ICT-56-2020 - Next Generation Internet of Things

D3.2
Initial report on the DL accelerator design

Document information	
Contract number	957197
Project website	www.vedliot.eu
Dissemination Level	PU (public)
Nature	R
Contractual Deadline	28.02.2022
Author	Mario Porrmann (UOS), Marco Tassemeier (UOS)
Contributors	Rene Griessl (UNIBI), Pedro Trancoso (CHALMERS), Fareed Mohammad Qararyah (CHALMERS), Stavroula Zouzoula (CHALMERS)
Reviewers	Elaheh Malekzadeh (EmbeDL), Micha vor dem Berge (CHR)

The VEDLIoT project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957197.

Changelog		
V 0.1	2021-11-04	Initial draft
V 0.2	2021-11-26	Update after discussion with all involved partners
V 0.21	2022-01-14	Initial version of Chapter 2
V 0.22	2022-01-18	Integration of HLS-based accelerators in Chapter 3
V 0.23	2022-02-05	Integration of input from Chalmers on open-source ML accelerators and planned co-designed accelerators
V 0.24	2022-02-06	Integration of performance measurements on SMARC
V 0.25	2022-02-07	Update of Chapter 2, pre-final testbed description
V 0.26	2022-02-12	Update of Chapter 3 with respect to SoA and accelerator design
V 0.3	2022-02-13	Pre-final version of Chapter 2 and 3
V 0.31	2022-02-14	Added Introduction, Executive Summary and Conclusion
V 0.4	2022-02-15	Version for internal review
V 0.5	2022-02-24	Incorporation of the feedback from the reviewers
V 0.6	2022-02-27	Final touches on Chapter 3
V 1.0	2022-02-28	Finalization

Table of Contents

Executive Summary

This deliverable summarizes the results of Task 3.2 (Design of a heterogeneous DL accelerator) of the VEDLIoT project and provides an initial report on the design of the reconfigurable accelerators (Task 3.3). Based on the first benchmarking results, summarized in D3.1, the most important accelerators to be integrated into the VEDLIoT hardware platforms have been selected, ranging from FPGAs via GPUs to TPUs. While the cloud platform RECS|Box and the near-edge platform t.RECS are already available in the VEDLIoT testbed, the far-edge platform u.RECS is not yet deployed. For early evaluation and development of the hardware-software solutions, a dedicated testbed has been set up for the u.RECS. It integrates the same compute and accelerator modules as the upcoming platform and is, e.g., used for the design of the basic FPGA infrastructure in Task 4.4. Additionally, it has been an important reference for the u.RECS hardware development and is used for the evaluation of new computing and acceleration modules.

For a first evaluation of the u.RECS setup, various accelerator implementations based on the Xilinx Deep Learning Processor Unit (DPU) have been realized on the embedded FPGA SoC. The results are compared in terms of performance and efficiency, extending the previous benchmarking activities and illustrating the large design space of the accelerators. These results are used as a basis for later comparison of the targeted reconfigurable accelerator designs in VEDLIoT. In Task 3.3, we started the development of a template-based approach that allows flexible realization of accelerators. A modular architecture is realized for efficient FPGA integration based on high-level synthesis. Enabling partial dynamic reconfiguration for enhanced efficiency is an inherent part of the design. The accelerator developments serve as a basis of the envisioned co-design approach in the upcoming Task 3.4 and are integrated into the hardware platforms developed in WP4.

1 Introduction

Being the second deliverable from WP3, this deliverable provides an overview of the results of Task 3.2, "Design of a heterogeneous DL accelerator", running from M6 to M15 of the project, and gives first insights into the developments of Task 3.3, "Design of a reconfigurable DL accelerator", which started in M12.

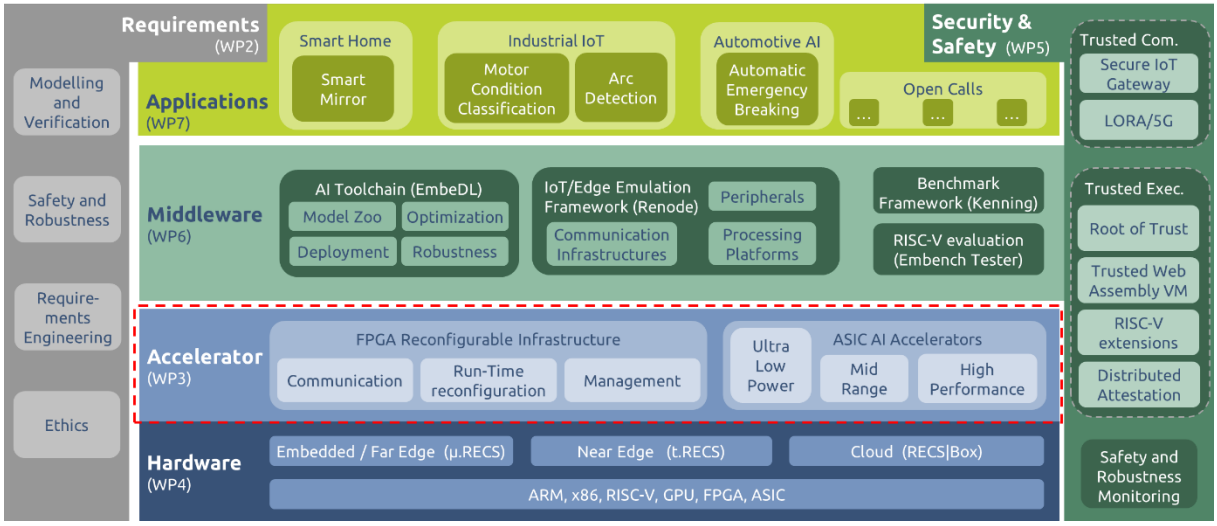

Figure 1: VEDLIoT project overview – this report covers Task 3.2 and Task 3.3 within WP3

Figure 1 illustrates the global picture of VEDLIoT. This report concentrates on the work on deep learning accelerators, carried out in WP3. Naturally, the work has a close link to the hardware platform design in WP4. The developed accelerators are utilizing the available hardware platforms and provide important input to the new developments. Via the middleware developed in WP6, the applications, as well as the use cases, will make use of the new accelerator designs.

Chapter 2 summarizes the work of Task 3.2. Based on the analysis of available accelerators, which is detailed in D3.1 [1], a subset of accelerators has been selected for integration into the RECS platforms used in VEDLIoT. Since the new u.RECS is not yet deployed, an emulation system has been set up, enabling FPGA and software development in parallel to the finalization of platform design. First experiments have been performed, utilizing the integrated reconfigurable SoCs for deep learning acceleration.

In Chapter 3, the ongoing accelerator design in Task 3.3 is summarized. As described in the Statement of Work, reconfigurable platforms are one of the main target platforms in VEDLIoT. The accelerator designs use a template-based approach, targeting high flexibility as well as high energy efficiency. Further improvements are envisioned by partially reconfiguring the FPGA designs at runtime, enabling to switch between different power/performance footprints depending on the current application requirements.

2 Heterogeneous DL Accelerator

Within VEDLIoT, three main hardware platforms are developed, used and extended, representing the complete compute continuum from far-edge via near-edge to the cloud. All platforms are based on the RECS architecture, following a highly modular and scalable approach, as detailed in D4.1 (First report on cognitive IoT hardware platform and microserver development) [2]. Based on the detailed analysis of hardware accelerators for machine learning, provided in D3.1 (Evaluation of existing architectures and compilers for DL) [1], the accelerators that match best the requirements within VEDLIoT have been selected for integration into the different platforms.

This chapter describes the results of Task 3.2 (Design of a heterogeneous DL accelerator). Based on existing components, reference implementations have been set up and are integrated into the VEDLIoT testbed. Our main goal was to provide a testbed setup that is scalable from small hardware platforms targeting minimum energy in the far-edge to high performance/cloud solutions. For the latter, we use the RECS|Box platform; near-edge solutions are realized based on t.RECS. For far-edge solutions, the new u.RECS is developed within VEDLIoT. In the following, the different variants of heterogeneous accelerators, used in VEDLIoT, are briefly sketched, focusing on the targeted ML accelerators. Details about the architecture of the different RECS platforms are provided in D4.1. A complete list of available accelerators and compute modules is provided in D7.1 (First report on testbed deployment and maintenance) [3].

RECS|Box Platform

RECS|Box is the high-end platform of the RECS family, enabling the integration of a wide variety of heterogeneous microservers, ranging from server-grade CPUs to GPUs and FPGAs. RECS|Box is available in the VEDLIoT testbed. Focusing on a high number of embedded modules, large FPGAs and GPUs with high-speed interconnects between the different microservers, RECS|Box features a wide variety of accelerators. With respect to DL acceleration in VEDLIoT, among others, the following accelerators are available or will be made available in the testbed:

- Intel Stratix10
  The available COM-Express module comprises an Intel Stratix 10 SX 2800 SoC FPGA, integrating an FPGA fabric with 2.8M logic elements and a Quad-core 64-bit ARM Cortex-A53 processor system. Combined with 56 GByte DDR4 memory and 2x10 Gbps network links, the module is especially suited for large FPGA designs that can be scaled to multi-FPGA systems.
- NVIDIA Xavier NX board
  The Jetson Xavier NX combines a 6-core NVIDIA Carmel ARM v8.2 64-bit CPU with a Volta GPU (384 cores and 48 tensor cores), two NVIDIA Deep Learning Accelerators (NVDLA) and 8 GByte unified CPU-GPU memory. The board has shown a high energy efficiency in the early benchmarks and is integrated into the RECS|Box with a dedicated carrier board.
- NVIDIA A100
  For training, PCIe-based GPUs are indispensable. Hence, NVIDIA A100 data center cards are integrated into the RECS|Box, providing high compute power for the entire consortium.

- Xilinx Versal
  Xilinx Versal ACAPs (Adaptive Compute Acceleration Platforms) combine the traditional reconfigurable FPGA fabrics with embedded processors and vector processing engines, interconnected by a programmable network on-chip. The new devices are very promising platforms for the reconfigurable accelerator designs targeted in VEDLIoT. Therefore, Xilinx Versal ACAPs will be integrated into the RECS|Box vie PCIe.

t.RECS Platform

Like the RECS|Box, also the t.RECS (tiny RECS) is integrated into the RECS testbed. The platform targets near-edge applications with low latency requirements, based on the new COM-HPC standard. A wide variety of additional compute and accelerator modules can be integrated. For boards that are not based on the COM-HPC standard, adaptor boards have been developed, enabling easy integration into the compact edge server. The following accelerators have been chosen for integration into t.RECS in the VEDLIoT testbed:

- Xilinx UltraScale+ ZU19
  For integration of the reconfigurable accelerators, developed in VEDLIoT, a module based on the new COM-HPC standard is used. It integrates a Xilinx UltraScale ZU11EG, ZU17EG or ZU19EG FPGA SoC, up to 8 GByte 72-bit DDR4 connected to PS and up to 16 GByte 72-bit DDR4 connected to PL.
- Intel Stratix10
  Like for the RECS|Box, the SoC FPGA is integrated as a COM-Express board in the t.RECS.
- NVIDIA Xavier AGX
  The NVIDIA Jetson Xavier AGX combines an 8-core NVIDIA Carmel ARM v8.2 64-bit CPU with a Volta GPU (512 cores and 64 tensor cores), two NVIDIA Deep Learning Accelerators (NVDLA) and 32 GByte unified CPU-GPU memory. Since the board has shown a high energy efficiency in the early benchmarks, it is integrated into the t.RECS via a COM-HPC to Xavier AGX adapter.
- NVIDIA A100
  Like for the RECS|Box, GPU accelerators for training will be integrated into the t.RECS via PCIe.
- Xilinx Versal (via PCIe)
  Like for the RECS|Box, Xilinx Versal ACAPs will be integrated into the t.RECS via PCIe.

u.RECS Platform

u.RECS is the new modular far-edge platform, developed in VEDLIoT (cf. D4.1). Since the deployment is ongoing, no complete system is available at the moment. Hence, an evaluation system has been set up based on discrete components that enable early access to the compute nodes and accelerators that are integrated into the u.RECS. A detailed description of the u.RECS testbed is provided in Chapter 2.1. For u.RECS, we identified the following accelerators based on the requirements deliverables and D3.1 [1]. Compared to the larger RECS systems, here the focus is on low power, high energy efficiency and direct connection to sensors.

