

ECSEL 783162

FitOptiVis

**From the cloud to the edge - smart IntegraTion and
OPTimisation Technologies for highly efficient Image and Video
processing Systems**

**Deliverable: D3.2 Design time, optimisation,
deployment and programming strategies V2**

Due date of deliverable: (31-05-2020)

Actual submission date: (31-05-2020)

Start date of Project: 01 June 2018

Duration: 36 months

Responsible: Tampere University (of Technology) (TUT)

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	jjuranek@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	20.2.2020	Starting version based on D3.1 – Jiri Kadlec, UTIA
V0.2	31.3.2020	Partner contributions
V0.3	19.5.2020	Integrated, ready for review by participating partners
V0.4	22.5.2020	Ready for internal review by Pekka Jääskeläinen
V0.9	25.5.2020	Edits by PJ before internal review.
V0.95	29.5.2020	Send for internal review by Marcos Martinez de Alejandro and Zaid Al-Ars
V0.96	3.6.2020	Update after review
V0.99	4.6.2020	After internal review, all internal comments addressed.
V1.0	5.6.2020	Final version.

Document data

Keywords	
Editor Address data	Name: Pekka Jääskeläinen Partner: TUT – Tampere University (of Technology) Address: Phone:

Distribution list

Date	Issue	E-mailer
5.6.2020	V1.0	Final version.

Table of Contents

1. DOCUMENT INFO	2
2. EXECUTIVE SUMMARY	10
3. INTRODUCTION	11
4. MODEL-DRIVEN ENGINEERING TECHNIQUES FOR ENERGY, PERFORMANCE AND OTHER QUALITIES	12
4.1. The S3D modelling methodology for real-time video processing systems	13
4.2. Design space exploration for reconfigurability	16
4.3. The SAGE verification suite	19
4.4. Dynamic performance tracking of image-based applications using control theoretic approaches	20
4.4.1. MODELLING APPLICATIONS AS A MAX-PLUS LINEAR SYSTEM	20
4.4.2. CONTROL-THEORETIC APPROACH	22
4.4.3. MODELLING DYNAMISM AS A SWITCHED MAX-PLUS LINEAR SYSTEM	22
4.5 Scenario- and platform-aware design flow for image-based control systems	23
4.5.1 Scenario- and platform-aware design (SPADe)	24
4.5.1.1 SPADe inputs	25
4.5.1.2 Formal modelling: application and platform models	25
4.5.1.3 Analysis and design	25
4.5.1.4 Implementation and runtime reconfiguration mechanism	27
4.5.2 Evaluation: IMACS framework	28
4.5.2.1 Case study	28
4.5.2.2 Results and comparison	29
4.6 Modelling of real-time video processing systems with limited precision	29
4.7 Design time support for high level tool chains	30
4.8 High-level abstract component model and DSL	30
4.9 Runtime reconfiguration Implementation of Embedded systems	31
5. PROGRAMMING AND PARALLELIZATION SUPPORT	33



5.1.	Static resource allocation and runtime scheduling	33
5.2.	Training WaldBoost detectors for FPGA	34
5.3.	OpenMP for real-time video systems	37
5.4.	Design time support for C/C++ compilers and OpenCV algorithmic libraries	37
5.5.	TTA-based Co-Design Environment (TCE).....	38
5.5.1.	SUPPORT FOR 64-BIT POINTERS AND INTEGERS	38
5.5.2.	LOOP OPTIMIZATION SUPPORT	38
5.5.3.	LOOP BUFFER AND INSTRUCTION REGISTER FILE SUPPORT	41
5.6.	BlockCopier: A programmable block transfer unit.....	44
5.7.	Deterministic timing in distributed systems and latency control with Time Sensitive Networks (TSN)	45
5.8.	Code generation for reconfigurable systems	49
6.	ACCELERATION SUPPORT	51
6.1.	OpenMP for HW accelerators	51
6.2.	HW accelerators generated by the Xilinx SG for DSP and SDSoC system level compiler.....	52
6.3.	The Multi-Dataflow Composer (MDC) tool: a dataflow-to-accelerator design suite	53
6.4.	NEURAghe a flexible and parameterized CNN accelerator	55
6.5.	TTA-Based customized soft core accelerators	57
6.5.1.	TCE: AUTOEXPLORER (AEX)	60
6.6.	Acceleration of individual algorithms in combination of FPGA and CPU	63
6.6.1.	OBJECT DETECTION ON FPGA USING WALDBOOST ALGORITHM	63
6.6.2.	HDR IMAGE ACQUISITION.....	66
6.7.	HDR merging and de-ghosting	66
6.8.	Tonemapping.....	68
6.9.	Convolutional HW accelerator	70
6.10.	Video-based Point Cloud Compression	71
6.11.	Acceleration of face detector on GPU and DSP	73
7.	DESIGN TIME SUPPORT FOR METHODOLOGIES AND TOOLS	75
7.1.	Y2 extension of support for the ZynqBerry board TE0726	76
7.1.1.	Y2 SUPPORT FOR DESIGNS WITH XILINX SG FOR DSP DATA STREAMING IPS FOR ZYNQBERRY MODULE	76
7.2.	Y2 extension of support for Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O	78
7.2.1.	Y2 SUPPORT FOR DESIGNS WITH XILINX SG FOR DSP DATA STREAMING IPS FOR ZYNQ ULTRASCALE+ WITH VIDEO I/O	79



7.2.2.	Y2 ACCELERATION RESULTS ON ZYNQ ULTRASCALE+ WITH VIDEO I/O.	83
7.2.3.	Y2 ACCELERATION RESULTS ON ZYNQBERRY WITH VIDEO I/O.	83
7.3.	Complete runtime reconfiguration of PL part of Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O	84
8.	CONCLUSIONS	86
9.	REFERENCES	87
10.	APPENDIX: FITOPTIVIS DESIGN TIME SUPPORT TOOLS	90
10.1.	TTA-Based Co-design Environment (TCE).....	92
10.2.	HW/SW CO-DEsign of HEterogeneous Parallel dedicated SYstems (HEPSYCODE).....	95
10.3.	Multi-Dataflow Composer (MDC) tool	101
10.4.	The SAGE Verification Suite (SAGE-VS)	103
10.5.	RIE – Re-configurable Implementation of Embedded systems	106
10.6.	S3D – Single Source Design Framework.....	108
10.7.	Design Time Resource Configurator (DTRC) Technology	109
10.8.	Design Time Resource Integrator of Model Composer IPs (DTRiMC) Technology.	112
10.9.	IMACS (IMAge in the Closed-loop System).....	118

Table of Figures

Figure 1: TE Single Source System Design Framework (SD3).....	14
Figure 2: Application view.....	15
Figure 3: Example of Component interfaces.....	15
Figure 4: S3D design flow.	16
Figure 5: HEPSYCODE ESL HW/SW co-design flow.	17
Figure 6: HEPSYCODE ESL HW/SW co-design flow.	18
Figure 7: Embedded multi-core platform with reconfiguration mechanism.	20
Figure 8: Synthetic SDF model.....	21
Figure 9: Inter-arrival time.	23
Figure 10: SPADe design flow.....	24
Figure 11: Multiprocessor System on Chip with two processor tiles and one memory tile.....	25
Figure 12: NVIDIA Drive PX2 platform graph structure.....	26
Figure 13: System mapping to MPSoC.....	27
Figure 14: IMACS evaluation framework. (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; c1, c2, c3 are the cameras.....	28
Figure 15: Comparison between SPADe and pipelined (state-of-the-art) approaches; bc=best-case timing and wc=worst-case timing; SPADe is executed with a number of scenario sequences; yL is the lateral deviation of the LKAS system under study.....	29
Figure 16: JSON produced by IP core and Chrome Tracing with loaded data for analysis.	34
Figure 17: Example of feature maps extracted from image - image scales in rows, channels in columns.	35
Figure 18: Example of detected license plates.	36
Figure 19: OpenMP-based reconfiguration methodology.....	37
Figure 20: The two different loop optimization modes. Red labels indicate re-used code from LLVM and blue labels indicate separate code managed by TCE code generator.....	39
Figure 21: High-level example of software pipelining.....	40
Figure 22: Architecture of the block copier ASIP.	45
Figure 23: VLAN frame format.....	46
Figure 24: Architecture of the Time-Aware traffic Shaper.	47
Figure 25: IEEE 802.1Qbv traffic scheduling and shaping.	47
Figure 26: Traffic scheduling and shaping along the TSN stream path.....	48
Figure 27: Automatic code generation.	50
Figure 28: UML/MARTE model generated from QRML description.	50
Figure 29: HW accelerators with OpenMP code.	51
Figure 30: Maximum clock frequency of the synthesized processors.	58
Figure 31: Simplified view of the wide-SIMD TTA template.	59
Figure 32: Simplified TCE Exploration process of AutoExplorer.	60
Figure 33: Un-optimized architecture (left), final best possible architecture (right).	61
Figure 34: AEx2 result pruning between passes. The configurations marked in red can reach the targeted execution time with the targeted clock frequency...	62
Figure 35: Overview of the detection architecture - (top row) Image scale space generation, (middle row) feature channel calculation, (bottom row) classifier evaluation on each location.....	64

Figure 36: Evaluation of classifier - a decision tree loads features from feature maps and produces response which is aggregated in the total response. It either continues with the next tree or exits with a negative decision.....	64
Figure 37: Overall schematics of HDR acquisition pipeline. The individual blocks create a dataflow pipeline, which is configured through AXI Lite interface from ARM CPU.....	66
Figure 38: Input image sequence (top) and two certainty maps (similar to ghostmaps in related algorithms) used during ghost-free HDR merging.....	67
Figure 39: Vivado Block design of ghostfree HDR IP core.....	67
Figure 40: Resulting HDR tonemapped images with ghost-free merging disabled (left) and enabled (right).	68
Figure 41: Tonemapping IP core in Vivado.....	69
Figure 42: The FPGA implementation of Durand tonemapping.	70
Figure 43: V-PCC encoding structure.....	71
Figure 44: V-PCC decoding structure.....	72
Figure 45: Power consumption measurement (mA) 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.	74
Figure 46: Power consumption measurement (mA) 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.	75
Figure 47: WP3 Y2 Initial Zynq Ultrascale+ platform with two serial connected accelerators.....	79
Figure 48: WP3 Y2 Initial Zynq Ultrascale+ platform with two serial connected accelerators.....	80
Figure 49: Full HD edge-detection and matrix multiplication on two FP03x8 accelerators.....	83
Figure 50: Function of all HW data movers have to be terminated before stop of SW app.....	85
Figure 51: Demonstration of platform supporting the PL reconfiguration in Prague on 5.3.2020.	86
Figure 52: FitOptiVis Design Support Tools.....	90

Table of Tables

Table 1: Overview and Comparison of Model-driven engineering techniques.....	13
Table 2: VLAN identification rules of user traffic types.....	46
Table 3: VLAN identification rules of user traffic types.....	49
Table 4: FPGA resource consumption for XC7Z020.....	65
Table 5: FPGA resource consumption after HLS synthesis for XC7Z020.	67
Table 6: Resource consumption.....	70
Table 7: Progress made in FitOptiVis in WP3 in Y1.....	76
Table 8: HW resources used by the FP01x8 Accelerator with different data movers. .	78
Table 9: HW resources used by the FP01x8 Accelerators with different data movers.....	82
Table 10: Use of WP3 tools and technologies by project partners (M24).....	91

Table of Acronyms

UML-MARTE	UML Profile for Modelling and Analysis of Real-Time and Embedded Systems
SDSoC	Software Defined System in Chip (System level HW design flow).
HLS	High Level Synthesis (Xilinx C/C++ to HW IP compiler).
ESL	Electronic System Level
RIE	Runtime reconfiguration Implementation of Embedded systems
QRML	Quality and Resource Management Language
SAGE-VS	SAGE Verification Suite is set of SW tools aimed to accomplish different formal verification tasks at design time.
DSL	Domain Specific Language
S3D	Single-Source System Design Framework
MDA	Model Driven Architecture
PIM, PDM, PSM	Platform Independent/Description/Specific Model
NFR	Non-Functional Requirements
J4CS	Joule for C statements
DSE	Design Space Exploration
D-HMPS	Dedicate Heterogeneous/homogeneous Multi-Processing System
PSPs	Property Specification Patterns
SDF	Synchronous Data Flow
MPL	Max Plus Linear graph (Serves for analysis of DSF graph)
IBC	Image Based Control
WCET	Worst Case Execution Time
SPADe	Scenario and Platform Aware Design
HiL	hardware-in-the-loop
CompSOC	predictable multiprocessor system-on-chip platform
QoC	Quality-of-Control
WC	Worst case
FPS	Frames per second
AXI4	Xilinx Interconnect standard
AXI4-Stream	Xilinx Interconnect standard serving for data streaming
AXI4-Lite	Xilinx Interconnect standard serving for access to registers
DMA	Direct Memory Access
SG-DMA	Scatter-Gather Direct Memory Access

OpenMP	Open Multi-Processing
SMP	Symmetric Multi-Processing
OpenCV	Open Computer Vision (C++ library of algorithms)
TTA	Transport Triggered Architecture
TCE	TTA-based Co-Design Environment
SIMD	Single Instruction Multiple Data
LLVM	Low-Level Virtual Machine compiler development infrastructure
VLIW	Very large Instruction Word
IRF	Instruction Register File
FPGA	Field programmable Gate Array
SoC	System on Chip
OpenCL	Open standard defined by khronos group supported by multiple vendors
TSN	Time Sensitive Networks
gPTP	generalized Precision Time Protocol
VID	VLAN ID
PCP	Priority Code Point
BMCA	Best Master Clock Algorithm
DTRiMC	Design Time Resource Integrator of Model Composer IPs technology
MDC	Multi-Dataflow Composer tool: a dataflow-to-accelerator design suite
CNN	Convolutional Neural Networks
NEURAghe	Flexible and parameterized CNN accelerator
NeuDNN	NEURAghe Deep Neural Network software stack
CSP	Convolution-Specific Processor
LSU	Load-Store Unit
AEx	AutoExplorer is a design space exploration flow
TMO	Tonemapping
YOLO	Open source content analysis software
V-PCC	Video-based Point Cloud Compression
CfP	Call for Proposals

2. Executive summary

Deliverable D3.2 (Design time, optimization, deployment and programming strategies V2), focuses on concrete design flows, tools and design time support packages used, developed and/or extended in the FitOptiVis project until end of project year 2. D3.2 is an update of D3.1, adding contributions delivered by M24 related to 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.

Y2 results which span over all three tasks (T3.1, T3.2 and T3.3) are described in Chapter 7. These design time technologies are released in Y2 of the project in the form of publicly accessible evaluation packages and documented in publicly accessible application notes [7.12] and [7.13]. These released resources serve as concrete WP3 design-time resources for FitOptiVis project partners, and also serve as support material which can be used by other developers outside of the project.

A major addition in D3.2 in comparison to D3.1 is Chapter 10 which summarizes all the developed tools and design technologies, highlighting their differences, granularities and use scenarios.

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 tasks T3.1, T3.2 and T3.3 after the second year of the project (M24), updating D3.1. The work of WP3 is organised in 3 tasks: T3.1, T3.2 and T3.3 of which contributions are provided in this deliverable as follows.

Task 3.1 – In this document, we describe common approaches to the design time resources covering model-driven engineering techniques for energy, performance, and other qualities. These activities are mainly in **Chapter 4**. One of the core developments in this Chapter 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 or are being 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 a link to WP5 (Devices) via its new hardware accelerator designs developed using the WP3 design 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 before M24 of the project in the form of publicly accessible well-documented evaluation packages [7.12], [7.13]. These released resources serve as concrete WP3 output for FitOptiVis project partners. The tools released in open source also serve other developers outside of the project.

Chapter 10 of this D3.2 deliverable which contains a summary describing all the tools, mapping them to a big picture related to their granularity and the software/hardware orientation.

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).

Project team included in D3.2 set of elaborated model-driven engineering techniques 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 exists 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. 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	4.4	Depending on the application requirements, the optimization algorithms find the best configuration	It is based on Synchronous Dataflow (SDF) graph which can be analysed to answer performance related questions

theoretic approaches)		(mapping, scheduling and voltage/frequency setting).	such as the minimum 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.

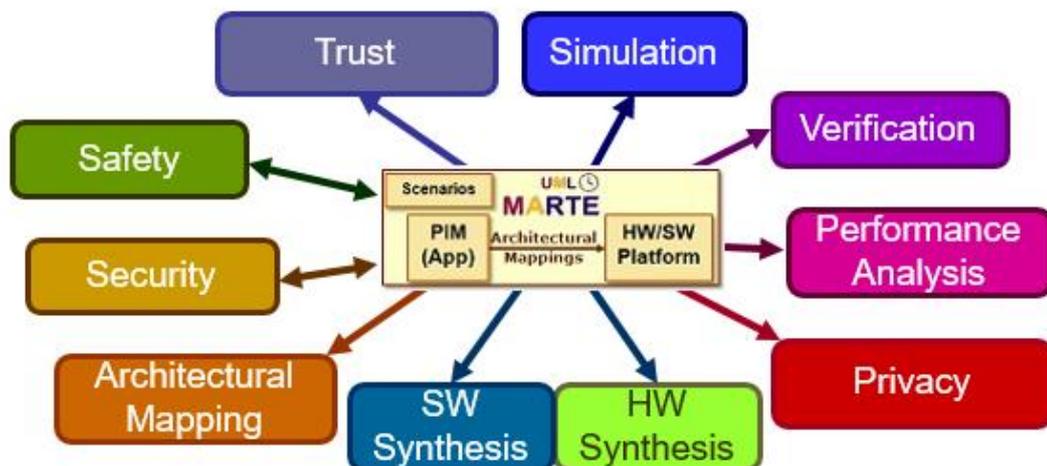


Figure 1: TE Single Source System Design Framework (SD3).

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 they relation. An example of Application View is presented in Figure 2

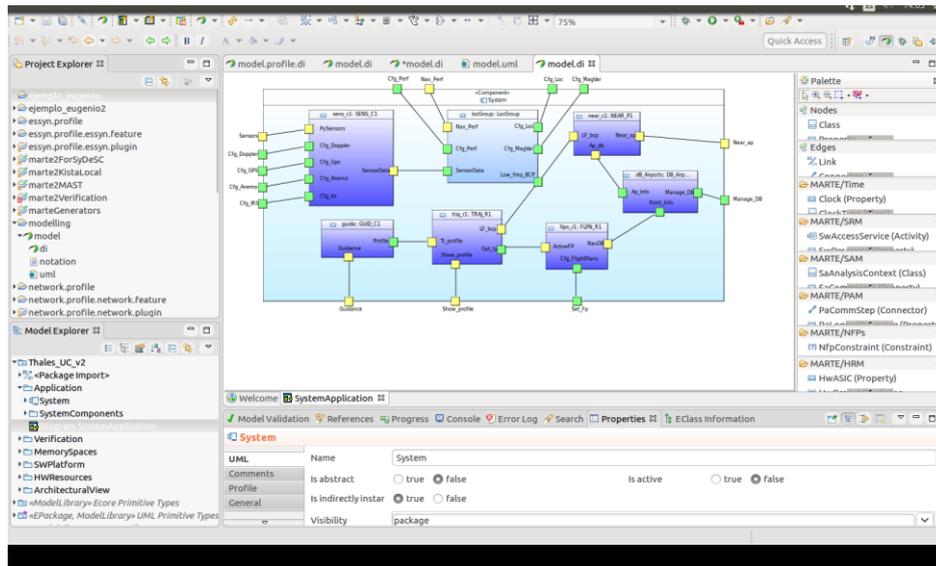


Figure 2: Application view

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.

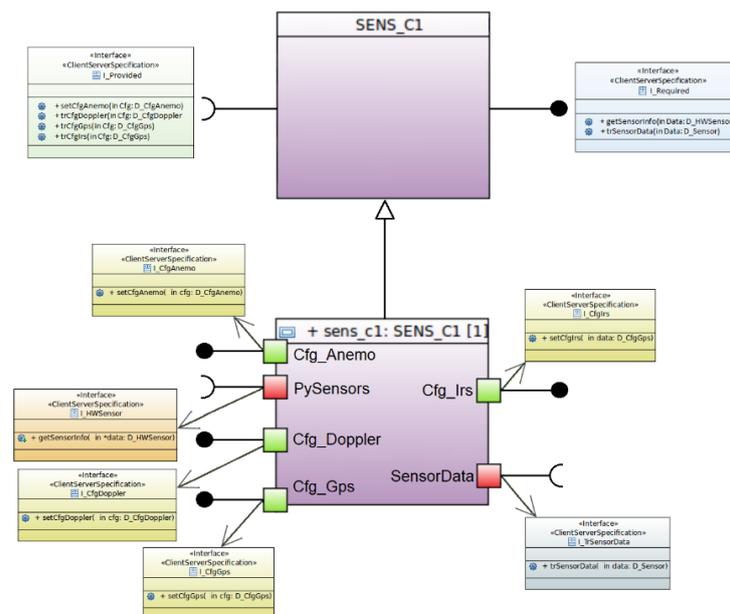


Figure 3: Example of Component interfaces.

During the last year, the framework has been extended with a tool that automatically generate UML/MARTE models from QRML descriptions. For this purpose, QRML has

been extended to support the service-oriented architectures that are typically modelled in UML/MARTE. The QRML extension supports the definition of provided/required interfaces as well as the definition of new datatypes for service parameters.

The generated code is imported by the S3D framework and this integrates the WP2 QRML language in an UML/MARTE design flow. Next figure presents the S3D design flow.

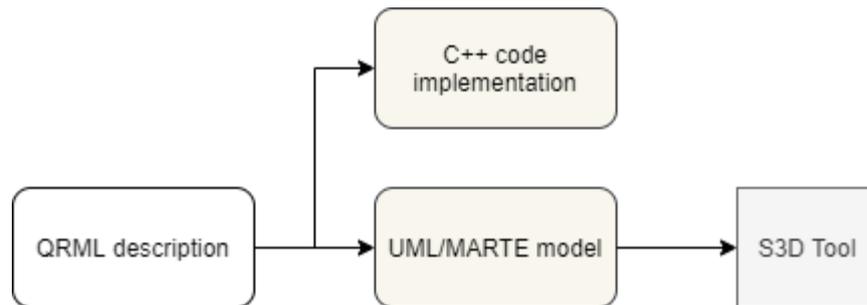


Figure 4: S3D design flow.

4.2. Design space exploration for reconfigurability

In Y2, UNIVAQ has extended 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 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 5 shows the reference HEPSYCODE ESL HW/SW co-design flow more in details.

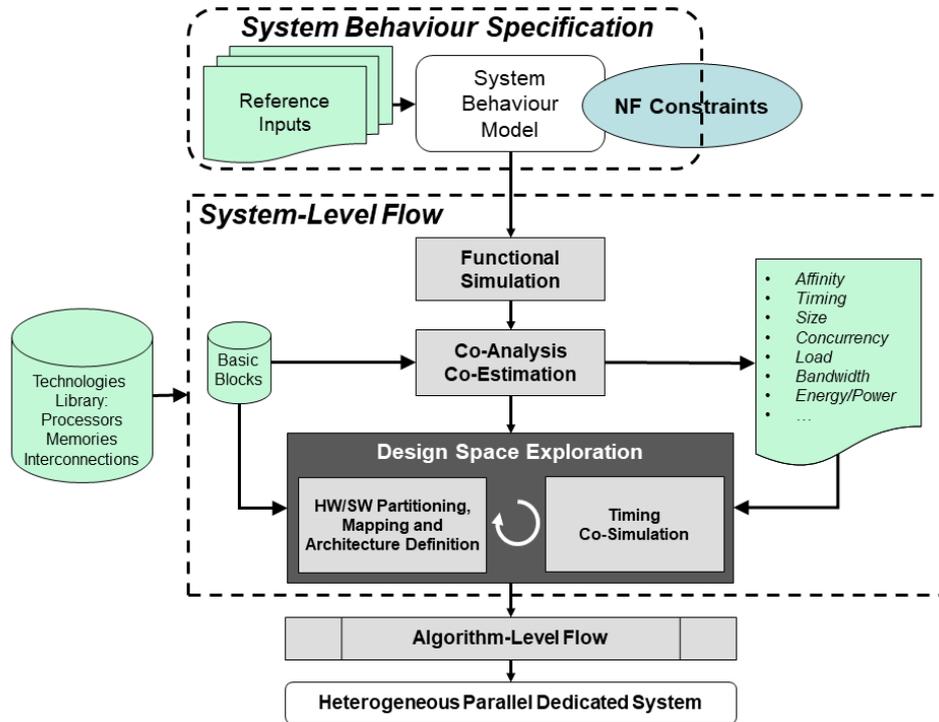


Figure 5: HEPHYCODE ESL HW/SW co-design flow.

