Transfer Project T1: Flexible Industrial Analytics on Reconfigurable Systems-On-Chip

Alexander Boschmann[2], Lennart Clausing[1], Felix Jentzsch[1], Hassan Ghasemzadeh Mohammadi[1], Marco Platzner[1]

1 Department of Computer Science, Paderborn University, Paderborn, Germany
2 Weidmüller Interface GmbH & Co. KG., Germany

1 Introduction

Industrial analytics refers to the current trend in automation technology to collect and analyze a variety of measured values from machines and from production processes in order to generate added value for future operations. Industrial analytics is a business field with great economic potential, and Weidmüller wants to position itself as a leading provider of industrial analytics solutions within the framework of the mission statement *Industry 4.0*[24]. Examples for industrial analytics include the detection of significant deviations from the target behavior of a machine [MKPN13], the detection of inefficiencies [MPK15], the creation of fault forecasts and the diagnosis of fault causes. The added value achieved is the avoidance of machine breakdowns, the minimization of downtime, or in general, the increase of plant productivity and production output [PKG+16].

In *embedded analytics*, i.e., the implementation of analysis functions directly in the automation devices within a production plant, Weidmüller relies on reconfigurable System-on-Chip (rSoC). The challenges in using rSoC for industrial analytics are on the one hand the required *flexibility in system design* and, on the other hand, the increasing *heterogeneity of rSoC platforms*. Flexibility is needed since the functions of industrial analytics have to be selected, configured and assembled on an application-specific basis, implemented as a hardware/software co-design and deployed on an rSoC. Flexibility can further be exploited during runtime to use the resources efficiently under varying load situations. The technological evolution of rSoC platforms is toward more heterogeneous architectures: for example, recent rSoCs combine multiple processor types with reconfigurable hardware, embedded graphics processors, and application-specific blocks.

The combination of increasing *dynamics* of tasks and *heterogeneity* of the underlying architectures is also the guiding theme of basic scientific research in subproject C2 of SBF 901. There, novel architectures and programming models for heterogeneous computing nodes are investigated and developed. By transferring these basic scientific results to the industrial analytics application domain, this transfer project aims to achieve the following goals:

Alexander.Boschmann@weidmueller.com (Alexander Boschmann), lennart.clausing@upb.de (Lennart Clausing), felix.jentzsch@upb.de (Felix Jentzsch), ghasemzadeh@gmail.com (Hassan Ghasemzadeh Mohammadi), platzner@ubp.de (Marco Platzner)

[24]`https://www.weidmueller.com/int/solutions/industrial_analytics/index.jsp`

1. Characterization of essential functions of industrial analytics and design of hardware/software partitionings suitable for an rSoC implementation.
2. Development of architectures and programming environments to enable transmodal migration on rSoC.
3. Demonstration of rSoC technology for industrial analytics use cases.

For rSoC architectures and programming environments we draw on preliminary work, the ReconOS [AHK+14] operating system for reconfigurable computers. ReconOS allows for multithreaded programming across the software/hardware boundary by turning accelerator functions into so-called hardware threads and semantically integrating them as threads into a guest operating system environment. Compared to related approaches such as BOPRH [KB08], Hthreads [ASA+08], FUSE [IS11], SPREAD [WZW+13], and LEAP [FYAE14], ReconOS is more flexible and more rapidly portable to new guest operating systems and FPGA technologies. In particular, ReconOS has demonstrated its usefulness in three scenarios: First, ReconOS supports a step-by-step development flow starting from a software application prototype on desktop under Linux. Only when the prototype is functionally correct, is the application ported to the embedded rSoC, which is typically a low effort since the embedded CPU cores also run Linux. As a last step, threads that are amenable to hardware acceleration are gradually moved from software to hardware. Second, ReconOS facilitates design space exploration since different hardware/software partitionings can easily be generated by simply modifying system configuration data and no changes are needed to unaffected threads or the operating system. This feature has been used, for example, to develop a video object tracking system [HLP13]. Finally, ReconOS even allows for the construction of adaptive or self-adaptive systems, where a hardware/software application monitors its own performance and changes the architecture, for example the number of used CPU cores and hardware threads or the hardware/software partitioning, in reaction to a varying workload [AHL+14].

2 Main Contributions

In the course of the transfer project, we have achieved the following results:

- We have developed ReconOS^{64} as a new version of the ReconOS architecture and operating system layer for the modern 64-bit rSoC technology used in the project, i.e., the Xilinx UltraScale+ MPSoC platform FPGAs [CP22; Cla21].
- We have created a build tool flow for ReconOS^{64} that includes a high-level synthesis (HLS) tool flow and thus allows for creating hardware threads not only in hardware description languages such as VHDL and Verilog, but also in C/C++.
- We have implemented several industrial analytics functions as software/hardware co-designs on the rSoC platform, including k-NN [Ria17], decison trees/random forests, SVM [BTW+17], and neural network models [Nga22].
- We have worked on several industrial analytics case studies for condition monitoring and anomaly detection, respectively, targeting wind turbines, molding machines and welding machines [Kau22].

