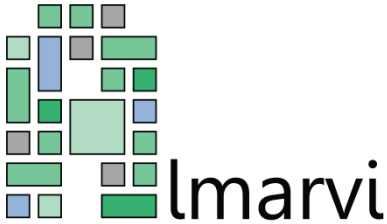


ALMARVI

"Algorithms, Design Methods, and Many-Core Execution Platform for Low-Power Massive Data-Rate Video and Image Processing"

Project co-funded by the ARTEMIS Joint Undertaking under the

ASP 5: Computing Platforms for Embedded Systems

ARTEMIS JU Grant Agreement n. 621439

D5.7 Evaluation of the ALMARVI Demonstrators

Due date of deliverable: March 31, 2017

Start date of project: 1-4-2014	**Duration:** 36 months
Organisation name of lead contractor for this deliverable:	PHILIPS
Author(s):	Steven van der Vlugt (PHILIPS), Keijo Haataja, Pekka Toivanen, Billy Braithwaite (UEF), Toygar Akgün (ASELSAN), H. Fatih Uğurdağ, V. Emre Levent (OZYEGIN), Lukáš Maršík (CAMEA), Antti Väänänen, Vili Härkönen (HURJA), Janne Keränen (VTT), Zdeněk Pohl (UTIA), Heikki Berg (NOKIA), Markus Turtinen (VISIDON), Zaid Al-Ars (TUD)
Validated by:	Zaid Al-Ars (TUD)
Version number:	V1.0
Submission Date:	18-04-2017
Doc reference:	D5.7 Evaluation of the ALMARVI Demonstrators_V1.0.docx
Work Pack./ Task:	WP5 Task 5.4
Description: ***(max 5 lines)***	This document describes the evaluation of the demonstrators in ALMARVI and discusses the results related to Deliverables 1.1 and 1.4

Nature:	**R**		
Dissemination Level:	**PU**	Public	**X**
	PP	Restricted to other programme participants (including the JU)	
	RE	Restricted to a group specified by the consortium (including the JU)	
	CO	Confidential, only for members of the consortium (including the JU)	

DOCUMENT HISTORY

Release	Date	Reason of change	Status	Distribution
V0.0	29/11/2016	Initial outlines	Draft	CO
V0.1	07/02/2017	Updated outlines	Draft	CO
V0.2	16/02/2017	Rough draft for all partners	Draft	CO
V0.3	10/03/2017	Improved draft for all partners	Draft	CO
V0.4	24/03/2017	Reviewed individual partner contributions	Draft	CO
V0.5	30/03/2017	OZYEGIN/ASELSAN contribution	Draft	CO
V0.6	31/03/2017	Finalized partner contributions and submitted for internal review	Draft	CO
V1.0	18/04/2017	Approved for submission to ARTEMIS JU	Final	PU

Table of Contents

Glossary

Abbreviation / acronym	Description
APCP	ALMARVI Python 1300 Camera Platform. Hardware platform with camera and HDMI output shared between ALMARVI partners.
ARM	Advanced RISC Machine, hard core processor with RISC architecture. Used as the PS on the Xilinx Zync device.
CCD	Charge-coupled device sensor. Another type of image sensor of cameras.
CMOS	Complementary Metal–Oxide–Semiconductor sensor. Common type of image sensor of modern cameras.
CPU	Central Processing Unit. Typically an ARM or Intel processor that is mainly used for running the host/control code of ALMARVI applications.
CSDF	Cyclo Static Dataflow Graphs
DFU	Dataflow unit
DMA	Direct Memory Access; IP providing access to shared DDR memory both from the PS as well as the PL
DTOCS	Distance Transform on Curved Space; is used in breast cancer analysis for image segmentation/classification purposes
Dyplo	DYnamic Process LOader: a middleware solution to seamlessly integrate software and hardware.
EdkDSP	UTIA execution platform component capable to execute rapidly reconfigurable hardware acceleration of floating point vector based operations. It consists of floating point DFU with programmable microcontroller. It also includes its own C compiler and API.
FPGA	Field Programmable Gate Array. Programmable hardware.
FPS	Frames Per Second.
GPU	Graphics Processing Unit. Nowadays general purpose programmable high performance processors originally designed for graphics rendering.
HDL	Hardware Description Language, VHDL or Verilog
HDR	High Dynamic Range of images.
HDMI	High-Definition Multimedia Interface
HLS	High Level Synthesis
IBCAS	Intelligent Breast Cancer Analysis System
IP	Intellectual Property; or a packaged HDL implementation
LP	Licence Plate of vehicle.
LRD	Local Rank Distance. A novel distance measure method.
LVDS	Low-voltage Differential Signaling. Serial communication standard, usually used for interfacing imaging sensors, etc.
OpenCL	Open Computing Language. An open heterogeneous programming standard that is used as a core of the system software stack of the ALMARVI project, thus also drove the design of the hardware integration interface.
OS	Operating System.