The HEPHYCODE goal is to identify (at design-time and, in future, also at run-time) suitable “reconfiguration plans” for different trade-offs (e.g., timing vs energy/power vs accuracy) by considering a heterogeneous set of HW components with multiple working points. Respect to the plan exposed in D3.1, UNIVAQ has improved an existing system-level metric for timing performance [4.3] and, based on this one, defined an innovative system-level metric for energy consumption.

The result of such an activity has been the definition of the following statement-level energy consumption metric.

Definition: J4CS (Joule for C statements)

Let $Z = \{z_i \mid i = 1, 2, \dots, n\}$ a benchmark set of n reference leaf C functions (i.e., no other internal nested function calls), $B = \{b_{i,k} \mid i = 1, 2, \dots, n \wedge k = 1, 2, \dots, t\}$ a set of t randomly generated inputs for each function z_i , and $P = \{p_j \mid j = 1, 2, \dots, m\}$ a set of m processing units (i.e., processors that are able to execute the considered software functions). The J4CS metric is the ratio between the number of assembly instructions executed by the target processor p_j running the functions z_i , and the number of executed C statements multiplied by the average energy of an assembly instruction execution $\bar{E}'(p_j)$, i.e.:

$$J4CS(p_j) = \left\{ \bar{E}'(p_j) \cdot \bar{I}(p_j, z_i, b_{i,k}) = \bar{E}'(p_j) \cdot \frac{I(p_j, z_i, b_{i,k})}{CS(z_i, b_{i,k})}, \forall z_i \in Z, b_{i,k} \in B \right\}$$

Where $I(p_j, z_i, b_{i,k})$ is the number of assembly instructions executed by the target processor p_j to execute the function z_i with inputs $b_{i,k}$, $\bar{I}(p_j, z_i, b_{i,k})$ is the mean number of assembly instructions executed for each generic statement C belonging on software function z_i on the processor p_j with inputs $b_{i,k}$, and $CS(z_i, b_{i,k})$ is the number

of executed C statements for the function z_i with inputs $b_{i,k}$ evaluated with static host profiling. The J4CS evaluation approach is shown in Figure 6.

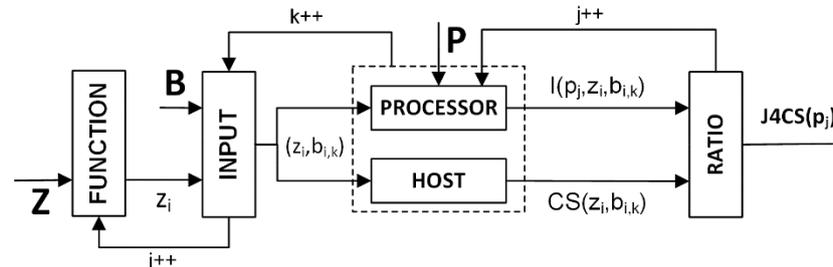


Figure 6: HEPHYCODE ESL HW/SW co-design flow.

UNIVAQ has exploited a statistical characterization approach so providing the J4CS metric useful to estimate, at an early-stage design phase, the energy consumption related to the execution of SW on a target embedded processor, so characterizing the processor itself. J4CS is suitable for very fast estimation, comparison and selection activities. J4CS evaluation considers an assembly-level analysis. UNIVAQ exploited also a framework to evaluate each parameter that contribute to the metric evaluation. The obtained value can be assigned to each statement of a given C function and exploited, with a host-based source-level profiling, to estimate the total amount of energy consumed when the C function with specific inputs is executed on the target embedded processor, as presented in the next point. Plans for Y3 mainly focus on the extension of the metric to consider also HW implementations: To extend an existing DSE approach [4.4] to consider also “energy consumption” as starting non-functional requirements. From the energy point of view, the goal is to suggest to the designer architecture/mapping solutions able to consume less than a given amount of energy while still satisfy timing requirements.

UNIVAQ has extended the HEPHYCODE methodology to consider energy requirement specifications, metrics (J4CS), and objective cost functions in the design space exploration approach. UNIVAQ is applying it to some reference examples, also w.r.t. image and video processing applications (e.g., Sobel or equivalent image/video filtering applications). User energy requirements can be related to the possibility to find system implementations based on a dedicate heterogeneous/homogeneous multi-processing system (D-HMPS) that consumes as less energy as possible, or a D-HMPS 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 analyses different possible solutions by means of an evolutionary algorithm that evaluates, at the same time, with a weighted sum method, several objectives (i.e., cost) functions. Considering different processor technologies, HEPHYCODE is able to find a HW/SW partitioning, to define a HW architecture and to suggest a mapping able to satisfy the input energy requirements. Finally, HEPHYCODE is able to the estimate the energy consumption by means of a SystemC simulator that consider the results found by the evolutionary algorithm. Examples and results will be presented in the next deliverable, while the whole approach will be considered to be applied to one or more of the FitOptiVis use cases.

Plans for Y3 mainly focus on the presentation of the results obtained by applying the extended (i.e., energy-aware) HEPHYCODE, and the analysis of a further extension to

consider also a set of alternative algorithm implementations providing results with different accuracy (i.e., by means of approximate computing techniques). The goals will be:

- To provide a classification of the existing approximated computing approaches in order to define a system-level metric to evaluate the “approximation degree” (i.e., the accuracy) of alternative implementations.
- To exploit such metric to drive the DSE to suggest to the designer architecture/mapping solutions able to provide results with reduced accuracy in order to help satisfying timing and energy/power constraints

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.

Concerning the usage within FitOptiVis, the objective at M12 is to provide for each tool some extensions according to the use-cases' needs. In particular:

- 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.
- HyDRA: define a more usable input language and improve the performance of the planner in terms of execution time.
- ATG: release the first stable version.

4.4. Dynamic performance tracking of image-based applications using control theoretic approaches

TUE is working on design-time optimization algorithms for imaging and video applications with dynamically changing performance. The basic application setting is shown in Figure 7. The image/video streams are processed on multi-core platforms with some reconfiguration mechanism. The application uses the processed information to meet some high-level requirements such as throughput and latency of the information obtained from the image content. Depending on the application requirements, the optimization algorithms finds the best configuration (e.g., mapping, scheduling and voltage/frequency setting). The application requirement may change at runtime (e.g. a user increasing the frame size of a requested video stream) and depending on a changed requirement an existing configuration may be suboptimal or the requirements might not be met. The developed algorithm finds the best configuration for the dynamically changing requirements. Thus, the system tracks the dynamically changing performance requirements by changing system reconfigurations.

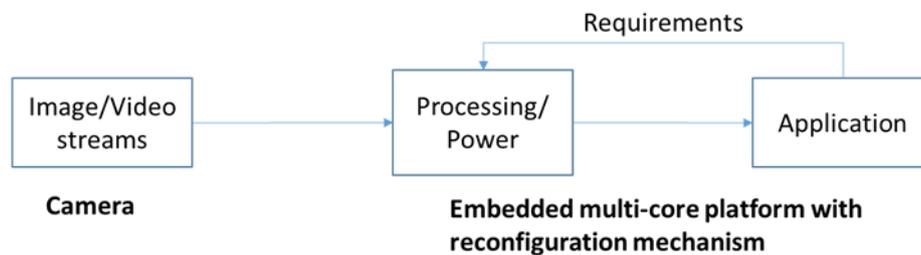


Figure 7: Embedded multi-core platform with reconfiguration mechanism.

4.4.1. Modelling applications as a max-plus linear system

The performance aspects of a static image/video processing application can be modelled as a Synchronous Dataflow (SDF) graph which can be analysed to answer performance related questions such as the minimum guaranteed throughput for a given mapping to a platform [4.14]. Temporal behaviour of an SDF graph can be modelled as a max-plus linear (MPL) system which provides a means to efficiently analyse SDF graphs. A MPL system has the following form:

$$x(k + 1) = A \otimes x(k) \oplus B \otimes u(k)$$

$$y(k) = C \otimes x(k) \oplus Du(k)$$

where $x(k)$ captures the state-vector containing time-stamps of k^{th} instance all the events of interest such as production of processed output, generation of intermediate data and so on; $y(k)$ captures the time-stamps of generation of output such as the k^{th} processed image; $u(k)$ specifies the time-stamps of when to apply the input signal (e.g., changing voltage and frequency of a processor) to the system. The A , B , C and D matrices capture the interplay between states, inputs and outputs and depend on the specific application being modelled. The basic idea is to design $u(k)$ such that $y(k)$ meets the application requirements. Since $u(k)$ can be adapted dynamically the performance requirement can be tracked by regulating $y(k)$.

For illustration, let us consider a simplified image-assisted surgery scenario where images are captured by a camera at a constant rate and they are processed on a 2-core system-on-chip. The processed images are displayed at a device which is used by a doctor to perform surgery. A synthetic SDF model of such an application is shown below (Figure 8) where SRC is the model of the camera that produces images at a constant rate and P, Q are tasks (modelled as actors) to process the image and produce output images at time instances $y(k)$ at the display. The processing of the images can be certain upscaling or downscaling depending on the required display resolution (decided by the doctor). The P and Q actors are mapped to two different processors of which the frequency can be controlled independently. $u_1(k)$ and $u_2(k)$ are the time instances when the frequency of the processors is scaled up or down. The system configurations thus consist of the frequency settings of the two processors. The $x_i(k)$ are the production time instances of the intermediate signals.

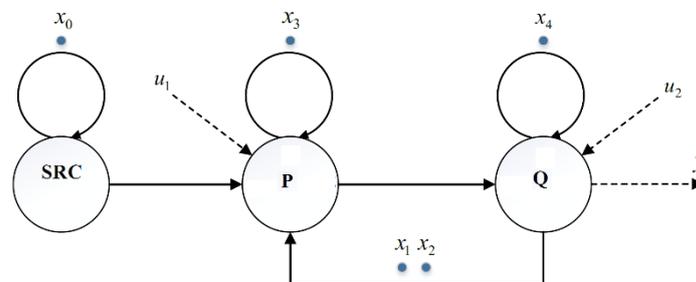


Figure 8: Synthetic SDF model.

The overall timing of the production of output images, intermediate signals and input signals can be modelled as a MPL system shown above. The performance requirement of such a system can be modelled by the desired/reference rate of frame updates at the display device. That is, the reference inter-arrival time $\Delta_{ref}(k)$ of frames can be used to find the reference arrival time of the next frame,

$$y_{ref}(k+1) = y(k) + \Delta_{ref}(k).$$

$u_1(k)$ and $u_2(k)$ decide which processor should run at what frequency and when/if they are to be changed. When one or both processors run at a low frequency, the inter-arrival time grows and vice versa when they run on a higher frequency.

4.4.2. Control-theoretic approach

The proposed idea is to design $u(k)$ as a state-feedback controller in the following form,

$$u(k) = K_i \otimes x(k)$$

where K_i is the feedback gain we design. That is, depending on the current state of the system $x(k)$, we adapt the input to the system $u(k)$ by choosing the right K_i . For each system configuration i (i.e., for each frequency setting in the example), we design a feedback gain K_i . All the signals in this framework are time-stamps. That is, we control the time when some event should happen in order to achieve a given performance requirement. The choice of K_i can be optimized for different optimization objectives such as minimizing error with respect to $y_{ref}(k+1)$ or minimizing energy/power. Overall, the closed-loop system behaviour is given by,

$$\begin{aligned} x(k+1) &= (A \oplus B \otimes K_i)x(k) = A_{cl}^i x(k) \\ y(k) &= (C \oplus D \otimes K_i)x(k) = C_{cl}^i x(k). \end{aligned}$$

The feedback gain K_i is designed in a way that $y(k)$ meets the system requirements.

4.4.3. Modelling dynamism as a switched max-plus linear system

We consider the scenario when the application requirements change at runtime. In the FitOptiVis context, this is particularly relevant. For example, in the case of the image-assisted surgical scenario, the doctor may want different output frame sizes and/or update rates depending on the stage of the surgery. This means that the reference inter-arrival time $\Delta_{ref}^i(k)$ changes depending on the doctor's requirement leading to a different reference arrival time of the next frame,

$$y_{ref}^i(k+1) = y(k) + \Delta_{ref}^i(k).$$

Control inputs $u_1(k)$ and $u_2(k)$ in our example adapt the processor frequency according to the requirements by adapting the gains K_i . This means the closed-loop system may switch between various (A_{cl}^i, C_{cl}^i) . This is referred to as a switched max-plus linear system. We show an initial result for the above synthetic example. The actual inter-arrival time closely follows the reference one by the right design of gains.

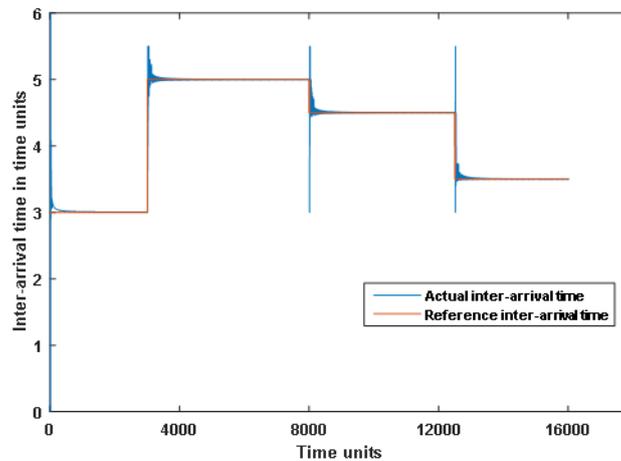


Figure 9: Inter-arrival time.

Figure 9 shows tracking performance of the proposed approach for a static application (where work-load variation is not modelled as different scenarios). The reference inter-arrival time is shown in red which changes over time. E.g., a frame is desired to be processed in every 3 time units from 0-4000 time units. From 4000 to 8000 time units, the reference inter-arrival time is 5 time units. The control adapts the configuration to track this dynamic requirement and the actual inter-arrival times are shown in blue lines. Moreover, typical imaging/video applications nowadays exhibit data-dependent workload variations (e.g. related to the image encoding or the number of features in an image). The above framework can be used to model workload variations as scenarios. The state-of-the-art approach considers static requirements (that do not change in runtime) and a dynamic requirement is usually modelled with the worst-case resulting in over-dimensioning of resources.

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

In Y2, 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 scenario- and platform-aware design flow (SPADe) for IBC systems that exploits *frequently occurring workload scenarios* to optimize performance. For industrial platforms, where tight task-level WCET bounds are difficult to obtain, *frequently occurring task execution times* are used instead of WCET estimates to obtain tight application-level temporal bounds. During controller design, performance optimization and guaranteed stability are achieved by identifying appropriate *system scenarios* and by designing a *switched controller* that switches between those scenarios. We illustrate the method considering a predictable multiprocessor system-on-chip platform - CompSOC.

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

4.5.1 Scenario- and platform-aware design (SPADe)

The SPADe flow comprises the following steps as shown in Figure 10:

- 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 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.

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.

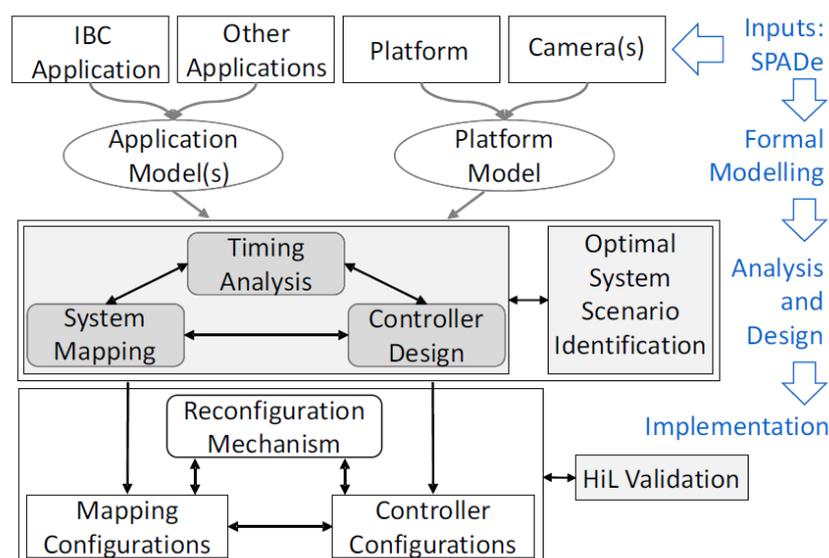


Figure 10: SPADe design flow.

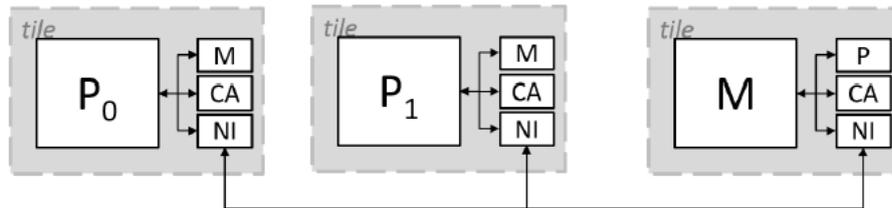


Figure 11: Multiprocessor System on Chip with two processor tiles and one memory tile.

4.5.1.1 SPADe inputs

The inputs to our design flow are details of the IBC application, other applications sharing the platform, given platform allocation for the IBC application and camera characteristics, e.g. fps. 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.

4.5.1.2 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 11 for the two platforms we considered.

4.5.1.3 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 13 illustrates three workload scenarios and their possible platform mapping. Each workload is associated with a SADFG. Figure 13 (a), (c), and (e) model the data flow graphs for different workloads and Figure 13 (b), (d) and (f) show their corresponding mappings on two or three processor tiles. Optimal mapping for a workload scenario to a platform graph generates binding-aware SDFG.

Having 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.

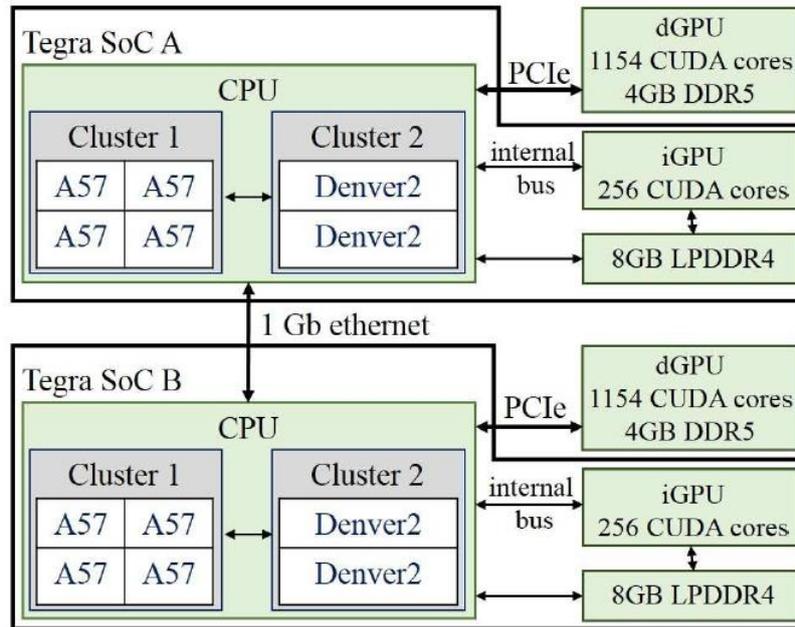


Figure 12: NVIDIA Drive PX2 platform graph structure.

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 13 (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 Rol).

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.

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

And sampling period

$$h_{s_i} = \left\lceil \frac{\tau_{s_i}}{f_h} \right\rceil f_h$$

where f_h is the camera frame arrival period.

Controller design: Once we obtain τ_{s_i} and h_{s_i} for mapped workload scenario S_i , they are then used for the discrete-time controller implementation and for designing the controller gains. See [4.14] for further details.

¹ A scenario- and platform-aware design flow for image-based control systems, SajidMohamed, DipGoswami, VishakNathan, RaghuRajappa, TwanBastena, Microprocessor and Microsystems, 2020.

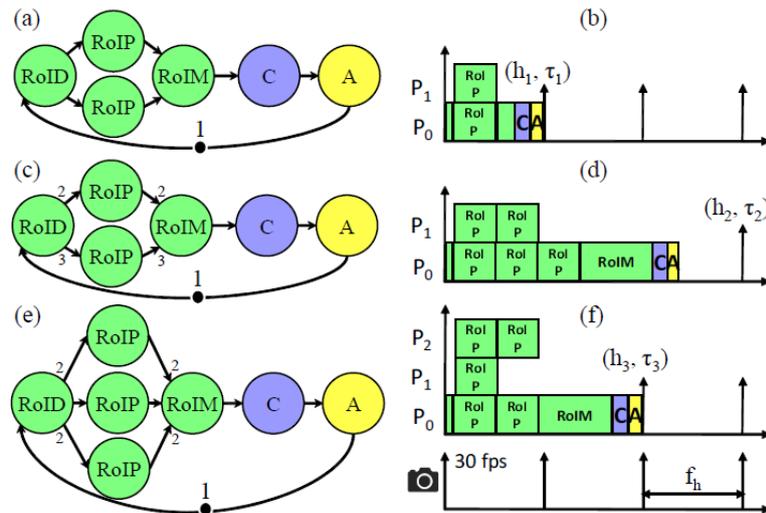


Figure 13: System mapping to MPSoC.

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 13 (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 13 (c), we would have the same timing parameters for $\#RoI = 5$ and 6.

A system scenario s_k abstracts multiple workload scenarios s_i such that for $h_k = n f_n$ for some $n > 0$, $(h_k - f_h) < h_i \leq h_k$ and $\tau_i \leq \tau_k$. Only system scenarios are then considered for defining the control configuration and for platform implementation.

4.5.1.4 Implementation and runtime reconfiguration mechanism

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. 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.

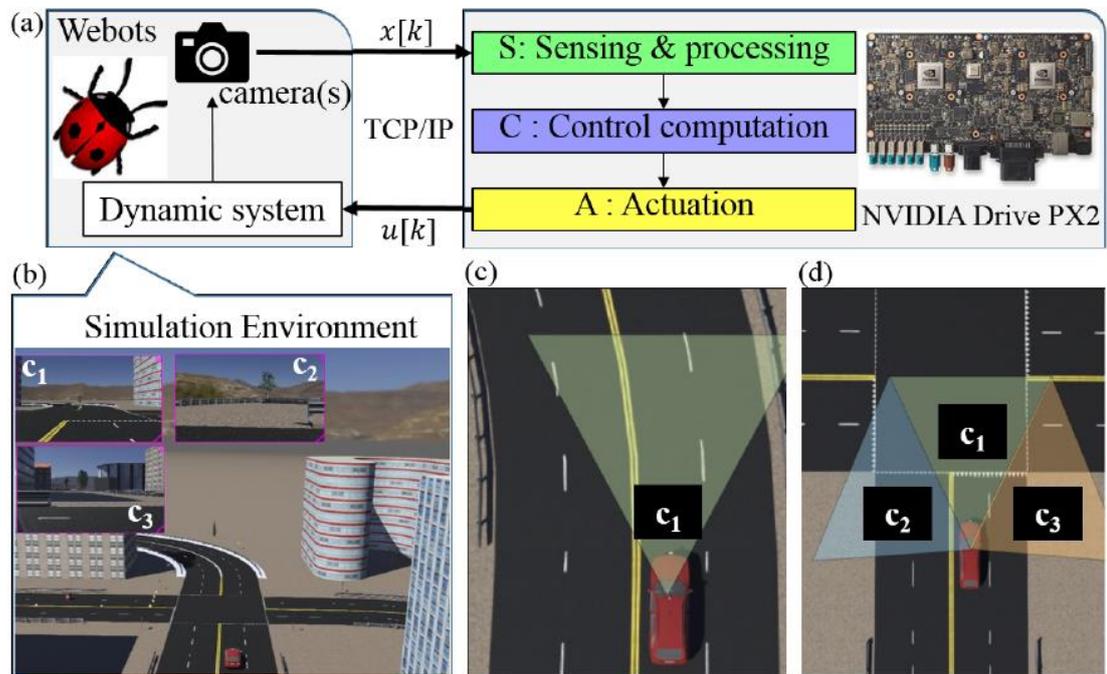


Figure 14: IMACS evaluation framework. (a) IBC system block diagram and the HiL simulator. (b) a snapshot of the HiL simulation environment in webots. (c) LKAS using single camera. (d) multi-camera LKAS; c_1, c_2, c_3 are the cameras.