In the following, we select two of these topics for elaboration, the ReconOS^{64} development and the DeepWind case study, a condition monitoring system for wind turbines.

2.1 The ReconOS^{64} Operating System for 64-bit Platform FPGAs

ReconOS^{64} bases on previous ReconOS [AHK$^+$14; LP09] implementations but targets modern platform FPGAs with 64-bit processors. The step towards 64-bit support and the use of modern platform FPGAs is important, since many applications, in particular industrial analytics functions, require the increased computing capabilities and resources provided by modern rSoC. ReconOS and its 64-bit version are freely available in open source[25].

Figure 69 shows the architecture of ReconOS^{64} on the Xilinx UltraScale+ MPSoC. The processing system (PS) comprises a 64-bit quad-core CPU and runs the 64-bit Xilinx PetaLinux as the host operating system. ReconOS^{64} extends the host operating system by the ReconOS driver in kernel space and several libraries in user space for, e.g., thread synchronization, communication, management and bitstream loading. The programmable logic part of the platform FPGA is structured into so-called reconfigurable slots that constitute rectangular areas of the programmable logic fabric. Reconfigurable slots accommodate hardware threads, which are ReconOS' abstractions of accelerated functions. A main feature of ReconOS is that hardware threads access the operating system using the same services as software threads running on the CPU, thus enabling the multithreaded programming abstraction across the hardware/software boundary. This is made possible by delegate threads, light-weight software threads that conduct operating system calls on behalf of their corresponding hardware threads.

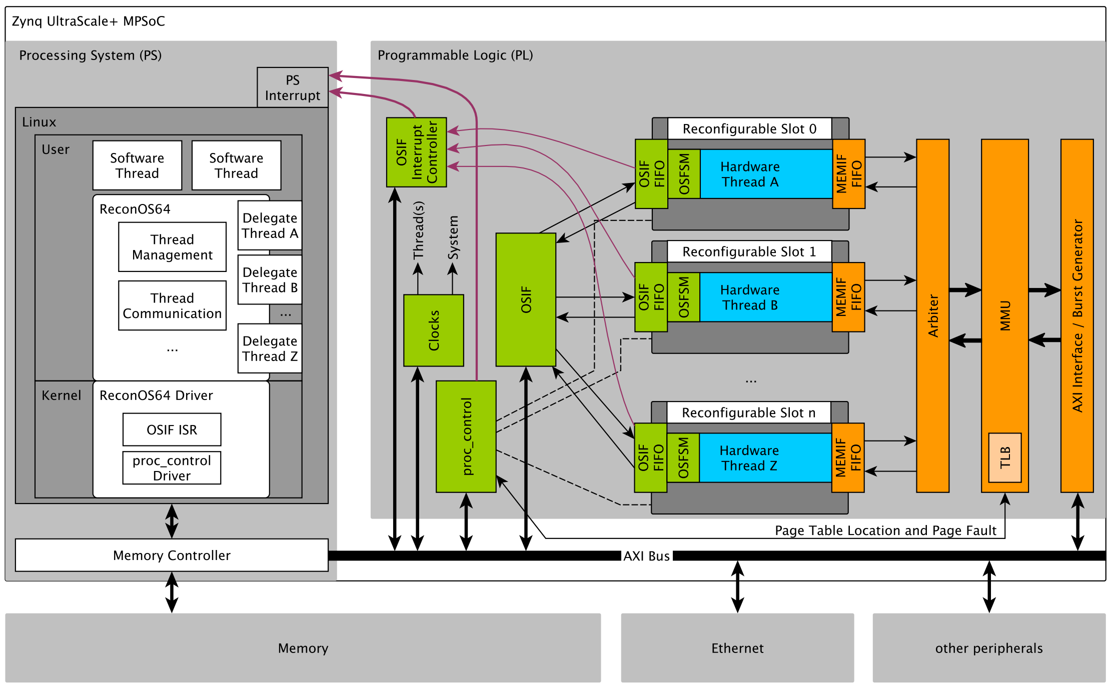

Figure 69: *ReconOS^{64} Architecture on the Xilinx UltraScale+ MPSoC (taken from [CP22]).*

The intellectual property (IP) cores of ReconOS^{64} responsible for connecting hardware threads with the host operating system are shown in green color in Figure 69. Each