Abbreviation / acronym	Description
PCB	Printed Circuit Board
PL	Programmable Logic area of Xilinx Zynq device.
PR	Partial Reconfiguration
PS	Processing System in Xilinx Zynq device. Two ARM Cortex A9 processors, with DDR3 memory controller and memory interfaces and ARM peripherals.
RISC	Reduced Instruction Set Computing
ROC	Receiver Operating Characteristic curve. A graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied.
ROM	Read only memory
rVex	Reconfigurable VLIW Example; a reconfigurable and extensible Very-Long Instruction Word (VLIW) processor that is part of the overall "Liquid Architectures" research theme within the Computer Engineering Lab at TUDelft, The Netherlands
SDF	Static Dataflow Graphs
SDSoC	Xilinx tool for Software Defined System on Chip
vDMA	Video DMA; IP from providing video frame buffer handling and streaming
XML	Extensible Markup Language
zDMA	Zero copy DMA; a pointer is passed on from PS to PL to access the shared memory opposed to normal DMA where data is copied from PS allocated memory to PL allocated memory

1. Introduction

The overall objective of the ALMARVI project is to provide foundation and platform solution to enable massive data rate image/video processing at low power budgets under variability conditions. The key is to leverage joint hardware-software adaptations and properties of image/video content and algorithms.

This deliverable "D5.7 Evaluation of the ALMARVI Demonstrators" is concerned with the integrated and rigorous evaluation of all demonstrators for performance and power consumption, where recommendations on the ALMARVI implementation and usage through the different application domains are analysed and evaluated against the overall project objectives.

The various ALMARVI application scenarios from the three application domains (healthcare, security/ surveillance/ monitoring, and mobile) are considered. To demonstrate and validate the project developments and results, the ALMARVI concepts will be evaluated using demonstrators from three different application domains:

1) Healthcare demonstrator: medical imaging assisted diagnosis with a mixture of real-time and non-real-time image-processing tasks for different healthcare applications like minimal invasive treatment for cardiovascular diseases.
2) Security/Surveillance and Monitoring demonstrator: distributed monitoring using mobile and fixed video sensor nodes; continuous monitoring of industrial processes.
3) Mobile demonstrator: novel ultra energy-efficient high performance heterogeneous multicore platform for mobile applications and nomadic embedded devices of the future.

For each domain, the common researched concepts for algorithms, design tools, system software stack, and many-core execution platform are described. We also show how the various demonstrators achieve their target requirements as defined in detail in deliverable D1.1 at the beginning of the project. The investigations of the various partners in the ALMARVI project target the specific objectives of the ALMARVI project: 1. Massive data rate, 2. Low power, 3. Composability, 4. Robustness. Each partner targets specific aspects of these objectives as discussed in this deliverable using the example objective coverage table shown below.

Coverage of Almarvi Objectives		Application domain	
		Partner 1	Partner 2
1	Acceleration fabrics	■	■
	Design tools	■	
	Application specific parallelization	■	■
	Quality - performance trade-off		
2	Low power cores		■
	SW quality - power trade-off		■
	HW quality - power trade-off		
	Algorithm resilience for power efficiency		
3	Software portability	■	■
	Interfaces for heterogeneous acceleration fabrics	■	
4	Guarantee system performance	■	
	Guarantee power consumption		

Table 1: Example coverage table of the Almarvi objectives by the various partners (dark entry = covered)

The specific objectives are as follows:

1) **Objective-1 – Enabling Massive Data Rate Processing:** ALMARVI enables massive data rate image/video processing on embedded devices executing advanced high-efficiency image/video processing algorithms. The goal is to develop:
   (i) adaptive many-core execution platforms that are scalable with heterogeneous acceleration fabrics (like FPGAs, DSPs, GPUs, etc.);
   (ii) application-specific image/video processing cores and coprocessors;
   (iii) design tools to expedite the development flow and to enable the IP reusability;
   (iv) application-specific parallelisation; and
   (v) methods for joint hardware-software adaptations based on algorithm resilience properties.