- NVIDIA Xavier NX
  As discussed above, the Jetson Xavier NX has shown a high energy efficiency in the early benchmarks. It serves as one of the main compute/accelerator modules in the u.RECS.

- Xilinx UltraScale via SMARC modules
  The SMARC standard has been selected for the u.RECS since it enables easy integration of a wide variety of modules. In addition to compute modules, e.g., FPGA-based SMARC modules are available. For integration of the reconfigurable accelerators, developed within VEDLIoT, into the u.RECS, Xilinx UltraScale+ FPGAs are used.
- u.RECS enables the integration of additional accelerators via M.2. Based on the benchmarks and analyses performed for D3.1 [1], the following accelerators have been selected:
  - Hailo Hailo-8
    The Hailo-8 edge AI processor features up to 26 tera-operations per second with a typical power consumption of 2.5 W. It can be used as a standalone processor or as co-processor, making it an interesting platform for integration into the u.RECS.
  - Google Coral
    Google coral provides a variety of products based on the Edge TPU (Tensor Processing Unit), ranging from development boards via USB accelerators to system-on-modules. Combining high energy efficiency and good tool support, the Google Edge TPU is a good candidate for integration into the u.RECS.
  - Intel Myriad
    Intel Movidius Myriad X is a vector processing unit, featuring a Neural Compute Engine for deep-learning inference. It is selected because of its high energy efficiency and good tool support.

Like all RECS platforms, u.RECS can be easily extended with new modules as soon as they become available. Due to their high efficiency and good tool flow integration, NVIDIA modules are important components for all platforms. In the hardware designs, special care is taken to not only support the currently available modules, but to enable also easy integration of the next generation Jetson Orin NX modules [4]. Furthermore, new modules with upcoming reconfigurable devices like the Xilinx Versal AI Edge series will be of high interest. Additional details about additional accelerator modules are provided in D4.1 [2].

2.1 u.RECS Testbed Integration

Since the u.RECS is not yet available, an emulation system has been set up, which is embedded into the VEDLIoT testbed, enabling early access for all partners. In the following, the main setup is discussed.

2.1.1 Requirements for the u.RECS Testbed

Main purpose of the u.RECS emulation system is to provide a platform for easy access to the critical hardware components while the final system is in development. This is especially critical for the software and FPGA developments that are foreseen in VEDLIoT. While in general various compute and accelerator platforms are available at the partners and in the VEDLIoT testbed, development of the basic software infrastructure as well as the basic FPGA implementations in WP4 require a development system that is as similar as possible to the final implementation. Hence, especially support for the same compute and accelerator modules that are targeted in the final u.RECS is required.

Densely integrated power and performance monitoring is an important feature of the RECS platforms and is also be available in the u.RECS. The u.RECS testbed targets not only the development of the basic software and FPGA infrastructure components but also the early

evaluation of new compute modules and accelerators. Hence, means for power and performance evaluation are integrated into the platform.

Since PCIe communication between the SMARC modules and the Jetson NX is not targeted in the first place, communication between these devices is established mainly via Gigabit Ethernet in the u.RECS testbed. Nevertheless, accelerators can be directly connected to the compute modules via M.2.

Since several of the targeted compute modules required the design of module-specific adaptors, these module-adaptor-combinations can also be efficiently tested first on the emulation system. This makes the u.RECS testbed an important setup, especially for debugging, even after the real hardware is available.

2.1.2 System Architecture

Figure 2 shows the high-level architecture of the u.RECS, highlighting the parts that are covered by the u.RECS testbed. As mentioned above, the main focus is on the early integration of the main compute and accelerator modules. Since Raspberry Pi compute modules are available locally at all partners, we mainly concentrated on the integration of SMARC modules and Jetson NX. Integration of additional modules is easily possible and will be pushed forward whenever necessary.

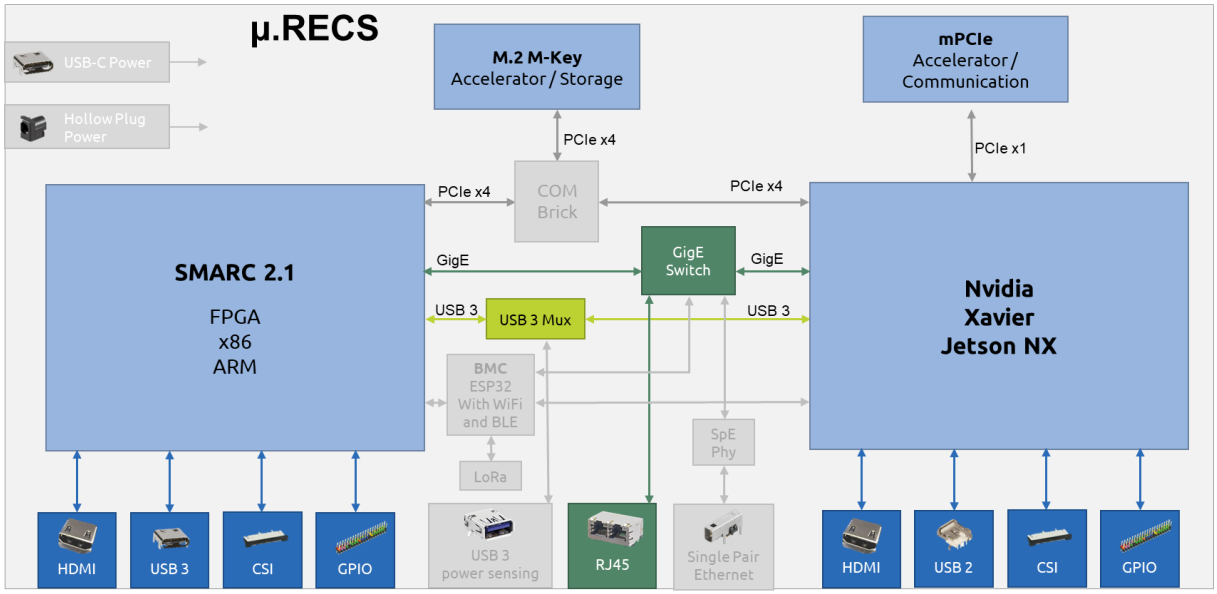

Figure 2: High-level block diagram of the u.RECS platform

The Jetson Xavier NX is directly integrated into the testbed using the developer kit provided by NVIDIA. The important interfaces required in this setup include:

- Gigabit Ethernet
- M.2 Key E and M
- 2x MIPI CSI-2 D-PHY lanes camera interface
- GPIOs, I2C, I2S, SPI, UART

Integration of modules based on the SMARC 2 standard is especially important since this is the main basis for FPGA integration into u.RECS. Hence, the design of the basic FPGA infrastructure in Task 4.4 as well as the accelerator development in Task 3.3 and Task 3.4 heavily depend on the availability of FPGA-based SMARC modules in the testbed. Focusing

on recent FPGA architectures suitable for integration into u.RECS, the following module has been selected as an ideal initial candidate for integration into µ.RESC. A discussion about available modules is provided in D4.1 [2].

SECO RUSSELL Module

Based on its feature set and current availability, the SECO RUSSELL (formerly codenamed SM-B71) module has been selected (cf. Figure 3). It integrates a Xilinx Zynq UltraScale+ MPSoC. Based on the requirements, the user can select between various devices, ranging from ZU2CG to ZU5CG, ZU2EG to ZU5EG, and ZU4EV or ZU5EV. In addition to different sized FPGA fabrics, all devices include a dual-core Arm Cortex-A53 application processing unit and a dual-core Arm Cortex-R5 real-time processing unit. Up to 8 GByte DDR4-2400 on-board memory is connected to the processing system. Additionally, up to 2 GByte is directly attached to the programmable logic.

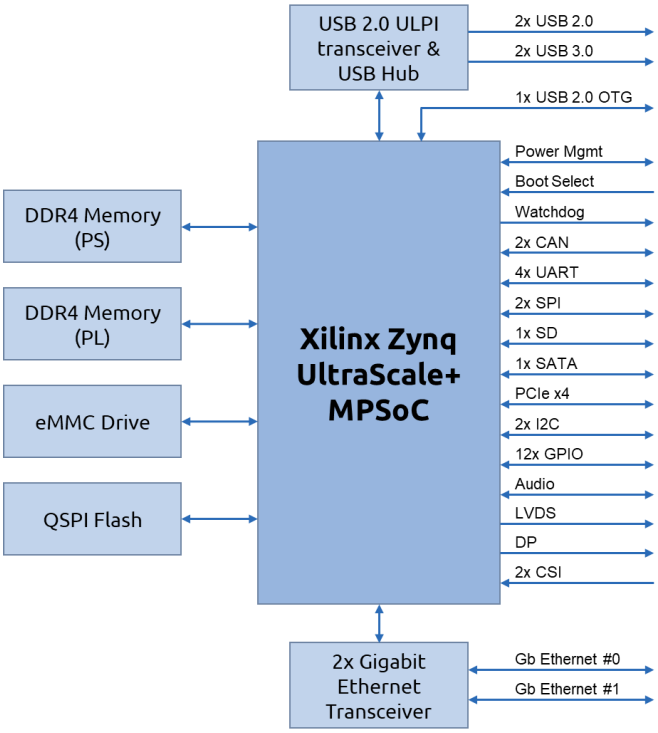

Figure 3: High-level block diagram of the SECO RUSSELL SMARC module

Additional on-board logic and external interfaces of the module include:

- Up to 2 x Gigabit Ethernet interfaces
- USB: 1x USB 2.0 OTG; 2x USB 2.0 Host; 2x USB 3.0 Host
- PCIe x4 interface
- External SATA Gen3 Channel
- SD interface
- On-board QSPI
- 18- / 24-bit Dual Channel LVDS interface
- DisplayPort interface
- 2 CSI interfaces
- Various serial ports, I2C, SPI and GPIOs

SMARC Baseboards

Among the available SMARC baseboards, two have been selected for integration of the u.RECS testbed. The boards provide all interfaces that are available on the u.RECS so that both remote evaluations, as well as filed operation of the platforms, are feasible. Having different baseboards available in the testbed enables early testing of compatibility between boards, relevant especially for design and device selection during the u.RECS development.