[25] `www.reconos.de`

hardware thread comprises the actual user logic and an operating system finite state machine (OSFSM) that sequentializes the thread's operating system interactions and handles synchronization between the user logic and the software. Further, each hardware thread is connected to an operating system interface (OSIF) FIFO that buffers the operating system calls, i.e., their commands with parameters and return values. The FIFO also serves to separate the clock regions of the ReconOS^{64} design from the hardware threads to allow them to run at different frequencies. The OSIF FIFOs connect to a central OSIF IP core that collects the commands and parameters for all service requests and, in addition, to a dedicated OSIF interrupt controller that raises a CPU interrupt whenever a hardware thread wants to execute an operating system call. On the software side, the raised interrupt will activate the OSIF interrupt service routine (ISR), which in turn sets the delegate thread corresponding to the hardware thread that is ready to run. The delegate thread then accesses the OSIF IP core, retrieves the command and parameters and actually performs the operating system call. In case there are return values, they are written back to the hardware threads.

The proc_control IP core together with the proc_control kernel driver are also involved in operating system communication as they propagate reset signals towards the hardware threads. In ReconOS^{64} the native data type is 64 bit. Hence, all IP cores involved in operating system communication support command, parameter, and return data structures in multiples of 64-bit. This is particularly important since many operating system calls, e.g., message box reads and writes, are typically used to communicate 64-bit pointers between software and hardware threads. A consequence of the 64-bit orientation is that data of smaller width has to be either padded to the next multiple of eight bytes or, if several small-sized data are to be written or read, concatenated to blocks of eight byte.

The IP cores of ReconOS^{64} responsible for supporting memory accesses of the hardware threads are displayed in orange color in Figure 69. While address pointers in ReconOS^{64} are 64-bit wide, the Linux configuration we use on the ARMv8 CPU architecture uses a virtual address space of 512 GB that is mapped to a physical address space of 256 TB. Hence, the systems' memory management unit (MMU) considers only the lower 39 bit of virtual addresses and deals with 48-bit physical addresses. The page size in our configuration amounts to 4 KB. Hardware threads use virtual addresses and access memory via their memory interfaces (MEMIF). ReconOS^{64} employs three IP cores for establishing memory access. The Arbiter resolves simultaneous accesses from different hardware threads, the MMU performs the translation to physical addresses, and the AXI Interface / Burst Generator interfaces to the AXI bus and ensures burst transfers. Initially, the content of the ARM CPU's Translation Table Base Register (TTBR) is transferred to the MMU via the kernel driver and the proc_control IP core to ensure that the MMU has the physical address of the ReconOS process' first-level page table. Then, the MMU performs the page table walk which results in at most three memory accesses. To speed up memory access for hardware threads, the MMU includes a translation look-aside buffer (TLB) that caches recent translations between the 27-bit virtual page numbers and the 36 bit physical page numbers. The size of the TLB is configurable. The proc_control component supports the handling of page faults during address translation. Therefore, proc_control needs to be able to raise an interrupt with the CPU.

Dynamic thread management is supported through partial reconfiguration in ReconOS^{64} . Generally, a hardware thread is assigned to a reconfigurable slot, which is a rectangular area of logic resources on the FPGA residing in the same clock region. A new feature of ReconOS^{64} are *reconfigurable slot groups*, which specify sets of reconfigurable slots of the same size. Each hardware thread is assigned to one or more such reconfigurable slot groups, and multiple hardware threads can be assigned to the same reconfigurable slot group. The introduction of reconfigurable slot groups makes the runtime mapping between hardware threads and reconfigurable slots more flexible.

ReconOS^{64} allows for the hardware threads to run at individual clock rates, in particular different ones from the clock of the static ReconOS part. These individual clock signals are fed from the ReconOS^{64} clocking IP core that utilizes a Mixed-mode Clock Manager (MMCM) tile with static multiplier and variable dividers for each clock output. Reconfigurable slot groups can be assigned to individual clocks as long as the clock tile resources are not exceeded. Using a function from the ReconOS^{64} thread management library, both software and hardware threads can set the clock frequencies for hardware threads by modifying the clock dividers in the corresponding ReconOS^{64} clocking IP core.

