACE, we provide two different operating modes of the system. The *stream mode* allows to process an ongoing stream of data. This mode is required when the signal for processing originates directly from analog to digital converters, e.g., force-torque sensors or electrodes for reading biosignals. The other operating mode is designed to process separate *windows* of data, where the samples within a window are adjacent to each other. Examples are feature vectors consisting of single feature elements or images that consist of pixels. We use a model-based design approach here that is currently based on System Generator for DSP [14] or VHDL. To *implement a complete system*, we provide two different alternatives.

Either a *library of predefined, parametrized nodes* can be used, that contains basic, widely used algorithms such as FIR and IIR filters, direct current offset removal, standardization, etc., are provided as directly usable nodes that can be directly arranged to build up the overall system. These nodes are generic and can be instantiated using different sets of parameters. This allows the designer to configure the overall system for, e.g., different number of channels or different precision of the calculations.

The other opportunity is *customizable circuit generation for matrix-multiply based algorithms*. We provide mechanisms to generate specific circuits for resource efficient parallel matrix operations. The multiply accumulate operations are mapped to the DSP slices of the FPGA. This is done using a domain specific language that allows to specify a sequence of matrix vector operations and choose a parallel or serial implementation. The parallel implementation uses a minimal number of clock cycles but high amount of logic resources, whereas the serial implementation is resource efficient but takes more cycles to run.

2.2 System Architecture

There are at least two possibilities for the integration of DFAs into a system:

Inside a *System on Chip (SoC)* In this setup, the DFA is connected to a host CPU by some type of bus system, e.g., an AXI bus [15]. This setup is applicable if the main system is controlled by software that is running on the host CPU.

Direct access This setup is useful if certain processing should be performed in a decentralized way. Applications are sensor data processing in proximity to a sensor to, e.g., perform *complex* preprocessing or dimensionality reduction of the data or to implement *intelligent* control algorithms.

In the *SoC setup*, the DFA has to be accessed from software. The software is responsible to either transfer the data or results to and from the DFA or to initialize the transfer if direct memory access is used. For this, *device drivers* are required. The implementation of device drivers is a tedious and error prone task. Therefore, reSPACE supports this task by using automatic driver and middleware generation (see Sec. 3). In contrast, the *direct access setup* requires that the DFA has to be accessed directly from other hardware components or from a remote hardware system via low-level communication interfaces. Here, reSPACE hides the technical details and allows to focus on the algorithm development.

2.3 Limitations

A general problem for FPGA-accelerated machine learning operations is the amount of available memory. The amount of data, that can be stored in the available block RAMs of even the current high-end FPGA devices, is in the range of some dozens of MB [16]. Hence, the storage of large amounts of data *inside the FPGA itself* is usually not feasible in practice, and external memory is needed. Another algorithmic solution approach is the usage of incremental or online learning methods.

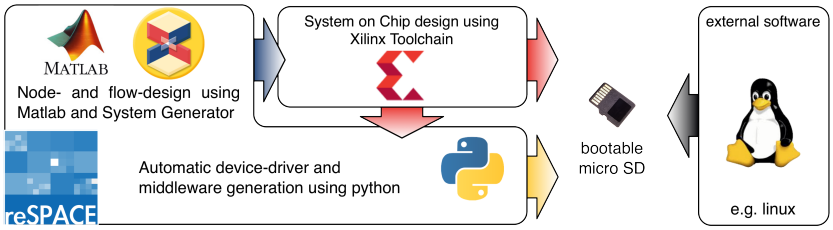

Fig. 1. Workflow for hardware accelerator integration. The processing-flow created by Matlab and System Generator gets integrated into the SoC using the Xilinx toolchain. Output products of the Xilinx toolchain are used for automatic driver and middleware generation. Third party software (e.g., bootloader and kernel) can be added to, e.g., get a bootable medium.

3 Software Infrastructure and Integration

In the in SoC setup, the DFA is used to accelerate a specific software task. Usually, an operating system, like Linux, is running on the host CPU. Hence, to access the DFA from a software application, device drivers are needed that interface with the DFAs and run as kernel modules.

