



**ECSEL2017-1-737451**

## **FitOptiVis**

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

**Deliverable: D3.1 Design time, optimisation,  
deployment and programming strategies V1**

Due date of deliverable: (31-05-2019)


Actual submission date: (31-05-2019)

Start date of Project: 01 June 2018

Duration: 36 months

Responsible: University of Cantabria (UC)

Revision: Accepted

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

## 1 DOCUMENT INFO

### Author

Author	Company	E-mail
Pablo Sanchez	UC	<a href="mailto:sanchez@teisa.unican.es">sanchez@teisa.unican.es</a>
Jiri Kadlec	UTIA	<a href="mailto:kadlec@utia.cas.cz">kadlec@utia.cas.cz</a>
Peka Jaakelainen	TUT	<a href="mailto:pekka.jaaskelainen@tuni.fi">pekka.jaaskelainen@tuni.fi</a>
Carlo Sau	UNICA	<a href="mailto:carlo.sau@diee.unica.it">carlo.sau@diee.unica.it</a>
Dip Goswami	TUE	<a href="mailto:D.Goswami@tue.nl">D.Goswami@tue.nl</a>
Roman Juranek	BUT	<a href="mailto:jjuranek@fit.vutbr.cz">jjuranek@fit.vutbr.cz</a>
Luis Medina	7SOLS	<a href="mailto:luis.medina@sevensols.com">luis.medina@sevensols.com</a>
Francesca Palumbo	UNISS	<a href="mailto:fpalumbo@uniss.it">fpalumbo@uniss.it</a>
Luigi Pomante	UNIVAQ	<a href="mailto:luigi.pomante@univaq.it">luigi.pomante@univaq.it</a>
Tero Sääntti	UTU	<a href="mailto:teansa@utu.fi">teansa@utu.fi</a>
Massimo Massa	AITEK	<a href="mailto:mmassa@aittek.it">mmassa@aittek.it</a>
Tomas Bures	CUNI	<a href="mailto:buress@d3s.mff.cuni.cz">buress@d3s.mff.cuni.cz</a>
Jukka Saarinen	Nokia	<a href="mailto:jukka.saarinen@nokia.com">jukka.saarinen@nokia.com</a>

### Document history

Document version #	Date	Change
V0.1		Starting version, template
V0.2	25/01/19	New template
V0.3	1/02/19	Contr. from UTIA, UNISS, UNICA, TUT, BUT
V0.4	4/02/19	Contr. from UC, UTIA, UNISS, UNICA, TUT, UNIVAQ, BUT
V0.5	8/04/19	Integrated 7SOLS and HIB
V0.6	8/04/19	Contr. from UC, UTIA, UNISS, UNICA AITEK, TUT, UNIVAQ, BUT, UTU, UE, 7SOLS
V0.7	15/04/19	Integrated Chapter 7
V0.8	17/05/19	Executive summary, Introduction, Conclusion.
V0.9	17/05/19	Formatting, ready for internal review.
V0.11	28/05/19	Submitted for internal review
V0.12	29/05/19	Modification based on the internal review
Sign off	30/05/19	Signed off version (for approval to PMT members)
V1.0	31/05/19	Approved Version to be submitted to ECSEL office

### Document data

<b>Keywords</b>	
<b>Editor Address data</b>	Name: Pablo Sanchez Partner: UC – University of Cantabria Address: ETSIIT. University of Cantabria. Spain Phone: +34 619045321

### Distribution list

Date	Issue	E-mailer
30/05/19	Final	fitoptivis-wp3@lists.utu.fi



---

## Table of Contents

<b>1</b>	<b>DOCUMENT INFO</b> .....	<b>2</b>
<b>2</b>	<b>EXECUTIVE SUMMARY</b> .....	<b>7</b>
<b>3</b>	<b>INTRODUCTION</b> .....	<b>8</b>
<b>3.1</b>	<b>Structure of WP3</b> .....	<b>8</b>
<b>4</b>	<b>MODEL-DRIVEN ENGINEERING TECHNIQUES FOR ENERGY, PERFORMANCE AND OTHER QUALITIES</b> .....	<b>9</b>
<b>4.1</b>	<b>The S3D Modelling methodology for real-time video processing systems</b> .....	<b>10</b>
<b>4.2</b>	<b>Design Space Exploration for Reconfigurability</b> .....	<b>14</b>
<b>4.3</b>	<b>The SAGE Verification Suite</b> .....	<b>15</b>
<b>4.4</b>	<b>Dynamic performance tracking of image-based applications using control theoretic approaches</b> .....	<b>16</b>
4.4.1	MODELLING APPLICATIONS AS A MAX-PLUS LINEAR SYSTEM	16
4.4.2	CONTROL-THEORETIC APPROACH .....	18
4.4.3	MODELLING DYNAMISM AS A SWITCHED MAX-PLUS LINEAR SYSTEM	18
<b>4.5</b>	<b>Modelling of real-time video processing systems with limited precision</b> .....	<b>19</b>
<b>4.6</b>	<b>Design time Support for High Level Tool Chains</b> .....	<b>20</b>
<b>4.7</b>	<b>High-level abstract component model and DSL</b> .....	<b>20</b>
<b>5</b>	<b>PROGRAMMING AND PARALLELIZATION SUPPORT</b> .....	<b>21</b>
<b>5.1</b>	<b>Static resource allocation and runtime scheduling</b> .....	<b>21</b>
<b>5.2</b>	<b>OpenMP for real-time video systems</b> .....	<b>22</b>
<b>5.3</b>	<b>Design Time Support for C/C++ Compilers and OpenCV Algorithmic Libraries</b> .....	<b>23</b>
<b>5.4</b>	<b>TTA-based Co-Design Environment (TCE)</b> .....	<b>23</b>
5.4.1	TCE: SUPPORT FOR 64-BIT POINTERS AND INTEGERS .....	23
5.4.2	TCE: LOOP SCHEDULING SUPPORT .....	24
5.4.3	TCE: LOOP BUFFER AND INSTRUCTION REGISTER FILE SUPPORT	26
<b>5.5</b>	<b>Block Copier: A Programmable Block Transfer Unit</b> .....	<b>28</b>
<b>5.6</b>	<b>Deterministic timing in distributed systems and latency control with Time Sensitive Networks (TSN)</b> .....	<b>29</b>
<b>6</b>	<b>ACCELERATION SUPPORT</b> .....	<b>31</b>
<b>6.1</b>	<b>OpenMp for HW accelerator</b> .....	<b>31</b>
<b>6.2</b>	<b>HW Accelerators Generated by the Xilinx SDSoC System Level Compiler</b> .....	<b>32</b>

<b>6.3</b>	<b>The Multi-Dataflow Composer (MDC) tool: a dataflow-to-accelerator design suite.....</b>	<b>32</b>
<b>6.4</b>	<b>TTA-Based Customized Soft Core Accelerators.....</b>	<b>34</b>
6.4.1	TCE: AUTOEXPLORER.....	35
<b>6.5</b>	<b>Acceleration of individual algorithms in combination of FPGA and CPU.....</b>	<b>38</b>
6.5.1	OBJECT DETECTION ON FPGA USING WALDBOOST ALGORITHM.....	38
6.5.2	HDR IMAGE ACQUISITION.....	38
<b>6.6</b>	<b>HDR Merging.....</b>	<b>39</b>
<b>6.7</b>	<b>Tonemapping.....</b>	<b>39</b>
<b>6.8</b>	<b>Convolutional HW accelerator.....</b>	<b>40</b>
<b>6.9</b>	<b>NEURAghe: a flexible and parameterized CNN accelerator 41</b>	
<b>6.10</b>	<b>Nokia acceleration work for PCC demo case.....</b>	<b>42</b>
<b>7</b>	<b>DESIGN TIME SUPPORT FOR METHODOLOGIES AND TOOLS .....</b>	<b>44</b>
<b>7.1</b>	<b>Board support package generation project for SDSoC 2018.2 compiler for ZynqBerry board TE0726.....</b>	<b>45</b>
7.1.1	DESCRIPTION.....	45
7.1.2	INPUTS:.....	46
7.1.3	OUTPUTS:.....	46
7.1.4	USAGE WITHIN FITOPTIVIS.....	46
<b>7.2</b>	<b>Design time support for SDSoC 2018.2 compiler for ZynqBerry board TE0726.....</b>	<b>47</b>
7.2.1	DESCRIPTION.....	47
7.2.2	INPUTS:.....	47
7.2.3	OUTPUTS:.....	47
7.2.4	USAGE WITHIN FITOPTIVIS.....	47
<b>7.3</b>	<b>Platform generator for SDSoC 2018.2 compiler on Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O 48</b>	
7.3.1	DESCRIPTION.....	48
7.3.2	INPUTS:.....	49
7.3.3	OUTPUTS:.....	49
7.3.4	USAGE WITHIN FITOPTIVIS.....	49
<b>7.4</b>	<b>Design time support for SDSoC 2018.2 compiler on Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O 49</b>	
7.4.1	DESCRIPTION.....	49
7.4.2	INPUTS:.....	50
7.4.3	OUTPUTS:.....	50



---

7.4.4	USAGE WITHIN FITOPTIVIS.....	50
<b>7.5</b>	<b>Platform generator for SDSoC 2018.2 compiler on Zynq Ultrascale+ module TE0808-15EG on TEBF0808 carrier with full HD HDMI Video I/O.....</b>	<b>52</b>
7.5.1	DESCRIPTION.....	52
7.5.2	INPUTS:.....	53
7.5.3	OUTPUTS:.....	53
7.5.4	USAGE WITHIN FITOPTIVIS.....	53
<b>7.6</b>	<b>Design time support for SDSoC 2018.2 compiler on Zynq Ultrascale+ module TE0808-15EG on TEBF0808 carrier with full HD HDMI Video I/O.....</b>	<b>53</b>
7.6.1	DESCRIPTION.....	53
7.6.2	INPUTS:.....	54
7.6.3	OUTPUTS:.....	54
7.6.4	USAGE WITHIN FITOPTIVIS.....	54
<b>8</b>	<b>CONCLUSIONS.....</b>	<b>56</b>
<b>9</b>	<b>REFERENCES.....</b>	<b>57</b>

## Table of Figures

Figure 1: TE Single Source System Design Framework (SD3).....	11
Figure 2 - Application View.....	12
Figure 3: Example of Component interfaces.....	13
Figure 4: Example of Non-Functional requirements.....	13
Figure 5: Reference ESL HW/SW co-design flow.....	14
Figure 6: Embedded multi-core platform with reconfiguration mechanism.....	16
Figure 7: Synthetic SDF model.....	17
Figure 8: Inter-arrival time.....	19
Figure 9: User interface with visualization of profiling.....	22
Figure 10: Extension of OpenMP offloading capabilities.....	23
Figure 11: High-level example of software pipelining.....	25
Figure 12: H Architecture of the block copier ASIP.....	29
Figure 13: OpenMP code offloading.....	31
Figure 14: Simplified TCE Exploration process of AutoExplorer.....	35
Figure 15: Unoptimized architecture (left), final best possible architecture (right). .....	37
Figure 16: HDR acquisition pipeline.....	38
Figure 17: The FPGA implementation of Durand tonemapping.....	40
Figure 18: MPEG V-PCC encoder block diagram.....	43
Figure 19: Two ZynqBerry boards and RPi3.....	46
Figure 20: TE0820-03-4EV-1E on TE0701-06, Imageon HDMI I/O FMC card.....	48
Figure 21: HW accelerated LK DOF input/output Full HD HDMI 60fps.....	51
Figure 22: TE0808-03-15EG-1EE on TEBF0808-04 and Imageon HDMI I/O FMC.....	52
Figure 23: HW accelerated LK DOF, Full HD HDMI 60 fps.....	55

## Table of Tables

Table 1: Overview and Comparison of Model-driven engineering techniques.....	9
Table 2: V-PCC decoding performance overview.....	43
Table 3 : V-PCC AR point rendering capabilities.....	43
Table 4: Progress made in FitOptiVis in WP3 in M12.....	44
Table 5: Performance of HW accelerated LK DOF and the load of Arm A53.....	51
Table 6: Performance of HW accelerated LK FOF and load Arm A53 processors. ....	55

## 2 Executive summary

Deliverable D3.1, “Design time, optimization, deployment and programming strategies V1”, focuses on concrete design flows, tools and design time support packages used, developed and/or extended by WP3 partners. The D3.1 report is prepared in M12 and collects results from tasks T3.1, “Model-driven engineering techniques for energy, performance and other qualities”, T3.2, “Programming and parallelization support”, and T3.3, “Accelerator support”, after the first year of the project.