4.5.2 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 debugging 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.

4.5.2.1 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 13 (c) and (d) show the two different scenarios in the LKAS system. The first scenario s_1 (see Figure 13 (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 13 (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).

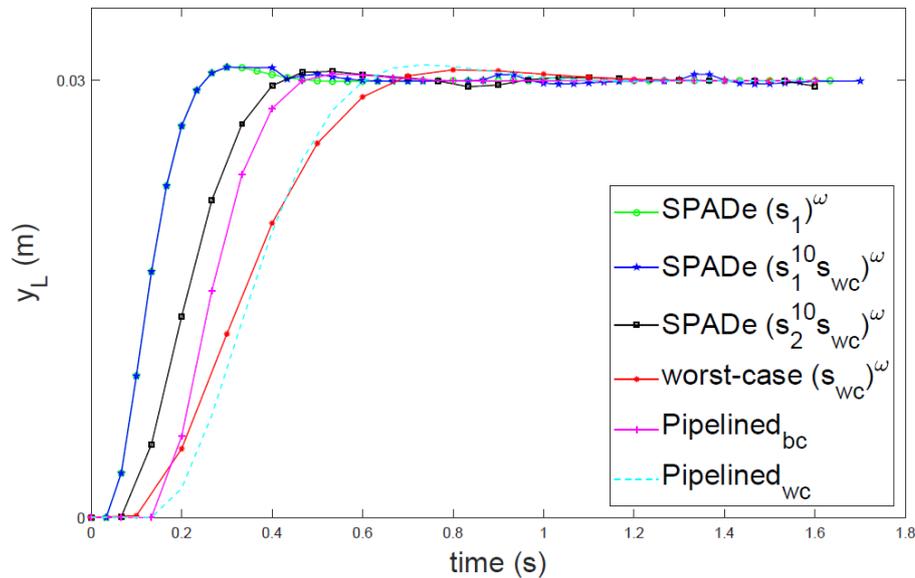


Figure 15: Comparison between SPADe and pipelined (state-of-the-art) approaches; bc=best-case timing and wc=worst-case timing; SPADe is executed with a number of scenario sequences; y_L is the lateral deviation of the LKAS system under study.

4.5.2.2 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 between the pipelined controller with respect to the SPADe approach are shown in Figure 15. The controller is supposed to bring the lateral deviation y_L to 0.03m as soon as possible. A shorter time to reach the reference, the better the Quality-of-Control (QoC). 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.

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

A limited precision approach will be applied to image / video processing pipeline. This will be modelled and analysed prior to actual implementation. Application areas will include CNN type processing and content analysis from a live video stream. The first

iterations will be simulation models, which will later be implemented in FPGA hardware and finally integrated to full custom ASIC along a RISC-V CPU core.

The key idea is to use non-linear number space. This approach allows 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 will be studied primarily in the field of object detection, and later a larger field of application domains will be looked into after that. The purpose is to find domains where the loss of precision is not a significant problem, and the benefits of the reduced precision processing outweigh the negative impacts. Also conversions between the typical binary domain and this reduced precision domain will need attention.

4.7 Design time support for high level tool chains

In Y2 UTIA developed support for high level modelling of IP blocks based on integration of the Xilinx System generator for DSP 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. The IP is integrated in Vivado 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. The automated generation of data movers is implemented in a high level tool chain based on Xilinx SDSoc 2018.2 compiler with design time support for the PetaLinux 2018.2 kernel, Debian “Stretch” operating system. Released evaluation package provide support for Ethernet data exchange based on the Arrowhead framework. See Chapters 7.1 – 7.3 of this deliverable for details.

4.8 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 are described in detail in D2.1. In this section, we connect the component model to model-driven design space optimization.

CUNI models 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. a 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 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.

4.9 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.
- 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 methodology supports on-the-edge component implementations that are implementation classes with remote interfaces. The remote interface is a particular implementation of the interface that provide support for remote procedure calls and it is independent of a particular component. This methodology facilitates the use of commonly use micro service frameworks such as Google grpc.

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 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 a next section.

5. Programming and parallelization support

This chapter describes the techniques that have been added or are being 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

BUT works on profiling of applications in design time and runtime planning:

- The main goal is to optimize execution of algorithms in multitasking/threading environment
- Sequential execution of tasks, that can but do not have to run in parallel
- The main benefit being prevention of unnecessary context (and cache) switching

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. Profiling on such platforms is to some extent covered with tools from Xilinx - AXI Performance Monitor core enables AXI system performance measurement for multiple slots (AXI4/AXI3/ AXI4-Stream/AXI4-Lite). This core captures configurable real-time performance metrics for throughput and latency for connected AXI interfaces. In addition, it logs the AXI transactions, external system events and performs real-time profiling for software applications. This is sufficient in many applications. A drawback is that AXI transactions are logged for AXI3/AXI4 interfaces, for AXI-Stream only summary statistics are gathered - data transfer, packet counts, etc., but there is no info on the individual packets (length, timestamp, processing time) which could be beneficial in analysis of complex algorithms. For this reason, we developed an IP core similar to the AXI Performance Monitor which can log AXI-Stream transactions. Input interface works in monitor mode and it can be connected to any existing AXI link.

Each log record is represented as 128 bit data. Timestamps are recorded with cycle-precision in 48 bit value. Up to 16 devices can be monitored (4 bits). Each interface can set 12 bits containing flags like image row start/end, specific value presence etc. AXI3/AXI4 can log read/write sequences. Additionally there is a 64 bit block for additional, user-defined values, like addresses, values etc. The output of the IP core is 32 bit AXI-Stream interface. Data is transferred to the CPU memory via DMA. A simple application then reads the data and stores them in JSON format.

Original profiling user interface (described in D3.1) was not sufficient for display of complex profiling information and it did not support filtering. We replaced the original tool with Chrome Tracing from Chromium project which can handle large logs, and supports zoom/pan, filtering, statistics, etc. Primary purpose of Chrome Tracing is analysis of web applications but there are also e.g. Python and C++ bindings. And it supports JSON import which we use for our purposes.

```

{
  "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 16: JSON produced by IP core and Chrome Tracing with loaded data for analysis.

Currently the tool fully supports AXI-Stream profiling. Profiling of AXI4 is currently under development. SW profiling (not developed yet) will be solved by direct writing to dedicated memory space of the profiling IP core. We still need to handle situations with a large number of records which could not sometimes fit to memory. Also DMA for data transfer takes many resources and we will, in the future, develop more simple DMA for this single purpose.

5.2. Training WaldBoost detectors for FPGA

As a support for BUT object detection architecture, we developed an open source python package for training object detectors [5.2]. The package 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. The package is still in the development stage and cannot be considered stable or suitable for production use.

WaldBoost training algorithm is a sequential algorithm which in each iteration adds one weak classifier to the sequence of already selected weak classifiers (strong classifier). And after each round, the detector is used to get new training samples from the training set. So the active training set always represents “hard” samples that need to be handled by the subsequent weak classifiers. Please refer to [5.1] for details on the algorithm.

In the waldboost package, training samples are represented as (N,H,W,C) array of N samples with spatial dimensions H,W and C channels. The samples are extracted from a training set provided by the user - sequence of images with a list of object bounding boxes.

After installation, the package is imported by:

```
import waldboost as wb
```

Then training parameters must be set up.

```
channel_opts = {
    "shrink": 2,
    "n_per_oct": 8,
    "smooth": 0,
    "channels": [ wb.fpga.grad_hist_4_u1 ]
}
shape = (12,12,4)
model = wb.Model(shape, channel_opts)
```

In this example, the input image will be rescaled with 8 scales per octave, channels will be computed by grad_hist_4_u1 function and shrunk by factor of 2. From these maps, samples of shape (12,12,4) will be extracted. Users can freely change feature extraction function and preprocessing parameters to achieve other behaviour. These parameters are passed to the instance of wb.Model which is a class representing the detection model.

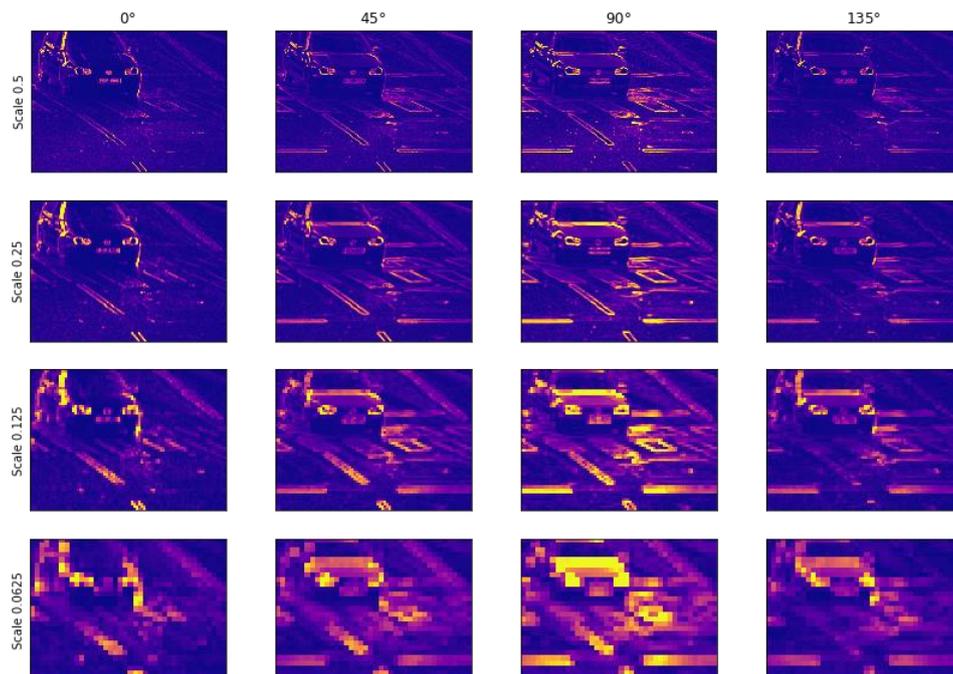


Figure 17: Example of feature maps extracted from image - image scales in rows, channels in columns.

The last thing before the training is dataset specification. The library does not provide a mechanism for dataset representation, this was intentionally omitted to keep the library as general as possible. The training function expects the user to provide an

infinite generator yielding tuples of image (numpy.ndarray) and list of bounding boxes (instances of wb.bbox.BoxList). The implementation of the generator is left for the user. The template can be implemented with the following template:

```
def training_data(self, ...):  
    ...  
    while True:  
        # read image, convert, resize, whatever  
        # compose boxes  
        yield image, boxes
```

The model is then trained by:

```
wb.fpga.train(model, training_data(), length=128, ...)
```

There are lots of other parameters to set up - learner behaviour, training sampler, etc. The function `wb.fpga.train` implements a training algorithm supporting static memory access scheduling and response quantization in weak classifiers which are important for implementation in FPGA. Otherwise, users can use `wb.train` which implements the unconstrained algorithm. After training, the model can be serialized to the binary file by: `model.save("model.pb")` and loaded back as `model=wb.load_model("model.pb")`. The detection model can be applied to an image by:

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

In `boxes`, there is a list of `wb.bbox.BoxList` with locations of detected objects.



Figure 18: Example of detected license plates.

Within FitOptiVis we developed the support for FPGA in the `wb.fpga` module. 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).

5.3. OpenMP for real-time video systems

UC is using the OpenMP standard programming paradigm for system implementation. In this task, we are defining 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.

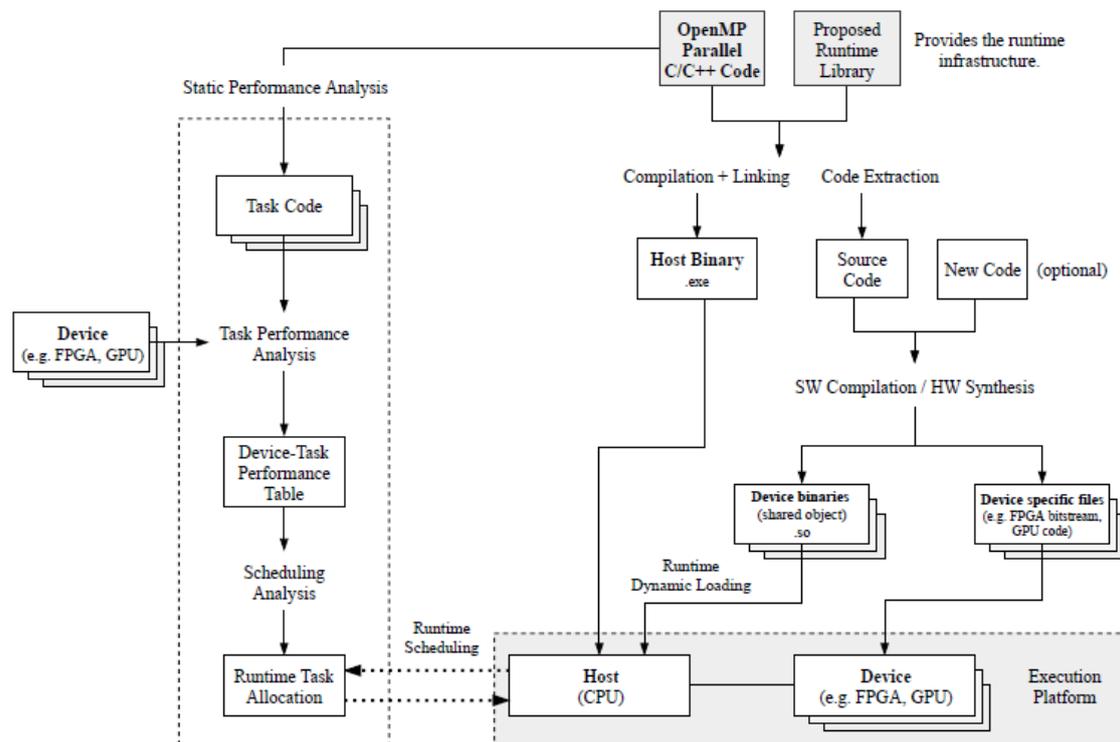


Figure 19: OpenMP-based reconfiguration methodology.

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 18. 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 synthesized 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, a 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

In Y2 UTIA extended design time support for C/C++ compilers and 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 Y2 extensions include SW and HW version of OpenCV algorithms for object detection from Full HD colour video input. See Chapters 7.1 – 7.3 for details.

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

Current versions of TCE support only 32-bit pointers and arithmetics. Wider datapaths are, however, possible with SIMD instructions, but the SIMD vectors may currently also not contain 64-bit elements. The implementation of the 64-bit support is underway. This adds a new target, “tcele64” to the compiler. The compiler automatically selects this mode when it notices that the compiling is performed to a 64-bit TTA architecture.

When compiling code for the tcele64 target, all pointers are 64-bit long and 64-bit integer arithmetics are supported. 64-bit TTAs must contain 64-bit versions of all the basic integer and memory operations, and the general purpose registers must be 64 bits long. The 64-bit instructions have the same base name as corresponding 32-bit instructions, but add a postfix “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 in 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 20 shows the last phases of the TCE compiler and the differences between these two loop optimization modes.

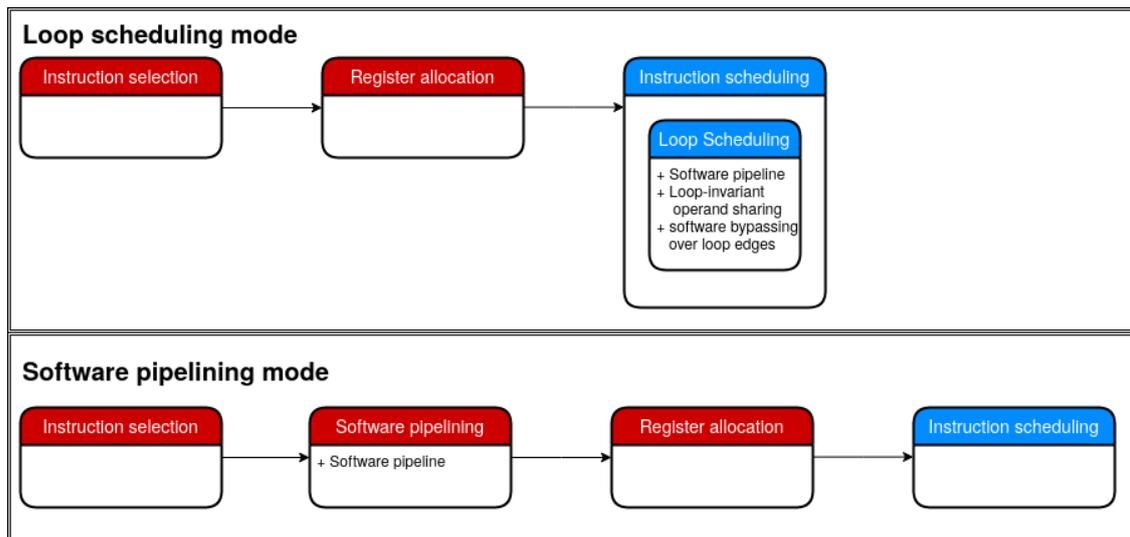


Figure 20: The two different loop optimization modes. Red labels indicate re-used code from LLVM and blue labels indicate separate code managed by TCE code generator.

The loop scheduling mode is 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 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 21. 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.

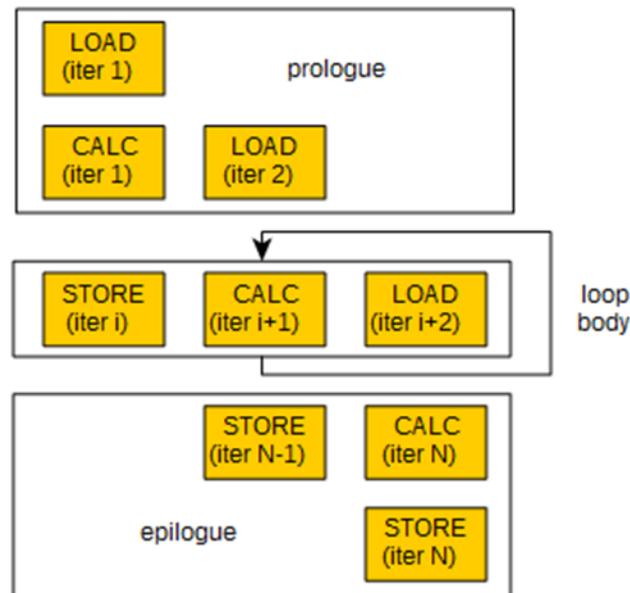


Figure 21: High-level example of software pipelining.

Figure 21 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.

The other loop optimization mode, the software pipelining mode, is created to solve this issue. In this mode, the software pipelining will be done before register allocation. The used software pipelining implementation will just use as many registers as needed and therefore will not be limited to overlapping only 2 iterations. However, this mode currently has other limitations. Because of implementational 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 work in progress in the early stages, it currently only can optimize a limited set of loops. For example, the amount of iterations has to be fixed and not all amount of iterations create a correct result. The implementation still has to be improved in year 3 of the FitOptiVis project to output correct code for all loops.

An example of a loop that can be pipelined in the software pipelining mode at this point is:

```
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. Loop buffer and instruction register file support

The TCE toolset supports multiple compiler-assisted architecture features for optimizing instruction fetch in loops. One is a simple loop buffer which has two interfaces: 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 command for breaking out from the loop. The loop scheduler of the TCE compiler can utilize these instructions 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.

Another mechanism for optimizing instruction fetches in loops is the Instruction Register File (IRF). The IRF is more flexible than the loop buffer, allowing for example if-statements and nested loops. The IRF is like compiler-assisted cache which can contain a single block of code, called IRF block. However, inside the IRF block there can be jumps to any location inside the IRF block, and also jumps which jump 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 a function call cannot be positioned into the middle of IRF block, as the return from a function is a jump. So when there is a function call, a new IRF block starts after the function call.

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, execution resumes from the main instruction memory. If there is a normal jump, 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 the 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 like there was no IRF.