3.1 Automatic Software Generation

In the given context, however, these drivers have a limited duty: the transfer of the data from main memory to the DFA and collection of the results, while the internal *business logic* of the driver is neglectable and can be implemented in the user space. If the systems physical memory-map is known, it is possible, to generate the required drivers *fully automatically*.

The same approach can be used to automatically generate additional interface libraries to other higher-level languages, such as Python. Consequently, the DFA can be directly used from popular machine learning frameworks with a minimum of effort. Currently, we provide automatic generation of code to integrate reSPACE nodes into pySPACE [12] to allow the use of software-centric mechanism like cross validation and performance evaluation. We refer to this process as *automatic driver and middleware generation*. For details about this software generation process, see Fig. 1.

3.2 Functional Verification

Still, the design and implementation of DFAs can be a complex and error-prone task, due to the FPGA properties, like cycle-accurate timing and numerical issues due to fixed-point computations that need to be solved. Accordingly, a high effort is required in order to verify and ensure the functionality of the DFAs. In reSPACE, we use a verification approach that is based on pySPACE. In pySPACE, it is straightforward to generate test data and information regarding

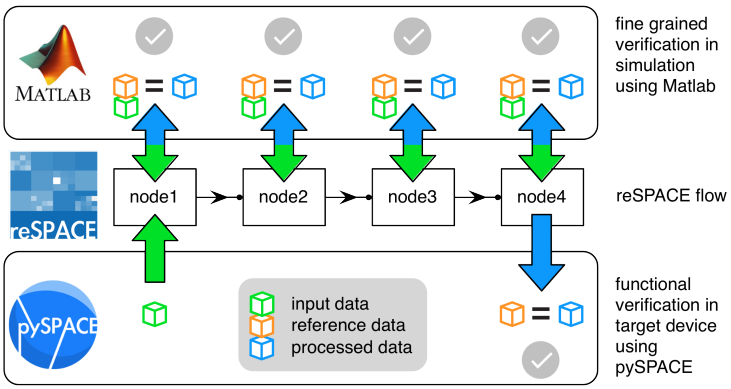

Fig. 2. Dataflow paradigm and verification for reSPACE generated flows. In simulation, the flow can be verified on node level whereas the final implementation inside the FPGA fabric is verified by comparing the final processed data with precomputed reference data inside pySPACE.

the configuration of the specific nodes. The pySPACE flows can be augmented with helper nodes that monitor and store all data persistently that is passed through them.

Simulation Verification Our tools for hardware verification build on this functionality of pySPACE. The intermediate data can be used directly to generate testbenches. These allow the verification of the hardware accelerator in simulation and an in-depth investigation and analysis of any differences between the pySPACE results and intermediate results of the dataflow hardware accelerator. These might occur due to, e.g., fixed-point computations that are usually employed in FPGAs.

Hardware Verification Besides the simulation based verification, reSPACE also supports verification directly on the target system. There, we use a hierarchical approach to verify the availability and functionality of the dataflow hardware accelerator. First, it is tested if the Linux device file is accessible and reachable from the software side. Second, a dedicated test code can be read from the device to check if the dataflow hardware accelerator is available. Finally, the persistent test data is used to test the dataflow hardware accelerator for full functionality. Therefore, the test data is used for stimulation and comparison with the results of the hardware accelerator.

4 Applications

In this section, we outline how the proposed framework can be used for different application areas. We briefly show results for online electroencephalogram (EEG) analysis as a more extensive example.