Chapter 7 of the report is dedicated to design time methodologies and tools released after the Y1 of the project in form of publicly accessible evaluation packages and documented in publicly accessible detailed application notes. 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 developer outside of the project.

Character of the deliverable is public and partner UC was main editor of the deliverable.

## 3 Introduction

This D3.1 report, “Design time, optimisation, deployment and programming strategies V1”, presents design methodologies, frameworks and design time support packages developed and/or improved by WP3 partners. The report integrates results of tasks T3.1, T3.2 and T3.3 after the first year of the project (M12). It is a public deliverable and UC partner is acting as main editor of the deliverable.

### 3.1 Structure of WP3

Work of partners involved in WP3 is organised in 3 tasks: T3.1, T3.2 and T3.3.

**Task 3.1** – Model-driven engineering techniques for energy, performance and other qualities.

The objective of the first period was to define, discuss, clarify and document common approaches to the design time resources covering model-driven engineering techniques for energy, performance and other qualities. These activities are mainly described in Chapter 4. Activities of partners in Task 3.1 also form an initial WP3 link to the WP2 developments.

**Task 3.2** – Programming and parallelization support.

Contributions of partners related to task T3.2 are mainly described in Chapter 5. It 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 partners in Task 3.2 also include links to the WP5 work package.

**Task 3.3** – Accelerator support.

Contributions of partners related to task T3.3 are mainly described in Chapter 6. It describes design time resources related specifically to the developed HW accelerators. Activities of partners in Task 3.3 also include relations to the WP5 work package.

Chapter 7 is dedicated to design time methodologies and tools that have been developed in Tasks 3.1 and 3.3, and they are released after the Y1 of the project in form of publicly accessible evaluation packages and documented in publicly accessible detailed application notes. These released resources serve as concrete WP3 support for FitOptiVis project partners, and also serve as support material which can be used by other developer outside of the project.



## 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 first 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 D3.1 initial 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.

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

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.	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. ATG: a tool for requirements-based test suites generation.
Dynamic performance tracking (control theoretic)	4.4	Depending on the application requirements, the optimization algorithms find the best configuration (mapping, scheduling and	It is based on Synchronous Dataflow (SDF) graph which can be analysed to answer performance related questions such as the minimum

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

#### 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. 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.

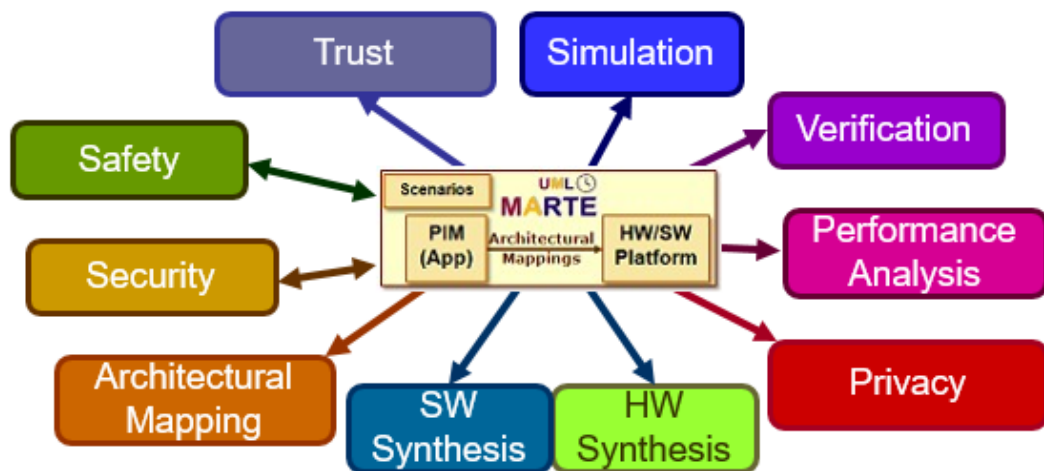


Figure 1: TE Single Source System Design Framework (SD3)

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*.

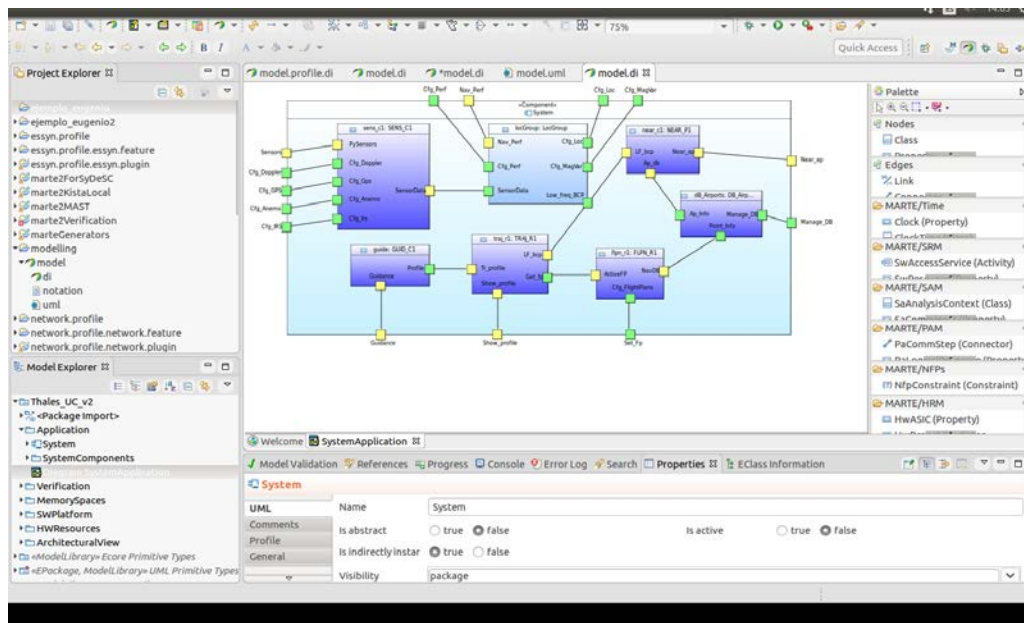


Figure 2 - Application View

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 components includes the services (functions) or signals (values) that they provide and/or require. Thus, the required interface of a structured component lists all the services/signals that the component requires from other components. The provided interface lists all the services/signals that the component offers to other components. *Figure 3* shows an example of component that presents all the interface services. The packages integrate components that are re-used in different applications. Every component has at least an implementation (or behaviour) and a specific verification test case.

The component and its interfaces could include non-functional parameters such as latency or required memory size (*Figure 4*). These non-functional parameters could be a function of platform specific parameters. For example, the component latency could be a function of the frequency on the processor in which the component executes.

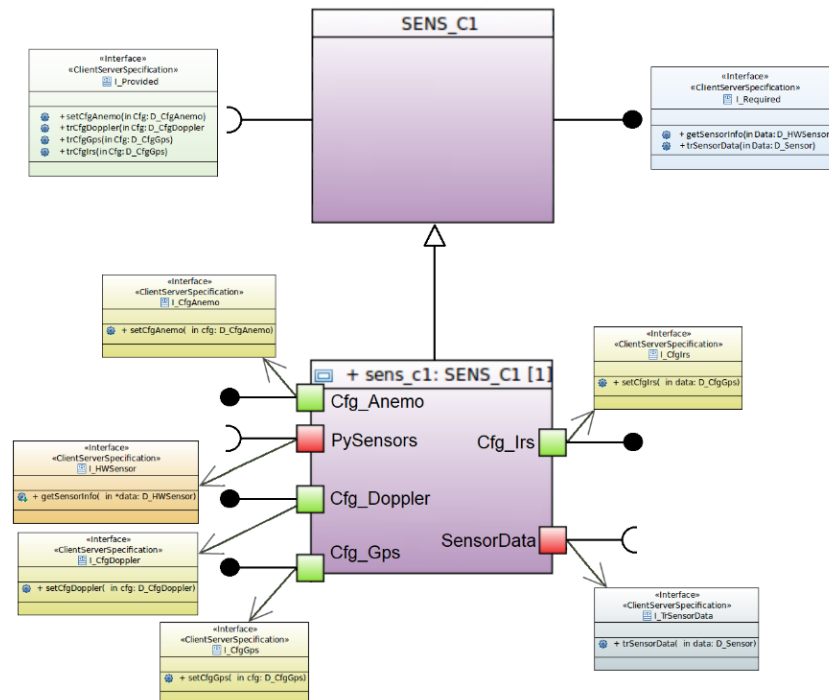


Figure 3: Example of Component interfaces

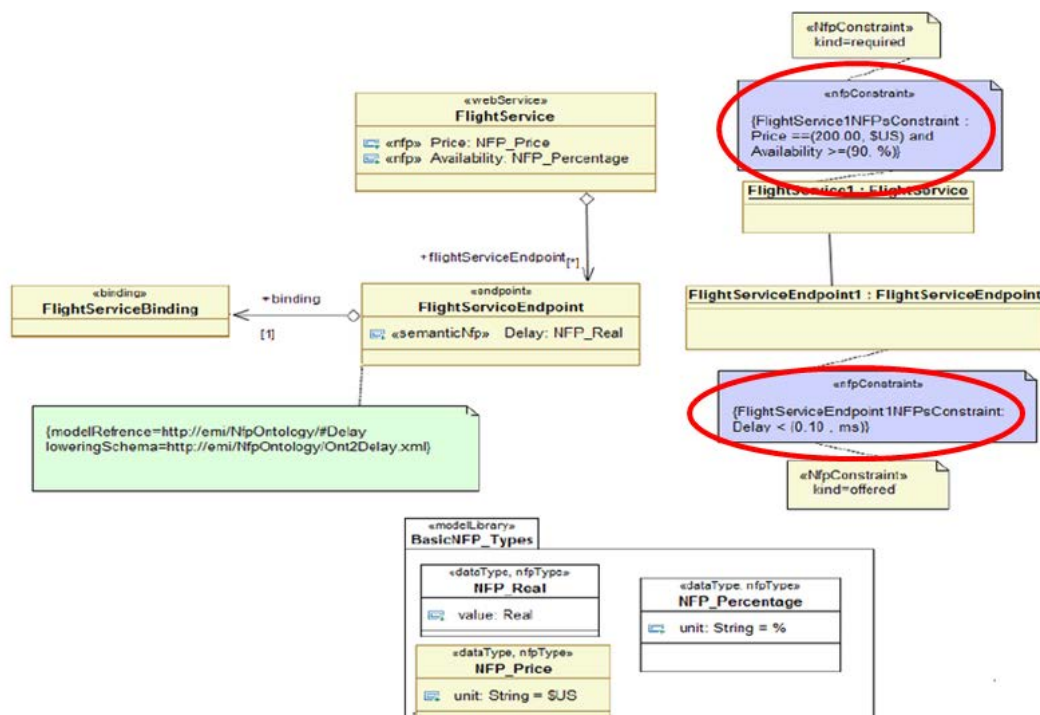


Figure 4: Example of Non-Functional requirements

The S3D FitoptiVis framework includes tools that use these parameters for performance estimation in different platforms at different abstraction levels. UC and TASE are collaborating in the development and evaluation of the S3D framework in FitOptiVis, especially in the design of the autonomous exploration use case (UC-10).

