The SELENE Deep Learning Acceleration Framework for Safety-related Applications

Laura Medina*, Salva Carrion*, Pablo Andreu*, Tomas Picornell*, Jose Flich*, Carles Hernàndez*, Michael Sandoval†, Markel Sainz†, Charles-Alexis Lefebvre†, Martin Rönnbäck‡, Martin Matschnig§, Matthias Wess§, Herbert Taucher§

* Universitat Politècnica de València (Spain) † Ikerlan Technology Research Centre (Spain)
‡ Cobham Gaisler (Sweeden) § Siemens Technology (Austria)

***Abstract*—The goal of the H2020 SELENE project is the development of a flexible computing platform for autonomous applications that includes built-in hardware support for safety. The SELENE computing platform is an open-source RISC-V heterogeneous multicore system-on-chip (SoC) that includes 6 NOEL-V RISC-V cores and artificial intelligence accelerators. In this paper, we describe the approach we have followed in the SELENE project to accelerate neural network inference processes. Our intermediate results show that both the FPGA and ASIC accelerators provide real-time inference performance for the analyzed network models at a resonable implementation cost.**

***Index Terms*—Neural Networks, real-time, Autonomy**

I. Introduction

Safety-related autonomous applications demand computing platforms able to achieve high computational power while meeting the stringent requirements imposed by functional safety standards. In particular, for autonomous systems, the efficient execution of neural network inference process is required. In this context, computing platforms for safety-related applications are evolving from multicore microcontroller units like the Infineon Aurix [1] to more complex heterogeneous multicore platforms like the NVIDIA Xavier [2].

The goal of the H2020 SELENE project is the development of a flexible computing platform for autonomous applications that includes built-in hardware support for safety. The SELENE computing platform is an open-source RISC-V heterogeneous multicore system-on-chip (SoC) that includes 6 NOEL-V RISC-V cores and artificial intelligence accelerators [3]. This paper describes the approach we have followed in the SELENE project to enable the deployment of use-cases requiring autonomous operation. In particular, two use-cases will be employed to validate the real-time inference capabilities of the platform: a robotic arm application from Airbus and an automatic train stop system developed by CAF signalling [4].

To achieve real-time neural network inference, the SELENE SoC incorporates specific hardware accelerators that are interconnected with the NOEL-V cores and memory devices using an AXI interconnect. To ease the utilization , integration, and programmability of the artificial intelligence accelerators we have developed the SELENE Acceleration Framework (SAF). The SAF consists of several building blocks: a deep learning library, an application programming interface (API) and the AI hardware accelerators. As deep learning library we have chosen the European Distributed Deep Learning Library (EDDL) [5] since it provides native support for FPGA acceleration which also eases HW/SW co-design. The API developed for Linux provides end-users with an easy way to offload computations from the cores to the accelerators. Finally, the neural network inference accelerators are described in C and synthesis tools are used to generate IPs targeting FPGAs, embedded FPGAs or ASIC technologies.

The rest of the paper is organized as follows. Section II describes the SELENE SoC and Section III introduces the SELENE acceleration Framework. The EDDL library adaptation to the SELENE platform is described in IV. The HLSinf FPGA accelerators is described in Section V and the HLS accelerator with ASIC target is described in Section VI. Finally, Section VII shows some preliminary integration results and conclusions are drawn in Section VIII.

II. The SELENE Computing Platform

The SELENE System-on-Chip (SoC) is depicted in Figure 1. It comprises a multi-core NOEL-V RISC-V system divided into General Purpose Processing (GPP) elements, L2 cache, an AI accelerator subsystem, memory controllers, and IO elements. The various building blocks of the SELENE SoC are interconnected by an AXI high-speed interconnect. The SELENE SoC is highly configurable and for the demonstration platform, a versatile six-core system has been chosen, to support the different use cases of the SELENE project.

The heart of SELENE SoC is Cobham Gaisler's 64-bit RISC-V processor, NOEL-V. The demonstrator design has six cores located in a single GPP element. The cores in the GPP element share a single L2 cache connected to the NoC to give access to the rest of the SoC.

