ECSEL 783162

FitOptiVis

From the cloud to the edge - smart IntegraTion and OPtimisation Technologies for highly efficient Image and VIdeo processing Systems

Deliverable: D3.3 Design time, optimisation, deployment and programming strategies V3

Due date of deliverable: (30-09-2021)
Actual submission date: (30-09-2021)

Start date of Project: 01 June 2018 Duration: 36 + 6 months

Responsible: Jiri Kadlec kadlec@utia.cas.cz (UTIA)

Revision: Draft

Dissemination level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission Service	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (excluding the Commission Services)	

1. DOCUMENT INFO

Author

Author	Company	E-mail
Pekka Jääskeläinen	TUT	pekka.jaaskelainen@tuni.fi
Jiri Kadlec	UTIA	kadlec@utia.cas.cz
Pablo Sanchez	UC	sanchez@teisa.unican.es
Carlo Sau	UNICA	carlo.sau@diee.unica.it
Dip Goswami	TUE	D.Goswami@tue.nl
Roman Juranek	BUT	ijuranek@fit.vutbr.cz
Luis Medina	7SOLS	luis.medina@sevensols.com
Francesca Palumbo	UNISS	fpalumbo@uniss.it
Luigi Pomante	UNIVAQ	luigi.pomante@univaq.it
Tero Säntti	UTU	teansa@utu.fi
Massimo Massa	AITEK	mmassa@aitek.it
Tomas Bures	CUNI	bures@d3s.mff.cuni.cz
Jukka Saarinen	Nokia	jukka.saarinen@nokia.com
Jari Hannuksela	Visidion	jari.hannuksela@visidon.fi

Document history

Document version #	Date	Change
V0.1	14.6.2021	Starting version based on D3.2 – Jiri Kadlec, UTIA
V0.2	31.8.2021	Integrated partner input
V0.3	3.8.2021	Template for 3.1 Progress made by each partner.
		Template for 8.1 Main achievements by each partner.
V0.4	6.8.2021	Update of UTIA chapters, introduction and conclusion
V0.5	13.9.2021	Draft sent for internal review
V09	29.9.2021	Integrated modifications proposed by the internal review
V1.0	30.9.2021	Ready for release. In ch. 3.1 8.1, partner input.
V1.1	6.10.2021	Integrated additional partner input into chapters 3.1 8.1,
V1.2	10.10.2021	Ready for release to ECSEL JU portal.

Document data

Keywords	
Editor Address data	Name: Jiri Kadlec Partner: UTIA Pod vodarenskou vezi 4, Prague 8. Czech republic: Phone:+420 2 6605 2216

Distribution list

Date	Issue	E-mailer
10.10.2021	V1.2	fitoptivis-wp3@lists.utu.fi

Table of Contents

Table of Figures

Table of Tables

Table of Acronyms

AEx	AutoExplorer is a design space exploration flow
AXI4	Xilinx Interconnect standard
AXI4-Stream	Xilinx Interconnect standard serving for data streaming
AXI4-Lite	Xilinx Interconnect standard serving for access to registers
BMCA	Best Master Clock Algorithm
CfP	Call for Proposals
CNN	Convolutional Neural Networks
CompSOC	predictable multiprocessor system-on-chip platform
CSP	Convolution-Specific Processor
D-HMPS	Dedicate Heterogeneous/homogeneous Multi-Processing System
DMA	Direct Memory Access
DSE	Design Space Exploration
DSL	Domain Specific Language
DTRiMC	Design Time Resource Integrator of Model Composer IPs technology
ESL	Electronic System Level
FPGA	Field Programmable Gate Array
FPS	Frames per second
gPTP	generalized Precision Time Protocol
HiL	Hardware-in-the-Loop
HLS	High Level Synthesis (Xilinx C/C++ to HW IP compiler).
IBC	Image Based Control
IRF	Instruction Register File
J4CS	Joule for C statements
LLVM	Low-Level Virtual Machine compiler development infrastructure
LSU	Load-Store Unit
MDA	Model Driven Architecture
MDC	Multi-Dataflow Composer tool: a dataflow-to-accelerator design suite
MPL	Max Plus Linear graph (Serves for analysis of DSF graph)
NeuDNN	NEURAghe Deep Neural Network software stack
NEURAghe	Flexible and parameterized CNN accelerator
NFR	Non-Functional Requirements

OpenCL	Open standard defined by khronos group supported by multiple vendors
OpenCV	Open Computer Vision (C++ library of algorithms)
OpenMP	Open Multi-Processing
PCP	Priority Code Point
PIM, PDM, PSM	Platform Independent/Description/Specific Model
PSPs	Property Specification Patterns
QoC	Quality-of-Control
QRML	Quality and Resource Management Language
RIE	Runtime reconfiguration Implementation of Embedded systems
S3D	Single-Source System Design Framework
SAGE-VS	SAGE Verification Suite is set of SW tools aimed to accomplish different formal verification tasks at design time.
SDF	Synchronous Data Flow
SDSoC	Software Defined System in Chip (System level HW design flow).
SG-DMA	Scatter-Gather Direct Memory Access
SIMD	Single Instruction Multiple Data
SMP	Symmetric Multi-Processing
SoC	System on Chip
SPADe	Scenario and Platform Aware Design
TCE	TTA-based Co-Design Environment
TMO	Tonemapping
TSN	Time Sensitive Networks
TTA	Transport Triggered Architecture
UML-MARTE	UML Profile for Modelling and Analysis of Real-Time and Embedded Systems
VID	VLAN ID
VLIW	Very large Instruction Word
V-PCC	Video-based Point Cloud Compression
WC	Worst case
WCET	Worst Case Execution Time
YOLO	Open source content analysis software

2. Executive summary

Deliverable D3.3 "**Design time, optimization, deployment and programming strategies V3**" focuses on concrete design flows, tools and design time support packages used, developed and/or extended in the FitOptiVis project until the end of the project. The deliverable D3.3 is an update of the D3.2, with partner contributions and results delivered in the final, M25-M40 period of the FitOptiVis project.

Results developed in task T3.1 "**Model-driven engineering techniques for energy, performance and other qualities**" are described in Chapter 4.

Results developed in task T3.2 "**Programming and parallelization support**" are described in Chapter 5.

Results developed in task T3.3 "**Accelerator support**" are described in Chapter 6.

Results **spanning over all three tasks (T3.1, T3.2 and T3.3)** are described in Chapter 7. The design time technologies described in Chapter 7 have been released by FitOptiVis project WP3 partners in form of publicly accessible evaluation packages and publicly accessible application notes [7.15]-[7.20]. These resources served as WP3 design-time resource for FitOptiVis project partners, and also serve as publicly accessible design time support material, which can be used by other developers outside of the project.

Chapter 8 provides main conclusions from the work performed by WP3 partners in the FitOptiVis project.

Chapter 9 contains references.

Chapter 10 forms an appendix to the D3.3 deliverable. It summarizes all the developed tools and design technologies, highlighting their differences, granularities and use scenarios.

Each developed tool is presented in a compact table format similar to a sort of data sheet of the tool. The tool descriptions provide links to the publicly accessible repositories with application notes and released evaluation packages.

3. Introduction

This deliverable presents design methodologies, frameworks and design time support packages developed and/or improved by WP3 partners. The deliverable integrates results of WP3 after the third year of the project.

The work of WP3 is organised in 3 tasks: T3.1, T3.2 and T3.3, which contributions are provided in this deliverable as follows.

Task 3.1 – It deals with common approaches to the design time resources covering model-driven engineering techniques for energy, performance, and other qualities. These activities are described in **Chapter 4**. One of the core developments is the RIE methodology, which supports runtime reconfiguration of the software components described in the QRML modelling language developed in WP2. It is possible to generate RIE code from the WP2 QRML language and UML/MARTE models.

Task 3.2 – Contributions of partners related to task T3.2 are mainly included in **Chapter 5**. It describes the techniques that have been added to the design and programming tools developed in WP3 to improve their programming and parallelization support. Activities of partners in Task 3.2 also include links to the WP5.

Task 3.3 – Accelerator related contributions of partners are mainly included in **Chapter 6**. It describes design time resources related specifically to developing new HW accelerators. It contains the link to WP5 (Devices) via its new hardware accelerator designs supported by the WP3 design time development flows.

Chapter 7 is dedicated to design time methodologies and tools that have been developed in Tasks 3.1 and 3.3, and have been released in form of publicly accessible documented evaluation packages [7.15]-[7.20]. These released resources serve as concrete WP3 output results for FitOptiVis project partners. The tools released as open source also serve to other developers outside of the project.

Chapter 8 is highlighting main achievements achieved by projects partners cooperating in WP3.

Chapter 9 contains all references, including the www links to the developed and released evaluation packages and application notes.

Chapter 10 is an appendix of D3.3 deliverable contains the final overview and summary of all tools and design technologies developed, documented and released by WP3 partners. It also provides mapping of these results in a bigger picture related to their granularity and to the software/hardware orientation.

3.1. Progress of development made in FitOptiVis by WP3 partners

This section briefly highlights progress made by technology development partners in WP3 in the duration of the project.

3.1.1. UTIA

UTIA started its development in FitOptiVis project from board support packages for video processing systems based on Zynq 7000 devices (28 nm) developed in the ARTEMIS JU project No. 621439 ALMARVI (4.2014 to 6.2017).

In FitOptiVis (6.2018 – 11.2021), UTIA in collaboration with WP3 partners progressed substantially with the development and documentation of board support packages for a wide range of Zynq Ultrascale+ industrial grade modules (16 nm) manufactured by company Trenz Electronics, with support for Xilinx SDSoC system level compiler (versions 2017.4 and 2018.2) and also with integration of custom, run-time reprogrammable 8xSIMD floating point HW accelerators.

In Y1, Debian OS, ZynqUltrascale+ Full HD video platform with support for Xilinx SDSoC compiler was developed. See D3.1 and application notes and evaluation packages [7.1]-[7.4], [7.11] (released for public access in M12).

In Y2, fixed-HW, precompiled platforms with 8xSIMD HW accelerators were developed and documented in D3.2 by UTIA in collaboration with WP3 partners. The Design Time Resource Configurator (DTRC, see chapter 10.7) was released and documented. Released precompiled platforms support HW accelerated Full HD video processing and in addition the SW compilation of host firmware for 8xSIMD HW accelerators in gcc or g++ compiler projects. See application notes and evaluation packages [7.12]-[7.14] (released for public access in M24).

Finally, in Y3, UTIA in collaboration with WP3 partners developed, released and documented complete HW/SW flow for Zynq Ultrascale+ with possibility to integrate external SIMD HW accelerators, supported by the Design Time Resource Integrator of Model Composer IPs (DTRiMC, see chapter 10.8). It supports complete HW/SW design flow for video processing systems. It is open for user-defined modifications/extensions of the initial HW platform. This design time resource is described in this final deliverable D3.3 and related application notes and evaluation packages [7.15]-[7.20] (released for public access in M38).

3.1.2. BUT

BUT started with legacy technologies for Zynq platforms stemming from previous projects (EMC2, ALMARVI). In FitOptiVis , BUT continued on the development of the technologies in order to improve performance, configuration capabilities and resource consumption. Specifically, we improved IP core for processing of multi exposure video where we worked on high quality image merging. We completely redesigned the IP core for object detection. We dropped the legacy technology and implemented the algorithm with better speed and resource usage parameters. Finally, we implemented

a machine learning software, which can produce models for the object detection IP core.

3.1.3. TUT

TCE was developed substantially within FitOptiVis , with most of the developments shaped to a level that they were contributed to the open source OpenASIP branch. As highlights of new work in FitOptiVis :

- 64b integer and pointer support,
- Loop optimizations (initial software pipelining study) and
- Various improvements related to instruction memory density, the main pitfall of VLIW/TTA designs.

Furthermore, the soft core use case was improved with various FPGA-specific optimizations done to the SIMD and multicore support, as well as the automated processor generator tool AEx.

3.1.4. UCAN

The WP4 work has been focused into 4 main areas: the development of a C++ implementation methodology and support library (RIE) for reconfigurable systems, the automatic generation of implementation code, the integration of the FitOptiVis abstraction models in the S3D UML/MARTE framework and the analysis of the use of OpenMP for video systems. The methodology and library for the implementation of reconfigurable systems (RIE) is a pure FitOptiVis development that began from a basic specification and generates a framework that provides efficient reconfigurable implementations. The last year release is based on grpc services and support interfaces with reconfigurable implementations. In the other hand, a WP2 result (the SDSL language) has been used as input of a RIE-based automatic code generator. This generator has been used to generate the UC10 use-case components.

Additionally, during the last year UCAN has finalized the integration of SDSL in the S3D framework, in order to automatically generate UML/MARTE models from WP2 SDSL models. In WP3, UCAN has been also evaluating the use of OpenMP for programming heterogeneous systems that integrate HW accelerators or OpenCL-based devices. During the first part of the project, the effort was mainly focused on HW accelerators while the OpenCL-based system has received more attention in the last half. The final results is an OpenMP-based framework for heterogeneous system programming.

3.1.5. UNIVAQ

In the context of the ECSEL FitOptiVis project (WP3), UNIVAQ has finalized the extension of the HEPSYCODE methodology (with respect to the baseline available as a result of the ECSEL MEGAMART2/AQUAS projects) to consider also non-functional requirements related to energy consumption. Accordingly, UNIVAQ has also improved the set of prototypal SW tools supporting the methodology.

3.1.6. TUE

Image-based control (IBC) systems are increasingly being used in various domains including healthcare and autonomous driving. TUE dealt with efficient implementation of image-based control systems. The starting point was the basic analytical infrastructure developed in the earlier projects (oCPS, rCPS). TUE has extended the idea introducing scenario- and platform-awareness in the design flow – SPADe – of image-based control loops as well as software support for application development for the same. The idea was demonstrated considering time-predictable platforms. The idea was further adapted for modern industrial heterogeneous platforms, such as NVIDIA Drive. The SPADe flow is integrated into IMACS framework allowing for software-in-the-loop (SiL) and Hardware-in-the-loop (HiL) testing and bugging IBC systems.

3.1.7. UNISS

In this last reporting period UNISS has completed his work on MDC tool and an the SAGE suite. MDC extension for AIPHS has been completed, while, concerning SAGE, UNISS has finalized an SMT-based approach for automated consistency checking and inconsistency finding of configuration specifications.

3.1.8. HURJA

We have defined and implemented design-time optimization, deployment, and programming strategies related to Hurja's Salmi Care Platform in order to better utilize computing resources (CPU/GPU) of advanced AR-glasses (HoloLens 2) and smart-phone/tablet platforms. HURJA has also made UC3 integration by utilizing FIVIS tool to centralize the gathering of Salmi Care demonstrator's rehabilitation data in UC3.

3.1.9. UTU

In WP3 UTU developed accelerators for the Aura line of microprocessors. The accelerators were developed fully from scratch, and the end results are in the form of VHDL code that can be implemented in FPGA or in ASIC. Full synthesis to both targets has been done. The main task to be accelerated in the FitOptiVis target domain was convolutional image processing. These algorithms are also usefull in AI applications, not only image/video processing. The accelerator was designed in two variants, one using normal arithmetics, and on using reduced precision arithmetics. The motivation for reduced precision was minimizing memory accesses and thus power consumption.

3.1.10. NOKIA

In Y1 and Y2, A significant amount of new technical know-how was developed during the implementation of video-based point cloud coding technology (V-PCC) with WP3 tools and developments, in particular on how to synchronise several video streams on low complexity, low reliability devices, such as Android mobile phones. We took the

opportunity to communicate this new knowledge to related industry and standardisation bodies to create awareness for new challenges.

In Y3, the Real-time Point Cloud Augmented Reality Rendering Case Study has been implemented with TUT. We studied a full application task offloading case study, a smartphone application that renders a streamed animated point cloud in augmented reality (AR). The point cloud is received as an HEVC-encoded V-PCC stream which is decompressed using the mobile device's hardware decoder and reconstructed using OpenGL shaders. We utilized a scalable low-latency distributed heterogeneous computing PoCL-R which is based on the standard OpenCL API's features. We also proposed an API extension that significantly improves buffer transfer times for cases with varying data sizes. The unique latency and scalability enhancing features were tested with a distributed real-time augmented reality case study which reached 19x improvement in FPS and 17x in EPF by remote of offloading a rendering quality enhancement kernel using the runtime. The remote kernel execution latency overhead of the runtime was only 60 microseconds on top of the network roundtrip time. This demonstrator's implementation was based on the usage of WP3 design tools.

3.1.11. CUNI

In the scope of WP3, CUNI has been mainly active in integrating QRML and FIVIS with design time optimization. In Y1, CUNI's main activity in WP3 was to integrate QRML with activities in WP3 and to support partners in WP3 in adopting QRML and in developing QRML models for their components. In Y2, we turned our focus mainly to developing FIVIS platform to support partners in WP3. This involved mainly integration of QRML with FIVIS. In Y3, our work on FIVIS continued and we focused mainly supporting partners by FIVIS-based visualizations aligned with QRML. Here, FIVIS supports design time evolution by allowing at design-time to drill into and compare data measured at runtime on different (previous) versions of a component.

3.1.12. TASE

TASE's work in WP3 has been mainly related with two different tasks. One of them is the development of WP5 components using VITIS. This tool has allowed the developers to perform efficient and fast implementations of hardware-based components needed for the UC10 demonstrator on the remote component based on a Zynq UltraScale+. The components were started from scratch and VITIS allowed a fast development of these components starting from OpenCV software implementations. The other main task of TASE in WP3 has been providing support to UCAN during the integrations of their developments into the Space Use Case.

4. Model-driven engineering techniques for energy, performance and other qualities

This section presents design and verification frameworks as well as techniques that WP3 partners have developed during the second year. The first four sections present design and verification frameworks while the other sections present specific techniques. Activities of all partners in these areas also form an initial WP3 link to WP2 (component models, abstractions, virtualization and methods).

The project team included the set of elaborated model-driven engineering techniques in D3.2 used by partners as design time resource. Table 1 describes why they have been chosen to be FitOptiVis model-driven techniques and what kind of features exist in each of them, how they differ and complement each other.

Model-driven engineering technique	**Chapter**	**Why chosen to be one of FitOptiVis model-driven techniques for the design time resource**	**Specific features**
FitOptiVis S3D Modelling Framework	4.1	Efficiently models real-time video processing systems with runtime re-configuration capabilities.	FitOptiVis S3D framework includes eclipse-based Papyrus modelling and requirement capture framework and automatic generation of SW and verification code.
Design Space Exploration for Re-configurability	4.2	Model-driven Design Space Exploration HW/SW co-design methodology. The goal is to identify suitable “reconfiguration plans” for different trade-offs	Set of prototypal SW tools to support the methodology. Algorithm implementations providing results with different accuracy (approximate computing techniques)
SAGE Verification Suite	4.3	Automated Consistency checking and Inconsistency finding of requirements Organization and storage of requirements in an online platform. Automatic synthesis for goal oriented "correct-by-construction" policies from a system model and an objective. Automatic test generation for black-box reactive systems starting from requirements formalized in a logical language.	SpecPro: library translating requirements from natural language to logical language. ReqV: tool for requirements management and consistency formal verification. HyDRA: a tool for synthesizing an optimal and “correct-by-construction” policy given a model and tasks in logical language. ReqT: a tool for requirements-based test suites generation.
Dynamic performance tracking (control theoretic	4.4	Depending on the application requirements, the optimization algorithms find the best configuration (mapping, scheduling and	It is based on Synchronous Dataflow (SDF) graph which can be analysed to answer performance related questions such as the minimum

approaches)		voltage/frequency setting).	guaranteed throughput for a given mapping to a platform.
Modelling with limited precision	4.5	This approach allows using a reasonable dynamic range while limiting the data-path width, and thus energy consumption.	Use of non-linear number space.
Support for High Level Tool Chains	4.6	Design time development methodology for fast modelling and development of algorithms in C/C++ code executable on ARM with real video I/O. Performance of the HW accelerator can be estimated from these C/C++ models without complete compilation to the HW	Compatibility with Xilinx High Level Synthesis design Flow (Vivado HLS) and Xilinx SDSoC system level compiler. It compiles user defined C/C++ from ARM to the programmable Logic of the Zynq device. Xilinx SDSoC requires board support packages provided by FitOptiVis WP3 partners.
High-level abstract component model and DSL	4.7	The specified High-level abstract component model and the specified domain specific language (DSL) serve as conceptual link of work performed in the WP2 and in the WP3.	From the perspective of WP3, the component model provides the structure (component architecture). Components are hierarchically composable (support for abstracting composition of components as another component).

Table 1: Overview and Comparison of Model-driven engineering techniques.

4.1. The S3D modelling methodology for real-time video processing systems

The Single-Source System Design Framework, S3D [4.1], follows a component-oriented approach and applies Model Driven Architecture (MDA) principles in the development of HW/SW embedded systems to deal with the increasing complexity of software development. It considers application components as units that can be allocated either on the software part or on the hardware part of the system. S3D has been developed by UC in several projects [4.2] and the main objective of the S3D development in FitOptiVis is to adapt and improve the capacity of the methodology to efficiently model real-time video processing systems with runtime re-configuration capabilities. Additionally, the capability of the methodology to capture non-functional requirements will also be evaluated and improved. S3D uses the UML/MARTE standard and its main goal is to minimize the modeling effort as much as possible. In order to facilitate capturing all the relevant information about the system for different purposes in a coherent, accessible and compressive way, the information is organized in views.

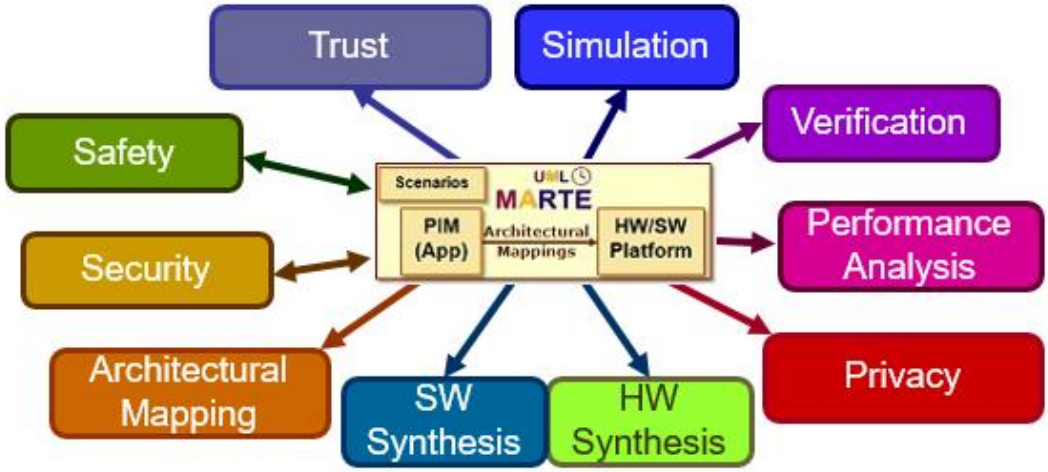


Figure 1: TE Single Source System Design Framework (SD3).

Each view encloses all the required information about a particular aspect of the system. The S3D ecosystem that is presented in Figure 1 includes different tools that perform different design tasks such as verification, simulation, performance analysis, scheduling analysis, etc. When the design satisfies all the functional and non-functional constraints, the code to be deployed on the different computational nodes of the distributed platform is automatically generated. The FitOptiVis S3D framework includes several design and verification tools such as an eclipse-based (Papyrus) modelling and requirement capture framework and automatic generation of SW and verification code.

The proposed approach uses three global models: PIM (Platform Independent Model), PDM (Platform Description Model) and PSM (Platform Specific Model). The PIM specifies the application structure (system components and their relation), behaviour and requirements. The PDM defines the structure and main performances of the physical HW/SW platform, in which the application will be implemented. The PSM model defines the allocation of the application components in the platform HW/SW resources.

The main view of the PIM is the Application View. This view defines the application components and their relations. An example of Application View is presented in Figure 2

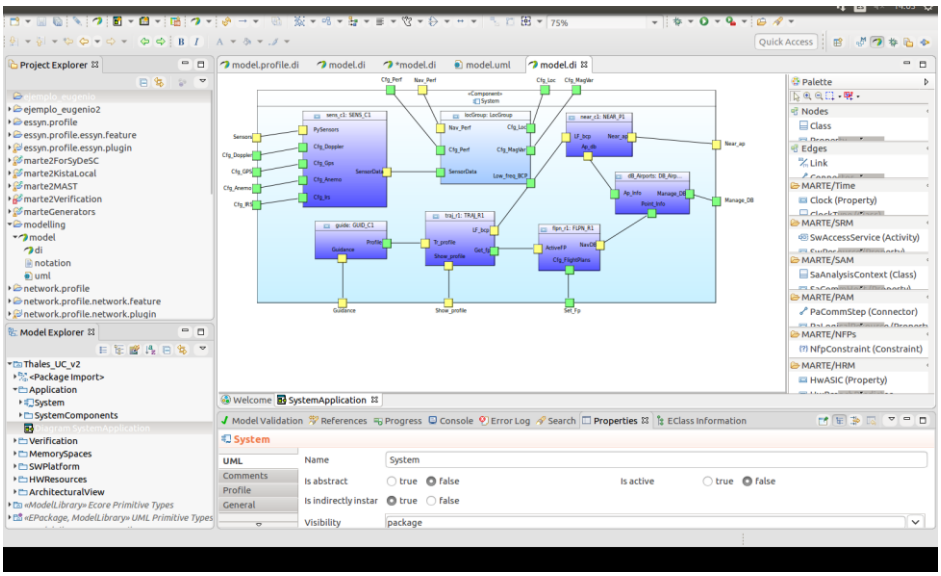


Figure 2: Application view.

The Application View uses generic components. In MARTE, these elements model real time units (concurrent elements) or passive component (non-concurrent elements). The external view of the component includes the services (functions) that they provide and/or require. Thus, the required interface of a component lists all the services that the component requires from other components. The provided interface lists all the services that the component offers to other components. Figure 3 shows an example of component that presents all the interface services. Every component has at least an implementation (or behaviour) and a specific verification test case.

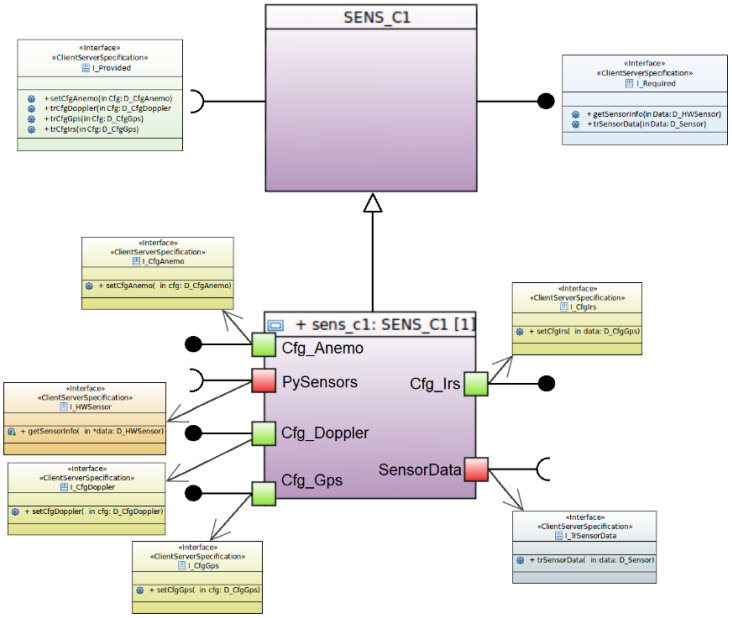


Figure 3: Example of Component interfaces.

In FitOptiVis , the S3D framework has been extended with new generators and modelling capabilities. The generator produces UML/MARTE models from QRML descriptions. The WP2 QRML models are mainly oriented to dataflow description and

this limits the UML/MARTE supports. In order to extent the modelling capabilities to services, a specialization of QRML (SDSL) was propose in WP2. The new language allows modelling software applications with service-oriented architecture and facilities UML/MARTE model generation. SDSL includes all the QRML elements, and, therefore, the generator supports all the QRML features related to application modelling.

During the last year (Y3), the UML/MARTE generator has been integrated in S3D and extended to support the last versions of SDSL and QRML. Next figure shows the structure of the developed generators. It integrates the previously commented UML/MARTE model generator and the C++ code producer. The C++ code generator produces runtime reconfigurable implementations. These elements will be commented in section 5.8.

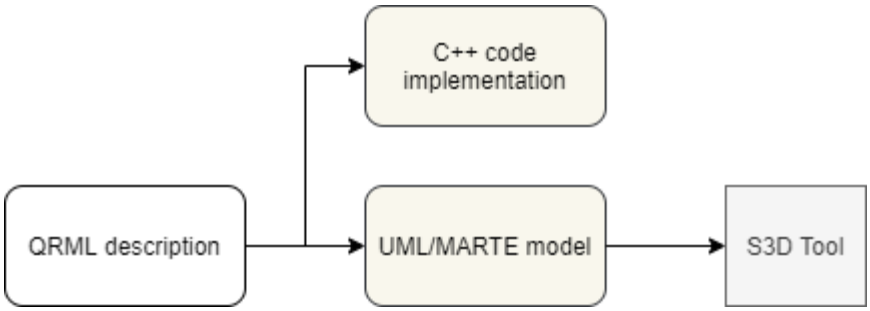


Figure 4: S3D design flow.

4.2. Design space exploration for re-configurability

In Y3, UNIVAQ has finalized the extension of the HEPSYCODE methodology to consider non-functional requirements (NFR) related to energy consumption, by exploiting a high-level (i.e., statement-level) energy performance metric (i.e., J4CS, described in D3.2) able to provide information about energy consumption of an embedded system and so useful for energy consumption estimation approaches. This metric is used inside the HEPSYCODE model-driven ESL HW/SW co-design methodology for the design of run-time reconfigurable heterogeneous parallel dedicated systems. Accordingly, UNIVAQ has also improved the set of prototypal SW tools supporting the methodology. Figure 4 shows the reference HEPSYCODE ESL HW/SW co-design flow more in details.

The HEPSYCODE goal is to identify (at design-time and, in the future, also at run-time) suitable “configurations” for different trade-offs (e.g., timing vs energy/power vs accuracy) by considering a heterogeneous set of HW components with multiple working points.

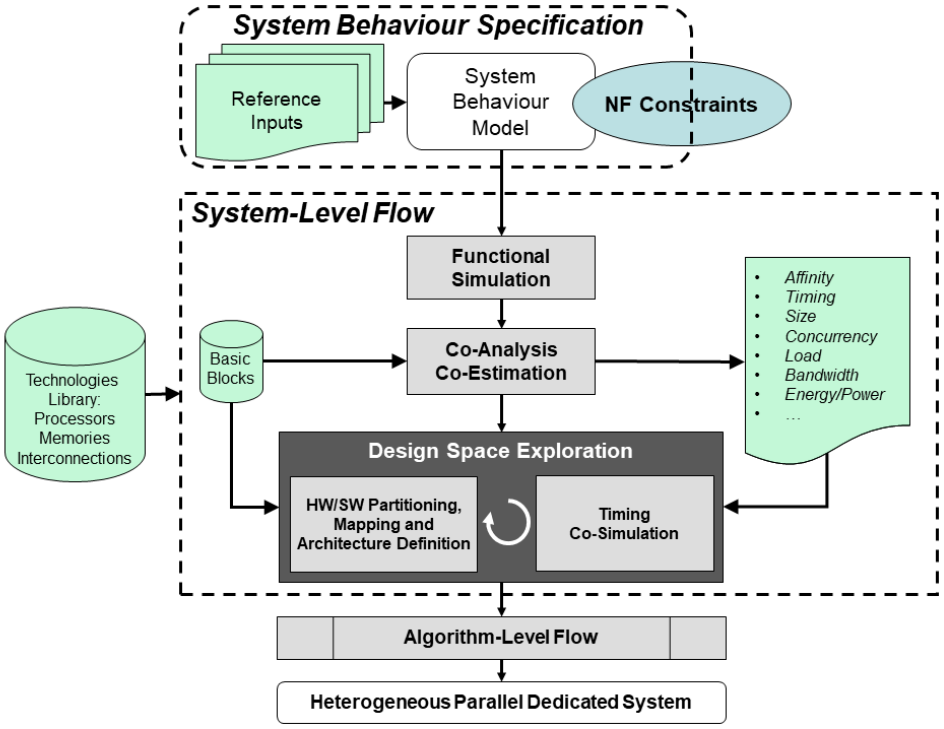


Figure 4: HEPSYCODE ESL HW/SW co-design flow.

4.2.1. Energy-aware HEPSYCODE

As stated in the previous paragraph, UNIVAQ has extended the HEPSYCODE methodology ([4.29], [4.30], [4.31], [4.32]) to consider energy-aware requirements, metric, and cost function in the design space exploration step. User energy requirements can be related to the possibility to find system implementations based on a *dedicated heterogeneous/homogeneous multi-processing system* (D-HMPS) that consumes as less energy as possible, or D-HMPSs that consumes less energy than a given energy threshold, while considering also other NFR (e.g., timing, cost, etc.). The considered energy metric is the J4CS, while the design space exploration analyzes alternative solutions by means of an evolutionary algorithm that considers, at the same time, with a weighted sum method, several objectives.

So, taking into account different processor technologies, HEPSYCODE is able to find a HW/SW partitioning, to define a HW architecture and to suggest a mapping potentially able to satisfy all the requirements.

Finally, HEPSYCODE is able to estimate timing performances and energy consumption by means of a SystemC simulator that considers the results found by the evolutionary algorithm.

Respect to the plan exposed in D3.2, UNIVAQ has applied the extended (i.e., energy-aware) methodology to two use cases, described below, to show the possible exploitation of the improved HEPSYCODE:

- FIR-FIR-GCD [4.31] [4.32]: a synthetic example;
- Digital Camera [4.33]: an example in the image processing domain (i.e., meaningful also for the Italian UC).

4.2.2. Use case #1: FIR-FIR-GCD

In order to show the main features of the extended methodology and to verify the related protype tools, a reference example is reported in the following. Let be the system behavior, represented by the CSP shown in Figure 5, composed of 8 processes and 12 internal channels.

Figure 6 provides a graphical representation of the main SC_MODULE representing the *System* (with internal processes and channels), while Figure 7 and Figure 8 show some parts of the correspondent SystemC description.

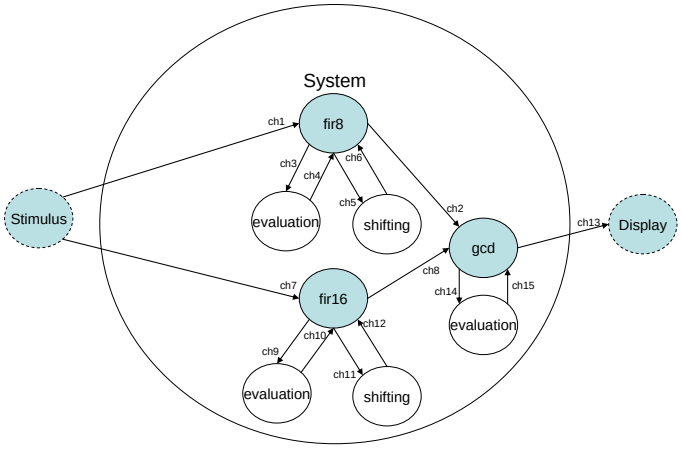


Figure 5: CSP representing the system behaviour.

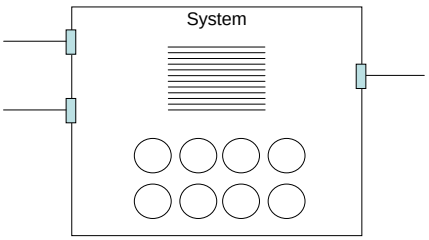


Figure 6: SC_MODULE representing the System module.

```
SC_MODULE(mainsystem)
{
      // Ports for testbench connections
      sc_port< sc_csp_channel_in_if< sc_uint<8> > >
            stim1_channel_port;
      sc_port< sc_csp_channel_in_if< sc_uint<8> > >
            stim2_channel_port;
      sc_port< sc_csp_channel_out_if< sc_uint<8> > >
            result_channel_port;

      // PROCESSES
      void fir8_main();
      void fir8_evaluation();
      void fir8_shifting();
      …

      // CHANNELS
      // fir8
      sc_csp_channel< fir8e_parameters >
            *fir8e_parameters_channel;
      sc_csp_channel< fir8e_results >
            *fir8e_results_channel;
      ...

      SC_CTOR(mainsystem)
      {
            SC_THREAD(fir8_main);
            SC_THREAD(fir8_evaluation);
            SC_THREAD(fir8_shifting);
            ...
```

Figure 7: Sketch of the System SC_MODULE SystemC description.

```
//f8s
void mainsystem::fir8_shifting()
{
      // datatype for channels
      fir8s_parameters fir8s_p;
      fir8s_results fir8s_r;
      // local variables
      sc_uint<8> sample_tmp;
      sc_uint<8> shift[8];

      while(1)
      {
            // read parameters from channel
            I(f8s) fir8s_p=fir8s_parameters_channel->read();

            // fill local variables
            sample_tmp=fir8s_p.sample_tmp;
            for( unsigned j=0; j<TAP8; j++)
                  shift[j]=fir8s_p.shift[j];

            // processing
            I(f8s)
            for(int i=TAP8-2; i>=0; i--)
            {I(f8s)
                  I(f8s) shift[i+1] = shift[i];
            }
            I(f8s) shift[0]=sample_tmp;

            // fill datatype
            for( unsigned j=0; j<TAP8; j++)
                  fir8s_p.shift[j]=shift[j];

            // send results by channel
            I(f8s) fir8s_results_channel->write(fir8s_r);

            P(f8s)
      }
}
```

Figure 8: Sketch of a CSP process SystemC description.

It is worth noting that the considered example, called FIR-FIR-GCD, doesn't perform a meaningful computation, but it is just used as a simple case study (i.e., it is a synthetic example).

The *Technology Library* considered for this case study is composed of three different processors: an Intel MPU8051 (12 MHz, GPP), a Gaisler LEON3 (75 MHz, GPP) and a Xilinx Spartan3AN (50 MHz, SPP). TL contains all the relevant information about processors, memories (local to processors) and interconnections among processors (in this case study they are limited to a single shared bus, i.e., I2C) needed to perform the DSE step.

The *Functional Simulation* allows checking correctness of the system behavior by analyzing outputs obtained by providing reference inputs.

Then, in this example, the *Co-Analysis* activity has been performed (partially manually) by the designer. Based on his experience, he has provided values for the *Affinity* (Figure 9), while the *Concurrency* has been estimated by means of the HEPSIM simulator [4.34].

```
f8m      =      {0.9, 0.7, 0.5}
f8e      =      {0.5, 0.7, 0.5}
f8s      =      {0.5, 0.8, 0.9}
f16m     =      {0.9, 0.7, 0.5}
f16e     =      {0.5, 0.7, 0.7}
f16s     =      {0.5, 0.8, 0.9}
gcdm     =      {0.9, 0.7, 0.5}
gcde     =      {0.5, 0.7, 0.7}
```

Figure 9: Affinity with respect to GPP, DSP, and SPP.

The *Co-Estimation* activity has been performed by exploiting CC4CS [4.35] and J4CS (see D3.2) metrics. The results about *Timing* are then several min-max pairs (one for each processor) related to the number of clock cycles needed to execute the statements composing the SystemC descriptions of each process. The results about *Energy* are also several min-max pairs (one for each processor) related to the energy (Joule) needed to execute the statements composing the SystemC descriptions of each process. The precise values to be used during timing/energy co-simulation are dependent on process allocation and also on the *Affinity* of the process with respect to the selected processor. Similarly, *Size* data are min-max pairs related to the number of bytes needed for code/data (to be used during DSE step when a process is implemented on GPP) and, since this case study refers to a FPGA as SPP, to the number of Slices/LUT (to be used during DSE step when a process is implemented on SPP). Such values have been estimated by means of some benchmarking activities. Finally, *Load Estimation* for 8051 and LEON3 is performed by means of timing co-simulations with respect to a Time-To-Completion (TTC) constraint [4.34]. The goal is to estimate the load that each process would impose to the selected processors to satisfy the constraints itself. The final results are a pair of estimated loads (for 8051 and LEON3) for each process.

As specified before, in this case study, the communication infrastructure has been fixed (i.e., processors with local memory and a single shared bus among them). So, the *Bandwidth Estimation* is not needed since the DSE step doesn't have to suggest interconnection links to be used. Moreover, the timing/energy co-simulator will take directly into account the characterization data, related to the selected shared bus, provided in TL. The Co-Estimation is so concluded.

Once collected all the metrics and all the estimations needed for the DSE step, the following additional constraints are imposed:

- *Timing/Energy constraints*: given the *worst case time-to-completion* (i.e. WCTTC), estimated by means of a timing simulation performed allocating all the processes on a single 8051 instance, the DSE step has to suggest architecture/mapping pairs able to satisfy both a timing requirement of x*WTTC, with x belonging to (0, 1), and an energy requirement of an energy consumption less than a given ETC (Energy-To-Completion).
- *Architectural constraints*: the DSE step can use max 4 instances of 8051, max 2 instances of LEON3 and max 1 instance of Spartan3AN.
- *Scheduling Policy*: processes implemented in SW and allocated on the same processor are subjected to a FCFS scheduling policy with 10% overhead for context change.

Considering different TTC and ETC constraints, the DSE step provides the results shown in Figure 10 (ordered by increasing execution time). Such results can be then used to identify the relevant configurations to be used at run-time for max timing performance, max energy saving, or a specific performance/energy trade-off.

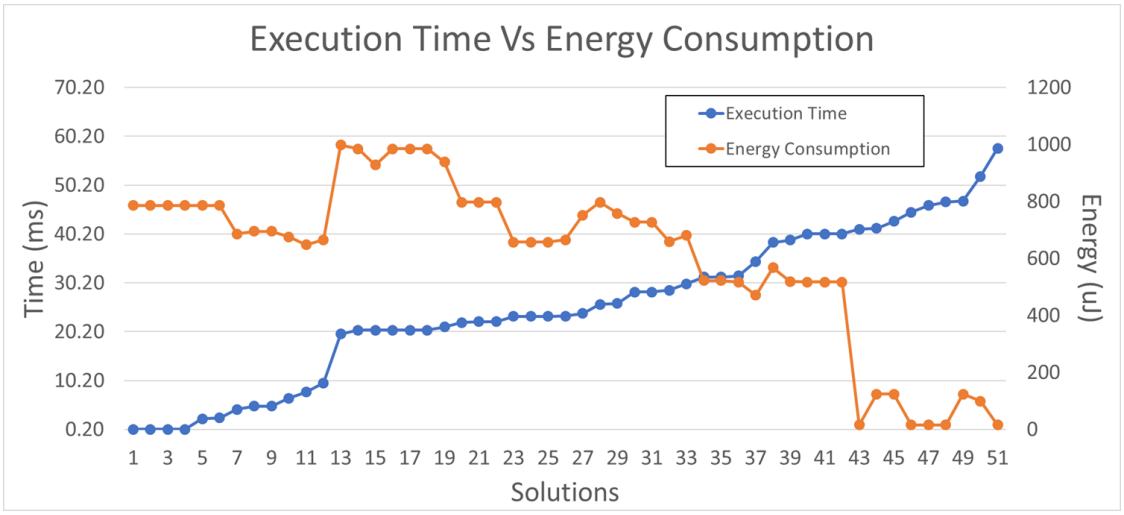


Figure 10: Timing/Energy trade-off.

As an additional example, Figure 11 shows as, by reducing TTC, suitable solutions are possible only by relaxing the ETC in a proper way.

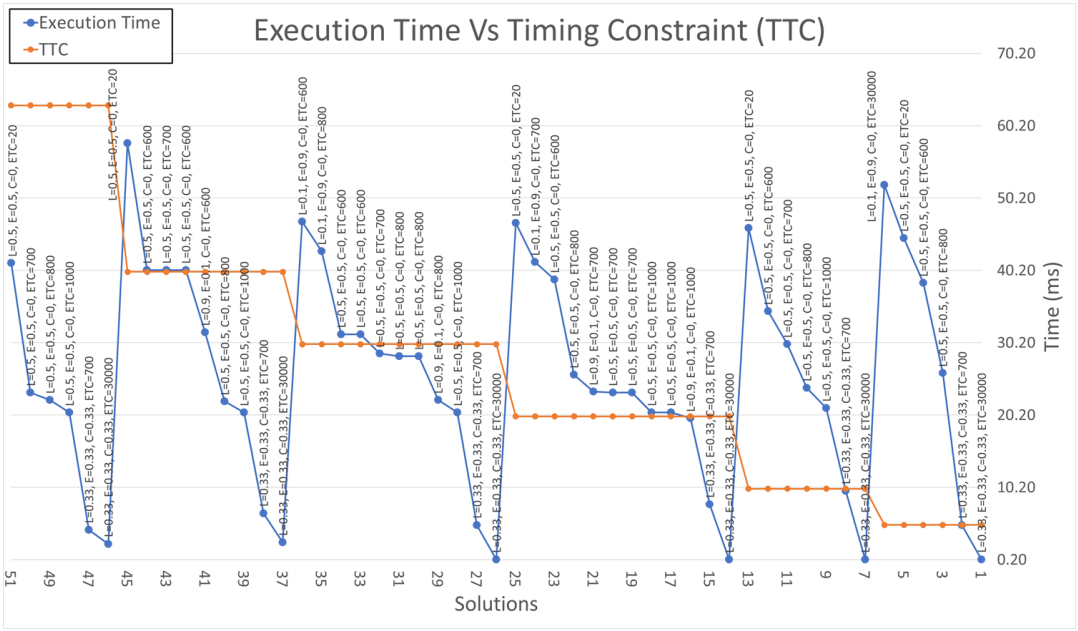


Figure 11: TTC/ETC relationship (L = load, E = energy, C = Cost).