## 4.2 Design Space Exploration for Reconfigurability

UNIVAQ is defining a model-driven ESL HW/SW co-design methodology for the design of run-time reconfigurable heterogeneous parallel dedicated systems with focus on “DSE for reconfigurability”. UNIVAQ is also developing a set of prototypal SW tools to support the methodology. *Figure 5* shows the reference ESL HW/SW co-design flow.

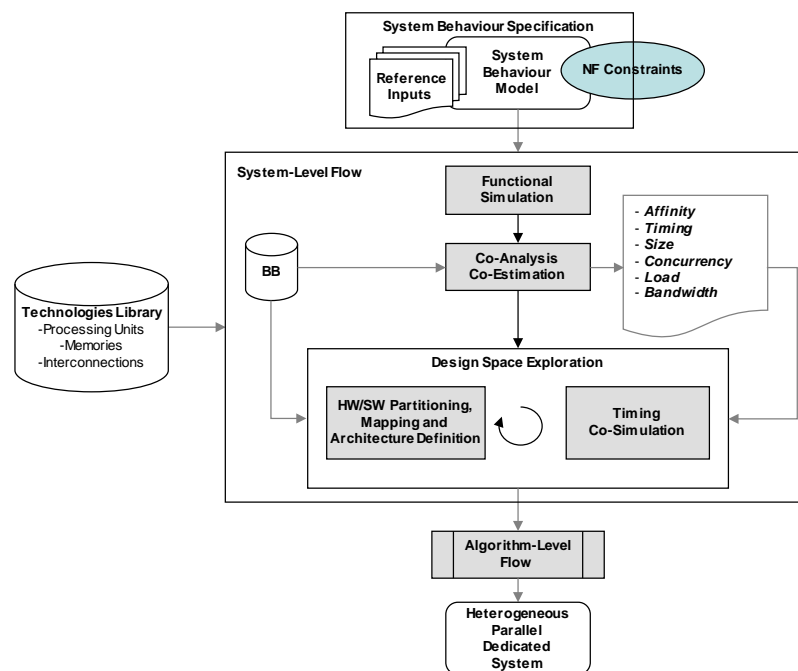


Figure 5: Reference ESL HW/SW co-design flow

The final goal is to identify (at design-time and possibly also at run-time) suitable “reconfiguration plans” for different trade-offs (e.g., timing vs energy/power vs accuracy) by trying to consider both a heterogeneous set of HW components (with possible multiple working points) and a set of alternative algorithms implementations providing results with different accuracy (i.e., by means of approximate computing techniques).

More in detail, the plan is the following:

- To improve an existing system-level metric for timing performance [4.3] and, based on this one, to define an innovative system-level metric for energy consumption.
- To extend an existing DSE approach [4.4] to consider also “energy consumption” and “max power dissipation” 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.
  - From the power point of view, the goal is to suggest to the designer architecture/mapping solutions able to never dissipate an amount of

---

power greater than a given threshold while still satisfy timing requirements.

- To evaluate the possibility to extend the previous DSE approach also to consider “approximated computing” techniques. The goal is twofold:
  - 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]
- Automated inconsistency finding in case of inconsistent requirements. [ReqV].
- Organization and storage of requirements in an online platform. [ReqV].
- Automatic synthesis for goal oriented "correct-by-construction" policies from a system model and an objective. [HyDRA].

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].
- Hybrid model of the system with safety limits. [HyDRA].

The outputs are:

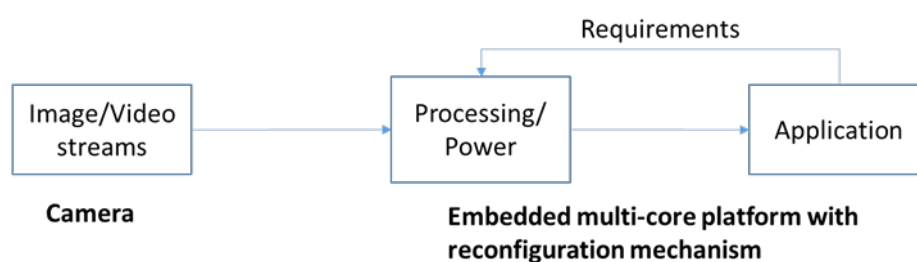
- Consistency result (yes/no). In the case of inconsistency, the tool returns the minimal set of requirements that causes the inconsistency. [ReqV]
- A yes/no answer on whether the system can be used to achieve the tested use case. A yes answer comes with a correct by design plan to achieve the given 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 6*. 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 6:* 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 [1]. 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:

$$\begin{aligned} x(k+1) &= A \otimes x(k) \oplus B \otimes u(k) \\ y(k) &= C \otimes x(k) \oplus Du(k) \end{aligned}$$

where  $x(k)$  captures the state-vector containing time-stamps of  $k^{\text{th}}$  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  $k^{\text{th}}$  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 7*) 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.

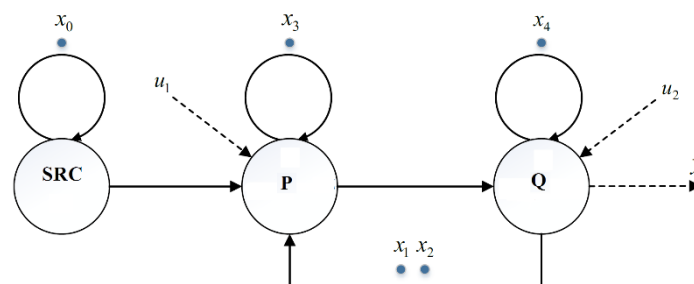


Figure 7: Synthetic SDF model

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.

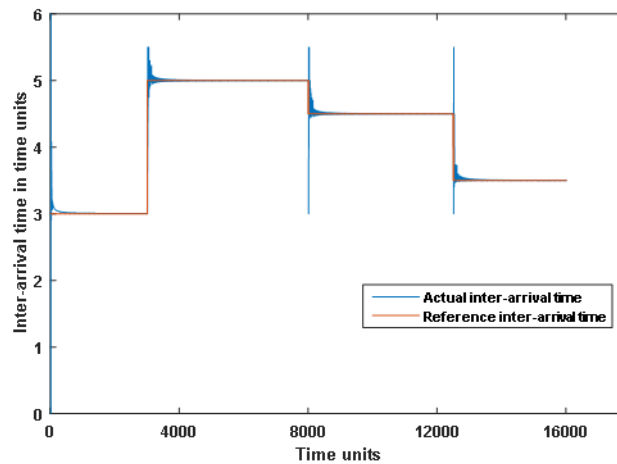


Figure 8: Inter-arrival time

Figure 8 shows tracking performance of the proposed approach for a static application (where workload 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 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 datapath 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.

---

## 4.6 Design time Support for High Level Tool Chains

UTIA prepared design time support for 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 – 7.6 for details.

## 4.7 High-level abstract component model and DSL

CUNI has been acting in the workpackage 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.

We model 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.

## 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 an initial WP3 link to the WP4 programming support developments.

### 5.1 Static resource allocation and runtime scheduling

Performance of applications can downgrade when the resources of the target platform are used inefficiently (e.g. cache, external memory, communication with devices, etc). This is especially relevant for platforms with limited resources like FPGA and single board computers. Our objective is to distribute algorithms to the target platform such that its resources are used in optimal way.

The motivation is twofold. 1/ acceleration of algorithms, so that one device can process more input data (e.g. higher FPS when processing video). 2/ reducing computational demands of the algorithms in order to reduce costs (power consumption, cheaper device). Saved resources can be used by other tasks, further reducing costs.

During algorithm development, the key part is profiling which can hint on resource consumption during real execution of the algorithm (i.e. not simulated). Based on the real resource utilization, algorithm can be transformed to different tradeoffs between memory and computation. 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 informations and statistics. Examples to this are detection of objects where the time required for analyzing an image is dependent on the image content. But we can determine the upper limit by analyzing large dataset of images.

Performance of applications can be improved by selection of appropriate combination of devices - CPU+GPU, CPU+FPGA, etc. The example of this is license plate recognition application, where plate detection is placed on FPGA and recognition is executed on CPU (which can be quite cheap). Running both tasks without FPGA would require much more powerful CPU or even GPU to handle high frame rate video stream.

On CPU most prominent causes of performance problems are: context switching by operating system and cache misses due to wrong memory access pattern. Context switch problem can be to some extent eliminated by assignment of tasks to dedicated computational nodes (CPU) which is usually supported by operating system. Cache miss problem can be solved e.g. by processing of image in small blocks which allows for sharing cached data. But this often leads to complete rewrite of the algorithm. Gathering statistical information about memory access can in such cases help to improve memory access pattern.

During development for CPU and GPU we usually use standard profiling tools like Intel VTune, gprof, CodeXL (for AMD GPU), NVidia Visual Profiler (for NVidia GPU). These tools help with identification of parts of algorithms that are computationally intensive and they usually record everything that is useful only for short time profiling (due to data volume). In practical applications and for long-time analysis we often need

to execute profiling using events and to record only few interesting variables. For this, we developed profiling tool that gathers statistical and profiling information using simple macros in program. Results can be displayed in GUI application (see screenshot).