Kontron SMARC Evaluation Carrier 2.0

Figure 4 shows a block diagram of the SMARC evaluation carrier 2.0 from Kontron. The board is compliant with the SMARC 2.0 specification and provides a wide variety of interfaces. A detailed specification of the board can be found in [5].

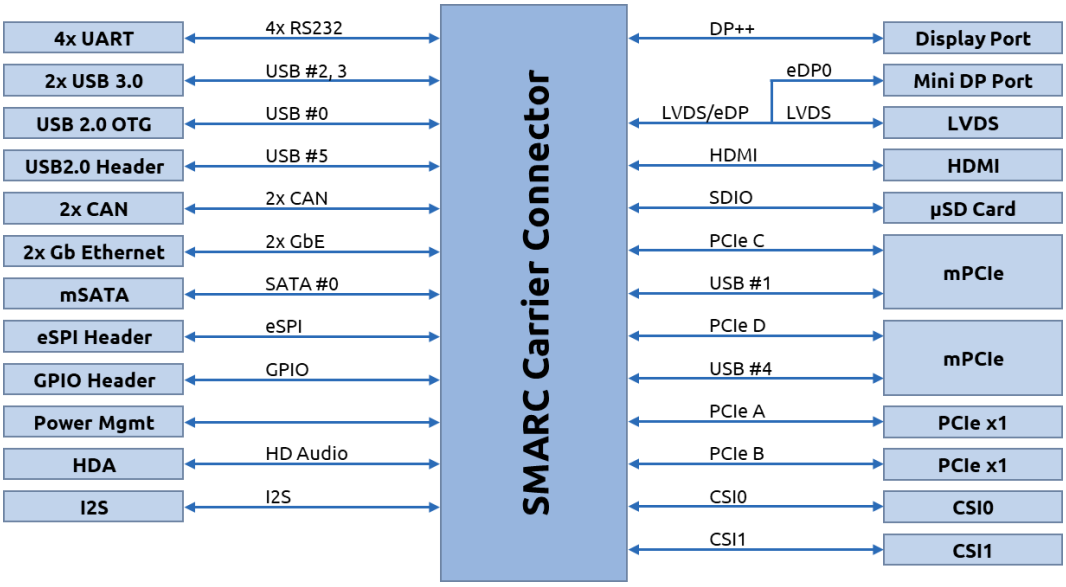

Figure 4: Functional block diagram of the Kontron SMARC evaluation carrier 2.0

Main interfaces provided by the Kontron SMARC evaluation carrier 2.0

- 2x MIPI CSI camera interface
- switchable audio interface
- 4x UARTS
- 2x miniPCIe with SIM card support
- 2x PCIe x1
- 2x CAN Bus interface
- 12x GPIO
- 2x Gbit LAN
- Dual channel 24-bit LVDS shared with eDP, HDMI interface, DP++ interface
- 1x USB 2.0 dual role, 1x USB 2.0, 2x USB 3.0, (2x USB connected to mPCIe)
- SD Card, mSATA

Advantech SOM-DB2500

Figure 5 shows the functional block diagram of the development carrier board SOM-DB2500 from Advantech. The board supports SMARC 2.0 modules as well as SMARC 2.1 modules. Like the carrier from Kontron, it supports a wide variety of interfaces. The SOM-DB2500 has been integrated into the testbed especially, because it supports M.2 connectors, enabling easy integration of additional ML accelerators. A detailed specification of the board can be found in [6].

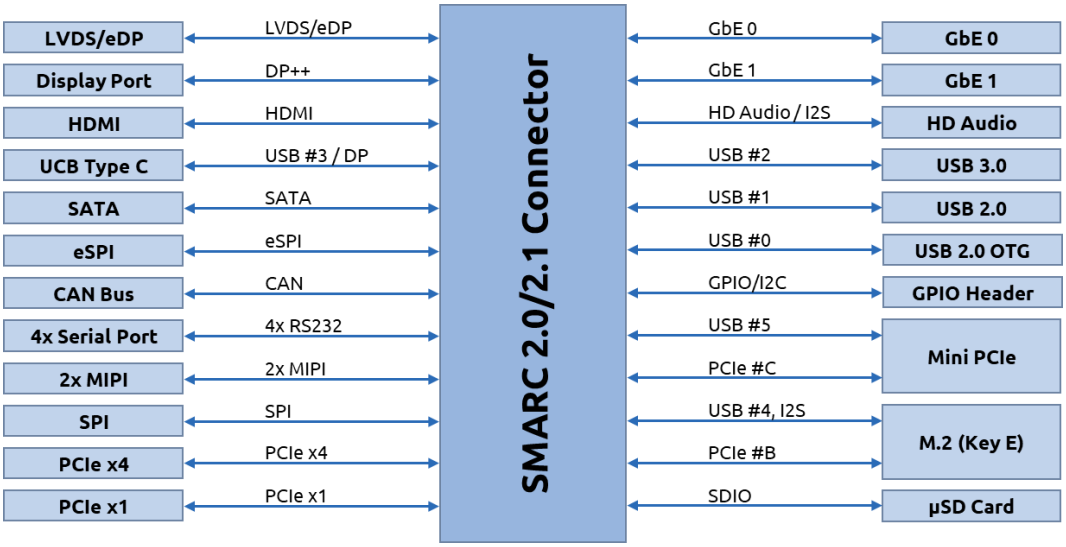

Figure 5: Functional block diagram of the ADVANTECH SOM-DB2500

Main interfaces provided by the ADVANTECH SOM-DB2500

- 2x MIPI CSI camera interface
- Realtek ALC888 HD Audio
- 4x COM Port
- 1x miniPCIe with SIM card support
- 1x PCIe x1, 1x PCIe x4
- 1x CAN Bus interface, 2x I2C Bus
- 12x GPIO
- 2x Gbit LAN
- Dual channel 24-bit LVDS
- 1x HDMI, 1x Display Port, 1x Display Port via USB Type-C
- 1x USB3.0 (Type A), 1x USB3.0 (Type C)
- SD Card, M.2 (Key E), SATA3.0

2.1.3 Testbed Access and Power Monitoring

The u.RECS emulation system is integrated into the testbed at Bielefeld University, enabling easy access for all project partners. The u.RECS setup uses the same infrastructure and access mechanisms as the RECS|Box and t.RECS servers. A detailed description of the testbed is provided in D7.1 [3].

Detailed power monitoring is facilitated with an oscilloscope, enabling detailed analysis of the power over time. Remote access to the oscilloscope is possible if the system is not used

locally. High-speed power measurements provide interesting insights into the running application, which is of high interest, especially for the development of the reconfigurable accelerators in VEDLIoT (cf. D3.1 [1]).

2.2 First Experiments using the u.RECS Testbed

For first experiments utilizing the u.RECS testbed, we have chosen the same models that are used for the evaluation of ML accelerators in D3.1 [1]. The results presented in this chapter have been achieved using the SECO RUSSELL (SM-B71) on the ADVANTECH SOM-DB8500 SMARC-baseboard. The SMARC module is available in a wide variety of options. The following configuration has been used in our experiments:

SECO RUSSELL (SM-B71)

- Xilinx Zynq UltraScale+ XCZU4EG-1 FPGA
  - Quad-core Arm Cortex-A53, Dual-core Arm Cortex-R5F, Mali-400MP2
  - 88k 6-input look-up tables (LUTs)
  - 176k Flip-Flops (FFs)
  - 728 DSP Slices
  - 128 36kb BRAM blocks (4.5 Mb total)
  - 48 288kb URAM blocks (13.5 Mb total)
- 2 GByte 64-Bit DDR4 SDRAM (PS)
- 512 MByte 64-Bit DDR4 SDRAM (PL)
- 4 GByte eMMC

As a baseline for later comparison to the accelerators developed in VEDLIoT, the Xilinx Deep Learning Processor Unit (DPU) [7] is used in VEDLIoT. A detailed discussion of the tool flow and measurements on other FPGA platforms is provided in D3.1 [1].

Utilizing the basic FPGA infrastructure, developed in Task 4.4, the FPGA design for this evaluation contains the Xilinx DPU and the necessary infrastructure, such as the AXI-buses for communication and an interrupt controller. Additionally, Ethernet, SD-card and serial interfaces of the processor system in the SoC are activated. For the measurements, a PetaLinux-image [8] was created (including the configuration for the FPGA fabric as well as the PetaLinux OS for the processing system) and booted from an SD-card. In addition to the boot image, the models, as well as the test data, are also stored on the SD-card.

Power measurements for the experiments have been performed with high-speed current logging. In addition to the complete system power, the u.RECS test setup offers the possibility to power the baseboard components and the SMARC modules separately. In the following tables, the power values for the SMARC module itself are presented. The power consumption for the baseboard stays constant. Independent of the DPU configurations and regardless of whether the system is in idle mode or actively running inferences, the power consumption of the baseboard is on average 1.02 W. The used fan, which is optional for cooling the module, adds 0.35W to the baseboard power consumption.

The PetaLinux-image was created using the Xilinx tool flow, version 2021.1. For the AI-models preparation, the Xilinx Vitis-AI tool flow [9], version 1.4, was used. The following models have been used: MobileNetV2, ResNet50 and YoloV4. For analyzing the benefit of further model optimizations, which is also an important topic in VEDLIoT, a pruned YoloV4 model has been added, which was optimized by the Xilinx model optimizer for inference on

the DPU. The number of operations needed for one inference are given in Table 1. These serve as the basis for the calculation of the performance ratios in the subsequent tables.

Table 1: Operations and multiply-adds of the used models

Model	Operations / Inference
MobileNetV2	0.61 GOps[1] (0.305 Billion Mul-Adds)
ResNet50	7.78 GOps (3.89 Billion Mul-Adds)
YoloV4	60.4 GOps (30.2 Billion Mul-Adds)
YoloV4 Pruned	38.2 GOps (19.1 Billion Mul-Adds)

Various DPU configurations have been implemented and are compared in terms of performance and efficiency. At design time, several parameters for the DPU configurations can be chosen, like high/low DPS usage or dedicated low power modes. For the presented implementations, the configurations that showed maximum performance in our experiments have been chosen. In addition to the size and number of DPUs, the number of software threads has been varied. Single-threaded refers to an implementation where pre-processing, DPU calls, and post-processing are performed by a single thread. In the multithreaded implementations, the threads are evenly distributed between pre-processing, DPU-calls, and post-processing [1]. Batch size is one for all reported FPGA implementations.

For evaluation, we have used several configurations based on three different DPU cores, ranging from the smallest one (B512) to the largest (B4096). The number behind the "B" corresponds to the peak INT8 operations per clock cycle for the DPU core. For all models, a small configuration has been realized, using one or two B512 DPUs, respectively. A mid-size version has been implemented based on the B2304 DPU. While all DPUs are evaluated based on clock frequency of 300 MHz, the B2304 is also used at 200 MHz, showing the impact of a scaled DPU clock frequency on the overall performance and energy efficiency. Finally, the two largest DPUs (B3136 and B4096) are evaluated. The theoretical peak performance, which depends on the DPU architecture and on the used clock frequency, is provided for all configurations in the subsequent tables.