Finally, Figure 12 shows a trade-off analysis that considers also the monetary cost of each suggested solution. It is worth noting as it is possible to identify some interesting outliers that would be difficult to imagine only on the base of the designer experience.

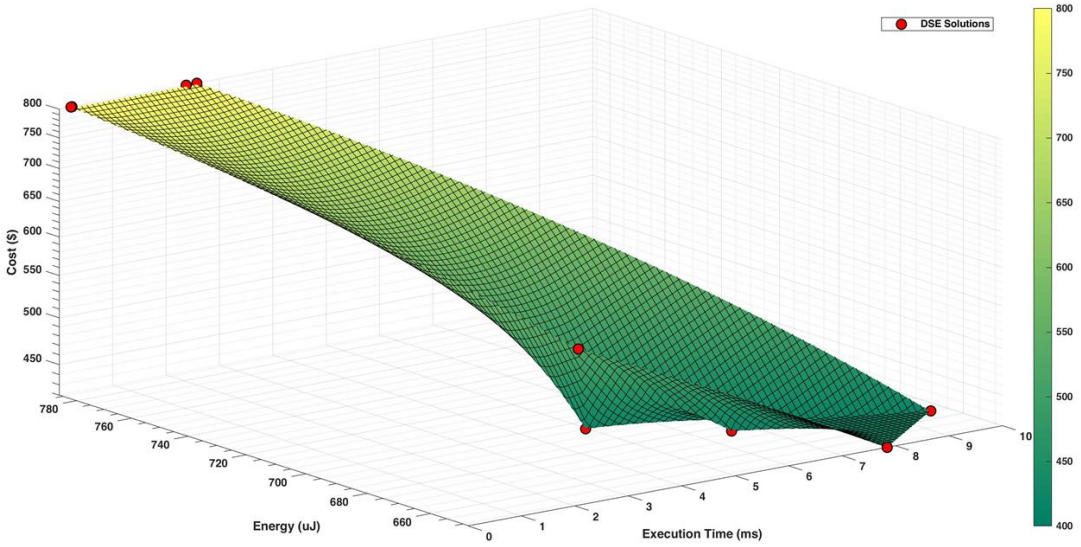


Figure 12: Timing/Energy/Cost trade-off.

4.2.3. Use case #2: Digital Camera

In order to show the main features of the extended methodology when applied to a domain related to the Italian UC (i.e., image processing), it has been applied to the *Digital Camera* (DC) case study proposed in [4.33], where a simplified JPEG compression is taken into account. The case study presents several implementations providing different performances with respect to timing, energy/power and cost. In

such a context, the energy-aware HEPSYCODE (in particular the HEPSIM simulator) has been used to both check the consistency of the results with those provided in [4.33], and to perform a wider DSE considering more degrees of freedom.

The DC main tasks are to capture images, store them in a digital format and download them on a PC for permanent storage (using a serial cable for the connection). The processing is initiated when the user presses the shutter button of the camera; a special sensor, called CCD (charge-coupled device), is used to convert the image into a digital form. A CCD is composed of many small cells that become electrically charged when exposed to light. The charge of each cell is converted into an 8-bit value that represents a pixel of the image. Some of the columns are covered with a black strip of paint in order to perform a zero-bias adjustment: due to manufacturing errors the cells of a CCD may measure a value that is slightly above or below the actual value; this error (zero-bias error) is typically the same across columns, but different across rows. For this reason, if a covered cell registers a value different from zero, we detect the zero-bias error for that row. A CCD is capable of discharging the cells, sending 8-bit at a time as an output (the 8-bit value represents the charge value of a cell, i.e., one pixel). At this point the digital image is available, with a 64x64 resolution (64x64 is the default value, but other size can be selected to trade-off timing performance and energy consumption) and two extra columns to perform the zero-bias adjustment: for each row the average of the two values of the zero-bias is performed and the error is corrected by subtracting this number from each element of the row. The next step involves the compression of the image in order to reduce the number of bits needed to store the image and to transmit the image to the PC in less time. The image is compressed by using the JPEG encoding, with a mode of operation that use the discrete cosine transform (DCT): the digital image is divided into blocks of 8x8 pixels and each block is processed in three steps: forward DCT, quantization and Huffman encoding (the last one, as done also [4.33], is omitted from the model). The quantization step reduces the bit precision of the encoded data; in this way fewer bits are needed to store the data, and compression is achieved. To do so, each pixel is multiplied by a factor of 2 (i.e., each 8-bit value is right-shifted). When the compression phase has been performed on every 8x8 block of the digital image, it can be transmitted serially to the PC, using a UART. The flowchart in Figure 13 describes the high-level functionality of the digital camera as considered in the use case.

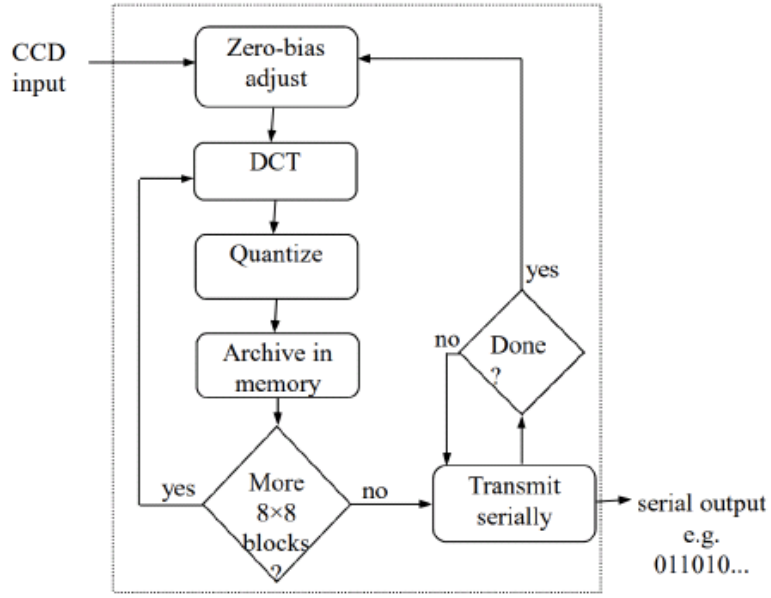


Figure 13: Block-diagram specification of a simplified digital camera.

The CSP-based system-level model of the DC behaviour is shown in Figure 14.

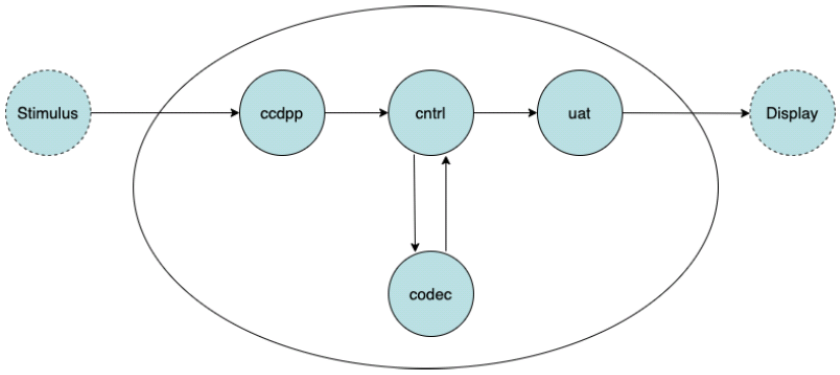


Figure 14: CSP-based DC system-level model.

Apart from the main behaviour of the DC, modelled by means of 4 processes and 4 internal channels, a testbench is required to execute the specification: *Stimulus* provides the required inputs to the system and *Display* shows the corresponding results. Then, the CSP-based model has been represented using SystemC as a specification language. In order to achieve this goal, processes have been modelled by using classic SC_THREAD, while channels have been modelled by introducing a proper SC_CSP_CHANNEL derived from the SC_FIFO with an interface that offers blocking *write* and *read*. An SC_THREAD modelling a process presents an infinite loop behaviour and accesses only to its local variables, so the communication with other processes occurs only by means of CSP channels. The whole system behaviour is enclosed into a single SC_MODULE containing all the processes and internal

channels. Moreover, the testbench is modelled by other SC_MODULE connected to the system by means of proper SC_PORT and channels. The high-level representation of the system and testbench is shown in Figure 15 and a description of the different SC MODULE and SC_THREAD in them is given below, while details about SC_MODULE System are provided in Figure 16.

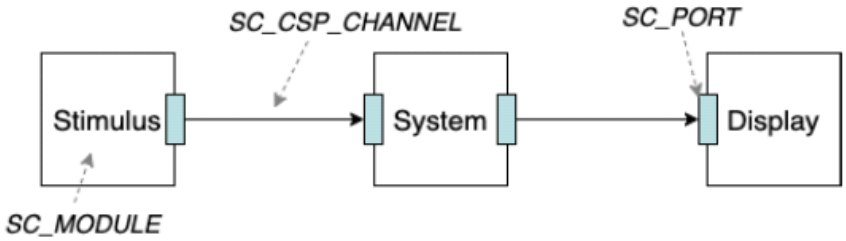


Figure 15: High-level representation of the system and testbench.

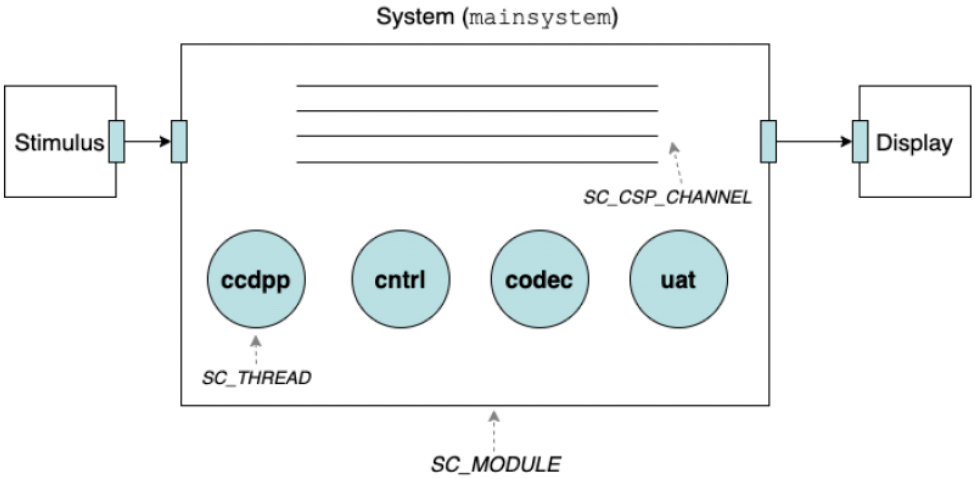


Figure 16: Detail of the System SC_MODULE.

The first step of the co-design flow is the *Functional Simulation*, where the system is simulated to check its correctness with respect to some input data sets. This simulation allows detecting errors in the model like wrong outputs or critical conditions (such as deadlocks). Functional Simulation does not consider the time needed to execute the statements composing the processes, so statements are executed in 0 time. Since the SystemC model is executable by construction, this simulation is directly based on the simulation kernel provided by the standard SystemC library.

Once the behaviour of the system has been verified, the design space exploration step is performed. This step consists of determining the system's architecture, i.e., the number and type of processors (SPP, GPP, ASP), memories and buses, and involves the mapping of the system processes to that architecture (multiple functions may be mapped to a single processor). An implementation is obtained by fixing a particular architecture and mapping and the solution space (or design space) is the set of all

possible implementations. The authors in [4.33] have analyzed four different implementations, starting from a single general-purpose processor connected to flash memory and RAM and mapping all the functionalities to software running on that processor. Although this implementation allows satisfying power, size and time-to-market constraints, it does not meet timing performance requirements imposing that the DC must process an image in 1 second. For this reason, the implementation has been modified in order to speed it up by using different approaches that involved the exploitation of single-purpose processors for time- critical functions. In summary, the considered implementations are the following ones:

- Implementation 1: Microcontroller Alone (AllSW)
- Implementation 2: Microcontroller and CCDPP/UART
- Implementation 3: Microcontroller and CCDPP/UART/Fixed-Point FDCT
- Implementation 4: Microcontroller and CCDPP/UART/CODEC

Timing performance of Implementation 1 has not been estimated, since a rough analysis allowed estimating that only *ccdpp* process would take nearly half of the time budget of 1 second. All the other implementations have been simulated using a VHDL RTL (Register Transfer Level) description. A synthesizable implementation of 8051 microcontroller (written in VHDL) is available, all the SPPs in the system are modeled in VHDL too, and the software modules are compiled and linked in order to obtain the final executable that can be translated into the VHDL representation of the ROM (using a ROM generator). In this way the entire system is represented in VHDL-RT, so the simulator interprets it and simulates the execution of the final system. Moreover, an ASIC synthesis tool has been used to estimate, at gate-level, energy and size. A summary of the estimations obtained with this procedure is shown in the table below.

	Implementation 2	**Implementation 3**	**Implementation 4**
Performance (second)	9.1	1.5	0.099
Power (watt)	0.033	0.033	0.040
Size (gate)	98,000	90,000	128,000
Energy (joule)	0.30	0.050	0.0040

In order to compare the DSE results provided by HEPSICODE, the different configurations proposed in [4.33] have been simulated at system-level by using HEPSIM. Moreover, in order to provide more information about the possible trade-offs obtainable by scaling the image size and considering different working frequencies and interconnection links bitrates, HEPSIM has also been used to perform worst- and best-case analysis. The final goal is twofold: to validate the results provided by HEPSIM and to show the possible trade-offs that can be considered to select different system configurations.

The *Processing Units* and *Physical Links* (that allow the communication among processes) that have been inserted in *Technologies Library* to simulate the different implementations are listed below:

- Processing Units
  - Intel 8051 (GPP) 12-20 MHz (GPP)
  - Xilinx Artix7 12-20 MHz (SPP)
- Physical Links
  - GPIO PORT of width 8 (to connect Stimulus and ccdpp, when ccdpp is implemented in SW)
  - CUSTOM of width 8 (to connect Stimulus and ccdpp, when ccdpp is implemented with a SPP)
  - BUS8051 of width 8 (to connect 8051 and memory/SPPs)
  - GPIO PIN of width 1 (to connect uat and Display, when uat is implemented in SW)
  - UART, of width 1 (to connect uat and Display, when uat is implemented with a SPP)

After Processing Units and Physical Links have been selected, the simulations have been performed. The mapping between processes and Processing Units and the mapping between channels and Physical Links have been performed according to the implementations suggested by [4.33], but also considering the most relevant trade-offs.

It is worth noting that in [4.33] not all the details are completely clear, so there has been the need to make some assumptions (i.e., a precise comparison is not always possible). The table below reports the results obtained for the *very worst* and *very best* scenarios (HEPSIM source code customized for the DC and all the files related to such simulations can be found in [4.36]):

- Very Worst Case
  - Application
    - Image size: 64x64
    - Affinity: 0 for each process
  - Timing
    - Working frequency: 12 MHz
    - CC4CS: max
    - Scheduler overhead (estimated by exploiting the info provided in [4.37]): max
    - Links bitrate: min for each link
  - Energy
    - J4CS: max

        - Link energy consumption (estimated by exploiting the info provided in [4.38]): max for each link
- Very Best Case
    - Application
        - Image size: 16X16
        - Affinity: 1 for each process
    - Timing
        - Working frequency: 20 MHz
        - CC4CS: min
        - Scheduler overhead: min
        - Links bitrate: max for each link
    - Energy
        - J4CS: min
        - Link energy consumption: min for each link

TIME			
Implementation	**Simulated time (s) [ESD2001]**	**Simulated time (s)-VeryWorstCase**	**Simulated time (s)-VeryBestCase**
2	9.1	10.7	0.096
3	1.5	10.7	0.096
4	0.099	5.73	0.021
ENERGY			
Implementation	**Energy (uJ) [ESD2001]**	**Energy (uJ)-VeryWorstCase**	**Energy (uJ)-VeryBestCase**
2	300000	1.08675 e+7	68337.3
3	50000	1.08675 e+7	68337.3
4	400	7.86243 e+6	48367.2

The simulation results show the timing performance and energy consumption intervals obtained by considering all the possible system configurations for the reference implementations. It is worth noting that all the results provided in [4.33] are included in such intervals. The only exception is related to the energy values for *Implementation 4*: this is because the estimation provided in [4.38], used as a reference for the simulations, are related to inter-chip communications, while [4.33] considers only intra-chip ones (characterized by a reduced energy consumption). Finally, it shall be highlighted as all the considered simulations have been performed by starting from the same system-level SystemC-based model without the need to perform any kind of modification. This, with respect to other approaches where different models at different abstraction levels are involved (e.g., 3 different VHDL-RT models in [4.33]), allows a faster and wider design space exploration useful to support very early alternatives evaluation.

4.2.4. Conclusions and future works

UNIVAQ has finalized the extension of the HEPSYCODE methodology to consider non-functional requirements (NFR) related to energy consumption, by exploiting a high-level (i.e., statement-level) energy performance metric. Respect to the plan exposed in D3.2, since more effort than expected has been needed to fix and improve

the management of multiple heterogeneous links in the HEPSYCODE simulator (i.e., HEPSIM), the planned work related to the accuracy (i.e., to consider also a set of alternative algorithm implementations providing results with different accuracy) has been postponed as work for future projects.

4.3. The SAGE verification suite

The SAGE Verification Suite (SAGE-VS) is a set of SW tools aimed to accomplish different formal verification tasks at design time. The main components of the SAGE-VS are:

- SpecPro: a software library to translate requirements from natural language to logical language.
- ReqV: a tool for requirements management and consistency formal verification.
- HyDRA: a tool aiming at synthesizing an optimal and “correct-by-construction” policy given a model and tasks in logical language.
- ATG: a tool for requirements-based test suites generation.

The key features of the SAGE-VS are

- Automated consistency checking of requirements expressed in natural language (ReqV component).
- Automated inconsistency finding in case of inconsistent requirements (ReqV component).
- Organization and storage of requirements in an online platform (ReqV component).
- Automatic synthesis for goal oriented "correct-by-construction" policies from a system model and an objective (HyDRA component).

The inputs are:

- Set of requirements in natural (controlled English) language, formulated as Property Specification Patterns. (PSPs) for Linear Temporal Logic extended to constrained numerical signals (ReqV component).
- Hybrid model of the system with safety limits (HyDRA component).

The outputs are:

- Consistency result (yes/no). In the case of inconsistency, the tool returns the minimal set of requirements that causes the inconsistency (ReqV component)
- A yes/no answer on whether the system can be used to achieve the tested use case. A yes answer comes with a correct by design plan to achieve the given objective. The plan accounts for both the discrete and continuous limits of the system so that the plan is valid and guaranteed to be executable and thus constitute a proof that the system has the targeted capability.

- ReqV: extends the expressivity of input PSPs to allow the translation in a logic language for hybrid systems and improve the usability of the GUI.
- HyDRA: defines a more usable input language and improve the performance of the planner in terms of execution time.
- ATG: is release of the first stable version.

4.4. Scenario- and platform-aware design flow for image-based control systems

TUE has developed a scenario- and platform-aware design flow image-based control loops as well as software support for application development for the same. Image-based control (IBC) systems are increasingly being used in various domains including healthcare and autonomous driving. The key challenge in IBC is to deal with high computation demand while guaranteeing performance and safety requirements such as stability. While modern industrial heterogeneous platforms, such as NVIDIA Drive, offer the necessary compute power, application development on these platforms with performance and safety guarantees is still challenging. Alternative time-predictable platforms are not yet in widespread use.

A typical design flow for IBC systems consists of three distinct elements: (i) *mapping* tasks onto platform resources; (ii) *timing analysis*, consisting of *task-level worst-case execution time* (WCET) analysis and *application-level* analysis to obtain worst-case performance bounds on aspects such as latency and throughput; (iii) *controller design* using the obtained performance bounds, ensuring performance and safety. While such a three-step design process is modular in nature, it usually leads to over-dimensioned systems with sub-optimal performance, because task- and/or application-level timing bounds are pessimistic.

TUE developed a coherent design flow for IBC systems modelling, design and implementation that exploits the application-specific and platform-aware characteristics of IBC systems to cope with the long variable sensing delay and to optimise the system performance. The SPADe flow explicitly considers image workload variations, parallelisation of sensing processing, pipelining of the control loop and approximation of the camera image signal pre-processing. The SPADe flow is platform-aware and can explicitly consider multiprocessor system-on-chip (MPSoC) and can also be adapted for industrial platforms. We illustrate the method considering a predictable multiprocessor system-on-chip platform - CompSOC. See [4.39].

We validate the proposed method using hardware-in-the-loop (HiL) experiments with industrial heterogeneous multiprocessor platforms - NVIDIA Drive PX2 and NVIDIA AGX Xavier. We obtain an improved control performance compared to the state-of-the-art IBC design.

4.4.1. Scenario- and platform-aware design (SPADe)

The SPADe flow comprises the following steps as shown in Figure 17:

- identify, model and characterise the frequently occurring workload scenarios that characterise the dynamic behaviour of the image processing in the control loop;
- find optimal mappings for these scenarios considering a defined implementation choice for the given platform allocation;
- identify optimal system scenarios combining workload and mapping information and taking into account constraints from the control domain, e.g. stability, and from the embedded domain, e.g. camera frame rate;
- design a controller with high overall QoC and guaranteed stability for the chosen system scenarios; and
- a runtime reconfiguration mechanism for implementation.

- explore the design-space for the optimal implementation choice that considers the degree of application parallelism and the degree of pipelining.

As already stated, we illustrate the SPADe design flow considering the predictability and composibility properties of the CompSOC platform. In the following, we detail the steps in the SPADe design flow.

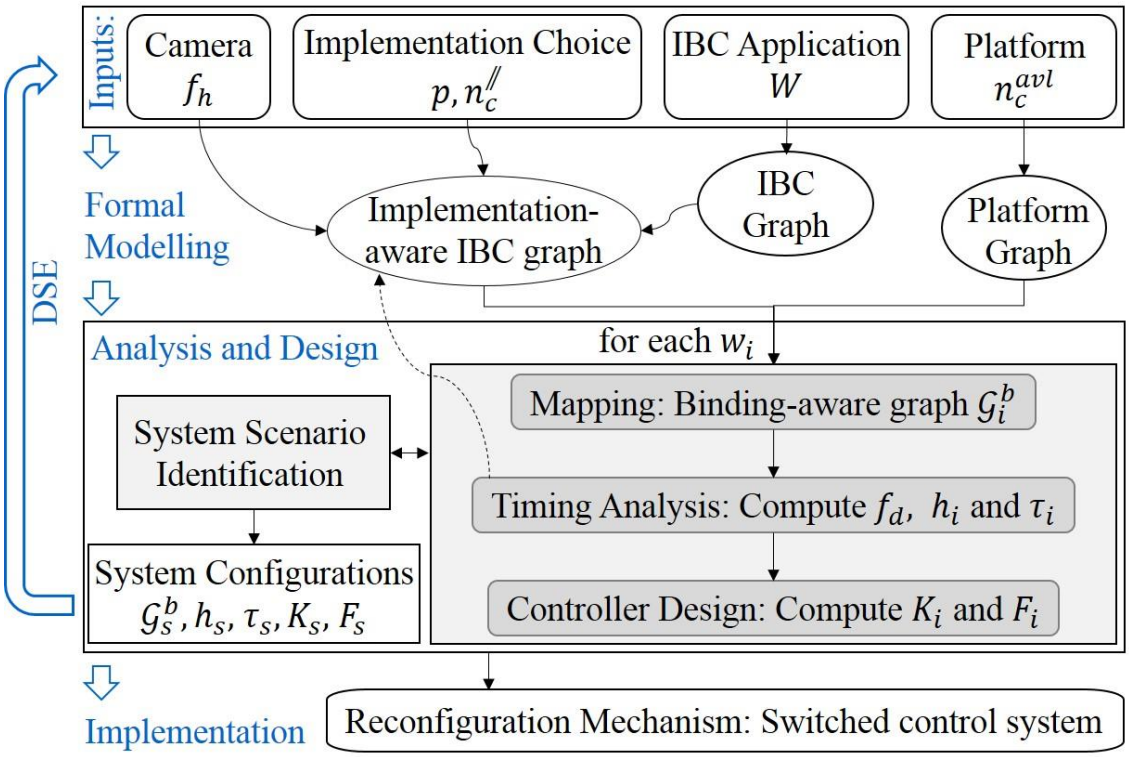


Figure 17: SPADe flow for multiprocessor image-based control systems.

Notation. f_h: Camera frame arrival period, p: number of pipes for pipelined implementation, $n_c^{//}$: number of cores per pipe for application parallelism, W: workload, n_c^{avl}: number of allocated cores for the application

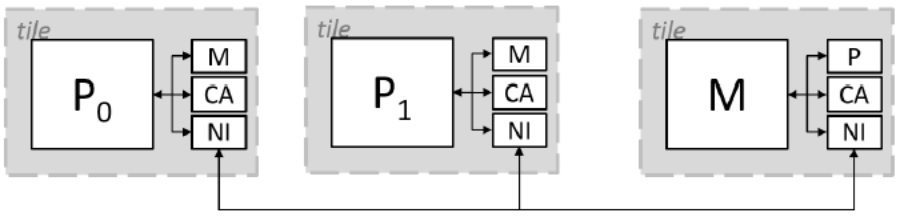


Figure 18: Multiprocessor SoC with two processor tiles and one memory tile.

4.4.2. SPADe inputs

The inputs to our design flow are details of the IBC application (e.g. the workload characterisation W), other applications sharing the platform, the implementation choices for the degree of application parallelism ($n_c^{//}$) and the degree of pipelining (p),

given platform allocation for the IBC application (n_c^{avl}) and camera characteristics, e.g. fps, f_h. These should be compliant with the application and platform models. Note that the details of the other applications sharing the platform are not relevant for a composable platform such as CompSOC.

4.4.3. Formal modelling: application and platform models

A typical IBC application model is modelled as a Scenario-Aware Dataflow Graph (SADFG) [4.14]. The SADFG of the sensing and processing algorithm receives the camera image frames and detects the regions-of-interest (RoID) in the frames. The detected regions-of-interest (RoI) can be processed in parallel on a multiprocessor platform. The number of allocated processors for our application determines the number of RoI processing (RoIP) actors in our model. Note that the sensor-to-actuator delay and sampling period vary based on the mapping to the processors. After processing the RoI, the data is merged and the controller state is computed by the RoI merging (RoIM) task. The control algorithm (C) then computes the controller input and feeds it to the actuation (A) task. This is explained later with examples.

Task-level WCET profiling is required to compute the WCETs on the CompSOC platform. The platform is modelled as a platform graph as shown in Figure 19 for the two platforms we considered. Model transformations are needed to obtain an implementation-aware graph to model the time-triggering of tasks, pipelining and inter-frame dependencies.

4.4.4. Analysis and design

System mapping: We first describe the system mapping, i.e., binding and scheduling, of our IBC application model to the platform. Figure 20 illustrates three workload scenarios (w_i) and their possible platform mapping. Each workload is associated with a SADFG. Figure 20 (a), (c), and (e) model the data flow graphs for different workloads and Figure 20 (b), (d) and (f) show their corresponding mappings on two or three processor tiles. Optimal mapping for a workload scenario w_i to a platform graph generates a binding-aware SDFG (G_i^b).

To have more processor tiles means that we can reduce sampling period h and sensor-to-actuator delay τ of IBC system by parallel execution of the sensing tasks. A lower h and τ are translated to a better performance of an IBC system. τ_i and h_i are the delay and period computed for a workload scenario.

System mapping refers to the mapping of application tasks (modelled as an SADF graph) to the platform. An application can have multiple mapping options for a given platform allocation. For example, in Figure 20 (c) and (e), the given platform allocation is two and three processor tiles respectively (visible in the number of RoIP actors) for the same workload (5 RoI).

Relation between dataflow and control design: The inverse throughput of the mapped binding-aware SDFG for scenario sequence S_i^{ω} gives the sensor-to-actuator delay τ, i.e.

$$\tau_i = \frac{1}{\overline{th}(s_i^{\omega})}$$

And sampling period

$$h_i = \left\lceil \frac{\tau_i}{f_h} \right\rceil f_h$$

where f_h is the camera frame arrival period.

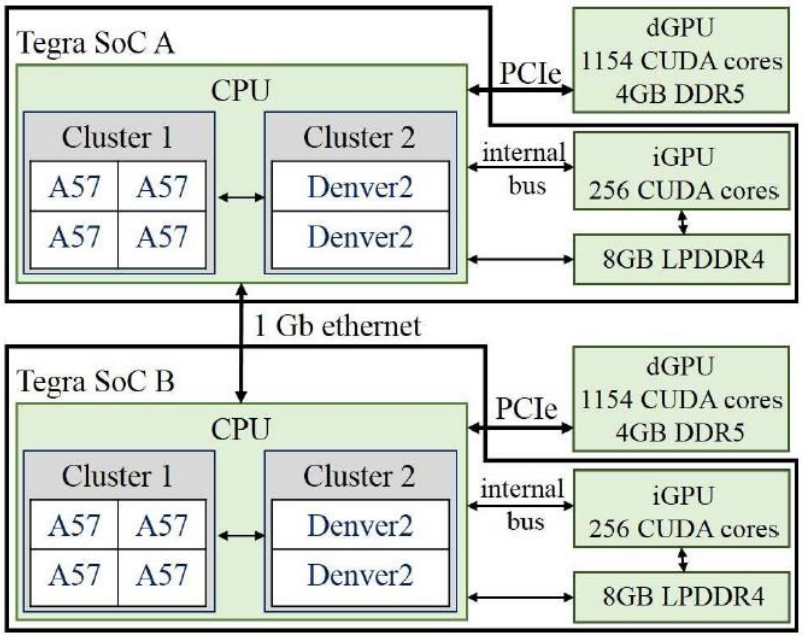


Figure 19: NVIDIA Drive PX2 platform graph structure.

Controller design: Once we obtain τ_i and h_i for mapped workload scenario w_i, they are then used for the discrete-time controller implementation and for designing the controller gains. See [4.14] and [4.28] for further details.

Further model transformations allow us to compute the inter-frame dependence time f_d as an inverse throughput of the transformed graph. Inter-frame dependence time is the minimum time to wait before we can start processing the next frame for a pipelined implementation.

Optimal system-scenario identification: It is possible for multiple workload scenarios to have the same sampling period due to implementation constraints like platform allocation and camera frame rate. For example, for the workload scenario represented in Figure 20 (a) with (h_i, τ_i), the number of RoI, #RoI =2. However, even for the workload scenario with #RoI =1 mapped to two processors, we would have the same timing parameters (h_1, τ_1) since the tasks would have to execute sequentially on one processor. Similarly, for the workload scenario in Figure 20 (c), we would have the same timing parameters for #RoI 5 and 6.

A system scenario s_s abstracts multiple workload scenarios s_i such that for $h_s = nf_n$ for some $n > 0$, $(h_s - f_h) < h_i \leq h_s$ and $\tau_i \leq \tau_{\dot{s}}$. Only the system scenarios are then considered for defining the control configuration and for platform implementation. The optimal system scenarios are identified and their corresponding control and mapping configurations are stored as a look-up table (LUT) in platform memory for runtime

implementation. A system configuration for a system scenario refers to the combination of the control and mapping configurations and contains the binding-aware graph G_s^b, the delay τ_s and sampling period h_s for the system scenario, and the controller feedback and feedforward gains K_s and F_s.

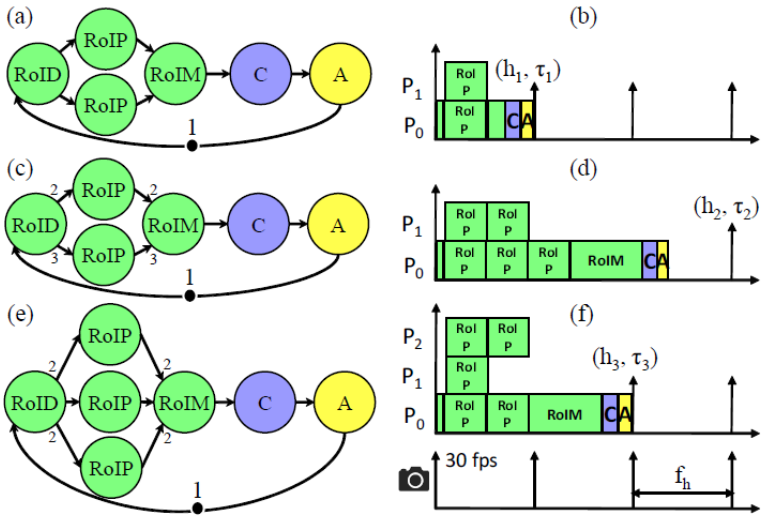


Figure 20: System mapping to MPSoC.

4.4.5. Implementation and runtime reconfiguration mechanism

During run-time, for every arriving input image frame, we compute the workload (e.g. through an image pre-processing step) and choose the correct system scenario associated with this workload from the LUT. Controller and mapping configurations of the corresponding system scenario are loaded from the LUT. A scheduler then reconfigures the mapping, the time-triggering of the actuation task and the controller gain parameters based on the chosen system scenario. The overhead cost for this reconfiguration has already been considered in our analysis model as a time cost in the start of sensing task.

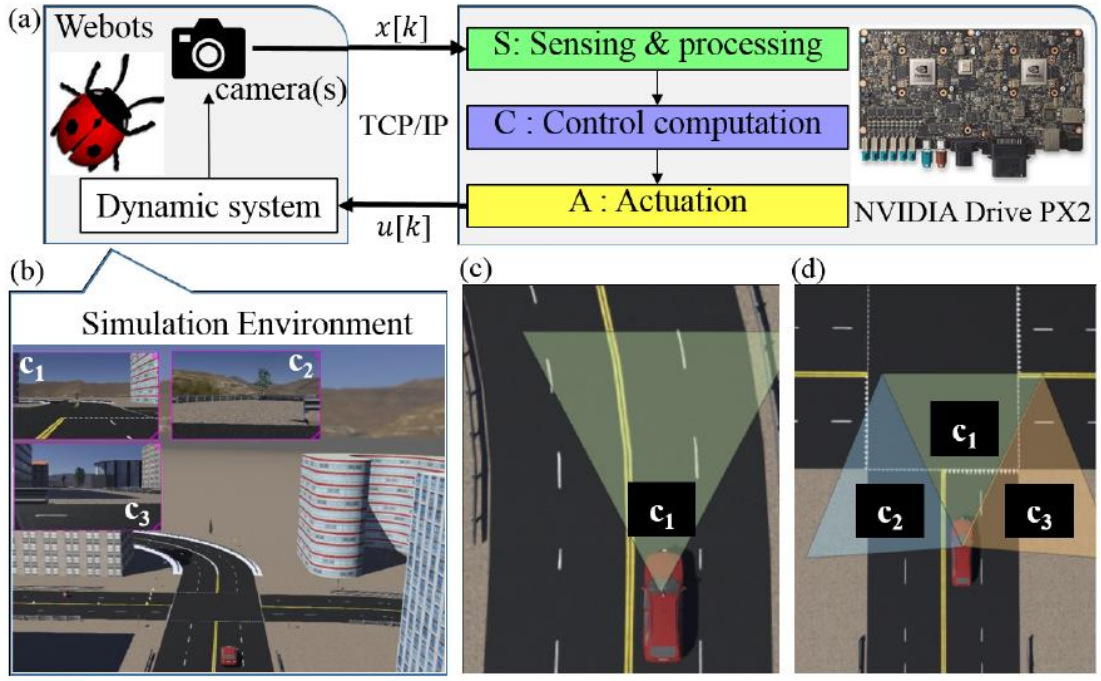


Figure 21: IMACS evaluation framework.

Notation in Figure 21: (a) IBC system block diagram and the HiL simulator. (b) a snapshot of the HiL simulation environment in webots. (c) LKAS using single camera. (d) multi-camera LKAS; c_1, c_2, c_3 are the cameras.

4.4.6. Evaluation: IMACS framework

IMACS [4.15] is an open-source framework for performance evaluation of IMAge in the Closed-loop System. This framework allows for software-in-the-loop (SiL) and Hardware-in-the-loop (HiL) testing and bugging IBC systems. We evaluated the proposed SPADe framework on the IMACS framework. The approach being developed and reported in Section 4.4 (on dynamic throughput tracking) will be integrated into IMACS framework once the method gets further matured.

4.4.7. Case study

We considered a concrete case study of a multi-camera lane keeping assist system (LKAS). The goal of the LKAS is to steer the vehicle autonomously to follow the centre line of a lane. Multiple cameras are used since the field-of-view of a single camera is not sufficient to detect the lanes when the vehicle has to make sharp turns, e.g., at a T-junction. Figure 21 (c) and (d) show the two different scenarios in the LKAS system. The first scenario s_1 (see Figure 21 (c) occurs when the vehicle is navigating on a road with no sharp turns. In scenario s_1, only one camera c_1 needs to be active. The second scenario s_2 (see Figure 21 (d) happens when the vehicle needs to take a sharp turn. In this case, all three cameras $c_1, c_2$and c_3 need to be active.

During runtime the scenarios are detected based on the following: i) when there is a lane detected by camera $c_1$and there is no request to make a turn, the LKAS executes in scenario s_1; ii) when there is no lane detected by camera $c_1$or there is a request to make a turn, the LKAS executes in scenario s_2. Our multi-camera LKAS is sharing the NVIDIA Drive PX2 platform with two other data-intensive applications - object detection and tracking (ODT) and automatic emergency braking (AEB).

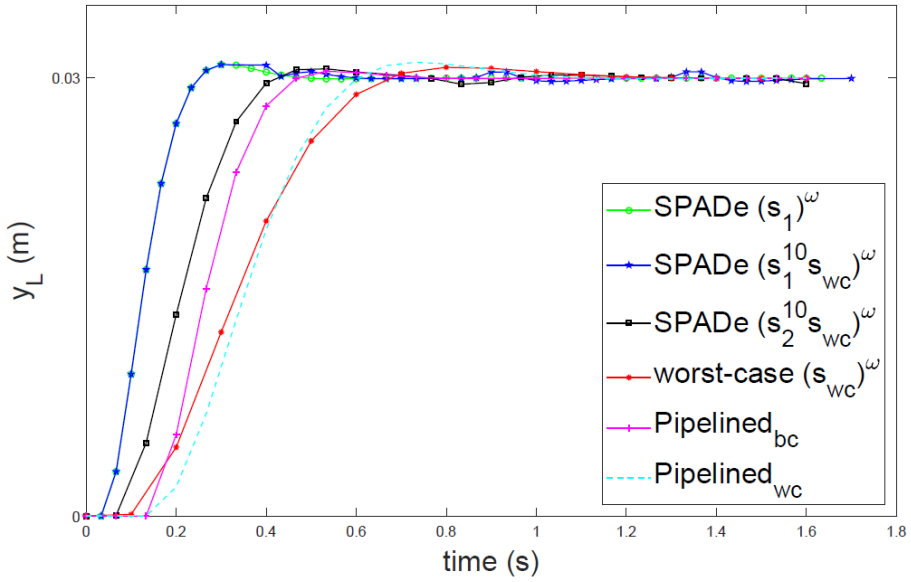


Figure 22: Comparison between SPADe and pipelined (state-of-the-art) approaches.

Notation: bc=best-case timing and wc=worst-case timing; SPADe is executed with a number of scenario sequences; y_L is the lateral deviation of the LKAS system under study.

4.4.8. Results and comparison

We compare our SPADe approach with a state-of-the-art pipelined control approach. For fairness in the comparison, we use the same control design technique - LQR with integral action - explained in [4.16] for SPADe. Further, we consider the same given platform allocation of two processors.

The results of the comparison of the pipelined controller with respect to the SPADe approach are shown in Figure 22. The controller is supposed to bring the lateral deviation y_L to 0.03m as soon as possible. The shorter time to reach the reference, the better the Quality-of-Control (QoC) is. Note that SPADe allows for parallelisation that reduces both sampling period and sensor-to-actuator delay. However, pipelining only reduces the sampling period. We observe that the QoC of the pipelined controller is always in the range of QoC between the worst-case (wc) design and the SPADe approach.

4.4.9. Design-Space Exploration (DSE)

We perform a design space exploration, if needed, to identify the optimal implementation choice for a pipelined parallelism implementation. A pipelined parallelism implementation considers both pipelining and parallelism together for a given platform allocation. In such a setting, it is not clear what the optimal choice for the best performance is. We compute the optimal choice analytically using the Gain Margin (GM) and the Phase Margin (PM). The higher the GM and PM are, the better the control performance is. The analytical computation can be further validated using Matlab simulations and/or HiL simulations where we compute the mean-squared error (MSE) and the settling time (ST) for our application. The lower the MSE and St are, the better the control performance is. A DSE for a HiL setting is shown in Figure 16.

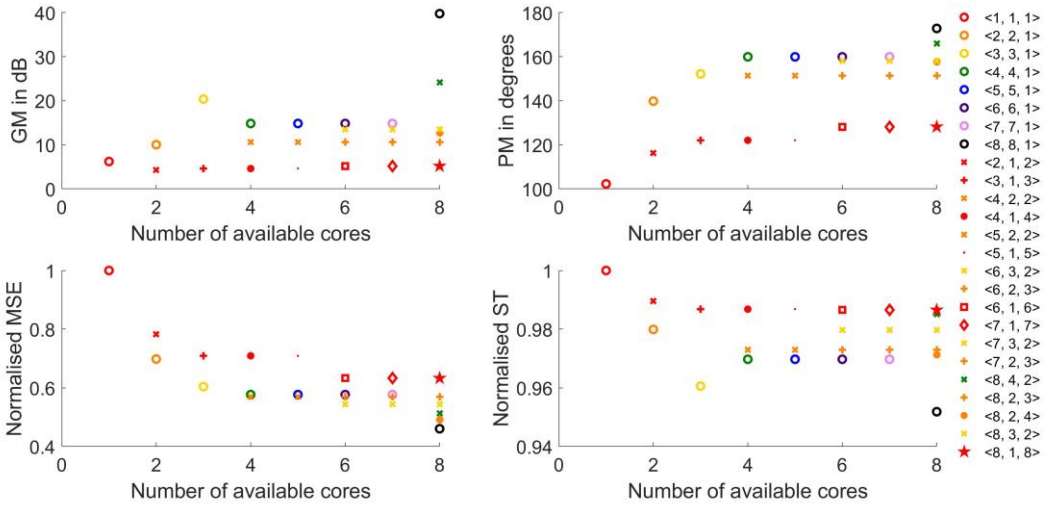


Figure 23: Design-space exploration for a HiL with different implementation choices.

Notation in Figure 23: The legend denotes <number of allocated cores, number of cores per pipe for application parallelism, number of pipes>, i.e. $<n_c^{avl}, n_c^{//}, p>$.

4.5 Modelling of real-time video processing systems with limited precision

A limited precision approach was applied to image / video processing pipeline. This was modelled and analysed prior to actual implementation. Application areas include CNN type processing and content analysis from a live video stream. After the simulation models, the approach was implemented in FPGA hardware and finally integrated to full custom ASIC along a RISC-V CPU core. The COVID-19 related delays at the processing site forced the IC to be abandoned, but the tests executed on FPGAs proved all the assumptions correct.

The key idea was to use a non-linear number space. This approach allowed using a reasonable dynamic range while limiting the data-path width, and, thus, energy consumption. Additional benefits include lower memory requirements and simplified arithmetic operations (for given operations).

The usability of this approach was studied primarily in the field of object detection. The purpose was to find domains where the loss of precision was not a significant problem,

and the benefits of the reduced precision processing outweighed the negative impacts. Also, conversions between the typical binary domain and this reduced precision domain were identified as a problematic aspect, potentially causing the approach to be unusable in some applications. To alleviate this problem, more arithmetic units using the same number space should be developed in the future.

4.6 Design time support for high level tool chains

UTIA developed support for high level modelling of IP blocks based on integration of the Xilinx System generator for DSP 2018.2 and Xilinx Model Composer 2018.2. Function of IPs can be modelled in bit-exact and cycle accurate Matlab/Simulink model before automated generation of the RTL code of the IP. Generated IP is integrated in Vivado 2018.2 based flow and communicates via the AXI-stream data interfaces with automatically generated data movers. SW API for these data movers is also automatically generated for the Debian OS applications running on ARM A9 or A53 processing systems.

The automated generation of HW data movers and corresponding PetaLinux kernel drivers is implemented in the high level tool chain based on the Xilinx SDSoC 2018.2 compiler with design time support for the PetaLinux 2018.2 kernel, Debian “Stretch” operating system. Developed and released evaluation packages are described in Chapters 7.1 – 7.4 of this deliverable.