2) **Objective-2 – Achieving Low-Power Consumption:** ALMARVI aims at enabling cross-domain power-efficient techniques and methods for efficient and lightweight resource and power management at various system layers, while jointly accounting for the architectural features, algorithmic properties, and image/video content characteristics. The goal is to develop:
   (i) low-power architectures of image/video processing cores;
   (ii) low-power adaptive algorithms with run-time quality vs. energy trade-off;
   (iii) novel hardware-software-collaborative power-management techniques; and
   (iv) methods to exploit algorithmic resilience for increased power-efficiency.

3) **Objective-3 – Composability and Cross-Domain Applicability:** To enable independent development of various system components and smooth integration for demonstrators from different domains. One of the key objectives of ALMARVI is to provide a cross-domain scalable platform solution with efficient design tool chain, IP reuse, composability, and system software stack for seamless interoperability, and scalability on commercially available heterogeneous acceleration fabrics.

4) **Objective-4 – Robustness to Variability:** ALMARVI targets consistent and predictable system performance and power consumption over different product categories and application domains that are subjected to variability in underlying processing hardware, communication channels, application workload behaviour, system state (available resources and energy budgets), environmental factors, etc. To achieve this, the goal is to devise power-aware scalability and adaptivity at the algorithm and system levels while exploiting inherent resilience properties of image and video processing applications for adaptive resource and power management.

General objectives are achieving low-cost solutions, high product quality, lifetime, high yield, and high engineering efficiency which will be accomplished through design tools, efficient algorithm designs, scalable and adaptive execution platforms, support for interoperability, etc.

2. Healthcare

2.1 Introduction

In the healthcare domain, ALMARVI has developed two main demonstrators: Interventional X-Ray and Breast Cancer Diagnostics. This chapter describes the evaluation of the demonstrators.

The healthcare demonstrators address the ALMARVI objectives as indicated in Table 2. The demonstrators use acceleration fabrics and parallelization to achieve image processing at the required performance level. The X-Ray demonstrators presented in section 2.2 use image processing on FPGA that has been implemented using high level synthesis tools in a product with a legacy image processing pipeline. Furthermore a challenging image processing algorithm has been ported from Matlab to C++ and then implemented on an FPGA platform and a PC CPU and GPU with OpenCL. The Breast cancer diagnostics demonstrator presented in section 0 is based on a Matlab implementation which was ported to C and implemented on a combination of a PC CPU and GPU with OpenCL, furthermore a C++ implementation of the application was created and was tested on TUDelft's rVex.

These demonstrators illustrate the applicability of the novel concepts developed and evaluated in the ALMARVI project. Through different demonstrators from diverse application domains and product categories, this deliverable illustrates the validation of ALMARVI project concepts in algorithms, design methods/tools, and execution platforms.

	Coverage of Almarvi Objectives	Healthcare	
		Interventional X-Ray (PHILIPS)	Breast Cancer Diagnostics (UEF)
1	Acceleration fabrics	■	■
	Design tools	■	
	Application specific parallelization	■	■
	Quality - performance trade-off		
2	Low power cores		■
	SW quality - power trade-off		■
	HW quality - power trade-off		
	Algorithm resilience for power efficiency		
3	Software portability	■	■
	Interfaces for heterogeneous acceleration fabrics	■	
4	Guarantee system performance	■	
	Guarantee power consumption		

Table 2: Objectives for the Healthcare demonstrators

2.2 PHILIPS: Interventional X-Ray

The goal of an interventional X-ray system is to provide the doctor with real-time images from the anatomy of the patient while performing a medical intervention. Typical interventions on the system include repairing blood vessel deformations such as aneurisms by positioning stents or replacing heart valves. During these procedures, blood vessels are filled with a contrast medium, which is visualized by X-rays and shown in real time as high resolution video images to the doctor.

The interventional X-ray system (see Figure 1) contains functions like the acquisition and display of an X-ray image series, the movement control of patient support and X-Ray gantry, patient administration, communication to hospital servers and many others. Though some tasks may be performed on a “best effort” base (for instance: patient administration, export of an image series to the hospital servers), others must satisfy strict real time requirements. Handling images during acquisition and movement control belong to the latter category.

Failure to meet movement control timing requirements may lead to injury to or even death of a patient, for example when the detector hits and hurts the patient. Another critical parameter is the image pipeline latency, which is the time between the irradiation of the body with X-rays and the display of the corresponding image on the display. In general no delay is perceived if the latency is 150ms or less. If the latency is significantly larger, the hand-eye coordination of the doctor is disrupted and treatment of the patient is impaired.