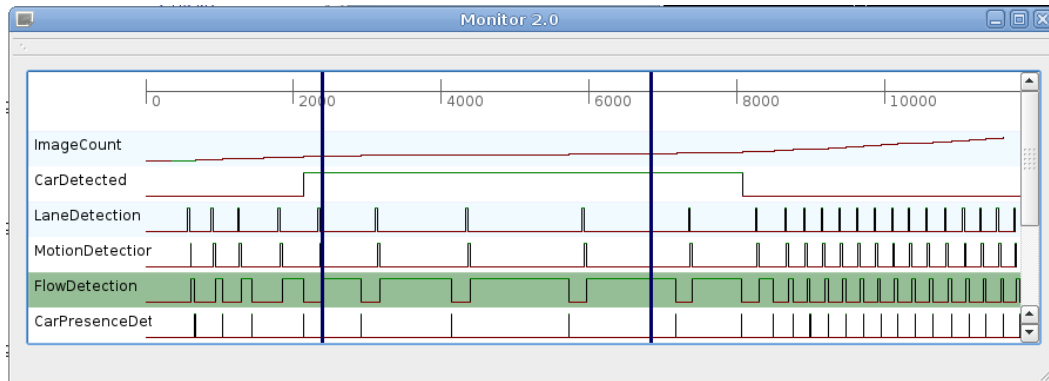


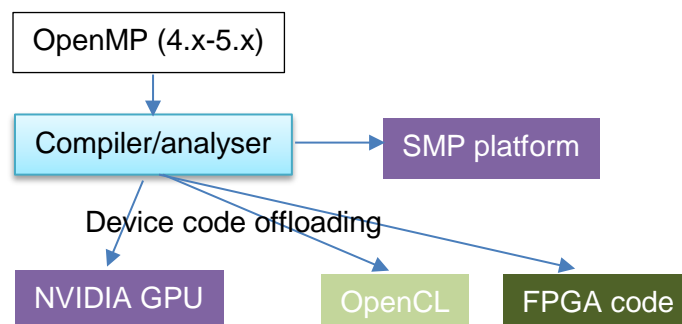
Figure 9: User interface with visualization of profiling

For development on FPGA, we use Xilinx Vivado Design Suite - for bitstream synthesis and simulations. For debugging of designs we use Integrated Logic Analyzer which allows for recording of selected signals in a short time frame on triggers. For profiling and long-time statistical analysis of IP Cores, ILA need to be connected to a custom block gathering the required informations.

In Task 3.2, BUT will focus mainly on development of a tool for on-the-fly profiling of IP Cores. We will mainly focus on video processing pipelines, but the profiling can be useful for other tasks too. The tool will specifically monitor latencies of data inputs and internal states of IP Core (using custom signals). We will use the profiling for optimization of object detection and HDR image acquisition - identification how to split the tasks to FPGA/ARM and to optimize memory access. This tool will be an extension to the current profiling tool described above.

## 5.2 OpenMP for real-time video systems

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. For example, the latest version of the most popular open source compiler (such as gcc and clang) support SMP architectures as well as code offloading to NVIDIA GPUs. In FitoptiVis,



*Figure 10: Extension of OpenMP offloading capabilities*

UC will extend the offloading capabilities of OpenMP (Version 5) with a new feature: source code offloading. This new feature allows generating source code from the original OpenMP code. This new framework supports two new targets: OpenCL and HW accelerator oriented code generation. The first framework will generate OpenCL code from OpenMP. It uses the source-code offloading capability to generate the source code that is automatically translated to OpenCL. The current version of the tool only allows to translate “parallel for” OpenMP constructions.

Additionally, UC and TASE are evaluating the extension of this methodology to FPGA platforms with LEON cores.

### **5.3 Design Time Support for C/C++ Compilers and OpenCV Algorithmic Libraries**

UTIA released design time support for C/C++ compilers and OpenCV algorithmic libraries suitable for development/debug/execution as ARM SW on the 32 bit dual-core ARM A9 systems and on the 64 bit quad-core ARM A53 in case of UltraScale+ systems. See Chapters 7.1 – 7.6 for details.

UC is planning to use 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

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

### **5.4 TTA-based Co-Design Environment (TCE)**

This section presents a co-design environment for Transport-Triggered Architectures (TTA).

#### **5.4.1 TCE: 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 basename as corresponding 32-bit instructions, but postfix “64”, i.e. the 64-bit add instruction is named “add64”.

### 5.4.2 TCE: Loop Scheduling Support

Instruction scheduling is one of the last phases of the TCE compiler. Instruction scheduler organizes the instructions into such order, that the original program semantics is preserved, but the hardware can execute the code in as efficiently as possible. Statically scheduled architectures such as VLIW and TTA processors execute the code in exactly the order specified by the compiler, so the quality of the instruction scheduler has big impact on the performance. On exposed datapath architectures such as TCE, the instruction scheduler can also perform various low-level optimizations which can further increase performance and save energy.

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

The idea of software pipelining is described in *Figure 11*. 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.



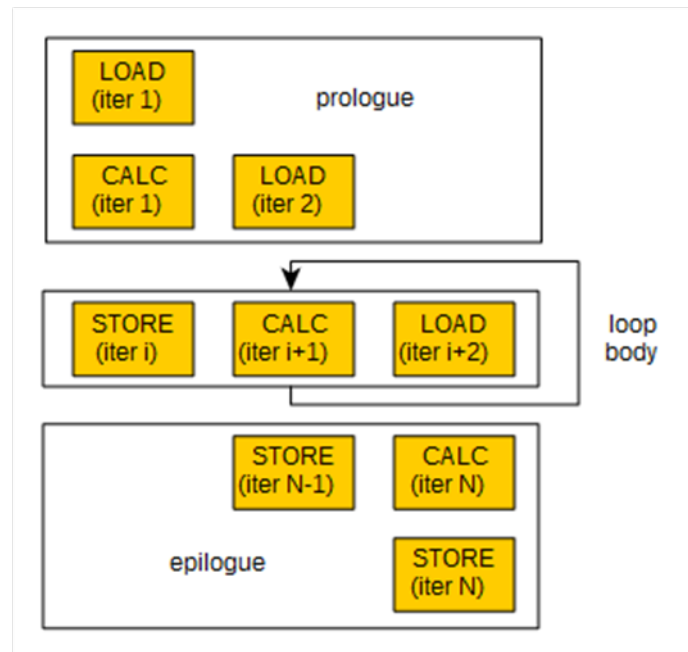


Figure 11: High-level example of software pipelining

Figure 11 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.

The compiler of TCE has a loop scheduling mode, which supports software pipelining. The Loop Scheduler of the TCE compiler can 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 TCE loop scheduler 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 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.

### 5.4.3 TCE: 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 while executing them. When a block of instructions does not contain any

backwards jumps, if 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++;
}
```

---

## 5.5 Block Copier: 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 12*. 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.

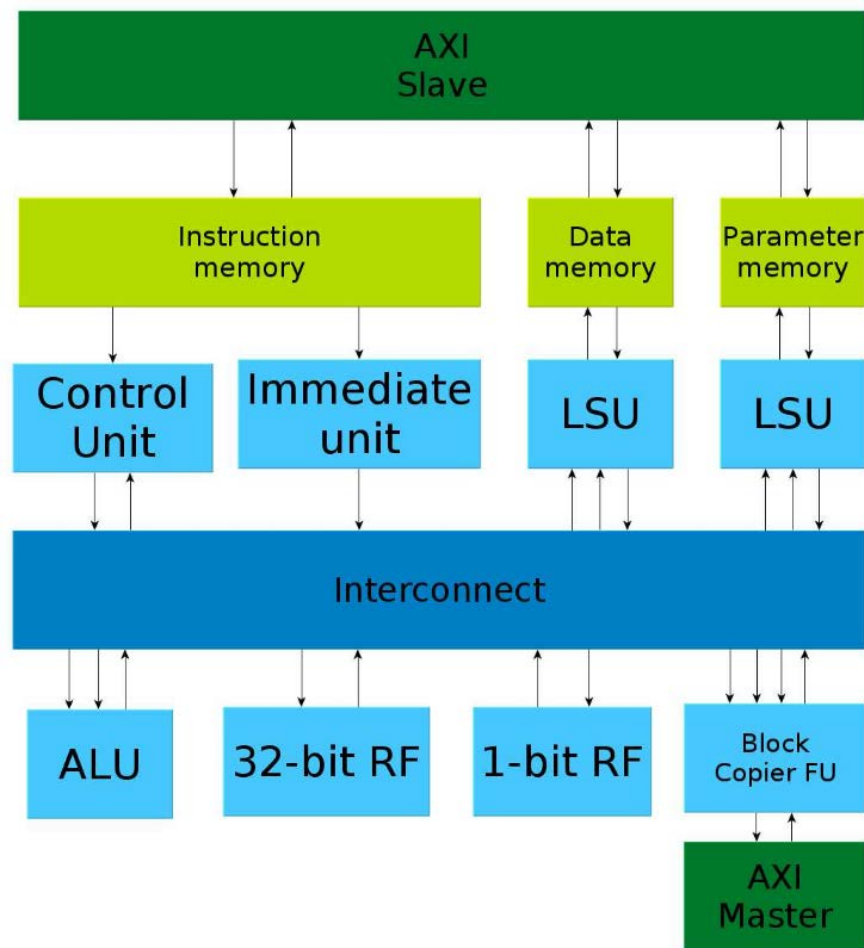


Figure 12: H Architecture of the block copier ASIP

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.

## 5.6 Deterministic timing in distributed systems and latency control with Time Sensitive Networks (TSN)

Time Sensitive Networks is a set of standard Ethernet technologies focused on distributed and synchronized real-time systems. This section addresses those TSN standards focused on time distribution and Real Time Quality of Service (RT-QoS), namely bounded latency or guaranteed bandwidth. On one hand, IEEE 802.1AS defines the generalized Precision Time Protocol (gPTP), which facilitates event coordination and data correlation on distributed sensors and actuators. On the other hand, IEEE 802.1Qbv defines the Time-Aware traffic Shaper (TAS), a mechanism to provide strict arbitration of the available bandwidth. TAS makes use of the network time synchronization, sourced by gPTP, to guarantee end-to-end RT-QoS.

gPTP provides fault tolerant synchronization requiring little network bandwidth and processing overheads. The protocol should be executed on every node or time-aware station, participating in the time synchronization, to provide the following 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.

Furthermore, the Time-Aware traffic Shaper, closed to the Medium Access Control layer, performs a time driven strict cyclic schedule of prioritized traffics. As a result, output bandwidth is slotted, thus guaranteeing the available bandwidth for each traffic type. As time-driven traffic schedulers are synchronized through gPTP, end to end deterministic packet latency is assured.

## 6 Acceleration support

This chapter describes HW accelerator oriented 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 accelerator

The extended offloading capability of OpenMP that was described in the previous section will be used to generate code for FPGA-based hardware accelerators. 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. Thus 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).