4.7 High-level abstract component model and DSL

CUNI has been acting in the work package as a bridge between WP2 and WP3 in regard to component modelling. The concepts of the component model and the corresponding domain specific language to capture components of the model in textual format have been described in detail in D2.1.

In this section, we describe work performed in Y3 related to connection of the component model to the model-driven design space optimization.

CUNI has been modelling devices and functions as components. Generally, we distinguish two principal types of components – platform component (corresponds to a device or an execution platform) and application component (corresponds to a function – typically a data processing block). The main relation between these two types of components is that an application component runs on a platform component.

Components further exhibit input and output ports that can be used to construct video processing pipelines.

Components are hierarchically composable, which allows abstracting composition of components as another component. In this sense a smart camera can be composed of embedded board (a platform component) and software (application component).

An important feature of components is that they are configurable (e.g. FPS, video quality, etc.) and exhibit distinct qualities in each configuration (e.g. power consumption).

When composing components together, relation between components and their configuration parameters gets established (e.g. that two neighbouring components in

video pipeline have to operate on the same FPS or that a hardware component must provide enough memory to the software component). This effectively limits the design space of configurations.

From the perspective of WP3, the component model provides the structure (i.e. component architecture). The interpretation of the configuration parameters and their relation and influence on component qualities is based on the models discussed above in the section.

As such, the component model provides a structural part of the reference architecture that is specialized by use of corresponding modelling techniques to deal with particular aspects of energy, performance and other qualities.

4.8 Runtime reconfiguration Implementation of Embedded systems

RIE (Runtime reconfiguration Implementation of Embedded systems) is a component-based C++ implementation methodology. It also provides software reconfiguration capabilities for managing component implementations and system configurations at runtime. The RIE methodology has five basic elements

- User-defined data types. Specific C++ classes implement these elements.
- Component interfaces. C++ classes with pure virtual functions are used to model the required and provided interfaces.
- Components. RIE use C++ classes to implement components. These classes derive from an important RIE element, the “RIEComponent” class. This class accesses to all the application components and provides common services such as component monitoring, runtime reconfiguration and set-point modification. In RIE, a component is implemented with a base class and several implementation classes. The base class identifies the component required and provided services. This class derives from the “RIEComponent” class and all the interface classes that model the provided services. The base class does not include service implementations. The implementation classes derive from the base class and provide different implementations such as CPU-oriented code, OpenCL or HW accelerator implementation. For example, an image-processing component, “ImgProc”, provides an “I_Image” interface while the “Rgb2gray” component class defines a particular implementation of the “ImgProc” base class.

```
class ImgProc:  virtual public RIEComponent,  virtual public I_Image { …
class Rgb2gray:  public ImgProc, virtual public RIEComponent,  virtual public I_Image
```

The interface “I_Image” provides a service that access to an image:

```
class I_Image { public:  virtual void get_image(imageType &image)=0; …
```

- Instances. In order to support hierarchical designs, the methodology defines a class (RIEInstance) that allows instantiating child components in the parent component. The instances are associated to base components in the C++ code. However, the RIE infrastructure can associate at runtime a particular instance to any implementation class that derives from the base class.

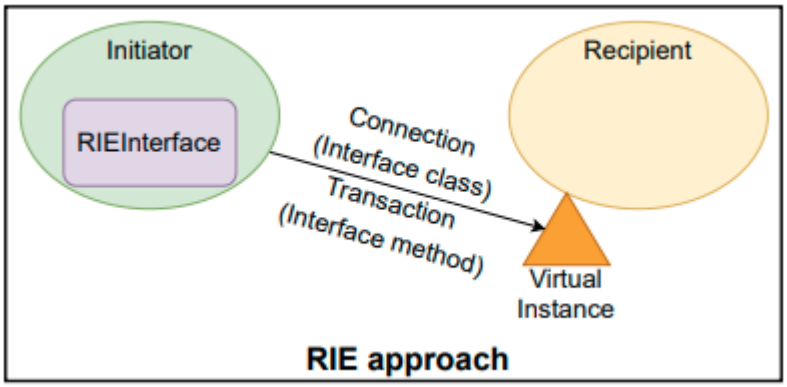


Figure 24: Example of RIE virtual instance.

- Required services. The C++ “RIEInterface” class defines the required interfaces of a component. These required services are associated to the instances that provide the services.

The RIE methodology supports runtime reconfiguration of the software components. In order to modify the configuration set point, several qualities are monitored at runtime. In the RIE-based methodology and WP2 abstract component model, a component may have several set points that define different component implementations and configurations. All the implementations of the basic component will share the same provided/required interfaces and a common set of configuration parameters and monitoring qualities. Each implementation or WP2 QRML alternative may have particular configuration parameters or qualities. The implementations represent different component mapping of the application into a physical platform (vertical composition in the abstract model of WP2).

The component implementations could also use different algorithms for the same behaviour in order to provide a different performance balance (e.g. reduce power consumption while increase service latency).

The RIE reconfiguration strategy minimizes the time, in which the system is not available. When reconfiguration starts, only those components that need to be reconfigured are stopped. Before stopping them, the system creates the new components, initializes them, and suspends their execution until the services provided by the old component are finished. After this, the instance associated with the component has to be replaced with the new version and the new component has to be activated to provide the new services. Finally, the old component is removed, completing the reconfiguration process. The diagram in Figure 25 shows the RIE reconfiguration strategy.

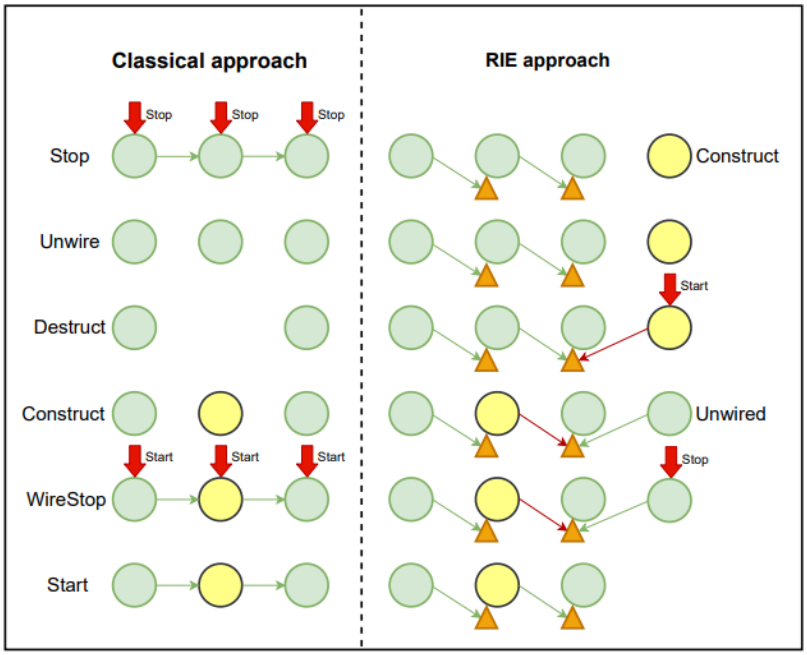


Figure 25: RIE reconfiguration approach.

The methodology supports on-the-edge component implementations. In this case, the components are integrated in component servers that are allocated in network nodes. These components use remote interfaces or particular implementations of the component interfaces that provide support for remote procedure calls (rpc methods). This methodology facilities the use of commonly used micro-service frameworks, such as Google grpc for rpc implementation.

As can be seen in Figure 26, the methodology specifies two platforms: local system and remote component server. Furthermore, a Domain Name Server, DNS, will provide the component server IP address and port number to the component that requires remote implementations (service discovery strategy).

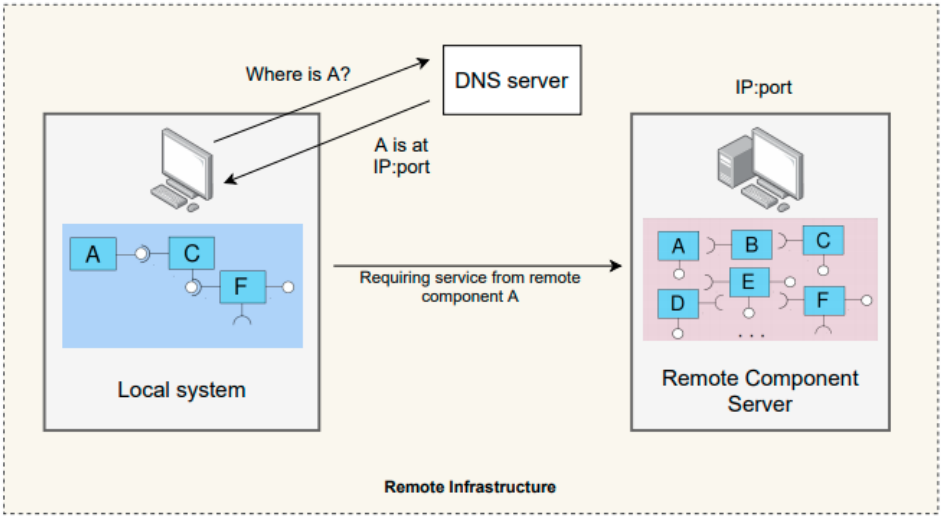


Figure 26: RIE remote infrastructure.

The system is implemented in the local platform, while the component server provides a set of functionalities that can be used by the local platform.

At runtime, the DNS server receives requests from the components with the value of a configuration parameter ("urlSink") and returns the IP address and port of the component server. This information allows establishing a connection between the local platform and the remote component server. Once the server that provides the remote component implementation is identified, the communication between both platforms is established by means of remote procedure calls, using the gRPC library.

This methodology defines three types of component implementations

- Local implementation. Implementation that uses platform resources to provide the functionality of the component. This implementation is allocated in the local platform and in the remote server.

- Local implementation of remote component. This implementation is found only in the local platform and is responsible for connecting the local platform with the remote one.

- Remote component. This implementation is allocated in the server and is responsible for communication with the corresponding "local implementation of remote component". The remote component uses a "local" implementation in the server to provide the required functionality.

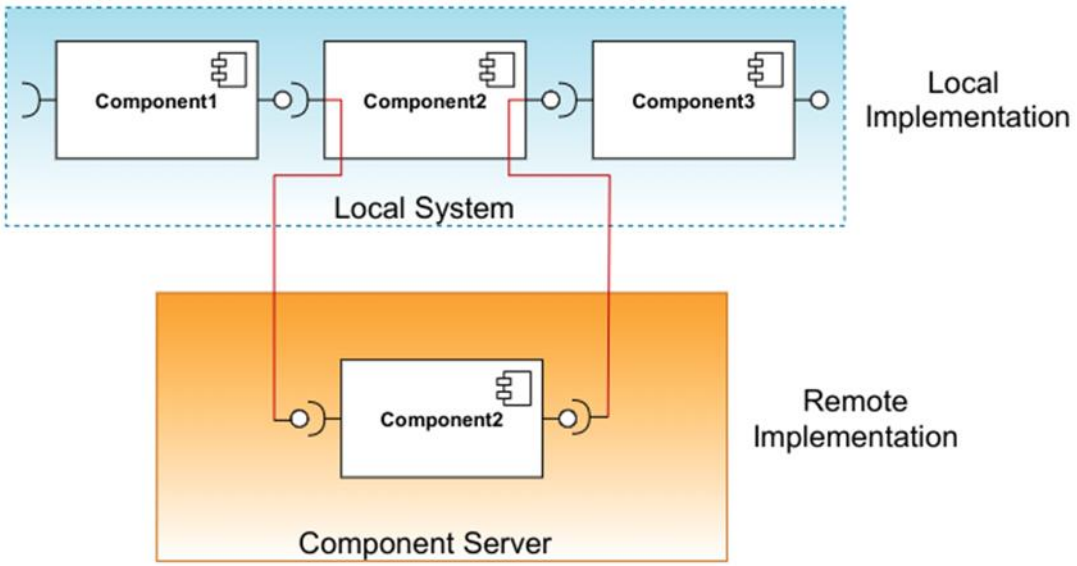


Figure 27: Remote component implementation.

In case of local implementations, the designer has to specify the component class that implements the component (RIE_Impl parameter), while for the local implementation of the remote component the designer has to indicate the server and the "local" implementation of the component. In systems with remote components, it is not normally possible to know at design time the IP address and port of the component servers, since these servers are developed and managed at runtime independently of the application. To solve this problem, a component configuration parameter ("urlSink") is used to identify the component server at runtime. In the component configuration,

the "RIE_Impl_Remote" parameter defines the implementation of the component in the remote server.

```
map < string , string > FaceDetection_configuration = {
{"s0",{"RIE_Impl":"FaceDetection","fps":30, ...}} ,
{"s1",{"RIE_Impl":"FaceDetectionHW","fps":50 , ...}} ,
{"s2",{"RIE_Impl":"FaceDetectionRemote", "urlSink":
     "urlRemote_FaceDetection_Comp0_base" , "RIE_Impl_Remote" : "s0" }}
}
```

Figure 28: Component configuration example.

Figure 28 shows the configurations of a component. The parameter "RIE_Impl" specifies the name of the implementation that is associated with a specific set point. For example, set point “s2” defines a local implementation of a remote component, which is specified by the "urlSink" parameter. The "RIE_Impl_Remote" parameter specifies the remote-component set point.

The RIE methodology and implementation library were designed taking into account the WP2 abstract models and the UML/MARTE design methodology. For this reason, it is possible to generate RIE code from the WP2 QRML language and UML/MARTE models. Some of these generators will be presented in the next section.

5. Programming and parallelization support

This chapter describes the techniques that have been added to the design and programming tools to improve their programming and parallelization support. Activities of all partners in this area also form a link to the WP4 programming support developments.

5.1. Static resource allocation and runtime scheduling

During the algorithm development, the key part is profiling which can hint on resource consumption during real execution of the algorithm. This is especially important for algorithms with strong data dependency where resource consumption is interlocked with the input and cannot be determined in advance. The only way is to actually run the algorithm and gather profiling information and statistics. Examples to this are detection of objects where the time required for analyzing an image is dependent on the image content.

We focused on platforms combining FPGA and ARM CPU like Xilinx Zynq. Tool enables logging of different types of events and their aggregation on both FPGA and CPU side, ensuring their interconnection in time. The log is stored in JSON format and can be viewed using the Chrome tracing tool. 128 bits are used to store the event, while the timestamp is 48 bits. Twelve different types of events are predefined, such as start/end of packet or frame, setting a specific value and others. It is possible to simultaneously monitor events on up to four interfaces with a data width of 8 to 64 bits.

SW profiling is solved by direct writing to the dedicated memory space of the profiling IP core. The number of events logged over time is limited by the size of the memory used to store the data and the throughput of the bus and DMA, which takes care of transferring the logs to application memory for subsequent storage in JSON.

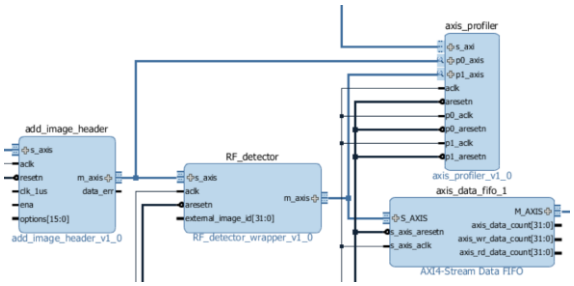


Figure 29: Profiling IP core connection.

Schematic diagram in Figure 29 presents connection of the profiling IP core to the detection IP for event logging and profiling.

The following code shows the JSON data produced by the IP Core.

```
{
"traceEvents": [
{"args":{"name":"AxiStream0-Image"},"cat":"__metadata","name":"thread_name","ph":"M","pid":1,"tid":1,"ts":0},
```

```
{"args":{"name":"AxiStream0-Line"},"cat":"__metadata","name":"thread_name","ph":"M","pid":1,"tid":2,"ts":0},
{ "pid":1, "tid":1, "ts":87705,  "ph":"B", "cat":"AXI-Stream0", "name":"Image", "args":{ "id":0 } },
{ "pid":1, "tid":2, "ts":87705,  "ph":"B", "cat":"AXI-Stream0", "name":"Line0" },
{ "pid":1, "tid":2, "ts":328154, "ph":"E", "args":{ "len":1280 } },
{ "pid":1, "tid":2, "ts":346867, "ph":"B", "cat":"AXI-Stream0", "name":"Line1" },
{ "pid":1, "tid":2, "ts":846867, "ph":"E", "args":{ "len":1280 } },
{ "pid":1, "tid":2, "ts":946867, "ph":"B", "cat":"AXI-Stream0", "name":"Line2" },
{ "pid":1, "tid":2, "ts":1146867, "ph":"E", "args":{ "len":1280 } },
{ "pid":1, "tid":2, "ts":1246867, "ph":"B", "cat":"AXI-Stream0", "name":"Line3" },
{ "pid":1, "tid":2, "ts":2228154, "ph":"E", "args":{ "len":1280 } },
{ "pid":1, "tid":1, "ts":2228154,"ph":"E" }
],
}
```

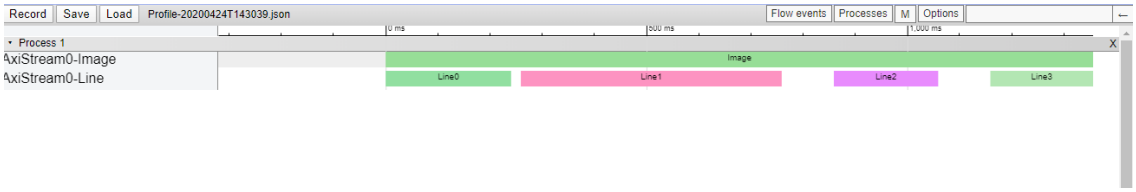



Figure 30: JSON loaded in Chrome Tracing for analysis.

5.2. Training WaldBoost detectors for FPGA

During the course of the project, we developed a package for training object detectors compatible with the ACF object detection IP Core (also developed in the project). We started with legacy Matlab-based code for training detectors with LBP features where compatibility with the FPGA solution was ensured by multiple workarounds. The main benefit of the package developed in this project is a simple use in other applications (since it is a small Python package). Compared to the older solution, we use a more recent detection model, and models have smaller footprint and higher accuracy.

The most recent version of the package can be downloaded from a GitHub repository [5.1]. It supports custom image channel features, decision trees as weak classifiers and full integer pipeline in training and inference of models. The parameters of the trained model are serialized as Protocol Buffer binary files and so they can be easily transferred to the target embedded system and uploaded to FPGA. We used the package to train models for license plate detection components in WP5, which is integrated in the demonstrator in WP6.

We described the package in D3.2. In the last project period, we improved data management and strategy for sampling training data. We dropped the requirement of TF object detection API (which was used for bounding box operations) in favor of a much simpler bbx package. And we improved the overall stability of performance of the training. Since the detailed description was given in the previous deliverable, we just repeat a few important points here.

The main purpose of the package is to generate object detector models for ACF Core. It takes the definition of a dataset (an iterator producing images with associated object locations) and runs a training process, which results in the definition of the model detecting the objects. In the training the input images are transformed to “feature channels” - a custom multi-channel representation of the image. In the case of ACF Core the channels are energy values in multiple directions (see Figure for an example). The detector is trained as a sliding window model over the images represented by the feature channels.

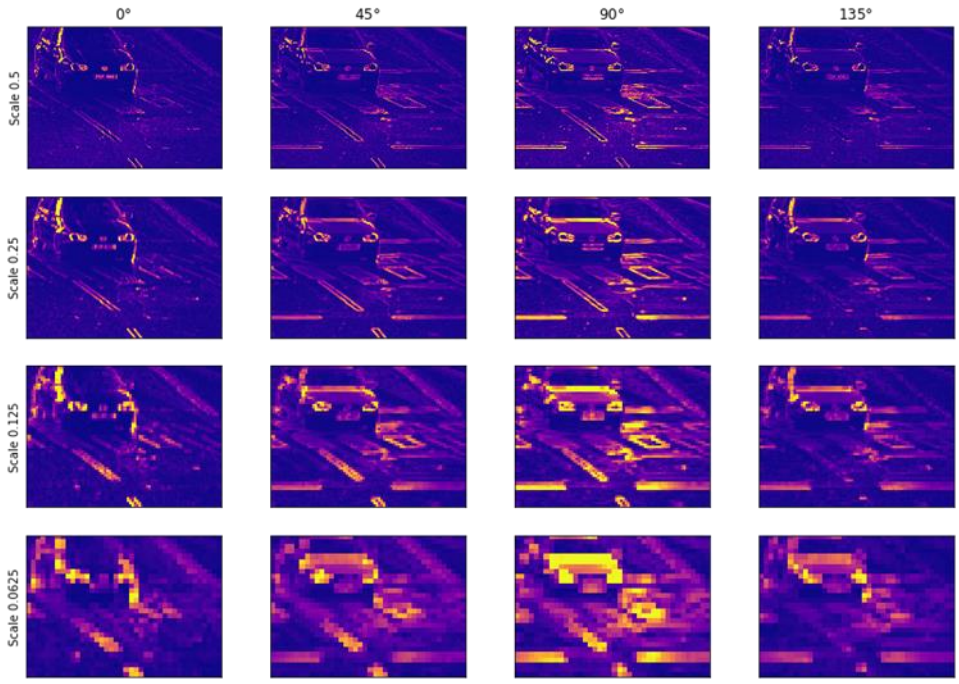


Figure 31: Feature channels extracted from the input image.

The final model can be saved to a binary file in ProtocolBuffer format and used for detection in the target application. The detection model can be applied to a new image by calling detect function.

```
boxes = wb.detect(image, model)
```

In boxes, there is a list of bbx.Boxes with locations of detected objects, which can be further passed to post processing (e.g. non maxima suppression, tracking, etc.)

Figure 32: Example of detected license plates.

Within FitOptiVis we greatly improved this software. We started with a few simple scripts for data sampling and training and progressively developed a fully usable package with a comprehensive interface. In the project, we focused mainly on the

development of the support for FPGA in the wb.fpga module but we also improved data management and internals of the package.

We use data from CAMEA to train license plate detectors, which are then incorporated in the FPGA object detector used in the license plate detection component within Traffic Surveillance Use case (UC5). We also experimented with other object types related to traffic surveillance scenarios - vehicle masks and unconstrained license plate detection.

5.2.1. Training example

The following code shows a simple example of how the new model is trained with the waldboost-python package.

```
import waldboost as wb
from waldboost import fpga
# An iterator producing dict with image and groundtruth_boxes keys
training_images = …
# Define detector window dimensions
h,w = 10,40
# Define how features are extracted from images
channel_opts = dict(
    shrink=2, n_per_oct=16, smooth=1, channels=fpga.grad_hist_4_u1)
# New model instance
model = wb.Model((h,w,4), channel_opts)
# Pool of training samples that are extracted from training images
pool = wb.SamplePool(min_tp=10000, min_fp=5000)
pool.update(model, train_images)
# learner defines type of weak classifiers within the model
# fpga.DTree is compatible with ACF Core architecture (if max_depth=2)
learner = wb.Learner(alpha=0.2, wh=fpga.DTree, max_depth=2)
# Train the WaldBoost model and save it to file
fpga.train(
    model, train_images, learner, pool,
    length=64, quantizer=64, bank_pattern_shape=(1,4), clip=3)
model.save("detector.wb")
```

The important thing, of course, is the ‘training_images’ object which supplies training data to the pool. The ‘train’ function incrementally builds the new model so it produces less false positives and keeps on objects.

The following code shows how the model is applied to a new image, producing a set of raw detections which are post-processed by non_max_suppression - final detections.

```
# Load image - np.array with shape (H,W) and unit8 dtype
import bbx  # package for processing of bounding boxes
image = …
dt = wb.detect(image, model)
dt = bbx.non_max_suppression(dt, iou_threshold=0.1, reduction="mean")
# dt is an instance of bbx.Boxes with locations of detected objects
```

This tool was used for training new models for a license plate detector component in WP5, which was applied in the Traffic surveillance use case.

5.3. OpenMP for real-time video systems

UC is using the OpenMP standard programming paradigm for system implementation. In this task, we defined the basic infrastructure to support OpenMP programming in the project platforms.

OpenMP (Open Multi-Processing) is a directive-based parallel programming language, mainly oriented to Symmetric Multi-Processing (SMP) architectures with shared memory. Traditionally, the OpenMP code was executed in a homogenous cluster of multi/many cores with shared memory. However, the latest versions support code offloading to other devices such as GPUs.

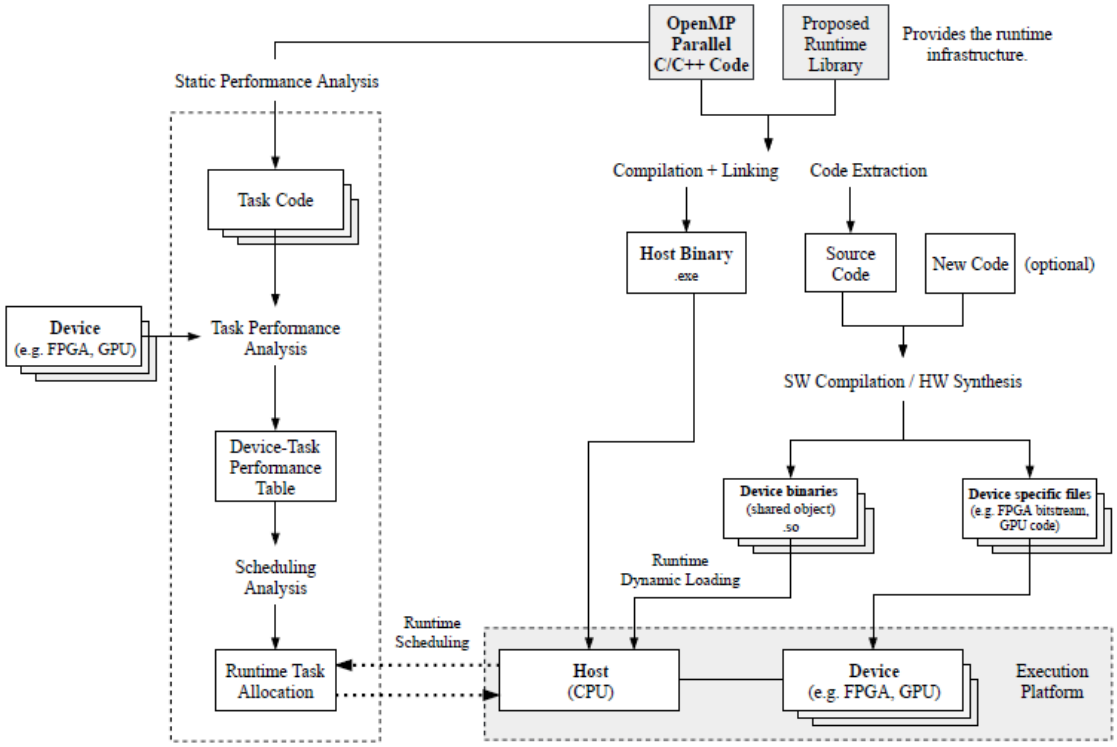


Figure 33: OpenMP-based reconfiguration methodology.

In this project, UC has extended the offloading capabilities of OpenMP (Version 5) with a new feature: source code offloading. This new feature allows extracting the source code of the OpenMP target regions. The OpenMP-based design methodology is presented in Figure 33. During compilation, the target region code is extracted and the OpenMP code is adapted to support runtime loading of functions that implements these target regions. The target regions are implemented with implementation platform-depended design flows. For example, for FPGA-based hardware accelerators the target region code is adapted and synthetized with standard FPGA design frameworks such as Xilinx SDSoC or Vitis. In this process, the performances of the platform-specific implementations are evaluated (static performance analysis). This information is used to define different system configurations. During execution, the system configuration management could select the best target region implementation taking into account the performance analysis results.

.

5.4. Design time support for C/C++ compilers and OpenCV algorithmic libraries

UTIA extended design time support for C and C++ compiler toolchains with OpenCV algorithmic libraries suitable for development/debug/execution on the 32 bit dual-core ARM A9 systems and on the 64 bit quad-core ARM A53 UltraScale+ systems. UTIA extensions include SW and HW versions of OpenCV algorithms for object detection from Full HD colour video input. See Chapters 7.1 – 7.4 for detailed descriptions.

5.5. TTA-based Co-Design Environment (TCE)

This section presents developments and activities done in the context of a co-design environment for customized Transport-Triggered Architectures called TCE. The produced processor cores can be realized as soft cores in FPGAs or integrated to new SoCs implemented as ASICs. TCE has been further developed in various aspects, which are described in the following subsections as well as in Section 6.5.

5.5.1. Support for 64-bit pointers and integers

Within the project a key missing feature was brought to TRL4, support for 64b pointers and integers. Earlier, TCE supported only 32-bit pointers and arithmetics. Although 32b can typically address a large enough address space in embedded applications or co-processing tasks the TCE cores are used for, the ability of sharing 64b pointers from 64b host CPUs allows easier integration of TCE-generated ASIPs to modern SoCs. The main limitation left for the future is that while wider than 64b datapaths are possible with SIMD instructions, the SIMD vectors may currently not contain 64-bit elements.

The implementation added a new top level target definition, “tcele64” to the compiler. The compiler automatically selects this mode when it notices that the compilation is performed to a 64-bit TTA architecture. When compiling code for the tcele64 target, all pointers are assumed 64-bit wide and 64-bit integer arithmetics are also supported. 64-bit TTAs must contain 64-bit versions of all the basic integer and memory operations to facilitate address computation, and the general purpose registers for compiler targets must be also 64 bits wide at this state.

In the base operation set of TCE, the 64-bit operations have the same base name as corresponding 32-bit operations, but add a suffix “64”. For example, the 64-bit add instruction is named “add64”.

5.5.2. Loop optimization support

Computer programs spend most of their time executing loops. Therefore, optimizing loops will have a large impact on the overall performance of executing a program.

The compiler of TCE has now two modes that optimize loops. The first mode is a loop scheduling mode (developed during the first project year of FitOptiVis), and the second mode is a new software pipelining mode, of which development started in year 2. Figure 34 shows the last phases of the TCE compiler and the differences between these two loop optimization modes.

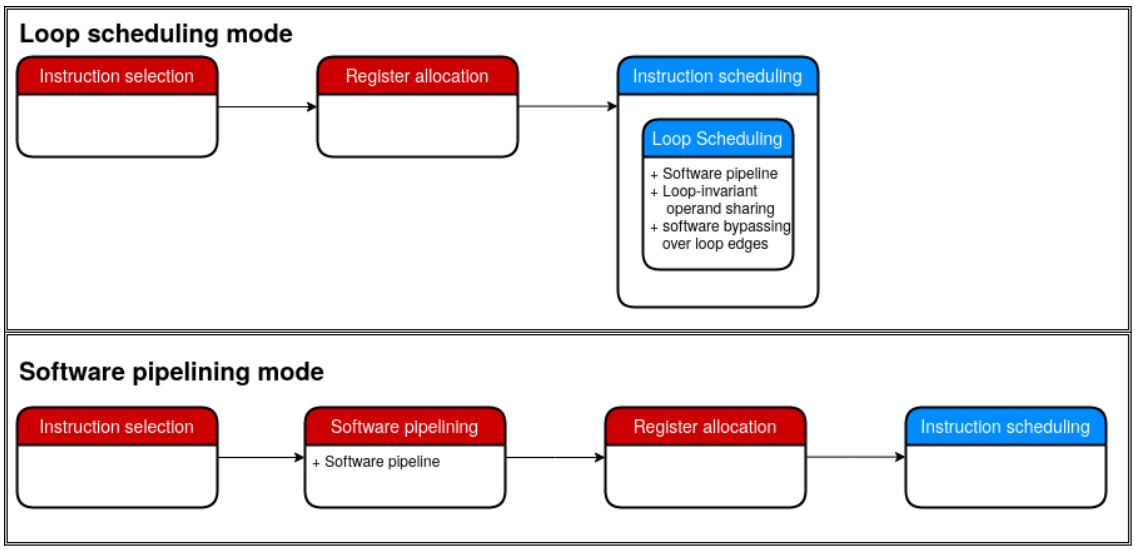


Figure 34: The two different loop optimization modes.

In Figure 34, red labels indicate re-used code from LLVM and blue labels indicate separate code managed by TCE code generator.

The loop scheduling mode is a part of the instruction scheduling phase of the TCE compiler. The instruction scheduler organizes the instructions into such order, that the original program semantics is preserved, but the hardware can execute the code in as efficiently as possible.

Statically scheduled architectures such as VLIW and TTA processors execute the code in exactly the order specified by the compiler, so the quality of the instruction scheduler has a big impact on the performance. On exposed datapath architectures such as TTA, the instruction scheduler can also perform various low-level optimizations, which can further increase performance and save energy.

Loop scheduling mode is a special mode of operation in the instruction scheduler that is used for scheduling code in inner loops. A loop scheduler typically interleaves multiple iterations of a loop, converting it to a “software pipeline”. This allows the performance of the loop to be considerably increased without unrolling the loop.

The basic idea of software pipelining is described in Figure 35. First, an initialization code called prologue is executed. It initiates the execution of the first iteration(s) of the loop. The loop body (also known as the kernel or the steady state of the loop) contains parts of code for multiple interleaved iterations of the original loop, so that each original instruction of the loop is there exactly once, but in a different order and for a different iteration than the original non-pipelined loop. After the body has finished executing, most of the original iterations have fully finished, but the very last ones are not. In order to finish the last iterations, a code block called epilogue is executed.

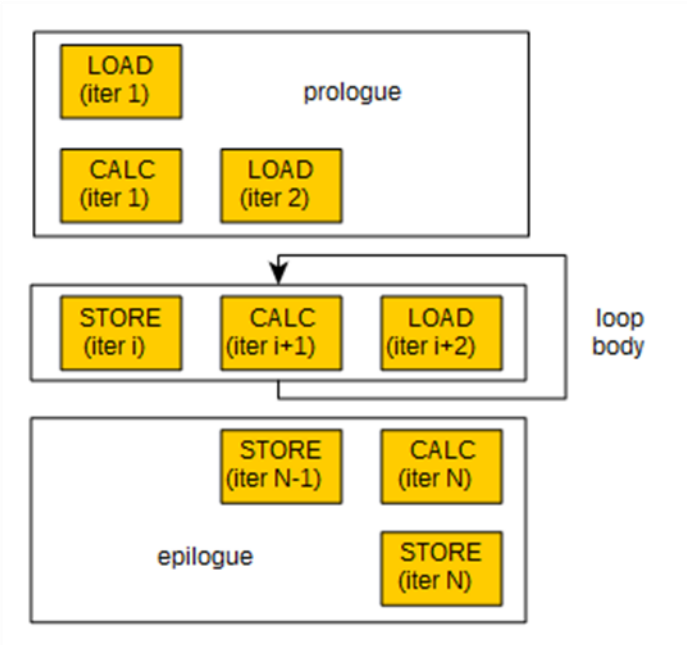


Figure 35: High-level example of software pipelining.

Figure 35 presents high-level Example of software pipelining with loop with 3 phases: load, calc and store. In this example three iterations of the loop are overlapped, so the prologue contains the beginning of two iterations and epilogue contains the end of two iterations.

Besides software pipelining the loop scheduling mode in the TCE compiler can also perform aggressive loop-specific optimizations, which take advantage of the TTA features; It can perform software bypassing over loop edges, which in some cases may even totally eliminate all register writes inside small loops, when all generated values are directly bypassed to instructions, which use it. The final result, which is generated by the last iteration, can then be written to a register in the epilogue only once.

Another loop-specific optimization the TCE compiler can perform is loop-invariant operand sharing, which means that immediate values or register-based values, which do not change inside the loop, may have to be read only once, in the prologue. The combination of these optimizations may allow creating code without any register reads or writes for small loops. However, the bigger body the loop has, the less effect these optimizations have.

The software pipelining in the TCE loop scheduling implementation currently has a limitation that it can currently overlap code from only two successive loop iterations. This can often limit the performance improvement achieved from it, as this means that speedups from over 2 can never be achieved from the loop scheduler over an optimized non-pipelined version of the loop, and the critical path of the loop easily dominates the cycle count, especially if the loop contains small amount of long latency operations.

This software pipelining limitation is a result of register allocation. The register allocation phase is done before the loop scheduler and chooses registers in a fashion that can limit the loop scheduler. For example, the register allocator creates output dependencies that reduce register pressure when not software pipelining, but prevents iteration overlapping when trying to software pipeline.

Another loop optimization mode, *the LLVM-based software pipelining mode*, was experimented in the last year of the project created to solve this issue. In this mode, the software pipelining will be done before register allocation. The used software pipelining implementation simply uses as many registers as needed and, therefore, is not be limited to overlapping only 2 iterations. However, this mode still has other limitations: Because of implementation reasons, TTA specific advantages are not yet utilized. Loop-invariant operand sharing and software bypassing over loop edges are not done in this software pipelining mode in the current status.

Furthermore, as the software pipelining mode is a complex work, it was still left unfinished within the FitOptiVis project and will be continued in other projects until it is stable enough to be used for daily work. For example, in its current state, the amount of iterations has to be fixed and not all amount of iterations create a correct result, thus, sometimes the code generator produced invalid code, fixing of which is the highest priority.

An example of a loop that can be pipelined in the software pipelining mode at the end of FitOptiVis with the new LLVM-based software pipeliner is shown in the following:

```
int sum = 0;
#define N 3


char dstBuf[N];
char dstBuf2[N];


for (int i = 0; i < N; i++) {
        sum+= dstBuf[i] * dstBuf2[i];
}
```

5.5.3. Instruction stream optimizations

The TCE toolset supports multiple compiler-assisted architecture features for optimizing instruction fetch to mitigate the pitfall of VLIW-style processors: the wide and “loose” instruction streams. Various mechanisms for implementing level 0 caches and other means for instruction streams were studied and validated down to implementation level within the FitOptiVis project. The focus on this work was on energy saving, while maintaining good performance with minimal runtime impact.

One such mechanism was a streamlined loop buffer which has two main variations: 1) *for-loop buffer*, where, before entering a loop, the compiler generates an instruction which specifies how many times the loop is executed and how many instructions it contains, and 2) *while-loop buffer*, where only the loop instruction count is specified, and there is a separate instruction for breaking out from the loop. At the end of the FitOptiVis project, the loop scheduler of the TCE compiler can utilize these variations automatically if the processor has any of these instructions and the processor is specified to have a loop buffer. The *for-loop buffer* may also allow removal of the loop counter update and comparison instructions from the code, reducing also the data path power consumption and leaving more space for other instructions in small processors. These loop buffers only work for loops, which do not have any control, such as if-statements inside them, and they do not support multiple nested loop levels. This is their main limitation made knowingly to simplify the hardware behind, and, thus, make it energy efficient and fast.

To answer research questions related to how much impact there is if we lift the control flow limitations of streamlined loop buffers, but utilize a smaller memory with its own control flow instructions for the hot spots, another mechanism for optimizing instruction fetches experimented within FitOptiVis was done: *Instruction Register File (IRF)* is more flexible than the loop buffer, allowing for example if-statements and nested loops. The IRF is like a compiler-assisted cache, which can contain a single block of code at a time, we call *IRF block*, which in our implementation is basically a *superblock* when using the global instruction scheduling terminology: Inside the IRF block there can be branches to any location inside the IRF block, and also jumps outside from the IRF block. Practically the only limitation the IRF has is that there may not be jumps, which jump into the middle of an IRF block from outside the IRF block. This also means that function calls cannot be positioned into the middle of IRF block, as the return from a function is a jump back to the call site. Thus, when there is a function call, a new IRF block starts after the function call. However, in our work that focuses on ASIPs for data-oriented applications, we consider function calls something that should be avoided anyhow in the hot spots of the application to be accelerated.

Since IRF is a compiler-oriented static L0 architecture, additional compiler analysis was needed to utilize it. The compiler analyses the control flow of the program and partitions the code into these IRF blocks, which can fit into the IRF and contains backwards jumps inside the same IRF block. These backwards jumps are then converted into special IRF jump, which tells the processor to stop fetching instructions from the instruction memory and execute them from the IRF instead. These jumps also use the index of the instruction as the jump target, instead of memory address of the instruction. If the execution flows outside the specified IRF block size, the execution resumes from the main instruction memory. If there is a normal jump, the execution resumes from the main instruction memory. The beginning of a IRF block is specified by special instruction, which also contains the length of the block. When this special

instruction is encountered when fetching instructions from the main memory, the following instructions will be stored to IRF while executing them. When a block of instructions does not contain any backwards jumps, it would be executed only once, it is not put into the IRF, but it is executed directly from the instruction memory, bypassing the IRF.

The support for IRF for validated for various applications and with different configurations within the FitOptiVis project, but still has room for research and development. Here are some code examples, which can or cannot be put to the loop buffer or the IRF:

```
// This whole first for loop can go to one IRF block,
// if the IRF is big enough.

// This loop count not be handled by the loop buffer
// due to the control inside.

for (int i = 0; i < N; i++) {
        if (A[i] % 1) {
                A[i] += 5;
        } else {
                A[i] -= 5;
        }
}

// the function call would cause an IRF block split,
// so that this loop cannot not be put to the IRF.
// it cannot be put into loop buffer either.

for (int i = 0; i < N; i++) {
        printf("%d ", A[0]);
}

// this whole loop can go to one IRF block, if there is enough space,
// as nested loop are allowed in the IRF.
// this whole loop could be put to the loop buffer due to the nesting.

for (int i = 0; i < N; i++) {
        // The loop buffer could only contain this inner loop.
        for (int j = 0; j < M; j++) {
                B[i*M + j] = A[i] * C[j];
        }
}
```

```
// this can be put to the IRF and while-loop buffer,
// but not into for-loop buffer, as the iteration count is not known
// before entering the loop.

// in case of an IRF, this could reside in the same IRF block
// as the code before or after this.

while (*a != 0) {
        a++;
}
```

Finally, mostly over the last year of the FitOptiVis project, *programmable instruction dictionary compression* support was researched and developed. An experimental, but validated, version of a programmable instruction dictionary was integrated in the toolset. The support is being validated with an example energy delay-product optimized DSP design that is being integrated to a test chip in another Finnish project focusing on refreshing regional SoC development and manufacturing skills and practices (see [5.2]).

Code compression reduces the amount of traffic between the processor and the energy-hungry memories that store the instructions. This is achieved by representing often occurring instructions in a shorter format. The method uses multiple parallel *dictionaries* to store often occurring parts of instructions. One of the novel aspects in this work is that these dictionaries can be updated on the fly during the program's execution, allowing enhanced compression ratios for the instruction mixes found in different program phases, leading to significant reductions in the instruction stream energy overhead. However, since the dictionaries are a static structure (for enhanced energy efficiency in comparison to dynamic caches), it requires analysis in the compiler.

Figure 36 presents an example of instruction placement in memory for the overall functionality of the dictionary compression. It demonstrates programmable dictionary compression flow and an example of instruction placement in memory. A publication of this work was made with an experimental implementation for the RISC-V ISA.

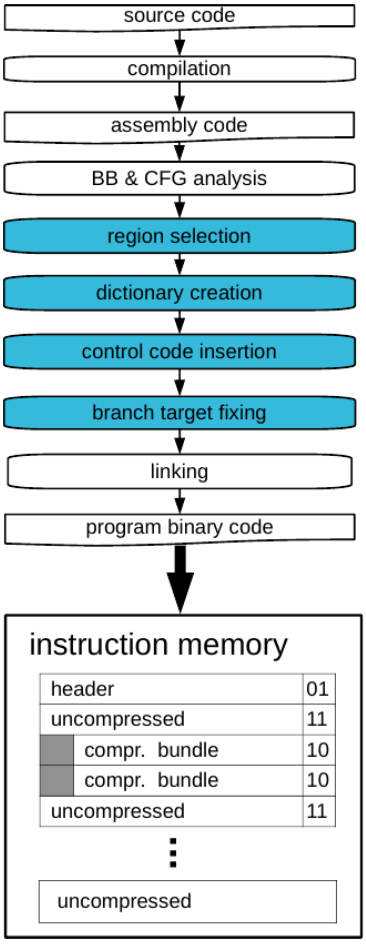


Figure 36: Programmable dictionary compression flow.

Figure 37 presents an example RISC-V ISA-based processor system (using the programmable dictionary compression before adapting it to the TTA architecture used in TCE.

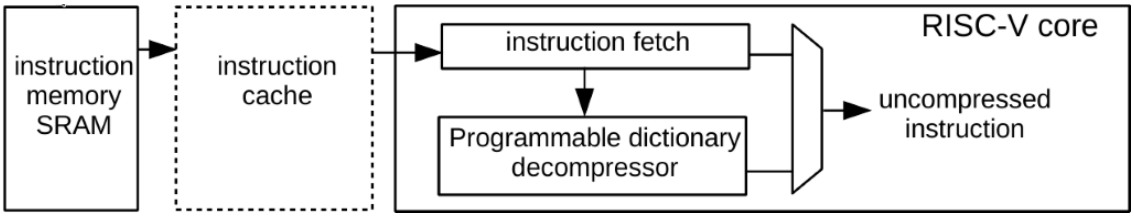


Figure 37: Example RISC-V ISA-based processor.

System presented in Figure 37 is using the programmable dictionary compression.