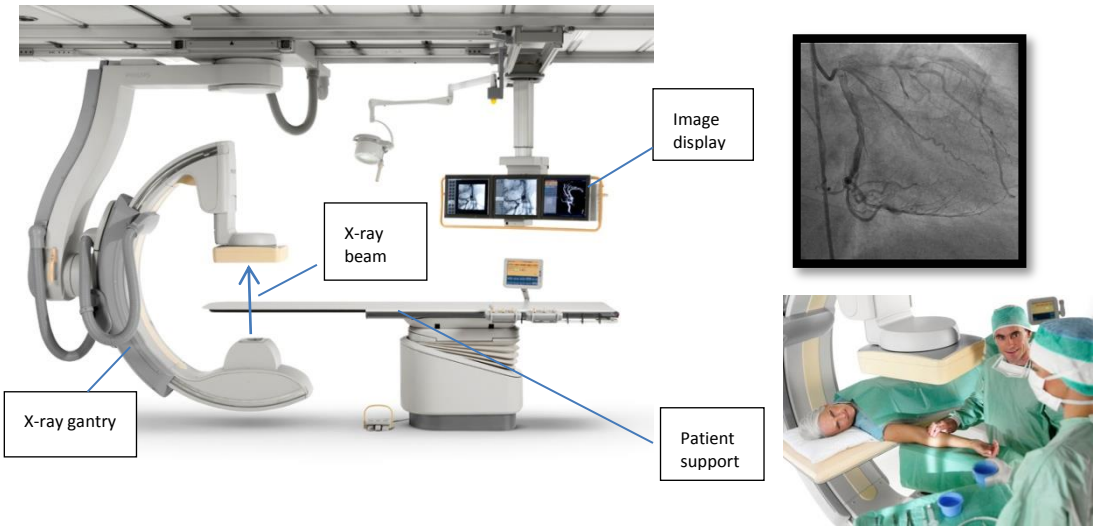


Figure 1: An interventional X-ray system and a typical still from a X-ray video. In the inset the patient is on the patient support while an interventions is performed.

The X-ray dose applied to the patient is chosen as low as reasonably possible, which increases noise and reduces contrast in X-ray images. Therefore, image processing is required to expose details to the doctor that are hidden in the original images. Typically, noise reduction and contrast enhancement are applied. For later review, images are recorded and stored in the system.

2.2.1 Introduction

This use case describes the procedure to acquire X-ray images during an interventional examination and focuses on the processing of images.

The interventional system is provided with one or more PCs, which accommodate various accelerators like processor cores, GPUs, FPGAs or DSPs. We observe a continuous proliferation of new accelerators for image processing in de form of advanced processor architectures, GPUs, DSPs and FPGAs. At the same time, we see that introduction of these devices in a product come with significant development efforts and long time to market. The reason for this is that each accelerator is programmed in its own programming language and library environment. The choice of design methodologies and tooling affect this development effort. Developments such as in OpenCL, carry the promise for a single, unified image processing code base for all types of accelerators.

Therefore, the main research question (Q1) is to investigate a system (processing pipeline) design that supports our clinical use case and that can be readily adapted to various accelerators for the image processing.

In this way, the system design can be adapted to the most economical accelerator during the life cycle of the design with a minimal development effort. Systems delivered to our customers may be upgraded in the field with more advanced accelerators for improved functionality or performance. This research question relates to ALMARVI Objectives 1 and 3.

The second research question (Q2) is to support this use case by as few as possible components for reasons of reliability and economy. The ultimate goal in this respect is characterized by “the single box” concept: a single PC provided with economical and reliable accelerators to support this use case and all other functionality of the system. This research question relates to ALMARVI Objectives 3 and 4.

Costs might be reduced by:

Q2a) Combining multiple functions on one PC, e.g. combine control of the gantry on the same PC as the patient database and viewing suite or combine image processing and geometry control on a single FPGA.

Q2b) Using the capabilities of newer, typically cheaper and more powerful, processing hardware, e.g. offloading image processing to the GPU or running it on an onboard FPGA.

Q2c) Reduce the development effort for innovative accelerators.

The current system design is based on a complex combination of PC based nodes, and additional nodes. For this study we take the baseline performance figures shown in Table 3 below. Figure 2 shows a generalized image processing pipeline from X-Ray source to Display.