2.2.1 Results for ResNet50

A set of 1,000 images has been used for the performance evaluation based on ResNet50. The following three tables summarize the results achieved for the SECO RUSSELL using different DPU configurations. For all analysed models, pre-processing includes reading the input data from the SD card and during post-processing, the results are stored in external DRAM. The metrics that are used for evaluation are discussed in detail in D3.1 (Chapter 6.1) [1] and are briefly summarized here:

- **Inference time**: Time for the completion of the inference operation. This time does not include any data pre- or post-processing times.
- **Latency:** Total inference time for a request including both data pre- and post-processing.

[1] Giga Operations

- **Achieved performance:** In contrast to peak performance, this is the performance that is achieved for the execution of a specific DL model. This value can be reported in GOPS (Giga Operations Per Second) or/and in inferences per second.
- **Peak performance:** Theoretical value determined by the available hardware and operating frequency.
- **Performance Ratio:** Ratio between the achieved performance and the peak performance. This metric is interesting to determine how effectively the hardware is being used for the inference operation.
- **Power:** Average system power (here: power of the SMARC module)
- **Energy/Inference:** Energy [J] required for a single inference
- **Power Efficiency:** Metric for the power efficiency of the execution of an inference operation on a particular architecture. This metric is a combination of the achieved performance and power and is reported in Giga Operations Per Second (GOPS) per Watt (W).

In the tables, power refers to average power during inference. Idle power is reported for two system states: First, the system is powered on but the reconfigurable SoC is not yet configured; second, the reconfigurable SoC is configured and Linux has finished booting on the embedded Arm cores (but ML inference has not yet started).

The achieved results significantly depend on the utilization of the DPU. For single thread implementations, the latencies of the single steps (pre-processing, inference, and post-processing) add up to the total time for one frame. When using multithreading, the latency increases but the throughput (inferences/s) also significantly grows. For all implementations, a higher number of threads results in higher performance in terms of inferences per second. In parallel, also the latency increases. Hence, for a given application, the optimal trade-off between latency and performance can be chosen.

Table 2: Evaluation of ResNet50 on the SECO RUSSELL (SM-B71) using B512 DPUs

Platform	**SM-B71 on SOM-DB2500**					
DPU	**B512 x1, 300MHz**			**B512 x2, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	62.78	121.51	243.59	63.57	84.26	164.33
Latency [ms]	79.69	138.50	260.58	80.51	101.29	181.44
Achieved performance [Inferences/s]	12.53	16.39	16.38	12.40	23.62	24.24
Achieved performance [GOPS]	97.48	127.51	127.44	96.47	183.76	188.59
Peak performance [GOPS]	153.6	153.6	153.6	307.2	307.2	307.2
Performance Ratio	63.46%	83.01%	82.97%	31.40%	59.82%	61.39%
Cost Metrics						
FPGA Resources	35k (39.3%) LUTs, 44k (24.8%) FFs, 110 (15.1%) DSP, 13.5 (10.6%) BRAM, 16 (33.3%) URAM			68k (77.4%) LUTs, 89k (50.9%) FFs, 220 (30.2%) DSP, 49 (38.3%) BRAM, 32 (66.7%) URAM		

Power [W]	6.51	6.76	6.79	7.00	7.91	7.83
Idle Power [W]	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14
Energy/Inference [J]	0.520	0.412	0.415	0.565	0.335	0.323
Power Efficiency [GOPS/W]	14.97	18.86	18.77	13.78	23.23	24.09

Table 3: Evaluation of ResNet50 on the SECO RUSSELL (SM-B71) using B2304 DPUs

Platform	**SM-B71 on SOM-DB2500**					
DPU	**B2304 x1, 200MHz**			**B2304 x1, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	27.55	51.36	102.87	19.26	34.86	70.01
Latency [ms]	44.49	68.36	119.94	36.18	51.98	87.26
Achieved performance [Inferences/s]	22.48	38.69	38.69	27.60	56.78	56.85
Achieved performance [GOPS]	174.89	301.01	301.01	214.73	441.75	442.29
Peak performance [GOPS]	460.8	460.8	460.8	691.2	691.2	691.2
Performance Ratio	37.95%	65.32%	65.32%	31.07%	63.91%	63.99%
Cost Metrics						
FPGA Resources	47k (53.7%) LUTs, 79k (45.0%) FFs, 422 (58.0%) DSP, 61 (47.7%) BRAM, 40 (83.3%) URAM					
Power [W]	8.14	8.24	8.30	9.92	10.08	10.40
Idle Power [W]	0.07/6.02	0.07/6.02	0.07/6.02	0.07/6.83	0.07/6.83	0.07/6.83
Energy/Inference [J]	0.362	0.213	0.215	0.359	0.178	0.183
Power Efficiency [GOPS/W]	21.49	36.53	36.27	21.65	43.82	42.73

Table 4: Evaluation of ResNet50 on the SECO RUSSELL (SM-B71) using B3136 and B4096 DPUs

Platform	**SM-B71 on SOM-DB2500**					
DPU	**B3136 x1, 300MHz**			**B4096 x1, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	16.61	29.75	59.94	13.49	23.75	48.33
Latency [ms]	33.50	46.94	77.43	30.39	40.91	66.05

Achieved performance [Inferences/s]	29.73	66.38	66.31	32.76	83.16	82.31
Achieved performance [GOPS]	231.30	516.44	515.89	254.87	646.98	640.37
Peak performance [GOPS]	940.8	940.8	940.8	1228.8	1228.8	1228.8
Performance Ratio	24.89%	54.89%	54.84%	20.74%	52.65%	52.11%
Cost Metrics						
FPGA Resources	51k (58.6%) LUTs, 90k (51.2%) FFs, 548 (75.3%) DSP, 71 (55.5%) BRAM, 44 (91.7%) URAM			57k (64.7%) LUTs, 108k (61.7%) FFs, 690 (94.8%) DSP, 81 (63.3%) BRAM, 48 (100.0%) URAM		
Power [W]	9.07	11.64	11.62	9.78	13.39	13.20
Idle Power [W]	0.07/7.09	0.07/7.09	0.07/7.09	0.07/7.56	0.07/7.56	0.07/7.56
Energy/Inference [J]	0.305	0.175	0.175	0.298	0.161	0.160
Power Efficiency [GOPS/W]	25.50	44.37	44.40	26.06	48.32	48.51

2.2.2 Results for MobileNetV2

MobileNetV2 was used as an example for a small neural network. A set of 1000 images has been used for the evaluations, summarized in the following three tables. For the medium and large DPUs, the utilization, i.e., the achieved performance vs. the theoretical maximum, is quite low. This is mainly due to the I/O overhead since in this case pre-processing requires more time than the inference on the DPU. Obviously, a small implementation like the B512 is best suited for such nets.

Table 5: Evaluation of MobileNetV2 on the SECO RUSSELL (SM-B71) using B512 DPUs

Platform	**SM-B71 on SOM-DB2500**					
DPU	**B512 x1, 300MHz**			**B512 x2, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	11.20	19.38	39.62	11.56	15.22	26.69
Latency [ms]	28.11	36.58	57.47	28.41	32.33	45.55
Achieved performance [Inferences/s]	35.45	101.64	100.39	35.04	117.11	148.68
Achieved performance [GOPS]	21.62	62.00	61.24	21.37	71.44	90.69
Peak performance [GOPS]	153.6	153.6	153.6	307.2	307.2	307.2
Performance Ratio	14.08%	40.36%	39.87%	6.96%	23.26%	29.52%
Cost Metrics						

FPGA Resources	35k (39.3%) LUTs, 44k (24.8%) FFs, 110 (15.1%) DSP, 13.5 (10.6%) BRAM, 16 (33.3%) URAM			68k (77.4%) LUTs, 89k (50.9%) FFs, 220 (30.2%) DSP, 49 (38.3%) BRAM, 32 (66.7%) URAM		
Power [W]	6.14	7.06	6.96	6.65	7.87	8.30
Idle Power [W]	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14
Energy/Inference [J]	0.173	0.069	0.069	0.190	0.067	0.056
Power Efficiency [GOPS/W]	3.52	8.78	8.80	3.21	9.08	10.93

Table 6: Evaluation of MobileNetV2 on the SECO RUSSELL (SM-B71) using B2304 DPUs

Platform	**SM-B71 on SOM-DB2500**					
DPU	**B2304 x1, 200MHz**			**B2304 x1, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	7.00	8.10	21.95	5.48	6.07	7.88
Latency [ms]	23.93	25.12	41.35	22.37	23.25	27.11
Achieved performance [Inferences/s]	41.60	117.81	179.74	44.44	116.95	201.78
Achieved performance [GOPS]	25.38	71.86	109.64	27.11	71.34	123.09
Peak performance [GOPS]	460.8	460.8	460.8	691.2	691.2	691.2
Performance Ratio	5.51%	15.59%	23.79%	3.92%	10.32%	17.81%
Cost Metrics						
FPGA Resources	47k (53.7%) LUTs, 79k (45.0%) FFs, 422 (58.0%) DSP, 61 (47.7%) BRAM, 40 (83.3%) URAM					
Power [W]	6.62	7.37	8.30	7.41	8.32	9.40
Idle Power [W]	0.07/6.02	0.07/6.02	0.07/6.02	0.07/6.83	0.07/6.83	0.07/6.83
Energy/Inference [J]	0.159	0.063	0.046	0.167	0.071	0.047
Power Efficiency [GOPS/W]	3.83	9.75	13.21	3.66	8.57	13.09

Table 7: Evaluation of MobileNetV2 on the SECO RUSSELL (SM-B71) using B3136 and B4096 DPUs

Platform	SM-B71 on SOM-DB2500					
DPU	**B3136 x1, 300MHz**			**B4096 x1, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	5.22	5.69	7.22	4.71	5.05	6.19
Latency [ms]	22.08	22.86	26.58	21.58	22.14	25.41
Achieved performance [Inferences/s]	44.93	116.81	201.00	46.05	117.42	202.00
Achieved performance [GOPS]	27.41	71.25	122.61	28.09	71.63	123.22
Peak performance [GOPS]	940.8	940.8	940.8	1228.8	1228.8	1228.8
Performance Ratio	2.91%	7.57%	13.03%	2.29%	5.83%	10.03%
Cost Metrics						
FPGA Resources	51k (58.6%) LUTs, 90k (51.2%) FFs, 548 (75.3%) DSP, 71 (55.5%) BRAM, 44 (91.7%) URAM			57k (64.7%) LUTs, 108k (61.7%) FFs, 690 (94.8%) DSP, 81 (63.3%) BRAM, 48 (100.0%) URAM		
Power [W]	7.73	8.59	9.77	8.13	8.95	10.03
Idle Power [W]	0.07/7.09	0.07/7.09	0.07/7.09	0.07/7.56	0.07/7.56	0.07/7.56
Energy/Inference [J]	0.172	0.074	0.049	0.176	0.076	0.050
Power Efficiency [GOPS/W]	3.55	8.29	12.55	3.46	8.00	12.29

2.2.3 Results for YoloV4

For performance evaluation of YoloV4, a set of 404 images has been used. The results are summarized in the subsequent tables, showing a high utilization of the different DPUs.