5.6. BlockCopier: A programmable block transfer unit

Perhaps, the most common bottleneck in FPGA execution is the available memory bandwidth. While the peak processing power of FPGAs is very large compared to, for example, a high-end CPU, the memory bandwidth is often similar. Furthermore, the

memory models required for conventional, hardware-controlled caches are difficult to implement on FPGAs.

Explicitly-controlled caches, where the data is selected and transferred to the cache by software, does not require such memory models, and ensures that the cached data is always relevant to the task at hand. It also eliminates cache misses, and thus decreases delay and throughput variance. This can help the application meet real-time constraints.

A simple approach for an explicitly controlled cache uses a portion of memory local to the accelerator, where the required memory can be transferred for the duration of the computation. Once the required data is present, the processor can access it within a more constrained time window, since cache misses are not possible. This can simplify the processor implementation, especially for statically scheduled processors.

The data transfer to and from the accelerator is usually handled by a direct memory access (DMA) controller. Most platforms, including the most common FPGA SoC chips, provide a DMA controller, but the interface and capabilities between platforms may vary.

For portability between platforms, we have developed a programmable block copier component, implemented as a TTA processor with a custom function unit capable of AXI burst transfers. The architecture for the TTA can be seen in Figure *38*. With minimal changes the same design would work on any AXI-based platform, and with a redesign of the custom function unit, other interconnect architectures could be supported as well.

Supporting high-level programming models like OpenCL can significantly ease the programming effort of TTAs, especially during processor and platform design space exploration. Abstracting data transfers between the host processor and the accelerator and internally between TTA accelerators removes some of the burden from the user, especially when the accelerators use local memories instead of or alongside caches. This could remove the need for long latency accesses to system-level memory.

Performed I integration of the block copier with the OpenCL runtime developed in WP4 has been an important step in ensuring ease of use of the accelerator platform. The primary target for improvement has been the signaling behavior.

The initial version supported rudimentary signaling – it supported postpone of the execution of a DMA transfer based on signals and broadcast a signal of its own, once a transfer has been completed – it relied on the host processor executing the OpenCL runtime to propagate those signals to the other devices.

Developed solution provides better handling of signals and moves the management of event waitlists onto the devices. This solution removes the event polling and propagation tasks from the host processor and frees it to perform useful computations, e.g. executing its own computational kernels.

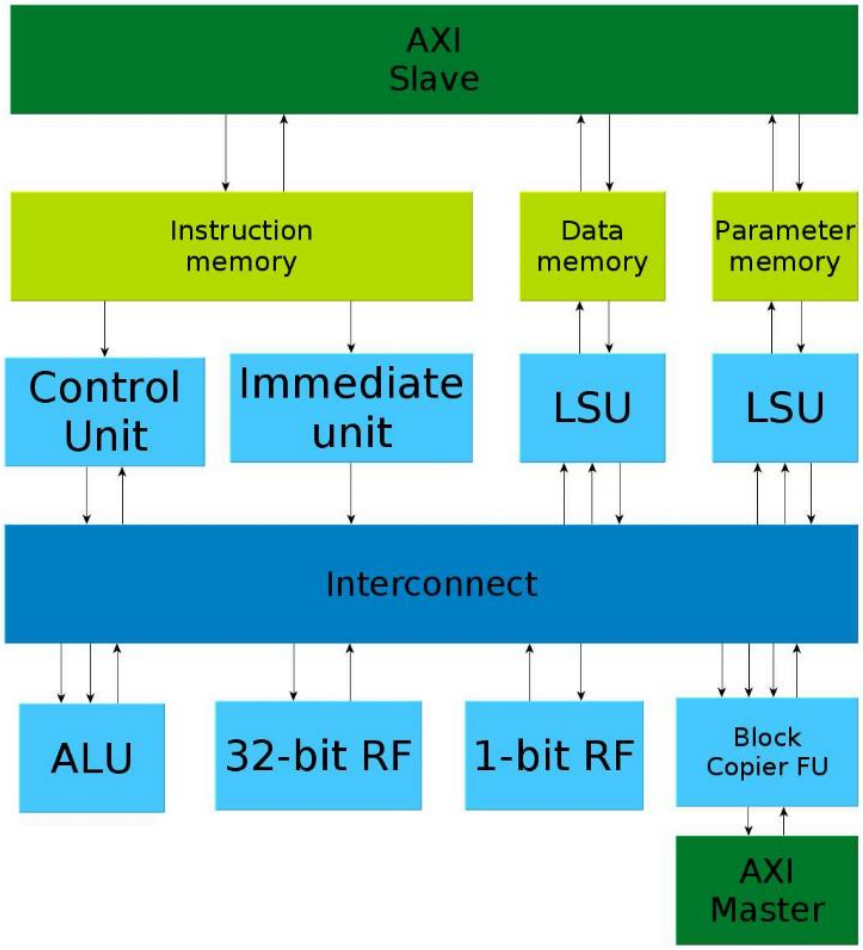


Figure 38: Architecture of the block copier ASIP.

5.7. Deterministic timing in distributed systems and latency control with Time Sensitive Networks (TSN)

This technology has been already described in Chapter 5.7 of D3.2.

5.8. Code generation for reconfigurable systems

From the WP2 QRML description, it is possible to generate a C++ system implementation. The generator produces C++ code that uses the RIE (Runtime reconfiguration Implementation of Embedded systems) library that was presented in section 4.7. The generator creates a C++ implementation template in which components are implemented as classes that make use of the RIE library to provide runtime reconfiguration and monitoring capabilities.

UC has also developed an UML/MARTE generator that transforms QRML/SDSL descriptions into UML/MARTE models, as shown in Figure 39. The generated UML/MARTE model is used as input of S3D tool for code generation as well as system verification purposes.

The generator produces C++ code with component class definitions. It also implements the component connections but the service implementations are derived to the system designers. In order to clarify the code generator features, the next table

presents the main transformations that are required to generate C++ code from a QRML system description.The generator produces C++ code with component class definitions.

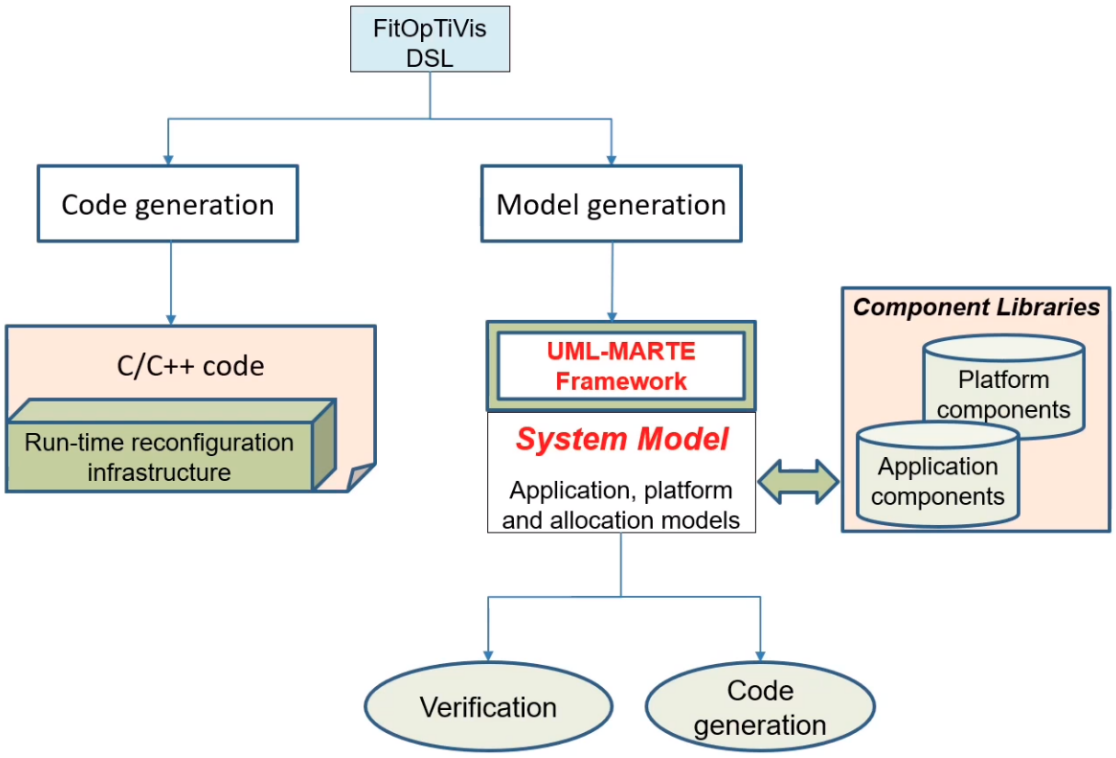


Figure 39: UC WP3 design flow.

QRML element	Generated code
Interface	C++ class with all its services declared as pure virtual methods in the component base class. The component implementations will implement the service functionalities.
Monitor	C++ class with all events defined as methods. The implementation of these methods depends on the tracer infrastructure (e.g: lttng implementation).
Component	C++ class deriving from RIE Component (to inherit runtime functions) and from provided interfaces. In case of component implementation, they also derive from the base component. Provided services are declared in the base class and implemented in the derived classes. Instead, required interfaces are declared as instances of the interface in the component class. Qualities and parameters are declared as variables in the class.
System	C++ class that includes system component instances and the connections among provided and required interfaces. It corresponds to the root component.
Channels	QRML channels are implemented with specific interfaces that provide stream-data read services.
Qualities and parameters	The qualities and parameters are implemented in component class members.

Table 2: VLAN identification rules of user traffic types.

It also implements the component connections, but the service implementations are derived to the system designers. In order to clarify the code generator features, the next table presents the main transformations that are required to generate C++ code from a QRML system description.

The next figure presents an example of code generation from the QRML language. The “linkComponent” function is also automatically generated. The function assigns components to instances and connects required and provided services.

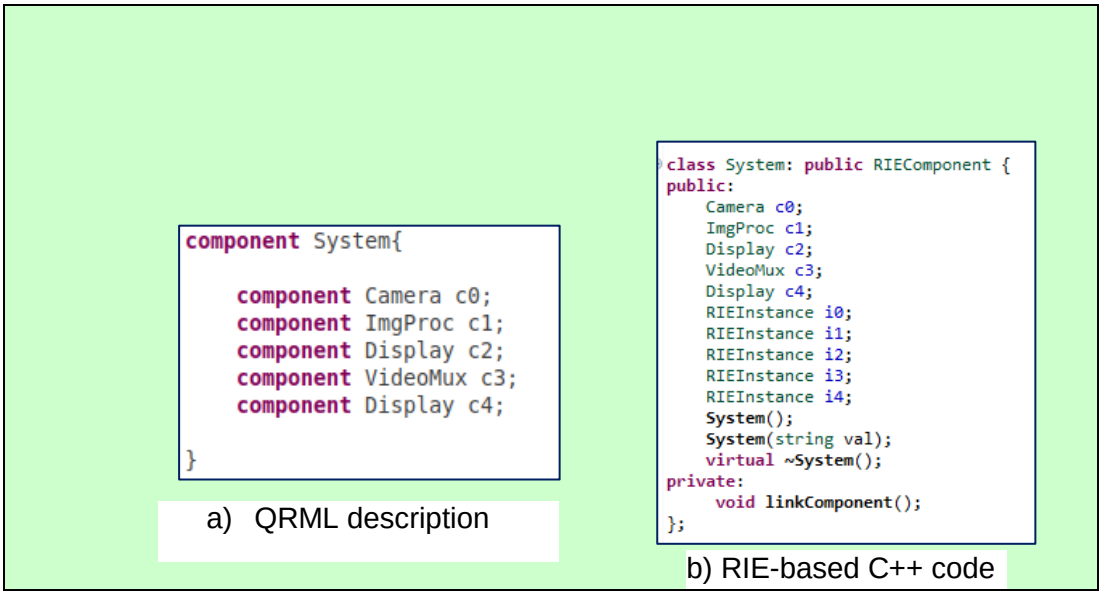


Figure 40: Automatic code generation.

The automatic tool generates a component library model for describing and modelling all application components described in SDSL description. In Figure 41 we can see a UML/MARTE component model that was automatically generated

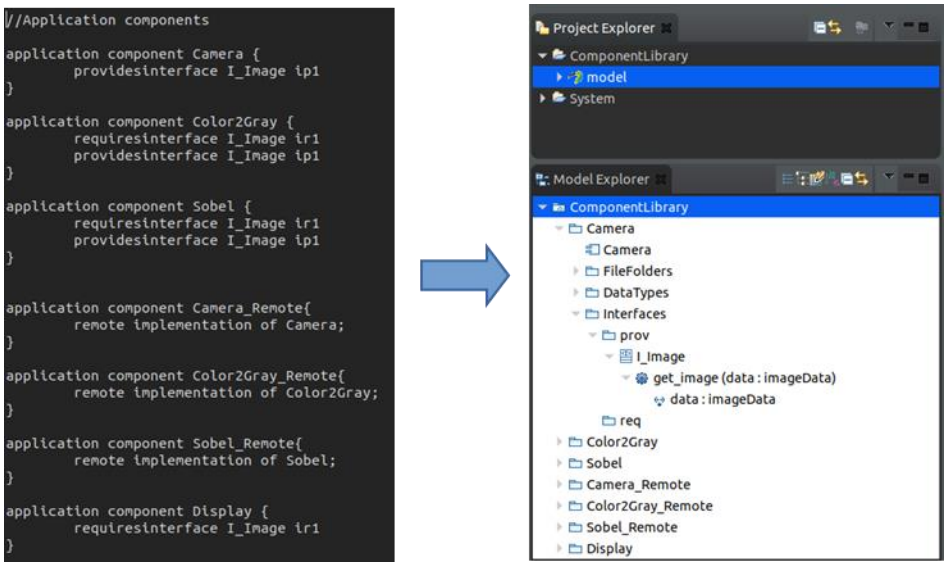


Figure 41: Automatic generation of component library.

6. Acceleration support

This chapter describes HW accelerator-oriented design flows and programming techniques that are being developed on task 3.3. The first part presents tools that generate HW accelerators from high-level programs (OpeMP, C++ and data flow descriptions) or for specific architectures (TTA-based soft processor). The second part presents HW generators that are oriented to particular applications as well as specific acceleration techniques.

6.1. OpenMP for HW accelerators

Two approaches have been used to implement HW accelerators with OpenMP. The first approach directly modifies the OpenMP code to support FPGA synthesis. The second approach uses the developed OpenMP target offloading. The first approach is used with Xilinx SDSoC (version 2018.3), while the second is used with the last Vitis versions (version 2020.2). For Xilinx SDSoC design flow, the OpenMP and the SDSoC oriented code cannot be in the same file because the Xilinx synthesis tools do not support the OpenMP directives. The next figure presents the code of both files.

The HW accelerators (target devices) are controlled by the system processors (hosts) that require their services to execute specific functions. The accelerators normally have a private memory and limited access to the processor main memory. Therefore, the processors have to transfer data from/to the program memory space to the accelerator memory before/after accelerator execution (copy-in/copy-out model). This protocol is explicitly implemented in OpenCL and it is implicit in OpenMP.

```
#pragma omp target data \
 map(to:a[0:size], b[0:size]) map(from:c[0:size])
    {   // Transfer data from host to accelerator
    #pragma omp target device(1)
        mulmat1(a, b, b1);
    }
```

a) OpenMP Code

```
#pragma SDS data copy(A[0:SIZE])
#pragma SDS data copy(B[0:SIZE])
#pragma SDS data copy(C[0:SIZE])
#pragma SDS data mem_attribute(A:NON_PHYSICAL_CONTIGUOUS, B:NON_PHYSICAL_CONTIGUOUS)
#pragma SDS data mem_attribute (C:NON_PHYSICAL_CONTIGUOUS)
void mulmat1(int A[SIZE], int B[SIZE], int C[SIZE]{
    int res;
    for(int i=0;i<N; i++)
        for(int j=0; j<N; j++ ) {
            res=0;
            for(int k=0; k<N;k++)
                res += A[(i*N)+k]*B[(N*k)+j];
            C[(N*i)+j]=res;
        }
}
```

b) Xilinx SDSoC code

Figure 42: HW accelerators with OpenMP code.

FPGA-based accelerator normally implements additional communication models. For example, SDSoC from Xilinx provides direct access to the software memory space from the accelerator or data streams. The shared memory model normally has an important disadvantage: the latency to the external non-cacheable memory, in which the shared data are stored, is typically higher than it is for the CPU. One way to

minimize the low performance of the shared variables is to access memory in a sequential way. For this reason, hardware accelerator design tools (e.g. SDSoC) recommend use of shared variables with sequential access based on DMA (Direct Memory Access).
The OpenMP accelerator strategy developed in WP3 by UC in FitOptiVis in Y3 itried to reduce these overheads.

6.2. HW accelerators generated by the Xilinx SG for DSP and SDSoC system level compiler

In Y3, UTIA updated design time support serving for design time integration of:

- IP blocks with streaming data interfaces. Blocks are generated in Xilinx SG for DSP Matlab/Simulink toolbox. See section 10.8 of this deliverable “Design Time Resource Integrator of Model Composer IPs (DTRiMC) technology”.
- HW data-movers connecting the Xilinx System Generator for DSP and Model Composer IP blocks to Arm A9 processing system on Zynq and for A53 processing systems on Zynq Ultrascale+ devices. These data movers are generated by Xilinx SDSoC 2018.2 system level compiler as part of the automated kernel compilation process. It is using the Xilinx Vivado HLS 2018.2 tool.
- The user defined IP blocks in C or C++ source code and OpenCV algorithmic libraries are compiled by the Xilinx SDSoC 2018.2 system level compiler and by the Xilinx Vivado HLS 2018.2 design flow.

This process is automated by UTIA DTRiMC tool. See Chapter 7.1 -7.4 for details.

In Y3 UTIA developed evaluation packages for the DTRiMC tool compatible with these versions of accelerators generated in Xilinx System generator for DSP:

- 8xSIMD fp01x8 HW accelerators with capabilities = 10, 20, 30 or 40 Zynq 7000 family of devices (without support for FP32 8xSIMD division).
- 8xSIMD fp03x8 HW accelerators with capabilities= 10, 20, 30 or 40 for Zynq Ultrascale+ family of devices (with support for FP32 8xSIMD division).

Implementation details of these run-time reprogrammable accelerators are described in D5.3. SW description and design time use of these of these run-time reprogrammable accelerators are described in D4.3.

In Y3, HW platforms supported by the DTRiMC tool include these HW accelerators:

- Edge detection accelerator based on Sobel filter in SW and in HW
- Canny edge detector in HW
- Motion detection accelerator based on two Sobel filters in SW and in HW
- Lucas Kande Dense Optical Flow accelerator in SW and in HW
- Object tracking HW accelerator (tracking of colour and position of four balls)

The **Y2 UTIA Design Time Resources** released in [7.12], [7.13] contained precompiled HW design in form of precompiled dynamic (.so) libraries. These libraries represented fixed HW. These libraries was dynamically linked to C or C++, user defined, applications for Debian OS running on the ARM A9 or A53 processor.

The **Y3 Design Time Resources** for the DTRiMC tool [7.15]-[7.20] contain HW project design, which can be extended by the user with additional custom HW. DTRiMC tool scripts support the compilation to dynamic (.so) libraries. These libraries represent fixed HW with user defined extensions. In the next DTRiMC tool supported stage, these libraries are dynamically linked to C or C++, user defined, applications for Debian OS running on the ARM A9 or A53 processor.

- PC cross-compiler can be used for compilation of the user-defined C or C++ SW application in the free Xilinx SDK 2018.2 Eclipse-based framework. Remote debug of the application on the target device is also possible for modules with Ethernet.
- Embedded gcc or g++ compiler can be used for compilation of the user-defined C or C++ application directly on the embedded device (Zynq A9 or Zynq Ultrascale+ A53 processing system).

These UTIA design time resources developed, documented and released in Y3 are summarised in Chapter 10.8 of this deliverable as the “Design Time Resource Integrator of Model Composer IPs (DTRiMC) technology”. Technical details are described in Chapters 7.1 – 7.4 of this deliverable.

6.3. The Multi-Dataflow Composer (MDC) tool: a dataflow-to-accelerator design suite

The Multi-Dataflow Composer (MDC) is a software tool, or rather a suite of different design features, for the automatic generation and management of coarse-grained reconfigurable systems and accelerators based on the dataflow Model of Computation. MDC main purpose is supporting software developers/embedded system engineers and/or hardware architects/embedded system engineers in defining flexible and performance-aware coarse-grained reconfigurable substrates, which can be embedded into FPGA-based hardware accelerators. The key features of this tool are:

- the ability to combine different high-level dataflow specifications, describing a set of functionalities, into a single accelerator, exploiting coarse-grained reconfigurable technologies and capable of accelerating all the provided functionalities
- automatic resource minimization
- transparent (to the user) reconfiguration management

The MDC features are:

- Baseline MDC Core – performing dataflow-to-hardware composition, by means of data-path merging techniques.
- Structural Profiler – performing the design space exploration of the implementable multi-functional systems, which can be derived from the input dataflow specifications set, to determine the optimal coarse-grained reconfigurable substrate according to the given input constraints.
- Dynamic Power Manager – performing, at the dataflow level, the logic partitioning of the involved resources to implement at the hardware level power- and clock-gating strategies and, in turn, to save both static and dynamic power consumption.

- Co-Processor Generator – performing the complete dataflow-to-hardware customization of a ready-to-use Xilinx compliant multi-functional accelerator IP. Starting from the input dataflow specifications set, such an accelerator can be either loosely coupled or tightly coupled, according to the user needs, and also its drivers are derived.

The inputs are:

- high level models (dataflow) of functionalities to be accelerated - XDF, Cal
- HDL description of the components (HDL Components Library, HCL) corresponding to the dataflow actors, manually or automatically generated - Verilog, VHDL
- hardware communication protocol between components - XML

And finally, the outputs are:

- (baseline functionality) HDL description corresponding to the multi-functional dataflow model - Verilog, VHDL
- (optional) multi-functional model resulting from the combination of the input applications models - XDF, Cal
- (optional) Xilinx IP wrapper logic, scripts and drivers - XML, Verilog, Tcl, C

MDC is available open source on GitHub, with a BSL 3-clause licence. Here in after the MDC useful links are provided in [6.8]:

MDC has been used, within the Water Supply use case, to generate accelerators for image classification (WP6 activities) and, contemporarily, it has been connected to the AIPHS monitoring infrastructure (WP4 activities). With respect to the former activity, the cooperation with AITEK has been established. In particular, AITEK in the last project phase will assess and compare the new accelerators with traditional implementation, providing relevant feedback to UNICA and UNISS for future improvements of MDC tool.

State of the art algorithms for image classification have been used with implementation in FPGA-based accelerators obtained using the MDC tool. AITEK provided such algorithms as Convolutional Neural Networks (CNNs) in ONNX format, which has been firstly translated into the corresponding C source code thanks to the ONNX2C flow, which is part of the NeuDNN software stack (see Section 6.4). Such C source code has been used for implementing the CNNs: the source code coming from the ONNX2C flow has been used as input point for the Vivado HLS tool, in order to derive the HCL required by MDC. So that, the CNNs have been described as dataflow models according to the initial ONNX structure and taking as HCL the one generated by Vivado HLS from the C source code corresponding to the same ONNX description. These activities have been completed in Y3. On the top of this setup, several versions of the CNNs have be derived and combined together by MDC, enabling multi-functional CNN hardware accelerators capable of playing with the different CNN versions.

We have been using as a possible metric for evaluating the different accelerators the execution time of CNN algorithms, processing images with different resolutions. To have a complete benchmarking, different processors have been tested. We have collected different execution times to be compared with the execution time achieved thanks to the accelerator.

In Y2, UNICA and UNISS enhanced availability of the MDC tool, which has been provided with

- a starting pack for easy and quick testing;
- extended documentation and open source diffusion;
- tutorials to getting familiar with MDC features and application fields;
- internal and external assessment has been planned, set-up and continued also during Y2 of the project
    - regarding internal assessment, UNIVAQ and AITEK are playing a central role: besides its usage within the Water Supply use case, MDC has been lately (M18) assigned to some UNIVAQ students to carry out their projects within the “Embedded Systems” course of the Laurea Degree in “Telecommunication Engineering”;
    - regarding external assessment, MDC is used in other regional [6.9] and EU projects [6.10].

In Y2, we also worked on extensions of MDC to:

- we have developed support for accelerators monitoring with AIPHS
    - a proof of concept has already be developed and the achieved, results are about to be submitted to a scientific journal,
    - automation of the whole accelerator deployment plus monitoring is still currently ongoing
- we have progressed with the development for supporting the ALMAIF front-end. In this regard, plans are there, but the activity will start at completion of the integration with AIPHS.

In Y3, MDC has been adopted in an almost fully automated toolchain, together with some features of NEURAghe and with Vivado HLS from Xilinx (details are provided in section 6.11). This toolchain is intended for supporting designers of neural network adaptive hardware accelerators, going from widely adopted application development frameworks down to the corresponding accelerator implemented on FPGA. The toolchain has been adopted within the scope of UC1 to provide an adaptive accelerator for a convolutional neural network capable of classifying humans and animals.

6.4. NEURAghe a flexible and parameterized CNN accelerator

NEURAghe is a hardware/software solution for the acceleration of Convolutional Neural Networks (CNNs) on Xilinx Zynq Systems on Chip (SoCs). In particular, it exploits both the hard-core ARM processors and a Convolution-Specific Processor (CSP) deployed on the configurable logic. As a result, the ARM processors are in charge of supervising the acceleration and of executing the hard-to-accelerate parts of the computational graph, while the accelerator takes care of the bulk of CNN workload and can be controlled by software at a very fine granularity.

The acceleration hardware is supported by a software stack, NEURAghe Deep Neural Network software stack (NeuDNN). NeuDNN allows the user to develop and reuse CNNs to be accelerated with the NEURAghe solution. It runs on top of Linux OS in order to favour system integration and it is basically constituted by a configurable C/C++ library, providing APIs to the user in order to seamlessly execute the CNN with or without acceleration, and by drivers (ARM-side) plus a resident runtime (CSP-side),

the former sending commands to the latter that properly executes them on the acceleration logic. Besides this, some extensions of the NeuDNN software stack are ongoing in order to provide automated conversion from ONNX (NN widely used formalism) to C with NEURAghe API calls, and to provide configuration of a C template with NEURAghe APIs starting from a darknet (NN state of the art framework) high level network configuration.

NEURAghe also offers several configuration points at design time, making it extremely flexible. Indeed, it is possible to configure:

- Data precision for input/output pixels, biases and weights (16 or 8 bits), providing a compromise between accuracy and performance;
- Baseline CNN hardware acceleration core size (sum-of-product units matrix size);
- Number of acceleration clusters (each cluster is independent from each other and can have its own baseline CNN hardware acceleration core size);
- Memory size of each cluster.

The inputs are:

- CNN host code or ONNX NN specification or darknet network configuration
- Target Xilinx Zynq SoC (among Z-7045, Z-7020, Z-7007s)

The outputs are:

- Zynq-based CNN hardware/software acceleration engine
- CNN host code with NEURAghe API calls (possibility of offloading computation to the acceleration engine)

According to the perspective adoptions in the FitOptiVis use-cases, NEURAghe and NeuDNN will be refined in particular to provide:

- model-based optimization of the scheduling of CNN actors on available processing elements (Task 3.1).
- implementation of dynamically
- variable precision computing in convolution cores, thus realizing different set points for the CNN accelerator (Task 3.2).

NEURAghe, constituting a CNN accelerator provided with the NeuDNN software stack, will be also part of the model-based working technology supporting the FitOptiVis design platform (Task 3.3).

State of the art algorithms for image classification are under evaluation on the NEURAghe platform. AITEK provided such algorithms as CNNs in ONNX format, which has been firstly translated into the corresponding C source code thanks to the ONNX2C flow, which is part of the NeuDNN software stack. Such C source code has been used for implementing the CNNs through the NEURAghe platform. The source code coming from the ONNX2C flow has been automatically populated with proper function calls to configure and manage the processing offloading on the NEURAghe CNN accelerator. In this activity UNICA and AITEK provide respectively the target platform and the applications. The implementation of the accelerators, with the support of both UNICA and AITEK, has been carried out by UNISS, which is assessing the ONNX2C flow. At UNISS in Y2, the accelerators have been under evaluation, and compared with AITEK proprietary implementations. Assessment at UNISS has been completed in June 2020.

The application provided by Aitek consisted of three different Neural Networks that have been detecting moving targets and distinguish between persons and animals. This has been a requirement specifically elicitated in Use case 1. In the first scenario it was needed to detect possible human intruders and limit false alarms caused by animals entering the same restricted area.

All the three provided networks processed 128x128 RGB images as input, providing detection and classification as output. They differ for the implemented architectures (i.e. VGG, Inception and MobileNet architectures). Moreover, they have been able to achieve different levels of accuracy; they are characterized by different complexities and require a specific amount of computational resources.

In Y3 NEURAghe, and in particular its ONNXparser (formerly ONNX2C) feature, has been adopted in an almost automated toolchain, together with MDC and with Vivado HLS from Xilinx (details are provided in section 6.11). This toolchain is intended for supporting designers of neural network adaptive hardware accelerators, going from widely adopted application development frameworks down to the corresponding accelerator implemented on FPGA. The toolchain has been adopted within the scope of UC1 to provide an adaptive accelerator for a convolutional neural network capable of classifying humans and animals.

6.5. TTA-Based customized soft core accelerators

Transport-triggered architectures (TTA) are a promising avenue in the field of soft processors. Compared with a traditional operation-triggered architecture, TTA has a simpler implementation, leading to lower logic requirements and higher frequency. Furthermore, the instruction encoding describes explicit parallelism without requiring complex decoding logic.

However, the processor design toolset for TTA-based co-processors, TCE, was primarily targeting ASIC implementations. FPGA architectures are more constrained in their logic, memory and routing resources. While the fine-grained logic components and their associated registers can theoretically implement any digital logic circuit, specifying the logic in such a way that it maps to the special-purpose blocks leads to significantly better synthesis results, both in terms of area and frequency.

These special-purpose blocks vary in complexity, from the ripple-carry logic or multiplexers associated with the look-up tables of the fine-grained logic to the pipelined multiply accumulate blocks with internal feedforward paths. The memory is similarly constrained: the high-density hardened memory blocks in modern FPGAs feature at most two bidirectional ports, and while the read port count of the smaller memories can be higher, they are limited to a single write port. This makes the implementation of complex memory components, particularly those required for dynamic caches, difficult on FPGAs.

We set out to optimize the individual components of our TTA implementation for FPGA devices. First, the interconnection network was examined. A complex interconnection network can be the largest individual component in a TTA processor, and it may affect the critical path within any function unit as their logic can be moved across the registers to the inter-connect or vice versa. Therefore, its efficient implementation is

paramount to a high-performance TTA design. The default implementation did not map efficiently on to FPGA hardware.

For the FPGA optimization, the input socket side of the inter-connect, originally implemented with an AND-OR network performing what is essentially a multiplexing operation, was replaced with an explicit switch-case structure in the RTL code. In addition to mapping better to the dedicated multiplexing logic of the FPGA device, the decode process needs to examine the source fields of a single bus, rather than the source fields of every bus a given input socket is connected to. This reduces the number of inputs to the logic function required to determine the control signals and, subsequently, the number of logic elements required to implement it.

The load-store unit (LSU) optimization was somewhat more straightforward. For scalar LSUs, the logic implementation had nothing specifically designed for FPGAs. However, lock signals are an issue on FPGAs, as they have a high fan-out, essentially enabling or disabling every function unit pipeline register. Therefore, fixed-latency LSUs are a better fit for FPGAs. This also discourages us from using dynamic caches, opting for scratchpads memory instead. For vector LSUs with a wide external bus, the bottleneck was found to be the word select from the wide read word to the scalar-width output. This can be alleviated by separating the scalar data output to its own port and increasing the architectural latency of scalar loads. Another approach allows us to get completely rid of the word select multiplexer. This can be achieved by having 2 separate different-sized LSUs connected to the same address space. We tested this by arbitrating the second port of the dual-port block ram between external AXI access and TTA's scalar LSU. Area improvement was significant with this approach.

The optimizations have been integrated to the TCE toolchain and can mostly be enabled without modifications to the processor architecture. Some recommendations, primarily those concerning LSUs, may require architectural changes. While the changes were aimed primarily for FPGA implementations of TTA processors – especially the modifications to the interconnect implementation – may also aid ASIC synthesis tools to reach better results.

The FPGA-centric optimizations were evaluated through synthesis on TTA processors with and without each optimization to determine the individual effects of the changes. The biggest difference was found to be from the interconnection network optimizations, where the network itself required up to 54 % less logic to implement with the optimizations than without. Taking all the optimizations into account, the logic utilization of the entire core was reduced by up to 30 %.

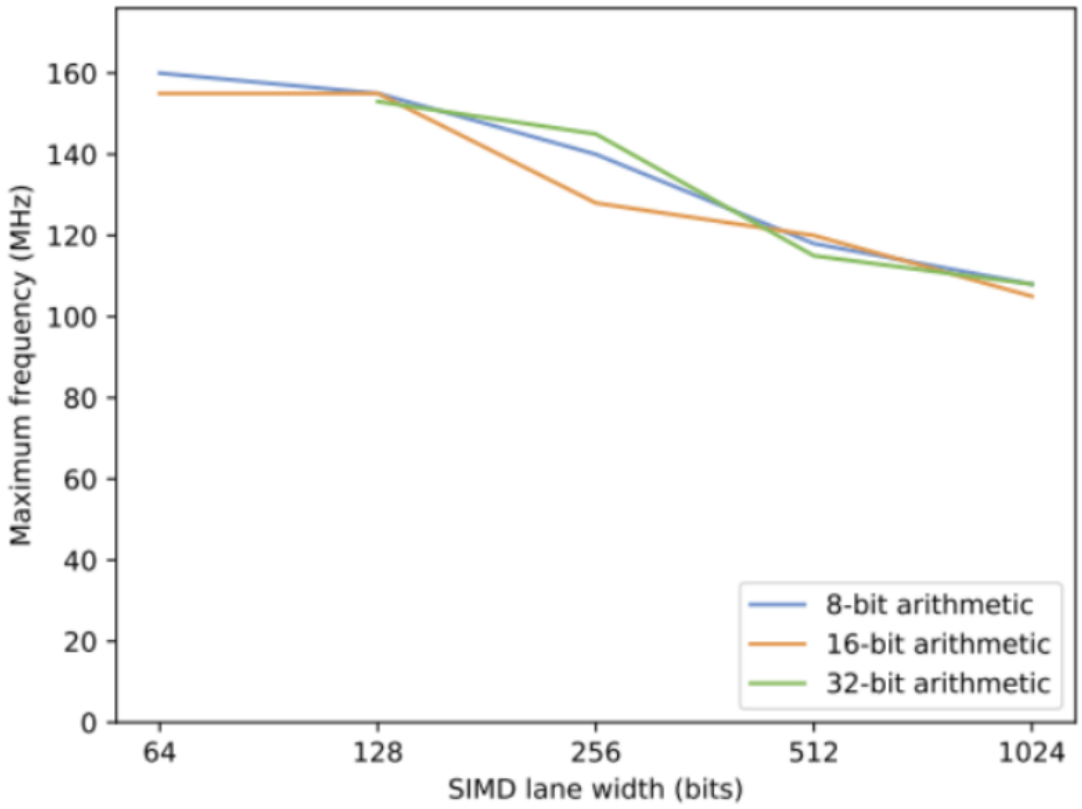


Figure 43: Maximum clock frequency of the synthesized processors.

Vector function units are an easy way to exploit the data level parallelism on programs. To this end, we evaluated the scalability of the TTA approach to large SIMD widths. The most important function units to vectorise are the load-store unit and the ALU. The ALU can utilize FPGA's hardened DSP blocks in parallel to implement efficient MUL and MADD operations.

Overall, SIMD processors share a challenge of complex inter-lane connectivity which is needed when passing data between the vector lanes, typically using so called "shuffle operations". To minimize the impact of a *complex shuffle network* required by a fully dynamic shuffle unit (runtime defined vector indices for the lane data), we implemented a few preselected static shuffle patterns based on the needs of the program, which matches the idea of a reduced programmability layer on top of the very dynamic FPGA fabric. Communication between scalar and vector busses was implemented with broadcast and element extraction function units.

By the end of project, we managed to show that the TTA-SIMD approach can quite easily scale up to 1024-bit SIMD lanes with over 100 MHz clock frequency on a small and cheap FPGA (Zynq 7020 SoC of the PYNQ board). Figure 43 shows the clock frequency trend with different lane widths up to 1 kbit vector width.

A performance comparison against the ARM hard processor system with NEON instruction set integrated on the same SoC showed that we can reach up to 2.4x speedup in some workloads, overcoming the 650 MHz clock frequency advantage of

the ARM core with additional data level parallelism. The benchmarking was done using OpenCL vector datatypes, so we simultaneously demonstrated the easy OpenCL programmability of SIMD-TTA processors in our platform.

In further analysis work in the final year of FitOptiVis, we found several low hanging fruits related to critical paths which will bring additional clock frequency improvements. However, it was also made very clear that beating CPUs and GPUs with FPGA-based SIMD workloads on standard arithmetic is a major research challenge. For example, the recent ARM processors introduce SVE that has better support for SIMD, after which it becomes even more challenging to demonstrate speedups with FPGA-based designs for data parallel workloads. However, it seems FPGA-based implementations might be able to demonstrate better energy-efficiency in cases where it's possible to trade more parallelism to lower clock frequencies (and possibly lower operating voltages).

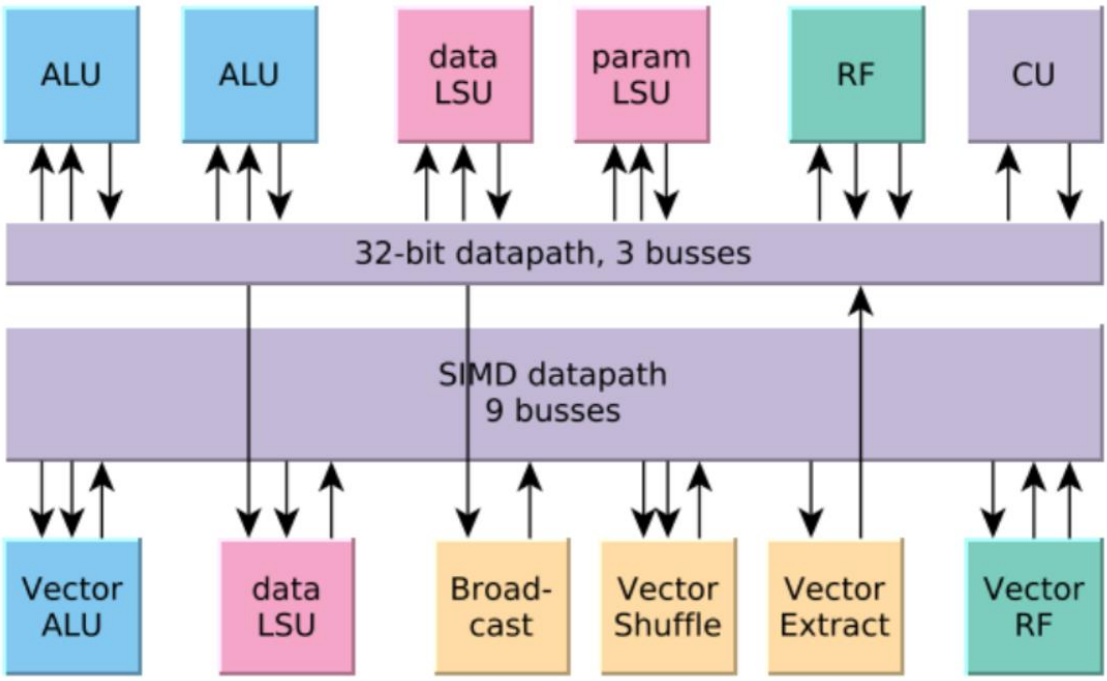


Figure 44: Simplified view of the wide-SIMD TTA template.

In order to evaluate the scalability of the TTA SIMD template to larger FPGAs and to demonstrate a real-life scenario, we presented a case study with application-specific optimizations targeting CNN inference.

Since TTAs present excellent instruction-level parallelism scalability potential, and also the SIMD usage shows a lot of promise, the final common degree of parallelism, that is, the thread level parallelism would complete the template in its goal of exploiting all forms of concurrency available in the application description for performance benefits. To this end, there was a research track where the homogeneous multicore support, initially added to the TCE toolchain over 10 years ago, was optimized for FPGA soft core use.

A master's thesis project of Topi Leppänen was supported with FitOptiVis funding to study the immediate scalability bottlenecks of TTA-based multicores on FPGA fabrics. The master's thesis work on purpose did not utilize the SIMD support, but used very small scalar cores to find the multicore scaling bottlenecks as the first priority.

The intermediate result of this thesis work was an 8-fold increase in performance achieved with 24 cores with scalar function units, compared to an equivalent single core system. The external memory bandwidth could be utilized with 11.4% efficiency even with the scalar load-store units in these cores. The remaining bottlenecks of the multicore soft processors were reported in the thesis which are being solved to unleash the full potential of customized multicore systems, potentially already within the FitOptiVis extended timeline. However, the most important bottleneck identified was not truly multicore related, but simply coming from the fact that scalar LSUs are not ideal when attempting to utilize wide memory buses and their burst modes: the scalar access to the external memory was shown to be a very inefficient way to utilize the external memory bandwidth, thus combining SIMD and multicore soft cores should yield the best utilization results, which is being studied currently.

Multicore scalability of the TTA SIMD approach was initially demonstrated on a Zynq UltraScale+ board in the beginning of the FitOptiVis project. In this experiment we fit 14 customized SIMD cores reaching up to **48.5 GOPS** total real application performance while running a face-detection neural network. However, co-optimizing combining SIMD and multicore for FPGA soft core use was left for the future work at this stage.

AutoExplorer (AEx)

The design space explorer tool is a part of the TCE framework. Its purpose is to run various exploration algorithms defined as plugin modules to find best possible architecture configurations for a given application. All exploration results are stored in a database as configurations in terms of processor architectures and its cost (clock cycle count, area, power). Each result is verified using framework compilation and simulation tools.

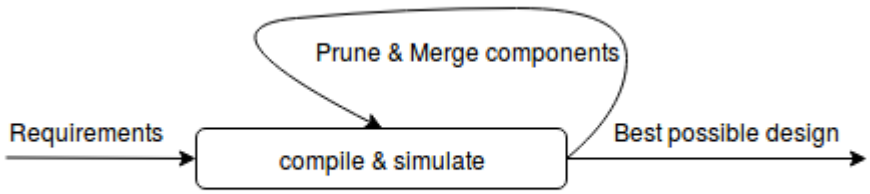


Figure 45: Simplified TCE Exploration process of AutoExplorer.

At the beginning of the exploration the application and the requirements are given by the processor designer. Usually starting point is some architecture configuration that can be compiled and simulated. The configuration architecture is later modified by merging or pruning its components producing multiple different designs that meet the requirements or improve the performance. A specific estimator algorithm is used to select best configuration that meets the requirements. The exploration is finished until there are no better configurations that can be generated.

AutoExplorer (later AEx) is a design space exploration flow researched within the FitOptiVis project. Its goal is to generate application specific processors by analyzing the application automatically. Processor designer can specify multiple design restrictions given as parameters. Several different algorithm stages are performed to produce several possible architecture component combinations by pruning and merging them and return them as configuration ids. AEx drives multiple exploration plugins in a sequence, and picks best possible configuration from each stage based on estimation information and uses configuration id as an input for the next plugin stage. The advantage of the automated exploration is that it can evaluate several hundreds of different designs before finding optimal solution. This allows automated rapid prototyping of different architectures for specific application set.

First, the algorithm creates huge processor architecture with all possible operations found from the TCE hardware database. For each operation depending on a given parameter one or more function units are created. The register files are set to the huge enough size to avoid register spilling. The purpose of this stage is to create starting design exploration point, where pruning and merging of components can begin.

In the next stage, the application is compiled for the architecture generated previously and simulated. From the compiled application we can analyze which operation are being used and prune the function units and register files of certain width which are not used. This simple trick greatly reduces the compilation and simulation times for further exploration stages. We can reduce operations even further by analyzing simulation results and prune function units for operations of which execution times are under certain threshold. Several operations such as multiplication, of which usage might be below the given threshold are given a higher priority, so they are not removed declining the results.

After unused components are pruned, we create a VLIW-like connected architecture, where each function unit input and output ports are connected to the register files. It results in a huge interconnection network which will be reduced in the later stages by merging function units, buses and ports. Also the dummy unconnected bus is created to provide the slot for long transfers.