For the intermediate SELENE demonstrator design, we decided to go with an approach with only one GPP element that comprises all NOEL-V cores. With a single GPP element, coherency between all cores is handled by the L1 caches in the GPP element. Furthermore, the IO subsystem is connected through an IOMMU directly to the internal bus of the GPP

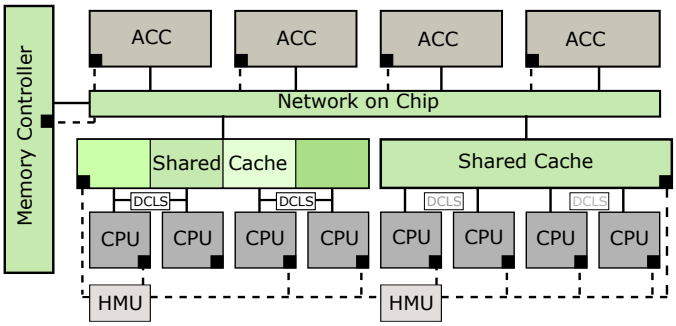

Fig. 1: Baseline SELENE SoC architecture.

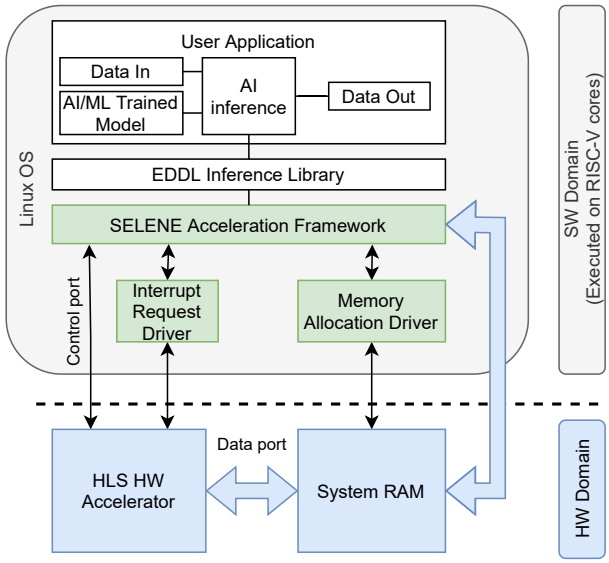

Fig. 2: Integration of the SELENE Acceleration Framework

element, thereby providing coherency between the IO peripherals and the processor cores. This structure could be changed later in the project.

For software development, the SELENE platform uses Linux as the default operating system. A Debian-based RISC-V Linux adaptation to SELENE can be found at https://github.com/siemens/isar-riscv/. This Linux distribution is the one we employ for the integration of the SELENE neural network toolchain.

III. The SELENE Acceleration Framework

The SELENE Acceleration Framework (SAF) is an Application Programming Interface (API) for interfacing hardware (HW) memory-mapped accelerators implemented in the SELENE System-on-Chip with the upper software (SW) layers, the inference library, and the user application. In particular, it focuses on High-Level Synthesis-based accelerators, and it ensures their interfacing under the Linux Operating System (OS).

Different frameworks providing enhanced support of specific features already exist [6], [7], [8], [9]. However, modifying and adapting an existing framework is a complex task and requires a significant development effort. Besides, many parts of these libraries would be unused in the project. We believe that capitalizing on a lightweight API tailored for the application is more valuable for the project, even if it reduces the number of features provided in the first version. Starting from scratch also brings the benefit of being independent of the provider.

The SAF provides the interface between the accelerators implemented in HW and the deep learning inference library EDDL used by the user application executing the AI/ML model, as shown in Figure 2.

The features offered in the first version are the following:

- Handling of the accelerators designed for the SELENE platform using HLS tools
- Controlling the memory and ensuring memory translation
- Handling the interrupts from the accelerators
- Easy configuration of the system
- Open-CL-like structure
- Designed to operate under Linux

To ease the interconnection of different accelerators modules, all the accelerators designed for the SELENE platform use AXI interfaces. These interfaces can be automatically generated with the different HLS tools. The SAF configures the accelerator through an AXI-lite slave port used as a control port for starting/stopping the accelerator. Through this port, we configure the interrupts, scalars, and pointers to memory. The control port is mapped directly to the processors using, for the time being, direct access to the physical memory (*/dev/mem*) through the SAF library.

With the SAF, results can be obtained by polling or interrupts. These interrupts are provided by an interrupt line that internally connects to the Platform-Level Interrupt Controller (PLIC) of the RISC-V cores. At the Linux level, the interrupt is handled in the kernel by the interrupt request driver developed within the SAF to notify the library the HW accelerator has ended its execution.