```
void vadd(float *a, float *b, float *c, int size) {
    #pragma omp target data \
    map(to:a[0:size], b[0:size],size) map(from:c[0:size])
    { // Transfer data from host to accelerator
        #pragma omp target device(1)
        { // execute next code in the accelerator
            int i;
            #pragma omp parallel for
            for (i = 0; i < size; i++)
                c[i] = a[i] + b[i];
        } // end accelerator execution
    } // Transfer data from accelerator to host
}
```

Figure 13: OpenMP code offloading

This protocol is explicitly implemented in OpenCL and it is implicit in OpenMP. Figure 13 shows a simple OpenMP code that implements this default protocol. The first pragma (`#pragma omp target data`) defines the hardware variables and transfers data from the host to the target. The second pragma (`#pragma omp target`) starts the accelerator execution (in this example “device(1)”) and locks the program until it finishes. In case of FPGA-based accelerators, this code has to be synthesized by a High-Level Synthesis (HLS) tool.

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 will try to minimize these overheads.

Finally, UC and TASE are exploring the application of this methodology to the autonomous exploration use case (UC-10).

## 6.2 HW Accelerators Generated by the Xilinx SDSoC System Level Compiler

UTIA released design time support for C/C++ compilers and OpenCV algorithmic libraries suitable for automated compilation to HW accelerator by the Xilinx SDSoC system level compiler and by the Xilinx HLS design flow.

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 M12, the supported HW accelerators include:

- Edge detection accelerator based on Sobel filter
- Motion detection accelerator based on two Sobel filters
- Lucas Kande Dense Optical Flow accelerator

See Chapters 7.1 – 7.6 for details

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

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

- the ability to compose different high-level abstract functional specifications into
- a single accelerator, exploiting on coarse-grained reconfigurable technologies
- automatic resource minimization
- automatic reconfiguration management

The MDC suite components are:

- Baseline MDC Core – performing dataflow-to-hardware composition, by means of datapath 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 substrate 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 Xilinx compliant multi-functional accelerator. Starting from the input dataflow specifications set, such an accelerator can be either loosely coupled or tightly coupled, according to the design needs, and also its drivers are derived.

The inputs are:

- 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

And finally, the outputs are:

- (baseline functionality) 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

Concerning the usage within FitOptiVis, the objective at M12 is offer some parts as open source. MDC so far has never been released open source. We intend to have available at M12 some basic documentation and a starting kit to allow its wide usage (Task 3.3).

In parallel, according to the perspective adoptions in the use-cases, extensions are going to be discussed to provide

- An extended support of heterogeneity and flexibility – connections with WP2 is envisioned here. MDC already offers topology exploration support, which has to be extended to: 1) define proper host to co-processor connectivity support, to define proper coupling strategies with the host and to be able to implement, and 2) starting from the high-level specifications, all the identified working points (Task 3.1).
- New APIs – to hide task delegation, data retrieval and communication medium management for the new supported functionalities (Task 3.2).

Within FitOptiVis such tool will be evaluated by users (i.e. AITEK which is focused on high level video processing software development), by accelerating video processing algorithms, collecting measurements and comparing them with traditional implementation and providing relevant feedback to UNICA and UNISS for future improvement.

In particular, as a possible first example, state of the art algorithms for the foreground detection could be used, which can be accelerated thanks to FPGA-based implementation obtained using the MDC suite. It is worth noticing that these algorithms are the first elaboration step of more complex video processing algorithms already used by Aitek in several of its systems to detect targets and to extract high level information from the video flow.

We are considering use as a possible metric the execution time of the foreground detection 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.

State that UNICA and UNISS Objective at M12 is to have available documentation and a starting kit, AITEK could be an early adopter able to provide useful preliminary feedback about the usability of this tool. Having already establish fruitful cooperation UNICA, UNISS can provide support to AITEK in the usage of MDC, in particular for the input creation, parsing AITEK C++ code to a suitable format (dataflow).

## 6.4 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 fanout, 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.

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 %.

### 6.4.1 TCE: AutoExplorer

The design space explorer tool is a part of the TCE framework. Its purpose is to run various exploration algorithms defined as plugins 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.

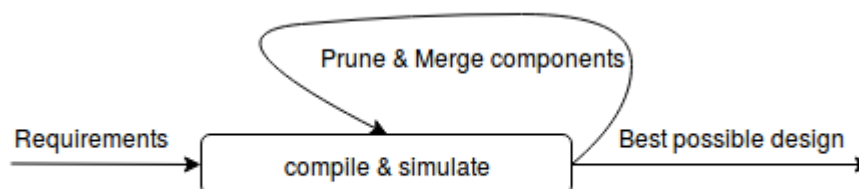


Figure 14: Simplified TCE Exploration process of AutoExplorer

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 is a design space exploration plugin of which goal is to generate application specific processors by analyzing the application. 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. AutoExplorer plugin drives multiple exploration plugins in a sequence, and picks best possible configuration from each stage based on estimation information and uses configuration id as an input for the next plugin stage. The advantage of the automated exploration is that it can evaluate several hundreds of different designs before finding optimal solution. This allows automated rapid prototyping of different architectures for specific application set.

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

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

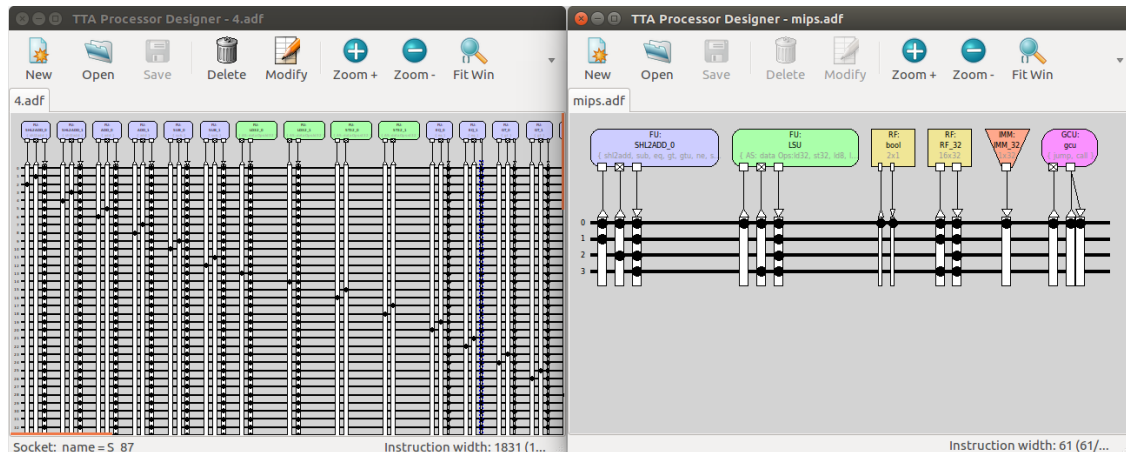


Figure 15: Unoptimized architecture (left), final best possible architecture (right).

Figure 15 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 autoexploration.

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 AutoExplorer'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. The AutoExplorer 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.

## 6.5 Acceleration of individual algorithms in combination of FPGA and CPU

### 6.5.1 Object detection on FPGA using Waldboost algorithm

The object detector is based on Waldboost algorithm. It achieves a real-time performance on multiple scales, without the requirement of external DRAM memory. The algorithm itself and FPGA design are published in article [6.1]. The detector is completely FPGA accelerated. The CPU of Xilinx Zynq is used only for configuration of FPGA detector, as loading of classifier tables. CPU and even GPU versions of the detector also exist, thus it is possible to implement it on wide range of embedded platforms (with varying performance), including Nvidia Tegra.

The interfaces are:

- Image input: accepting an AXI Video Stream of predefined resolution (set during FPGA synthesis).
- Detection output: Axi stream with detections, containing X and Y coordinates of detection, threshold of Waldboost classifier (probability of object) and scale, in which the object was detected
- Axi Lite memory-mapped interface - memory mapped BRAMs for storing classifiers, which are fed by CPU

In FitOptiVis, we will perform the training process for detection of license plates for road traffic surveillance use case. The license plate detection will be connected to license plate recognition based on neural networks (CNN). Recognition will be placed on CPU since CNNs are not suitable for FPGA. From the algorithmic point, we will focus on update from LBP/LRD-based detector to more powerful Aggregated Channel Features detector based on decision trees.

### 6.5.2 HDR Image acquisition

The HDR image acquisition is composed from three main blocks: image capturing, HDR merging and tone mapping (See image IMG1). 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.

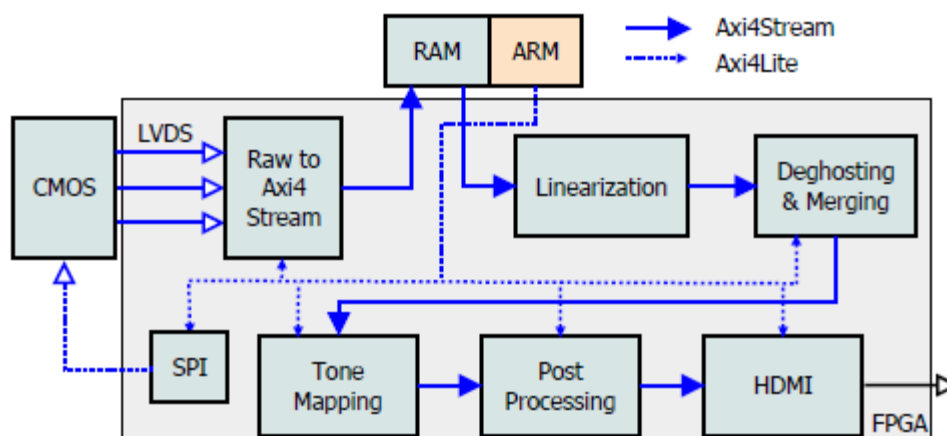


Figure 16: HDR acquisition pipeline

Figure 17 presents overall schematics of HDR acquisition pipeline. The individual blocks create a dataflow pipeline, which is configured through AXI Lite interface from ARM CPU.

## 6.6 HDR Merging

This block merges configurable number of images into one HDR according to algorithm presented in paper [6.2]. 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. Merging is possible to perform on grayscale, Bayer mask, and RGB data. Output consists of AXI Video stream with wider bit depth, by default in representation of 10.8 bits - 10 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).

Interface:

- Input - HDR merging block expects multiple AXI Video streams, according to configuration. Data format is one/three 8-bit values per pixel according to desired input: grayscale/Bayer image or RGB.
- AXI Lite - Merging block is continuously configured from software. Every change in exposure times, gains etc. results in software modification of constants used by the HW blocks. It is convenient to calculate these constants by software, since it saves big amount of FPGA resources. For example, the division in merging equation is converted into multiplication, where the constant  $1/n$  is calculated by software and updated based on current exposition settings.

Within FitOptiVis, we are planning to involve more advanced deghosting algorithm into HDR merging block. Alternatively, we are planning to experiment with a multi-sensor HDR acquisition, where the computationally demanding deghosting algorithm will not be necessary since all images are captured at the same time.

## 6.7 Tonemapping

We have an implementation of Durand TMO, which is based on bilateral filtering and also several basic TMOs, linear, logarithmic and exponential. The TMO block is written in HLS, enabling 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 17. The input image is separated into base and detail layer, 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.