Here are some code examples of 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++;  
}
```

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 22. 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.

Integrating the block copier with the OpenCL runtime developed in WP4 is therefore an important step in ensuring ease of use of the accelerator platform. The primary target for improvement is the signaling behavior. While the current version supports rudimentary signaling – it can postpone the execution of a DMA transfer based on signals and broadcast a signal of its own once a transfer is completed – it currently relies on the host processor executing the OpenCL runtime to propagate those signals to the other devices.

Better handling of signals and moving the management of event waitlists onto the devices would remove the event polling and propagation tasks from the host processor, freeing it to perform useful computations, e.g. executing its own computational kernels.

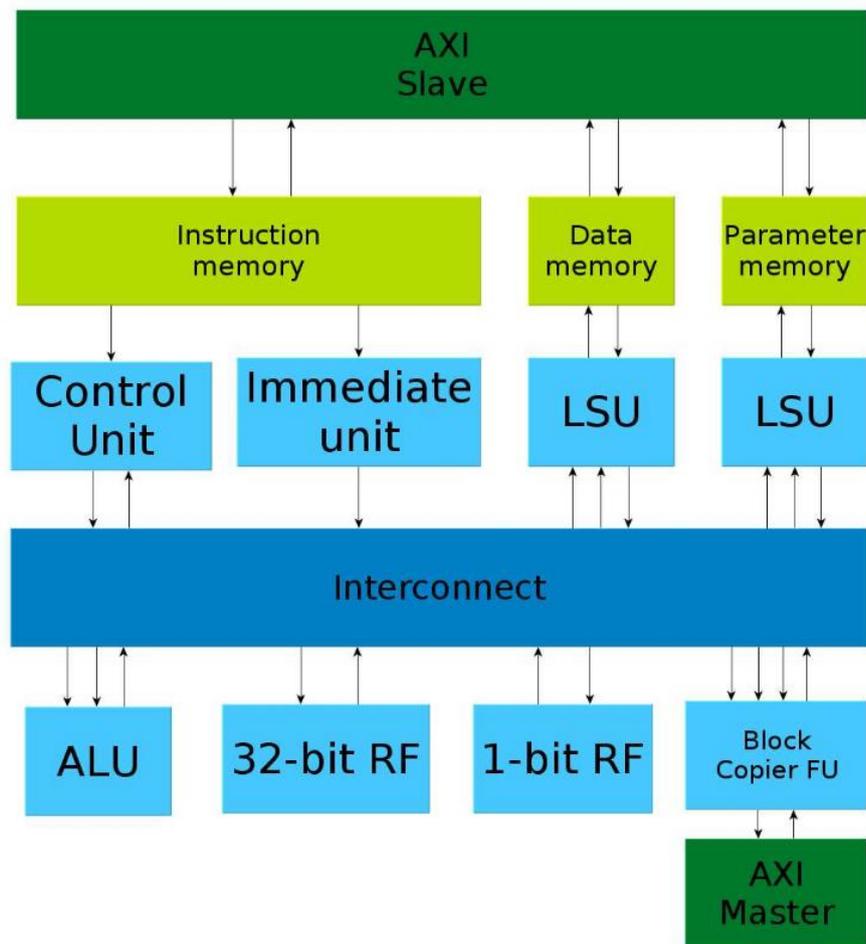


Figure 22: Architecture of the block copier ASIP.

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

Time Sensitive Networks is a set of IEEE 802.1 and IEEE 802.3 technologies providing convergence of time-critical and less critical traffics over bridged Ethernet networks. The latency control, or more specifically, bounded and deterministic latency guarantees are achieved thanks to the end-to-end isolation of the highest traffic priorities from the lower ones. To this end, stringent timely coordination is required between all the stations participating on the data stream propagation.

The different user traffics are classified in four different traffic types by the IEEE 802.1Q VLAN switching logic. Each priority is queued and scheduled according to the IEEE 802.1Qbv Time-Aware traffic Shaper mechanism. The generalized Precision Time Protocol (IEEE 802.1AS, gPTP) allows the required coordination among the stations participating in the data stream forwarding. Furthermore, distributed real-time applications usually require time synchronization for the talker and listener stations, not only in the specific TSN logic but also in the application layer. This way, deterministic delivery between sending and receiving sockets is assured.

The VLAN module is an FPGA based logic providing deterministic switching capability. Incoming data streams are identified by their specific Ethernet, IP or TCP/UDP header values. The following table presents the possible combinations in the prevalence order as they are processed:

Priority	Header Match Events		
	F1	F2	F3
8	Destination MAC		
7	Destination MAC	DSCP	
6	Destination IPv4 address		
5	Destination IPv4 address	IPv4 Protocol	TCP/UDP dest. port
4	Destination IPv4 address	IPv4 Protocol	TCP/UDP source port
3	IPv4 protocol		
2	TCP/UDP source port		
1	TCP/UDP dest. port		

Table 2: VLAN identification rules of user traffic types.

The different traffic types are encapsulated in VLAN Ethernet frames. The figure below shows the different header fields. A VLAN tag is given by a unique VLAN ID (VID) and the Priority code point (PCP), which determines the scheduling and shaping policy applied on the TAS module.

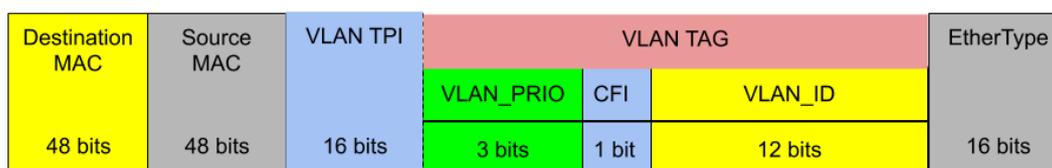


Figure 23: VLAN frame format.

The time-aware traffic shaper performs a priority-based queuing. Each queue is released by a time-aware gate, according to the configuration associated to the gate control list. Each gate control list entry provides a gate configuration, indicating which queues are released or not as well as the time interval.

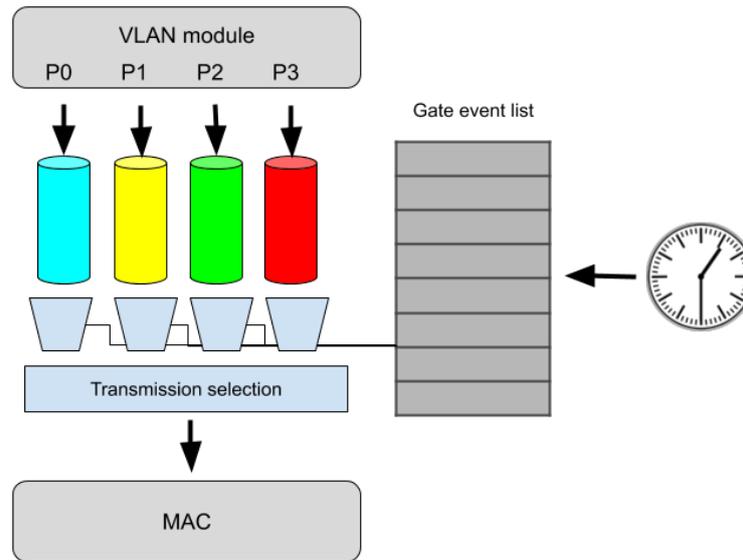


Figure 24: Architecture of the Time-Aware traffic Shaper.

The following figure depicts a possible traffic scheduling and shaping along the time. Usually, guard bands are required to avoid the interference of frames transmitted in previous time intervals. In the first interval, traffic with P3, P1 and P0 intervals are scheduled. In the third interval P3 is transferred without any disturbance and is suitable for deterministic communication. The second and fourth intervals are for guard bands, and the duration is given by the Ethernet maximum transfer unit (MTU), 1500 bytes.

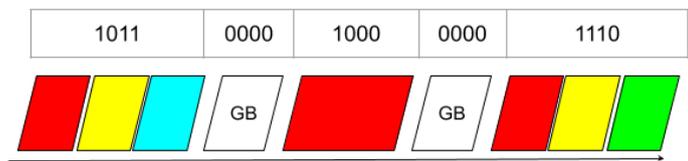


Figure 25: IEEE 802.1Qbv traffic scheduling and shaping.

Guard bands can be minimized if the TAS and the MAC layers cooperate to provide TSN the ability to pre-empt lower priority traffics. IEEE 802.1Qbu in the TAS module and IEEE 802.3br in MAC layer are currently under development. On one hand, the IEEE 802.1Qbu standard defines mechanism to classify traffic priorities in express or preemptable categories. On the other hand, the IEEE 802.3br describes improvements on the Medium Access Layer to enable the frame pre-emption mechanism.

However, the end-to-end determinism can only be achieved if traffic scheduling and shaping is timely aligned along the path, can be seen in the picture below. Moreover, a time shift should be provided to consider the message propagation delay. This is provided to each station by means of the base time configuration parameter (bt_0 , bt_1 , bt_2).

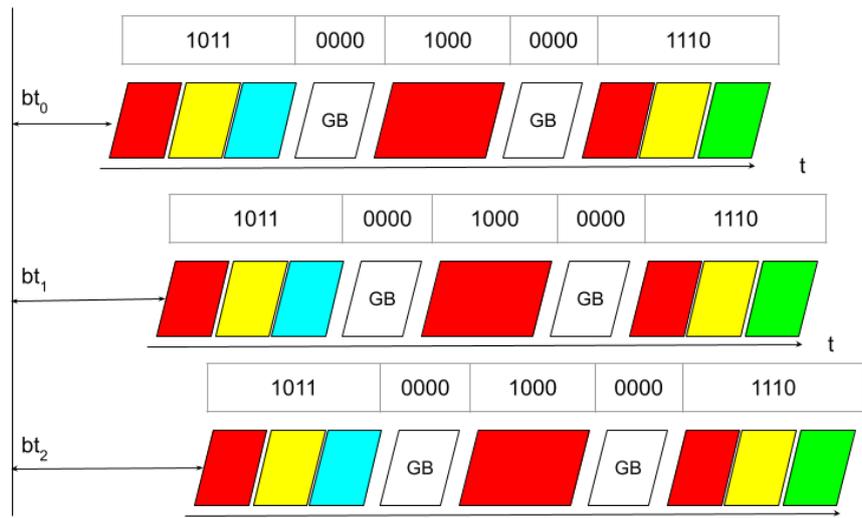


Figure 26: Traffic scheduling and shaping along the TSN stream path.

The generalized Precision Time Protocol (IEEE 802.1AS, gPTP) is key to accomplish fault-tolerant coordination between the stations participating on the data stream forwarding. For this reason, every TSN station should perform the time synchronization. gPTP messages are forwarded through the highest priority queue and require little network bandwidth and processing overhead. Time synchronization is achieved through three major functionalities:

- **Peer delay mechanism.** This service monitors the capability of remote nodes attached to each active interface of a given time-aware station, as performs periodical measurements of the network path delay between each peer node. Besides, nearest-neighbour frequency offset is computed enables better synchronization accuracy.
- **Best Master Clock Algorithm (BMCA).** This mechanism elects the network time reference or grandMaster and announces its attributes to active remote peers. This functionality provides fast convergence and switch-over of the time reference.
- **Transport of the time synchronization messages.** Each participating node propagates synchronization information to remote peer, after accumulating self-computed information such as residence time, frequency rate ratio between local and grandMaster clocks or the link delay with the previous hop.

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.

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, next table presents the main transformations that are required to generate C++ code from a QRML system description.

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 3: VLAN identification rules of user traffic types.

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.

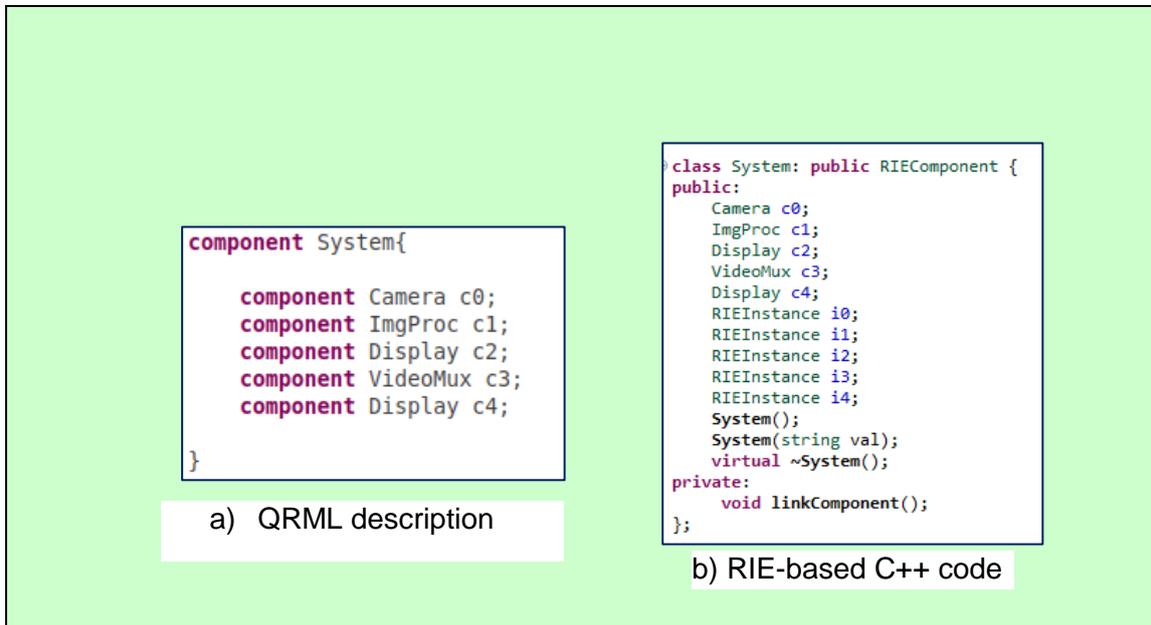


Figure 27: Automatic code generation.

UC has also developed an UML/MARTE generator that transform QRML descriptions into UML/MARTE models. Next figure presents an example of transformation from QRML to UML/MARTE models.

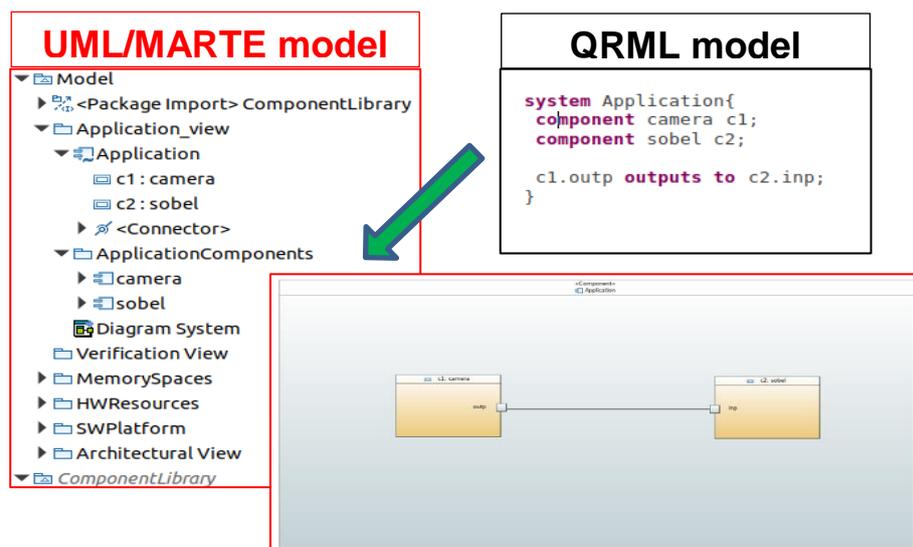


Figure 28: UML/MARTE model generated from QRML description.

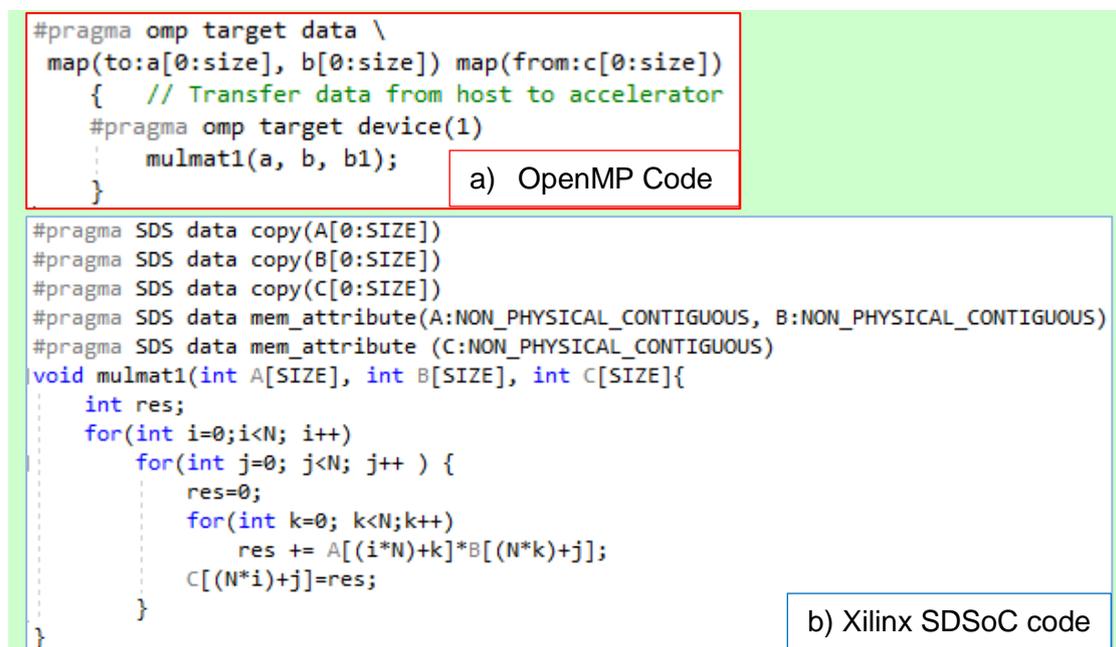
6. Acceleration support

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 (OpenMP, 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. 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. Next figure presents the code of both files. The second approach uses an automatic code extraction tool to offload the OpenMP target region code.

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 29: 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 using shared variables with sequential access based on DMA (Direct Memory Access). The OpenMP accelerator strategy the UC is developing in FitOptiVis try to minimize these overheads.

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

In Y2 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 SG for DSP IP blocks to Arm A9 and A53 on Zynq and Zynq Ultrascale+ devices. These data movers are generated by Xilinx SDSoC 2018.2 system level compiler.
- IP blocks for C/C++ compilers and OpenCV algorithmic libraries suitable for automated compilation to HW accelerator by the Xilinx SDSoC 2018.2 system level compiler and by the Xilinx HLS design flow.

In Y2 UTIA developed these versions of accelerators generated in Xilinx System generator for DSP:

- fp01x8_capabilities capabilities= 10, 20, 30 or 40
- fp03x8_capabilities capabilities= 10, 20, 30 or 40

for

- Zynq 7000 family of devices
- Zynq Ultrascale+ family of devices.

HW description and implementation details of these run-time reprogrammable accelerators are described in D5.2.

SW description and design time use of these of these run-time reprogrammable accelerators are described in D4.2 and D4.3.

The accelerators are compiled from the ARM SW functions executable on the 32 bit dual-core ARM A9 systems or from the SW functions executable on the 64 bit quad-core ARM A53 in case of Zynq UltraScale+ systems.

In M24, the supported HW accelerators include:

- 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 demo (tracking of colour and position of four balls)

The updated Design Time Resources released by UTIA before M24 [7.12], [7.13] export HW design into C++ dynamic libraries. These libraries represent the HW for SW applications. The libraries are dynamically linked to C/C++ user-space applications for Debian OS running on the ARM processor.

- PC cross-compiler can be used for compilation of the top application in free Xilinx SDK 2018.2 Eclipse-based framework. Remote debug of the application on the target device is also possible.
There is no need for SDSoC license for this compilation.
- Embedded g++ compiler can be used for compilation of the C/C++ application directly on the embedded device.
There is no need for SDSoC license for this compilation.

Design time resources introduced in this section are listed in section 10.8 of this deliverable as the “Design Time Resource Integrator of Model Composer IPs (DTRiMC) technology”. Details of the technology are described in Chapters 7.1 – 7.3 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, a 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.

In particular, as a possible first example, state of the art algorithms for image classification are being used, which can be implemented thanks to FPGA-based accelerators obtained using MDC. 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 are going to be 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. Such activities are currently ongoing and are expected to be completed in June 2020. On the top of this setup, several versions of the CNNs will be derived and combined together by MDC, enabling multi-functional CNN hardware accelerators capable of playing with the different CNN versions.

We are considering 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 will be tested collecting different execution times to be compared with the execution time achieved thanks to the accelerator.

UNICA and UNISS enhanced availability of the MDC tool, which is now 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 the latest 12 months 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 worked on extensions of MDC to:

- support 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
- start 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, we expect to receive the feedback of the industrial evaluation carried out by AITEK within the assessment of UC1.

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 the accelerators are currently under evaluation, and the next stage will be the comparison with AITEK proprietary implementations. Assessment at UNISS is planned to be completed in June 2020.

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

All the three provided networks process 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 can achieve different levels of accuracy; they are characterized by different complexities and require a specific amount of computational resources.

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 interconnect 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 interconnect, 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 %.

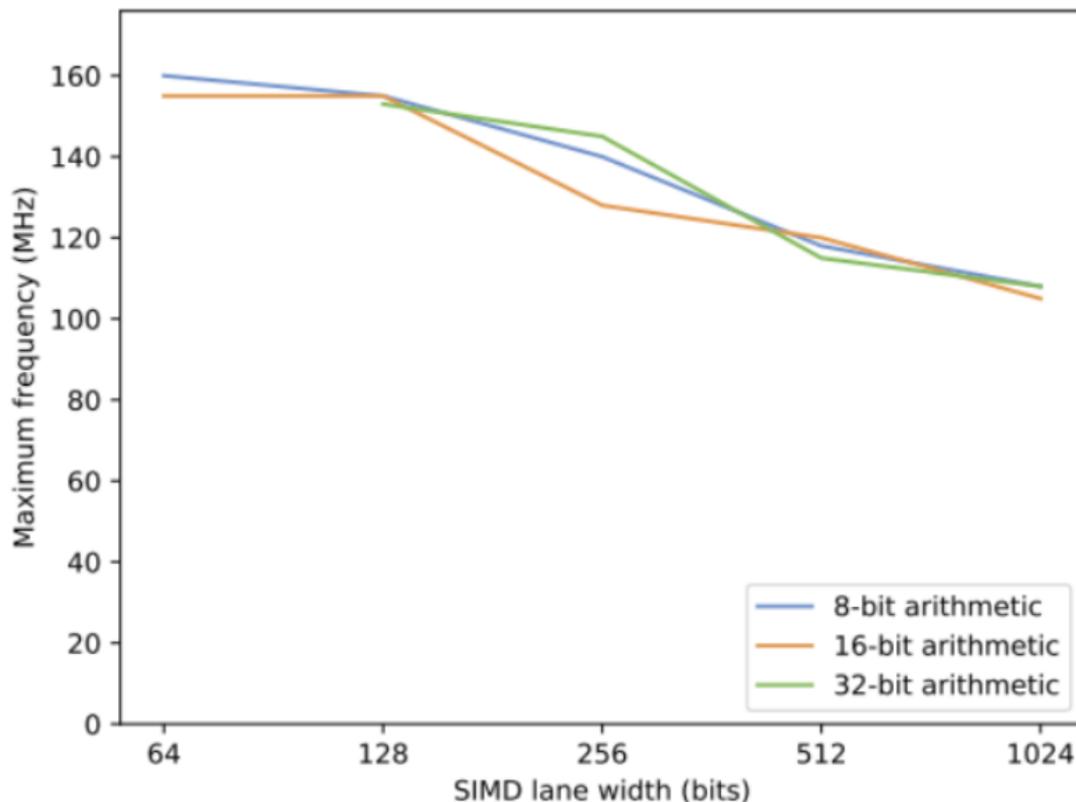


Figure 30: Maximum clock frequency of the synthesized processors.

Vector function units are an easy way to exploit the data level parallelism on programs. To this end, we evaluated the scalability of TTAs to high SIMD widths. The most important function units to vectorize are the load-store unit and 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 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 year 2 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 30 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.

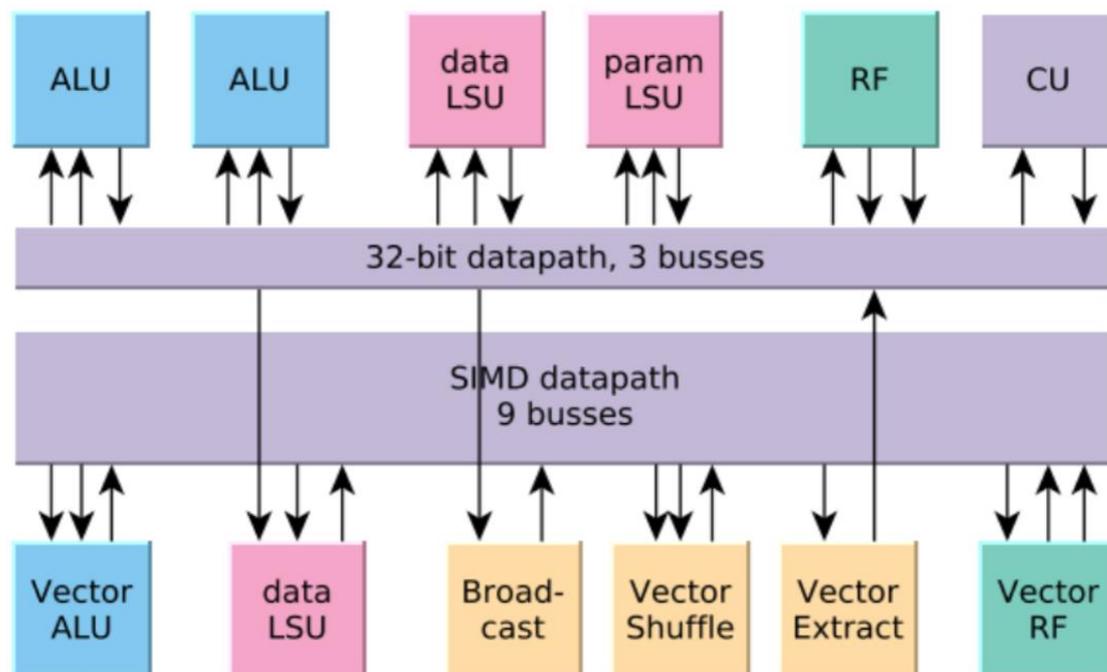


Figure 31: Simplified view of the wide-SIMD TTA template.

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.

Multicore scalability of the TTA-SIMD approach was initially demonstrated on a Zynq UltraScale+ board which could fit 14 customized cores reaching up to 48.5 GOPS real application performance while running a face-detection neural network.

6.5.1. TCE: 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.

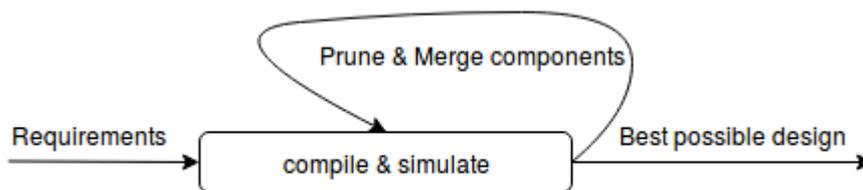


Figure 32: Simplified TCE Exploration process of AutoExplorer.

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 configuration that can be generated.

AutoExplorer (later AEx) is a design space exploration flow of which 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 operations 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 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. This 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 immediates transfers.

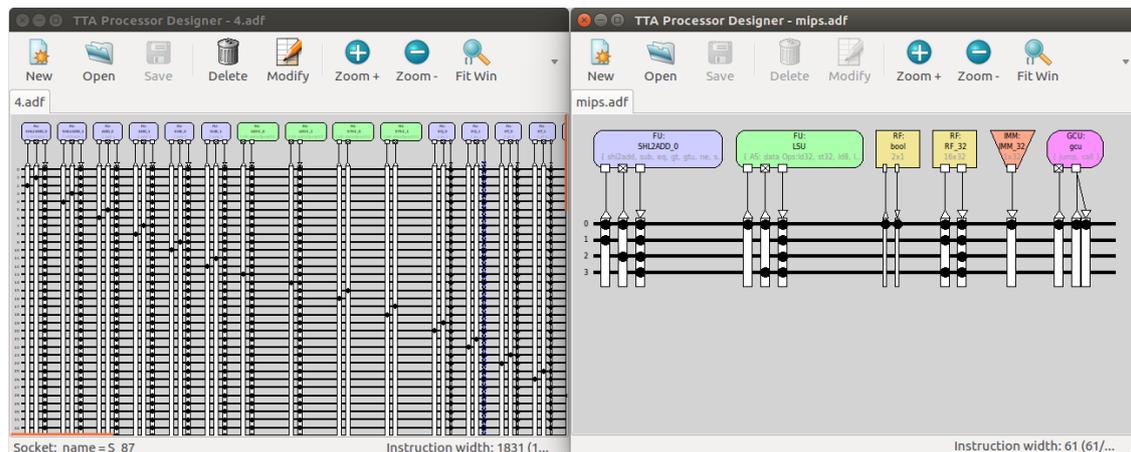


Figure 33: Un-optimized architecture (left), final best possible architecture (right).

Figure 33 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 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 simply AEx2 has been under development in year 2 of FitOptiVis. The overall goal for the new algorithm is 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 those 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 configuration 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 34.

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

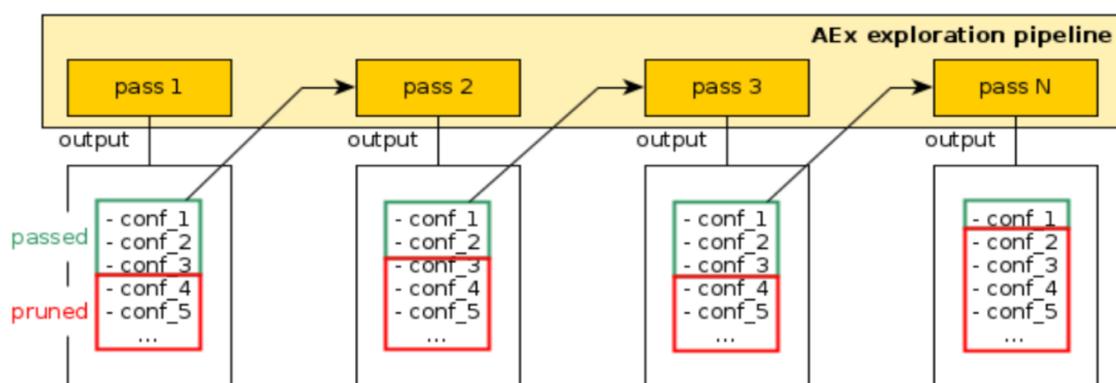


Figure 34: AEx2 result pruning between passes. The configurations marked in red can reach the targeted execution time with the targeted clock frequency.

AEx2: SIMD processor generation with LLVM autovectorization

The autogeneration of efficient SIMD architectures is now implemented utilizing LLVM's autovectorization support. LLVM's offers two autovectorizers, 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 predefined 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