The memory exchanges between the accelerator and the processors are of a DMA type. Depending on the desired throughput, one or more AXI master memory-mapped ports can be used to access main memory. As the accelerator and the SoC share the same memory, the Linux OS must allocate the memory region used for each accelerator. The memory allocation driver is in charge of requesting physical memory to the kernel and ensuring memory translation. The physical address is used to configure the HW accelerator for DMA accesses, whereas the API uses the virtual address for input and output data.

For operation, the SAF provides a straightforward yet functional call sequence presented in Table I. The call sequence is not the same as OpenCL but tends to be similar.

A configuration file eases the acceleration initialization phase. It contains the memory base address of the accelerator, type of control signal, number and size of the data ports, etc. Then, using the memory allocation driver, memory buffers are allocated for input and processed output data. Once the input buffer is filled with data, the accelerator runs, either in polling

Sequence number	Function	Operation
1	SAF_Init()	Initialize the runtime
2	SAF_getKernel()	Initialize the accelerator
3	SAF_bufferAlloc()	Allocate memory buffers
4	SAF_setArg()	Configure the accelerator
5	SAF_bufferAddItem()	Fill the input buffer
6	SAF_runKernel()	Run the accelerator
7	SAF_bufferGetItem()	Read the output buffer
8	SAF_bufferFree()	Release the memory

TABLE I: API provided by the acceleration framework

or interrupt mode. Output data is available in the memory and accessible via the output buffer.

The introduced SAF is the first version containing the minimal API and features to handle the HW accelerators used in the SELENE project. We are aware of some limitations of the current version. For example, it does not consider reconfiguration at run-time of the accelerator. The accelerated function is static, and only a complete bitstream reconfiguration can change it. Though this is not a real drawback as the SELENE project mainly targets ASIC. Our next steps include improvements in the accelerator control (e.g., avoiding the use of */dev/mem*) and tests to check if the SAF adds overhead to the throughput of the accelerated system.

IV. EDDL Deployment in SELENE

The European Distributed Deep Learning Library (EDDL) is a general-purpose open-source deep-learning library with hardware transparency support for CPU, GPU, and FPGA.

Written in C++, and inspired by frameworks such as Keras or PyTorch, the EDDL provides the user with a user-friendly API with which to build the most common types of neural architectures (VGGs, ResNets, YOLOs, U-Nets, etc), and train them transparently via the computing service, on either a single machine or in a cluster of heterogeneous machines. In addition to this, the EDDL includes bindings for Python, an easy to read documentation, and dozens of examples to make the transition from other frameworks effortless.

The API of this library is built around the concepts of *Tensor* and *Neural Network Model*. The Tensor class serves two purposes: i) First, it provides the library with the capability to work with multi-dimensional arrays along with the high-level mathematical functions needed to operate on them. ii) It encapsulates the operations so that this class can support the most fundamental part of the Hardware Abstraction Layer (HAL). The Neural Network Model is the object that contains the neural layers defined by the end-user, along with its topology and serialization capabilities.

Having framework interoperability as a top priority, we added support for the Open Neural Network Exchange (ONNX) format, which defines a common set of operators and establishes a common file format to enable model exchange across other deep-learning frameworks, tools, runtimes, and compilers.

Concerning the deployment into the SELENE platform, the EDDL is integrated into the SELENE SW architecture having into consideration implications on safety, the SELENE platform HW acceleration features, and interoperability. To achieve this interoperability, we added support for the following operation modes:

- The end-user uses the EDDL and executes the model in an accelerator that supports ONNX (i.e. ONNX runtime)
- The end-user uses the EDDL and the accelerator is directly supported by the EDDL (i.e. HLS)
- The end-user uses any framework that supports exporting to the ONNX format and then, the EDDL imports this file and enables the communication with an accelerator that supports ONNX (e.i. Tensorflow $\rightarrow$ EDDL $\rightarrow$ HW accelerator)

V. A HLS Inference Accelerator for FPGAs

HLSinf is an accelerator for inference processes of neural network models based on convolutions. HLSinf is an open-source project designed in High-Level Synthesis [10].

A. Accelerator Architecture

The HLSinf accelerator is designed using the channel slicing concept, where a set of input channels are taken as an input in parallel and a set of output channels are produced in parallel. The channels per input define the input speedup and the channels per output define the output speedup. We can define these two parameters to implement different accelerator sizes and parallelism.