The interfaces are:

- AXI Video Stream standard is used for input and output data interface. While the input is the grayscale image with Bayer mask with 18bits per pixel, the output is 8bit RGB image.
- AXI Lite interface - this interface is used for grabbing the image histogram and other statistic data, which are used for passing back the software adjusted tonemapping properties (e.g. min/max values, scale coefficients for histogram balancing).

Within FitOptiVis, we are planning to involve the temporal coefficient filter, which will enable the smoother brightness adaptation - without this smoothing, the resulting image might suffer with flickering, e.g. from light sources powered from PWM power sources or by AC power.

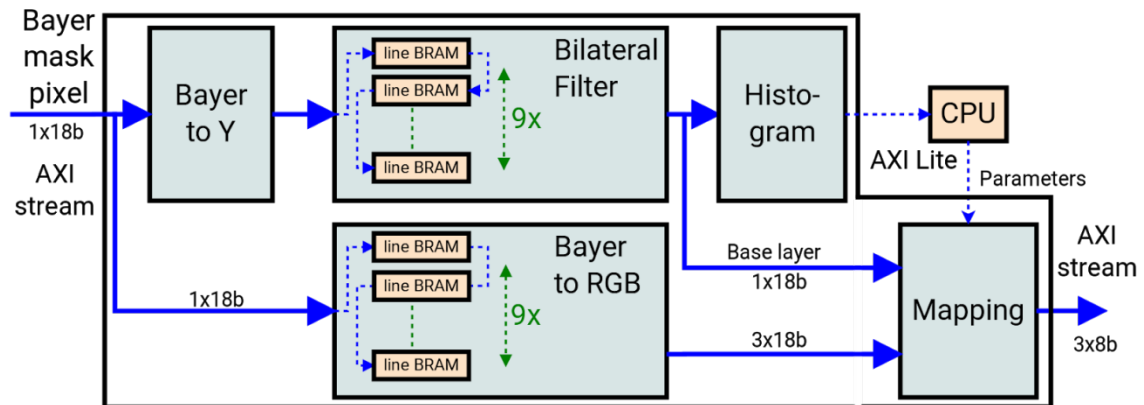


Figure 17: The FPGA implementation of Durand tonemapping

## 6.8 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 <https://arxiv.org/abs/1804.02767>

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.



---

## 6.9 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).

Aitek will contribute to these tasks by using and evaluating the CNN accelerator provided by UNICA.

In particular, Aitek will provide its own video monitoring platform, consisting of different video processing software modules, which will be accelerated thanks to NEURAghe. Such video processing methods were based on traditional foreground and background separation: recently there is an increasing interest for the deep learning algorithms, which can increase dramatically the performance achievable, in particular for some tasks like target classification.

While such algorithms provide extremely promising results, they are quite resource demanding, requiring specific HW or as possible thanks to NEURAghe, a reconfigurable acceleration support.

In more detail, Aitek is going to use different single shot object detectors, which enable the detection and the classification of multiple objectives within the image, by analysing a single frame.

The performance of NEURAghe will be evaluated in comparison to traditional implementation, focusing in particular on execution time and supported video quality and frame rate. To have a relevant comparison, different algorithms will be used, images with different characteristics are analysed and traditional implementation on different hardware will be tested as a benchmark.

Aitek, being the leader of Use Case 1, focused on the water supply monitoring and control, could be not only a simple user of NEURAghe but can support also UNICA in defining the specific customization needed to adapt their tool to meet the requirements for UC1, in which they are involved.

## 6.10 Nokia acceleration work for PCC demo case

The upcoming MPEG standard for video-based point cloud compression (V-PCC) is built around 2D video encoding technology, as illustrated in Fig. 18. As standard video coding technology can be utilised, existing hardware solutions and distribution infrastructure can be utilised. I.e. existing hardware video encoders and decoders, available on any modern mobile handset, can carry the bulk of the processing operations.

Our experiments have shown that most modern mobile handsets are capable of achieving real-time decoding of at least 25 frames per second, as shown in Tab. 2, as well as real-time AR rendering, as shown in Tab. 3. Thus, proving the general claim of real-time capability of V-PCC.

However, bottlenecks exist in the synchronisation and buffering of the parallel video streams. A particular problem is the handling of decoded video outputs on Android devices. Here, FitOptiVis will improve over the existing standard with a efficient and

effective synchronisation solution, enabling V-PCC real-time decoding and playback on modern Android handsets.

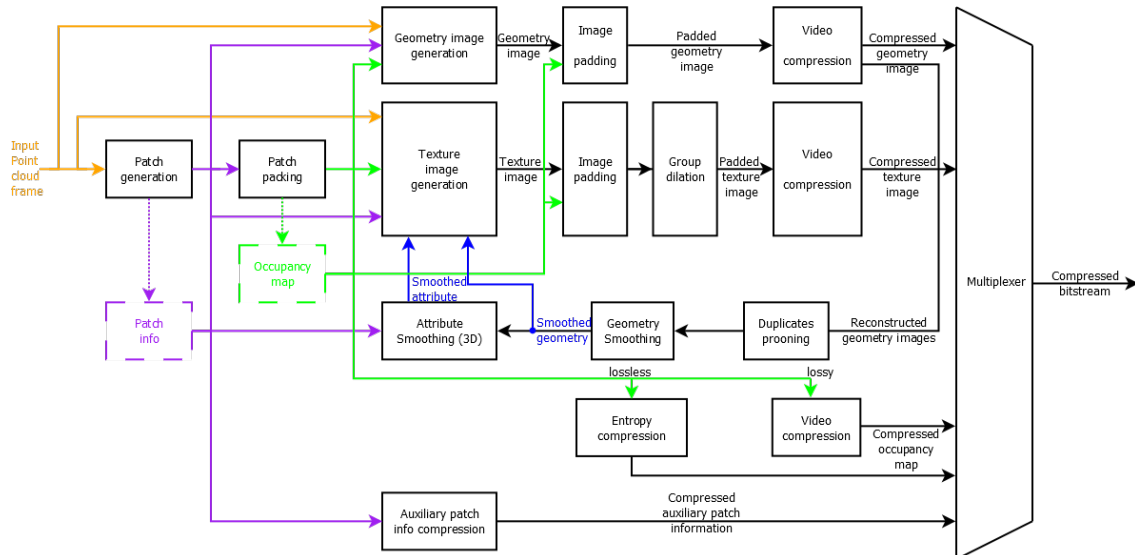


Figure 18: MPEG V-PCC encoder block diagram

DEVICE	GPU / CHIPSET	FPS
iPhone 6S	Apple A9 GPU	15.4
iPhone 8	Apple A11	30.3
iPhone X	Apple A11	30.3
iPhone XS	Apple A12 GPU	30.3
Samsung Galaxy S10	Mali-G76 / Exynos 9820	25.0
Huawei Mate Pro 20	Mali-G76 / Kirin 980	23.3
Google Pixel	Adreno 530 / Snapdragon 821	29.4
Google Pixel 2 XL	Adreno 540 / Snapdragon 835	30.3
Nokia 8 Sirocco	Adreno 540 / Snapdragon 835	30.3
Google Pixel 3	Adreno 630 / Snapdragon 845	25.6
OnePlus (A6013)	Adreno 630 / Snapdragon 845	25.6

Table 2: V-PCC decoding performance overview

Release year	Device	GPU	Mpoints/frame @ 60 fps
2014	Samsung Galaxy Note 4	Adreno 420	1.04
2016	Samsung Galaxy S7 Edge	Mali-T880	0.43
2016	Google Pixel	Adreno 530	1.60
2017	Google Pixel 2	Adreno 540	1.81
2018/Q4	OnePlus 6T	Adreno 630	1.90
2018/Q4	Huawei Mate Pro 20	Mali-G76	1.32

Table 3 : V-PCC AR point rendering capabilities

## 7 Design Time Support for Methodologies and Tools

This chapter describes design time support for methodologies and tools released by FitOptiVis WP3 partners to project partners and to general public use before M12 of the project (end of April 2019). This activities have been developed in T3.1 and T3.3.

WP3 partners developed, documented and released for public use first design time resource support for a family of Xilinx Zynq and zynq Ultrascale+ systems with support for PetaLinux 2018.2 kernel, Debian 9.8 operating system, the Xilinx SDSoC 2018.2 system level compiler and the Xilinx Vivado 2018 design tool.

Project released support for Xilinx Zynq systems has these specific features:

- ZynqBerry system presents small, low cost system with design time support being developed in FitOptiVis. It has the RaspBerry form factor and works with the (28nm) Xilinx 32bit Zynq device with small programmable logic area. See Chapter 7.1 and 7.2 and application note with evaluation package [7.1]
- Medium size 16nm 64bit Zynq UltraScale system with design time support being developed in the FitOptiVis. It is re-using the carrier board and the Full HD video I/O FMC card used in the Almarvi project. See Chapter 7.3 and 7.4 and application note with evaluation package [7.2]
- Large 16nm 64bit Zynq UltraScale system with design time support being developed in the FitOptiVis. 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. See Chapter 7.5 and 7.6 and application note with evaluation package [7.3]

Table 2 is summarizing progress made by the FitOptiVis partners in the WP3 from M1 to M12.

Table 4: Progress made in FitOptiVis in WP3 in M12

<ul style="list-style-type: none"> <li>➤ <b>ALMARVI - end of project:</b> <ul style="list-style-type: none"> <li>▪ Zynq 7000 family (28nm)</li> <li>▪ Stand-alone only</li> </ul> </li> <li>➤ <b>ALMARVI limitations:</b> <ul style="list-style-type: none"> <li>▪ Limiting PL size of Zynq</li> <li>▪ no VCU, no GPU</li> <li>▪ no USB support</li> <li>▪ no Ethernet board 2 board communication framework</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>➤ <b>FitOptiVis – M12:</b> <ul style="list-style-type: none"> <li>▪ + Zynq Ultrascale+ 16nm</li> <li>▪ + Small: ZynqBerry 28nm</li> <li>▪ + PetaLinux OS</li> <li>▪ + Debian FS support</li> </ul> </li> <li>➤ <b>FitOptiVis progress:</b> <ul style="list-style-type: none"> <li>▪ + Large PL of Ultrascale+</li> <li>▪ + VCU, + GPU,</li> <li>▪ + USB</li> <li>▪ + Ethernet board 2 board communication based on Arrowhead Framework</li> </ul> </li> </ul>
---	--

The released and documented design time resource results are described next.

---

## 7.1 Board support package generation project for SDSoC 2018.2 compiler for ZynqBerry board TE0726

This design time resource is addressing small ZynqBerry board. It is system without video I/O support. It supports the SDSoC compiler and video processing algorithms can be compiled and evaluated in file-to-file mode, with the Debian file system, 100 Mbit Ethernet and board-to-board communication based on the Arrowhead framework running on separate RaspberryPi 3B (RPI3) board.

### 7.1.1 Description

The board support package generation project serves for generation of the board support package for the TE0726-03M board. The board support package provides all necessary files needed for the Xilinx SDSoC 2018.2 compiler. The compiler needs this board support package to be able to compile selected C/C++ Arm A53 functions into HW accelerators and the corresponding bit-stream for the programmable part of the design.

The board support package includes all necessary information for preparation of the low level SW support for the preconfigured and precompiled Petalinux 2018.2 kernel and for the precompiled Debian 9.8 “Stretch” image for the for the TE0726-03M board. The platform generator for SDSoC 2018.2 compiler on ZynqBerry board TE0726 is released for use by project partners and for general public (for free of charge download) in form of application note with evaluation package [7.1].

This application note describes FitOptiVis design time and run time resources supporting the ZynqBerry board and Xilinx SDSoC 2018.2 system level compiler.

The ZynqBerry TE0726-03M [7.5] board works with Xilinx XC07010-1C device with the dual core Arm A9 32 bit and relatively small programmable logic area on single 28 nm chip.

See *Figure 18*. The ZynqBerry PCB has RaspberryPi 2 form factor. The ZynqBerry board is designed and manufactured by company Trenz Electronic [7.5].

The Xilinx SDSoC 2018.2 compiler requires preparation of SDSoC platform. It is specific Vivado 2018.2 design with metadata, enabling to the SDSoC LLVM system level compiler to add additional HW accelerator blocks and data movers on top of the initial Vivado design.

Configurations of Petalinux kernel and Debian image are described in chapters 4 and 5 of [7.1].



Figure 19: Two ZynqBerry boards and RPi3

### 7.1.2 Inputs:

- Vivado 2018.2 HW reference block design with possible user modifications.
- Reference configuration files for PetaLinux 2018.2 with possible user modifications.
- Reference configuration files for Debian 9.8 with possible user modifications.

### 7.1.3 Outputs:

- The board support package for the Xilinx SDSoC 2018.2 compiler for the TE0726-03M board with PetaLinux 2018.2 kernel and Debian 9.8 OS.

### 7.1.4 Usage within FitOptiVis

This design time resource serves to the FitOptiVis partners as supported reference methodology for creation of the board support package for the TE0726-03M board. It is needed for the Xilinx SDSoC 2018.2 compiler.

- Partners can modify the provided reference Vivado 2018.2 HW block design with own, application specific HW blocks and modules.