- During the VLIW-connectivity pass, additional connections from each function unit to the boolean register files and 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.

6.6. Acceleration of individual algorithms in combination of FPGA and CPU

6.6.1. 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. This hardware design is an evolution of the older version exploiting LBP features (described in D3.1, [6.2]). The new version is completely redesigned (though it shares some aspects of the old version) and it offers better detection performance, speed and lower memory requirements (actually the memory footprint of one model is about 16x lower). This enables the use of multiple models within one device (e.g. different detection window sizes, multiple object

classes, etc.) which makes the new design more flexible and applicable to different tasks.

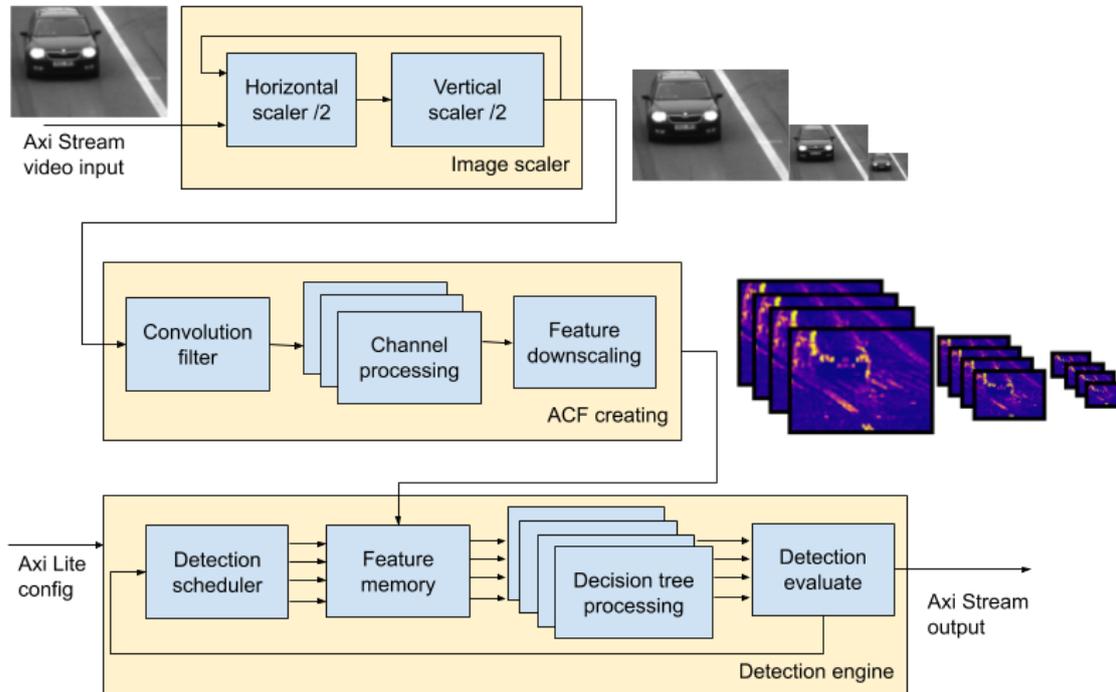


Figure 35: Overview of the detection architecture - (top row) Image scale space generation, (middle row) feature channel calculation, (bottom row) classifier evaluation on each location.

Overview of the detection architecture - (top row) Image scale space generation, (middle row) feature channel calculation, (bottom row) classifier evaluation on each location.

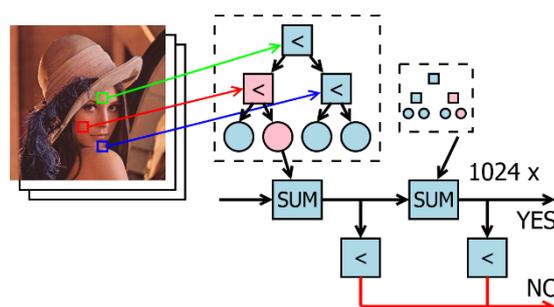


Figure 36: Evaluation of classifier - a decision tree loads features from feature maps and produces response which is aggregated in the total response. It either continues with the next tree or exits with a negative decision.

In principle, the IP core is a micro programmable engine for evaluation of soft cascade classifiers. A classifier is evaluated on all positions of the input image. The classifier uses image features calculated from the input image (not the original pixel values), it supports generic convolution kernels of size 3x3 and HoG-like features (edge response in 4 directions). The feature maps can be optionally pooled by factor of 2 or 4

using max or avg pooling. The features are calculated on-demand and cached in BRAM.

Evaluation of classifier - a decision tree loads features from feature maps and produces response which is aggregated in the total response. It either continues with the next tree or exits with a negative decision.

Multiscale detection is solved by combination of image scaling (by factor of 2) and multiple detectors with different object sizes. In practice we use 4 classifiers detecting the same object type but in 4 different scales (1, 0.84, 0.7 and 0.59). This approach dramatically reduces memory requirements since it requires 4 classifiers (low memory footprint) and saves 3 image scales with high footprint. A great advantage of the new design is static memory access planning which allows for parallel evaluation of multiple features (multiple locations in image). As a result, we can linearly increase performance of the engine with only a minor impact on memory requirements. The static memory access planning is solved in design time during training of the detection model (in WaldBoost package).

The flexibility of the design allows to find tradeoffs in speed, accuracy, resources (logic and memory) for the particular application (there are around 30 parameters configuring the design apart from the model).

	Configuration	LUT	REG	BRAM	Zynq XC7Z020
LBP	HD 60 fps	2x10K	2x8K	2x40	40%
ACF	HD 60 fps	4K	5K	48	10%
ACF	FullHD 60 fps	11K	10K	54	22%

Table 4: FPGA resource consumption for XC7Z020.

The detector speed is not fixed but it depends on the input data (rejecting background positions is almost instant but verifying location with object requires the whole classifier sequence to be evaluated). In order to monitor the state of the device, we implemented gathering of statistical information from the IP Core - processing time, waiting time, pipeline load, etc. We use this runtime information for comparison with theoretical predictions. And based on this runtime data, we can check the device behavior in different conditions (lighting, weather, etc.).

We use Axi Stream Video as an input interface for image data. Depending on the parameters, it can handle up to 4K video. For output (detection coordinates, object likelihood, scale index and statistical data) we use Axi Stream. Engine configuration is managed by Axi Lite memory mapped interface and the configuration is managed by CPU.

The design is written in VHDL and the control library in C. We target the design and test it on Xilinx Zynq platform, but we do not use any platform-dependent libraries and

it can be used with other Xilinx or Intel FPGAs, or it can be synthesized as ASIC (we did not test these possibilities though)

6.6.2. HDR image acquisition

The HDR image acquisition is composed from three main blocks: image capturing, HDR merging and tone mapping (see Figure 37. The image capturing part is driving the exposure time and is grabbing the images from sensor. HDR merging processes multiple images (in our architecture three) into the HDR. The tone-mapping block is compressing the high dynamic range into standard, 8-bit image while preserving the details from HDR.

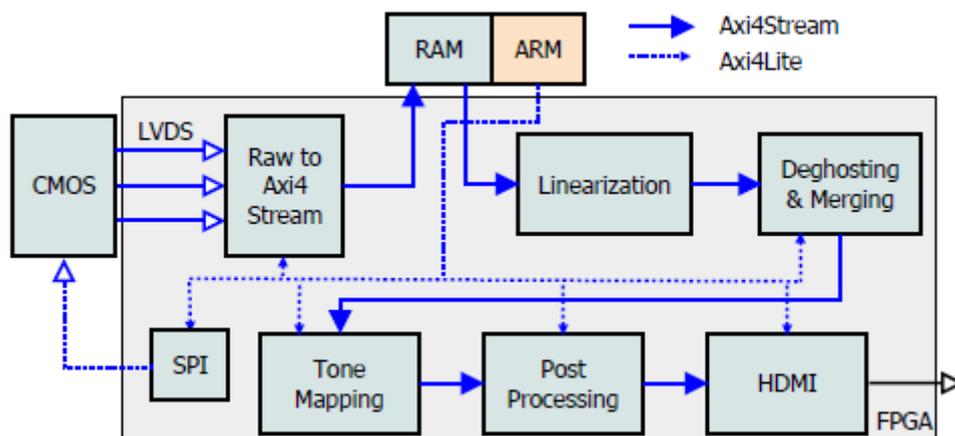


Figure 37: Overall schematics of HDR acquisition pipeline. The individual blocks create a dataflow pipeline, which is configured through AXI Lite interface from ARM CPU.

6.7. HDR merging and de-ghosting

This block implements a ghost-free HDR merging on the configurable number of images. The algorithm is based on the similar principle as the algorithm presented in paper [6.3], while it achieves visually much better results. We implemented the HDR acquisition pipeline for SoC Xilinx Zynq, but we also have a reference implementation running on CPU and GPU based platforms, including SoC Nvidia Tegra. The software version supports merging on grayscale, Bayer mask, and RGB data. The FPGA IP core supports grayscale and Bayer mask only, but we plan to extend it. However, according to new insights, it is convenient to merge HDR in Bayer mask and perform deBayer afterward, e.g. as a part of tonemapping. This approach could save almost $\frac{1}{3}$ of FPGA resources. The IP cores contain two subcores, certainty map (ghostmap) creation and merging block.



Figure 38: Input image sequence (top) and two certainty maps (similar to ghostmaps in related algorithms) used during ghost-free HDR merging.

The output of IP block consists of AXI Video stream with pixels of wider bit depth, by default in the representation of 12.8 bits - 12 fixed point bits and 8 fractional. This block is written in HLS and thus provides an easy change of configuration, input image number and format and also data representation (which depends on previous parameters and desired HDR bit depth).

	LUT	Registers	Brams 18k	DSP
Ghost Detection	3532	3339	4	4
Merging	1650	3636	10	16
Total	5182	6975	14	20
Available	53200	106400	140	220
Utilization [%]	9.8	6.6	10.0	9.1

Table 5: FPGA resource consumption after HLS synthesis for XC7Z020.

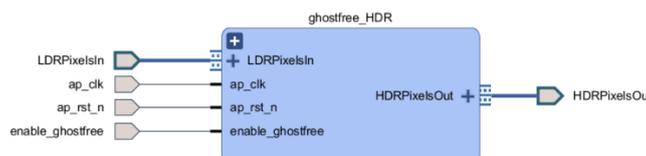


Figure 39: Vivado Block design of ghostfree HDR IP core.

Interfaces:

- LDRPixelsIn - HDR merging block expects multiple AXI Video streams, according to configuration. Input data format is one 8-bit value per pixel with either grayscale or Bayer representation. The streams have to pass

simultaneously; therefore they are concatenated into one stream. Overall width for three images is (23:0), where in (7:0) are the least exposed pixels, (15:8) the middle exposed (reference) pixels and (23:16) the most exposed pixels.

- HDRPixelsOut - Axi Video stream with HDR pixels in format 12.8 (20 bits).
- Enable_ghostfree - set '1' to enable ghost-free feature, '0' to disable.
- AXI Lite (to be done) - Merging block is configured from software. Every change in exposure times results in software modification of constants used by the HW blocks. It is convenient to calculate these constants by software, since it saves a big amount of FPGA resources. The necessary parameters include the exposure ratio between exposures and the pre-calculated values of the gauss function used in the ghost-free algorithm.

In current design, parameters are "hardwired" and the block expects images with exposure ratio of 4, which is equivalent to 2EV (for example 1, 4 and 16ms). During the following months, we are going to work on the block parametrization, we can enable the full adjustment of the ghost-free merging block.



Figure 40: Resulting HDR tonemapped images with ghost-free merging disabled (left) and enabled (right).

6.8. Tonemapping

BUT implemented Durand TMO [6.4], which is based on bilateral filtering and also several basic TMOs, linear, logarithmic and exponential. The TMO block is written in HLS, enabling an easy change of parameters, including the bilateral filter kernel size. We implemented the HDR acquisition pipeline for SoC Xilinx Zynq. We also have a reference implementation running on CPU and also port for GPU based platforms and SoC Nvidia Tegra. The schematics of the TMO is shown on Figure 41. The input image is separated into base and detail layers, where the base layer is compressed and again added back to the detail layer. The histogram is obtained from the base layer in order to tune the base layer scaling factor and other parameters over time.

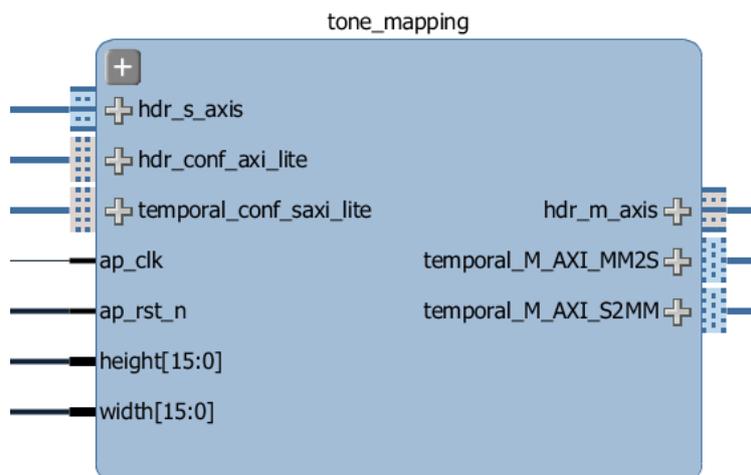


Figure 41: Tonemapping IP core in Vivado.

Interfaces:

- `hdr_s_axis` - AXI Video Stream standard is used for input and output data interface. While the input is the grayscale image with Bayer mask with 32bits per pixel,
- `hdr_m_axis` - the output of TMO is 8bit RGB image.
- `hdr_conf_axi_lite` - AXI Lite interface is used for grabbing the image histogram and other statistical data, which are used for passing back the software adjusted tonemapping properties (e.g. min/max values, scale coefficients for histogram balancing).
- `temporal_conf_axi_lite` - AXI Lite interface is used for configuration of temporal part of the algorithm
- `width` and `height` constants - size of the input image

Within FitOptiVis, we plan to involve the temporal coefficient filter, which will enable the smoother brightness adaptation - without this smoothing, the resulting image might suffer from flickering, e.g. from light sources powered from PWM power sources or by AC power. We experimented with several software implementations of temporal TMOs. We incorporated temporal attributes into software implementation of the Durand algorithm and we evaluated two possible temporal extensions of this algorithm. The first one is temporal filtering of the base layer with values of base layer from previous frames similarly to [6.5]. This approach requires one additional DMA for buffering of previous values which consumes a large amount of resources. Second possible extension is temporal filtering of global parameters (maximum and minimum values of luminance in the frame). Our implementation is based on filtering using IIR filter. This approach is also very efficient for FPGA implementation because additional DMA is not necessary. AXI4 Lite interface is sufficient because the data rate is low (3 fixed-point values for one frame - maximum minimum and average) and values can be computed in the ARM CPU.

We also converted the algorithms to fixed point arithmetic which is natural for FPGA. As a consequence, the resource consumption decreased by 35%, while it allowed the bilateral filter to extend the convolution kernel from 7x7 to 9x9 pixels.

	LUT	Registers	Brams 18k	DSP
Floating Point	29314	21883	51	71
Fixed Point	19707	2570	45	41

Table 6: Resource consumption.

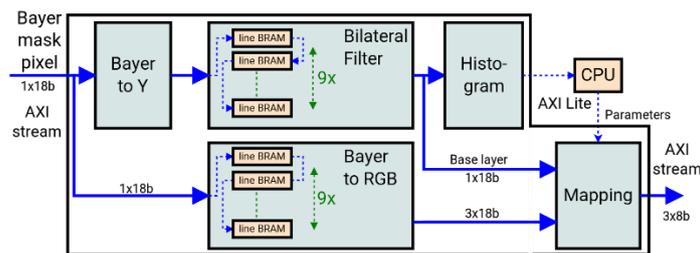


Figure 42: The FPGA implementation of Durand tonemapping.

6.9. Convolutional HW accelerator

This accelerator core will perform convolutional filtering on image data. The approach will use limited precision number space with high dynamic range (compared to the number of bits used). The details will be derived from the modelling task 3.1, after a suitable parameter space has been found, based on simulations.

The core will be compatible with CNN based image / video content analysis, and initially it is planned to be integrated with YOLO open source content analysis software. Description of the YOLO can be found in [6.12].

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 will be especially valuable in case of neural network running several convolutional kernels over the same input image. In hardware, especially with limited precision, 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 will be implemented in FPGA for prototyping and testing and then in silicon for performance evaluation. It is expected that the silicon version may be scaled down, as the amount of memories and / or bandwidth issues may impose restrictions. On the other hand the FPGA prototype can be scaled to match the silicon device, to provide a reference point between the two implementations.

6.10. 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 43 is used for encoding while for decoding the block structure in Figure 44 is used instead.

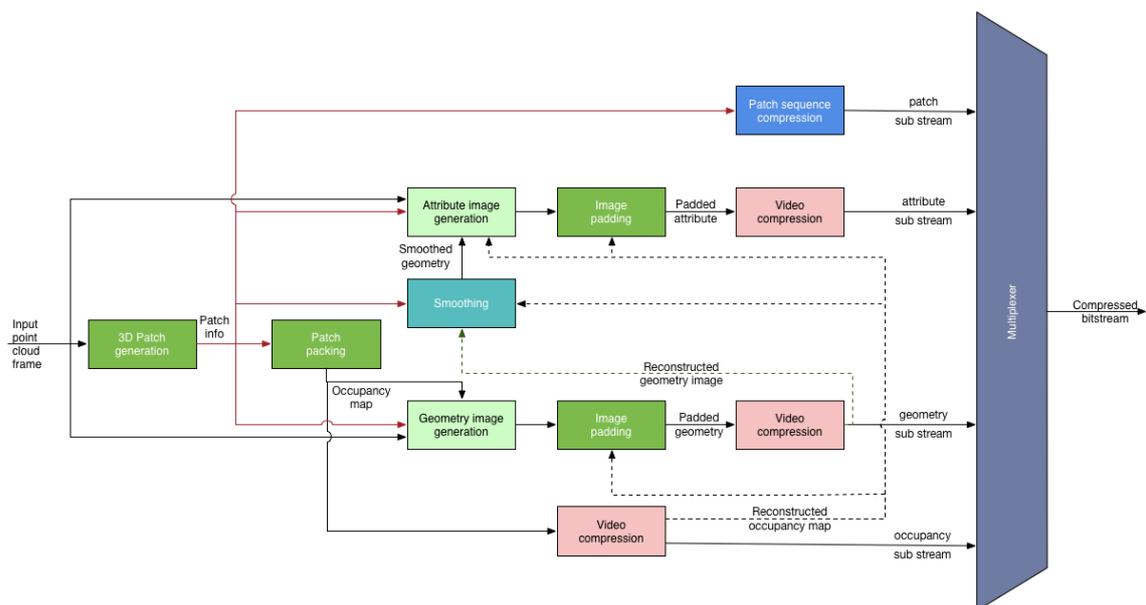


Figure 43: V-PCC encoding structure.

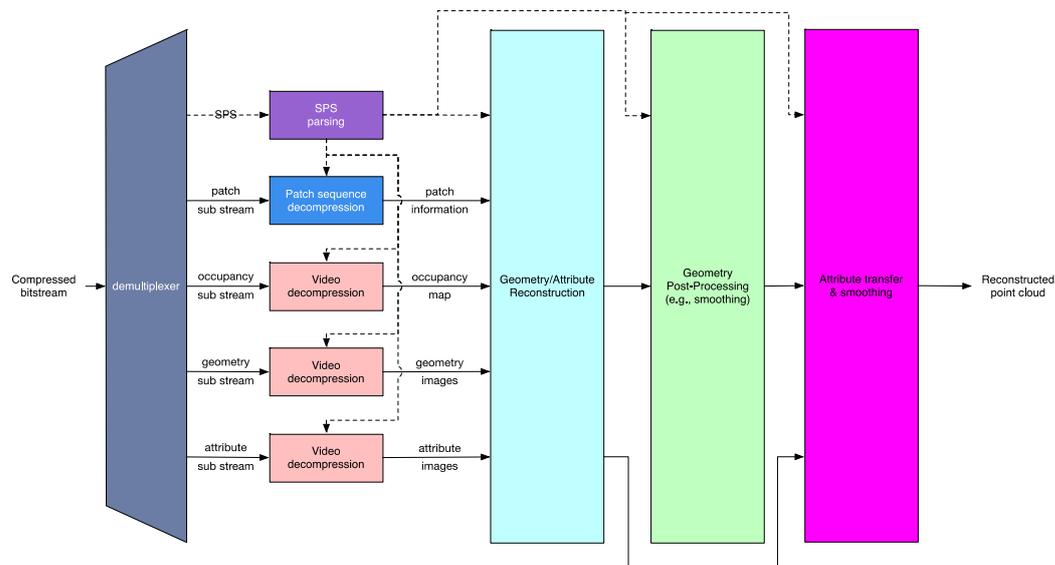


Figure 44: V-PCC decoding structure.

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 from 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 upscled 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.

6.11. 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 a detection is approximately 30 x 40 and 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. Results are shown in Figure 45 and Figure 46. 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.

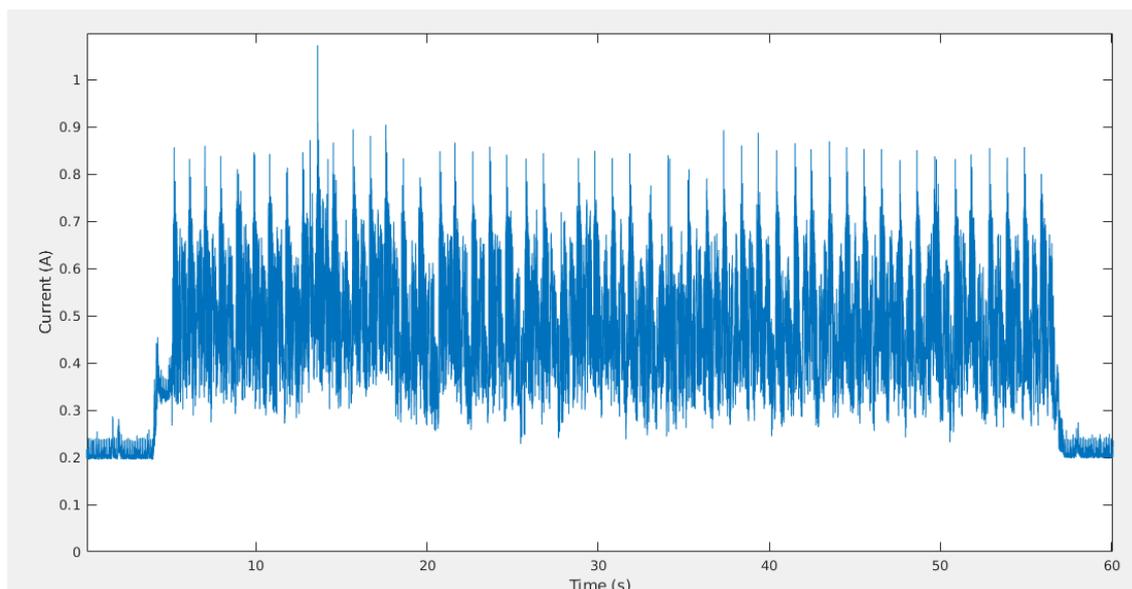


Figure 45: Power consumption measurement (mA) when running face detector for 1 (4096 × 2156) size frames with CPU implementation. X-axis shows time and y-axis shows current in the range between 0 and 1100 mA.

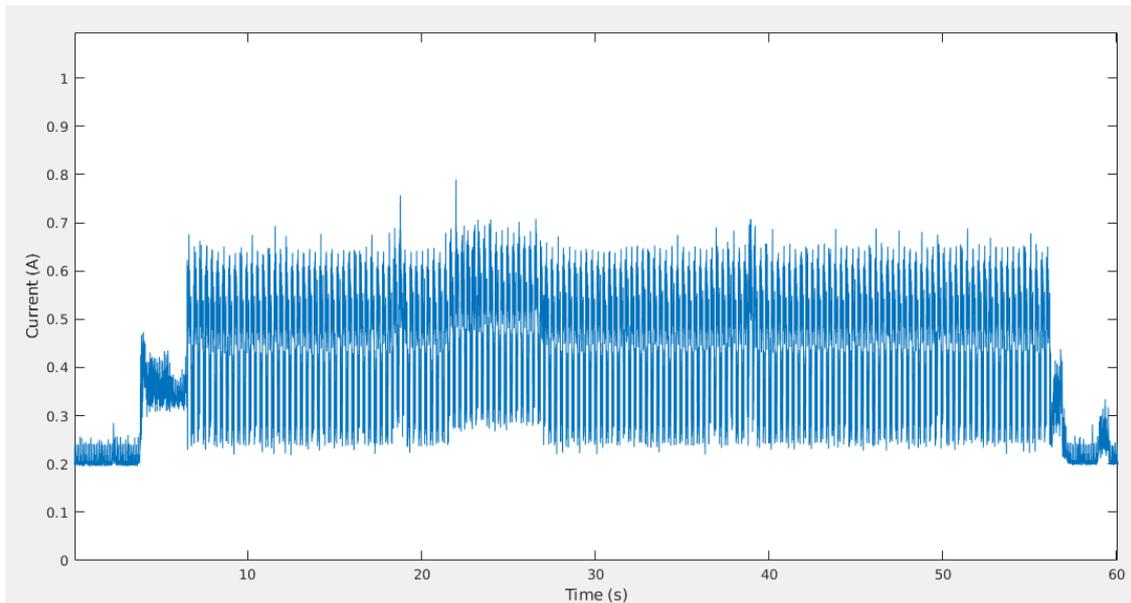


Figure 46: Power consumption measurement (mA) when running face detector for (4096 × 2156) size frames with GPU implementation. X-axis shows time and y-axis shows current in the range between 0 and 1100 mA.

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.

7. Design Time Support for Methodologies and Tools

This chapter describes design time support for methodologies and tools released in Y2 by FitOptiVis WP3 partners and to general public use before M24 of the project (end of April 2020) in [7.12] and [7.13]. These released Y2 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 T3.1, T3.2 and T3.3.

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 6 summarizes the progress made by the FitOptiVis partners in the WP3 from M1 to M12.

<ul style="list-style-type: none"> ➤ ALMARVI - end of project: <ul style="list-style-type: none"> ▪ Zynq 7000 family (28nm) ▪ Stand-alone only ➤ ALMARVI limitations: <ul style="list-style-type: none"> ▪ Limiting PL size of Zynq ▪ no VCU, no GPU ▪ no USB support ▪ no Ethernet board 2 board communication framework 	<ul style="list-style-type: none"> ➤ FitOptiVis – Y1: <ul style="list-style-type: none"> ▪ + Zynq Ultrascale+ 16nm ▪ + Small: ZynqBerry 28nm ▪ + PetaLinux OS ▪ + Debian FS support ➤ FitOptiVis progress: <ul style="list-style-type: none"> ▪ + Large PL of Ultrascale+ ▪ + VCU, + GPU, ▪ + USB ▪ + Ethernet board 2 board communication based on Arrowhead Framework