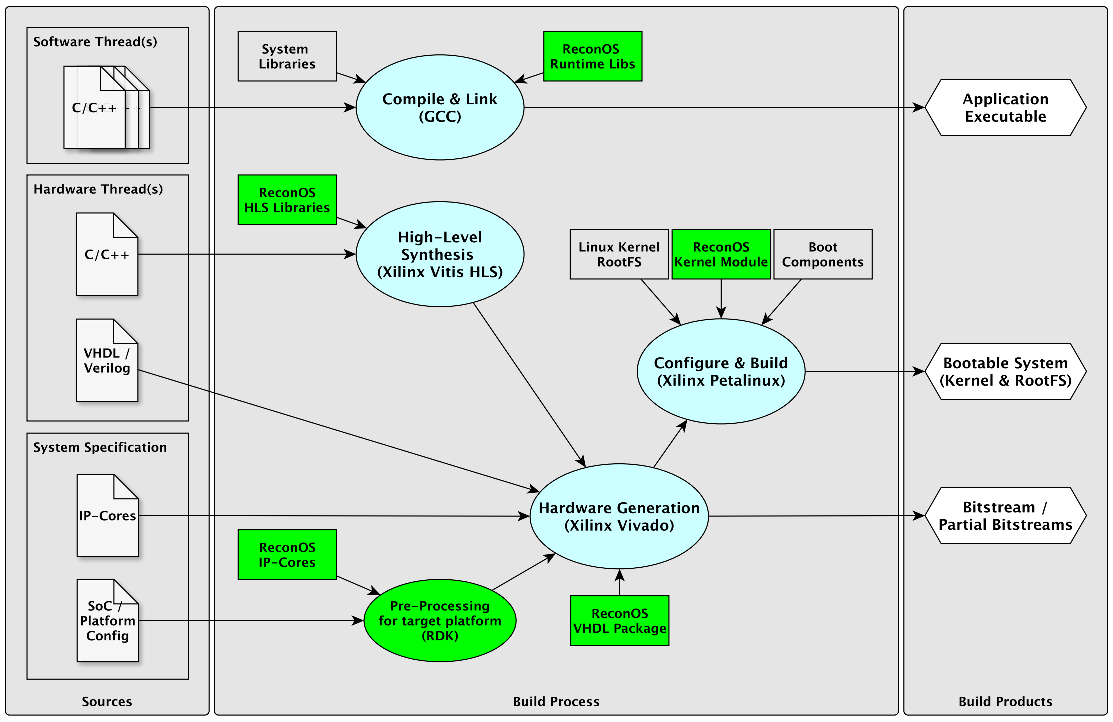

Figure 70: *ReconOS^{64} build tool flow (adapted from [AHK^{+}14]).*

The ReconOS^{64} build tool flow takes as inputs the sources for the application's software threads in C/C++ and the sources for the hardware threads in either VHDL/Verilog or C/C++ for use with high-level synthesis (HLS). For both, a predefined set of ReconOS^{64}-specific library functions is provided. A further input is the system specification comprising a set of ReconOS^{64} IP cores and the configuration file. The configuration file includes definitions for the target platform, the reconfigurable slots, and reconfigurable slot groups and assigns the hardware threads to reconfigurable slot groups. Further, all operating system service objects, such as message boxes, mutexes, and semaphores, are listed in the configuration file. The build process relies on a Python-based templating system. The application software is cross-compiled with the `aarch64`-gcc compiler, which results in the application executable. On the hardware side of the build tool flow, the ReconOS Development Kit (RDK) processes the configuration file and generates IP sources from architecture- and board-specific templates. The flow then generates a Xilinx Vivado project incorporating the user-provided hardware threads, either directly from the VHDL/Verilog code or the result from HLS, with all necessary components and connections. The hardware build process results in the static bitstream for the ReconOS^{64} system and a set of partial bitstreams for the hardware threads. Information from the hardware build process (e.g., used address ranges for IP cores) together with a generic Xilinx PetaLinux template project, the ReconOS^{64} kernel module and device tree, and boot components are used to configure and generate a bootable system including the operating system kernel and the root file system.

2.2 DeepWind: An Accurate Wind Turbine Condition Monitoring Framework via Deep Learning on Embedded Platforms

The generation of electricity using wind turbines is rapidly growing and becoming more important since it is considered as an affordable and clean substitute for fossil fuel-based electricity production. Wind turbines are used in a large variety of environments, both onshore and offshore, and they are exposed to harsh working conditions, such as unbalanced wind load, wind turbulence, and large temperature variations [QL15]. To service running wind turbines unceasingly and safely, and particularly reduce the maintenance costs, adequate online *condition monitoring systems* (CMSs) are required [Wei]. CMSs identify the type and the location of faults and, more importantly, diagnose the transformation of a fault into an error and possibly into a failure.

During wind turbine operation, a CMS constantly takes measurements that determine the condition of the critical components, e.g., the rotor blades. The measurements provide indications for problems such as blade damages after lightning strikes, heavy vibrations of the blades, or the ice accretion on a rotor blade. Ice accretion may lead to dangerous ice throw, which is a major risk for the surroundings of wind turbines. Therefore, more and more local authorities insist on blade measuring ice detection systems. By processing the information of a CMS, a diagnosis, e.g., *inspection*, *necessary repair*, or *necessity of turbine shutdown*, is reached and an adequate maintenance plan is formulated. Consequently, the CMS facilitates low-cost maintenance before a critical failure happens while diminishing the downtime of the wind turbine, also increasing its dependability and lifetime. Defects can cause abnormal vibrations of the blades, which can be sensed by accelerometers installed on the blades. Earlier work applied frequency spectrum analyses [CG05; WX06] to detect such defects. However, such analyses require manual feature engineering and extensive trial-and-error to identify patterns in the vibrations that correctly match to faulty cases.