- The board support package generation project propagates the corresponding changes to the Petalinux kernel and to the user defined custom board support package for the Xilinx SDSoC 2018.2 compiler for the TE0726-03M board with Debian OS.

---

## 7.2 Design time support for SDSoC 2018.2 compiler for ZynqBerry board TE0726

This design time resource serves as introduction to C/C++ SW code compatible with the SDSoC compiler. The released package [7.1] contains 24 SW projects.

### 7.2.1 Description

The released design time support for SDSoC 2018.2 compiler is collection of 24 predefined and tested SW C/C++ projects with guidelines for compilation and use on the for ZynqBerry board TE0726 with Debian OS. The SW is released in the evaluation package [7.1] with use on ZynqBerry board TE0726 documented in Chapter 4 of the application note [7.1]. Examples test the Xilinx SDSoC 2018.2 compiler. The SDSoC LLVM system level compiler compiles selected SW functions into HW blocks for the programmable logic part of the Zynq device. Generated HW accelerator blocks and data movers are automatically integrated with the initial Vivado 2018.2 design present in the board support package for the TE0726-03M board. The additional HW accelerator blocks are defined as C/C++ user defined functions. These functions can be compiled, debugged and executed in Petalinux user space on ARM A9. But in addition, the selected C/C++ functions can be compiled also to form of Vivado HLS HW accelerators. Blocks are compiled by the Vivado HLS compiler and automatically interfaced with dedicated data movers like DMA or SG DMA.

### 7.2.2 Inputs:

- The Xilinx SDSoC 2018.2 compiler requires the board support package for the TE0726-03M board.
- Predefined SW projects with algorithms coded in C/C++ for the Arm A9.
- User defined new SW projects with algorithms in C/C++ for the Arm A9.

### 7.2.3 Outputs:

- Compiled bitstream with HW accelerator(s) and SW boot files serving for programming of the QSPI flash of the TE0726-03M board.
- Compiled SW project for Arm A9 with integrated HW accelerator(s) in form of executable code for the TE0726-03M board with Debian OS.

### 7.2.4 Usage within FitOptiVis

This design time resource serves to the FitOptiVis partners as supported set of reference SW examples for the TE0726-03M board.

- Partners can modify provided reference SW examples with own code and compile to HW with the SDSoC 2018.2 compiler for the TE0726-03M board with Debian OS.
- Matrix multiplication and addition of int32 [18x18] matrices example is elaborated in detail. Generated HW accelerator runs with 100MHz clock provides acceleration 4x over the SW running on 650MHz Arm A9.
- The resulting compiled system running on the TE0726-03M board is compatible with the FitOptiVis WP4 run time resources:
  - 32bit Debian OS
  - local cloud Ethernet communication of C++ clients via the Arrowhead framework [7.6] (result of ECSEL Productive 4.0 project) .

The user defined application SW running on TE0726-03M board can exchange information with other boards in the local cloud. See in Chapter 9-14 of the application note [7.1] for details.

### 7.3 Platform generator for SDSoc 2018.2 compiler on Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O

This design time resource describes use of medium size Zynq Ultrascale+ system equipped with the Full HD HDMI with Video I/O support for the SDSoc 2018.2 compiler.

#### 7.3.1 Description

The board support package generation project serves for generation of the board support package for the Zynq Ultrascale+ TE0820-03-4EV-1E module [7.7] on the 0701-06 carrier board [7.10]. It works with Xilinx XCZU4EV-1SFVC784E with the quad core Arm A53 64 bit, dual Arm Cortex R5 and programmable logic area on single 16nm chip.



Figure 20: TE0820-03-4EV-1E on TE0701-06, Imageon HDMI I/O FMC card

The board support package provides all necessary files needed for the Xilinx SDSoc 2018.2 compiler. The compiler needs this board support package to be able to compile selected C/C++ Arm A53 functions into HW accelerators and the corresponding bit-stream for the programmable part of the design. The board support package includes all necessary information for preparation of the low level SW support for the



preconfigured and precompiled Petalinux 2018.2 kernel and for the precompiled Debian 9.8 “Stretch” image for the for the board. The platform generator for SDSoC 2018.2 compiler is released for use by project partners and for general public (for free of charge download) in form of application note with evaluation package [7.2].

The Full HD HDMI video I/O IP cores are described in for the TE0701 carrier in [7.4] and for the Xilinx university evaluation board XCU102 in [7.11].

The Zynq Ultrascale+ PCB module has the 4x5cm form factor. The Zynq Ultrascale+ board is designed and manufactured by company Trenz Electronic.

### 7.3.2 Inputs:

- Vivado 2018.2 HW reference block design with possible user modifications
- Reference configuration files for PetaLinux 2018.2 with possible user modifications
- Reference configuration files for Debian 9.8 with possible user modifications

### 7.3.3 Outputs:

- The board support package for the Xilinx SDSoC 2018.2 compiler for the TE0820-03-4EV-1E module on the 0701-06 carrier board with PetaLinux 2018.2 kernel and Debian 9.8 OS.

### 7.3.4 Usage within FitOptiVis

This design time resource serves to the FitOptiVis partners as supported reference methodology for creation of the board support package for TE0820-03-4EV-1E module [1] on the 0701-06 carrier board with PetaLinux 2018.2 kernel and Debian 9.8 OS needed for the Xilinx SDSoC 2018.2 compiler. See [7.2] for details.

- Partners can modify the provided reference Vivado 2018.2 HW block design with own, application specific HW blocks and modules.
- The board support package generation project propagates the corresponding changes to the Petalinux kernel and to the user defined custom board support package for the Xilinx SDSoC 2018.2 compiler for the TE0820-03-4EV-1E module with Debian OS.

## 7.4 Design time support for SDSoC 2018.2 compiler on Zynq Ultrascale+ module TE0820-4EV on TE0701 carrier with Full HD HDMI Video I/O

This design time resource serves as introduction to programming of C/C++ SW code for real-time Full HD HDMI I/O video processing on a medium size Zynq Ultrascale+ module. The released package [7.2] contains 30 projects for the SDSoC 2018.2 compiler.

### 7.4.1 Description

The released design time support for SDSoC 2018.2 compiler is collection of 24 predefined and tested SW C/C++ projects with guidelines for compilation and use on the TE0820-03-4EV-1E module on the 0701-06 carrier board with PetaLinux 2018.2 kernel and Debian 9.8 OS. The SW is released in the evaluation package [7.2].

The SDSoC LLVM system level compiler compiles selected SW functions into HW blocks for the programmable logic part of the Zynq Ultrascale+ device. Generated HW accelerator blocks and data movers are automatically integrated with the initial Vivado 2018.2 design present in the board support package for TE0820-03-4EV-1E module [7.7] on the TE0701-06 carrier [7.10]. The additional HW accelerator blocks are defined as C/C++ user defined functions. These functions can be compiled, debugged and executed in Petalinux user space on the 64bit ARM A53 processor. But in addition, the selected C/C++ functions can be compiled also to form of Vivado HLS HW accelerators. Blocks are compiled by the Vivado HLS compiler and automatically interfaced with dedicated data movers like DMA or SG DMA.

SW projects include synchronisation of the accelerated video processing with the input and output Full HD HDMI 60FPS video streams.

### 7.4.2 Inputs:

- The Xilinx SDSoC 2018.2 compiler requires the board support package for the TE0820-03-4EV-1E module on the 0701-06 carrier board.
- Predefined SW projects with algorithms coded in C/C++ for the Arm A53.
- User defined new SW projects with algorithms in C/C++ for the Arm A53.

### 7.4.3 Outputs:

- Compiled bitstream with HW accelerator(s) and SW boot files serving for programming of TE0820-03-4EV-1E module on the 0701-06 carrier board from the SD card
- Compiled SW project for Arm A53 with integrated HW accelerator(s) in form of executable code for the TE0820-03-4EV-1E module with Debian OS.

### 7.4.4 Usage within FitOptiVis

This design time resource serves to the FitOptiVis partners as supported set of reference SW examples for TE0820-03-4EV-1E module on the 0701-06 carrier board.

- Partners can modify provided reference SW examples with own code and compile to HW with the SDSoC 2018.2 compiler for the TE0820-03-4EV-1E module with Debian OS and video I/O via the Imageon FMC card.
- Matrix multiplication and addition of int32 [75x75] matrices is elaborated in detail. Generated HW accelerator runs with 200 MHz clock provides acceleration **33x** over the SW running on 1.2 GHz Arm A53.

This design time resource includes description of demo performing video processing with HW acceleration is described in this section. We demonstrate the LK Dense Optical Flow (LK DOF) algorithm with Full HD HDMI video input and video output.

The algorithm works with two subsequent Full HD frames. It computes for each pixel of the frame vector characterizing the direction and the speed of movement of a given pixel relative to its background.

The LK Dense Optical Flow algorithm involves massive fixed point computation and also floating point matrix inversion computed for each pixel of the frame. The fixed point moving sum of the pixel background is computed for a square area of 45x45 pixels for each pixel.



Figure 21: HW accelerated LK DOF input/output Full HD HDMI 60fps

Figure 20 present set-up for computation of the LK Dense Optical Flow input movie with Full HD HDMI input 60 FPS from the PC and output in Full HD HDMI to the HDMI monitor. See Table 3 summarizing the performance of HW accelerated implementation and also the load of two most utilized Arm A53 processors.

Table 5: Performance of HW accelerated LK DOF and the load of Arm A53

LK DOF algorithm with per pixel integral tile size [45x45]	Frames per second	A53 SDSoC CPU load	A53 DeskTop CPU load	Acceleration of LK DOF
In SW	0.121	100%	3%	<b>1x</b>
In HW DMA	60	100%	80%	<b>495x</b>
In HW SG DMA	60	30%	80%	<b>495x</b>

## 7.5 Platform generator for SDSoC 2018.2 compiler on Zynq Ultrascale+ module TE0808-15EG on TEBF0808 carrier with full HD HDMI Video I/O

This design time resource describes use of really large Zynq Ultrascale+ system equipped with the Full HD HDMI with Video I/O support for the SDSoC 2018.2 compiler.