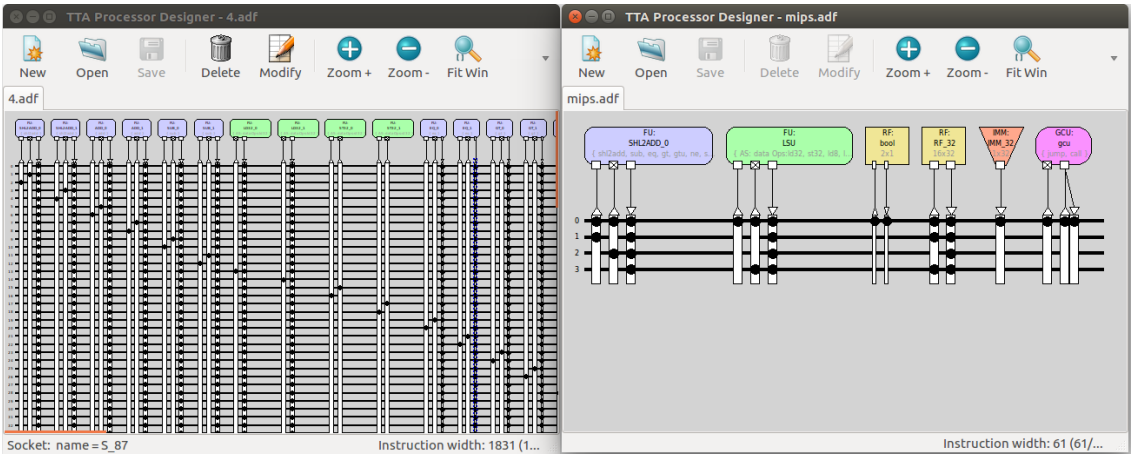


Figure 46: Un-optimized architecture (left), final best possible architecture (right).

Figure *46* depicts a part of the huge VLIW-like architecture where the components are not yet pruned and merged and the final desired result of the auto-exploration.

The next stage is optional and it simply splits the register files into two parts.

Several function units might not be used simultaneously and can be merged together. The algorithm produces the covariance matrix for the function unit executions from the previous simulation results and merges the function unit pair with the lowest covariance. The compilation and simulation process is repeated until all function units are merged into one. The AEx's job is to estimate the best amount of merged function units and to pass the configuration id to the next exploration plugin.

To further reduce the architecture, the buses and register file ports are also merged based on the same covariance matrix calculation algorithm each after another. The exploration stage ends until there is one bus left and register files with one write and read port. AEx here again estimates the best combination of buses and register files ports. After this the merging is done and the architecture looks much simplified than at the beginning of exploration.

At the beginning we set register file sizes huge enough to avoid register spilling. At this stage the size is reduced and simulated until it does not affect the performance significantly.

To inflate the instruction word size even further we split long immediate bus over all buses in the architecture. This is the final stage where the best possible architecture for the application is generated and can be further optimized manually by designer.

The processor architecture can be then fed into the platform integrator tools to generate it into the hardware description language and generate program image for FPGA verification tools like Vivado.

AEx2: User Inputs Only the Target Frequency and Target Execution Time

The next generation of AEx we call AEx2 was developed in the latter part of the FitOptiVis project. The overall goal for the new algorithm was to simplify the usage so that the end user simply defines the desired target frequency and execution time parameters to efficiently prune design space configurations, which cycle count does not fit. At each pipeline pass, only suitable configurations are left and the ones with the minimal resource usage are picked for the next pass (phase) or selected as the final architecture presented to the designer. If at some point of the pass pipeline there is no single suitable configuration that can deliver the targeted execution time with the given target clock frequency, AEx2 backtracks to the previous pass and picks configurations with more hardware resources. It can fall back through multiple passes until a suitable configuration is selected, or report of an error saying that there are no single fitted configuration could be found that suits the designer's input parameters. This process is illustrated in Figure 47.

This heuristic slightly increases the design space, but it gets rid of "magic threshold numbers" in the first AEx version, which were hand-picked based on empirical observations, and used in several algorithmic passes to prune configurations that do not fit the cycle count.

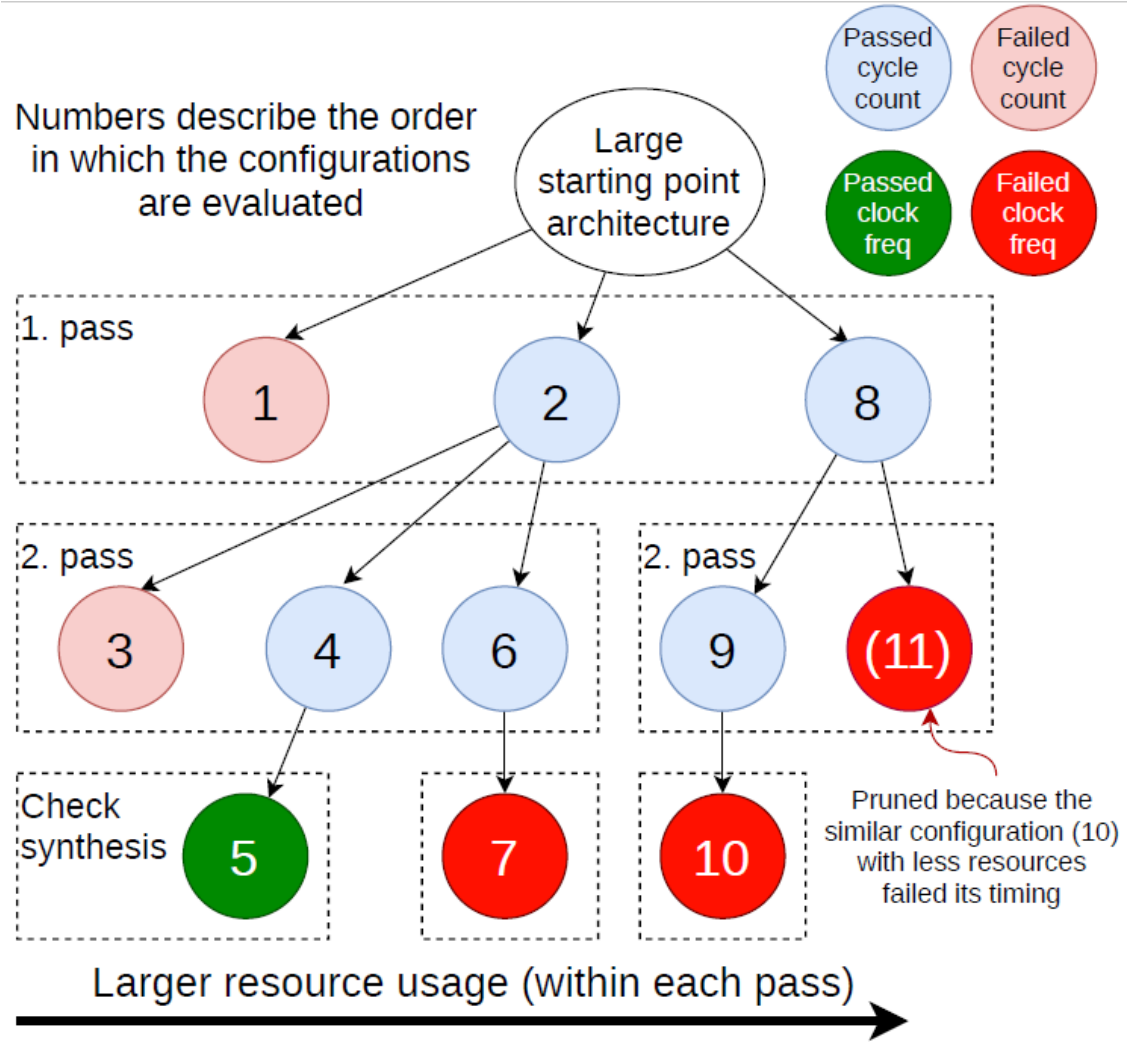


Figure 47: AEx2 result pruning between passes.

The configurations marked in Figure 47 in green can reach the targeted execution time with the targeted clock frequency.

AEx2: SIMD processor generation with LLVM auto-vectorization

The auto-generation of efficient SIMD architectures was implemented utilizing LLVM's auto-vectorization support. LLVM's offers two auto-vectorizers, one that operates on loops and the SLP vectorizer which works on basic block level. Both focus on different optimizations and use different techniques. Loop vectorizer widens instructions inside the loops and handles multiple consecutive iterations simultaneously. SLP vectorizer simply merges several scalar operations it finds inside a basic block into a vector operation. If the vector operations are found from in generated LLVM IR code, then during the operation pruning pass these operations are added to the generated architecture and their particular sized register files. Initial tests show around 20% improvement in execution time for certain CHStone tests using vector architectures.

AEx2: "I/O-Skeleton" Starting Point Architectures

To support different memory interfaces, the exploration can be started using a pre-defined architecture "skeleton" containing only some function units with specific operations, delays and address spaces. The skeleton approach is useful when integrating the produced core to a predefined system bus or memory hierarchy: In that case it can contain mainly the load-store units used to access the addresses through the buses. While creating the initial huge architecture, other function units for operations are simply added to this predefined architecture, without adding existing operations found in predefined units. The predefined function units are kept untouched during FunctionUnitMergePass, while other units can be merged based on their parallel usage.

AEx2: Miscellaneous Improvements vs AEx

- During the VLIW-connectivity pass, additional connections from each function unit to the boolean register files and the immediate unit are now made. That should help the compiler optimization and minimizes the use of temporal registers.
- The significantly long compilation time of the first pass has been reduced. Compiling huge architecture with thousands of function units took too much time and memory. Instead of adding N-multiple function units for each operation, only a single unit is added. Then after the operation pruning pass, when used operations are known the needed N-1 function units are added.

A publication was made of the AEx2 to the FitOptiVis special issue of JSPS. In this publication, we analyzed the current performance and the remaining bottlenecks that are feasible to tackle with well-identified future work. See Figure 48 for conclusive numbers for a set of single thread benchmarks with the various different means to execute on the FPGA fabric, including a commercial Vitis HLS flow.

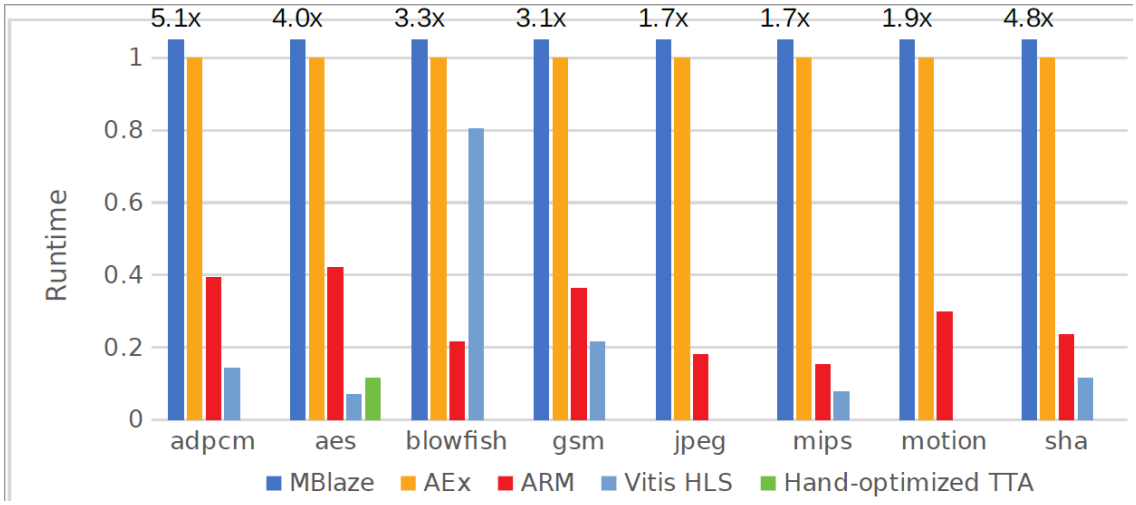


Figure 48: Overall runtime comparison.

Figure 48 presents overall runtime comparison between MicroBlaze, AEx (initial version), Arm and Vitis HLS tool. MicroBlaze runtime values presented in Figure 48 are truncated.

In order to find the bottlenecks that limit reaching the Vitis HLS output (a non-programmable fixed function design per application) with a programmable TTA soft core, there was also a hand-optimized TTA for one of the most “FPGA-potential” of the applications (aes). In the publication we thoroughly identified analyzed these bottlenecks and found out that they are within reach, thus, it should be possible to reach close to fixed function commercial HLS results after those issues have been tackled. The publication is pending review, and since we believe the results have commercialization potential details of those findings cannot be reproduced here due to this report being a public one.

6.6. Object detection on FPGA using Waldboost algorithm

We developed an IP Core for object detection in video based on Aggregated Channel Features (the particular variant of the algorithm is WaldBoost with decision trees over aggregated channel features). The models for the IP core can be trained with the WaldBoost package described above.

In the last period of the project, the detector IP was slightly modified for its easy integration into applications. The input interface was extended to include an image header that stores the image ID and its dimensions, the time of capture and other capture information. The image ID is also stored in the detection results, simplifying the association of the corresponding detection and image in the processing stream. By knowing the size of the image from the header, the detector can process images of different sizes without the need for settings in the registers. If the width of the image is larger than the memory size the image is cropped and an error is signaled in the detection results.

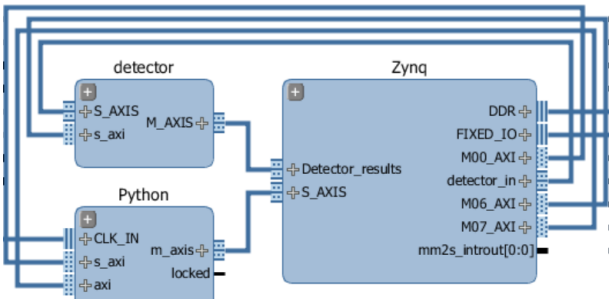


Figure 49 Accelerator in an example project in Vivado.

The IP core has been extended to support simultaneous detection of objects of multiple types. This is advantageous, for example, in traffic applications for detecting one and two row license plates which require separate models. Also, the IP configuration library has been extended to support multiple models.

The detector allows you to configure up to 30 different parameters that affect speed, maximum image size and object size range, accuracy, and resources (logic and memory) for a specific application. For this reason, a python script was created to estimate the required configuration and resources based on the application parameters and the results from training. In the script, it is possible to set the number of detectors, their properties, the maximum image resolution, the object size range, and the required maximum processing time for a given resolution. The script generates a configuration file in VHDL language.

Finally, the detection IP kernel was tested in a traffic application, where it detected single-line and double-line license plates with high accuracy. The test was performed in difficult lighting conditions under bright sunlight in a scene with sharp shadows. Out of 150 passing cars, the detector correctly detected 148 cars – detection rate > 98 % in this particular traffic application.

The detection IP core can process video up to 4K resolution, classify up to 8 different object classes, and provides sufficient performance to process FullHD video at 60 frames per second. It has low resource consumption, achieves high detection accuracy and can be synthesized on even the cheapest FPGA, enabling fast and robust object detection even on low power platforms.

6.7. HDR image acquisition, merging and tone-mapping

The HDR image acquisition is composed from three main blocks: image capturing, HDR merging and tone-mapping (See Figure 50). The image capturing part is driving the exposure time and is grabbing the images from the sensor. HDR merging processes multiple images (in our architecture three) into the HDR frame. The tone-mapping block is compressing the high dynamic range into the standard, 8-bit image while preserving the details from HDR. We provide two of the main blocks in the form of IP core, the Merging (with de-ghosting) and Tone-mapping.

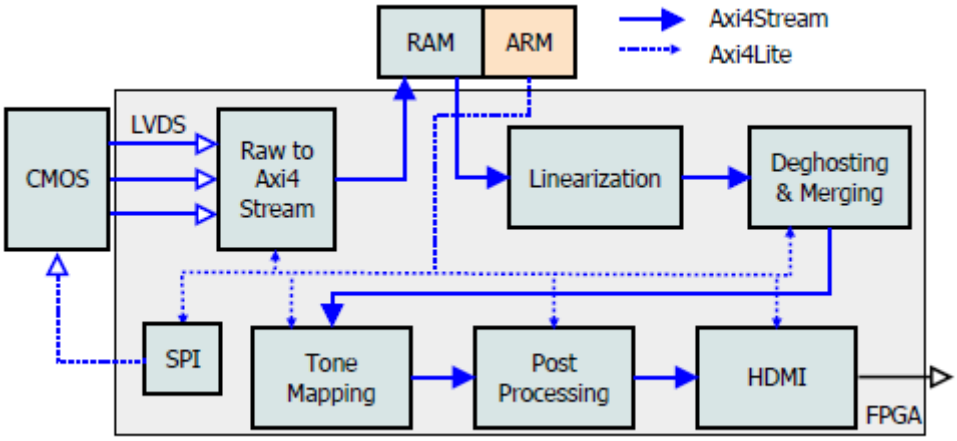


Figure 50: Overall schematics of HDR acquisition pipeline.

The individual blocks create a dataflow pipeline, which is configured through AXI Lite interface from ARM CPU.

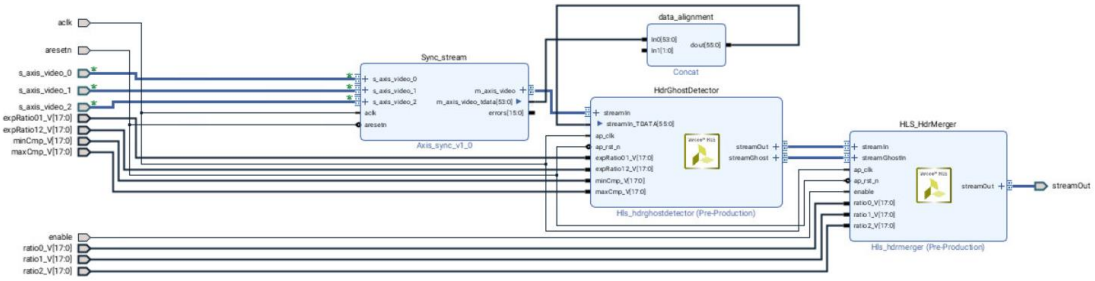


Figure 51: Vivado schematics of ghost-free HDR merging block.

"Ghost detection" and HDR merging blocks are divided into separate IP cores in Figure 51.

The input of the Ghost detection block are three separate image streams (in the form of AXI Video stream), which have to be synchronized (as can be seen in Figure *51*) for pixel-by-pixel processing. The input is in the fixed-point arithmetic, by default in the representation of 12.8 bits - 12 fixed point bits and 8 fractional. It provides sufficient accuracy with minimal resource consumption - compared to floating point arithmetic. Both blocks are written in HLS and, thus, provide an easy change of configuration, input image number and format and also data representation (which depends on previous parameters and desired HDR bit depth). The output stream (also in the form of Axi Video stream) consists of HDR pixels; therefore, we provide it in 16.12 bits format.

Figure 52: (left) standard merging algorithm, (right) output of our algorithm.

Implementation of the IP made in Y3allows the full parametrization of IP cores through the AXI4-Lite interface. It is possible to change the parameters regarding the exposure time of individual images and, therefore, to adapt it for desired input sequence / dynamic range in the input.

Table 3 shows the resources consumed by our Ghost free merging IP cores. The overall utilization is related to the FPGA XC7Z020 on the ZC702 development kit, for which we compiled the demo design. The table contains post-implementation resource consumption.

	Merging	Tonemapping (Durand)
Flip-flops (FF)	2025	9018
Look-up tables (LUT)	1259	15269
Look-up tables as RAM (LUTRAM)	222	97
BlockRAM (BRAM)	1	24
DSP multipliers	16	76

Table 3: Resources consumed by our Ghost free merging IP cores.

6.8. Convolutional HW accelerator

This accelerator core performs convolutional filtering on image data. The user selectable design time option uses limited precision number space with high dynamic range (compared to the number of bits used). The details were derived in the

modelling task 3.1, after a suitable parameter space had been found, based on simulations. Other parameters were left to the runtime domain, because many CNN applications scale the image or data array during iterations. Therefore, resolution, or array dimensions in case of non-image data, is best left as a runtime configurable option.

The core is compatible with CNN based image / video content analysis, and in the future, it is planned to be integrated with YOLO open source content analysis software. Description of the YOLO can be found in [6.11].

Especially CNN like algorithms will benefit from the reduced precision approach, but the system can be used for other convolutional operations also. Main concern is the precision required. For some applications this approach will be sufficient, while others will suffer from the quality degradation. This means that any user adopting the methods developed here must be aware of this trade-off.

Using a hardware approach also allows performing several operations in parallel. This is especially valuable in case of neural network running several convolutional kernels over the same input image, most likely in iterative manner. In hardware, especially with limited prevision, several of these convolution kernels can be run in parallel. This will reduce the number of memory accesses to retrieve the image / video data for the kernel, thus, improving energy efficiency.

The accelerator core has been implemented in FPGA for prototyping and testing and also netlisted for silicon implementation. The core has the same external interfaces and programming model, regardless of which arithmetic option is selected. This allows the designers to freely choose the implementation, and even reconsider the selection after preliminary tests. This flexibility is most useful in FPGA designs, but can be used during ASIC simulation stages also. As an added bonus, the identical interfaces and programming model allow the core to be used in dynamic reconfiguration applications, where the arithmetic type can be swapped on the fly.

Besides the IP core in VHDL and a QRML model, the work here has produced two M.Sc. theses directly discussing this topic and one paper is currently in review for publication. As future extensions we are considering nesting of convolutions and also other arithmetic units using limited precision number space. The nesting would pass results of one unit directly to the next, without memory accesses. This would be very useful in deeper neural networks, in layers that do have only single layer after them. The limited precision approach could be applied to almost any data, even if images are the main target at this time. However, if the numbers are in the limited precision scale for this core, then all other arithmetic operations should support it to avoid unwanted conversions to and from simple binary. Therefore designing more units with support for the approach is vital for wider success.

6.9. Video-based Point Cloud Compression

During the last two years Nokia has worked on the development of Video-based Point Cloud Compression (V-PCC) as main part of Virtual Reality use case. This use case has been utilised and tested in MPEG standardisation forum. The upcoming MPEG standard for video-based point cloud compression is built around 2D video encoding technology. The standard video coding technology can be utilised with existing

hardware mobile phone solutions and distribution infrastructure, i.e. existing hardware video encoders and decoders, available on any modern mobile handset, can carry the bulk of the processing operations.

The Test Model video-based point cloud compression (V-PCC) is project that was started after the Call for Proposals (CfP) for Point Cloud Coding in MPEG [6.6], [6.7]. The core encoding and decoding process for V-PCC were inherited from the solution that demonstrated the highest compression efficiency among all proponents as was agreed during the MPEG 119 meeting in Macau.

We will describe shortly the main architecture structures and essential technical blocks used in V-PCC model. The description of the encoding strategies is also provided. The block structure shown in Figure *53* is used for encoding while for decoding the block structure in Figure *54* is used instead.

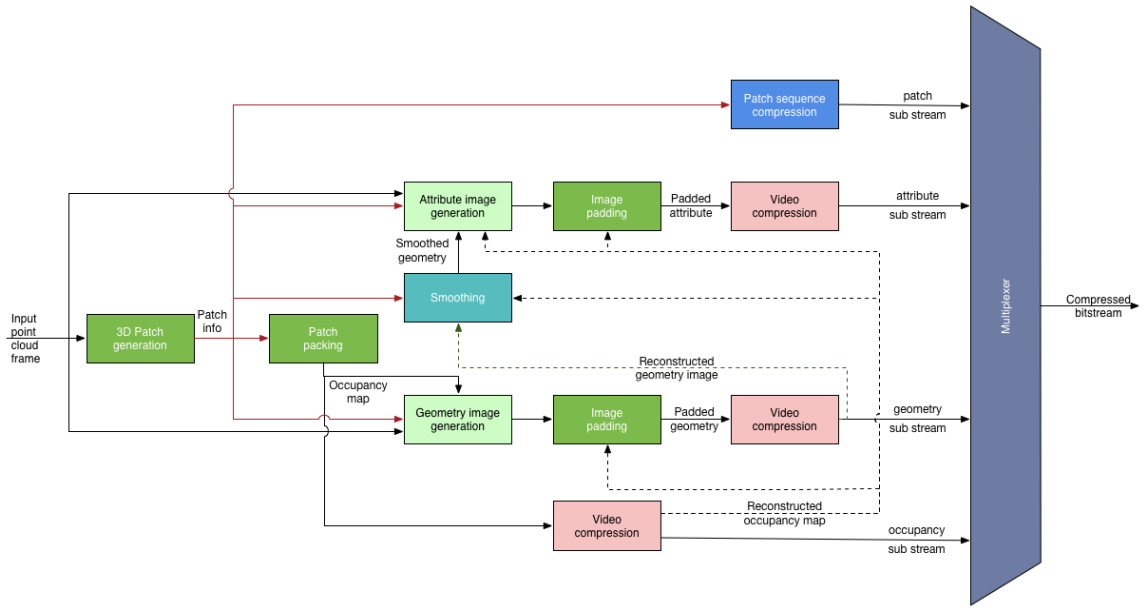


Figure 53: V-PCC encoding structure.

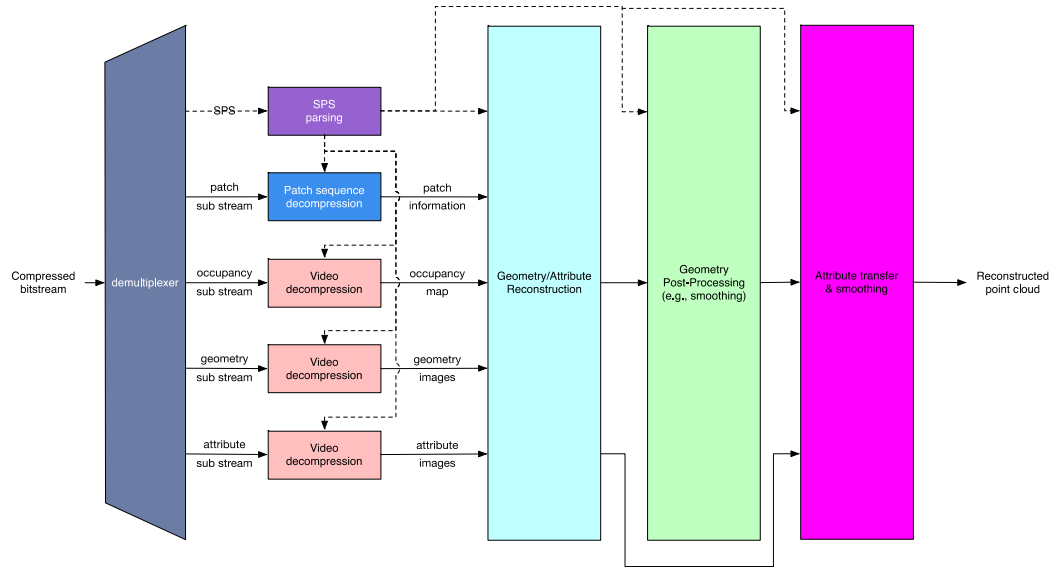


Figure 54: V-PCC decoding structure.

At the encoding stage input point, cloud frame is processed in the following manner.

First, the volumetric 3d data must be represented as a set of 3d projections in different components. At the separation, stage image is decomposed into far and near components for geometry and corresponding attributes components, in addition, an occupancy map 2d image is created to indicate parts of an image that shall be used. The 2d projection is composed of independent patches based on geometry characteristics of the input point cloud frame.

Patch generation method, patch packing strategies and padding methods are out of the scope of the standard. Nokia has been studied their implementations in the best practices. After the patches have been generated and 2d frames for video encoding were created the occupancy map, geometry information, attribute information and the auxiliary information may be compressed.

The reconstructed geometry information may be smoothed outside the encoding loop as a post processing step. Additional smoothing parameters that were used for the smoothing process may be transferred as supplemental information for the decoding process. At the end of the process, the separate bit streams are multiplexed into the output compressed binary file.

Decoding process starts form demultiplexing of the input compressed the binary file into geometry, attribute, occupancy map and auxiliary information streams. The auxiliary information stream is entropy coded and the detailed description of coding methods for auxiliary information compression is provided in WP6.

Occupancy map is compressed using video compression and must be upscaled to the nominal resolution. The nearest neighbour method is applied for upscaling. Geometry stream is decoded and in combination with occupancy map and auxiliary information, smoothing may apply to reconstruct point cloud geometry information.

Based on the decoded attribute video stream and reconstructed information for smoothed geometry if present, occupancy map and auxiliary information the attributes of the point cloud can be reconstructed. After attribute reconstruction stage additional attribute smoothing method is used for point cloud refinement.

In WP3 tasks 3.1-3.3 Nokia has been profiling of algorithms in design time from system performance point of view. The main goal is to optimize execution of algorithms in the point cloud system environment. This means the identification of sequential and parallel execution of tasks in different design phases. In this way, the main benefit is to understand the main computational challenges when implementing V-PCC system and standard.

Nokia has been focused on acceleration of individual algorithms in combination of GPU and CPU. This has been done in image processing in the latest GPU generations from ARM (Mali) and Qualcomm (Adreno). The algorithms have been covered rendering, decoding reconstruction and filtering operations as discussed before in V-PCC architecture description.

Also, some special challenges as bottlenecks exist in the synchronisation and buffering of the parallel video streams have been studied in very detail HW and SW levels. A particular problem is the handling of decoded video outputs on Android devices. Here, FitOptiVis will improve over the existing standard with an efficient and effective synchronisation solution, enabling V-PCC real-time decoding and playback on modern Android handsets.

Based on the results of WP3 tasks Nokia can provide the analysis for profiling and optimization, implementation recommendations, and performance understanding in the V-PCC system and algorithm levels. As the results of these research studies Nokia’s V-PCC demo source code is available for other partners [6.12]

Our experiments have shown that most modern mobile handsets are capable of achieving real-time decoding of at least 25 frames per second as well as real-time AR rendering, thus, proving the general claim of real-time capability of V-PCC system.

6.10. Acceleration of face detector on GPU and DSP

The implementation of the face detector is based on RetinaFace project [6.13] which has achieved state-of-the art performance in benchmarks [6.14]. RetinaFace is a single stage detector. This architecture means that object localization and classifying are both conducted on each inference cycle. This branch of CNN object detectors has been growing rapidly over the last couple years. The system relies on usage of predefined anchor boxes that are used for bounding box placement in the decoding phase where detections are mapped into an image plane. The minimum dimension requirements for detection are approximately 30 x 40 and the maximum is around 1000 x 1000 in pixel width and height.

CNN algorithms are often computationally expensive and the most power consuming parts in applications. To meet this challenge, it is nowadays common to integrate several different computing devices in a single chip each accelerating specific algorithms. Designing, implementing, and tuning new algorithms for ISPs has remained a high-cost exercise. Therefore, a more general purpose solution such as

mobile graphics processing units (GPUs) and digital signal processors (DSP) that handle image and video processing tasks is desired.

In this work, scalability of the solution using mobile GPUs and DSPs helps to achieve improved energy efficiency and low power dissipation which is needed when dealing with battery powered devices. In our experiments, we have evaluated the detector implementations on a mobile development board. The optimization act here is a balancing one between model complexity and inference optimization. Inference can be optimized using quantization method, which means dropping the accuracy and specificity of used variables by using smaller variable sizes.

The results were measured on the Qualcomm's Snapdragon 855 mobile platform. The CPU in this platform is Kryo 485 CPU, Octa-core CPU with clock speed up to 2.84 GHz and the GPU is an Adreno 640. The DSP is Qualcomm Hexagon™ 690 Processor with Hexagon Vector eXtensions (HVX) and Hexagon Tensor Accelerator.

The measured computation time for the CPU implementation was 1020 ms with the image size 4096x2156. Using the GPU implementation, the processing time was 293ms. Power consumption was measured as the total system power on the platform. We used the National Instruments NI 4065 measurement device for measuring the electric current. First, the baseline system current without the algorithm running was measured in order to determine the actual power consumption of the algorithm. The baseline current was 207mA. Next, we measured electric current of the CPU and GPU versions of the algorithm using image resolution 4096x2156. Figure *55* and Figure *56*. The measured average electric current for CPU implementation was 294mA and for GPU implementation it was 241mA. Thus, energy efficiency is much better with the GPU implementation.

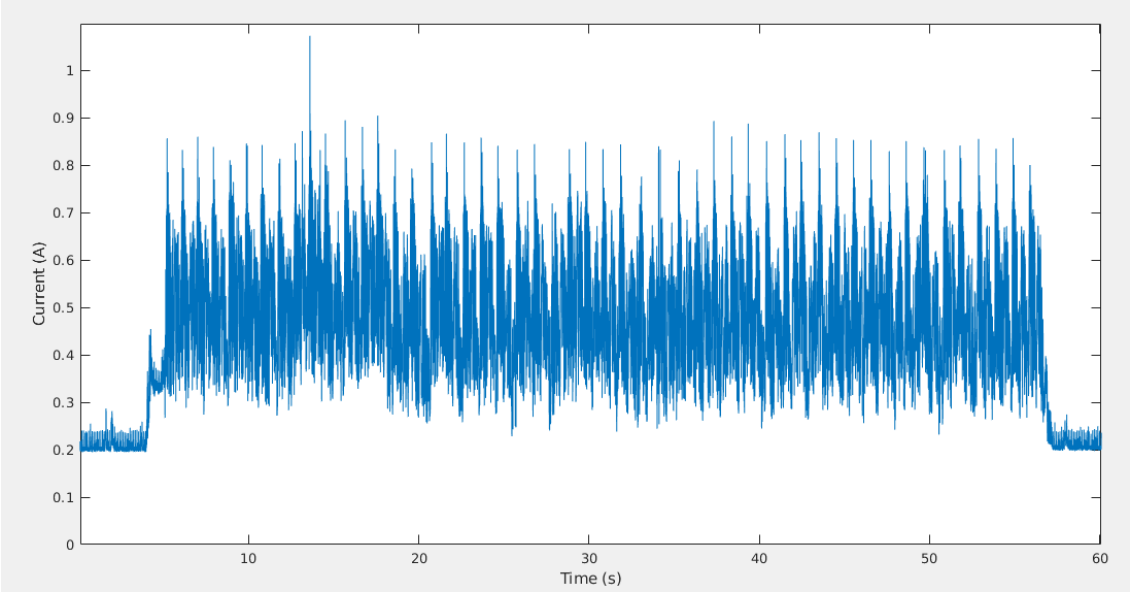


Figure 55: Power consumption in case of CPU implementation.

Figure 55 presents measurement when running face detector for (4096 × 2156) size frames with CPU implementation. X-axis shows time and Y-axis shows current in the range between 0 and 1100 mA.

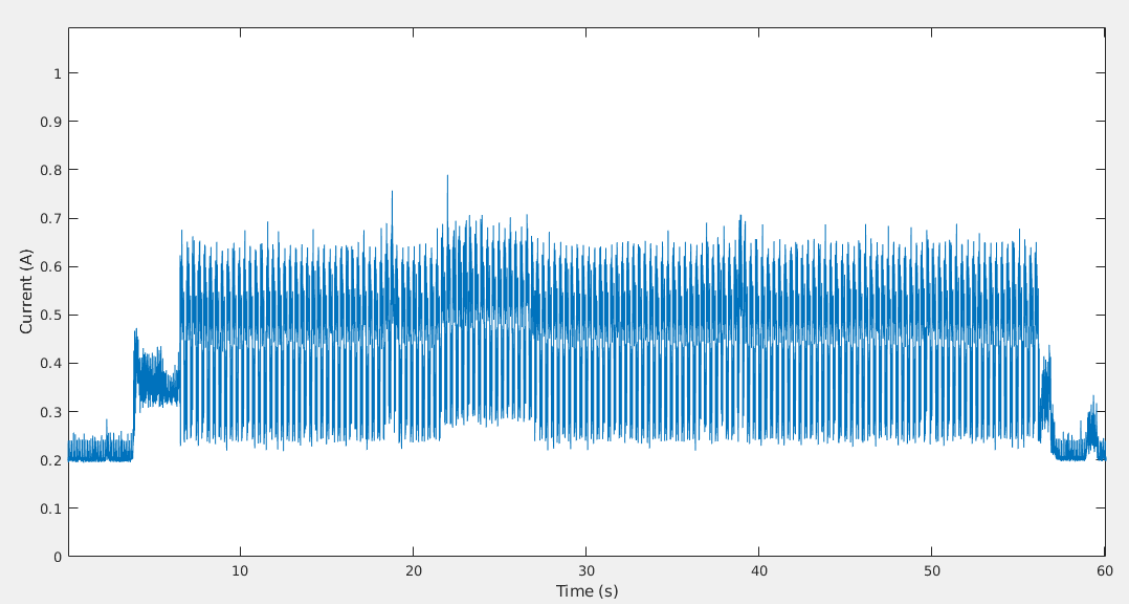


Figure 56: Power consumption in case of GPU implementation

Figure 56 presents power consumption measurement when running face detector for (4096 × 2156) size frames with GPU implementation. X-axis shows time and y-axis shows current in the range between 0 and 1100 mA.

We will continue GPU/DSP/CPU inference optimization of native implementation, embedded and real-time (video) detection implementation. Also re-training the detector with GPU/DSP supported operations and training heavier models is considered.

6.11. Automated Toolchain for Adaptive Neural Network Accelerator

During Y3, with the aim of delivering support and design automation for the development of adaptive Neural Network (NN) accelerators, a toolchain has been assembled by interfacing different tools and frameworks available in literature and in the market. As depicted in Figure 57, the toolchain adopts MDC (see Section 6.3), NEURAghe (see Section 6.4) and Vivado HLS [6.15]. The entry point of the toolchain is a NN model in ONNX format (widely adopted NN exchange format), easily generated from the main NN development frameworks (Pytorch, Keras, etc.) commonly used by NN application developers. A dataflow based flow is adopted, for which application are modelled as graphs, whose nodes are computational units according to operations involved in the application (actors), which connections are point to point buffered links between computational units which are described through an XML file (network) compliant with MDC input format. NN described in ONNX can be directly mapped into such dataflow formalism: actors reflect NN layers and dataflow connections reflect NN tensors between layers. Adaptivity is implemented through the definition of different set points for the application, which means different NN or dataflow models, each presenting a different behavior under different metrics (e.g. latency, consumption, quality, etc.). Two parallel flows are envisioned starting from input ONNX model(s) of application or application set points:

- Actor (blue flow in Figure 57):

- A1: ONNX model(s) are automatically converted into the corresponding C implementation through the ONNXparser, which is part of the NEURAghe software stack, to provide C description of actors (NN layers);
- A2: refactoring and pragmas are applied by the user on top of the generated C actors in order to i) optimize them to obtain the best execution efficiency; ii) provide different set points to shape adaptivity (e.g. actors with different consumption versus latency trade off). Figure 57 gives overview of the assembled toolchain for adaptive NN hardware accelerators.
- A3: Vivado HLS is then exploited to automatically generate the hardware description (in HDL) corresponding to C actors (and in turn NN layers), to be used in the final NN adaptive hardware accelerator.

- Network (red flow in Figure 57):
  - N1: ONNX model(s) is(are) converted (1 to 1 matching) into dataflow network(s), and, if necessary, new dataflows are modelled according to the way the user is shaping adaptivity (each dataflow model corresponds to one application set point);
  - N2: dataflow networks are automatically combined together by MDC front-end sharing common resources through multiplexer logic, and generating a reconfigurable dataflow network capable of executing all the different input dataflow networks;
  - N3: MDC back-end automatically generates the reconfigurable hardware accelerator for Xilinx environment whose computing core is corresponding to the reconfigurable dataflow network using the hardware description of the actors generated within step A3.

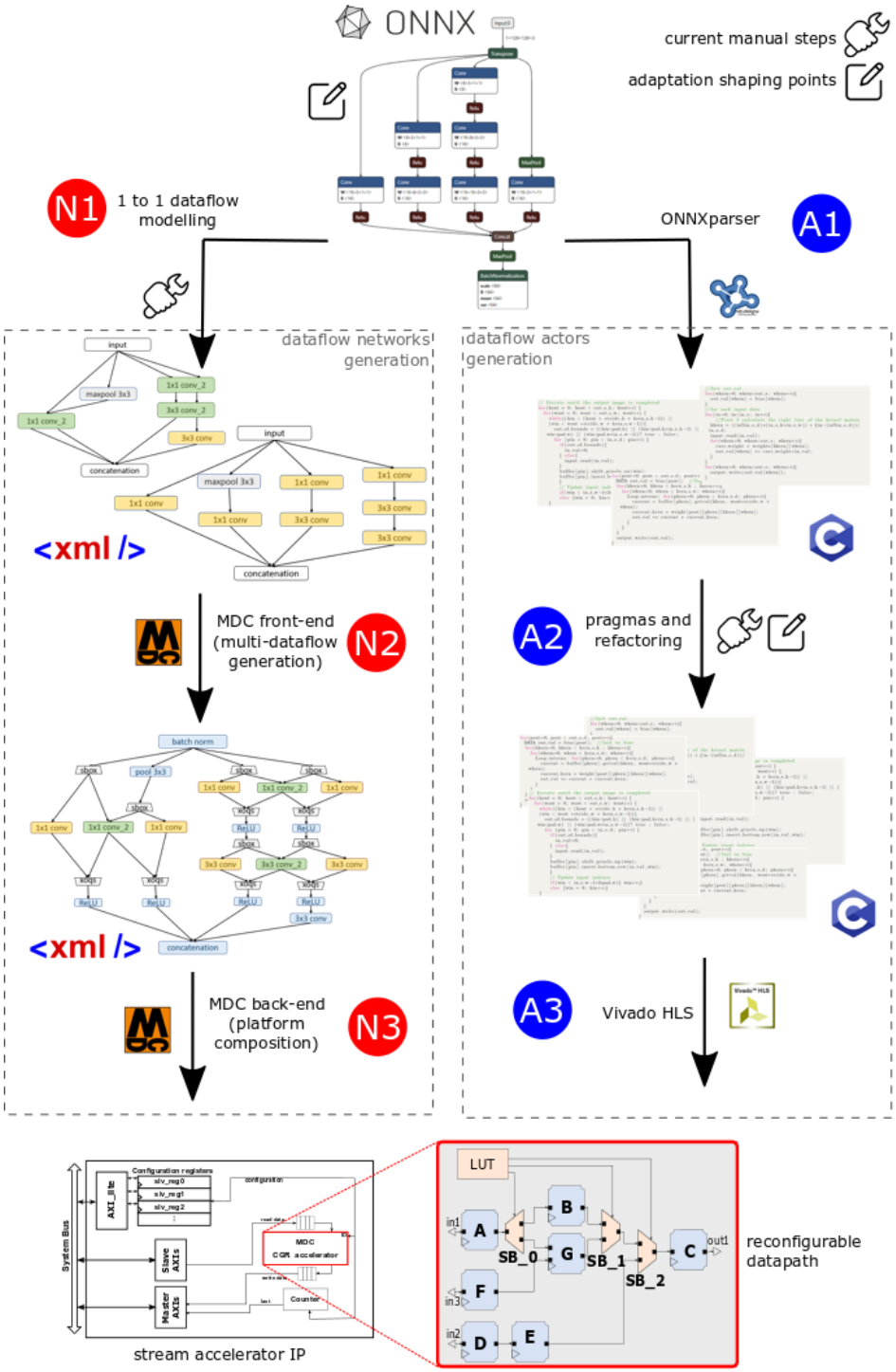


Figure 57: Adaptive NN hardware accelerators

The toolchain is almost fully automated, only the steps where users can shape adaptivity (A2 and N1) are left manual, since they depend on their specific needs. All the other steps are performed automatically, and the final result is a ready-to-use accelerator which can be implemented and easily adopted in the practice on Xilinx FPGA devices thanks to the provided scripts and drivers coming from the MDC performed N3 step.

The assembled toolchain is being used by UNICA, UNISS and AITEK within the scope of UC1 – Water Supply to provide an adaptive CNN accelerator capable of classifying humans and animals to monitor accesses on the targeted critical infrastructure.

7. Design Time Support for Methodologies and Tools

This chapter describes design time support for methodologies and tools released in Y3 by FitOptiVis WP3 partners and general public in Y3 of the project in [7.15]-[7.20]. These released design time resources are also summarised in section 10.8 as "Design Time Resource Integrator of Model Composer IPs (DTRiMC) Technology" These activities have been developed in tasks T3.1, T3.2 and T3.3.

Development evolution

In Y1, project released support for Xilinx Zynq systems with these specific features:

- ZynqBerry system presents small, low cost system with design time support being developed in FitOptiVis [7.1]. It has the RaspBerry form factor and works with the (28nm) Xilinx 32bit Zynq device with small programmable logic area.
- Medium size 16nm 64bit Zynq UltraScale system with design time support being developed in the FitOptiVis [7.2]. It is re-using the carrier board and the Full HD video I/O FMC card used in the Almarvi project.
- Large 16nm 64bit Zynq UltraScale system with design time support being developed in the FitOptiVis [7.3]. It is re-using the video Full HD video I/O FMC card used in the Almarvi project. The carrier has the Mini ITX form factor.

Table *4* summarizes the progress made by the FitOptiVis partners in the WP3 from M1 to M12.

➢ **ALMARVI - end of project:** ▪ Zynq 7000 family (28nm) ▪ Stand-alone only	➢ **FitOptiVis – Y1:** ▪ + Zynq Ultrasscale+ 16nm + Small: ZynqBerry 28nm ▪ + PetaLinux OS + Debian FS support
➢ **ALMARVI limitations:** ▪ Limiting PL size of Zynq ▪ no VCU, no GPU ▪ no USB support ▪ no Ethernet board 2 board communication framework	➢ **FitOptiVis progress:** ▪ + Large PL of Ultrascale+ ▪ + VCU, + GPU, ▪ + USB ▪ + Ethernet board 2 board communication based on Arrowhead Framework

Table 4: Progress made in FitOptiVis in WP3 in Y1. Comparison to ALMARVI.