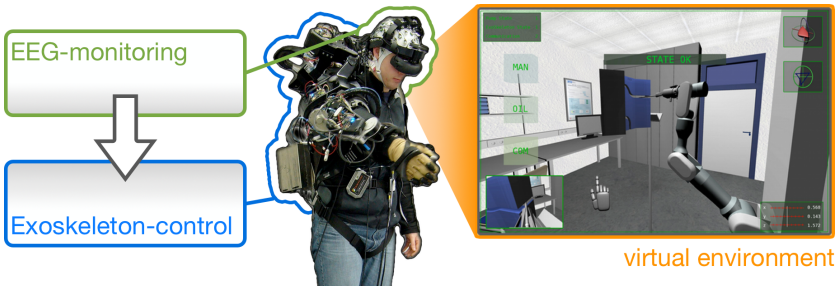

Fig. 3. Biosignal-augmented exoskeleton. The operator controls a robotic arm in a virtual environment (right). The task is to move the end-effector of a robotic arm like in an *hot wire game* through a labyrinth without touching it. For control, the operator wears an exoskeleton that maps the position of the operator's hand to the end-effector. The operator's EEG is continuously analyzed to detect upcoming movements that can be used to adapt the control algorithms of the exoskeleton.

4.1 Biomedical Signal Processing for Teleoperation and Rehabilitation Robotics

Performing teleoperation of robotic systems using current input devices like joysticks is usually a demanding task. To simplify this, the exoskeletons can be used as more intuitive command interfaces. The details of the investigatio have been described in [17, 18] and are depicted in Fig. 3. To enhance the movability of the exoskeleton, the joint control algorithms can be enhanced by integrating predictions of upcoming movements based on the detection of movement-related cortical potentials [19]. For the predictions, the EEG of the operator is continuously analyzed by a movement prediction system, which performs a prediction each 50 ms. EEG data is high dimensional (64 channels sampled at 5kHz), require a range of different signal processing and machine learning operations to detect the to upcoming movements, and fast computations, since the processing has to be finished *before* a movement is executed. Since the operator should be able to move freely, the exoskeleton has to be be independent from stationary hardware. In future, the exoskeleton will also be used as a rehabilitation systems [20]. Therefore, all computations should be performed in devices that are *embedded* in the exoskeleton, were reSPACE can be used to map the operations to FPGAs to provide the necessary computational power. For the detections, different signal processing operations have to be applied to the data *online*. We use the following procedure: DC offset removal, anti-alias filtering and downsampling to 20 Hz, spatial filtering, feature vector extraction, standardization, and classification using a passive-aggressive perceptron, whose parameter c was optimized using a grid search. We realized this twice using pySPACE and reSPACE and compare the classification and computation performance for both approaches.

The obtained results are shown in Table 1. It can be observed that there exist no major differences regarding classification performance. However, by using the DFA an $\approx 7\times$ speedup is achieved.

Table 1. Application oriented metrics for online and pseudo-online sessions in terms of Balanced Accuracy (BA) and computation time for 10 recording sessions on 3 subjects. The BA is defined as the mean of true positive and negative rates. The two different devices were a mobile processor (MP, ARM Cortex A9, 666 Mhz) and a combined mobile processor with DFA (running at 100 Mhz). All computations on the MP were performed as double precision floating point operations, all computations in the DFA used fixed point arithmetic. The reported computation times are the amount of time that is required to process 1s of EEG data for the different processing platforms.

	Performance Mobile CPU	Performance Mobile CPU + DFA
BA (%)	76.012 ± 4.106	75.717 ± 4.018
Computation time (ms)	1199.943 ± 7.170	174.403 ± 5.872

4.2 Further Application Areas

There are various other application areas in machine learning, robotics and autonomous systems that would gain a substantial benefit from FPGA-based accelerators. Obvious examples are various image processing techniques, such as SURF generation [21] and traffic sign detection [22] or enhancing the control of robots [23].

5 Conclusion and Future Work

In this paper, we discussed the necessity, requirements and state of the art of FPGA-based machine learning accelerators that should be employed in mobile and robotic systems. As a solution to the current problems, we proposed our framework reSPACE that supports the implementation of such accelerators. We described the techniques that we use in reSPACE to simplify the design process in order to allow algorithm developers to use FPGAs and to verify the resulting hardware implementations in simulation or in the final system. In future, we will enhance the framework to improve the usability further. First, we want to transfer all components to pure VHDL implementations to achieve a higher degree of vendor and third party independence, and to provide the framework as open source to allow it to be used easily by machine learning and robotics researchers.