In DeepWind, we propose a novel framework to build an end-to-end condition monitoring and fault detection system for rotor blades. The framework starts with a preprocessing step to reduce the complexity of the raw sensor data. Then, inspired by the success of deep learning in time series analysis, we train a multi-channel convolutional neural network (MC-CNN) that can automatically extract a set of discriminative features from the sensor data. Finally, the trained MC-CNN is automatically mapped to an embedded

FPGA platform, where a combination of software and hardware identifies fault occurrences within the data streamed from accelerometer sensors.

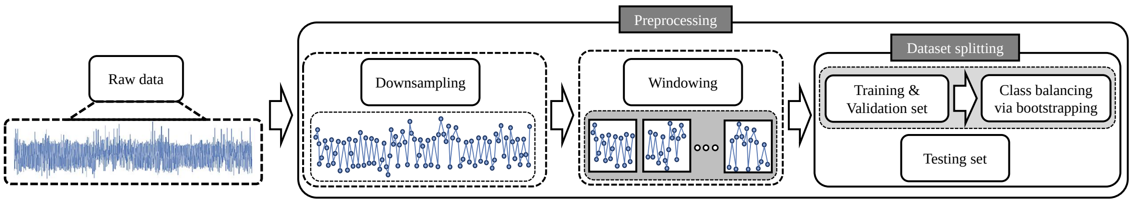

Figure 71: *Main steps of data preprocessing: downsampling, windowing, data set splitting, and training set bootstrapping (taken from [GAR+20]).*

Figure 71 illustrates the main steps of the sensor data preprocessing. The main goal of the preprocessing step is to convert the raw sensor data into a cleanly formatted data set that can be used later by the MC-CNN for fault detection purposes. The complexity of sensor data is reduced by applying downsampling, in which the number of sample points in the input data is reduced. We utilize the *Mode-Median-Bucket* algorithm [Ste13], in which every window is divided into several subwindows in such a way that each subwindow contains the same number of samples. The algorithm considers important features from each bucket with mode, median, global peaks, or global trough values and filters out the other samples in each subwindow. In the next step, we utilize the windowing technique to divide each input data frame into a number of smaller segments called windows. Each window simply adopts the label of its data frame. Finally, we form our training and testing data sets from the obtained windows. We modify the training set by bootstrapping with replacement to ensure that the number of samples from both faulty and non-faulty classes are comparable. This is an important step to be able to train a high-quality classifier that provides high accuracy and recall on the testing set [Man22].

As the target feature extractor and classifier, we exploit a multi-channel CNN, in which the training of each individual univariate data, e.g., raw data from each sensor, is performed independently [Kau22]. Indeed, we can draw a lot of inference from the local properties of each sensor without losing the generality of our classifier, by decoupling the data of different sensors. The architecture of the MC-CNN model we have used in this work is shown in Figure 72. After preprocessing the sensor data we apply a Fast Fourier Transform (FFT) on each input window to extract its Spectrum Frequency (SF). The obtained SFs are then fed into the MC-CNN.

The first part of the MC-CNN performs feature extraction and contains two 1-D convolution layers as well as two max-pooling layers. For each sensor, the so-called channel, we utilize 50 and 40 feature maps in the first and second convolution layers with the size of 8 and 4, respectively. As we have two sensors per blade, we exploit six 1-D convolution channels. The outcomes of each convolution layer are downsampled by a max-pooling layer to control the growth in the size of the extracted features. Finally, the obtained features are fed into a fully connected layer with 400 neurons. This layer is followed by a softmax layer that generates the conditional probability of faulty and non-faulty classes. Note that the training of the MC-CNN is performed offline, and then the trained model is quantized and mapped on the hardware for the inference phase.

On the hardware side of our framework, called TFPGA, we utilize an rSoC as the target platform. We exploit hardware/software codesign to both efficiently distribute the