Table 8: Evaluation of YoloV4 on the SECO RUSSELL (SM-B71) using B512 DPUs

Platform	SM-B71 on SOM-DB2500					
DPU	**B512 x1, 300MHz**			**B512 x2, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	464.48	895.26	1786.11	465.03	617.53	1194.42
Latency [ms]	475.76	907.73	1798.71	476.31	629.82	1206.90
Achieved performance [Inferences/s]	2.10	2.23	2.23	2.10	3.23	3.33
Achieved performance [GOPS]	126.84	134.69	134.69	126.84	195.09	201.13

Peak performance [GOPS]	153.6	153.6	153.6	307.2	307.2	307.2
Performance Ratio	82.58%	87.69%	87.69%	41.29%	63.51%	65.47%
Cost Metrics						
FPGA Resources	35k (39.3%) LUTs, 44k (24.8%) FFs, 110 (15.1%) DSP, 13.5 (10.6%) BRAM, 16 (33.3%) URAM			68k (77.4%) LUTs, 89k (50.9%) FFs, 220 (30.2%) DSP, 49 (38.3%) BRAM, 32 (66.7%) URAM		
Power [W]	6.72	6.82	6.80	7.23	7.83	7.90
Idle Power [W]	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14
Energy/Inference [J]	3.202	3.057	3.051	3.442	2.424	2.372
Power Efficiency [GOPS/W]	18.88	19.75	19.81	17.54	24.92	25.46

Table 9: Evaluation of YoloV4 on the SECO RUSSELL (SM-B71) using B2304 DPUs

Platform	**SM-B71 on SOM-DB2500**					
DPU	**B2304 x1, 200MHz**			**B2304 x1, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	194.17	355.29	708.87	135.02	237.10	473.15
Latency [ms]	205.48	368.15	721.69	146.39	250.291	486.51
Achieved performance [Inferences/s]	4.86	5.60	5.62	6.81	8.38	8.42
Achieved performance [GOPS]	293.54	338.24	339.45	411.32	506.15	508.57
Peak performance [GOPS]	460.8	460.8	460.8	691.2	691.2	691.2
Performance Ratio	63.70%	73.40%	73.67%	59.51%	73.23%	73.58%
Cost Metrics						
FPGA Resources	47k (53.7%) LUTs, 79k (45.0%) FFs, 422 (58.0%) DSP, 61 (47.7%) BRAM, 40 (83.3%) URAM					
Power [W]	8.23	8.57	8.59	10.13	10.87	10.89
Idle Power [W]	0.07/6.02	0.07/6.02	0.07/6.02	0.07/6.83	0.07/6.83	0.07/6.83
Energy/Inference [J]	1.693	1.530	1.528	1.488	1.297	1.293
Power Efficiency [GOPS/W]	35.67	39.47	39.52	40.60	46.56	46.70

Table 10: Evaluation of YoloV4 on the SECO RUSSELL (SM-B71) using B3136 and B4096 DPUs

Platform	SM-B71 on SOM-DB2500					
DPU	**B3136 x1, 300MHz**			**B4096 x1, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	109.05	185.40	370.14	82.14	131.71	263.31
Latency [ms]	120.34	198.62	383.72	93.42	144.51	276.33
Achieved performance [Inferences/s]	8.28	10.76	10.76	10.66	15.12	15.12
Achieved performance [GOPS]	500.11	649.90	649.90	643.86	913.25	913.25
Peak performance [GOPS]	940.8	940.8	940.8	1228.8	1228.8	1228.8
Performance Ratio	53.16%	69.08%	69.08%	52.40%	74.32%	74.32%
Cost Metrics						
FPGA Resources	51k (58.6%) LUTs, 90k (51.2%) FFs, 548 (75.3%) DSP, 71 (55.5%) BRAM, 44 (91.7%) URAM			57k (64.7%) LUTs, 108k (61.7%) FFs, 690 (94.8%) DSP, 81 (63.3%) BRAM, 48 (100.0%) URAM		
Power [W]	11.20	12.49	12.51	13.14	15.42	15.44
Idle Power [W]	0.07/7.09	0.07/7.09	0.07/7.09	0.07/7.56	0.07/7.56	0.07/7.56
Energy/Inference [J]	1.352	1.161	1.173	1.233	1.020	1.021
Power Efficiency [GOPS/W]	44.65	52.03	51.95	49.00	59.23	59.15

2.2.4 Results for YoloV4 Pruned

Compared to the original YoloV4, used in the evaluation above, the pruned version of YoloV4 reduces the number of required operations per inference from 60.4 GOps to 38.2 GOps. Although the power efficiency in terms of GOPS/W stays comparable to the non-pruned version, the energy efficiency in terms of Energy/Inference is significantly reduced, as can be seen from the tables below.

Table 11: Evaluation of YoloV4 Pruned on the SECO RUSSELL (SM-B71) using B512 DPUs

Platform	SM-B71 on SOM-DB2500					
DPU	**B512 x1, 300MHz**			**B512 x2, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	302.40	571.4	1140.19	302.82	401.12	762.17
Latency [ms]	313.71	583.86	1152.67	314.12	413.42	774.62

Achieved performance [Inferences/s]	3.18	3.50	3.49	3.18	4.96	5.22
Achieved performance [GOPS]	121.48	133.70	133.32	121.48	189.47	199.40
Peak performance [GOPS]	153.6	153.6	153.6	307.2	307.2	307.2
Performance Ratio	79.09%	87.04%	86.80%	39.54%	61.68%	64.91%
Cost Metrics						
FPGA Resources	35k (39.3%) LUTs, 44k (24.8%) FFs, 110 (15.1%) DSP, 13.5 (10.6%) BRAM, 16 (33.3%) URAM			68k (77.4%) LUTs, 89k (50.9%) FFs, 220 (30.2%) DSP, 49 (38.3%) BRAM, 32 (66.7%) URAM		
Power [W]	6.71	6.83	6.83	7.21	7.84	7.94
Idle Power [W]	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14	0.07/6.14
Energy/Inference [J]	2.111	1.953	1.958	2.267	1.580	1.522
Power Efficiency [GOPS/W]	18.10	19.58	19.52	16.85	24.17	25.11

Table 12: Evaluation of YoloV4 Pruned on the SECO RUSSELL (SM-B71) using B2304 DPUs

Platform	**SM-B71 on SOM-DB2500**					
DPU	**B2304 x1, 200MHz**			**B2304 x1, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	134.03	235.00	469.70	94.85	156.96	313.32
Latency [ms]	145.47	248.05	482.81	106.24	169.88	326.44
Achieved performance [Inferences/s]	6.86	8.50	8.47	9.38	12.70	12.69
Achieved performance [GOPS]	262.05	324.7	323.55	358.32	485.14	484.76
Peak performance [GOPS]	460.8	460.8	460.8	691.2	691.2	691.2
Performance Ratio	56.87%	70.46%	70.21%	51.84%	70.19%	70.13%
Cost Metrics						
FPGA Resources	47k (53.7%) LUTs, 79k (45.0%) FFs, 422 (58.0%) DSP, 61 (47.7%) BRAM, 40 (83.3%) URAM					
Power [W]	8.02	8.57	8.55	9.77	10.90	10.89
Idle Power [W]	0.07/6.02	0.07/6.02	0.07/6.02	0.07/6.83	0.07/6.83	0.07/6.83
Energy/Inference [J]	1.169	1.008	1.009	1.042	0.858	0.858
Power Efficiency [GOPS/W]	32.67	37.89	37.84	36.68	44.51	44.51

Table 13: Evaluation of YoloV4 Pruned on the SECO RUSSELL (SM-B71) using B3136 and B4096 DPUs

Platform	**SM-B71 on SOM-DB2500**					
DPU	**B3136 x1, 300MHz**			**B4096 x1, 300MHz**		
Number of threads	1	2	4	1	2	4
Inference Time [ms]	76.76	121.32	242.01	63.31	94.67	188.47
Latency [ms]	88.06	134.19	255.12	21.53	107.49	201.76
Achieved performance [Inferences/s]	11.30	16.42	16.44	13.32	21.03	21.11
Achieved performance [GOPS]	431.66	627.24	628.01	508.82	803.35	806.40
Peak performance [GOPS]	940.8	940.8	940.8	1228.8	1228.8	1228.8
Performance Ratio	45.88%	66.67%	66.75%	41.41%	65.38%	65.63%
Cost Metrics						
FPGA Resources	51k (58.6%) LUTs, 90k (51.2%) FFs, 548 (75.3%) DSP, 71 (55.5%) BRAM, 44 (91.7%) URAM			57k (64.7%) LUTs, 108k (61.7%) FFs, 690 (94.8%) DSP, 81 (63.3%) BRAM, 48 (100.0%) URAM		
Power [W]	10.77	12.49	12.43	12.49	15.24	15.24
Idle Power [W]	0.07/7.09	0.07/7.09	0.07/7.09	0.07/7.56	0.07/7.56	0.07/7.56
Energy/Inference [J]	0.953	0.760	0.756	0.938	0.725	0.722
Power Efficiency [GOPS/W]	40.08	50.22	50.52	40.74	52.71	52.91

2.2.5 Comparison of Results

Except for the final softmax layers, the complete networks are run on the DPUs. The examples clearly indicate that the DPU configurations need to be selected based on the available hardware platform in combination with the application requirements, i.e., the complexity of the model that is used. For small models, such as MobileNetV2, a bigger DPU configuration does not increase the energy efficiency nor inference performance in comparison to a small or mid-range DPU. For computationally more complex models like YoloV4, the energy efficiency is best with larger DPU configurations. Despite the higher performance ratio on smaller DPUs, the overall shorter execution time with the B4096-DPU decreases the impact of static power dissipation and therefore increases power efficiency. The same holds true for different clock frequencies as shown with the B2304-DPU. The increased frequency leads to a shorter inference time and a higher efficiency, despite the higher power draw.