The FitOptiVis Y1 design time resources have been described in D3.1.

The technology developed in Y1 is summarized in section 10.7 of this D3.2 deliverable as the “Design Time Resource Configurator (DTRC) technology” [7.1], [7.2], [7.3].

In Y2, WP3 partners developed, documented and released for public use the second release of the design time resource support for a family of Xilinx Zynq and Zynq Ultrascale+ systems. See [7.12] and [7.13]. This is summary of the new features developed in Y2:

- Support for designs with Xilinx SG for DSP data streaming IPs for Zynq
- Support for designs with Xilinx SG for DSP data streaming IPs for Zynq Ultrascale+
- Geneation of data movers for external IP blocks based on SDSoC 2018.2
- Export of generated Vivado/SDSoC HW sub-systems as shared C++ SW library API
- SW developer can program „main“ applications without SDSoC 2018.2 compiler license with the standard g++ compiler and „make“.
- Swap of complete programmable logic during run-time, while Debian OS based application continues to run.

The FitOptiVis Y2 design time resources have been described in D3.2. All these Y2 design time resources worked with fixed, precompiled HW. There was no possibilty for the end-user to extend the precompiled HW platform with own custom HW.

In Y3, the design time resources released by UTIA removed this restriction. The (DTRiMC) Technology released in Y3 for public access in [7.15]-[7.20] work with open Vivado 2018.2 and 2017.4 HW projects. The end-user can extend the initial HW platform with own custom HW IPs, configure and compile complete system with DTRiMC tool support. See Figure 58.

This compilation of HW projects requires the commercial Xilinx SDSoC 2018.2 compiler license. It also requires UTIA license for the 8xSIMD HW accelerator IP.

To ease these license-related requirements, the Y3 released design time resources [7.15]-[7.20] also include precompiled fixed HW designs which are ready for use by the user SW code without license.

UTIA provides the evaluation versions of the 8xSIMD HW accelerator IPs in the pre-compiled HW designs [7.15]-[7.20]. These evaluation versions of HW accelerators have built-in HW limitation of the number of operations which can be executed. If this HW limit expires, power-down circle is needed to re-activate evaluation IPs.

Application notes and evaluation packages released in Y3 [7.15]-[7.20] also demonstrate the run-time re-configurability of systems with 8xSIMD HW accelerators operating in parallel to the HW accelerated video processing. Different firmware programs program are compiled, downloaded and executed in each video processing frame (in 16 ms = 60x per second). These programs test vector operations supported by the 8xSIMD HW accelerators.

Parallel processing is possible due to the asynchronous nature of the implemented SW API. Video processing HW is processing each video frame autonomously, without blocking of ARM A9 or A53 processor cores. These processing cores are used for re-

compilation of firmware, program transfer, data transfer and control of execution of the 8xSIMD HW accelerators.

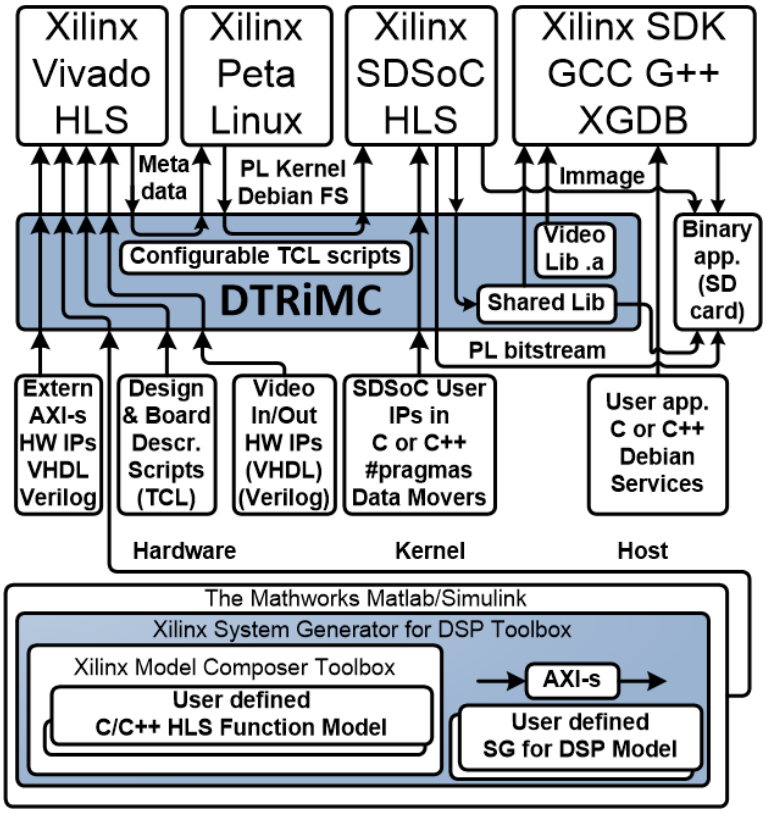


Figure 58: Block diagram of DTRiMC supported design flow.

Both these parallel computing processes/activities (HW accelerated processing of the video frame and computation in 8xSIMD HW accelerators) are joined and ended before the end of each incoming video frame and started again with the next in-coming video frame. See Chapters 7.1 – 7.4 and the application notes and evaluation packages [7.15]-[7.20] for the details.

7.1. DTRiMC for TE0820-3CG and TE0820-4EV modules

This section presents released application notes and evaluation packages [7.15] and [7.16]. It is design time resource integration of model composer DTRiMC tool for TE0820-3CG and for TE0820-4EV device. See Figure 58. It serves for integration of two 8xSIMD, FP03x8, floating-point, run-time-reconfigurable HW accelerators for the Zynq Ultrascale+ TE0820-02-3CG-1E module [7.16] and for TE0820-03-4EV-1E module [7.15] on TE0701 carrier board. The TE0820 modules and the TE0701 carrier board are designed and manufactured by the company Trenz Electronic.

The supported initial HW platform for Zynq Ultrascale+ ZU3CG and ZU4EV devices is described in Figure 59. It contains FULL HD HDMI video input and video output HW IPs, Sobel video processing filter, SDSoC 2018.2 matrix multiplication HW accelerator and two 8xSIMD FP32 run-time reprogrammable accelerators. ILA IP serves as in-circuit logic analyser. The DTRiMC tool scripts create this platform project. User can extend it with its own IPs. Next steps involve platform creation, PetaLinux kernel

configuration, export of SDSoC 2018.2 platform, data mover creation in the SDSoC 2018.2 and finally export of HW kernel se shared object for the standard C or C++ Debian user space SW application.

The two devices (ZU3CG and ZU4EV) have been selected for detailed documentation in [7.15] and [7.16] to compare the impact of dual core and quad core A53 processing system.

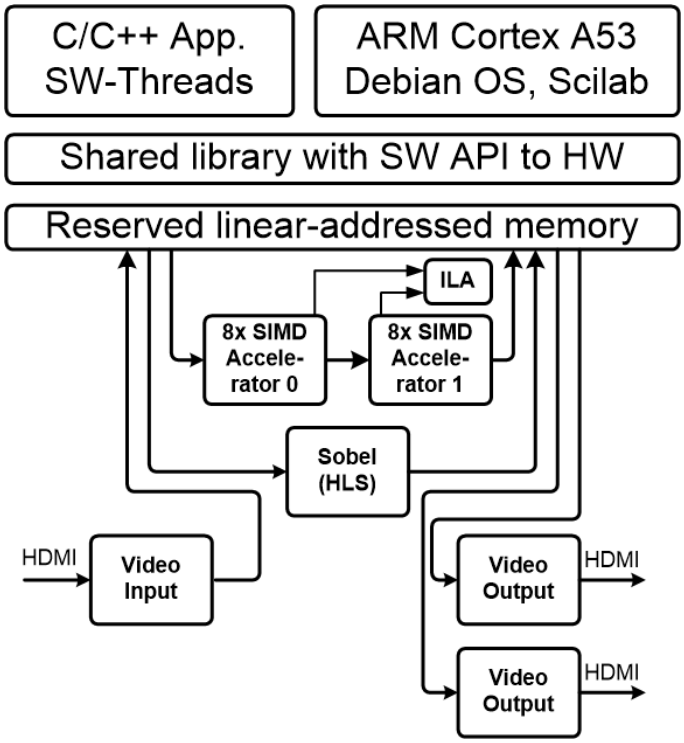


Figure 59: HW platform for Zynq Ultrascale+ ZU3CG and ZU4EV devices.

Figure 60 presents running systems with HW accelerated video processing and parallel FP32 matrix multiplication accelerated by the two 8xSIMD run-time reprogrammable HW accelerators.

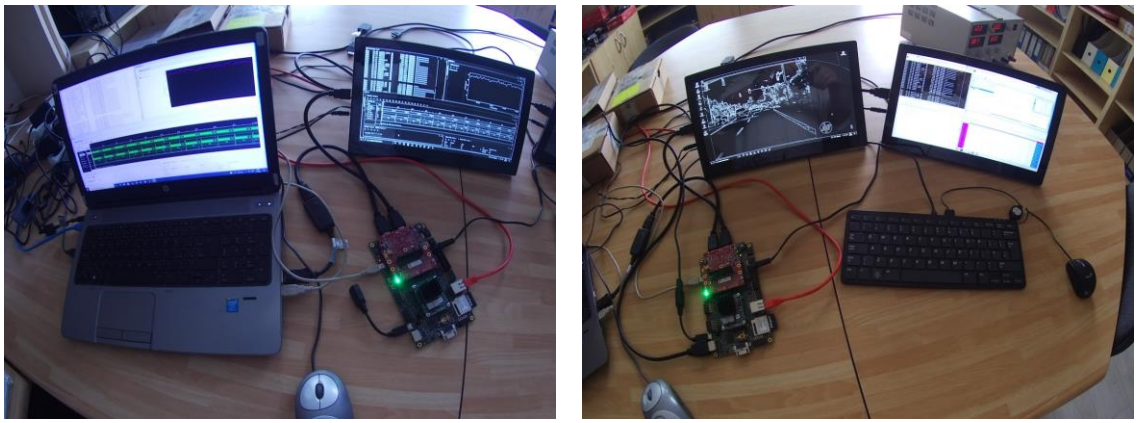


Figure 60: SW application running on Zynq Ultrascale+ ZU3CG and ZU4EV devices.

Comparison of matrix multiplication performance on Zynq Ultrascale+ ZU3CG and ZU4EV devices:

System	Function	MFLOPs
ZU03-CG-1E	8x mmult use B1..B4, include parallel copy of B1..B4	5039
	8x mmult use B1..B8, include parallel copy of B1..B8	4862
	SW mmult Scilab MEX style, 4 threads (on 2 cores)	807
ZU04-EV-1E	8x mmult use B1..B4, include parallel copy of B1..B4	5047
	8x mmult use B1..B8, include parallel copy of B1..B8	4883
	SW mmult Scilab MEX style, 4 threads	1521
I7 PC 3.0 GHz SW Ubuntu, SciLab C MEX style, 1 thread:		1933

The video input FPS is 60.0 FPS (see Figure 60). This is defined by the information received from the Full HD HDMI video input sensor. The video output is also fixed, 60 FPS and this is defined in the video output HW IPs (See Figure 59).

Power	ZU04EV Power [W]	ZU03CG Power [W]
Linux system is running with all HW (See Figure 59). present in the device. No user app.	7,32	7,08
As above, with user interface in Full HD HDMI desktop.	8.88	8.16
Linux system is running with all HW (See Figure 59) present in the device. SW app. sobel_all.elf is running. It performs HW accelerated edge detection and scrolls through tests of two 8xSIMD HW accelerators.	9,84	9,24
As above (sobel_all.elf is running.), with user interface in Full HD HDMI desktop.	11.4	10.44

Table 5: Power consumption of ZU04EV and ZU03CG systems

Power consumption is measured on input power line 12V. Power difference presented in Table 5 is related to the dual core (ZU03CG) and quad core (ZU04EV) A53 processing systems. See details in [7.15] and [7.16].

7.2. DTRiMC for TE0808-15EG and TE0808-09EG-ES1

This section describes results of application notes and evaluation packages [7.17] and [7.19] for the Design Time Resource integration of Model Composer DTRiMC tool. See Figure 1. It serves for integration of eight 8xSIMD, FP03x8, floating-point, run-time-reconfigurable accelerators for Zynq Ultrascale+ TE0808-15EG-1EE module and TE0808-09EG-ES1 module on TEBF0808 carrier board.

We have selected these two devices to demonstrate support for different Xilinx tool chain versions (Vivado HLS and SDSoC 2017.4 and 2018.2). Xilinx device ZU09-EG-ES1 device requires in the design phase the Xilinx Vivado tool version 2017.4. This tool must have enabled support for the Xilinx ZU09-EG-ES1 device. The Xilinx Vivado 2017.4 is the last Xilinx toolchain still supporting the ZU09-EG-ES1 device. We have selected this device, because UTIA owns three TE0808-09EG-ES1 modules systems with this device.

Created DTRiMC support enabled use of these modules by UTIA researchers and PHDs for research of mapping of DSP algorithms to the array of eight 8xSIMD HW accelerators.

The initial HW platform (See Figure 61) created by DTRiMC tool scripts serves for integration of eight 8xSIMD, FP03x8, floating-point, run-time-reconfigurable HW accelerators for the Zynq Ultrascale+ TE0808-15EG-1EE module on TEBF0808 carrier board. The TE0808 module and the TEBF0808 carrier board are designed and manufactured by the company Trenz Electronic.

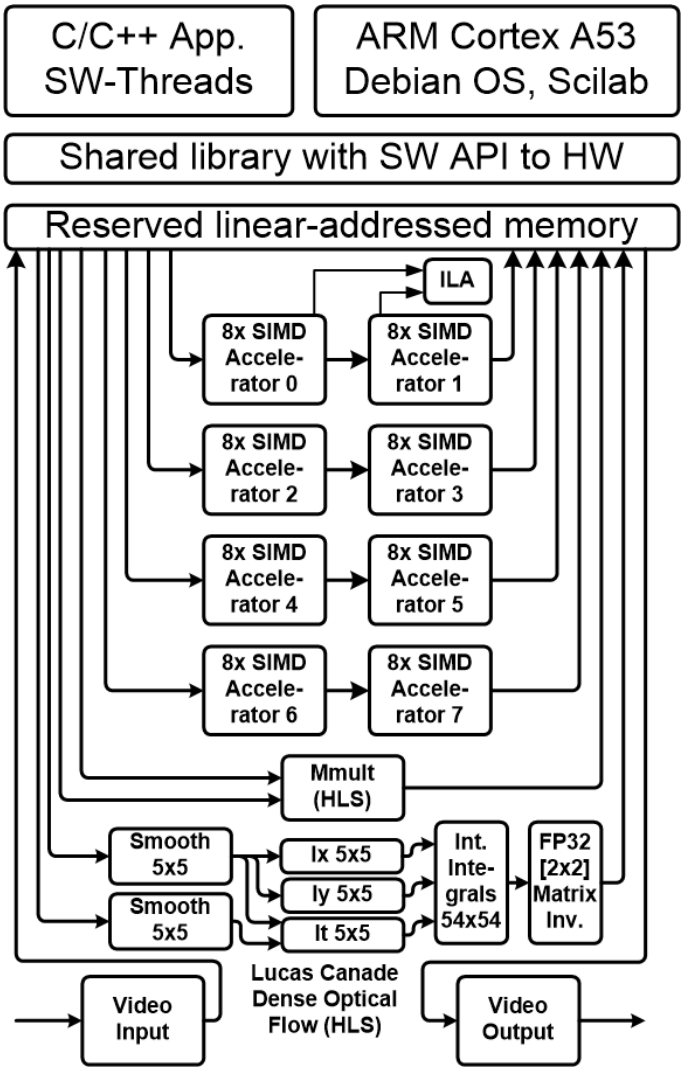


Figure 61: HW platform for Zynq Ultrascale+ ZU15EG device.

The supported initial HW platform for Zynq Ultrascale+ TE0808-15EG-1EE device is described in Figure 61. It contains FULL HD HDMI video input and video output HW IPs, LK Dense Optical Flow video processing algorithm, SDSoC 2018.2 matrix multiplication HW accelerator and eight 8xSIMD FP32 run-time reprogrammable accelerators. ILA IP serves as in-circuit logic analyser.

The DTRiMC tool scripts create this platform project. User can extend it with its own IPs. Next steps involve platform creation, PetaLinux kernel configuration, export of SDSoC 2018.2 platform, data mover creation in the SDSoC 2018.2 and finally export of HW kernel se shared object for the standard C or C++ Debian user space SW application.

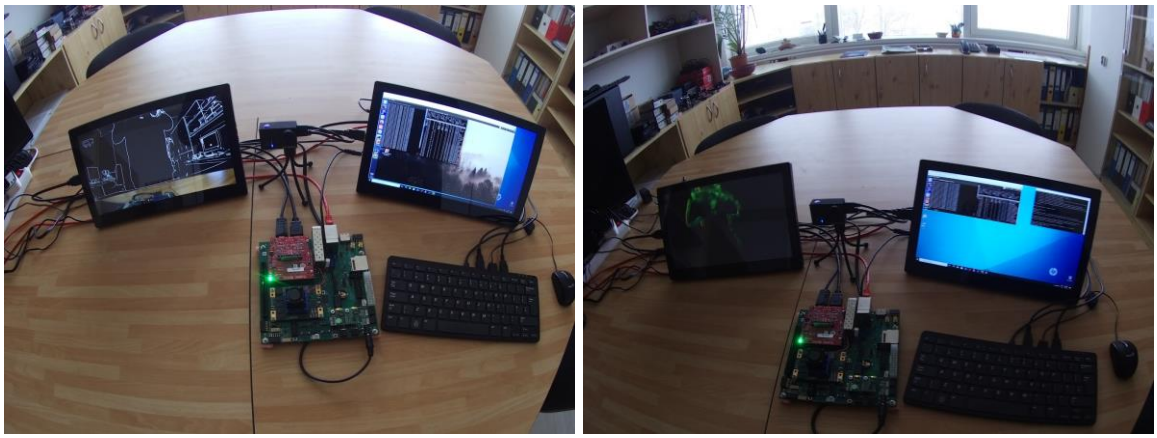


Figure 62: SW (Sobel and LK DOF) on the Zynq Ultrascale+ ZU15EG device.

Comparison of matrix multiplication performance for Zynq Ultrascale+ ZU15EG:

System	Function	MFLOPs
ZU15-EG-1EE	8x mmult use B1..B4, include parallel copy of B1..B4	19111
	8x mmult use B1..B8, include parallel copy of B1..B8	15417
	SDSoC HW mmult use B1..B8, include copy of A, B, Z	6366
	SW mmult Scilab MEX style, 4 threads	1364
	SW mmult C style, 1 thread	895
	SW Scilab C MEX style , 1 thread	166
I7 PC 3.0 GHz	SW Ubuntu, SciLab C mex, 1 thread:	1933

Power consumption is measured on input power line 12V. All power supply is derived from this single power source.

Power consumption	Power [W]
Linux system is running with all HW interfaced by the library libte02_4x2_async_mulf64_sgdma_hw.so is present in the device. No user app is running	14,64
Linux system is running with all HW interfaced by the library libte02_4x2_async_mulf64_sgdma_hw.so is present in the device. SW app. sobel_all.elf is running. It performs HW accelerated edge detection and scrolls through all tests of all 8 8xSIMD HW accelerators. These tests are controlled by 4 SW threads.	17,04
Linux system is running with all HW interfaced by the library libte02_4x2_async_mulf64_sgdma_hw.so is present in the device. SW app. sobel_mmultf1_4xB.elf is running. The application performs It performs HW accelerated edge detection and in parallel it also performs repeated tests of HW accelerated floating point matrix multiplications on eight 8xSIMD HW accelerators. These tests are controlled by 4 SW	18,72

threads.	
Linux system is running with all HW interfaced by the library libte03_4x2_async_mulf64_sgdma_hw.so is present in the device. No user app is running	14,64
Linux system is running with all HW interfaced by the library libte03_4x2_async_mulf64_sgdma_hw.so is present in the device. SW app. dof_all.elf is running. In each frame, the application performs first the HW accelerated dense optical flow controlled from single SW thread and then performs sequence of repeated tests of supported firmware operaions on all eight 8xSIMD HW accelerators.	17,52
Linux system is running with all HW interfaced by the library libte03_4x2_async_mulf64_sgdma_hw.so is present in the device. SW app. dof_mmultf1_3xB.elf is running. This application performs the HW accelerated dense optical flow controlled from single SW thread and in parallel it also performs repeated tests of HW accelerated floating point matrix multiplications on 6 8xSIMD HW accelerators. These tests are controlled by 3 SW threads. Two 8xSIMD HW accelerators are not used.	19,32

Table 6: Power consumption of Zynq Ultrascale+ ZU15EG system.

Table 6 documents the increase of power consumption related to the increased complexity and increased DDR data traffic of the HW accelerated LK DOF algorithm in comparison to the simple HW accelerated Sobel filter video processing. We can also see the increased power consumption related to the HW accelerated FP32 matrix multiplication performed in parallel to the HW accelerated video processing on the eight 8xSIMD run-time reprogrammable HW accelerators.

7.3. DTRiMC for TE0726-03M and TE0726-03-07S board

This chapter describes results of application note and evaluation packages for the Design Time Resource integration of Model Composer DTRiMC tool [7.18] and [7.20]. It serves for integration of 8xSIMD, FP03x8, floating-point, run-time-reconfigurable accelerator for Zynq device on TE0726-03M board [7.18] and support for HW data-movers on the TE0726-03-07S board [7.20].

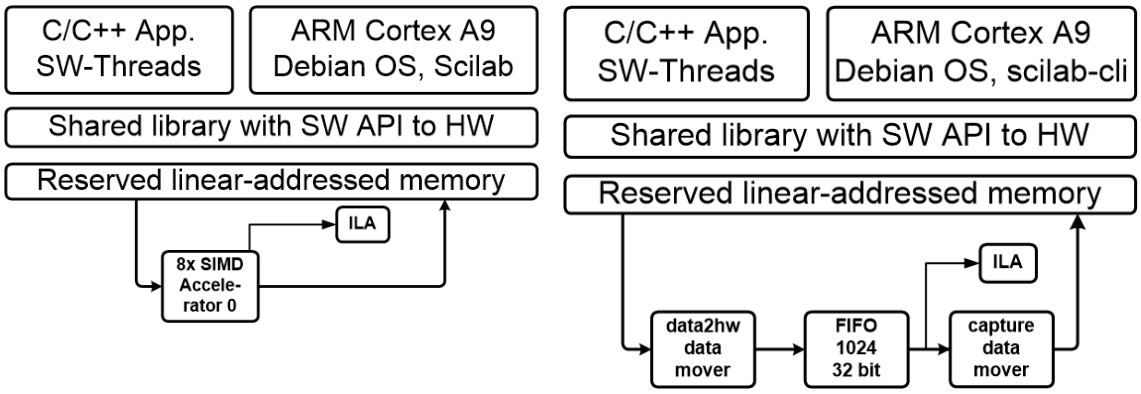


Figure 63: Two HW platforms for Zynq 7000 device: 8xSIMD accelerator and FIFO.

We have selected these tow platforms to demonstrate portability of the DTRiMC tool to the Zynq 7000 family of devices.

The platform for the TE0726-03-07S Zynq 7000 device was selected to demonstrate and document complete the DTRiMC tool supported design process without UTIA 8xSIMD HW IPs. User can use the FIFO HW IP or replace it with own HW IP.

Left part of Figure 64 is presenting running TE0726-03M board performing matrix multiplication accelerated by single 8xSIMD HW accelerator together with ILA display. Right part of Figure 69 is presenting running TE0726-03M board performing also the matrix multiplication accelerated by single 8xSIMD HW accelerator with X11 display on a remote desktop and power measurement. The remote desktop is running on an Ubuntu PC connected by 100 Mbit Ethernet to the TE0726-03M board by standard PuTTY client.

Figure 64: SW (floating point benchmark) on Zynq 7000 device.

Comparison of matrix multiplication performance for TE0726M board:

System	Function	MFLOPs
TE0726M	HW accelerated matmul() on 1x 8xSIMD, 1 thread	1269
	SW matmul(); Scilab MEX style, 1 thread.	225
I7 PC 3.0 GHz	SW Ubuntu, SciLab C MEX style, 1 thread.	1933

Power consumption	Power [W]
Debian OS system is running with all HW interfaced by the library libfp01x8_v26x1_hw.so is present in the device. Remote desktop with user terminal and mousepad editor open. No user application is running.	2,8
Linux system is running with all HW interfaced by the library libfp01x8_v26x1_hw.so is present in the device. SW application fp01x8_v26x1_sw.elf is running. It performs HW accelerated matrix multiplication on one 8xSIMD HW accelerator.	3,3

Table 7: Power consumption of Zynq zc7z010 with 8xSIMD accelerator.

Table 8 presents measured HW data mover performance on the TE0726-03-07S Zynq 7000 board. It is compared with the optimized (-O3) ARM host SW implementation of SW data copy of a single precision floating point matrix [64x64] from user space memory to linear addressable non-cacheable memory area and back to user space memory. See details in [7.18].

TE0726M-07S	***MByte/s***
ZC HW data movers	170.4
ZC SW copy	19.9
DMA HW data movers	151.1
DMA SW copy	19.9
SG HW data movers	79.1
SG SW copy	19.9
SG-malloc HW data movers	9.8
SG-malloc SW copy	229.6

Table 8: Measured performance of HW data movers for Zynq xc7z07s device.

We can see HW acceleration over the SW implementation of data copy in case of ZC, DMA and SG HW data movers performing data copy from/to the reserved un-cacheable linear addressable memory area.

SW implementation significantly outperforms the SG-malloc HW implementation of data movers in case of data copy from/to the cacheable standard Debian OS user-space memory area. See details in [7.20].

7.4. Tool development directions after the end of project

In Y3, the development of design time resources has been affected by wider context of the development directions taken by Xilinx. In 2019, Xilinx SDSoC compiler development stopped with the last ver. 2019.1. In 2020, Xilinx moved to a unified Vitis 2019.2 Acceleration flow with OpenCL. See the roadmap of Xilinx SW/HW design tools in Figure 65.

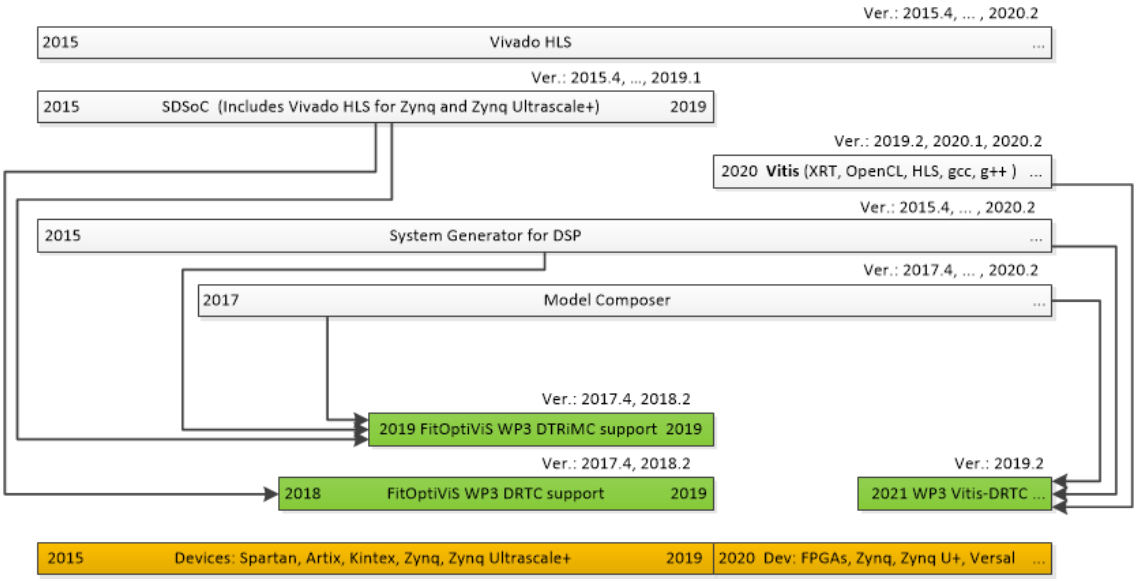


Figure 65: Time frame and roadmap of Xilinx SW/HW design tools.

In Y3, based on this context, UTIA with WP3 partners decided to continue support for:

- Xilinx Vivado HLS ver. 2017.4 and SDSoC ver. 2017.4 by DTRC and DTRiMC tools (Zynq, ZU+, ZU+ ES1 devices). See [7.14], [7.19].
- Xilinx Vivado HLS ver 2018.2 and SDSoC ver. 2018.2 by DTRC and DTRiMC tools (Zynq, ZU+ devices. See [7.12], [7.13], [7.15]-[7.18] and [7.20].

In 2021, UTIA with WP3 partners also started an initial, exploratory work on support for the Xilinx Vitis 2019.2 acceleration flow (named as Vitis-2019.2 DTRC) to support at least some custom Zynq Ultrascale+ modules.

On April 20-th 2021, UTIA presented preliminary results to the FitOptiVis partners in form of an internal tutorial demonstrating an initial support for Zynq Ultrascale+ device ZCU04-EV on industrial grade modules TE0820 and TE0803 manufactured by company Trenz Electronics.

Figure 66 and Figure 67 present measured memory bandwidth in developed te0820-04ev and te0823-04ev Vitis 2019.2 platform. In the figures, the horizontal axis is size of transferred data in Mbytes. The vertical axis is the achieved bandwidth in MBytes/s, (logarithmic scale).

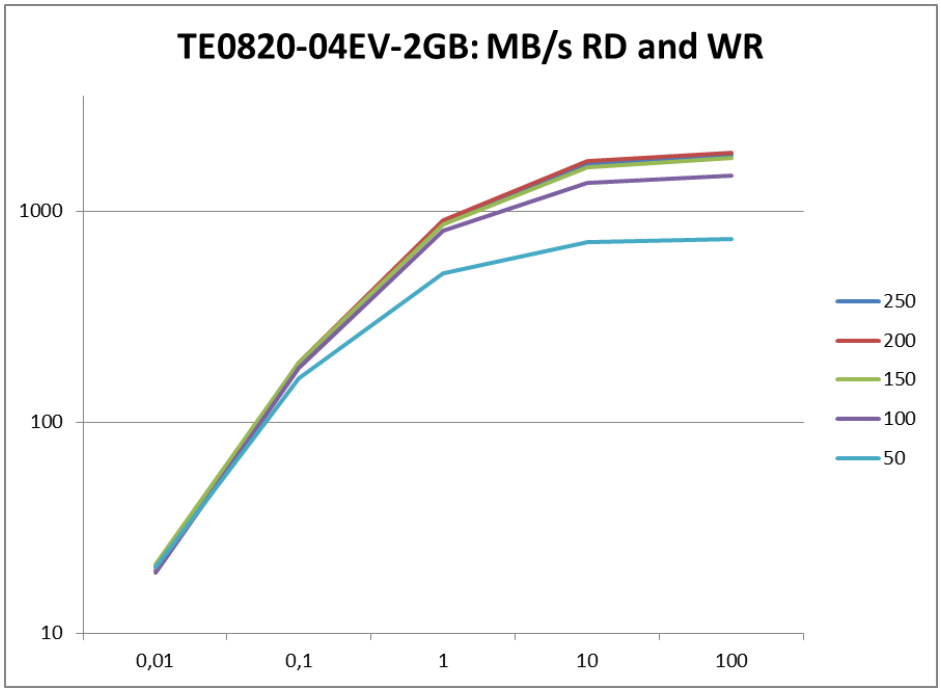


Figure 66: Measured memory bandwidth in te0820-04ev Vitis platform.

The measured bandwidth is presented in Figure 66 and Figure 67, by different colors, parametrized by the device programmable logic clock in ranges 50, 100, 150, 200 and 250MHz. The bandwidth includes the ARM SW overhead needed to start and stop each data transfer transaction.

The quad core ARM A53 processing system is running with 1.2 GHz clock. The data width is set to the maximal possible value supported by Vitis 2019.2 compiler: 512 bits.

Figure 67: Measured memory bandwidth in te0803-04ev Vitis platform.

The te0803-04ev module has better bandwidth due to the 2x wider DDR4 data interface implemented on the larger te0803-04ev module. See Figure 66 and Figure 67.

At the end of the project, both developed Vitis 2019.2 Zynq Ultrascale+ HW platforms support only the HW acceleration Vitis 2019.2 compiler flow. The effective interface to custom input and output video frame buffers with video content defined by the HW VDMA data movers remains unsolved for both developed custom Vitis 2019.2 platforms.

The effective interface to custom input and output video frame buffers with video content defined by the HW VDMA data movers is solved and documented for the SDSoC 2018.2 platforms in released application notes and evaluation packages [7.15]-[7.17].

UTIA plans to continue this development, documentation and creation of evaluation packages for the Vitis 2019.2 Zynq Ultrascale+ custom, industrial grade HW modules also after the end of the FitOptiVis project.

8. Conclusions

This deliverable describes the status of design time methodologies, frameworks and strategies developed by FitOptiVis project partners up to the end of the project. It collects the results achieved in tasks T3.1 “Model-driven engineering techniques for energy performance and other qualities” in Chapter 4, T3.2 “Programming and parallelization support” in Chapter 5, and T3.3 “Accelerator support” in Chapter 6.

8.1. Main achievements in WP3 of the FitOptiVis project

8.1.1. UTIA

For UTIA, the main achievement in FitOptiVis is the development, documentation and release of the board support packages [7.15]-[7.20] for a wide range of custom Zynq (28 nm) and Zynq Ultrascale+ (16 nm) modules together with the DTRiMC tool developed in the project. The DTRiMC tool serves for support of integration of custom, run-time reprogrammable, SIMD, floating point HW accelerators into Debian systems with Full HD HDMI HW-accelerated video processing capability together with support for compilation of HW accelerators by the Xilinx SDSoC system level compiler.

8.1.2. BUT

The main achievement for BUT is the successful development of real-time object detection technology - ACF Core. The tools around ACF Core allow to easily create custom detectors and to configure them for a wide range of target platforms. In certain configurations (e.g. known object size or low image resolution) we support even low-end FPGAs with limited resources. The technologies developed in WP3 were successfully integrated in WP5 in license plate detection component, and demonstrated in Traffic surveillance use case in WP6.

8.1.3. TUT

The main quantitative achievements related to TCE soft core work were received from the optimization of SIMD support and multicore support: Since an FPGA is a highly parallel customizable structure, in addition to the automated customization work done with AEx, it is essential to utilize the data and task level parallelism to maximum with minimal overheads; this is in addition to the scalable instruction-level parallelism of the TTA approach. The biggest numerical difference was found to be thanks to the interconnection network optimizations: the network itself now takes up to 54 % less logic to implement and the entire core up to 30 % less, when reflecting to the start of the project.

8.1.4. UCAN

Important design time resource development made in Y3 by UCAN is the RIE methodology and C++ library for component-based implementation of embedded systems. RIE supports runtime reconfiguration of the software components described in the QRML modelling language developed in WP2. It is possible to generate RIE code from the WP2 QRML language and UML/MARTE models from the S3D – Single Source Design Framework. The library classes are used to implement components and monitors. Additionally, the library also simplifies component deployment in the

cloud and edge. See Sections 4.1, 5.8 and Sections 10.5, 10.6 in the appendix.

8.1.5. UNIVAQ

In the context of the ECSEL FitOptiVis project (WP3), UNIVAQ has extended the HEPSYCODE methodology to consider energy-aware requirements, metric, and cost function in the design space exploration step. User energy requirements can be related to the possibility to find system implementations based on a *dedicated heterogeneous/homogeneous multi-processing system* (D-HMPS) that consumes as little energy as possible, or to find D-HMPSs that consumes less energy than a given threshold, while considering also other non-functional requirements (e.g., timing, cost, etc.). The considered energy metric is the J4CS, while the design space exploration analyzes alternative solutions by means of an evolutionary algorithm that considers, at the same time, with a weighted sum method, several objectives. So, taking into account different processor technologies (GPP, ASP, SPP), HEPSYCODE is able to find a HW/SW partitioning, to define a HW architecture and to suggest a mapping potentially able to satisfy all the requirements. Finally, HEPSYCODE is able to estimate timing performances and energy consumption by means of a SystemC simulator that considers the results found by the evolutionary algorithm. In this way, HEPSYCODE is also able to identify, at design-time, suitable “configurations” for different trade-offs (e.g., timing vs energy/power) by considering a heterogeneous set of processors.

8.1.6. TUE

The key achievements in the course of the FitOptiVis project are (i) integrally considering parallelism and pipelining in the analytical SPADe framework in optimizing the quality of control of image-based control loops, (ii) extending the SPADe framework to take into account several practical aspects such as work-load variation, inter-frame dependencies, and resource limitations making the flow usable in a wider set of use cases (iii) making necessary adaptation for implementation/integration in industrial platforms (iv) integration of the design flow in the existing toolchain IMACS. These steps brought the developed method/technique one step forward for wider applicability, usability and higher maturity in terms of TRL.

8.1.7. UNISS

Regarding UNISS contribution to WP3, the main achievements can be summarized as follow: 1) definition of a runtime monitoring extension of MDC mature enough for publication on an ACM transaction, 2) maintenance of the MDC repository and publication of a journal paper describing all the MDC features with step-by-step examples, 3) the design and the implementation of an SMT-based approach for automated consistency checking and inconsistency finding of configuration specifications, and 4) the design and the implementation of a tool for automated test suite generation (ReqT).

8.1.8. HURJA

The main achievement for HURJA is the successful development of design-time optimization and programming strategies of Hurja's Salmi Care Platform for better utilization of computing resources (CPU/GPU) of HoloLens 2 AR-glasses as well as UC3 integration by utilizing FIVIS tool to centralize the gathering of rehabilitation data in UC3.

8.1.9. UTU

The accelerators designed in WP3 were proven correct in both simulations and in FPGA implementation. The performance of both arithmetic models was measured and evaluated against software implementations running on the RISC-V core. The SW reference designs included versions running plain RISC-V instruction set and also improved version using the vector extensions, to ensure proper comparison. In all cases the HW acceleration was faster and especially the reduced precision version provided significant savings in terms of power usage.

8.1.10. NOKIA

Nokia demonstrated compression and real-time visualisation of dynamic point cloud data at the International Broadcasting Convention 2019 and gained lots of interest from the industry. International Broadcasting Convention, more commonly known by its initials IBC, is an annual trade show and "the World's Most Influential Media, Entertainment & Technology Show", aimed at broadcasters, content creators/providers, equipment manufacturers, professional and technical associations, and other participants in the general broadcasting, entertainment and technology industry. Nokia's paper on "real-time decoding and AR playback of the emerging MPEG video-based point cloud compression standard" won the highly prestigious best technical paper award for IBC 2019. In this paper the essential achievements of WP3 design and implementation research work has been presented in detail in the word wide.

Also based on WP3 trial learnings the demonstration of the Augmented Reality extension of the evaluation platform has been used in MPEG (ISO/IEC JTC1 SC29/WG11) standardization process, highlighted the importance of extended reality in the future media codec standardization. These design and tool aspects have become one of the core evaluation criteria for the upcoming standard for immersive multimedia.

Figure 68: Nokia Technology demonstrating point cloud compression technology at IBC 2019.

Figure 69: Sebastian Schwarz (middle) and Mika Pesonen (right) from Nokia Technology received the IBC 2019 Best Technical Paper Award for their work on point cloud compression and visualisation.

8.1.11. CUNI

The key achievement for CUNI in the FitOptiVis project related to WP3 has been the FIVIS data monitoring platform, analysis and visualization platform. We developed support for the QRML models and enabled engineers to visualize the online and offline data measured on FitOptiVis components. The integration with QRML allows visually relating the monitoring data with with particular components or their ports (as defined in QRML). In the scope of WP3, this enables engineers to understand how their component behaves at runtime and provides insights for design time optimization.

8.1.12. TASE

TASE's main achievement in the course of WP3 in the whole project have been the integration of RIE components into the UC10. It allowed an easy integration of components developed by UCAN into the use case. Another key achievement has been the use for the very first time of VITIS in the Space Industry for demonstration purposes. This tool has allowed the adoption of easy design-time support techniques for the development of video components.

9. References

[4.1] F Herrera, J Medina, E Villar, Modeling Hardware/Software Embedded Systems with UML/MARTE: A Single-Source Design Approach. Handbook of Hardware/Software Codesign, 141-185. 2017.

[4.2] Wasif Afzal et al, The MegaM@Rt2 ECSEL Project: MegaModelling at Runtime – Scalable Model-Based Framework for Continuous Development and Runtime Validation of Complex Systems, DSD 2017.

[4.3] V. Muttillo, G. Valente, L. Pomante, V. Stoico, F. D'Antonio, and F. Salice, "CC4CS: an Off-the-Shelf Unifying Statement-Level Performance Metric for HW/SW Technologies", In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18), ACM, New York, NY, USA, 2018, pp. 119-122.

[4.4] L. Pomante. "HW/SW Co-Design of Dedicated Heterogeneous Parallel Systems: an Extended Design Space Exploration Approach". IET Computers & Digital Techniques, Institution of Engineering and Technology, 2013, Vol. 7, Iss. 6, pp. 246–254.

[4.5] Sphinx Needs Requirements, Bug, Test Case suite. https://sphinxcontrib-needs.readthedocs.io/en/latest/

[4.6] IBM Rational DOORS tool for requirements management. https://www.ibm.com/us-en/marketplace/rational-doors

[4.7] Using JIRA for requirements management. https://confluence.atlassian.com/jirakb/using-jira-for-requirements-management-193300521.html

[4.8] Haugen, Ø., Wąsowski, A. and Czarnecki, K., 2013, August. CVL: common variability language. In Proceedings of the 17th International Software Product Line Conference (pp. 277-277). ACM.

[4.9] Haugen, Ø. and Øgård, O., 2014, September. BVR–better variability results. In International Conference on System Analysis and Modeling (pp. 1-15). Springer, Cham.

[4.10] https://www.tensorflow.org/ - Google TensorFlow Deep Learning framework.

[4.11] http://torch.ch/ - Torch Deep Learning framework.

[4.12] https://github.com/jcjohnson/densecap - DenseCap image recognition description Deep Learning network.

[4.13] https://github.com/CMU-Perceptual-Computing-Lab/openpose CMU OpenPose network for recognition of human pose and gestures.

[4.14] Sander Stuijk, Marc Geilen, Bart D. Theelen, Twan Basten: Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications. ICSAMOS 2011: 404-411

[4.15] IMACS is an open-source framework for performance evaluation of IMAge in the Closed-loop System: www.es.ele.tue.nl/ecs/imacs

[4.16] Róbinson Medina Sánchez, Juan Valencia, Sander Stuijk, Dip Goswami, Twan Basten: Designing a Controller with Image-based Pipelined Sensing and Additive Uncertainties. TCPS 3(3): 33:1-33:26 (2019)

[4.18] C. Brandolese, W. Fornaciari, L. Pomante, F. Salice and D. Sciuto, "Affinity-driven system design exploration for heterogeneous multiprocessor SoC," in IEEE Transactions on Computers, vol. 55, no. 5, pp. 508-519, May 2006.

[4.19] L. Pomante. "System-Level Design Space Exploration for Dedicated Heterogeneous Multi-Processor Systems". IEEE International Conference on Application-specific Systems, Architectures and Processors, Santa Monica, September 2011.

[4.20] L. Pomante, P. Serri. “SystemC-based HW/SW Co-Design of Heterogeneous Multiprocessor Dedicated Systems”. International Journal of Information Systems, Journal ISSN Online: 2356-5896, Vol. 1, July 2014.

[4.21] Luigi Pomante, Vittoriano Muttillo, Marco Santic, Paolo Serri, SystemC-based electronic system-level design space exploration environment for dedicated heterogeneous multi-processor systems, Microprocessors and Microsystems, Volume 72, 2020.

[4.22] Embedded System Design: A Unified Hardward/Software Introduction: A Unified Hardware/Software. Frank Vahid, Tony D. Givargis, John Wiley & Sons Inc, 2001. Chapter 7: Digital Camera.

[4.23] D. Ciambrone, V. Muttillo, L. Pomante, G. Valente. “HEPSIM: an ESL HW/SW Co-Simulator/Analysis Tool for Heterogeneous Parallel Embedded Systems”, In 6th EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems (ECYPS’2018), 2018.

[4.24] V. Muttillo, G. Valente, L. Pomante, V. Stoico, F. D’Antonio, and F. Salice, “CC4CS: an Off-the-Shelf Unifying Statement-Level Performance Metric for HW/SW Technologies”, In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18), ACM, New York, NY, USA, 2018, pp. 119-122.

[4.25] http://www.pomante.net/sito_gg/HEPSYCODE_DC_timing_energy.zip

[4.26] I. Ungurean and N. C. Gaitan, "Performance analysis of tasks synchronization for real time operating systems," 2018 International Conference on Development and Application Systems (DAS), 2018, pp. 63-66, doi: 10.1109/DAAS.2018.8396072.