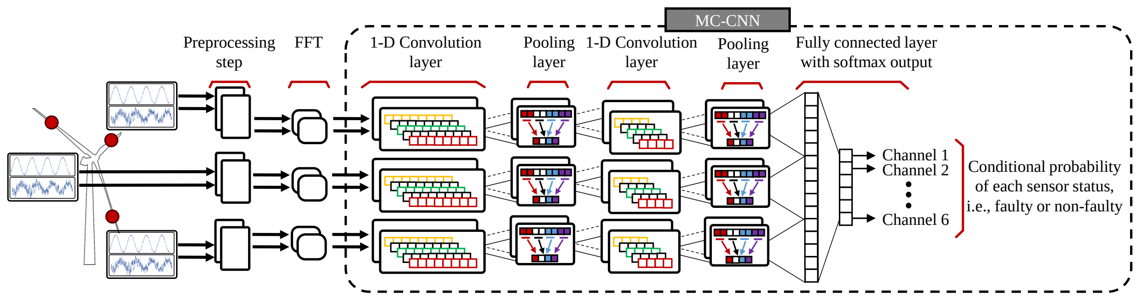

Figure 72: *The architecture of multi-channel CNN for fault detection. The convolution layers have 50 and 40 filters with kernel sizes of 8 and 4, respectively. For each window, obtained from the preprocessing step, the FFT spectrum is computed and the outcome is fed to the MC-CNN (taken from [GAR+20]).*

framework tasks on various rSoC resources and benefit from the customizability and parallelism offered by FPGAs. For inference, the tasks of the preprocessing step, e.g., downsampling and windowing, as well as the FFT computation for each obtained window are assigned to the CPUs of the rSoC. Note that the bootstrapping task is just performed on the training data set and is omitted from consideration in the inference phase. Next, the trained MC-CNN network is analyzed, and a C++ implementation of this model is created. This code is then given to Xilinx SDSoC to create a bitstream needed for the target FPGA. To improve the execution time of the software model and reduce its size, the framework exploits a custom precision scaling feature that enables a designer to utilize the underlying hardware more efficiently by tuning the parameters of the given network.

We have used a real-word data set provided by Weidmüller Monitoring Systems GmbH to evaluate the approach. The data set comprises time series data measured with a sampling rate of 1 kHz from the edge-wise and flap-wise sensors for each of three rotor blades. For every half hour (i.e., 1.8 million sample points), the data is labeled with the sensor status in that interval as *faulty* or *non-faulty*. After preprocessing, we have obtained samples with a window size of 1 second along with corresponding labels. We have used 80% of the data for training and validation and the remaining 20% for testing. All the models have been implemented with the Keras library [Ker].

Figure 73 shows the classification result of our DeepWind framework. The figure plots the achieved F1-score versus the six channels, i.e., two per blade, where each channel represents a sensor blade. The F1-score is the harmonic mean of precision and recall and reaches its best value at 1, which translates to perfect precision and recall. Our proposed MC-CNN based fault detection scheme provides an average of 0.94 for the F1-score. As a baseline technique, we have experimented with a Support Vector Machine (SVM) used previously by the application partner, which results in an F1-score of 0.64 on average. Importantly, for all of the six channels MC-CNN provides better classification results in comparison with SVM, making our MC-CNN approach a successful technique to capture the most discriminative features for the sensor blade fault detection problem.

Figure 74 represents the accuracy for various quantization settings and for the reference software implementation, which utilizes double-precision floating point. The results show an accuracy penalty of 6% for a 16-bit quantization, which we deem acceptable without model retraining. When only the weights are quantized further to 8 bit, we even observe a slight increase in accuracy to 87%. We attribute this to the inherent regularization characteristic of the quantization, since we know that the original model benefits from dedicated regularization, namely through dropout. We have also measured the resource usage for different quantization settings. The initial results revealed that going from 16 to 8 bit leads to a slight saving in lookup tables of 12%. Quantization of weights to 8 bit achieves a 34% decrease in embedded memory (BRAM) usage. These results show that weight quantization is effective for reducing the memory footprint in an MC-CNN hardware accelerator.

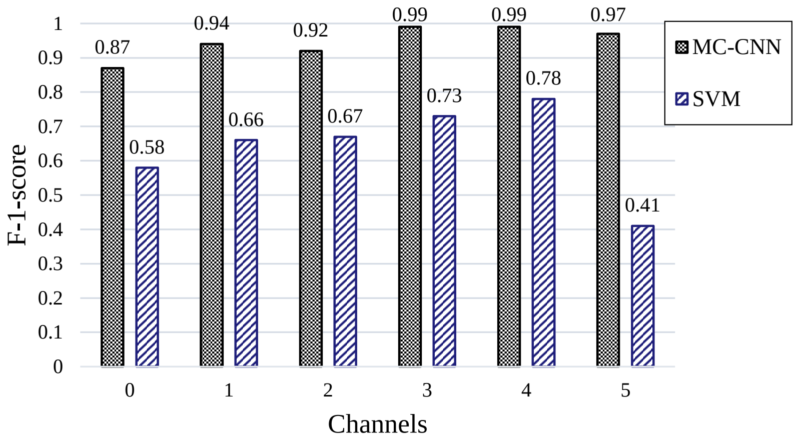

Figure 73: *F1-score of MC-CNN and SVM methods for six channels (taken from [GAR+20]).*