The overall design of the HLSinf accelerator is shown in the Figure 3. The accelerator is implemented around the dataflow model using modules interconnected by streams where each module performs one or more operations needed in neural networks. This module-based design allows us to pipeline convolutions with additional supported operations. As we can see, HLSinf currently supports several functions widely used in neural networks such as convolution, ReLU, leakyReLU, STM (an aggregation of softmax, hyperbolic tangent and element-wise multiplication), element-wise addition as well as pooling operations.

HLSinf also allows the implementation of specific designs since the accelerator can be customized in several dimensions. In the first dimension, the type of operations to support can be defined at design time. Therefore, specialized implementations can be prepared for specific models. Specifically, the convolution operation can be selected among three alternatives: direct convolution, winograd algorithm and depthwise separable convolutions. In a second dimension, the accelerator can be customized to specific data types and precision formats. The accelerator currently supports single-precision floating-point format (FP32), fixed-point formats and integer formats. Finally, in a third dimension, the input speedup and the output speedup can be defined.

B. HW/SW co-design

HLSinf has been designed to run in the EDDL library providing the needed support to run offloaded AI model layers

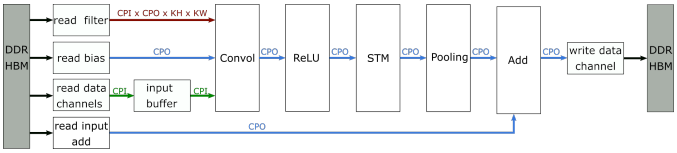

Fig. 3: HLSinf accelerator overall design.

on the FPGA. HLSinf can be used to configure and compile a given subset of network layers for its use in an inference process running with EDDL. HLSinf and EDDL allow a perfectly coupled HW/SW co-design approach where some parts of the model run in the FPGA whereas the rest run in the CPU or GPU when available.

To allow for an effective HW/SW codesign with the SELENE platform, we have implemented a new layer in EDDL called HLSinf. This layer encapsulates all the possible functionalities of our accelerator such as, for example, the operations to perform or the input channels. Then, the EDDL engine will run each model layer and each one will use the target device, either CPU/GPU for regular layers and FPGA for the HLSinf layer. Data transfers between the FPGA and the CPU/GPU memories are performed transparently when needed. However, in the SELENE SoC these data movement can be minimized since CPU and FPGA can be configured to share the same DDR.

In order to use our HLSinf accelerator, we have designed a new functionality in EDDL, which transforms an input model into a new adapted FPGA model with HLSinf layers where needed. Indeed, this new functionality enables three transformations of input regular models. First, layers in the original model are merged into a single one if the HLSinf accelerator is capable of performing all those layers to allow the pipelined operations. Second, data transformation is implemented when a layer runs on the CPU feeds a layer running on the FPGA, and vice-versa. For such purposes, a Transform layer has been implemented and these new layers are included while the model transformation is running. Finally, weights are adapted and tensors are reorganized as needed by the HLSinf accelerator. All tensors needed by the FPGA device are stored in the FPGA memory.

Figure 4 shows the transformation of the VGG16 [11] model as an example. This function generates a new FPGA adapted model taken from an input CPU model with regular layers. As we can see, this new adaption consists of, first, the generation of the new Transform layers. These new layers are placed between the FPGA layers (shown as HLSinf layers) and the regular CPU layers (e.g. AveragePool). Also, in this example, all the HLSinf layers groups two or three layers. With this new adapted model, all the layers placed between the lines will run on the FPGA. In contrast, the rest of the model will run on the CPU or GPU.