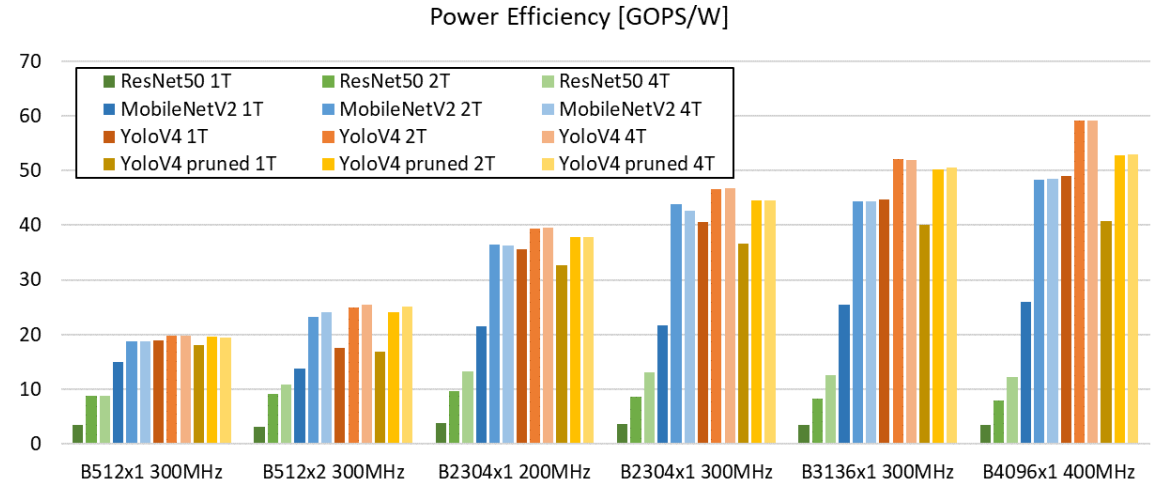

Figure 6: Power efficiency of the implementations for ResNet50, YoloV4 and MobileNetV2

Figure 6 summarizes the achieved power efficiency (in GOPS/Watt) for the various implementations. As mentioned above, efficiency increases for larger models. Extending the number of threads beyond two typically did not result in significant additional performance or efficiency gains. Figure 7 shows a different view on the results for YoloV4 and MobileNetV2, being the largest and smallest examples in our evaluation. Here, some of the results are skipped to increase readability. The individual labels for the data points are named by the DPU type followed by the clock frequency (only for the B2304 DPU) and by the number of threads. Including the pruned version of YoloV4 in this figure would result in very close values compared to the non-pruned version. Hence, for showing the benefits of the pruned version, Figure 8 illustrates the efficiency by combining the energy per inference with the achieved performance (in terms of inferences per second).

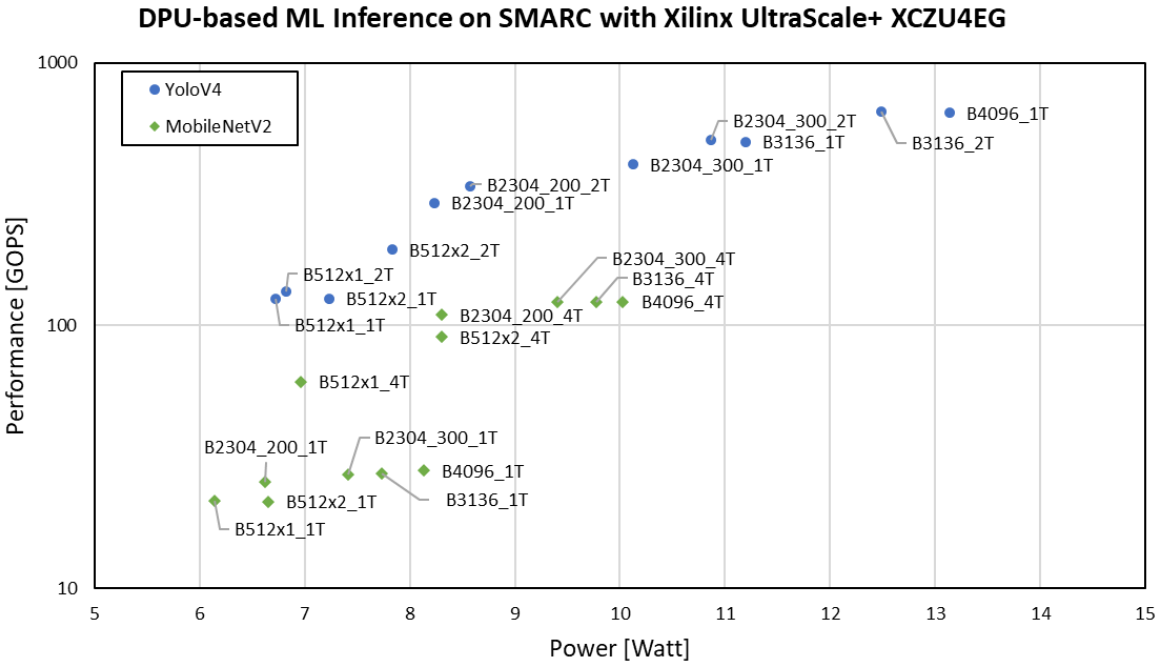

Figure 7: Comparison of performance and power for YoloV4 and MobileNetV2

The diagrams illustrate the design space that is achieved only by varying the DPU sizes. Varying additional parameters of the DPU, like optimization for low power will further

increase the number of possible solutions. Based on the application requirements, the best solution needs to be selected. In VEDLIoT, we want to target this challenge by utilizing the EmbeDL tool flow for optimizing the models toward the architectures. This will result in a first approach for model-hardware-co-design, that will be further developed with the architectures and methodologies discussed in the next chapter.

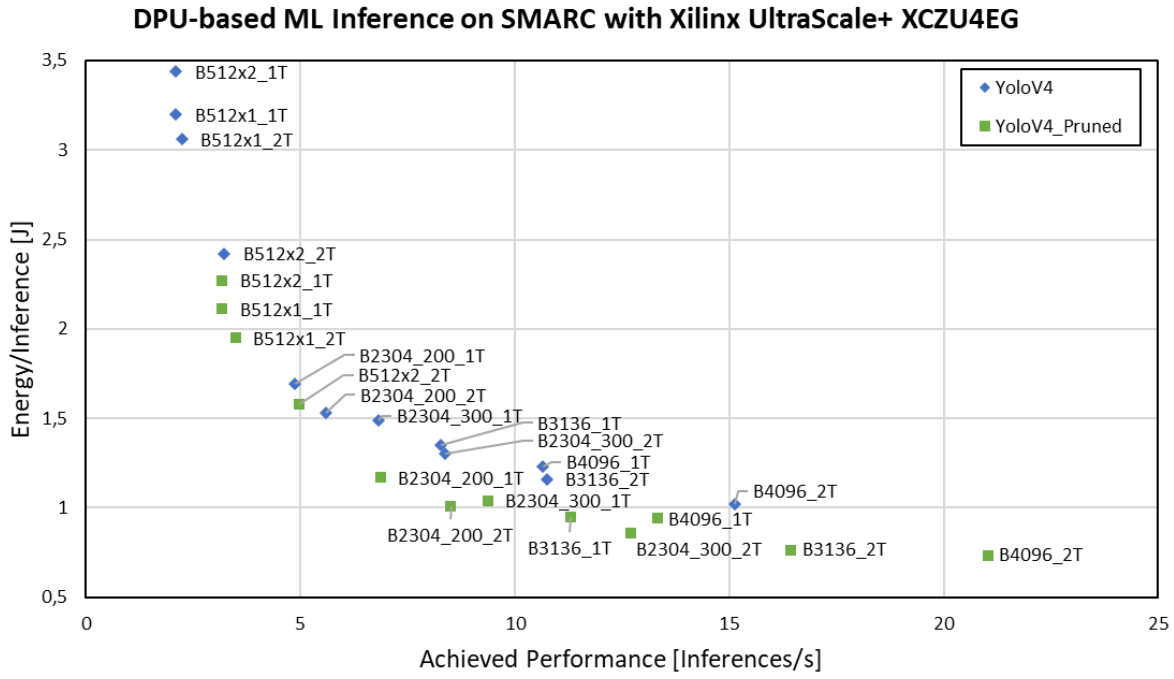

Figure 8: Comparison of energy efficiency (in terms of Energy per Inference) and achieved performance for YoloV4 and the pruned version of YoloV4

3 Reconfigurable DL Accelerators

Based on the evaluation of existing architectures (see D3.1 [1]), new accelerators for deep learning will be developed in VEDLIoT. This chapter summarizes the ideas and first results developed within Task 3.3. The focus of our implementations will be on efficient utilization of low-precision units, whenever possible. Implementations will be realized on FPGAs, making use of the specific features of the reconfigurable devices, including runtime reconfiguration. Targeting Xilinx FPGAs in the first place, the DPU implementations that are presented in Chapter 2.2 will serve as a basis for later comparison to the developments within VEDLIoT.

3.1 Evaluation of Open-source Solutions for ML Acceleration

A wide variety of IP cores and hardware accelerators for machine learning has been identified and compared in D3.1 [1]. For FPGA implementations, two general approaches can be distinguished. On the one hand, dedicated, typically configurable accelerator IP-cores are available, typically targeting a wide variety of ML models. In Chapter 3.1.1 we briefly discuss the ML accelerators that have been identified as being of high interest for utilization in VEDLIoT, focusing on open-source architectures. On the other hand, hardware generators are available, that can be used to generate an architecture optimized for a specific model. These will be discussed in Chapter 3.1.2.

3.1.1 Open-source ML Accelerators

In this chapter, we include a description of some open-source IP-cores targeting ML acceleration that we are considering for our project. The accelerators are NVDLA [10], VTA [11] and Gemmini [12]. An extended description of the IP-core accelerators can be found in D3.1 [1] (Chapters 7.61 and 7.6.2), D6.1 [13] (Chapters 4.6.1 and 4.6.2) and in the appendix of this deliverable. We are interested in the evaluation of such accelerators, not only as to compare with the other accelerators already evaluated in Task 3.1 (and results reported in D3.1 [1]) but also as potential starting points or baselines for the accelerators to be developed within Task 3.3 and Task 3.4.

The NVDLA package provides a simulation and an FPGA platform. The simulation platform is based on GreenSocs QBox [14]. The FPGA platform is used only for validation, not optimizations and it uses the Amazon EC2 “F1” environment [15]. The TVM-VTA package [16] contains a simulation infrastructure that supports fast simulation (sim) for testing compiler passes and also a cycle-accurate simulator (tsim [17]) that does RTL simulation with Verilator from TVM. VTA can also be ported to FPGAs. So far, it provides support for the PYNQ [18] and Ultra96 [19] development boards, equipped with Xilinx FPGAs. The Gemmini package includes a flow to generate RTL, which can then be used for simulation. Some simulators that can be used to simulate Gemmini are Spike [20], Verilator [21], MIDAS [22] and FireSim [23].

Table 14 summarizes the considered ML accelerators. All three IP-core accelerators mentioned above are open-source. VTA and Gemmini have an active community on github that keeps updating the IP-cores. On the other hand, NVDLA does not have an active community as the last update was in 2019.

Table 14: Overview of the open-source ML accelerators that can serve as a baseline for the accelerators developed in VEDLIoT

IP-core Accelerator	Environment for Simulation	Ported to FPGA	Year of last modification
NVDLA	NVDLA virtual simulator, FireSim-NVDLA	x*	2019
VTA	Fast Sim, TSIM	x	2022
GEMMINI	Spike, Verilator, MIDAS, FireSim**	-	2022

**: available only with cloud FPGAs on AWS*
***: FireSim runs on cloud FPGAs*

Considering all the information gathered so far, we plan on evaluating all three IP-cores, but we will start with VTA as it supports simulation and can also be ported on actual FPGAs which match the hardware that is available in VEDLIoT. The steps of the evaluations are to map the deep neural network (DNN) models used in the evaluation of the other accelerators (see D3.1 [1]), such as Resnet50, to VTA's simulator first and later deploy them on the actual hardware. We will try to collect the same metrics as done for the evaluation of the rest of the accelerators and use those results to guide us into exploring ways of accelerating the execution. More details on potential optimizations considered are discussed in Chapter 3.2.2.