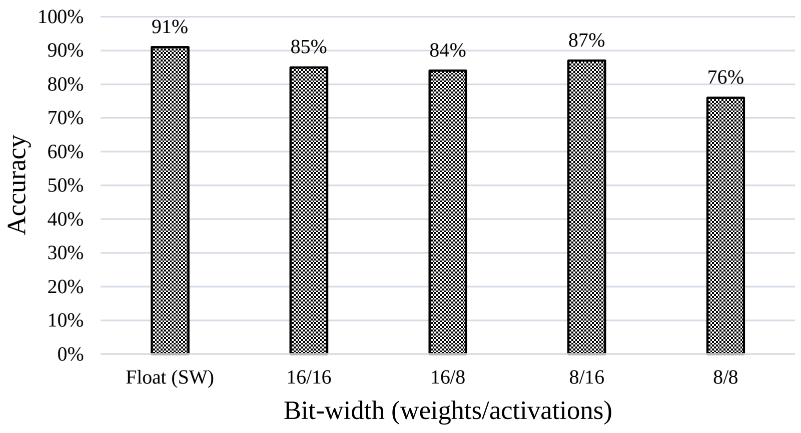

Figure 74: *Accuracy vs. bit width of the MC-CNN on a Xilinx UltraScale+ MPSoC (taken from [GAR+20]).*

3 Impact and Outlook

This transfer project allowed us to further develop and apply reconfigurable system-on-chip technology to the concrete application domain of industrial analytics functions, together with the application partner Weidmüller. Overall, the project was successful since the newly developed 64-bit ReconOS^{64} architecture provides CPU cores with sufficient compute power and hardware acceleration for industrial analytics functions. The corresponding build tool flow has shown to support the specification of runtime-reconfigurable functions in a rather simple way. We have implemented a number of typical functions of industrial analytics as software/hardware co-designs with ReconOS^{64} and demonstrated that different trade-offs between performance and resource consumption can be explored. Our developments are open source and can thus be used and leveraged by others. We have also worked on several use cases, where condition monitoring for wind turbines is so far the most successful one. For this use case, we could propose an industrial analytics function that greatly improves the existing solution in terms of quality. The mapping to an embedded rSoC is also of great interest, since then the condition monitoring system can be placed near the sensors in the rotor blades and running such functions on servers in wind turbines can be avoided.

One aspect planned for this transfer project could not yet be realized in a use case, the transmodal migration of industrial analytics functions. While this feature has been demonstrated in the lab, for the concrete use cases it was more important to spend time for developing industrial analytics functions that excel in functional quality. One particular challenge for developing good solutions is that often only small data sets or data with very imbalanced classes are available.

Ongoing and future work includes the development of more solutions for condition monitoring and predictive maintenance, in particular for welding machines [Kum23], and the further optimization of ReconOS^{64} .

For mapping DNN architectures to rSoC, in this transfer project we have first used our TFPGA framework (cf. Section 2.2) and, later, we have developed a framework that focuses on TF Lite[26] with its backend delegate modules. We have developed an FPGA delegate for TF Lite that facilitates the necessary hardware/software co-design using the ReconOS^{64} architecture and operating system (cf. Section 2.1). The partial reconfiguration support of ReconOS^{64} enables the instantiation of model-tailored accelerator architectures. Mapping DNNs to rSoC technology remains an area of active research. Recently, we have switched to the open source FINN framework [BPF$^+$18] that maps DNNs as streaming dataflow architectures to FPGAs and features flexible quantization as well as quantization-aware DNN training.

Bibliography

[AHK$^+$14] Agne, A.; Happe, M.; Keller, A.; Lubbers, E.; Plattner, B.; Platzner, M.; Plessl, C.: ReconOS: An Operating System Approach for Reconfigurable Computing. In: *IEEE Micro* 34 (Jan. 2014), no. 1, pp. 60–71

[AHL$^+$14] Agne, A.; Happe, M.; Lösch, A.; Plessl, C.; Platzner, M.: Self-awareness as a Model for Designing and Operating Heterogeneous Multicores. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 7 (2014), no. 213

[ASA$^+$08] Andrews, D.; Sass, R.; Anderson, E.; Agron, J.; Peck, W.; Stevens, J.; Baijot, F.; Komp, E.: Achieving Programming Model Abstractions for Reconfigurable Computing. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 16 (2008), no. 1, pp. 34–44

[BPF$^+$18] Blott, M.; Preusser, T. B.; Fraser, N. J.; Gambardella, G.; O'brien, K.; Umuroglu, Y.; Leeser, M.; Vissers, K.: FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. In: 11 (2018), no. 3.