C. Automating the SoC Integration

In order to integrate AI accelerators, into the SELENE platform we have designed an integration tool. With this tool

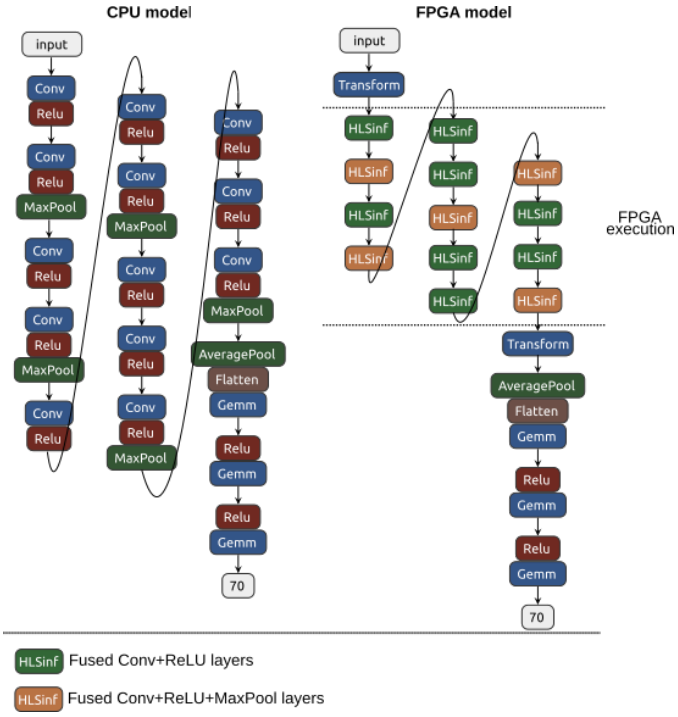

Fig. 4: VGG16 model transformation.

we are able to easily port the HLS generated AI accelerators to our SoC design. As different accelerator configurations may have different interconnection interfaces automating the process requires being able to wire accelerators with arbitrary number of ports and widths and type. The supported interface types are AXI and AXI lite.

The integration process consist of several steps. Initially, the target solution is generated with the specific FPGA vendor tools (e.g Vitis HLS) and exported as an IP. Then, the tool analyses the accelerator interfaces and generates a customized interconnection wrapper. Note that such interconnect is possible since the AXI interconnect inclused in the SELENE SoC can be fully parameterized. The final modules are automatically instantiated in the top design of the SELENE SoC.

D. Supported Network Models

The HLSinf accelerator is a general purpose neural network accelerator that is able to efficiently accelerate neural network models that are built using convolutional, relu, maxpool, averagepool and batch normalization layers. This includes state of the art network models such as the VGG16 [11], Yolo [12], SegNet [13], Stacked Hourglass [14], and HRNet [15] amongst others. However, the combination of the HLSinf and the EDDL support for CPU employed in the SELENE acceleration framework can be exploited to support a vast amount of network models. Future vector extensions of the NOEL-V cores can be also leveraged to improve the performance of network models that execute layers on the CPUs.

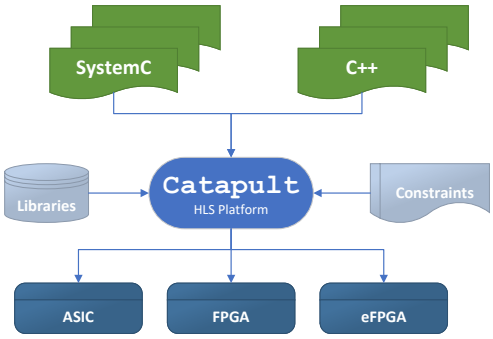

Fig. 5: Catapult HLS Workflow

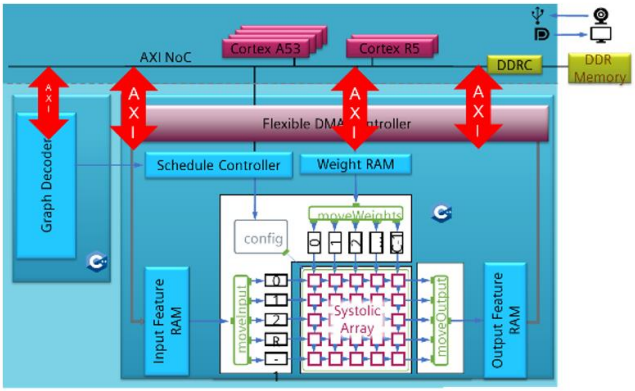

Fig. 6: SELENE HLS ASIC Accelerator architecture

VI. AI Acceleration for ASIC

One central expectation to the final SELENE platform is that it is finally independent of any specifics of individual target technologies. It is the ambition of the project to offer a flexible solution which can either be used on FPGA nodes as well as backbone for customized ASIC designs. Especially for the SELENE AI accelerators, which are developed using HLS techniques, this requirement implies that the HLS compiler in use can generate vendor-independent code to be further used without legal or technological restrictions. As an alternative to the widely used Vivado HLS compiler for Xilinx FPGAs the Mentor/Siemens Catapult [16] HLS tools is used for the synthesis of AI acceleration. Main motivation at this point is to pave the way towards a potential future ASIC based on the SELENE open-source platform.