---	---

Table 7: Progress made in FitOptiVis in WP3 in Y1.

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+
- Generation 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

7.1. Y2 extension of support for the ZynqBerry board TE0726

The Y2 design time support is based on Xilinx Vivado-HLS 2018.2, Vivado SDSoC 2018.2 and Petalinux 2018.2. In Y2, we have added support for designs with Xilinx SG for DSP HW IPs with data streaming interfaces.

7.1.1. Y2 support for designs with Xilinx SG for DSP data streaming IPs for ZynqBerry module

Xilinx SG for DSP with data streaming IP can be designed and modelled in Matlab and Simulink ver. 2018a. The IPs are exported as RTL blocks for Xilinx Vivado 2018.2 and become part of the initial HW platform for the SDSoC compiler. The technology

supporting this export is summarized in section 10.8 as “Design Time Resource Integrator of Model Composer IPs (DTRiMC) Technology” [7.12], [7.13].

The Y1 design time resource HW platform for the ZynqBerry board [7.1] was in Y2 extended with one 8xSIMD run-time reprogrammable single precision HW accelerator FP01x8 [7.12] The accelerators is run-time reprogrammable and the provided SW demo performs single precision matrix by matrix multiplication $C[64,64] = A[64,64] * B[64,64]$ as an application example. See D5.2 for internal details of the accelerator and D4.2 and d4.3 for SW API details and descriptions.

The Y2 design time resource provides for the SW designer set of precompiled shared libraries representing the HW platforms for Zynq 7000 (ZynqBerry) TE0726 with Single 8xSIMD FP01 Accelerator without support for floating point division operation:

- `libfp01x8_v26x1_dma_hw.so`
- `libfp01x8_v26x1_sg_malloc_hw.so`
- `libfp01x8_v26x1_hw.so`
- `libfp01x8_v26x1_zc_sg_hw.so`

The libraries represent different data movers used for connection of the 8xSIMD run-time reprogrammable single precision HW accelerator FP01x8 in these HW configurations:

- **`libfp01x8_v26x1_hw.so`** is using Zero Copy HW data movers. It is not using the DMA IP cores. The data movers are realised as C++ function compiled to HW by the SDSoc 2018.2 compiler. The HW supported data transfer requires data to be present in “sd_alloc” memory (continuous physical section reserved in the DDR3). Start of the data transfer is no blocking. The end of data transfer is tested by pooling. The SW overhead needed to start this data transfer is minimal.
- **`libfp01x8_v26x1_dma_hw.so`** is using DMA HW data movers. The HW supported data transfers require data to be present in “sd_alloc” memory (continuous physical section reserved in the DDR3). Start of the data transfer is no blocking. The end of data transfer is tested by pooling. The SW overhead needed to start this data transfer is larger in comparison to the Zero Copy data mover.
- **`libfp01x8_v26x1_sg_malloc_hw.so`** is using combination of Zero Copy HW data mover and DMA SG HW data mover with interrupt. The HW supported data allocated by “sd_alloc” memory (continuous physical section reserved in the DDR3). Start of the data transfer is no blocking. The end of data transfer is tested by interrupts. The SW overhead needed to start this data transfer is larger in comparison to the DMA data mover. Data can be allocated in the standard Linux user-space memory, allocated by the standard Linux “malloc” function. This is the only HW implementation capable to work directly with standard “malloc” allocated linux data.
 - If “malloc” data allocation is used, the overhead of this SG DMA is really large.
 - If “sd_alloc” data allocation is used (continuous physical section reserved in the DDR3), the overhead of this SG DMA is larger in comparison to DMA based HW support, but much shorter in comparison to the case of data allocation based on standard “malloc”.

- **libfp01x8_v26x1_zc_sg_hw.so** is using DMA SG data mover with “sd_alloc” allocated data and interrupts. Start of the data transfer is no blocking. The end of data transfer is based on interrupt. The SG FMA is using the advanced coherent port of the Zynq device. There is no need to flush the Zynq cache before accessing of data.

Device: 7z010clg225-1	lut	reg	bram	dsp
Available (100%)	12462	15376	60	80
One FP01X8_v26_40 Accelerator				
fp03x8_v26x2_dma_hw	70,81%	43,68%	69,17%	40,00%
fp03x8_v26x2_hw	66,19%	39,06%	63,33%	40,00%
fp03x8_v26x2_sg_malloc_hw	84,39%	55,68%	77,50%	40,00%
fp03x8_v26x2_zc_sg_hw	82,55%	53,89%	75,00%	40,00%

Table 8: HW resources used by the FP01x8 Accelerator with different data movers.

7.2. Y2 extension of support for Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O

The Y2 design time support extensions have been based again on Xilinx Vivado-HLS 2018.2, Vivado SDSoC 2018.2 and Petalinux 2018.2. Figure 46 presents illustrative example of an initial HW design for Zynq Ultrascale+ used for the Y2 design time resource generation. Figure 47 presents illustrative example of the final HW platform for Zynq Ultrascale+ used for the Y2 design time resource generation.

The released HW design has fixed HW with selected set of auto-generated data movers. The HW design is represented to SW if form of shared (.so) library with unified API. SW designer can link the shared library to the SW application written in C/C++ and compiled by standard gcc compiler and “make”.

The “fixed” PL hardware remains run-time programmable by change of the firmware of the integrated 8xSIMD HW accelerators. SW designer does not need the SDSoC compiler license for compilation of the SW application.

In Y2, we have added additional support on top of the Y1 designs serving for integration of multiple Xilinx SG for DSP IPs with data streaming interfaces. In addition, we now support configurations with two HW accelerators connected in serial chain or connected in parallel, to demonstrate the basic scalability.

We also provide application example with parallel computation of video processing algorithm (Sobel filter edge detection on Full HD video) working in parallel with matrix multiplication on two 8xSIMD HW accelerators.

See D5.2 for internal details of the Accelerator and D4.2 and d4.3 for SW API details and descriptions. The Y2 design time resource provides for the SW designer set of precompiled shared libraries representing the HW platform.

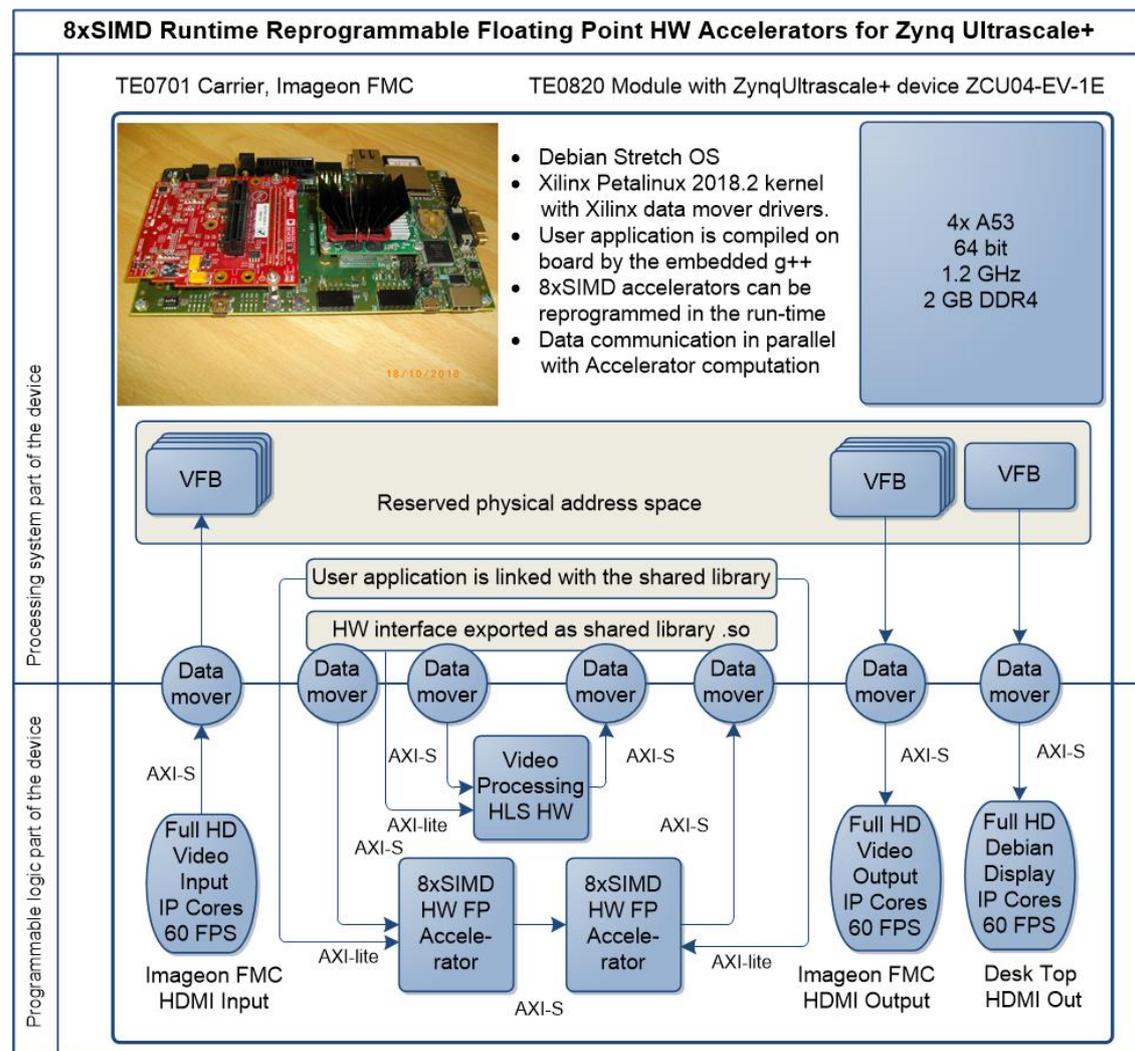


Figure 48: WP3 Y2 Initial Zynq Ultrascale+ platform with two serial connected accelerators.

These released shared libraries represent different data movers used for connection of the two 8xSIMD run-time reprogrammable single precision HW accelerators FP02x8 in these HW configurations:

- Two 8xSIMD FP03 Accelerators, serial connect, no or only SW video processing:
 - libfp03x8_v26x2_dma_hw.so
 - libfp03x8_v26x2_sg_malloc_hw.so
 - libfp03x8_v26x2_hw.so
 - libfp03x8_v26x2_zc_sg_hw.so
 - libfp03x8_v26x2_sg_hw.so

- Two 8xSIMD FP Accelerators, parallel connect, no or only SW video processing:
 - libfp03x8_v26x2p_dma_hw.so
 - libfp03x8_v26x2p_sg_malloc_hw.so
 - libfp03x8_v26x2p_hw.so
 - libfp03x8_v26x2p_zc_sg_hw.so
 - fp03x8_v26x2p_sg_hw.so
- Two 8xSIMD FP Accelerators, serial connect, HW video processing (sobel)
 - libsobel_dma_v26x2p_hw.so

The released systems exported for the SW programmers have these properties:

- **libfp02x8_v26x2_hw.so** is working with two serial connected accelerators. It is using Zero Copy HW data movers. It is not using the DMA IP cores. The data movers are realised as C++ functions compiled to HW by the SDSoc 2018.2 compiler. The HW supported data transfers require data to be present in “sd_alloc” memory (continuous physical section reserved in the DDR3). Starts of the data transfers are no blocking. The end of data transfers are tested by pooling. The SW overhead needed to start this type of data transfer is minimal. It works with two Zero Copy HW IPs.
- **libfp02x8_v26x2_dma_hw.so** is working with two serial connected accelerators. It is using DMA HW data movers. The HW supported data transfers require data to be present in “sd_alloc” memory (continuous physical section reserved in the DDR3). Start of the data transfer is no blocking. The end of data transfer is tested by pooling. The SW overhead needed to start this data transfer is larger in comparison to the Zero Copy data mover. It works with two DMA HW IPs.
- **libfp02x8_v26x2_sg_hw.so** is working with two serial connected accelerators. The HW supported data transfer requires data to be present in “sd_alloc” memory (continuous physical section reserved in the DDR3). Start of the data transfer is no blocking. The end of data transfer is tested by pooling. The SW overhead needed to start this data transfer is larger in comparison to the DMA data mover. It works with two SG DMA HW IPs.
- **libfp02x8_v26x2_sg_malloc_hw.so** is working with two serial connected accelerators. It is using DMA SG data mover with “malloc” allocated data and interrupts. Data can be allocated in the standard Linux user-space memory, allocated by the standard Linux “malloc” function. Start of the data transfer is no blocking. The end of data transfer is based on interrupt. The SG FMA is using the advanced coherent port of the Zynq device. There is no need to flush the Zynq cache before accessing of data. This is the only HW implementation capable to work directly with standard “malloc” allocated linux data.
 - If “malloc” data allocation is used, the overhead of this SG DMA is really large.
 - If “sd_alloc” data allocation is used (continuous physical section reserved in the DDR3), the overhead of this SG DMA is larger in comparison to DMA based HW support, but much shorter in comparison to the case of data allocation based on standard “malloc”.It works with two SG DMA HW IPs.
- **libfp02x8_v26x2_zc_sg_hw.so** is working with two serial connected accelerators. It is using one Zero copy data mover and one DMA SG data mover with interrupt. It requires “sd_alloc” allocated data. The end of data transfer is based on interrupt.

- **libfp02x8_v26x2_hw.so** is working with two parallel connected accelerators. It works with four zero copy data movers implemented in HW.
- **libfp02x8_v26x2_dma_hw.so** is working with two parallel connected accelerators. It works with four DMA HW IPs.
- **libfp02x8_v26x2_sg_hw.so** is working with two parallel connected accelerators. It works with four SG DMA HW IPs with interrupts and data are allocated by “sd_alloc” or “malloc”.
- **libfp02x8_v26x2_sg_malloc_hw.so** is working with two parallel connected accelerators. It works with four SG DMA HW IPs with interrupts and data can be allocated by “malloc” or “sd_alloc”.
- **libsobel_sw_v26x2_hw.so** computes in SW the sobel-filter edge detection video processing algorithm in Full HD. It computes in parallel two floating point matrix multiplications on two serial connected accelerators. It works with two Zero Copy HW IPs.
- **libsobel_dma_v26x2p_hw.so** performs HW accelerated sobel-filter edge detection video processing algorithm in Full HD. It computes in parallel two floating point matrix multiplications on two serial connected accelerators. It works with two DMA HW IPs.
- **libsobel_sg_v26x2_hw.so** It computes in parallel two floating point matrix multiplications on two serial connected accelerators. It works with two SG DMA HW IPs

Device: xczu4evsfc784-1	lut	reg	bram	dsp
Available (100%)	41322	48499	128	728
Two serial connected FP03X8				
fp03x8_v26x2_dma_hw	48,08%	28,13%	75,39%	9,07%
fp03x8_v26x2_hw	47,04%	27,61%	73,83%	9,07%
fp03x8_v26x2_sg_hw	53,69%	33,81%	89,06%	9,07%
fp03x8_v26x2_sg_malloc_hw	53,69%	33,81%	89,06%	9,07%
fp03x8_v26x2_zc_sg	50,70%	30,90%	82,03%	9,07%
Two parallel connected FP03X8				
fp03x8_v26x2p_dma_hw	55,00%	34,00%	89,45%	9,07%
fp03x8_v26x2p_hw	53,66%	33,46%	87,89%	9,07%
fp03x8_v26x2p_sg_hw	60,29%	38,30%	98,05%	9,07%
fp03x8_v26x2p_sg_malloc_hw	60,29%	38,30%	98,05%	9,07%
fp03x8_v26x2p_zc_sg	59,28%	37,70%	97,29%	9,07%
Two serial con. FP03X8 and Sobel filter				
sobel_dma_v26x2_hw	53,86%	33,20%	87,11%	9,07%
sobel_sg_v26x2_hw	58,07%	36,72%	94,53%	9,07%
sobel_sw_v26x2_hw	47,04%	27,61%	73,83%	9,07%

Table 9: HW resources used by the FP01x8 Accelerators with different data movers.

7.2.2. Y2 acceleration results on Zynq Ultrascale+ with video I/O.

The two run-time reprogrammable HW accelerators FP03x8 with released provided SW demo performs two single precision matrix by matrix multiplications $C(64,64) = A(64,64) * B(64,64)$ as an application example. The two 8xSIMD HW accelerators running in HW with 240 MHz clock accelerate the SW computation compiled by g++ with -O3 optimisation for single core of Arm A53 (1.2 GHz) 5x.

7.2.3. Y2 acceleration results on ZynqBerry with video I/O.

Single run-time reprogrammable HW accelerator FP01x8 with released SW demo performs one single precision matrix by matrix multiplication $C(64,64) = A(64,64) * B(64,64)$ as an application example. The 8xSIMD HW accelerator running ZynqBerry HW with 115 MHz clock accelerates the SW computation compiled by g++ with -O3 optimisation for single core of Arm A9 (650 MHz) 2.5x.



Figure 49: Full HD edge-detection and matrix multiplication on two FP03x8 accelerators.

Figure 49 demonstrates processing of the full HD HDMI 60 FPS video by HW accelerated Sobel filter performing edge detection and in parallel computation of two single precision floating point matrix multiplications in two serial connected FP03x8 HW accelerators.

The Edge detection is accelerated 20x by the fixed SDSoC HW accelerator and the matrix multiplications are accelerated **5x** by the two run-time reconfigurable 8xSIMD accelerators FP03x8.

7.3. Complete runtime reconfiguration of PL part of Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O

In Y2 WP3 released design time support package enabling to perform complete reconfiguration of PL part of Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O. This Y2 design time support extensions have been based again on Xilinx Vivado-HLS 2018.2, Vivado SDSoC 2018.2 and Petalinux 2018.2.

Requirements for HW/SW

- Standard HW/SW design for the HW supported by the WP3 Y1 design time resources for Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O.
- Debian OS with installed UTIA GUI for run-time reconfiguration support
- SW user application capable to start and stop the Video FNA engines.
- SW user application without command-line input. Ascii output to console is OK.

The realised design time support includes all necessary elements needed for stop of the data traffic in the Video DMA engines as well as in the Full HD Video Input and Output connections.

Petalinux executes two applications.

- The GUI based on Quick Time serving for the user control of reconfiguration
- The actual SW application dealing with video processing and PL logic accelerators.