### 7.5.1 Description

The board support package generation project serves for generation of the board support package for the Zynq Ultrascale+ TE0808-03-15EG-1EE module [7.8] on the TEBF0808-04 carrier board [7.9]. It works with large Xilinx XCZU15EG-1FFVC900E device with the quad core Arm A53 64 bit, dual Arm Cortex R5 and programmable logic area on single 16nm chip.



Figure 22: TE0808-03-15EG-1EE on TEBF0808-04 and Imageon HDMI I/O FMC

The Zynq Ultrascale+ [7.8] module has the 52 x 76 mm form factor. The Zynq Ultrascale+ board is designed and manufactured by company Trenz Electronic.

The board support package provides all necessary files needed for the Xilinx SDSoC 2018.2 compiler. The compiler needs this board support package to be able to compile selected C/C++ Arm A53 functions into HW accelerators and the corresponding bit-stream for the programmable part of the design. The board support package includes all necessary information for preparation of the low level SW support for the

---

preconfigured and precompiled Petalinux 2018.2 kernel and for the precompiled Debian 9.8 “Stretch” image for the for the board. The platform generator for SDSoC 2018.2 compiler is released for use by project partners and for general public (for free of charge download) in form of application note with evaluation package [7.3].

### 7.5.2 Inputs:

- Vivado 2018.2 HW reference block design with possible user modifications
- Reference configuration files for PetaLinux 2018.2 with possible user modifications
- Reference configuration files for Debian 9.8 with possible user modifications

### 7.5.3 Outputs:

- The board support package for the Xilinx SDSoC 2018.2 compiler for the TE0808-03-15EG-1EE on TEBF0808-04 carrier with Imageon HDMI I/O FMC with PetaLinux 2018.2 kernel and Debian 9.8 OS.

### 7.5.4 Usage within FitOptiVis

This design time resource serves to the FitOptiVis partners as supported reference methodology for creation of the board support package for TE0808-03-15EG-1EE on TEBF0808-04 carrier with PetaLinux 2018.2 kernel and Debian 9.8 OS needed for the Xilinx SDSoC 2018.2 compiler. See [7.3] for details.

- Partners can modify the provided reference Vivado 2018.2 HW block design with own, application specific HW blocks and modules.
- The board support package generation project propagates the corresponding changes to the Petalinux kernel and to the user defined custom board support package for the Xilinx SDSoC 2018.2 compiler for the TE0808-03-15EG-1EE on TEBF0808-04 carrier with Debian OS.

## 7.6 Design time support for SDSoC 2018.2 compiler on Zynq Ultrascale+ module TE0808-15EG on TEBF0808 carrier with full HD HDMI Video I/O

This design time resource serves as introduction to programming of C/C++ SW code for real-time Full HD HDMI I/O video processing on large Zynq Ultrascale+ module.. The released package [7.3] contains 30 projects for the SDSoC 2018.2 compiler.

### 7.6.1 Description

The released design time support for SDSoC 2018.2 compiler is collection of 24 predefined and tested SW C/C++ projects with guidelines for compilation and use on the TE0808-03-15EG-1EE on TEBF0808 carrier with PetaLinux 2018.2 kernel and Debian 9.8 OS. The SW is released in the evaluation package [7.3].

The SDSoC LLVM system level compiler compiles selected SW functions into HW blocks for the programmable logic part of the Zynq Ultrascale+ device. Generated HW accelerator blocks and data movers are automatically integrated with the initial Vivado 2018.2 design present in the board support package for TE0808-03-15EG-1EE on TEBF0808 carrier. The additional HW accelerator blocks are defined as C/C++ user defined functions. These functions can be compiled, debugged and executed in Petalinux user space on the 64bit ARM A53 processor. But in addition, the selected C/C++ functions can be compiled also to form of Vivado HLS HW accelerators. Blocks

are compiled by the Vivado HLS compiler and automatically interfaced with dedicated data movers like DMA or SG DMA.

SW projects include synchronisation of the accelerated video processing with the input and output Full HD HDMI 60FPS video streams.

### 7.6.2 Inputs:

- The Xilinx SDSoC 2018.2 compiler requires the board support package for the TE0808-03-15EG-1EE on TEBF0808 carrier.
- Predefined SW projects with algorithms coded in C/C++ for the Arm A53.
- User defined new SW projects with algorithms in C/C++ for the Arm A53.

### 7.6.3 Outputs:

- Compiled bitstream with HW accelerator(s) and SW boot files serving for programming of TE0808-03-15EG-1EE on TEBF0808 carrier. from the SD card
- Compiled SW project for Arm A53 with integrated HW accelerator(s) in form of executable code for the TE0808-03-15EG-1EE on TEBF0808 carrier with Debian OS.

### 7.6.4 Usage within FitOptiVis

This design time resource serves to the FitOptiVis partners as supported set of reference SW examples for the TE0808-03-15EG-1EE on TEBF0808 carrier.

- Partners can modify provided reference SW examples with own code and compile to HW with the SDSoC 2018.2 compiler for the the TE0808-03-15EG-1EE on TEBF0808 carrier with Debian OS and video I/O via the Imageon FMC card.
- Matrix multiplication and addition of int32 [400x400] matrices is elaborated in detail. Generated HW accelerator runs with 287,5 MHz clock provides acceleration 282x over the SW running on 1.05 GHz Arm A53.

This design time resource includes description of demo performing video processing with HW acceleration is described in this section. We demonstrate the LK Dense Optical Flow (LK DOF) algorithm with Full HD HDMI video input and video output.

The algorithm works with two subsequent Full HD frames. It computes for each pixel of the frame vector characterizing the direction and the speed of movement of a given pixel relative to its background.

The LK Dense Optical Flow algorithm involves massive fixed point computation and also floating point matrix inversion computed for each pixel of the frame.

The fixed point moving sum of the pixel background is computed for a square area 53x53 pixel. Computation is performed for each pixel of each video frame.

The VDMA is part of the initial platform and serves for the Full HD HDMI 60fps video in and the Full HD HDMI 60fps video out via the Imageon FMC card.



Figure 23: HW accelerated LK DOF, Full HD HDMI 60 fps

Figure 22 present set-up for computation of the LK Dense Optical Flow input movie with Full HD HDMI input 60 FPS from the PC and output in Full HD HDMI to the HDMI monitor. Table 4 is summarizing the performance of HW accelerated implementation and also the load of two most utilized Arm A53 processors.

Table 6: Performance of HW accelerated LK FOF and load Arm A53 processors.

LK DOF algorithm with per pixel integral tile size [53x53]	Frames per second	A53 SDSoC CPU load	A53 DeskTop CPU load	Acceleration of LK DOF
In SW	0.0907	100%	3%	<b>1x</b>
In HW DMA	60	100%	80%	<b>661x</b>
In HW SG DMA	60	30%	80%	<b>661x</b>

This acceleration is reached with design using only:

- 15% of BRAM (block rams) of the PL logic and no ULTRA RAM
- 14% of CLB (logic block tiles)
- 2% of DSP resources

This indicates the potential of the large TE0808-03-15EG-1EE module in the area of video processing algorithm acceleration.

## 8 Conclusions

Deliverable D3.1, “Design time, optimisation, deployment and programming strategies V1”, describes the status of design time methodologies, frameworks and strategies developed by project partners in Y1 of the project.

The D3.1 collects results achieved in tasks T3.1 “Model-driven engineering techniques for energy performance and other qualities” in Chapter 4, results achieved in T3.2 “Programming and parallelization support” in Chapter 5, and results achieved in T3.3 “Accelerator support” in Chapter 6.

It was important for the project to document the re-usable design time methodologies and tools released after the Y1 of the project in form of publicly accessible evaluation packages and documented in publicly accessible detailed application notes.

This support material goes beyond the “standard” university evaluation boards. The released design time support is provided for the commercial modules and evaluation boards.

Partners documented how to provide design time support for families of custom modules with Zynq and Zynq Ultrascale+ devices with different sizes and different carrier boards.

Team of project partners involved in WP3 released and documented in Chapter 7 publicly available resources [7.1], [7.2], [7.3], which serve not only to the FitOptiVis project partners, but also serve as support material which can be used by other developers outside the project.



## 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. Mutillo, 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 and description Deep Learning network.
- [4.13] <https://github.com/CMU-Perceptual-Computing-Lab/openpose> - CMU OpenPose network for recognition of human pose and gestures.
- [6.1] 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
- [6.2] 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>
- [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](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](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](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] Trenz Electronic, "TE0726 TRM," [Online].  
<https://shop.trenz-electronic.de/en/27229-Bundle-ZynqBerry-512-MByte-DDR3L-and-SDSoC-Voucher?c=350>
- [7.6] Documents for Arrowhead Framework  
Available:[https://forge.soa4d.org/docman/?group\\_id=58](https://forge.soa4d.org/docman/?group_id=58)
- [7.7] Trenz Electronic, "MPSoc Module with Xilinx Zynq UltraScale+ ZU4EV-1E, 2 GByte DDR4 SDRAM, 4x5cm", [Online].  
<https://shop.trenz-electronic.de/en/TE0820-03-04EV-1EA-MPSoc-Module-with-Xilinx-Zynq-UltraScale-ZU4EV-1E-2-GByte-DDR4-SDRAM-4-x-5-cm>
- [7.8] Trenz Electronic, "UltraSOM+ MPSoc Module with Zynq UltraScale+ XCZU15EG-1FFVC900E, 4 GB DDR4", [Online].  
<https://shop.trenz-electronic.de/en/TE0808-04-15EG-1EE-UltraSOM-MPSoc-Module-with-Zynq-UltraScale-XCZU15EG-1FFVC900E-4-GB-DDR4?c=450>
- [7.9] Trenz Electronic, ""=UltraTX+ Baseboard for Trenz Electronic TE080X UltraSOM+” [Online].  
<https://shop.trenz-electronic.de/en/TEBF0808-04-UltraTX-Baseboard-for-Trenz-Electronic-TE080X-UltraSOM?c=261>
- [7.10] Trenz Electronic, “Carrier Board for Trenz Electronic 7 Series” [Online].  
<https://shop.trenz-electronic.de/en/TE0701-06-Carrier-Board-for-Trenz-Electronic-7-Series?c=261>
- [7.11] Lukas Kohout, Jiri Kadlec, Zdenek Pohl: Video Input/Output IP Cores for Xilinx ZCU102 with Avnet HDMI Input/Output FMC Module , Application note and Evaluation package [Online].  
<http://sp.utia.cz/index.php?ids=results&id=zcu102-hio>