[26]`https://www.tensorflow.org/lite`

[BTW+17] Boschmann, A.; Thombansen, G.; Witschen, L.; Wiens, A.; Platzner, M.: A Zynq-based dynamically reconfigurable high density myoelectric prosthesis controller. In: *In Proceedings of Design, Automation and Test in Europe (DATE)*. IEEE, 2017

[CG05] Caselitz, P.; Giebhardt, J.: Rotor condition monitoring for improved operational safety of offshore wind energy converters. In: *J. Sol. Energy Eng.* 127 (2005), no. 2, pp. 253–261

[Cla21] Clausing, L.: ReconOS64: High-Performance Embedded Computing for Industrial Analytics on a Reconfigurable System-on-Chip. In: *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*. ACM, 2021

[CP22] Clausing, L.; Platzner, M.: ReconOS64: A Hardware Operating System for Modern Platform FPGAs with 64-Bit Support. In: *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2022, pp. 120–127

[FYAE14] Fleming, K.; Yang, H.-J.; Adler, M.; Emer, J.: The LEAP FPGA operating system. In: *International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014

[GAR+20] Ghasemzadeh Mohammadi, H.; Arshad, R.; Rautmare, S.; Manjunatha, S.; Kuschel, M.; Jentzsch, F. P.; Platzner, M.; Boschmann, A.; Schollbach, D.: DeepWind: An Accurate Wind Turbine Condition Monitoring Framework via Deep Learning on Embedded Platforms. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2020

[HLP13] Happe, M.; Lübbers, E.; Platzner, M.: A Self-adaptive Heterogeneous Multi-core Architecture for Embedded Real-time Video Object Tracking. In: *International Journal of Real-time Image Processing* 8 (2013), no. 1, pp. 95–110

[IS11] Ismail, A.; Shannon, L.: FUSE: Front-End User Framework for O/S Abstraction of Hardware Accelerators. In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2011

[Kau22] Kaur, P.: Analysis of Time-Series Classification in Conditional Monitoring Systems. MA thesis. Paderborn University, 2022

[KB08] Kwok-Hay So, H.; Brodersen, R.: Runtime Filesystem Support for Reconfigurable FPGA Hardware Processes in BORPH. In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2008

[Ker] Keras: *The Python Deep Learning API*. `https://keras.io`

[Kum23] Kumar, N. Y. M.: Data Analytics for Predictive Maintenance of Time Series Data. MA thesis. Paderborn University, 2023

[LP09] Lübbers, E.; Platzner, M.: ReconOS: Multithreaded Programming for Reconfigurable Computers. In: *ACM Trans. Embed. Comput. Syst.* 9 (Oct. 2009), no. 1

[Man22] Manjunatha, S.: Dealing with Pre-Processing and Feature Extraction of Time-Series Data in Predictive Maintenance. MA thesis. Paderborn University, 2022

[MKPN13] Maier, A.; Köster, M.; Paiz Gatica, C.; Niggemann, O.: Automated Generation of Timing Models in Distributed Production Plants. In: *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2013

[MPK15] Michels, J. S.; Paiz Gatica, C.; Köster, M.: Anomalien und Ineffizienz in Produktionsanlagen erkennen. In: *atp edition - Automatisierungstechnische Praxis* 57 (2015), no. 10, p. 26

[Nga22] Ngayap, V. I. T.: FreeRTOS on a MicroBlaze Soft-Core Processor with Hardware Accelerators. MA thesis. Paderborn University, 2022

[PKG+16] Paiz Gatica, C.; Köster, M.; Gaukstern, T.; Berlin, E.; Meyer, M.: An Industrial Analytics Approach to Predictive-Maintenance for Machinery Applications. In: *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation*. 2016

[QL15] Qiao, W.; Lu, D.: A survey on wind turbine condition monitoring and fault diagnosis—Part I: Components and subsystems. In: *IEEE Transactions on Industrial Electronics* 62 (2015), no. 10, pp. 6536–6545

[Ria17] Riaz, U.: Acceleration of Industrial Analytics Functions on a Platform FPGA. MA thesis. Paderborn University, 2017

[Ste13] Steinarsson, S.: Downsampling time series for visual representation. PhD thesis. 2013

[Wei] Weidmüller: *Monitoring Systems GmbH: BLADEcontrol condition monitoring system*. `https://mdcop.weidmueller.com/mediadelivery/asset/900_87890`

[WX06] Watson, S.; Xiang, J.: Real-time condition monitoring of offshore wind turbines. In: *Proceedings of European Wind Energy Conference & Exhibition (EWEC), Athens, Greece*. Vol. 27. 2006, p. 647654

[WZW+13] Wang, Y.; Zhou, X.; Wang, L.; Yan, J.; Luk, W.; Peng, C.; Tong, J.: SPREAD: A Streaming-Based Partially Reconfigurable Architecture and Programming Model. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21 (12 2013), pp. 2179–2192