[4.27] K. Mikhaylov and J. Tervonen, "Evaluation of Power Efficiency for Digital Serial Interfaces of Microcontrollers," 2012 5th International Conference on New Technologies, Mobility and Security (NTMS), 2012, pp. 1-5.

[4.28] Sajid Mohamed, Dip Goswami, Vishak Nathan, Raghu Rajappa, Twan Basten: A scenario- and platform-aware design flow for image-based control systems, Microprocessor and Microsystems, 2020.

[4.29] C. Brandolese, W. Fornaciari, L. Pomante, F. Salice and D. Sciuto, "Affinity-driven system design exploration for heterogeneous multiprocessor SoC," in IEEE Transactions on Computers, vol. 55, no. 5, pp. 508-519, May 2006.

[4.30] L. Pomante. “System-Level Design Space Exploration for Dedicated Heterogeneous Multi-Processor Systems”. IEEE International Conference on Application-specific Systems, Architectures and Processors, Santa Monica, September 2011.

[4.31] L. Pomante, P. Serri. “SystemC-based HW/SW Co-Design of Heterogeneous Multiprocessor Dedicated Systems”. International Journal of Information Systems, Journal ISSN Online: 2356-5896, Vol. 1, July 2014

[4.32] Luigi Pomante, Vittoriano Muttillo, Marco Santic, Paolo Serri, SystemC-based electronic system-level design space exploration environment for dedicated heterogeneous multi-processor systems, Microprocessors and Microsystems, Volume 72, 2020.

[4.33] Embedded System Design: A Unified Hardward/Software Introduction: A Unified Hardware/Software. Frank Vahid, Tony D. Givargis, John Wiley & Sons Inc, 2001. Chapter 7: Digital Camera.

[4.34] D. Ciambrone, V. Muttillo, L. Pomante, G. Valente. “HEPSIM: an ESL HW/SW Co-Simulator/Analysis Tool for Heterogeneous Parallel Embedded

Systems", In 6th EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems (ECYPS'2018), 2018.

[4.35] V. Muttillo, G. Valente, L. Pomante, V. Stoico, F. D'Antonio, and F. Salice, "CC4CS: an Off-the-Shelf Unifying Statement-Level Performance Metric for HW/SW Technologies", In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18), ACM, New York, NY, USA, 2018, pp. 119-122.

[4.36] http://www.pomante.net/sito_gg/HEPSYCODE_DC_timing_energy.zip

[4.37] I. Ungurean and N. C. Gaitan, "Performance analysis of tasks synchronization for real time operating systems," 2018 International Conference on Development and Application Systems (DAS), 2018, pp. 63-66, doi: 10.1109/DAAS.2018.8396072.

[4.38] K. Mikhaylov and J. Tervonen, "Evaluation of Power Efficiency for Digital Serial Interfaces of Microcontrollers," 2012 5th International Conference on New Technologies, Mobility and Security (NTMS), 2012, pp. 1-5.

[4.39] S. Mohamed, D. Goswami, S. De, T. Basten. Optimising Multiprocessor Image-Based Control Through Pipelining and Parallelism. IEEE Access. 9:112332-112358, August 2021.

[5.1] Python package for training object detectors: https://github.com/RomanJuranek/waldboost

[5.2] http://SoChub.fi

[6.1] Dollár, Piotr, et al. "Fast feature pyramids for object detection." IEEE transactions on pattern analysis and machine intelligence 36.8 (2014): 1532-1545.

[6.2] P. Musil, R. Juránek, M. Musil and P. Zemčík, "Cascaded Stripe Memory Engines for Multi-Scale Object Detection in FPGA," in IEEE Transactions on Circuits and Systems for Video Technology. doi: 10.1109/TCSVT.2018.2886476

[6.3] Nosko, S., Musil, M., Zemcik, P. et al.," Color HDR video processing architecture for smart camera", Journal of Real-Time Image Proc (2018). https://doi.org/10.1007/s11554-018-0810-z

[6.4] Durand, Frédo, and Julie Dorsey. "Interactive tone mapping." Rendering Techniques 2000. Springer, Vienna, 2000. 219-230.

[6.5] Ozan Aydin, T.; Stefanoski, N.; Croci, S.; et al.: Temporally Coherent Local Tone Mapping of HDR Video. vol. 33. 11 2014: pp. 1–13.

[6.6] PCC Test Model Category 2v0, ISO/IEC JTC1/SC29/WG11 N17248, Macau, China, October 2017.

[6.7] S. Shwartz, P. Chou, I Shinharoy, D. Flynn Common test conditions for point cloud compression. ISO/IEC JTC1/SC29/WG11 N17766, Ljubljana, SI, July 2018.

[6.8] The Multi-Dataflow Composer (MDC) tool: a dataflow-to-accelerator design suite. Get MDC link https://github.com/mdc-suite/mdc
Documentation link https://github.com/mdc-suite/mdc/wiki
Video Lecture link https://youtu.be/_cyYFJCDR3U
Tutorials link https://github.com/mdc-suite/mdc/wiki/MDC-Tutorial

[6.9] Italian project using MDC http://www.cluster-prossimo.it/progetti-partner/

[6.10] EU project using MDC https://www.cerbero-h2020.eu/

[6.11] Joseph Redmon, Ali Farhadi.: YOLOv3: An Incremental Improvement. Technical report. Cornel university. https://arxiv.org/abs/1804.02767

[6.12] V-PCC demo source code: https://github.com/nokiatech/vpcc

[6.13] RetinaFace project: https://arxiv.org/abs/1905.00641

[6.14] RetinaFace benchmarks: http://shuoyang1213.me/WIDERFACE/WiderFace_Results.html

[6.15] Xilinx Vivado High-Level Synthesis: https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0012-vivado-high-level-synthesis-hub.html

[7.1] Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout: “Design Time and Run Time Resources for the ZynqBerry Board TE0726-03M with SDSoC 2018.2 Support”, Application note and Evaluation package [Online]. http://sp.utia.cz/index.php?ids=results&id=FitOptiVis-te0726-SDSoC-2018_2

[7.2] Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout: “Design Time and Run Time Resources for Zynq Ultrascale+ TE0820-03-4EV-1E with SDSoC 2018.2 Support”, Application note and Evaluation package [Online]. http://sp.utia.cz/index.php?ids=results&id=FitOptiVis-te0820-SDSoC-2018_2

[7.3] Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout: “Design Time and Run Time Resources for Zynq Ultrascale+ TE0808-04-15EG-1EE with SDSoC 2018.2 Support”, Application note and Evaluation package [Online]. http://sp.utia.cz/index.php?ids=results&id=FitOptiVis-te0808-SDSoC-2018_2

[7.4] Lukas Kohout, Jiri Kadlec, Zdenek Pohl: “Video Input/Output IP Cores for TE0820 SoM with TE0701 Carrier and and Avnet HDMI Input/Output FMC Module”, Application note and Evaluation package [Online]. http://sp.utia.cz/index.php?ids=results&id=te0820-hio-ho

[7.5] Trenz Electronic, "TE0726 TRM," [Online]. https://shop.trenz-electronic.de/en/27229-Bundle-ZynqBerry-512-MByte-DDR3L-and-SDSoC-Voucher?c=350

[7.6] Documents for Arrowhead Framework Available: https://forge.soa4d.org/docman/?group_id=58

[7.7] Trenz Electronic, "MPSoC Module with Xilinx Zynq UltraScale+ ZU4EV-1E, 2 GByte DDR4 SDRAM, 4x5cm", [Online]. https://shop.trenz-electronic.de/en/TE0820-03-04EV-1EA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU4EV-1E-2-GByte-DDR4-SDRAM-4-x-5-cm

[7.8] Trenz Electronic, "UltraSOM+ MPSoC Module with Zynq UltraScale+ XCZU15EG-1FFVC900E, 4 GB DDR4", [Online]. https://shop.trenz-electronic.de/en/TE0808-04-15EG-1EE-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-XCZU15EG-1FFVC900E-4-GB-DDR4?c=450

[7.9] Trenz Electronic, “”=UltraITX+ Baseboard for Trenz Electronic TE080X UltraSOM+” [Online]. https://shop.trenz-electronic.de/en/TEBF0808-04-UltraITX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261

[7.10] Trenz Electronic, “Carrier Board for Trenz Electronic 7 Series” [Online]. https://shop.trenz-electronic.de/en/TE0701-06-Carrier-Board-for-Trenz-Electronic-7-Series?c=261

[7.11] Lukas Kohout, Jiri Kadlec, Zdenek Pohl: Video Input/Output IP Cores for Xilinx ZCU102 with Avnet HDMI Input/Output FMC Module , Application note and Evaluation package [Online]. http://sp.utia.cz/index.php?ids=results&id=zcu102-hio

[7.12] Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout: FP01x8 Accelerator on TE0726-03M http://sp.utia.cz/results/te0726_fp01x8/AppNote-FitOptiVis-te0726_fp01x8_short.pdf Evaluation package download page: http://sp.utia.cz/index.php?ids=results&id=te0726_fp01x8

[7.13] Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout: Two serial connected evaluation versions of FP03x8 accelerators for TE0820-03-4EV-1E module on TE0701-06 carrier board
http://sp.utia.cz/results/te0820_fp03x8x2s/AppNote-FitOptiVis-te0820_fp03x8x2s.pdf
Evaluation package download page:
http://sp.utia.cz/index.php?ids=results&id=te0820_fp03x8x2s

[7.14] Jiři Kadlec: Eight FP03x8 accelerators for TE0808-09-EG-ES1 module on TEBF0808 carrier board.
AppNote-2017_4-te0808_fp03x8_4x2.pdf (utia.cz)
Evaluation package download page:
http://sp.utia.cz/index.php?ids=results&id=TS74fp03x8

[7.15] Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout: DTRiMC tool for TE0820-03-4EV-1E module on TE0701-06 carrier board
http://sp.utia.cz/results/2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu4ev/AppNote_2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu4ev.pdf
Evaluation package download page:
http://sp.utia.cz/index.php?ids=results&id=2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu4ev

[7.16] Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout: DTRiMC tool for TE0820-02-3CG-1E module on TE0701-06 carrier board
http://sp.utia.cz/results/2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu3cg/AppNote_2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu3cg.pdf
Evaluation package download page:
http://sp.utia.cz/index.php?ids=results&id=2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu3cg

[7.17] Jiři Kadlec, Zdenek Pohl, Lukas Kohout: DTRiMC tool for TE0808-15-EG-1EE module on TEBF0808 carrier board.
http://sp.utia.cz/results/2018_2_te0808_fp03x8_4x2_ila_mulf64_DTRiMC/AppNote_2018_2_te0808_fp03x8_4x2_ila_mulf64_DTRiMC.pdf
Evaluation package download page:
http://sp.utia.cz/index.php?ids=results&id=2018_2_te0808_fp03x8_4x2_ila_mulf64_DTRiMC

[7.18] Jiři Kadlec, Raissa Likhonina: DTRiMC tool for TE0726-03M board.
http://sp.utia.cz/results/2018_2_te0726_fp01x8_ila_DTRiMC/AppNote_2018_2_te0726_fp01x8_ila_DTRiMC.pdf
Evaluation package download page:
http://sp.utia.cz/index.php?ids=results&id=2018_2_te0726_fp01x8_ila_DTRiMC

[7.19] Jiři Kadlec, Raissa Likhonina: DTRiMC tool for TE0808-09-EG-ES1 module on TEBF0808 carrier board.
http://sp.utia.cz/results/2017_4_te0808_fp03x8_4x2_ila_mulf64_DTRiMC/AppNote_2017_4_te0808_fp03x8_4x2_ila_mulf64_DTRiMC.pdf
Evaluation package download page:
http://sp.utia.cz/index.php?ids=results&id=2017_4_te0808_fp03x8_4x2_ila_mulf64_DTRiMC

[7.20] Jiři Kadlec, Zdeněk Pohl, Lukáš Kohout, Raissa Likhonina: Data Movers in DTRiMC tool for TE0726-03M-07S board
http://sp.utia.cz/results/2018_2_te0726_07s_ila_DTRiMC/AppNote_2018_2_te0726_07s_ila_DTRiMC.pdf
Evaluation package download page:
http://sp.utia.cz/index.php?ids=results&id=2018_2_te0726_07s_ila_DTRiMC

10. Appendix: FitOptiVis Design Time Support Tools

The tools developed in WP3 for design time support for new co-processors, hardware accelerators and SoCs each serve their own specific design spaces and purposes. This appendix summarizes all the tools developed in the project along with their key features, inputs and outputs, as well as intended users.

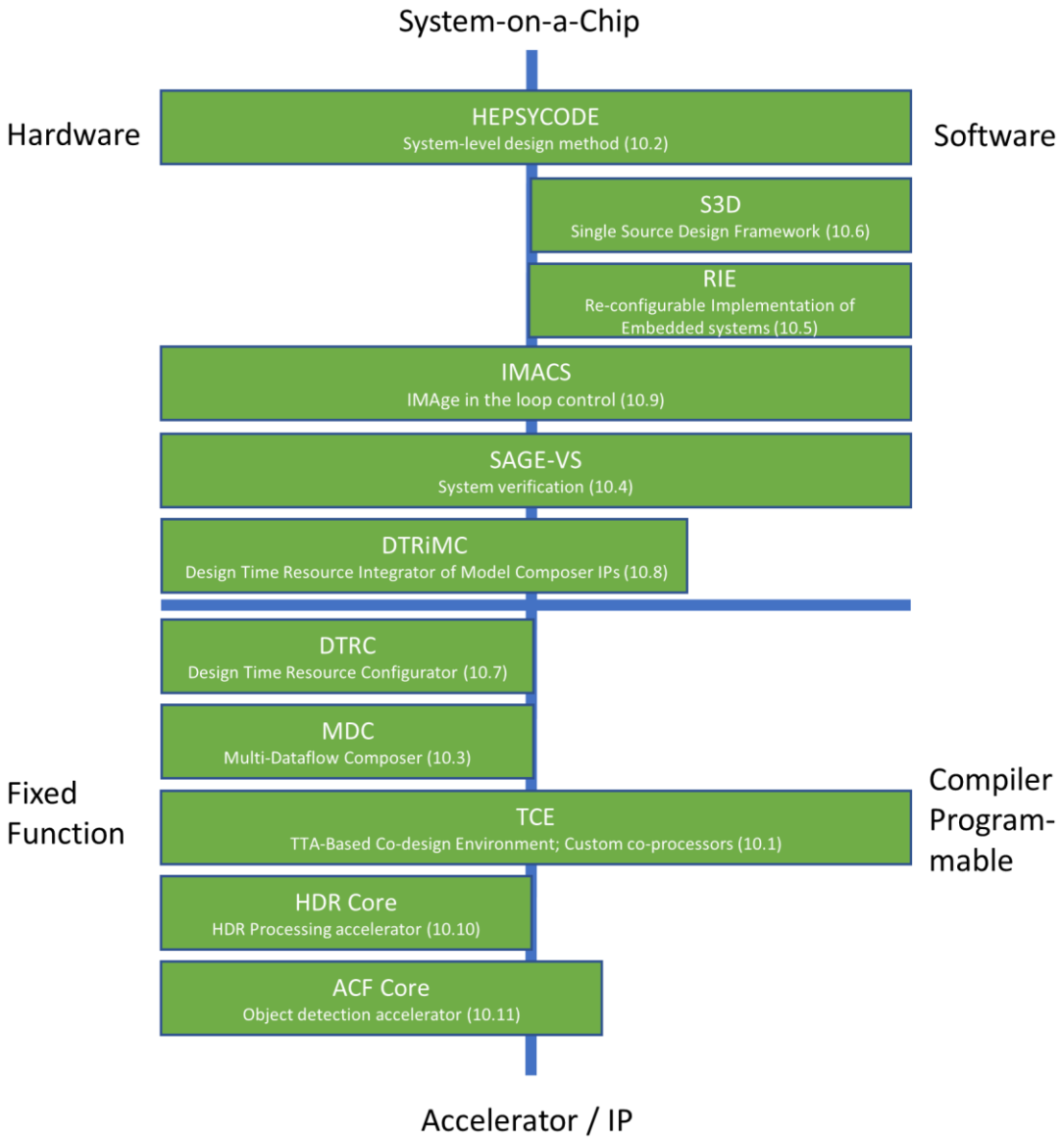


Figure 70: FitOptiVis Design Support Tools.

In order to put the tools to a big picture, an interesting way to visualize them is to map them by their two main characteristics: The granularity and their software/hardware-orientation. When dealing with the accelerator or “SoC component” development tools, the “flexibility” or “the programmability” of the accelerators the tool produces is also an interesting aspect since it affects the reuse of the produced components. The overall

view is shown in Figure 70: Table *9* present status of usage of WP3 tools by project partners in Y3 of the project.

The tools can be categorized according to their granularity: whether they are used assisting the design of the whole system of a chip or a single component (an accelerator or an "IP block") inside the system. Further, some of the system design tools are more software oriented, some focus on hardware, some on both.

For the accelerator design tools, an interesting characteristic is the flexibility of the designed components in the scale of single function hardware accelerators to fully compiler programmable co-processors that can support any C/C++/OpenCL C program from high-level languages. E.g. MDC can generate CGRAs that support multiple functions whereas the programmability of TCE-generated accelerators varies from single function to fully compiler programmable.

Detailed descriptions of the tools listed in Table *9* are shown in the following subsections 10.1 – 10.11.

Partner	Tool:	10_1			10_2			10_3			10_4			10_5			10_6			10_7			10_8			10_9			10_10			10_11		
PHL	NL	n	n	n	n	n	n	n	n	?	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
TUD	NL	n	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
TUE	NL	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	y	y	y	n	n	n	n	n	n
CAMEA	CZ	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	y?	n	n	n	n	n	n	n	n	n	y	y	y	y	y	y
BUT	CZ	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	y	y	y	y	y	y
UTIA	CZ	y	y	y	n	n	n	n	n	y	n	n	n	n	n	y	n	n	y	y	y	y	y	y	y	n	n	y	n	n	n	n	n	n
NOKIA	FI	n	y?	y?	n	n	n	n	?	?	n	n	n	n	n	n	n	?	?	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
TUT	FI	y	y	n	n	n	n	n	n	y	n	n	n	n	n	y	n	y	n	n	n	y	n	n	y	n	n	n	n	n	n	n	n	n
UTU	FI	n	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
VISI	FI	n	n	y	n	n	n	n	n	n	n	n	n	n	n	n	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
HIB	ES	n	n	y	n	n	n	n	n	y	n	n	n	n	n	n	n	y?	y	n	n	n	n	n	n	n	n	y	n	n	n	n	n	n
ITI	ES	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
RGB	ES	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
7SOLS	ES	n	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
TASE	ES	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
UC	ES	n	n	y	n	n	n	n	n	n	n	n	n	y	y	y	y	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
AITEK	IT	n	n	n	n	n	y	y	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
UNICA	IT	n	n	y	n	n	n	y	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
UNISS	IT	n	n	y	n	n	n	y	y	y	y	y	y	n	n	n	n	n	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
UNIVAQ	IT	n	n	n	y	y	y	y	y	y	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n
Partner	Tool:	10_1			10_2			10_3			10_4			10_5			10_6			10_7			10_8			10_9			10_10			10_11		

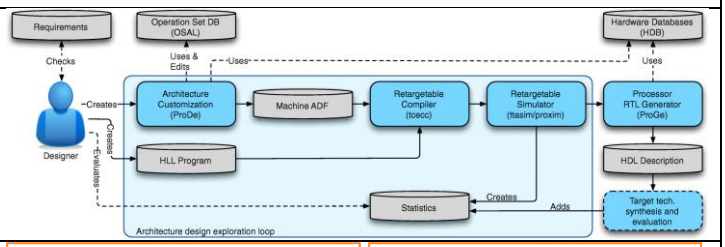
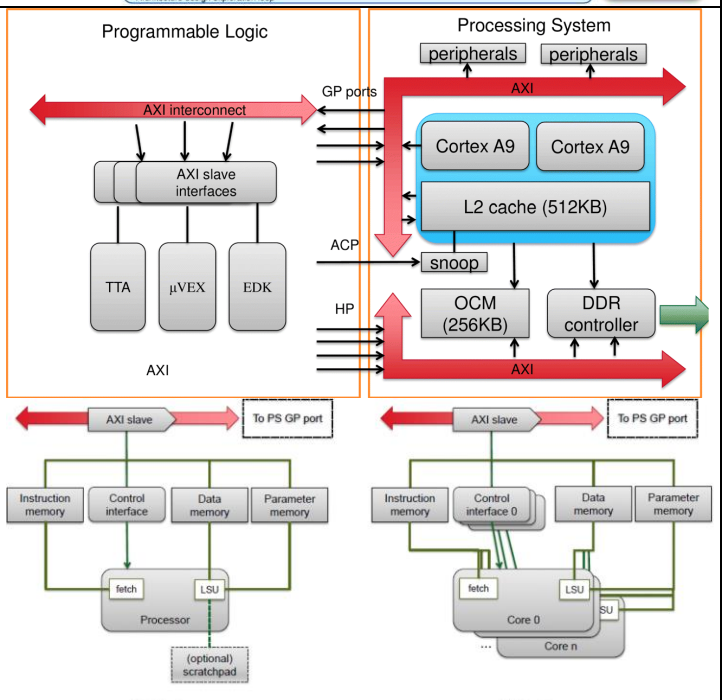
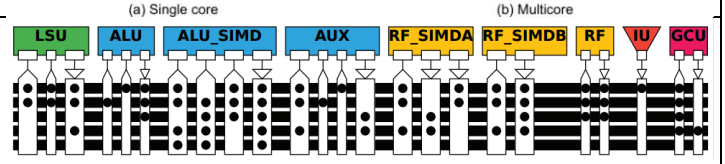
Tool:	WP3 TOOLS and TECHNOLOGIES
10_1	**TCE:** TTA-based Co-design Environment; Custom co-processors
10_2	**HEPSYCODE**: System-level design method
10_3	**MDC:** Multi-Dataflow Composer (requires: Xilinx Vivado HLS)
10_4	**SAGE-VS**: Sage Verification Suite
10_5	**RIE:** Re-configurable Implementation of Embedded systems
10_6	**S3D:** Single Source Design Framework
10_7	**DTRC:** Design Time Resource Configurator (requires: Xilinx SDSoC 2018.2)
10_8	**DTRiMC:** Design Time Resource Integrator of Model Composer IPs (requires: SysGen for DSP, SDSoC 2018.2)
10_9	**IMACS**: Image-in-the-loop control (requires: Simulink, Embedded codder, OpenCV)
10_10	**HDR Core:** HDR Processing accelerator
10_11	**ACF Core:** Object detection accelerator

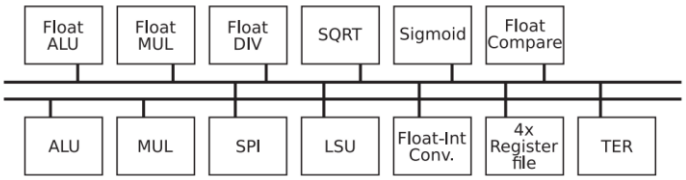
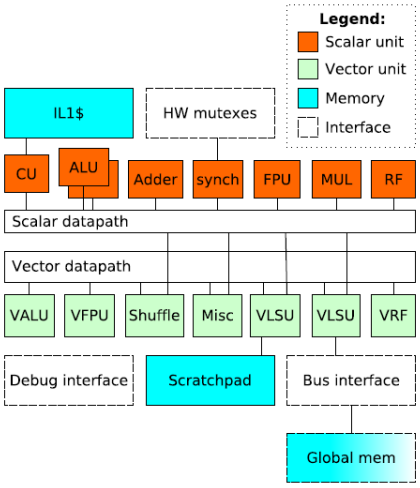
Explanation to the three columns:	
Left	We have used it (y/n/?)
Central	We will continue to use it (y/n/?)
Right	We have evaluated it (y/n/?)

Table 9: Use of WP3 tools and technologies by project partners.

10.1. TTA-Based Co-design Environment (TCE)

Tool/Technology in a Nutshell		An open application-specific instruction-set toolset. It can be used to design and program customized processors based on the energy efficient Transport Triggered Architecture (TTA). The toolset provides a complete re-targetable co-design flow from high-level language programs down to synthesizable processor RTL (VHDL and Verilog back-ends supported) and parallel program binaries. Processor customization points include the register files, function units, supported operations, and the interconnection network.
Key Features – FitOptiVis Starting Point		TCE has been developed and maintained in various research projects since 2002. Some of the key features at the start of FitOptiVis: • Complete runtime re-targetable tool flow from source code down to customized processor and its target-specific binaries • LLVM-based compiler at version LLVM 5.0 • Component library based RTL generation to VHDL and Verilog • Manual processor customization tool steps that can be invoked from the command line to assist in processor design TRL level @ 2018 – 6/7 for the previous features, 2/3 for the new
Intended Users		• Designers of hardware accelerators who could benefit from the flexibility of a software programmed customized co-processor instead • Developers of FPGA soft IP who benefit from the easier way to describe the control using software instead of FSMs • Target at the end of FitOptiVis : Software engineers with no hardware skills that need to develop accelerators: co-processors generated totally automated from software sources with minimal target-specific pragmas etc.
Benefits for the User		• Software programmable, yet very energy efficient accelerators • No “vendor lock-in” of commercial tools since output is targetable to and efficient on different FPGAs and ASIC technologies
Tool/Technolo	**Inputs**	• C (some C++ supported), OpenCL C

gy Requirements		• HDL description of special function units
	Outputs	• RTL (VHDL or Verilog) along with integration/project files for different flows, one of which is AlmaIF which is an IP wrapper developed in ALMARVI project and further developed in FitOptiVis • Architecture description file that drives the different target-specific tools • Program binaries produced from the re-targetable compiler
	Target	• Any ASIC or FPGA technology
	Dependencies	• HW synthesizer, i.e. Vivado or Synopsis tools. • Multiple open source libraries available with liberal licenses (LLVM, wxWidgets, Boost libraries, editline)
Tool/Technology Block Diagram(s)	**TCE Design Flow**	
	Co-processors in AlmaIF IP interface	
Example 1: Custom DSP for Binaural Speaker Localization		 Custom DSP targeted to hearing aid devices with

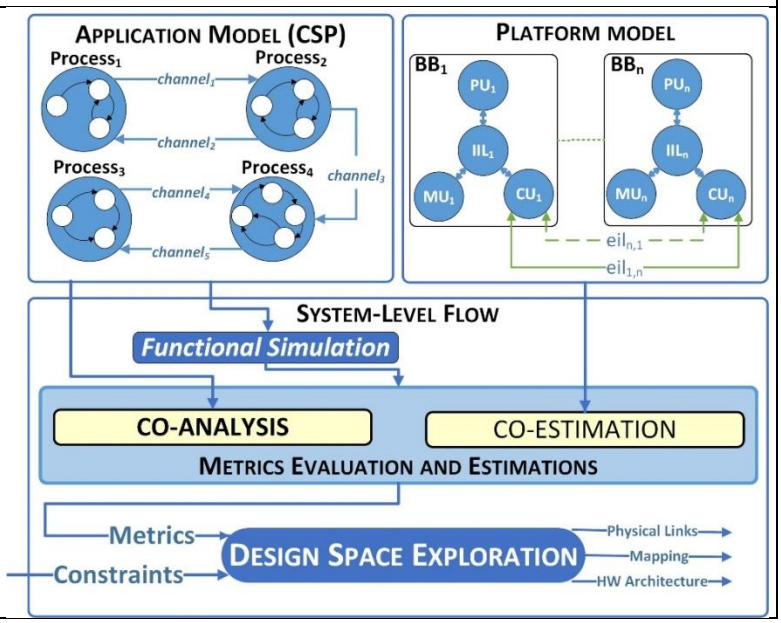
	support for advanced algorithms. 32 x int32 SIMD (1024b) datapath. Synthesized on 28 nm FDSOI. 12 mW at 50 MHz, 1V. 2-split SIMD RF, 1 write port each. Only 10.5% of total power thanks to software bypassing and DRE. Published in 2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS).
Example 2: **A 5.3 pJ/op Approximate TTA VLIW Tailored for Machine Learning**	 Minimum energy point 0.35 V near threshold operating voltage for ultra low power execution. Features for approximate computing. Detect errors in computation, replace with safe values. Manufactured on 28 nm FDSOI. About 320 µW (incl. memories) on ML workloads. Published in Elsevier Microelectronics Journal 61 (2017) 106–113.
Example 3: **LordCore: High Performance Low Power Wide-SIMD Floating Point SDR Multicore**	 32-element FP16 SIMD FUs. Quite generic design, only a few special instructions. OpenCL C programmed. Quad core: 28 nm FDSOI power analysis: 280 mW at 900 MHz, 237 GFLOPS (846 GFLOPS / W). Approx. 18% datapath energy savings through the TTA programming model. **Three orders of magnitude more power efficient than GPU designs.** Closer to fixed function HW power efficiency scale. **Published in IEEE TVLSI in 2019.**
FitOptiVis Technological Advances	Implemented technology additions: • AEx: Fully automated co-processor exploration. This allows using TCE as an HLS engine which produces re-programmable IPs as an output. • Improved FPGA efficiency on the end results for more beneficial soft core use. • More extensive OpenCL support and ONNX input for AI. • Compiler improvements, enable programming also CGRAs with TCE compiler.

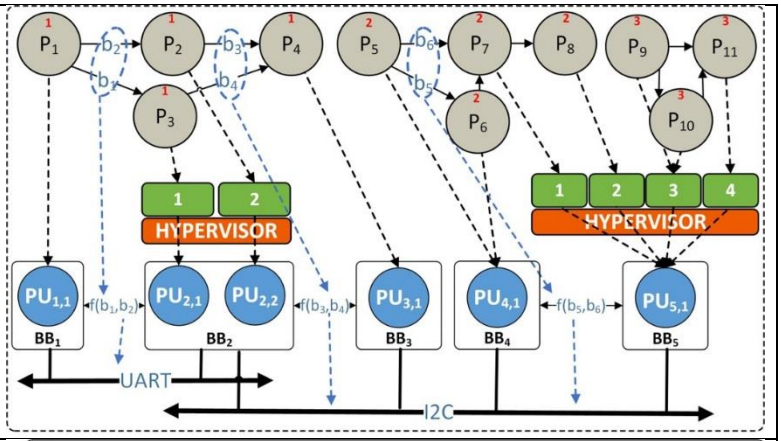
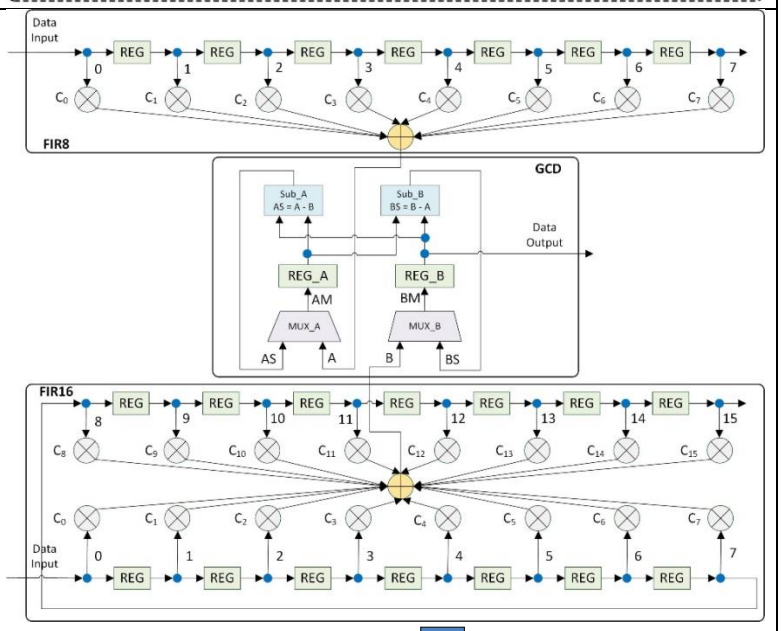
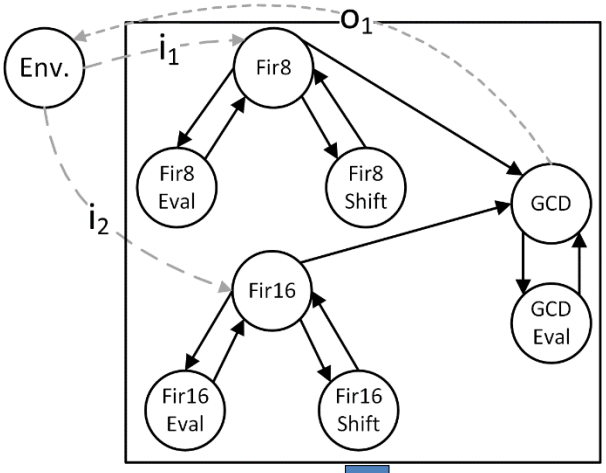
		TRL level of the new planned features @ 2021 – 4/5
Use within FitOptiVis Demonstrators	**Use**	Virtual Reality Use Case: • To produce compiler programmable co-processors for FPGA acceleration of low latency high resolution frame streaming compression using texture compression algorithms
	Foreseen Links	Multi-source Streaming: • Programmable wide-SIMD FPGA soft cores Autonomous Exploration: • Custom wide-SIMD multicore DSP for eventual ASIC implementation and custom SoC integration
Open-Source		http://openasip.org
Licence Type		https://opensource.org/licenses/MIT
Commercial license		N/A

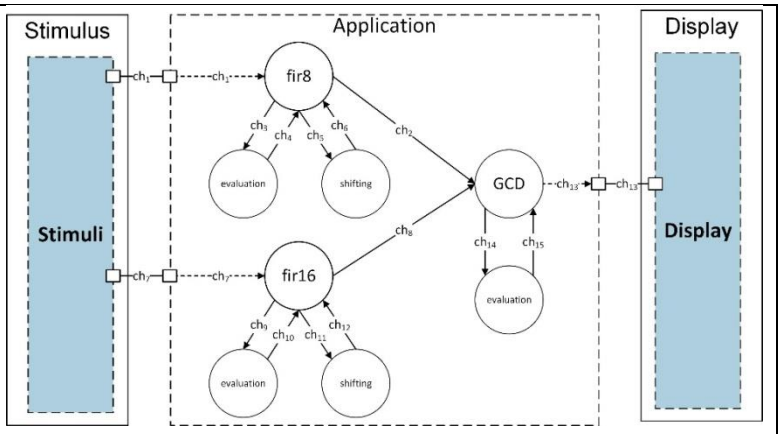
10.2. HW/SW CO-DEsign of HEterogeneous Parallel dedicated SYstems (HEPSYCODE)

Name	HW/SW **CO**-**DE**sign of **HE**terogeneous **P**arallel dedicated **SY**stems (HEPSYCODE)
Tool in a Nutshell	HEPSYCODE is a prototype toolchain that aims to support the design of embedded applications. It is based on a System-Level methodology for HW/SW Co-Design of Heterogeneous Parallel Dedicated Systems. HEPSYCODE uses Eclipse MDE technologies, a customized SystemC simulator and an evolutionary genetic algorithm for HW/SW partitioning, architecture definition and mapping activities, all integrated into an automatic framework that drives the designer from the specification to the implementation.
Key Features – FitOptiVis Starting Point	HEPSYCODE toolchain drives the designer from an Electronic System-Level (ESL) behavioral model, with related NF requirements, including real-time and mixed-criticality ones, to the final HW/SW implementation, considering specific HW technologies, scheduling policies and Inter-Process Communication (IPC) mechanisms. It has been adopted and extended within several European project (i.e., EMC² - Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments, https://www.artemis-emc2.eu/; MegaM@Rt^2 - MegaModelling at Runtime, https://megamart2-ecsel.eu/; AQUAS - Aggregated Quality Assurance for Systems, https://aquas-project.eu/), while it will be improved during FitOptiVis . Features at the start of the project: • HEPSYCODE defines a behavioral modeling language, named HML (*Hepsy Modeling Language*), based on the *Communicating Sequential Processes* (CSP) *Model of Computation* (MoC) and SystemC. By means of HML it is possible to define the *System Behaviour Specification* (SBS), composed by the *System Behavior Model* (SBM), a set of *Non Functional Constraints* (NFC) and a set of *Reference Inputs* (RI) to be used for simulation-based activities. The SBM is a CSP-based executable model of the system behavior that explicitly defines also a model of communication among processes (PS) using unidirectional point-to-point blocking channels (CH) for data exchange. • Designers select basic HW components available to build the final HW platform based on the selected *Target Template Architecture* (TTA). The final HW platform is composed of several basic HW components. These components are collected into a *Technologies Library* (TL). TL can be considered as a

		generic “database” that provides the characterization of the available processor technologies. • HEPSYCODE evaluates and estimates some system metrics that exploits as much information as possible about the system by analysing the SBM, while considering the available basic HW components (i.e., timing performance, cost, energy/power, area). • Finally, HEPSYCODE reference co-design flow reaches the DSE step. Starting mainly from *Application Model* and *Platform Model*, it includes two iterative activities: (1) “*Search Methods*”, that consider HW/SW partitioning, architecture definition and mapping using a genetic algorithm that allows to explore the design space looking for feasible architecture/mapping items suitable to satisfy imposed constraints; (2) “*Timing Co-Simulation*”, that considers suggested mapping/architecture items to actually check for timing constraints satisfaction.
Intended Users		• Embedded systems engineers and designers • Software developers • Hardware architects • EDA industries
Benefits for the User		• Reduce design productivity gap: focus on system-level requirements and get suggestions from the framework about possible implementations able to satisfy them. • Reduce time to market: compare embedded systems designers experience-based intuitions with the ones proposed by the framework to avoid costly early-stage errors. • Find the best design metrics trade-off: define designers’ custom library of basic HW components and let the framework propose how to use them.
Tool Requirements	**Inputs**	• High-level application models (HML - CSP) • UML models represents system behaviour • HW basic components in terms of processors, memories and communication links – XML • Input F/NF requirements and constraints • Test-benches • SystemC behaviour implementation
	Outputs	• HW/SW final architecture: ○ HW/SW CSP process partition on a Heterogeneous multi-processor embedded system composed by different HW components that fulfil architectural constraints, and the mapping between CSP processes and HW components, able to satisfy input

		constraints. ○ Logical and physical links allocation and mapping that fulfil input constraints
	Target	• COTS (i.e., Common-Off-The-Shelf) General-Purpose Processors (GPP, e.g., ARM, MIPS, MicroBlaze, Nios II, etc.); • COTS domain-oriented processors (e.g., DSP, Digital Signal Processor; GPU, Graphical Processing Unit; etc.); • Custom domain-oriented processors (ASIP, Application Specific Instruction-set Processor); • COTS Single-Purpose Processors (SPP, e.g., AES coder, JPEG coder, UART/SPI/I2C Controller, etc.); • Custom Single-Purpose Processor (SPP, i.e., the actual ad-hoc developed digital HW components)
	Dependencies	• EMF technologies • SystemC Library • Embedded Linux Distributions (Petalinux, Gaisler Buildroot) • HW synthesizer, i.e. Vivado • (optional) High Level Synthesis tool to generate the HDL Components
Tool Block Diagram(s)	**HEPSYCODE Design Flow**	

	Results	
Example: **FIR Variable TAP8 filter (FIR8)** **FIR Variable TAP16 filter (FIR16)** **Greatest Common Divisor (GCD)**		 

<table>
<tr>
<td></td>
<td>


Fir-Fir-GCD is a synthetic application that takes in input two values (triggered by Stimulus), makes two filtering actions (FIR8 and FIR16) and then makes the greatest common divisor (GCD) and displays the result.
Figure at the top shows the data flow model associated to the application.
Figure at the center is the FIR-FIR-GCD HML model, where the application is composed of eight processes and twelve channels. Two more processes (Stimulus and Display) and three more channels are then used to describe and connect the testbench (represented by 2 input channel i_1 and i_{72} and 1 output channel o_1).
Finally, it is possible to realize the System Behavioral Model (SBM), represented by the CSP shown in Figure at the bottom, that provides a schematic view of FirFirGCD system, composed of eight processes and twelve channels. Two more processes and three more channels are then used to describe and connect (input signals) the test-bench (output signal).
Stimuli are numerical and random values that represent the system input. This data are sent to two distinct blocks: Fir8 and Fir16. These blocks represent two FIR filters (Finite Impulse Response). The outputs of the filtering operations are then transferred to a GCD block, which evaluates the maximum common divisor of the two values. The FIR blocks computation is divided into two parts: one performs a certain number of multiplications using coefficients (FIR evaluation), while the other part performs shifting operations (FIR shifting).
</td>
</tr>
<tr>
<td>FitOptiVis Technological Advances</td>
<td>
Expected additions:

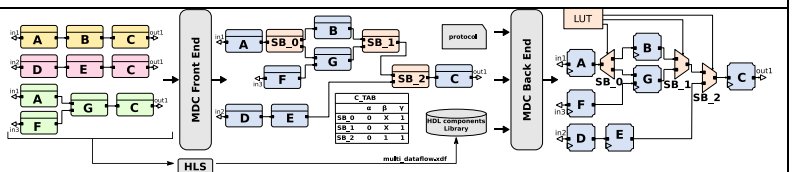
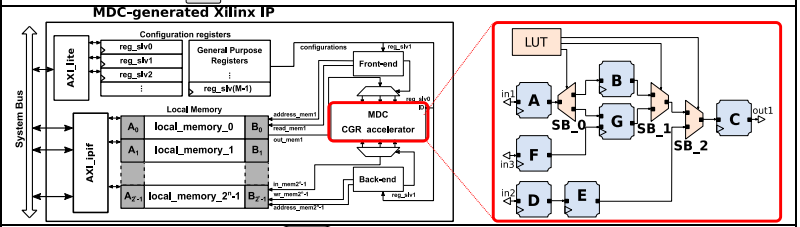
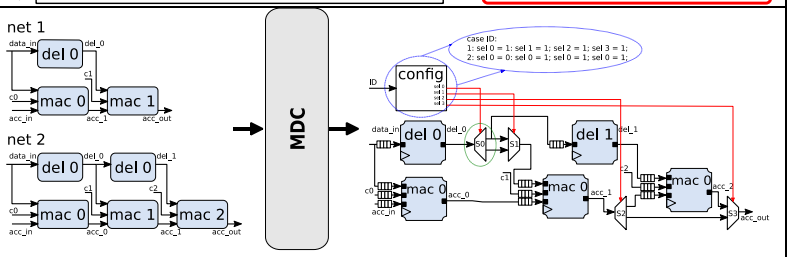
DSE able to consider power/energy constraints at system level
Monitoring support based on AIPHS in order to validate the methodology
Possible runtime adaptive design points based on DSL specifications

</td>
</tr>
</table>

Use within FitOptiVis Demonstrators	**Use**	• Under Evaluation
Open-Source		Git repository: https://bitbucket.org/vittorianomuttillo87/tool-hepsycode/src/master/ Official website: http://www.hepsycode.com
Licence Type		GPL2
Commercial license		N/A

10.3. Multi-Dataflow Composer (MDC) tool

Name		Multi-Dataflow Composer (MDC) tool
Tool/Technology in a Nutshell		MDC tool is an automated dataflow-to-hardware tool for the generation and system integration of Coarse-Grained Reconfigurable datapath/accelerators
Key Features – FitOptiVis Starting Point		MDC tool is the primary outcome of a Sardinian Regional project concluded in 2012 (http://sites.unica.it/rpct/). Along the years, and throughout its adoption within the CERBERO H2020 project (https://cerbero-h2020.eu), it has been extended to its actual definition. Features at the start of FitOptiVis : • composition of different high-level abstract functional specification to be implemented on a single accelerator (implementable both on ASIC and FPGA), based on coarse-grained reconfigurable technologies • automatic resource minimization • automatic reconfiguration management TRL level @ 2018 – 3/4
Intended Users		• Software developers/embedded system engineers with little to no knowledge of the hardware • Hardware architects/embedded system engineers requesting for additional features (e.g. power optimization)
Benefits for the User		• design automation from high level models (dataflows, i.e. xdf files) to hardware • handling of complex and time consuming design issues, such as topology exploration or power optimization • easy system integration within Xilinx platforms
Tool/Technology Requirements	**Inputs**	• high level models (dataflow) of functionalities to be implemented - XDF, Cal • HDL description of the components (HDL Components Library, HCL) corresponding to the dataflow actors, manually or automatically generated - Verilog, VHDL • hardware communication protocol between components - XML
	Outputs	• (baseline) HDL description corresponding to the multi-functional model - Verilog, VHDL • (optional) multi-functional model resulting from the combination of the input applications models - XDF, Cal • (optional) Xilinx IP wrapper logic, scripts and drivers - XML, Verilog, Tcl, C

	Target	• ASIC (baseline, profiling, and power management) • FPGA (baseline, power management, accelerator deployment)
	Dependencies	• HW synthesizer, i.e. Vivado. • (optional) High Level Synthesis tool to generate the HDL Components Library, i.e. Vivado HLS or CAPH.
	....	•
Tool/Technology Block Diagram(s)	**MDC Baseline Flow**	
	MDC Accelerator	
Example: FIR Variable TAP filter		 Example: Given 2 input dataflows (2-tap and a 3-tap FIR filters). Output: accelerator capable of switching among the filters. Four switching elements are inserted automatically to manage reconfiguration (configuration pattern size: 4 bits). APIs for filter delegation are provided.
FitOptiVis Technological Advances		Expected additions: • Multi-Level monitoring support based on AIPHS 2.0 • OpenCL APIs extension TRL level @ 2021 – 4/5
Use within FitOptiVis Demonstrators	**Use**	Water Supply Use Case: • build, manage and monitor application specific HW accelerators
Open-Source		Git access to be provided soon, executable and tutorials already available: http://sites.unica.it/rpct/download/
Licence Type		https://opensource.org/licenses/BSD-3-Clause
Commercial license		N/A

10.4. The SAGE Verification Suite (SAGE-VS)

Name		The SAGE Verification Suite (SAGE-VS)
Tool in a Nutshell		The SAGE Verification Suite (SAGE-VS) is a set of SW tools aimed to accomplish different formal verification tasks at design time.
Key Features – FitOptiVis Starting Point		The SAGE-VS has been designed and developed (from TRL 0/1) in the context of the CERBERO H2020 project (https://cerbero-h2020.eu. At the start of FitOptiVis , it was composed of the following tools: • ReqV: a tool for formal consistency checking of requirements. • Hydra: a domain-independent tool for Goal-Oriented control of Cyber-Physical Systems. TRL level @ 2018 – 3/4
Intended Users		• [ReqV] Requirements engineers without any prior knowledge related to formal methods. • [ReqV] Software developers without any knowledge of formal methods and logical languages. • [ReqV] System engineers interested to formally verify a model w.r.t. some properties. • [Hydra] System engineer interested in generating controllers from a system model. • [ReqT] Software developers without any knowledge of formal methods and logical languages.
Benefits for the User		• [ReqV] Automated consistency checking of a set of requirements written in controlled natural language. • [ReqV] No prior knowledge related to specification languages is required to input the requirements (GUI support). • [ReqV] Human-readable feedback in the case of inconsistent requirements. • [ReqV] Domain and application independent. • [Hydra] Domain independent through the use of high level models of the system. • [Hydra] No prior knowledge of the inner working of planning algorithms. • [ReqT] Automated testing of the implemented system with respect to the requirements formalized and verified in ReqV.
Tool Requirements [ReqV]	**Inputs**	Set of requirements in natural (controlled English) language, formulated as Property Specification Patterns for Linear Temporal Logic (LTL) extended to constrained numerical signals
	Outputs	Consistency result (yes/no). In the case of inconsistency,

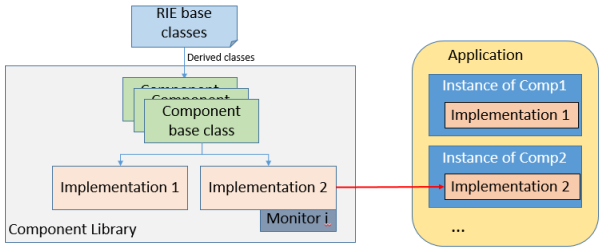
		the tool returns the minimalset of requirements that causes the inconsistency.
Tool Requirements [ReqT]	**Input**	A set of requirements formalized and verified with ReqV and the system to test.
	Output	A list of tests (i.e., sequences of inputs and outputs assignments) executed on the system under test (SUT) and their corresponding evaluation (passed/failed).