If we decide to stop the SW application, the GUI application on Debian Desktop, the Petalinux 2018.2 kernel, the Debian file system and its services like the Ethernet support continue to run.

The GUI first sends signal to the running SW application to stop. The application has to stop all DMA and VDMA and terminates. Then the GUI waits for user input (selection of new SW-only or SW with HW support application). The PL logic is reprogrammed internally in the device with new bitstream. This process is controlled by the GUI application.

Finally the GUI starts the new selected application SW. This new application re-starts all DMA and VDMA units and reconnects to the HDMI I/O. This design time resource enables fast evaluations of different HW accelerated versions of algorithms.

The limitation of the framework is the temporary loss of control over the PL logic and all its I/O pins and the need to support automatic reconnection of the broken Full HD HDMI I/O digital video data paths.

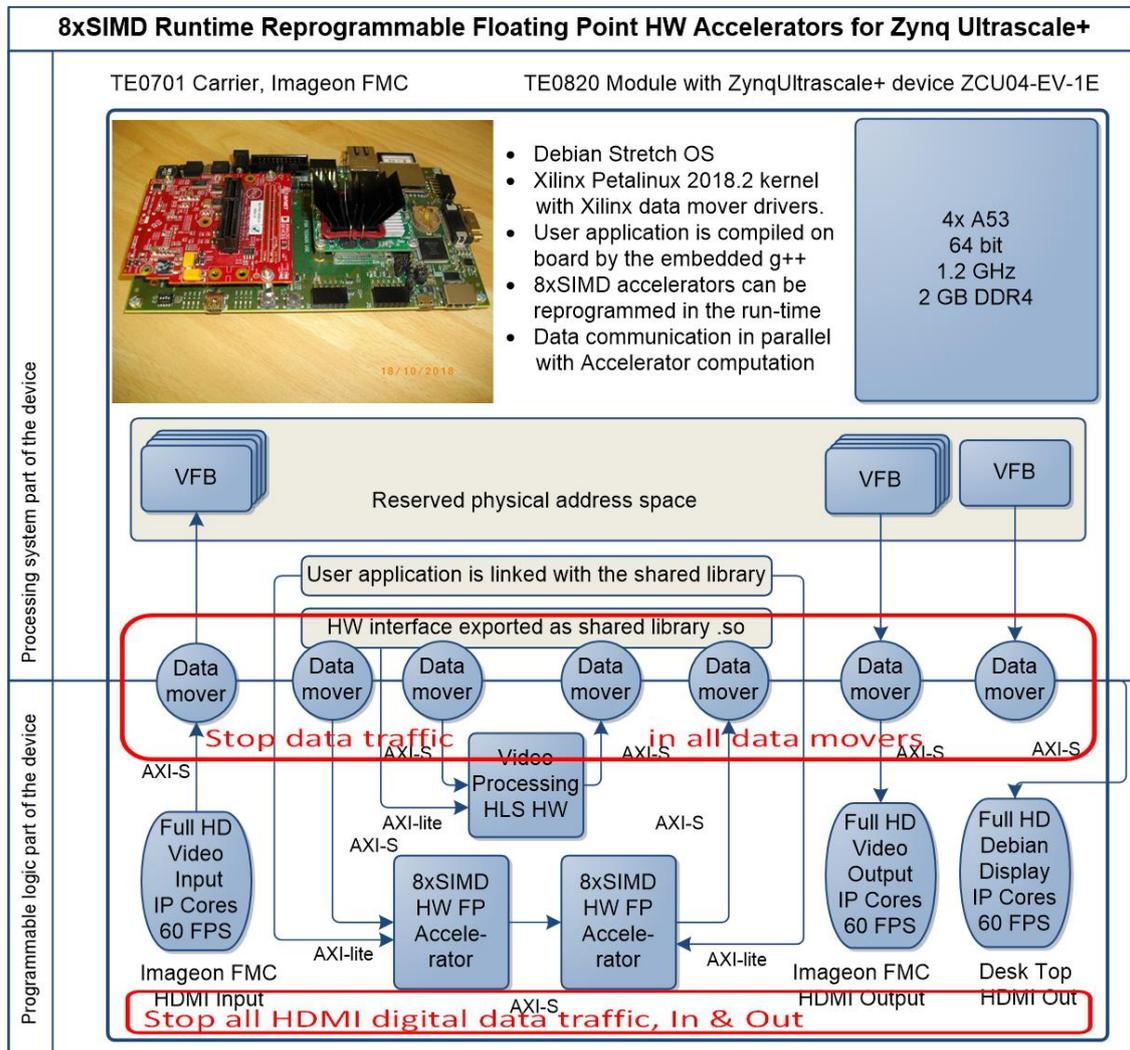


Figure 50: Function of all HW data movers have to be terminated before stop of SW app.

Platform supporting the PL reconfiguration was presented to project partners during the FiOptiVis F2F meeting 3-5.3 2020 in UTIA, Prague, Czech Republic. In March 2020 (M22), the HW reconfigurable platform includes:

- 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 demo (tracking of colour and position of four balls)

Figure 51 presents the demonstrated addition of a new HW-accelerated Colour Ball Tracking Demo into the set of HW supported SW applications with supported reconfiguration in the run-time.



Figure 51: Demonstration of platform supporting the PL reconfiguration in Prague on 5.3.2020.

8. Conclusions

This deliverable describes the status of design time methodologies, frameworks and strategies developed by FitOptiVis project partners in M24 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.

We value progress made in Y2 in TTA-Based Co-design Environment (TCE) (see Sections 5.5 and 6.5 and table describing the tool in Section 10.1 in the appendix). TCE provides 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 and parallel program binaries. TCE supports any HW synthesizer, i.e. Vivado or Synopsis tools and it is not restricted to single technology vendor. Processor customization points include the register files, function units, supported operations, and the interconnection network. In Y2, several FitOptiVis partners have tested the TCE technology provided by TUT (CUNI, UTIA).

Another important design time resource development made in Y2 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.

As an additional project outcome to this deliverable, we have released re-usable design time methodologies and tools released in the form of publicly accessible evaluation packages and documented in publicly accessible detailed application notes, tools or design methodologies [7.12], [7.13]. This support material goes in some cases already beyond the “standard” university evaluation boards, tools and design flows. Some of the tools provide design time support for commercial modules and evaluation boards and have potential to be considered as an initial stage of commercial tools or commercial HW IP blocks.

In Y3 we will continue work on additional contributions to open source repositories for various tools and technologies forming the design time resources. Description of status of these tools and technologies in M24 is appended to this D3.2 deliverable in an easily readable form as an Appendix in Chapter 10.

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. Muttillio, 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] <https://sphinxcontrib-needs.readthedocs.io/en/latest/> - Sphinx Needs Requirements, Bug, Test Case suite.
- [4.6] <https://www.ibm.com/us-en/marketplace/rational-doors> - IBM Rational DOORS tool for requirements management.
- [4.7] <https://confluence.atlassian.com/jirakb/using-jira-for-requirements-management-193300521.html> - Using JIRA for requirements management.
- [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)
- [5.1] J. Sochman and J. Matas, "WaldBoost - learning for time constrained sequential detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 150-156 vol. 2.
- [5.2] Python package for training object detectors: <https://github.com/RomanJuranek/waldboost>
- [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.cerberio-h2020.eu/>
- [6.11] Joseph Redmon, Ali Farhadi.: YOLOv3: An Incremental Improvement. Technical report. Cornell 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
- [7.1] Jiří 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] Jiří 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] Jiří 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 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] Trenez 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] Trenez 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] Trenez 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] Trenez Electronic, "UltraTX+ Baseboard for Trenez Electronic TE080X UltraSOM+" [Online].
<https://shop.trenz-electronic.de/en/TEBF0808-04-UltraTX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261>
- [7.10] Trenez Electronic, “Carrier Board for Trenez 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] Jiří 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] Jiří 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

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.

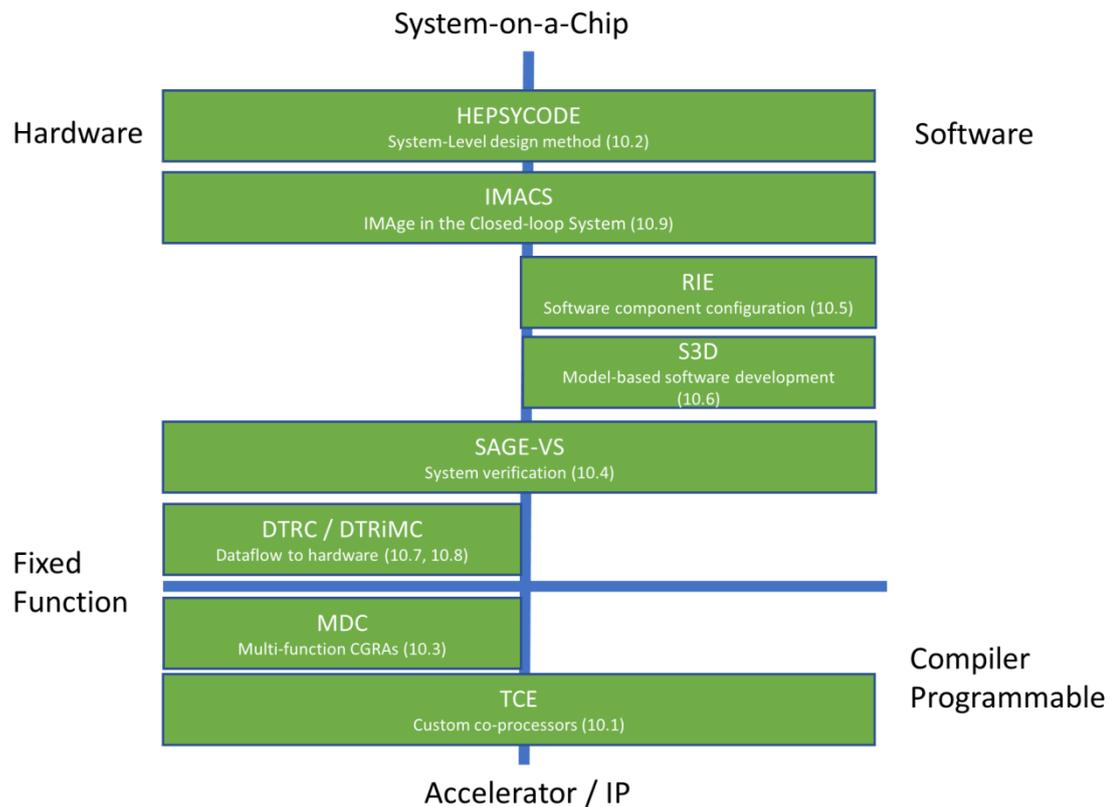


Figure 52: FitOptiVis Design Support Tools.

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 52. Table 10 present status of usage of WP3 tools by project partners in M24. The use of WP3 tools and technology in FitOptiVis demonstrators will be reported in the deliverable D3.3.

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 10 are shown in the following subsections 10.1 – 10.9.

		10_1	10_2	10_3	10_4	10_5	10_6	10_7	10_8	10_9
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
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
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
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
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
NOKIA	FI	n? y? y?	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 n n	n n y	n n y	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
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
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
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
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
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
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
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
AITEK	IT	n n n	n n y	y v v y	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 v v y	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 v v y	y v v y	n n n	n n y	n n n	n n n	n n n
UNIVAO	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

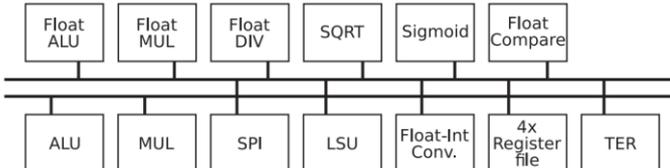
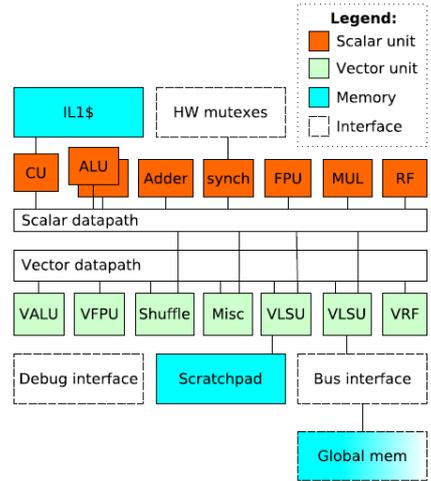
	10_1	10_2	10_3	10_4	10_5	10_6	10_7	10_8	10_9	
WP3 TOOLS and TECHNOLOGIES										
10_1	TCE: TTA-Based Co-design Environment					Explanation to the three columns:				
10_2	HEPSYCODE: Model-based.					Left We use it (y/n/?)				
10_3	MDC: Multi-Dataflow Composer					Central We plan to use it (y/n/?)				
10_4	SAGE-VS: Sage Verification Suite					Right We consider to evaluate (y/n/?)				
10_5	RIE: Re-configurable Implementation of Embedded systems									
10_6	S3D: Single Source Design Framework									
10_7	DTRC: Design Time Resource Configurator (req. Xilinx Vivado, SDSoC 2018.2)									
10_8	DTRiMC: Design Time Resource Integrator of Model Composer IPs (req. Simulink, SDSoC)									
10_9	IMACS: Image-in-the-loop control (req. Simulink, Embedded coder, OpenCV)									

Table 10: Use of WP3 tools and technologies by project partners (M24)

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

<p>Tool/Technology in a Nutshell</p>	<p>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.</p>
<p>Key Features – FitOptiVis Starting Point</p>	<p>TCE has been developed and maintained in various research projects since 2002.</p> <p>Some of the key features at the start of FitOptiVis:</p> <ul style="list-style-type: none"> • 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 <p>TRL level @ 2018 – 6/7 for the previous features, 2/3 for the new</p>
<p>Intended Users</p>	<ul style="list-style-type: none"> • 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.
<p>Benefits for the User</p>	<ul style="list-style-type: none"> • 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
<p>Tool/Technolo</p>	<p>Inputs</p> <ul style="list-style-type: none"> • C (some C++ supported), OpenCL C

<p>gy Requirements</p>	<p>Outputs</p>	<ul style="list-style-type: none"> HDL description of special function units RTL (VHDL or Verilog) along with integration/project files for different flows, one of which is AlmalF 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
<p>Target</p>	<p>Dependencies</p>	<ul style="list-style-type: none"> Any ASIC or FPGA technology HW synthesizer, i.e. Vivado or Synopsis tools. Multiple open source libraries available with liberal licenses (LLVM, wxWidgets, Boost libraries, editline)
<p>TCE Design Flow</p>		
<p>Tool/Technology Block Diagram(s)</p>	<p>Co-processors in AlmalF IP interface</p>	
<p>Example 1: Custom DSP for Binaural Speaker Localization</p>	<p>Custom DSP targeted to hearing aid devices with</p>	

	<p>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).</p>
<p>Example 2: A 5.3 pJ/op Approximate TTA VLIW Tailored for Machine Learning</p>	 <p>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.</p>
<p>Example 3: LordCore: High Performance Low Power Wide-SIMD Floating Point SDR Multicore</p>	 <p>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.</p>
<p>FitOptiVis Technological Advances</p>	<p>Expected additions:</p> <ul style="list-style-type: none"> • 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.

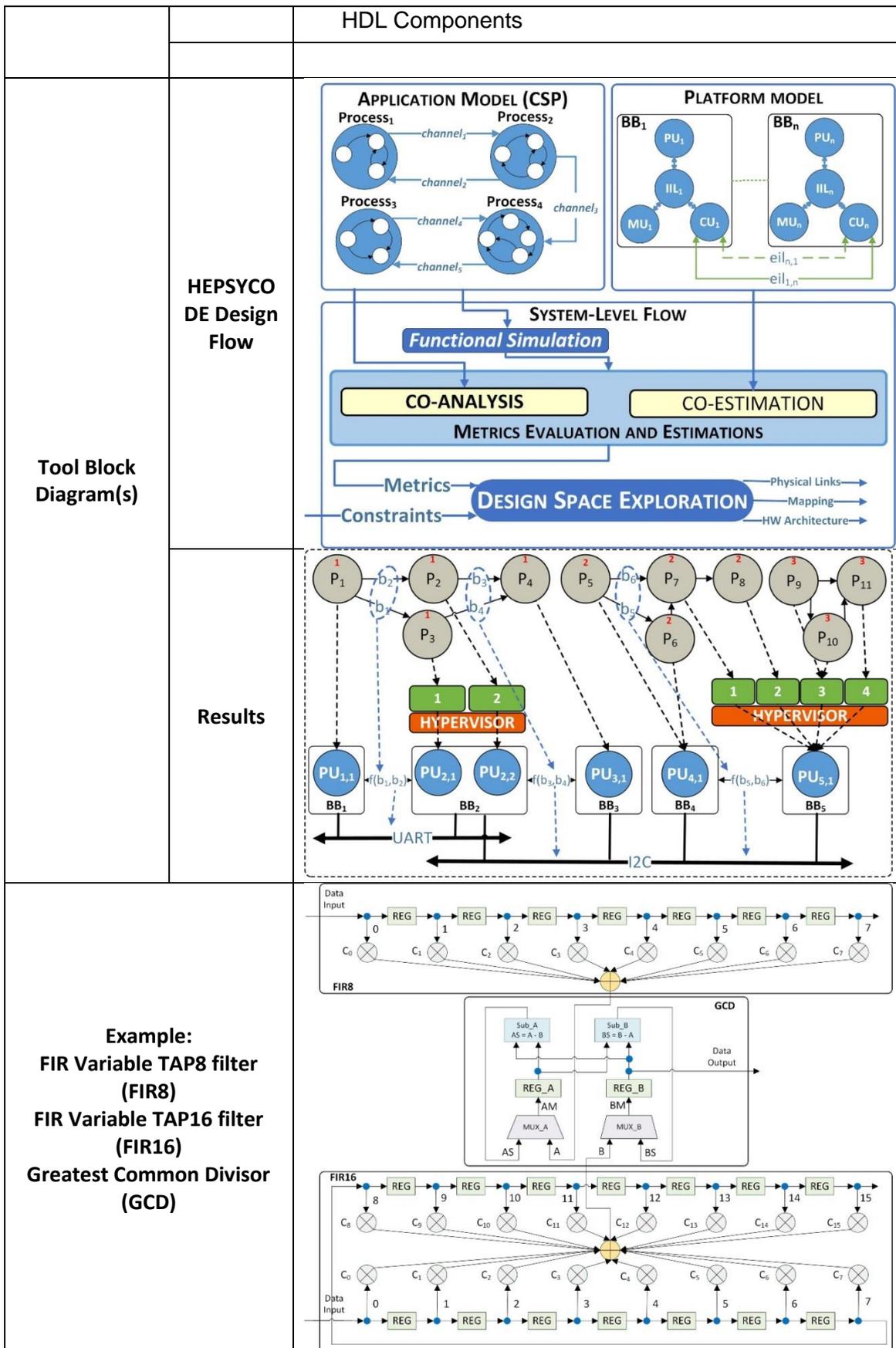
		TRL level of the new planned features @ 2021 – 4/5
Use within FitOptiVis Demonstrators	Already Planned Use	Virtual Reality Use Case: <ul style="list-style-type: none"> To produce compiler programmable co-processors for FPGA acceleration of low latency high resolution frame streaming compression using texture compression algorithms
	Potential Foreseen Links	Multi-source Streaming: <ul style="list-style-type: none"> Programmable wide-SIMD FPGA soft cores Autonomous Exploration: <ul style="list-style-type: none"> 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

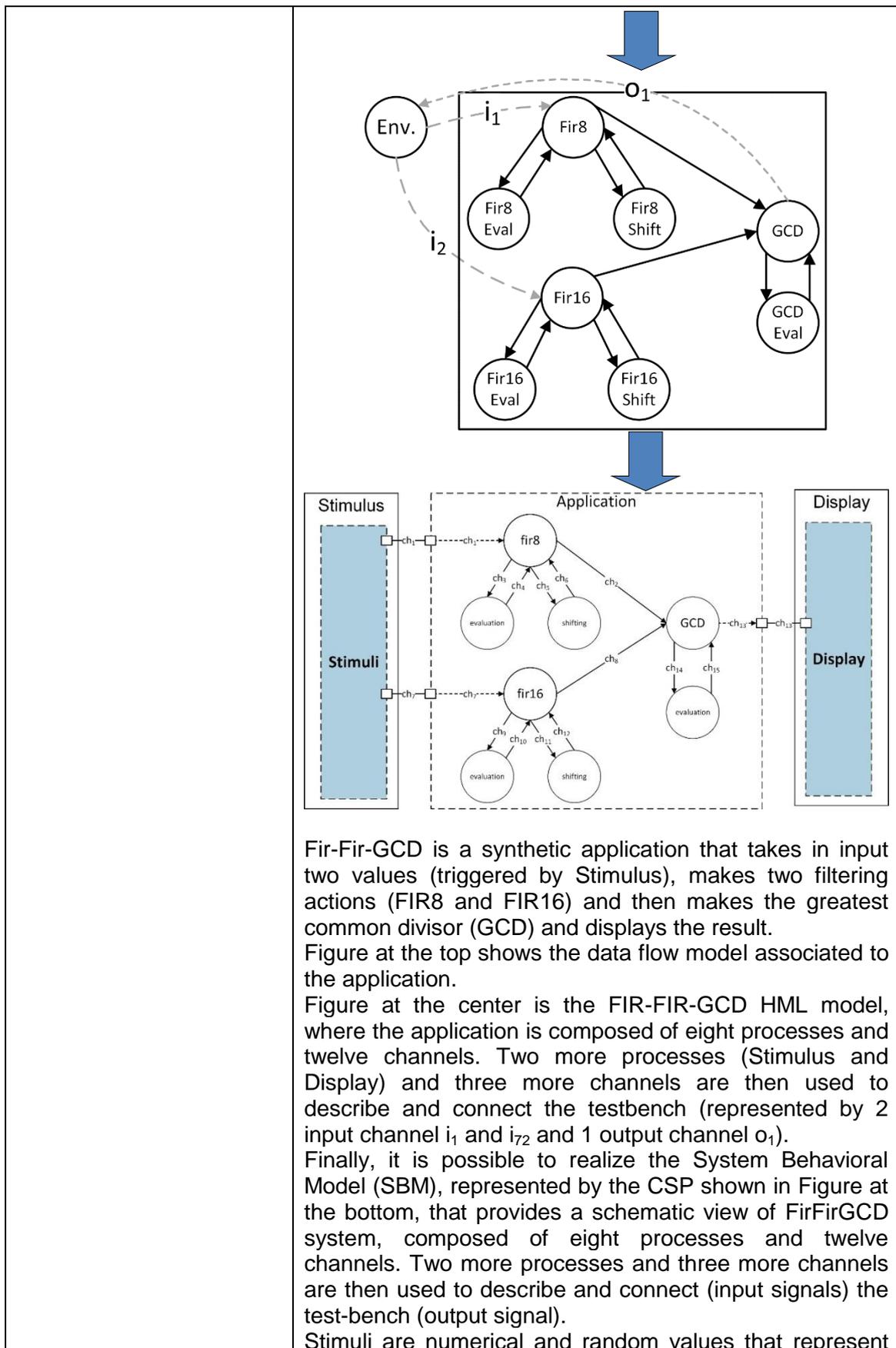
10.2. HW/SW CO-DEsign of HEterogeneous Parallel dedicated SYstems (HEPSYCODE)

Name	HW/SW CO-DE sign of HE terogeneous P arallel dedicated SY stems (HEPSYCODE)
Tool in a Nutshell	HEPSYCODE is a prototypal 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., EMC ² - Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable

	<p>real-time environments, https://www.artemis-emc2.eu/; MegaM@Rt² - MegaModelling at Runtime, https://megamart2-ecsel.eu/; AQUAS - Aggregated Quality Assurance for Systems, (https://aquas-project.eu/), while it will be improved during FitOptiVis.</p> <p>Features at the start of the project:</p> <ul style="list-style-type: none"> • HEPSYCODE defines a behavioral modeling language, named HML (<i>Hepsy Modeling Language</i>), based on the <i>Communicating Sequential Processes (CSP) Model of Computation (MoC)</i> and SystemC. By means of HML it is possible to define the <i>System Behaviour Specification (SBS)</i>, composed by the <i>System Behavior Model (SBM)</i>, a set of <i>Non Functional Constraints (NFC)</i> and a set of <i>Reference Inputs (RI)</i> 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 <i>Target Template Architecture (TTA)</i>. The final HW platform is composed of several basic HW components. These components are collected into a <i>Technologies Library (TL)</i>. 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 <i>Application Model</i> and <i>Platform Model</i>, it includes two iterative activities: (1) “<i>Search Methods</i>”, 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) “<i>Timing Co-Simulation</i>”, that considers suggested mapping/architecture items to actually check for timing constraints satisfaction.
<p>Intended Users</p>	<ul style="list-style-type: none"> • Embedded systems engineers and designers • Software developers • Hardware architects

		<ul style="list-style-type: none"> • EDA industries
Benefits for the User		<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • HW/SW final architecture: <ul style="list-style-type: none"> ○ 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • EMF technologies • SystemC Library • Embedded Linux Distributions (Petalinux, Gaisler Buildroot) • HW synthesizer, i.e. Vivado • (optional) High Level Synthesis tool to generate the





		<p>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).</p>
FitOptiVis Technological Advances		<p>Expected additions:</p> <ul style="list-style-type: none"> • 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
Use within FitOptiVis Demonstrators	Already Planned Use	<ul style="list-style-type: none"> • Under Evaluation
	Potential Foreseen Links	
Open-Source		<p>Git repository: https://bitbucket.org/vittorianomuttillio87/tool-hepsycode/src/master/ Official website: http://www.hepsycode.com</p>
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		<p>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.</p> <p>Features at the start of FitOptiVis:</p> <ul style="list-style-type: none"> • 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 <p>TRL level @ 2018 – 3/4</p>
Intended Users		<ul style="list-style-type: none"> • 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		<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • (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	<ul style="list-style-type: none"> ASIC (baseline, profiling, and power management) FPGA (baseline, power management, accelerator deployment)
	Dependencies	<ul style="list-style-type: none"> 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		<p>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.</p>
FitOptiVis Technological Advances		<p>Expected additions:</p> <ul style="list-style-type: none"> Multi-Level monitoring support based on AIPHS 2.0 OpenCL APIs extension <p>TRL level @ 2021 – 4/5</p>
Use within FitOptiVis Demonstrators	Already Planned Use	<p>Water Supply Use Case:</p> <ul style="list-style-type: none"> build, manage and monitor application specific HW accelerators
	Potential Foreseen Links	
Open-Source		<p>Git access to be provided soon, executable and tutorials already available: http://sites.unica.it/rpct/download/</p>
Licence Type		<p>https://opensource.org/licenses/BSD-3-Clause</p>
Commercial license		N/A

10.4. The SAGE Verification Suite (SAGE-VS)

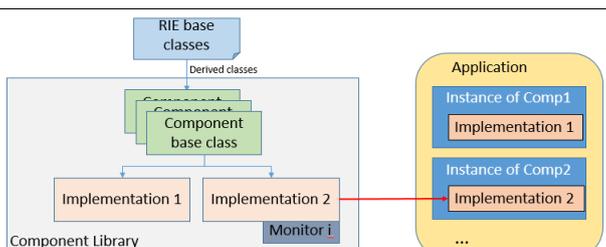
Name	The SAGE Verification Suite (SAGE-VS)	