3.1.2 Open-source Hardware Generators for Deep Neural Networks

Automatic generation of a hardware architecture based on a high-level description of the DNN model is a promising approach to efficiently utilize the high flexibility of FPGAs. Architectures can be optimized towards the specific requirements of a model and a target application, e.g., with respect to the required accuracy or performance/power trade-off, resulting, among others, in different quantization schemes. Various frameworks have been proposed, targeting customizable hardware accelerators, as discussed, e.g., in [24].

Among the different approaches, two have been identified in D3.1, being of special interest for the accelerator design in VEDLIoT: FINN [25] and DNNBuilder/AccDNN [26] (cf. [1]). FINN is an experimental framework from Xilinx research Labs targeting the generation of dataflow-architectures for quantized neural networks on FPGAs. It is supported by an active community on GitHub [27]. FINN uses a template-based approach with C++ descriptions of multiple layers and an HLS (high-level synthesis) backend that generates the required hardware. AccDNN (Accelerator Core Compiler for Deep Neural Network) has been developed in cooperation between the University of Illinois, Urbana-Champaign and IBM. The framework (which is called DNNBuilder in the corresponding research papers) automatically converts deep neural networks, trained using Caffe, to RTL code that can be synthesized for FPGAs without additional FPGA design effort. In its current version, AccDNN only supports models trained with the Caffee framework. Additionally, only convolutional layers, max-pooling layers, fully connected layers, and batch normalization layers can be used. The total number of convolutional and fully connected layers in the network should be less than 15. The tool is available on GitHub [28] with last updates in 2019. Since the accelerator designs in VEDLIoT should also serve as a basis for the co-design approach,

targeted WP3, we focus on a template-based approach like proposed in FINN, which is discussed in more detail in the next chapter.

3.2 DL-Accelerator Development in VEDLIoT

The accelerator development in VEDLIoT has just started. In the following, we summarize the ongoing work in Task 3.3 and provide an outlook for the subsequent Task 3.4, focusing on the co-design of models and architectures.

3.2.1 Template-based Accelerator Design

The accelerator developments in VEDLIoT target a highly flexible, modular approach that will be made available to the community as an open-source project. Focusing on reconfigurable architectures, the implementations shall be scalable from small low-power implementations to large FPGAs and even combinations of FPGAs. In this way, the accelerators can be utilized for the complete compute continuum – from far-edge applications (based on u.RECS) via near-edge (t.RECS) towards the cloud (in the RECS|Box). The accelerator shall be optimized for the DL models explored in the project's use cases but needs to be flexible enough to fit other models so that it can be utilized and optimized within the open-call cases.

As mentioned above, we want to achieve the required flexibility and scalability by a template-based approach. The templates will be primarily based on C++ code, enabling easy fast design cycles utilizing high-level synthesis, as well as options for hardware-software co-design and co-verification when combining compiled C++ code (running on the processor systems) and synthesized implementations (running in the FPGA fabric). For model components that need to be optimized, e.g., for minimum latency or maximum resource efficiency, the set of templates can be extended with optimized RTL building blocks, wherever necessary.

The authors of [24] identified various research questions that are well in line with the use case requirements and targeted accelerator designs in VEDLIoT. In addition to inference, we want to provide the possibility for FPGA-based training. Among others, this opens up new opportunities for the acceleration of federated learning and deep reinforcement learning, providing interesting use-case opportunities for the VEDLIoT open calls. Therefore, we not only target different integer quantizations within the C++ templates but also will support variable precision floating-point numbers.

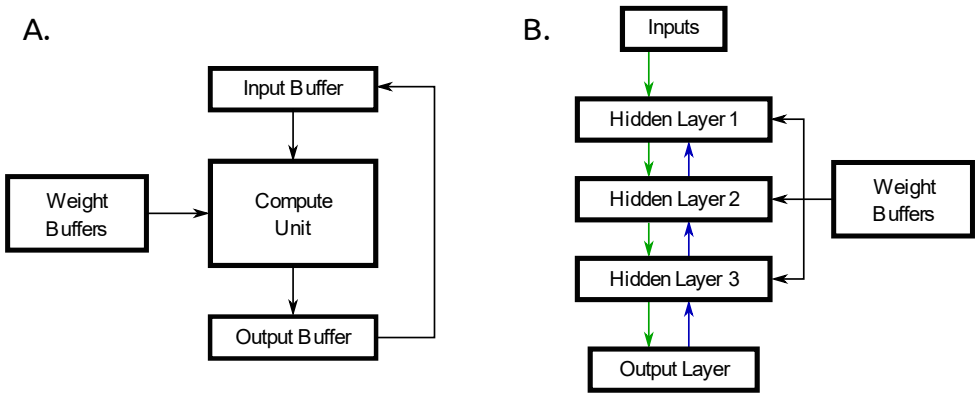

Figure 9: General architectures for the implementation of the neural network accelerators

When implementing neural networks in HLS, we can think of two different basic architectures, shown in Figure 9. Architecture A uses a single compute unit for all layers

which gets supplied with the appropriate weight matrices and inputs for each layer. Architecture B implements each layer as individual hardware modules that can be pipelined in a dataflow architecture. For inference, architecture B can be implemented quite efficiently. However, for training, it can be difficult to efficiently pipeline the different layer modules as the weights need to be accessed and updated multiple times in the pipeline.

As a starting point, first designs for the inference and training of networks of fully connected layers with architecture B have been implemented. The library consists of C++-templates that allow parameterizable neural network layers (e.g., number of neurons in a fully connected layer) and configurable HLS-implementation of these layers (e.g., by configuring pipeline depths or loop unrolling factors).

Next steps are adding more layer types, such as convolutional layers and pooling layers, and to enable configurability of the data types used for the computations. The current implementation uses floating point values for all computations. While this might be necessary for the training, inference will benefit from fixed-point values of configurable size. With additional layer types and fixed-point values, it will be possible to implement state-of-the-art CNNs, like YOLOv4, MobileNetV3, and Resnet50, with this neural network HLS library.

The accelerators will be integrated in the platforms developed in WP4. In addition to utilizing reconfigurable SoCs with embedded Arm cores, we plan to integrate the accelerators into the Soft SoC system, developed in Task 4.6 and 4.7. In this way, the developments can not only be used for FPGA implementations but can also serve as a blueprint for future ASIC implementations.

First Case Study – Accelerator for Deep Reinforcement Learning

In addition to inference acceleration for DNNs, deep reinforcement learning has been identified as a promising approach with high relevance for future applications on the one hand and high potential for hardware acceleration on the other hand. In reinforcement learning, an agent interacts with an environment by receiving an observation of the environment's state and choosing an action accordingly, as shown in Figure 10. The agent then receives rewards, and the goal of the reinforcement learning agent is to maximize these rewards over time. Research in reinforcement learning has progressed quickly within the last years, especially in deep reinforcement learning, where deep neural networks are used to learn the optimal policy. This led to a drastic increase in the required training time and hardware resources required for successful training. GPUs have often been used to speed-up the neural network training in the reinforcement learning algorithms. An interesting alternative that has started to be explored, are specialized accelerators on reconfigurable hardware.

Figure 10: The basic Reinforcement Learning setup. An agent interacts with an environment by observing the state and choosing an action

In [29] we provide a detailed overview of the current state-of-the-art of reinforcement learning accelerators. Specialized domain-specific architectures often surpass the efficiency of GPU-based approaches to deep reinforcement learning. In addition to the overview of existing architectures, the survey gives some directions for future research. While the existing work shows that specialized accelerators can be useful in reinforcement learning, much additional research needs to be done. Many of the state-of-the-art reinforcement learning algorithms have not yet been implemented, or at least not fully implemented, on reconfigurable hardware. For example, Deep Q-Learning implementations often employ extremely simplified versions of experience replay, a central feature of Deep Q-Learning.

A significant part of all Deep Reinforcement Learning algorithms is the training of (often multiple) neural networks. Hence, the template library will not only focus on DNN inference but will be enhanced with specific functions that are required especially for reinforcement learning. Based on the HLS library, state-of-the-art Deep Reinforcement Learning algorithms will be implemented, evaluated on standard reinforcement learning benchmarks, and applied to real world problems.

3.2.2 Co-designed Hardware Accelerators

As part of the tasks in this project we will be evaluating and developing different ML accelerators. After having evaluated different off-the-shelf accelerators, we have started to focus on the development of more dedicated accelerators and using FPGA-based design to explore different optimizations. One development path for these accelerators is by starting with the generic or model-agnostic accelerators as presented in Chapter 3.1.1. Next, we will explore the benefits of moving away from generic designs to mapping specific models into hardware. This is the work that is being performed within the context of the activities in Task 3.3. Our plan is to follow the model customized approach where, as the name suggests, an FPGA is reconfigured based on the targeted model. We follow this approach as it has been extensively used and shown to provide relatively good results, as it is tailored to the model-specific structure [30] [31] [32] [33]. We name this design the customized accelerator. This accelerator will be implemented in an FPGA and will be integrated into the hardware being developed in WP4. From the customized accelerator, we will work together with the model optimizations to further improve performance and efficiency by applying the principles of hardware-software co-design towards the accelerator to be developed in Task 3.4.

For the customized accelerator we will consider applying different optimizations. One of the optimizations that have been proven to reduce the area and energy requirement is quantization. Doing inference with INT8 has become almost a standard [34] [35]. This is because it could achieve almost the same inference accuracy of 32-bit floating-point inference using much fewer resources [35]. Our plan is to investigate using more aggressive quantization with bit widths smaller than 8. This could include using mixed precision weights either intra-layer or inter-layer of a combination of them.

Another optimization is to harness the sparsity of the DNNs. Different DNNs exhibit various degrees of sparsity both in the weights and activations [36]. This sparsity could potentially help to reduce the computation, storage, and bandwidth requirements leading to more efficient inference. However, while there are efficient techniques to exploit structured sparsity or high degrees of sparsity; dealing with unstructured sparsity is still a challenging area [37].

The design of the customized accelerator could be realized either using existing frameworks that map high-level DNN representation to an FPGA implementation or by implementing the

algorithm for the execution of the model using High-Level Synthesis (HLS). We plan on starting with the former approach and using FINN. FINN looks to be a good choice as it is aligned to our planned approach of designing model-specific implementations. It emphasizes on generating dataflow-style architectures customized for each network and has been commonly used in research as a design and evaluation tool [38] [25] [39].

The second approach could be realized using specialized High-Level Synthesis (HLS) implementations of one of SOTA designs [30] [31] [32] [33]. For achieving this goal, we want to make use of the library of the C++-templates described in Chapter 3.2.1. More specifically, for this approach we plan to apply the acceleration schemes described in [32]. This acceleration scheme targets the parallelism in both within the feature maps and across filters; it has been analytically shown to be the most promising [31] [32].

The analysis of the performance of the two approaches mentioned above, given the workloads, the use cases of the project, and the targeted hardware will lead us to identify which optimizations we could leverage (including the aforementioned aggressive quantization and sparsity). Consequently, we will be deriving the ongoing process of co-designing our components and having the next versions of our accelerator.