As depicted in figure 5 Catapult HLS compiler can process C++ or SystemC input and gererate RTL suitable for different target technologies as ASIC, FPGA as well as eFPGA. As a performant alternative to the previously introduced general purpose accelerator (HLSinf) a more application specific variant and better suited for ASIC targets is available for the SELENE platform. Figure 6 gives an architectural overview of the accelerator module which is generated using the Catapult tool flow. Core component of this accelerator is a systolic array with int8 support for 2D convolution and fused maxpool operations. The module is connected to the main system bus via four bus interfaces, where one internal DMA controller orchestrates DDR traffic via three AXI master interfaces without loading the main CPU.

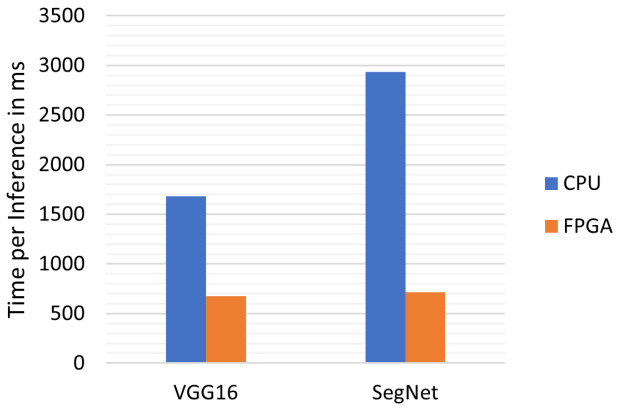

Fig. 7: A comparison of computational time required for the VGG16 and SegNet models between the Alveo U200 board and Intel Core i7-7800X CPU.

VII. Integration Results

This section shows performance results of the FPGA and ASIC accelerators when prototyped in FPGAs. Finally, we also show the HLSinf accelerator area oveheads when integrated in the SELENE SoC.

A. HLSinf Performance Results

Figure 7 compares the inference speed of the Alveo U200 board and Intel Core i7-7800X CPU, both presenting the same accuracy in each model. For this, we calculate the average inference time of one image in different models using the EDDL library. All the models present a single-precision floating-point format. For the CPU results, the entire model runs on the CPU. The CPU implementation creates 12 threads for the 6 cores of the CPU. On the other side, for the FPGA results, the supported layers run on the FPGA while the unsupported layers run on CPU but no parallelization is performed at the CPU computations. The models used for the results are, in the first place, the VGG16 convolutional neural network model [11]. For the second model, we have used the SegNet semantic segmentation model [13].

LUTs	Registers	BRAMs	DSPs
98603	95028	530	294

TABLE II: Vivado Synthesis Results ZCU102

B. HLS accelerator targeting ASIC

To date a prototype for Xilinx SoC FPGA is available which is currently being integrated into the SELENE platform. Within this environment a basic TinyYOLOv3 model was

TABLE III: Synthesis utilization estimates for several configurations on a Xilinx VCU118 evaluation board.

Device	Synthesis Utilization Estimates		
	DSP	FF	LUT
HLSinf_4x4_fp	336 (4%)	103378 (4%)	76885 (6%)
HLSinf_4x4_fp32	1000 (14%)	215404 (9%)	117833 (9%)
HLSinf_8x8_fp	960 (14%)	205694 (8%)	136624 (11%)
HLSinf_8x8_fp32	3440 (50%)	589422 (24%)	295845 (25%)
SoC_4_cores	72 (1.05%)	92055 (3.89%)	200921 (16.99%)
SoC_6_cores	108 (1.58%)	131181 (5.55%)	293153 (24.80%)

instantiated for validation and benchmarking purposes. The prototype implementation features a 16x16 systolic compute array at 225 MHz and achieves a theoretical peak performance of 115 Gops. Tab. II lists the reported resource consumption for the accelerator module on a Xilinx ZCU102 evaluation board after synthesis with Vivado. For the TinyYOLOv3 benchmark network the accelerator reaches a PE utilization of 51% leading to an execution time of the convolution and maxpool layers of 110 ms.