Tool in a Nutshell	The SAGE Verification Suite (SAGE-VS) is a set of SW tools aimed to accomplish different formal verification tasks at design time.	
Key Features – FitOptiVis Starting Point	<p>The SAGE-VS has been designed and developed (from TRL 0/1) in the context of the CERBERO H2020 project (https://cerbero-h2020.eu).</p> <p>At the start of FitOptiVis, it was composed of the following tools:</p> <ul style="list-style-type: none"> • ReqV: a tool for formal consistency checking of requirements. • Hydra: a domain-independent tool for Goal-Oriented control of Cyber-Physical Systems. <p>TRL level @ 2018 – 3/4</p>	
Intended Users	<ul style="list-style-type: none"> • [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	<ul style="list-style-type: none"> • [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 minimal set 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	<p>Requires a hybrid model of the system:</p> <ul style="list-style-type: none"> • definition of the state of a system • definitions of the system's capabilities <ul style="list-style-type: none"> ○ available discrete actions and their effect on the system and its environment ○ operating limits of the controller • safety limits <p>Specification of a target problem: initial state, goal state of the system and invariants that should hold.</p>
	Outputs	<p>A yes/no answer on whether the system can be used to achieve the tested use case.</p> <p>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.</p>
Examples		<p>[ReqV] A requirement engineer has to start the requirements definition of a new system. She opens the browser, logs in into ReqV and create 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 run the inconsistency explanation task, and after few minutes ReqV returns a list of few requirements that are conflicting. The engineer inspects those requirements and fix 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 enter in ReqV again and insert 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 update the requirements in ReqV accordingly, and this time the verification process returns a positive answer.</p> <p>[Hydra]: Let us consider a robotic manipulator. The mobile manipulator must be operated in a constrained</p>

		<p>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 to verify 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.</p> <p>[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 fails, 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 fix it, then she repeats the test execution. This time all tests are successful, so the users can finally deploy the system.</p>
FitOptiVis Technological Advances		<p>Expected additions:</p> <ul style="list-style-type: none"> 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. <p>TRL level @ 2021 – 4/5</p>
Use within FitOptiVis Demonstrators	Already Planned Use	<p>Water Supply Use Case:</p> <ul style="list-style-type: none"> build, manage and monitor application specific HW accelerators
	Potential Foreseen Links	
Open-Source		<p>https://gitlab.sagelab.it/sage/ReqV https://gitlab.sagelab.it/sage/ReqT</p>

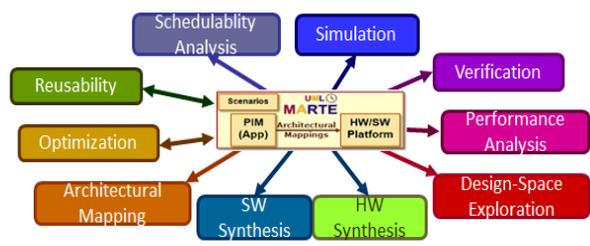
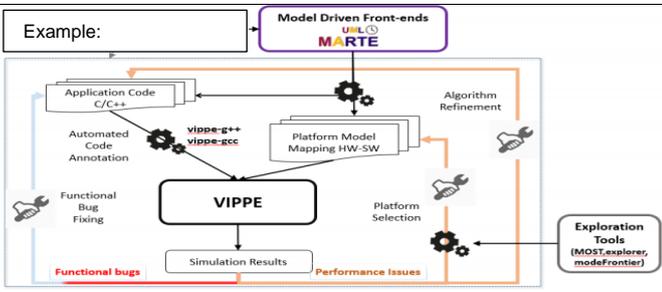
Licence Type	LGPL
Commercial license	N/A

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	<ul style="list-style-type: none"> Component-based embedded system developers that require software component reconfiguration at runtime. 	
Benefits for the User	<ul style="list-style-type: none"> Simplify and standardize component-based development. Integrate software reconfiguration Provide a common framework to access different monitoring strategies. By default, the library supports lttnng 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	<ul style="list-style-type: none"> A system description that uses RIE-based component.
	Outputs	<ul style="list-style-type: none"> A system implementation that can be reconfigurable and traced at runtime.
	Target	<ul style="list-style-type: none"> Networked embedded systems.
	Dependencies	<ul style="list-style-type: none"> C++11 compiler. For edge computing, protocol buffer and grpc. For linux event monitoring, lttnng.
	<ul style="list-style-type: none">
Tool Block Diagram(s)	MDC Baseline Flow	 <p>The diagram illustrates the RIE architecture. On the left, a 'Component Library' contains 'RIE base classes' (blue box) which are 'Derived classes' (green boxes) leading to a 'Component base class' (green box). This base class has two 'Implementation' boxes (Implementation 1 and Implementation 2) and a 'Monitor i' box (blue box). On the right, an 'Application' box (yellow rounded rectangle) contains 'Instance of Comp1' (blue box), 'Implementation 1' (orange box), 'Instance of Comp2' (blue box), 'Implementation 2' (orange box), and 'Monitor i' (blue box). A red arrow points from 'Implementation 2' in the Component Library to 'Implementation 2' in the Application.</p>
	MDC Accelerator	

<p style="text-align: center;">Example: FIR Variable TAP filter</p>	<pre> //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); </pre>				
<p style="text-align: center;">FitOptiVis Technological Advances</p>	<ul style="list-style-type: none"> • Simplify component-based development • Support data-flow and service-oriented architectures. • Provide software re-configuration capability • Support different types of monitor implementation. 				
<p style="text-align: center;">Use within FitOptiVis Demonstrators</p>	<table border="1" style="width: 100%;"> <tr> <td style="text-align: center;">Already Planned Use</td> <td data-bbox="588 1232 1386 1344"> <ul style="list-style-type: none"> • RIE is used to implement the autonomous exploration use case. </td> </tr> <tr> <td style="text-align: center;">Potential Foreseen Links</td> <td data-bbox="588 1344 1386 1451"> <p>TBD. Available 1Q-2020</p> </td> </tr> </table>	Already Planned Use	<ul style="list-style-type: none"> • RIE is used to implement the autonomous exploration use case. 	Potential Foreseen Links	<p>TBD. Available 1Q-2020</p>
Already Planned Use	<ul style="list-style-type: none"> • RIE is used to implement the autonomous exploration use case. 				
Potential Foreseen Links	<p>TBD. Available 1Q-2020</p>				
<p style="text-align: center;">Open-Source</p>	<p>Yes</p>				
<p style="text-align: center;">Licence Type</p>	<p>GPL after publication</p>				
<p style="text-align: center;">Commercial license</p>	<p>Yes</p>				

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		<ul style="list-style-type: none"> • HW/SW system development • Embedded system application designers
Benefits for the User		<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • UML/MARTE models
	Outputs	<ul style="list-style-type: none"> • Performance estimation of different implementations • C++ implementation templates
	Target	<ul style="list-style-type: none"> • Edge and cloud computing
	Dependencies	<ul style="list-style-type: none"> • Eclipse, Papyrus, Ivm.
	...	<ul style="list-style-type: none"> •
Tool Block Diagram(s)	MDC Baseline Flow	
	MDC Accelerator	
Example: FIR Variable TAP filter		
FitOptiVis Technological Advances		

Use within FitOptiVis Demonstrators	Already Planned Use	<ul style="list-style-type: none"> Use in Autonomous Exploration use case
	Potential Foreseen Links	
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		<p>DTRC 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:</p> <ul style="list-style-type: none"> Support for Xilinx SDSoC 2015.4 standalone Zynq modules without OS with Python 1300 Video sensor or Full HD HDMI Video I/O. <p>TRL level @ 2017 – 4/5.</p>
Intended Users		<ul style="list-style-type: none"> 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		<ul style="list-style-type: none"> design automation of configuration of Debian OS with SDSoC support integration of Full HD video input and Video output
Tool/Technology Requirements	Inputs	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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

		<p>DMA and Scatter Gather (SG) DMA data transfers to/from HW accelerators.</p> <ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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 UltraTX+ Baseboard for Trenz Electronic TE080X UltraSOM+ https://shop.trenz-electronic.de/en/TEBF0808-04A-UltraTX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261
	Dependencies	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> On Win7/Win10/Ubuntu 16.04: Compile HW and export hdf file. On Ubuntu 16.04: Configure and compile Petalinux and Debian.

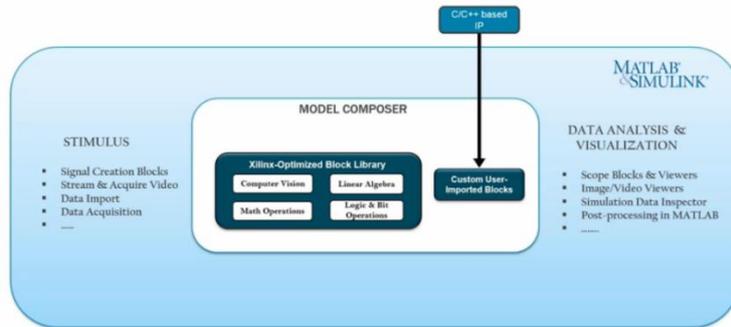
		<ul style="list-style-type: none"> On Win7/Win10/Ubuntu 16.04: Compile in SDSoC 2018.2 HW accelerators from C/C++ functions to HW. Run on supported boards.
	<p>DTRC Packages and Applications notes</p>	<p>[7.1] Design Time and Run Time Resources for the ZynqBerry Board TE0726-03M with SDSoC 2018.2 Support</p> <p>[7.2] Design Time and Run Time Resources for Zynq Ultrascale+ TE0820-03-4EV-1E with SDSoC 2018.2 Support</p> <p>[7.3] Design Time and Run Time Resources for Zynq Ultrascale+ TE0808-04-15EG-1EE with SDSoC 2018.2 Support</p> <p>TRL level @ 8.2019 – 5/6</p>
FitOptiVis Technological Advances		<p>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</p>
Use within FitOptiVis		<p>DTRC 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</p>
Use within FitOptiVis Demonstrators	Potential Foreseen Links	<p>Robotic Use Case: Build HW accelerators on the ZynqBerry board</p>
	Access	<p>http://sp.utia.cz/index.php?ids=projects/fitoptivis</p>
Licence Type		<p>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/ .</p>

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

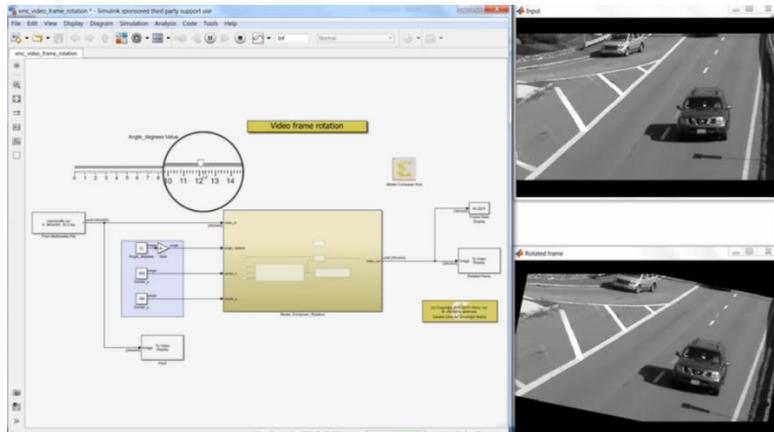
Name	Design Time Resource Integrator of Model Composer IPs (DTRiMC) Technology
Technology in a Nutshell	<p>DTRiMC technology [7.12], [7.13] serves for FitOptiVis system integration of IPs designed, modelled and validated in Xilinx Model Composer (MC) and Xilinx System Generator for DSP (SG for DSP).</p> <p>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.</p> <p>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+).</p> <p>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.</p>
Key Features – FitOptiVis Starting Point	<p>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</p> <p>ECSEL JU project ALMARVI. Features at the start of FitOptiVis:</p> <ul style="list-style-type: none"> • Support for Xilinx SDSoc 2015.4 standalone Zynq modules without OS with Python 1300 Video sensor or Full HD HDMI Video I/O. <p>TRL level @ 2017 – 4/5.</p>
Intended Users	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • 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.

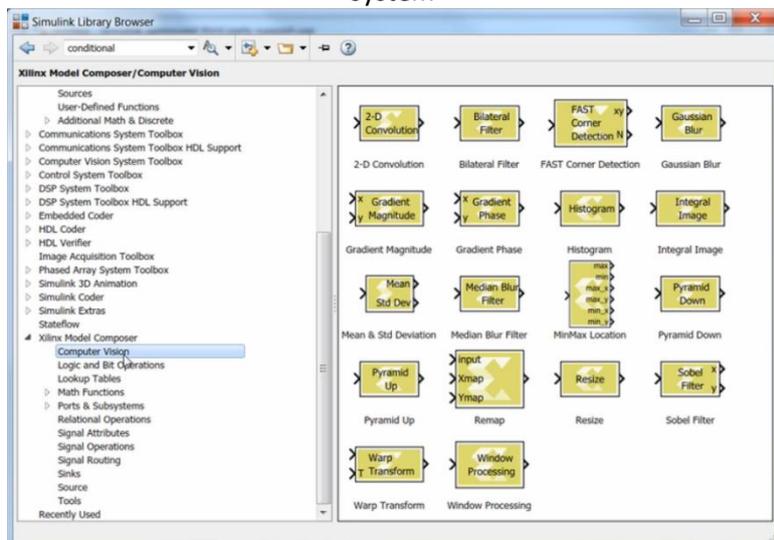
Key features of the supported Xilinx Model Composer framework:



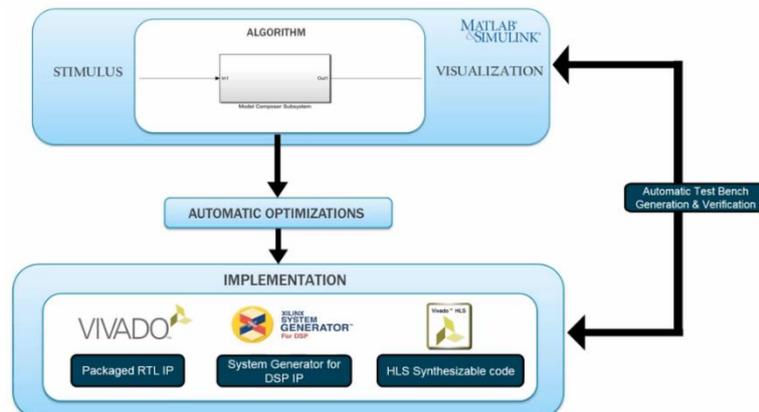
MODEL COMPOSER supports fast modelling of blocks written in C



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>

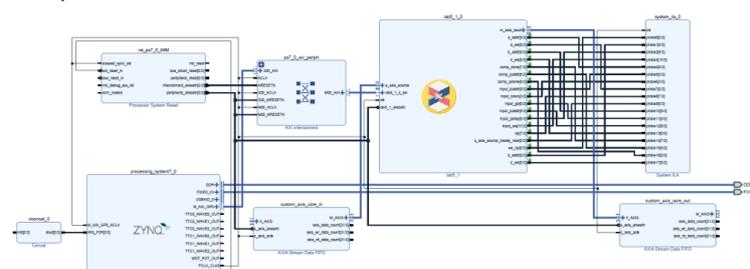
<https://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html>

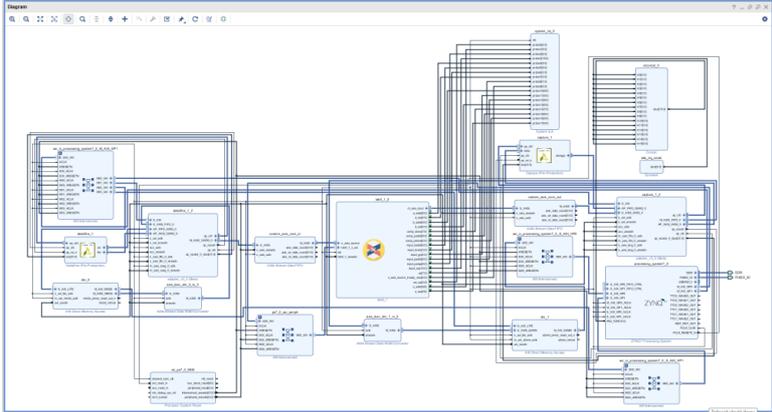
MC supports model based design, simulation and HW IP generation. It targets Xilinx FPGAs/SoCs via generated IP cores for Vivado flow.

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

	<p>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.</p> <p>MC supports video I/O from/to files, visualisation of video with relatively high FPS.</p>	
<p>Tool/Technology Requirements</p>	<p>Inputs</p>	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> ○ 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.  <p>Input to the DTRiMC tool: Exported MC IP block in base Zynq system.</p>
	<p>Outputs</p>	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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

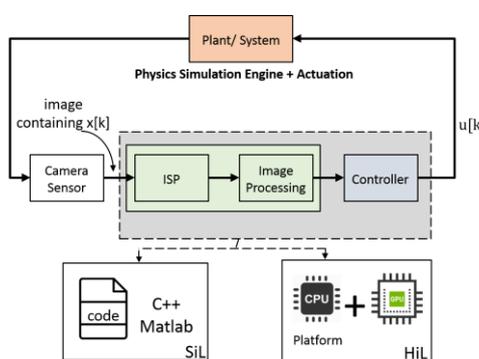
		<p>app. Code for Debian.</p> <ul style="list-style-type: none"> • Support for Arrowhead framework 4.0 compatible C/C++ SW clients for authentication and management of Ethernet access rights. • Support for FiVis compatible C++ clients for Ethernet data transfer and visualisation via FiVis server. FiVis server generates graphical visualisation pages accessible from standard www browsers.  <p>Output from DTRiMC: MC IP block integrated with DMA I/O to Debian</p>
	<p>Targets</p>	<ul style="list-style-type: none"> • 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
	<p>Dependencies commercial</p>	<ul style="list-style-type: none"> • Matlab, Version 9.3 (R2017b) MathWorks (commercial tool) • Simulink Version 9.0 (R2017b) MathWorks (commercial)

	<p>al tools</p> <ul style="list-style-type: none"> 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) 	
	<p>Dependencies Open source tools</p> <ul style="list-style-type: none"> 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	<p>DTRiMC extends DRTC technology</p> <p>DTRiMC [7.12], [7.13] extends FitOptiVis DRTC technology [7.1], [7.2], [7.3].</p>	
	<p>Base DRTC Packages and Application notes</p> <p>Design Time and Run Time Resources for the ZynqBerry Board TE0726-03M with SDSoC 2018.2 Support Design Time and Run Time Resources for Zynq Ultrascale+ TE0820-03-4EV-1E with SDSoC 2018.2 Support Design Time and Run Time Resources for Zynq Ultrascale+ TE0808-04-15EG-1EE with SDSoC 2018.2 Support</p>	
FitOptiVis Technological Advances		<p>DTRiMC technology supports export of Model Composer IP (as SG for DSP) IP to Zynq and Zynq Ultrascale+ SoCs .</p> <p>DTRiMC technology generates DMA HW/SW data movers for of Model Composer IPs.</p>
Use within FitOptiVis		<p>DTRiMC technology is released for public access [7.12], [7.13] for Zynq & Zynq Ultrascale+. It is used by UTIA.</p>
Use within FitOptiVis Demonstrators	Links	<p>DTRiMC technology release [7.12] for the entry level ZynqBerry board:</p> <p>http://sp.utia.cz/index.php?ids=results&id=te0726_fp01x8 FP01x8 Accelerator on TE0726-03M http://sp.utia.cz/results/te0726_fp01x8/AppNote-FitOptiVis-te0726_fp01x8_short.pdf</p> <p>DTRiMC technology release [7.13] for the Zynq Ultrascale+:</p> <p>http://sp.utia.cz/index.php?ids=results&id=te0820_fp03x8x2s</p>

		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
	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 would be further developed in FitOptiVis with specific focus on the FitOptiVis objective.
Intended Users	Embedded and cyber-physical systems developers
Benefits for the User	<ul style="list-style-type: none"> • 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
	<ul style="list-style-type: none"> • Details of the image-in-the-loop applications; e.g., system model, scenarios of interest and so on;

		<ul style="list-style-type: none"> • Camera and other sensor specifications; • Platform details; e.g, processors, memory; • Performance and quality requirements.
	Outputs	<ul style="list-style-type: none"> • 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		
Example		
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	Already Planned Use	The results of design-time optimization in WP3 will be partially implemented;
	Potential Foreseen Links	Reconfiguration solution of WP4 will also be a part of it.
Open-Source	Yes	
Licence Type	Apache2.0	
Commercial license	N/A	