3.2.3 Dynamically Reconfigurable Accelerators

An important feature of FPGAs is their flexibility. For being able to compete with GPUs and TPUs in terms of performance and energy efficiency, the implemented architectures on the FPGAs need to be highly optimized. Using the template-based approach, discussed above, we target fine-grained architectural optimizations that enable us to build accelerators specifically tailored towards different applications or application requirements for the same model. One implementation can, e.g., offer maximum performance, another one can provide the best energy efficiency and a third requires minimum FPGA resources. Within VEDLIoT we do not only want to statically choose between the implementations at design-time but offer possibilities to adapt the architecture at runtime. This will be done by dynamic partial reconfiguration of the FPGA fabric.

First experiments utilizing partial dynamic reconfiguration will be based on the DPU implementations on the u.RECS that have been sketched in Chapter 2.2. We will add the possibility to dynamically switch between the different DPU implementations. Depending on its actual requirements or on environmental conditions, an application can select the best trade-off between performance and power or between performance and accuracy, among others. Resources that become available if a smaller DPU implementation is selected shall be made available for other accelerators, enabling, e.g., additional pre- or post-processing steps. The dynamic reconfiguration is closely linked to the design of the basic FPGA infrastructure in Task 4.4 and will be supported by the block-based design developed in WP4.

As soon as the first implementation based on our HLS templates are available, the flexibility of the approach will be further enhanced. For minimizing the reconfiguration times, we want to investigate the possibility to change parts of the accelerator at runtime. This approach will be closely coupled to the soft SoC system, developed in WP4, Tasks 4.6 and 4.7. We especially want to establish a tight integration of the accelerator with the reconfigurable RISC-V architectures that are developed in WP4. On system level, we will provide the possibility to exchange accelerators between FPGA devices with minimum overhead. Here, we especially want to investigate the possibility to relocate accelerators from one platform to another without re-implementing the designs. The results will be further used in WP6, targeting on-demand reconfiguration to increase efficiency and reliability in Task 6.4.

4 Conclusion

This deliverable provides an overview about the work performed in the Task 3.2 and the ongoing work in Task 3.3. A testbed for the u.RECS system that is developed within VEDLIoT has been realized and is now available to all partners for early evaluations. It serves as an important platform for the realization and evaluation of the basic FPGA infrastructure in Task 4.4. Additionally, it is used as a reference for the hardware development towards the u.RECS in Task 4.1 and can be used for evaluation of new compute and accelerator modules. The integration of FPGAs via SMARC modules enables the analysis of available and newly developed reconfigurable accelerators. As a baseline for the accelerator designs in VEDLIoT, various implementations based on the Xilinx DPU have been realized using the u.RECS testbed. The results are compared with respect to performance and efficiency, extending the previous benchmarking activities, summarized in D3.1 [1].

The design of reconfigurable accelerators in VEDLIoT uses a template-based approach, enabling flexible realization of accelerators. Like for the system architecture, we focus on a modular, scalable approach. The template-based design will be used for the envisioned co-design approach targeted in Task 3.4. As soon as the first designs are finished, further efficiency enhancements are planned by utilizing partial dynamic reconfiguration of the FPGAs. The accelerators will be integrated and evaluated on the RECS hardware platforms developed in WP4. Additionally, they will be combined with the soft SoC system, developed in Tasks 4.6 and 4.7. In WP6, the approach towards runtime reconfiguration of accelerators will be further utilized to increase efficiency and reliability.

5 Appendix

NVDLA

NVDLA is a free, standardized open architecture for accelerating deep learning inference. For a brief description of the IP, the reader is referred to D3.1 [1] or D6.1 [13].

NVDLA can be configured to balance performance, power and area. Some of the parameters that can be configured are:

- Data types: Supports binary, int4, int8, int16, int32, fp16, fp32, fp64
- Supported image memory formats: planar, semi-planar, other packed memory formats
- Convolution buffer size: The buffer is formed by banks, where the number of banks can be from 2 to 32 and the size of each bank from 4 KiB to 8 KiB.
- MAC array size: 2D array where the width can be from 8 to 64 and the depth from 4 to 64
- Activation engine size: Adjusts the number of activation outputs per cycle from 1 through 16.
- Pooling engine size: Adjusts the outputs per cycle between 1 and 4
- Memory interface bit width: It is adjusted based on the width of the external memory interface
- Memory read latency tolerance

VTA

VTA is an open source, parameterized accelerator for fast dense linear algebra operations. A brief description can be found in D6.1.

VTA can be configured by modifying the parameters in the configuration file. Some of those parameters are:

- TARGET: Supports “pynq”, “ultra96”, “sim” and “tsim”
- LOG_BATCH: Batch dimension of inner tensor computation
- LOG_BLOCK: Input/output channel dimensions of the inner tensor computation
- LOG_INP_BUFF_SIZE: Size in Bytes of input buffer
- LOG_WGT_BUFF_SIZE: Size in Bytes of weight buffer
- LOG_ACC_BUFF_SIZE: Size in Bytes of accumulator

The parameters with “LOG” on the name can only be powers of 2.

Gemmini

Gemmini is an open-source accelerator for DNN workloads. Gemmini's architecture consists of the following major components:

- Three controllers:
  - Execute Controller: Executes "execute"-type ISA commands
  - Load Controller: Responsible for instructions that move data from main memory
  - Store Controller: Responsible for instructions that move data to main memory
- Scratchpad: Where the inputs are stored
- Accumulator: Where partial sums and final results are stored
- Spatial array, composed of tiles that consist of arrays of PEs
- Two DMAs: One for moving data from main memory to SRAMs and one for moving data to main memory
- Re-Order Buffer (ROB): Detects hazards between instructions in different controllers

Gemmini is configurable. Some of the parameters that can be configured are:

- Spatial array
  - Dataflow: Weight and/or output stationary
  - Dimensions
  - Levels of pipelining
- Scratchpad and Accumulator
  - Capacity
  - Banks
  - Single/dual-port
- Global memory
  - Capacity
  - Banks
  - Optional L3
- Host CPU
  - In-order/out-of-order
  - ROB capacity
  - L1 capacity

Gemmini can be programmed on high level with ONNX models.

6 References

[1] VEDLIoT, "Deliverable D3.1 Evaluation of existing architectures and compilers for DL," 2021.

[2] VEDLIoT, "Deliverable D4.1 First report on cognitive IoT hardware platform and microserver development," 2022.

[3] VEDLIoT, "Deliverable D7.1 First report on testbed deployment and maintenance," 2021.

[4] NVIDIA, "NVIDIA Jetson Orin NX," [Online]. Available: https://developer.nvidia.com/embedded/jetson-orin-nx.

[5] Kontron, "SMARC Eval-Carrier-2 User Guide Rev. 1.9," 2021. [Online]. Available: https://www.kontron.com/products/boards-and-standard-form-factors/smarc/en/products/smarc-evaluation-carrier-2.0/p147568.

[6] Advantech, "SOM-DB2500 User Manual," 2021. [Online]. Available: https://www.advantech.eu/support/details/manual?id=1-24ZGYH3.

[7] Xilinx, "DPUCZDX8G for Zynq UltraScale+ MPSoCs - Product Guide PG338," 2021.

[8] Xilinx, "PetaLinux Tools," [Online]. Available: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[9] Xilinx, "Vitis-AI," [Online]. Available: https://github.com/Xilinx/Vitis-AI.

[10] NVIDIA, "NVIDIA Deep Learning Accelerator (NVDLA)," [Online]. Available: http://nvdla.org.

[11] Moreau, T. et al, "A hardware-software blueprint for flexible deep learning specialization," *IEEE Micro,* vol. 39, no. 5, pp. 8-16, 2019.

[12] H. Genc et al, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021.

[13] VEDLIoT, "Deliverable D6.1 Report on existing hardware and software interfaces for DL and compilers," 2021.

[14] G. Delbergue, M. Burton, F. Konrad, B. Le Gal and C. Jego, "QBox: an industrial solution for virtual platform simulation using QEMU and SystemC TLM-2.0," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[15] "Amazon EC2 F1-Instances," [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/.

[16] Apache Software Foundation, "About Apache TVM," [Online]. Available: https://tvm.apache.org/.

[17] "TSIM: Cycle Accurate Simulation for Custom HW in TVM," [Online]. Available: https://colab.research.google.com/github/uwsampl/tutorial/blob/master/notebook/05_TVM_Tutorial_TSIM.ipynb.

[18] Digilent, "PYNQ-Z1," [Online]. Available: https://digilent.com/reference/programmable-logic/pynq-z1/start.

[19] Linaro Limited, „Ultra96," [Online]. Available: https://www.96boards.org/product/ultra96/.

[20] "Spike RISC-V ISA Simulator," [Online]. Available: https://github.com/riscv-software-src/riscv-isa-sim.

[21] "Verilator," [Online]. Available: https://www.veripool.org/verilator/.

[22] "MIDAS/Strober v1.0," [Online]. Available: https://github.com/ucb-bar/midas-release.

[23] "FireSim: Easy-to-use, Scalable, FPGA-accelerated Cycle-accurate Hardware Simulation," [Online]. Available: https://github.com/firesim/firesim.

[24] S. I. Venieris, A. Kouris and C.-S. Bouganis, "Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions," *ACM Computing Surveys,* vol. 51, no. 3, 2019.

[25] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser and K. Vissers, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS),* vol. 11, no. 3, pp. 1-23, 2018.

[26] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu and D. Chen, "DNNBuilder: an Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

[27] „FINN - Dataflow compiler for QNN inference on FPGAs," [Online]. Available: https://xilinx.github.io/finn/.

[28] "AccDNN (Accelerator Core Compiler for Deep Neural Network)," [Online]. Available: https://github.com/IBM/AccDNN.

[29] M. Rothmann and M. Porrmann, "A Survey of Domain-Specific Architectures for Reinforcement Learning," *IEEE Access,* vol. 10, pp. 13753-13767, 2022.

[30] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016.

[31] Y. Ma, Y. Cao, S. Vrudhula and J. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 26, no. 7, pp. 1354-1367, 2018.

[32] Y. Ma, Y. Cao, S. Vrudhula and J. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Array*, 2017.

[33] Qiu, J. 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*, 2016.

[34] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[35] M. Sun, Z. Li, A. Lu, Y. Li, S.-E. Chang, X. Ma, X. Lin and Z. Fang, "FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization," 2022.

[36] O. M. Awad, M. Mahmoud, I. Edo, A. H. Zadeh, C. Bannon, A. Jayarajan, G. Pekhimenko and A. Moshovos, "FPRaker: A Processing Element For Accelerating Neural Network Training," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.

[37] Z. Wang, "SparseRT: Accelerating unstructured sparsity on GPUs for deep learning inference," *arXiv preprint arXiv:2008.11849,* 2020.

[38] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, California, USA, 2017.

[39] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn and M. Blott, "FINN-L: Library extensions and design trade-off analysis for variable precision LSTM

networks on FPGAs," in *2018 28th international conference on field programmable logic and applications (FPL)*, 2018.

[40] [Online]. Available: https://github.com/IBM/AccDNN.

[41] L. Gwennap, „Intel Gains Myriad Customers,“ *Microprocessor Report,* 2018.

7 List of Figures

8 List of Tables