References

1. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone. In: Proceedings of the 2010 USENIX conference. (2010) 21–21
2. Zhang, Z.: Microsoft kinect sensor and its effect. MultiMedia, IEEE **19**(2) (2012) 4–10
3. Steinkrau, D., Simard, P.Y., Buck, I.: Using gpus for machine learning algorithms. In: Proceedings of the Eighth International Conference on Document Analysis and Recognition, IEEE Computer Society (2005) 1115–1119

4. Xilinx Corporation: UG479: 7 Series DSP48E1 Slice User Guide (2014)
5. Xilinx Corporation: DS444: IP Processor Block RAM, Product Specification (2011)
6. Omondi, A.R., Rajapakse, J.C.: FPGA implementations of neural networks. Springer (2006)
7. Anguita, D., Boni, A., Ridella, S.: A digital architecture for support vector machines: theory, algorithm, and fpga implementation. Neural Networks, IEEE Transactions on **14**(5) (2003) 993–1009
8. Graf, H.P., Cadambi, S., Durdanovic, I., Jakkula, V., Sankaradass, M., Cosatto, E., Chakradhar, S.T.: A massively parallel digital learning processor. In: NIPS. (2008) 529–536
9. Domingos, P.: A few useful things to know about machine learning. Communications of the ACM **55**(10) (2012) 78–87
10. Zito, T., Wilbert, N., Wiskott, L., Berkes, P.: Modular toolkit for Data Processing (MDP): a Python data processing framework. Frontiers in Neuroinformatics **2**(8) (2008)
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12** (2011) 2825–2830
12. Krell, M.M., Straube, S., Seeland, A., Wöhrle, H., Teiwes, J., Metzen, J.H., Kirchner, E.A., Kirchner, F.: pySPACE - a signal processing and classification environment in Python. Frontiers in Neuroinformatics **7**(40) (2013)
13. Flagg, M.: Dataflow principles applied to real-time multiprocessing. In: COMPCON Spring'89. Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers., IEEE (1989) 84–89
14. Xilinx Corporation: UG640: System Generator for DSP User Guide (2012)
15. ARM: AMBA AXI and ACE Protocol Specification (2013)
16. Altera Corporation: Stratix V Device Overview (2014)
17. Folgheraiter, M., Kirchner, E.A., Seeland, A., Kim, S.K., Jordan, M., Woehrle, H., Bongardt, B., Schmidt, S., Albiez, J., Kirchner, F.: A multimodal brain-arm interface for operation of complex robotic systems and upper limb motor recovery. In: Proc. of the 4th International Conference on Biomedical Electronics and Devices (BIODEVICES-11), Rome (2011) 150–162
18. Folgheraiter, M., Jordan, M., Straube, S., Seeland, A., Kim, S.K., Kirchner, E.A.: Measuring the improvement of the interaction comfort of a wearable exoskeleton. International Journal of Social Robotics **4**(3) (2012) 285–302
19. Seeland, A., Woehrle, H., Straube, S., Kirchner, E.A.: Online movement prediction in a robotic application scenario. In: 6th International IEEE EMBS Conference on Neural Engineering (NER), San Diego, California (2013) 41–44
20. Kirchner, E.A., Albiez, J., Seeland, A., Jordan, M., Kirchner, F.: Towards assistive robotics for home rehabilitation. In Chimeno, M.F., Solé-Casals, J., Fred, A., Gamboa, H., eds.: Proceedings of the 6th International Conference on Biomedical Electronics and Devices (BIODEVICES-13), Barcelona, ScitePress (2013) 168–177
21. Bay, H., Tuytelaars, T., Van Gool, L.: Surf: Speeded up robust features. In: Computer Vision–ECCV 2006. Springer (2006) 404–417
22. Zaklouta, F., Stanciulescu, B.: Real-time traffic sign recognition in three stages. Robotics and Autonomous Systems **62**(1) (2014) 16–24
23. Langosz, M., von Szadkowski, K., Kirchner, F.: Introducing particle swarm optimization into a genetic algorithm to evolve robot controllers. In: Proceedings of the 16th Annual Conference on Genetic and Evolutionary Computation. GECCO '14, ACM (2014)