Tool Requirements [Hydra]	**Inputs**	Requires a hybrid model of the system:
		• definition of the state of a system
		• definitions of the system's capabilities
		  ○ available discrete actions and their effect on the system and its environment
		  ○ operating limits of the controller
		• safety limits
		
		Specification of a target problem: initial state, goal state of the system and invariants that should hold.

	Outputs	A yes/no answer on whether the system can be used to achieve the tested use case. A yes answer comes with a correct by design plan to achieve the given objective. The plan accounts for both the discrete and continuous limits of the system so that the plan is valid and guaranteed to be executable and thus constitute a proof that the system has the targeted capability.
Examples		[ReqV] A requirement engineer has to start the requirements definition of a new system. She opens the browser, logs in into ReqV and creates a new project. In the project, she starts adding requirements one by one, with the support of the GUI. When she has finished, she presses the verification button, and finds out that the specification is inconsistent. Therefore, she runs the inconsistency explanation task, and after few minutes ReqV returns a list of few requirements that are conflicting. The engineer inspects those requirements and fixes the problem. She runs again the verification button and this time ReqV reports that everything is ok. One month later, a client asks for the introduction of a new feature. The requirements engineer enters in ReqV again and inserts the new requirements. Running the verification task, she finds out that one of such requirements conflicts with an old one. She returns to the client and discuss the issue. They decide to modify the old requirements so to be compliant with the new ones. The requirements engineer updates the requirements in ReqV accordingly, and this time the verification process returns a positive answer. [Hydra]: Let us consider a robotic manipulator. The mobile manipulator must be operated in a constrained

		environment in order to move objects into target locations. The system engineer has a specification of the controller which include the discrete actions (e.g. release object, scan environment) and limits of the system and of its controller (e.g. joint limits, maximal acceleration).Based on this model, the system engineer can test whether the currently designed system is capable of fulfilling a particular use case where Hydra will autonomously explore the set of possible high-level and low-level controls to achieve the target task. This would allow verifying that the system design is adapted to the targeted use case and catch modeling errors early in the design process. Once the de-sign process is finished, Hydra can also be used as a goal-oriented controller to exploit the system. [ReqT]: After the system has been implemented, the software engineer wants to check if the implemented system is compliant with the requirements defined at the beginning of the design process. Firstly, the user exports the formalized and verified requirements from ReqV and save them in a text file. Secondly, the user writes a small wrapper to let ReqT interact with the system (also called System Under Test, or SUT for short). Therefore, the user starts ReqT on her/his desktop and a simple GUI appears, in which the user can select the requirement file, the SUT wrapper and set few more options. Once the user finished, she/he presses the run button and ReqT starts to generate and executed some tests on the SUT. At the end of the process a report appears, showing the executed tests and their status. The user discovered that few tests fail, so she/he double clicks on them to see the details of the execution. Hence, the user returns to the system source code and checks the faulty behaviours. She/he finds a bug and fixes it, then, she/he repeats the test execution. This time all tests are successful, so the users can finally deploy the system.
FitOptiVis Technological Advances		Expected additions: • ReqV: extend the expressivity of input PSPs to allow the translation in a logic language for hybrid systems and improve the usability of the GUI. TRL level @ 2021 – 4/5
Use within FitOptiVis Demonstrators	**Use**	Water Supply Use Case: • build, manage and monitor application specific HW accelerators
Open-Source		https://gitlab.sagelab.it/sage/ReqV https://gitlab.sagelab.it/sage/ReqT
Licence Type		LGPL
Commercial license		N/A

10.5. RIE – Re-configurable Implementation of Embedded systems

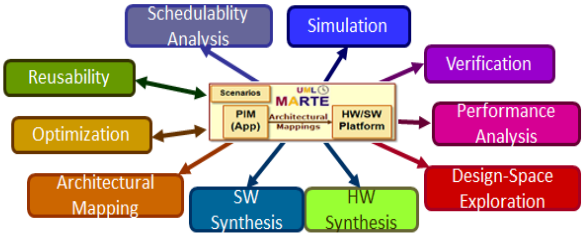
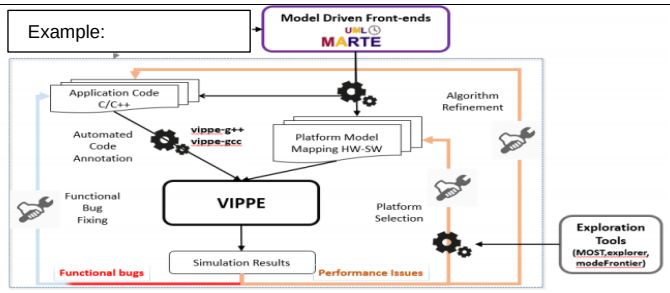
Name		RIE – Re-configurable Implementation of Embedded systems
Tool in a Nutshell		Methodology and C++ library for component-based implementation of embedded systems. The library classes are used to implement components and monitors. RIE provides support for runtime re-configuration of software components. In the RIE methodology, a component could have several implementations that are selected at runtime. Additionally, the library also simplifies component deployment in the cloud and edge.
Key Features – FitOptiVis Starting Point		The library development has been started in FitOptiVis . Therefore it is a complete FitOptiVis result.
Intended Users		• Component-based embedded system developers that require software component reconfiguration at runtime.
Benefits for the User		• Simplify and standardize component-based development. • Integrate software reconfiguration • Provide a common framework to access different monitoring strategies. By default, the library supports lttng monitors on linux but it could be adapted to other methodologies. Currently, we are working with UAQ to support hardware monitors. We plan to support CUNI monitors.
Tool Requirements	**Inputs**	• A system description that uses RIE-based component.
	Outputs	• A system implementation that can be reconfigurable and traced at runtime.
	Target	• Networked embedded systems.
	Dependencies	• C++11 compiler. For edge computing, protocol buffer and grpc. For linux event monitoring, lttng.
	....	•
Tool Block Diagram(s)	**MDC Baseline Flow**	
	MDC Accelerator	

<table>
<tr><td colspan="2">Example:
FIR Variable TAP filter</td><td>

```
//System component implementations
    cameraimp1 c1("camera1");
    cameraimp2 c12("camera2");
    cameraimp3 c13("camera3");
    rgb2grayimp1 c2("rgb2gray1");
    sobelimp1 c3("sobel1");
    sobelimp2 c32("sobel2");
    displayimp1 c4("display1");
    imagenimp c5("imagen1");
    imagen_grayimp c6("imagen_gray1");
    imagen_edgesimp c7("imagen_edges1");


//Instances
    inst<camera> ic1;
    inst<rgb2gray> ic2;
    inst<sobel> ic3;
    inst<display> ic4;
    inst<imagen> ic5;
    inst<imagen_gray> ic6;
    inst<imagen_edges> ic7;

//Association between components and default implementation
    ic1.set(&c1);
    ic2.set(&c2);
    ic3.set(&c32);
    ic4.set(&c4);
    ic5.set(&c5);
    ic6.set(&c6);
    ic7.set(&c7);

//Association of provider & required services
    ic2.prov_services(&ic5);
    ic3.prov_services(&ic6);
    ic4.prov_services(&ic7);
    ic5.prov_services(&ic1);
    ic6.prov_services(&ic2);
    ic7.prov_services(&ic3);
```

</td></tr>
<tr><td colspan="2">FitOptiVis Technological Advances</td><td>• Simplify component-based development
• Support data-flow and service-oriented architectures.
• Provide software re-configuration capability
• Support different types of monitor implementation.</td></tr>
<tr><td rowspan="2">Use within FitOptiVis Demonstrators</td><td>Already Planned Use</td><td>• RIE is used to implement the autonomous exploration use case.</td></tr>
<tr><td>Potential Foreseen Links</td><td>TBD. Available 1Q-2020</td></tr>
<tr><td colspan="2">Open-Source</td><td>Yes</td></tr>
<tr><td colspan="2">Licence Type</td><td>GPL after publication</td></tr>
<tr><td colspan="2">Commercial license</td><td>Yes</td></tr>
</table>

10.6. S3D – Single Source Design Framework

Name		S3D – Single Source Design Framework
Tool/Technology in a Nutshell		UML/MARTE based framework that provides model capture, performance analysis and SW code synthesis.
Key Features – FitOptiVis Starting Point		The S3D framework is mainly oriented to service architecture (SOA). The framework has been extended in FitOptiVis to efficiently support video/image processing application with software-reconfiguration capabilities.
Intended Users		• HW/SW system development • Embedded system application designers
Benefits for the User		• Use an UML standard for software development • Performance analysis integration (VIPPE) • Automatic software synthesis (essyn) that support different MoCs (model of computations).
Tool Requirements	**Inputs**	• UML/MARTE models
	Outputs	• Performance estimation of different implementations • C++ implementation templates
	Target	• Edge and cloud computing
	Dependencies	• Eclipse, Papyrus, llvm.
	....	•
Tool Block Diagram(s)	**MDC Baseline Flow**	
	MDC Accelerator	
Example: FIR Variable TAP filter		
FitOptiVis Technological Advances		

Use within FitOptiVis Demonstrators	**Use**	• Use in Autonomous Exploration use case
Open-Source		Yes. Visit http://umlmarte.teisa.unican.es/
Licence Type		Free for research
Commercial license		Contact villar@teisa.unican.es

10.7. Design Time Resource Configurator (DTRC) Technology

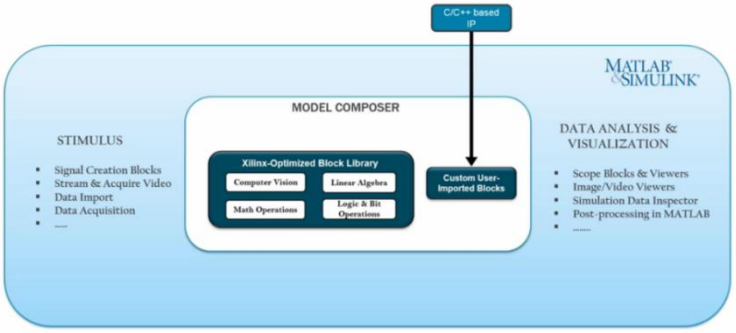
Name		Design Time Resource Configurator (DTRC) technology
Technology in a Nutshell		DTRC technology is design-time resource for configuration and system integration of FitOptiVis design time resources for Zynq and Zynq Ultrascale+ systems with Debian OS and HW accelerators which can be generated from C/C++ by Xilinx SDSoC system level compiler.
Key Features – FitOptiVis Starting Point		DRTC technology extends the board-support bring-up scripts provided by company Trenz Electronic https://www.trenz-electronic.de/ for Zynq and Zynq Ultrascale+. See: http://sp.utia.cz/index.php?ids=projects/almarvi ECSEL JU project ALMARVI. Features at the start of FitOptiVis : • Support for Xilinx SDSoC 2015.4 standalone Zynq modules without OS with Python 1300 Video sensor or Full HD HDMI Video I/O. TRL level @ 2017 – 4/5.
Intended Users		• Software developers/embedded system engineers with little to no knowledge of the hardware • Hardware architects/embedded system engineers requesting configuration of the Debian OS and support for HW accelerator design flow which can be generated from C/C++ by Xilinx SDSoC system level compiler.
Benefits for the User		• design automation of configuration of Debian OS with SDSoC support • integration of Full HD video input and Video output
Tool/Technology Requirements	**Inputs**	• HW module description files from Trenz Electronic https://www.trenz-electronic.de/ • Petalinux configuration files • SW C/C++ functions and main programs for the SDSoC compiler.
	Outputs	• Board support package describing HW for Petalinux OS kernel, Debian OS file system and for the SDSoC compiler. • Configured and compiled Xilinx Petalinux kernel with installed and compiled Xilinx SDSoC support drivers for the DMA and Scatter Gather (SG) DMA data transfers to/from HW accelerators. • Configured and compiled Debian OS file system in form of SD card image with two partitions: • FAT32 Win7/Win10 compatible partition for file transport • Configured and populated Debian file system partition • Configured support for X11 Desk top GUI on separate Full HD Display • Configured SW projects for the SDSoC compiler. Project can be executed with actual video I/O in SW on ARM. Projects can be compiled by SDSoC compiler and then executed in

		HW with ARM SW support and video I/O. • Support for Arrowhead framework 4.0 compatible C/C++ SW clients.
	Targets	• Module with Xilinx Zynq 7010 in Raspberry Pi Form Faktor ZynqBerry PCB TE0726-03M https://shop.trenz-electronic.de/en/TE0726-03M-ZynqBerry-Module-with-Xilinx-Zynq-7010-in-Raspberry-Pi-Form-Faktor?c=350 • MPSoC Module with Xilinx Zynq UltraScale+ ZU4EV-1E, 2 GByte DDR4 SDRAM https://shop.trenz-electronic.de/en/TE0820-03-4DE21FA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU4EV-1E-2-GByte-DDR4-SDRAM-4-x-5-cm Carrier Board for Trenz Electronic 7 Series https://shop.trenz-electronic.de/en/TE0701-06-Carrier-Board-for-Trenz-Electronic-7-Series?c=261 • UltraSOM+ MPSoC Module with Zynq UltraScale+ XCZU15EG-1FFVC900E, 4 GB DDR4 https://shop.trenz-electronic.de/en/TE0808-04-BBE21-A-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-XCZU15EG-1FFVC900E-4-GB-DDR4 • UltraITX+ Baseboard for Trenz Electronic TE080X UltraSOM+ https://shop.trenz-electronic.de/en/TEBF0808-04A-UltraITX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261
	Dependencies	• Xilinx Vivado HLS High Level Synthesis tool version 2018.2 • Xilinx SDSoC system level compiler version 2018.2 • Xilinx Petalinux version 2018.2
	Open source	• Debian “Stretch” repositories for 32bit ARM A9 and 64 bit ARM A53 • Ubuntu 16.04 LTE is needed for the automated configuration of Xilinx Petalinux kernel and for generation of Debian file system.
Tool/Technology	**DTRC Baseline Flow**	• On Win7/Win10/Ubuntu 16.04: Compile HW and export hdf file. • On Ubuntu 16.04: Configure and compile Petalinux and Debian. • On Win7/Win10/Ubuntu 16.04: Compile in SDSoC 2018.2 HW accelerators from C/C++ functions to HW. Run on supported boards.
	DTRC Packages and Application notes	[7.1] Design Time and Run Time Resources for the ZynqBerry Board TE0726-03M with SDSoC 2018.2 Support [7.2] Design Time and Run Time Resources for Zynq Ultrascale+ TE0820-03-4EV-1E with SDSoC 2018.2 Support [7.3] Design Time and Run Time Resources for Zynq

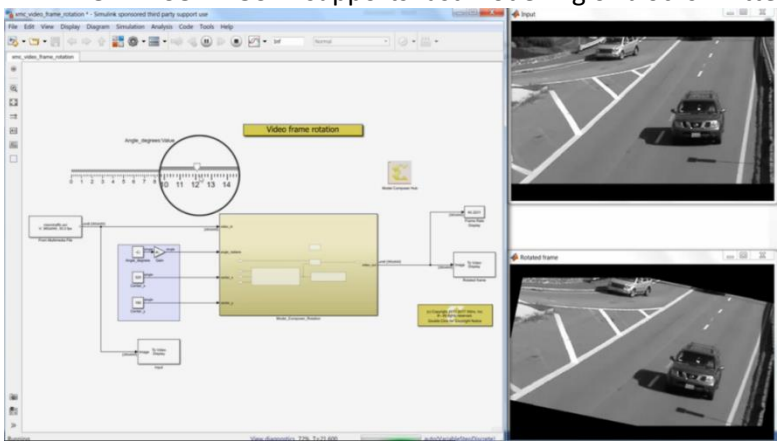
		Ultrascale+ TE0808-04-15EG-1EE with SDSoC 2018.2 Support TRL level @ 8.2019 – 5/6
FitOptiVis Technological Advances		Expected additions: Support for runtime reconfiguration of complete programmable logic part of the device with Quick-time GUI TRL level @ 2019 – 3/4, @ 2020 – 4/5, @ 2021 – 5/6
Use within FitOptiVis		DRTC technology and boards are evaluated by 8 FitOptiVis partners (6x ZynqBerry board, 2x Zynq Ultrascale+ board): UNIVAQ, UNICA,VISIDON, CUNI, TUT, UWB, UTU and UTIA
Use within FitOptiVis Demonstrators	**Links**	Robotic Use Case: Build HW accelerators on the ZynqBerry board
	Access	http://sp.utia.cz/index.php?ids=projects/fitoptivis
Licence Type		Open Source license with these exceptions: (1) UTIA video I/O drivers provided as pre-compiled libraries. (2) Vivado HLS and SDSoC 2018.2 require commercial license from Xilinx: https://www.xilinx.com/ .

10.8. Design Time Resource Integrator of Model Composer IPs (DTRiMC) Technology.

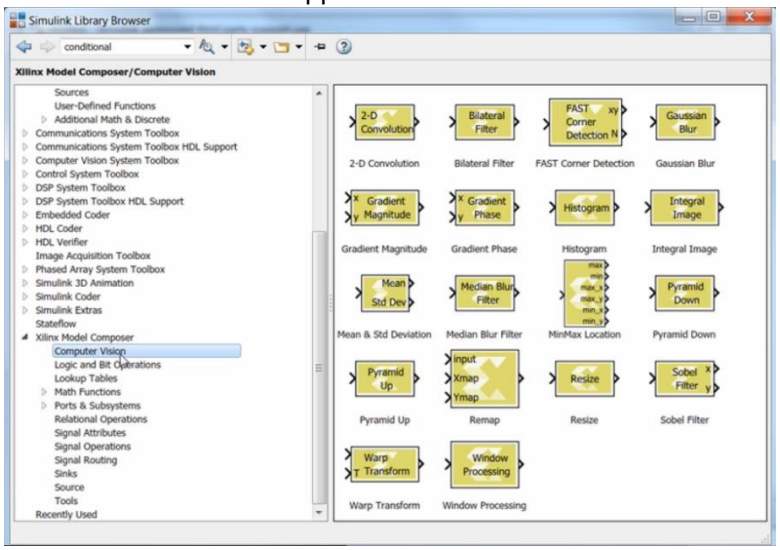
Name	Design Time Resource Integrator of Model Composer IPs (DTRiMC) Technology
Technology in a Nutshell	DTRiMC technology [7.12], [7.13] serves for FitOptiVis system integration of IPs designed, modelled and validated in Xilinx Model Composer (MC) and Xilinx System Generator for DSP (SG for DSP). DTRiMC technology supports integration of MC IPs into Zynq (32bit) and Zynq Ultrascale+ (64bit) systems by support of automated generation of (1) HW data movers IPs; (2) SW API needed for the 32bit DMA or SG DMA or Zero Copy based data movers. DTRiMC tool automates generation of needed HW support for communication with the user defined C/C++ SW applications. SW applications run in user space of Debian OS on Arm A9 (Zynq) or on Arm A53 (Zynq Ultrascale+). DTRiMC technology targets platforms supporting the Xilinx SDSoC 2018.2 compiler for Zynq and Zynq Ultrascale+. These platforms are generated by the FitOptiVis Design Time Resource Configurator (DTRC) technology.
Key Features – FitOptiVis Starting Point	DTRiMC technology is extending the Board support bring up scripts provided by company Trenz Electronic https://www.trenz-electronic.de/ for Zynq and Zynq Ultrascale+. See http://sp.utia.cz/index.php?ids=projects/almarvi ECSEL JU project ALMARVI. Features at the start of FitOptiVis: • Support for Xilinx SDSoC 2015.4 standalone Zynq modules without OS with Python 1300 Video sensor or Full HD HDMI Video I/O. TRL level @ 2017 – 4/5.
Intended Users	• Software developers/embedded system engineers with little to no knowledge of the hardware. • Hardware architects/embedded system engineers requesting DMA connections of Debian application with HW accelerator IP. The integrated HW IP is imported from MC via SG for DSP.
Benefits for the User	• DTRiMC technology supports integration of HW IPs from Xilinx MC and SG for DSP models by automation of DMA connection to Debian app. • DTRiMC technology supports integration of MC HW IPs with Full HD video input/output for Zynq and Zynq Ultrascale+ systems with Debian OS. Key features of the supported Xilinx Model Composer framework:



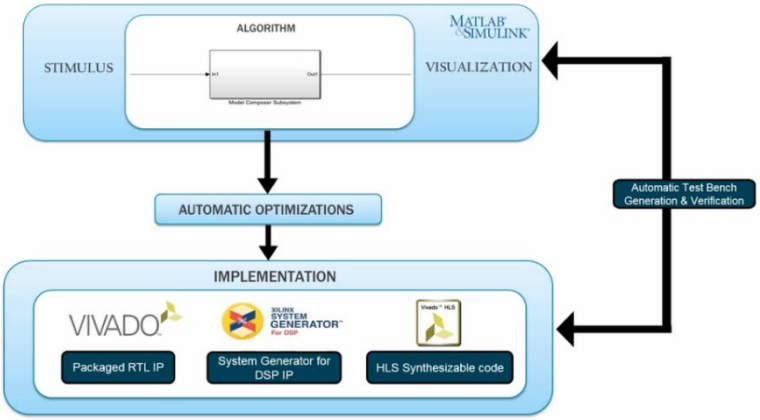
MODEL COMPOSER supports fast modelling of blocks written in C



MODEL COMPOSER supports models with Video data from file system



MODEL COMPOSER supports Computer Vision and Math blocks



MODEL COMPOSER supports generation of IP cores for:

- Xilinx VIVADO (in form of packed RTL HDL IP cores)
- Xilinx SG for DSP (in form of RTL HDL subsystems)
- Xilinx Vivado HLS compiler (in form of synthesizable C++ code)

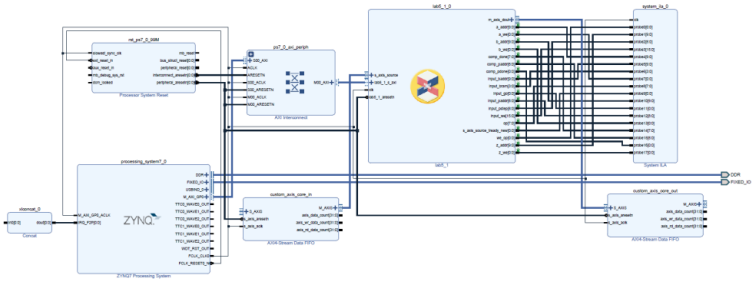
Model Composer (MC) and System Generator for DSP (SG for DSP) are commercial tools provided by Xilinx.
https://www.xilinx.com/video/hardware/model-composer-product-overview.html
https://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html
MC supports model based design, simulation and HW IP generation. It targets Xilinx FPGAs/SoCs via generated IP cores for Vivado flow.
MC targets the Vivado design flow directly (in form of Vivado HLS SW) or indirectly via the integration/simulation in the SG for DSP. MC and SG for DSP work both with support from Matlab and Simulink. SG for DSP supports finite state machines and logic blocks defined as user defined special Matlab m-code functions. These m-code functions are compiled via conversion to C source code into binary format to accelerate simulation. Complete system composed from these blocks can be compiled to HDL RTL. SG for DSP targets Xilinx FPGAs/SoCs via generated packed IP cores for Vivado design flow.
SG for DSP supports bit-exact and cycle accurate modelling. It is usually an order of magnitude faster than the bit-exact and cycle-accurate simulation of hdl RTL code in tools like Questa or the Vivado hdl simulator). SG for DSP supports inclusion, bit-exact and cycle accurate simulation and RTL IP generation from user-defined SW blocks coded in Vivado HLS C++. SG for DSP supports inclusion, bit-exact and cycle accurate simulation and RTL IP generation for RTL hdl subsystems exported from Xilinx Model Composer to Xilinx SG for DSP. SG for DSP serves in this case as a common, bit-exact and cycle accurate simulation environment.
MC supports only bit-exact modelling. MC blocks can process large objects like matrices or video frames and process them by algorithms defined in C SW code. MC simulation can be an order of magnitude faster than the cycle accurate simulation in the SG for DSP. Acceleration is significant, especially for modelling and design of video processing IPs.
MC supports video I/O from/to files, visualisation of video with relatively high FPS.

<table>
<tr><td rowspan="2">Tool/Technology Requirements</td><td>Inputs</td><td>

User defined HW IP designed and tested in MC and exported via the SG for DSP into HW IP core for Vivado 2018.2. The HW IP must have:

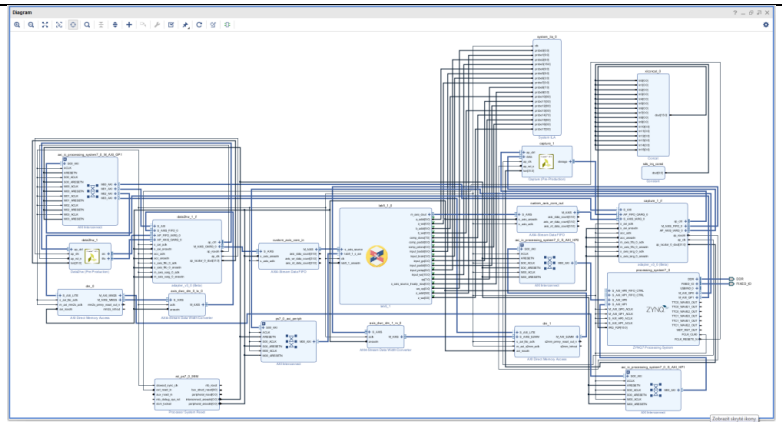
One 32bit AXI-stream input.
One 32bit AXI-stream output.
One 32bit AXI-lite interface to 32bit control registers.

HW module description files (version 2018.2) from Trenz Electronic. Configuration files for Xilinx Petalinux (version 2018.2)
User defined application SW C/C++ for the Debian OS user space.


Input to the DTRiMC tool: Exported MC IP block in base Zynq system.
</td></tr>
<tr><td>Outputs</td><td>

Board support package describing HW for Petalinux OS 2018.2 kernel, Debian OS file system and for the SDSoC 2018.2 compiler with integrated user defined IP designed and tested in Xilinx Model Composer.
Configured and compiled Xilinx Petalinux kernel with installed and compiled Xilinx SDSoC support drivers for the DMA and Scatter Gather (SG) DMA data transfers to/from HW accelerators.
Configured and compiled Debian OS file system in form of SD card image with two partitions:
FAT32 Win7/Win10 compatible partition for file transport
Configured and populated Debian file system partition
Configured support for X11 Desk top GUI on separate Full HD Display
Configured SW projects for the SDSoC compiler. Project can be executed with actual video I/O in SW on ARM. Projects can be compiled by SDSoC compiler and then executed in HW with ARM SW support and video I/O as user defined app. Code for Debian.
Support for Arrowhead framework 4.0 compatible C/C++ SW clients for authentication and management of Ethernet access rights.
Support for FiVis compatible C++ clients for Ethernet data transfer and visualisation via FiVis server. FiVis server generates graphical visualisation pages accessible from standard www browsers.

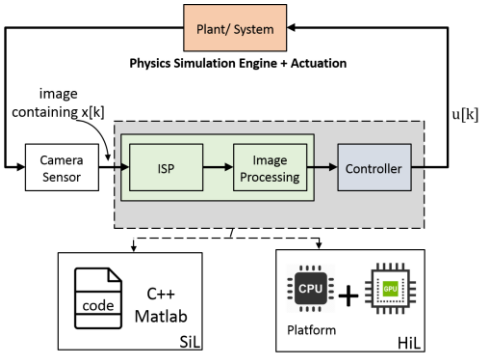
</td></tr>
</table>

		 Output from DTRiMC: MC IP block integrated with DMA I/O to Debian
	Targets	• Module with Xilinx Zynq 7010 in Raspberry Pi Form Faktor ZynqBerry PCB TE0726-03M https://shop.trenz-electronic.de/en/TE0726-03M-ZynqBerry-Module-with-Xilinx-Zynq-7010-in-Raspberry-Pi-Form-Faktor?c=350 • MPSoC Module with Xilinx Zynq UltraScale+ ZU4EV-1E, 2 GByte DDR4 SDRAM https://shop.trenz-electronic.de/en/TE0820-03-4DE21FA-MPSoC-Module-with-Xilinx-Zynq-UltraScale-ZU4EV-1E-2-GByte-DDR4-SDRAM-4-x-5-cm Carrier Board for Trenz Electronic 7 Series https://shop.trenz-electronic.de/en/TE0701-06-Carrier-Board-for-Trenz-Electronic-7-Series?c=261 • UltraSOM+ MPSoC Module with Zynq UltraScale+ XCZU15EG-1FFVC900E, 4 GB DDR4 https://shop.trenz-electronic.de/en/TE0808-04-BBE21-A-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-XCZU15EG-1FFVC900E-4-GB-DDR4 UltraITX+ Baseboard for Trenz Electronic TE080X UltraSOM+ https://shop.trenz-electronic.de/en/TEBF0808-04A-UltraITX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261
	Depend en-cies comme rcial tools	• Matlab, Version 9.3 (R2017b) MathWorks (commercial tool) • Simulink Version 9.0 (R2017b) MathWorks (commercial tool) • Fixed-Point Designer toolbox Version 6.0 (R2017b) MathWorks (commercial tool) • System Generator for DSP toolbox 2018.2 Xilinx (commercial tool) • SG for DSP 2018.2 Xilinx (commercial tool) • Vivado HLS High Level Synthesis tool 2018.2 Xilinx (commercial tool) • SDSoC system level compiler 2018.2 Xilinx (commercial tool)
	Depen-dencies Open source tools	• FitOptiVis Design Time Resource Configurator (DTRC) tool (UTIA) • Petalinux 2018.2 (Xilinx) • Debian “Stretch” repositories for 32bit ARM A9 and 64 bit ARM A53 • Ubuntu 16.04 LTE is needed for the automated configuration of Petalinux kernel and for generation of the Debian file-system.

		• FitOptiVis FiVis tool for remote data visualisation (optional) • ArrowHead Framework 4.0 tool for access management (optional)
Tool/Technology	**DTRiMC extends DRTC techno-logy**	DTRiMC app. notes and evaluation packages [7.12], [7.13] released in Y2 have extended FitOptiVis DTRC [7.1], [7.2], [7.3]. DTRiMC app. notes and evaluation packages [7.14], [7.15], [7.16], [7.17] [7.18], [7.19], [7.20] released in Y3 have extend FitOptiVis DRTC [7.12], [7.13].
	Base DTRC Packages	[7.12] http://sp.utia.cz/index.php?ids=results&id=te0726_fp01x8 [7.13] http://sp.utia.cz/index.php?ids=results&id=te0820_fp03x8x2s [7.14] http://sp.utia.cz/index.php?ids=results&id=TS74fp03x8 [7.15] http://sp.utia.cz/index.php?ids=results&id=2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu4ev [7.16] http://sp.utia.cz/index.php?ids=results&id=2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu3cg [7.17] http://sp.utia.cz/index.php?ids=results&id=2018_2_te0808_fp03x8_4x2_ila_mulf64_DTRiMC [7.18] http://sp.utia.cz/index.php?ids=results&id=2018_2_te0726_fp01x8_ila_DTRiMC [7.19] http://sp.utia.cz/index.php?ids=results&id=2017_4_te0808_fp03x8_4x2_ila_mulf64_DTRiMC [7.20] http://sp.utia.cz/index.php?ids=results&id=2018_2_te0726_07s_ila_DTRiMC
FitOptiVis Technological Advances		DTRiMC technology supports export of Model Composer IP (as SG for DSP) IP to Zynq and Zynq Ultrascale+ SoCs . DTRiMC technology generates DMA HW/SW data movers for of Model Composer IPs.
Use within FitOptiVis		DTRiMC technology is released for public access [7.12]- [7.20] for Zynq & Zynq Ultrascale+. It is used by UTIA.
Use within FitOptiVis Demonstrators	**Links App. Notes**	[7.12] FP01x8 Accelerator on TE0726-03M http://sp.utia.cz/results/te0726_fp01x8/AppNote-FitOptiVis-te0726_fp01x8_short.pdf [7.13] Two serial connected evaluation versions of FP03x8 accelerators for TE0820-03-4EV-1E module on TE0701-06 carrier board http://sp.utia.cz/results/te0820_fp03x8x2s/AppNote-FitOptiVis-te0820_fp03x8x2s.pdf [7.14] Eight FP03x8 accelerators for TE0808-09-EG-ES1 module on

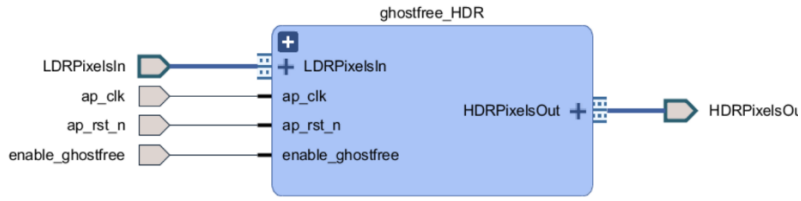
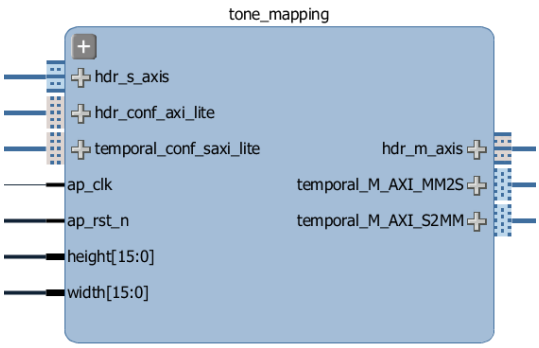
		TEBF0808 carrier board. AppNote-2017_4-te0808_fp03x8_4x2.pdf (utia.cz) [7.15] DTRiMC tool for TE0820-03-4EV-1E module on TE0701-06 carrier board http://sp.utia.cz/results/2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu4ev/AppNote_2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu4ev.pdf [7.16] DTRiMC tool for TE0820-02-3CG-1E module on TE0701-06 carrier board http://sp.utia.cz/results/2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu3cg/AppNote_2018_2_te0820_fp03x8_1x2_ila_DTRiMC_zu3cg.pdf [7.17] DTRiMC tool for TE0808-15-EG-1EE module on TEBF0808 carrier board. http://sp.utia.cz/results/2018_2_te0808_fp03x8_4x2_ila_mulf64_DTRiMC/AppNote_2018_2_te0808_fp03x8_4x2_ila_mulf64_DTRiMC.pdf [7.18] DTRiMC tool for TE0726-03M board. http://sp.utia.cz/results/2018_2_te0726_fp01x8_ila_DTRiMC/AppNote_2018_2_te0726_fp01x8_ila_DTRiMC.pdf [7.19] DTRiMC tool for TE0808-09-EG-ES1 module on TEBF0808 carrier board. http://sp.utia.cz/results/2017_4_te0808_fp03x8_4x2_ila_mulf64_DTRiMC/AppNote_2017_4_te0808_fp03x8_4x2_ila_mulf64_DTRiMC.pdf [7.20] Data Movers in DTRiMC tool for TE0726-03M-07S board http://sp.utia.cz/results/2018_2_te0726_07s_ila_DTRiMC/AppNote_2018_2_te0726_07s_ila_DTRiMC.pdf
	Access	Evaluation SD cards for ZynqBerry TE0726, Ultrascale+ TE0820.
Licence Type		Open source license with these exceptions: (1) UTIA Video I/O interfaces for boards supported by the FitOptiVis are provided only as pre-compiled Arm A9 and Arm A53 SW libraries. (2) Vivado, SDSoC, MC , SG for DSP 2018.2 require licensing from Xilinx. (3) Matlab, Simulink, Fixed-Point Designer require MathWorks license. (4) Evaluation versions of integrated HW IPs have evaluation license enabling evaluation, but limiting permanent use of these IPs in the final applications. (5) Release version of integrated HW IPs requires NDA with UTIA and commercial license from UTIA.

10.9. IMACS (IMAge in the Closed-loop System)

Name		IMACS (image in the closed-loop system)
Tool/Technology in a Nutshell		A framework to design, analyse, validate and generate code for systems where image-processing or other data-intensive processing is in a closed-loop. It allows for simulation of physics of various dynamic systems including camera and other sensors, Matlab front-end for designing feedback/supervisory control and processing, code generation support for multi-core platforms, and (efficient) implementation on platforms like CompSOC, MPSoC and NVIDIA AGX Xavier.
Key Features – FitOptiVis Starting Point		The basic infrastructure is developed under the Marie Curie European project oCPS and ECSEL project I-MECH. It has been further developed in FitOptiVis with specific focus on the FitOptiVis objective.
Intended Users		Embedded and cyber-physical systems developers
Benefits for the User		• Applications can be developed, tested, validated and debugged in hardware-in-the-loop and software-in-the-loop settings; • Performance evaluation and prediction of image-based systems; • Automatic code generation (for CompSOC).
Tool Requirements	**Inputs**	• Details of the image-in-the-loop applications; e.g., system model, scenarios of interest and so on; • Camera and other sensor specifications; • Platform details; e.g, processors, memory; • Performance and quality requirements.
	Outputs	• Controller design satisfying quality and performance requirements; • Generated code.
	Target	CompSOC and NVIDIA Xavier
	Dependencies	Matlab and Simulink with embedded coder, physics simulation engine (e.g., V-REP, Webots, LGSVL), OpenCV
Tool/Technology Block Diagram		

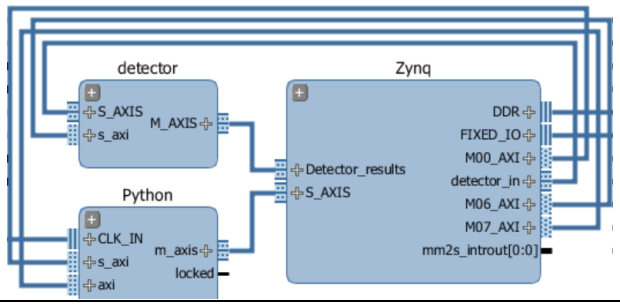
FitOptiVis Technological Advances		Implementation of FitOptiVis resource management architecture developed under WP4. Moreover, it will cover the quality management aspects where design-time optimization techniques will used/validated along with runtime reconfiguration/decisions.
Use within FitOptiVis Demonstrators	**Use**	The results of design-time optimization in WP3 will be partially implemented;
	Links	Reconfiguration solution of WP4 will also be a part of it.
Open-Source		Yes
Licence Type		Apache2.0
Commercial license		N/A

10.10. HDR Processing accelerator (HDR Core)

Name		HDR processing IP Cores
Tool/Technology in a Nutshell		FPGA IP core for real-time ghost-free HDR image acquisition and tone-mapping. It merges three standard images into a single HDR frame. The Tone-mapping core implements Durand operator.
Key Features – FitOptiVis Starting Point		The IP core did not contain the ghost-free algorithm. Tone-mapping was just a reference implementation in C++ with floating point arithmetic.
Intended Users		Embedded system developers
Benefits for the User		Acquires ghost-free HDR images in real-time. The image from sensor is much clearer with no motion artefacts.
Tool/Technology Requirements	**Inputs**	Sequence of three images with known exposure time
	Outputs	HDR image with fixed point pixel format 16.12 (12 fractional bits) / RGB tone-mapped image (3x 8b)
	Target	7-series Xilinx FPGA
	Dependencies	Vivado HLS 2016.4 and older
Tool/Technology Block Diagram(s)	**Vivado Block Diagram**	
		
Example: HDR merge core		Example of HDR image without(left) and with (right) ghost-free merging applied. 
FitOptiVis Technological Advances		We implemented the ghost-free merging and tone-mapping algorithms in FPGA. Tone-mapping algorithm was redesigned for fixed point arithmetic. The implantation run faster than real-time, achieves up to 96 FPS on FullHD

		images. Therefore, it is feasible for traffic and industrial applications.
Use within FitOptiVis Demonstrators	**Use**	FitOptiVis Traffic surveillance use case (image acquisition and pre-processing)
	Links	This algorithm is well suitable for various platforms, including CPU and GPU based machines and/or embedded systems.
Commercial license		Yes

10.11. Object detection accelerator (ACF Core)

Name		Object detection accelerator (ACF Core)
Tool/Technology in a Nutshell		FPGA IP core detection of objects. The accelerator is purposed for low power embedded systems with low resources (like Xilinx Zynq).
Key Features – FitOptiVis Starting Point		We started with old technologies developed in previous projects. The detector core was based on LBP features (high memory and logic footprint) and did not support any configuration parameters.
Intended Users		Embedded system developers
Benefits for the User		Easy-to-use IP core for robust detection of specific objects in real-time. The type of object is defined by model (binary data for the IP core) which can be trained with waldboost algorithm. One of biggest benefits is that the algorithm for detection of objects does not need to be re-developed by the designer, only new model must be created. This accelerator is suitable for rigid objects in controlled conditions (which is common in industrial usage), for example, license plates (which is demonstrated in FitOptiVis), faces, markers, etc.
Tool/Technology Requirements	**Inputs**	Image data on AXI Video bus
	Outputs	Detection results are written to RAM using DMA. The results contain locations of detected objects, scores and identifier of the source image (for synchronization purpose). On CPU in the Linux OS, the results are available on the block device associated with the DMA.
	Target	7-series Xilinx FPGA
	Dependencies	Xilinx Vivado
Tool/Technology Block Diagram(s)	**Vivado Block Diagram**	This block diagram shows the basic usage of the accelerator. 
FitOptiVis Technological Advances		In FitOptiVis , we completely re-designed the detection engine. We dropped LBP feature-based algorithm and changed it to decision tree-based algorithm. We developed new static memory access scheduling scheme for classifier evaluation. As a result, the classifier models used

		by the accelerator can be evaluated on many image locations in parallel and the model size is very small (thousands of parameters, compared to hundreds of thousands when LBP features were used). The new detection IP core fits even small FPGAs and can process Full HD image at up to 60 frames per second.
Use within FitOptiVis Demonstrators	**Use**	FitOptiVis Traffic surveillance use case – detection of license plates (as a part of License plate detection component)
	Links	This algorithm is well suitable for various platforms, including CPU and GPU based machines and/or embedded systems.
Commercial license		Yes