C. HLSinf Integration

The HLSinf accelerator has been already integrated into the SELENE SoC and prototyped in a VCU118 board. Table III shows the area overhead of the HLSinf accelerator and the area overheads of the SoC without considering the accelerators when integrated in the Xilinx VCU118 FPGA board. In particular, we show the area overheads of the SoC for 4-cores and 6-cores. These overheads consider the resources of the NOEL-V cores as well as the interconnect and the different peripherals. As shown in the table the HLSinf accelerator makes an intense use of DSP resources but it requires less LUTs than the SoC for both 4-core and 6-core configurations for the smaller configurations. We also observe that the resources required by the HLSinf accelerator fit well in the FPGA board we use as a prototype in SELENE.

Regarding accelerator configurations, we show results for 4×4 and 8×8 IO channels when using 32-bit floating point and fixed point data types. As we can observe, the 4×4 configuration with floating point uses 10% more DSPs and 5% more Flip-Flops when compared to the 4×4 fixed point solution. When incrementing the accelerator IO channels from 4×4 to 8×8, the used DSPs increases by $3.5\times$ in both fixed point and floating poing solutions. For floating point configurations, the Flip-Flops and LUTs have balanced utilization estimates.

VIII. Conclusions

In this paper we have introduced the SELENE open-source acceleration framework. In particular, we have described how existing neural network models can be deployed in our RISC-V SoC. We have described the architecture of the HLSinf FPGA accelerator and how HLS catatult tool allow us to move from FPGA acceleration targets to ASIC. Our preliminary results are promising and show that the performance provided by our accelerators is sufficient to achieve real-time inference performance for the neural networks required by the use-cases at a resonable hardware cost. The next steps will focus on the deployment of the final application use-cases to validate the performance, usability, and limitations of our framework.

References

[1] Infineon, *AURIX. 32-bit microcontrollers for automotive and industrial applications. Highly integrated and performance optimized*, 2019, https://www.infineon.com/.

[2] M. Ditty *et al.*, "Nvidia's xavier soc," in *Hotchips 2018*.

[3] H2020 SELENE consortium, "SELENE RISC-V open source hardware platform," https://gitlab.com/selene-riscv-platform, 2021.

[4] C. Hernández *et al.*, "SELENE: self-monitored dependable platform for high-performance safety-critical systems," in *23rd DSD 2020, Kranj, Slovenia, August 26-28, 2020*. IEEE, 2020, pp. 370–377. [Online]. Available: https://doi.org/10.1109/DSD51259.2020.00066

[5] H. D. project, "European distributed deep learning (eddl) library," 2020, https://github.com/deephealthproject/eddl.

[6] Xilinx, "Xilinx runtime (xrt)." [Online]. Available: https://xilinx.github.io/XRT/master/html/index.html

[7] E. Luebbers *et al.*, "Simplify Software Integration for FPGA Accelerators with OPAE (white paper)." [Online]. Available: https://01.org/sites/default/files/downloads/opae/open-programmable-acceleration-engine-paper.pdf

[8] C. Heinz *et al.*, "Improving Job Launch Rates in the TaPaSCo FPGA Middleware by Hardware/Software-Co-Design," *Proceedings of ROSS 2020: 10th International Workshop on ROSS, Held in conjunction with SC 2020: The International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 22–30, 2020.

[9] K. D. Pham *et al.*, "ZuCl: A ZynQ ultrascale+ framework for OpenCL HLS applications," in *5th International Workshop on FPGAs for Software Programmers, FSP 2018, co-located with International Conference on Field Programmable Logic and Applications, FPL 2018*, 2019, pp. 19–27.

[10] UPV, "Hlsinf neural network inference accelerator for fpgas," https://github.com/PEAK-UPV/HLSinf, 2021.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[12] A. Bochkovskiy *et al.*, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020.

[13] V. Badrinarayanan *et al.*, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE TPAMI*, vol. 39, no. 12, pp. 2481–2495, 2017.

[14] A. Newell *et al.*, "Stacked hourglass networks for human pose estimation," in *Computer Vision - ECCV 2016*, pp. 483–499.

[15] J. Wang *et al.*, "Deep high-resolution representation learning for visual recognition," 2020.

[16] "https://eda.sw.siemens.com/en-us/ic/ic-design/high-level-synthesis-and-verification-platform/."