Table 3: Baseline (current) and desired imagine pipeline performance values

Parameter	Baseline	Desired values
Image size	1024x1024 at 16 bit	1024x1024 at 16 bit
Frame rate	30 frames/s	60 frames/s
Latency	200 ms	100 ms
Maximum Jitter	20 ms	10 ms

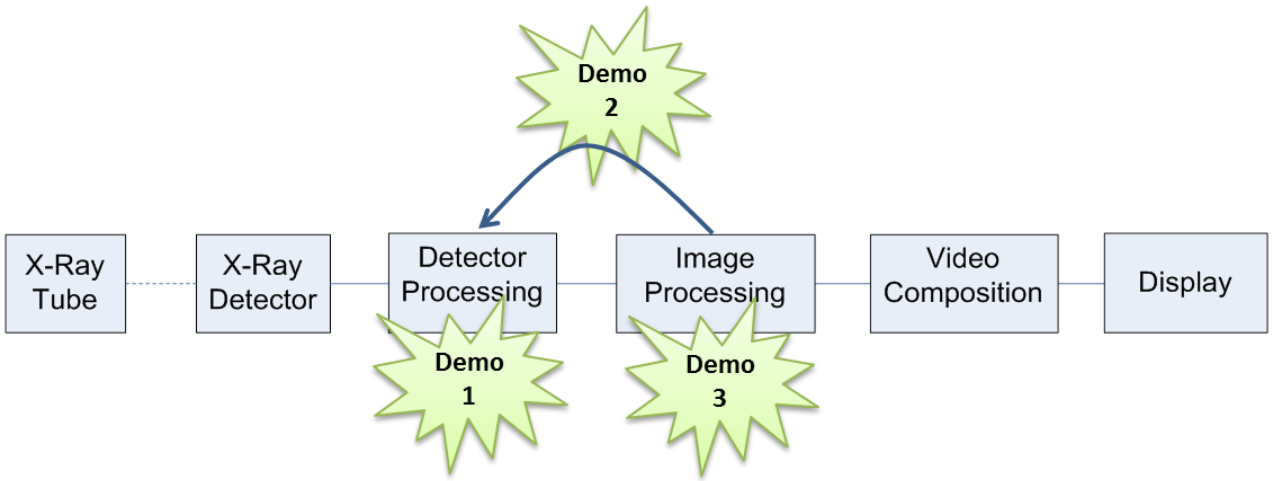


Figure 2: Generalized X-Ray image processing pipeline

The above defined research questions are addressed with three demonstrators:

- Demonstrator 1 is a redesign of a processing step in an FPGA based image processing pipeline. This demonstrator addressed our research questions by studying:
  - Reduction of development effort for innovative FPGA accelerators;
  - Comparing commercially available tools for High Level Synthesis of FPGA accelerators;
  - Using these accelerators in a legacy image processing pipeline;
- Demonstrator 2 is a FPGA implementation of a PC noise reduction algorithm, representative for the real system implementation. This demonstrator addressed our research questions by studying:
  - Tools for software to hardware portability (High Level Synthesis);
  - Reduction of development effort for innovative FPGA accelerators;
  - An integral tool flow or way of working;
  - A commercially available middle ware solution for the integration of (embedded) software and hardware accelerators in a combined image processing pipeline;
  - Combining multiple functions on a single device;
  - Modelling and Analysis techniques to answer design time questions;
  - Performance and challenges of a FPGA based implementation;
- Demonstrator 3 is an OpenCL implementation of the same noise reduction algorithm as in Demonstrator 2, starting from a common code base. This demonstrator addressed our research questions by studying:
  - Software portability to multiple PC based platforms;
  - An integral tool flow or way of working;
  - Combining multiple functions on a single device;
  - Performance and challenges of an OpenCL implementation;

Figure 3 below shows the structure of the use case for Demonstrator 2 and 3; a Matlab model of a noise reduction algorithm was manually converted to C++, no algorithmic optimizations were made. Next the common C++ implementation was used to make a FPGA and PC based (CPU or GPU) implementation. Demonstrator 2 was split up in a) a full implementation of the algorithm in fixed-point software on the Zync ARM processor (PS) with the 4 most compute intensive processing steps accelerated on the Zync hardware (PS) in the Dyplo network and b) a scaled down version of the algorithm fully implemented on the Zync hardware (PL) in the Dyplo network to isolate the design time challenges for a complex streaming hardware implementation and to verify dataflow models and analysis done by TUE.

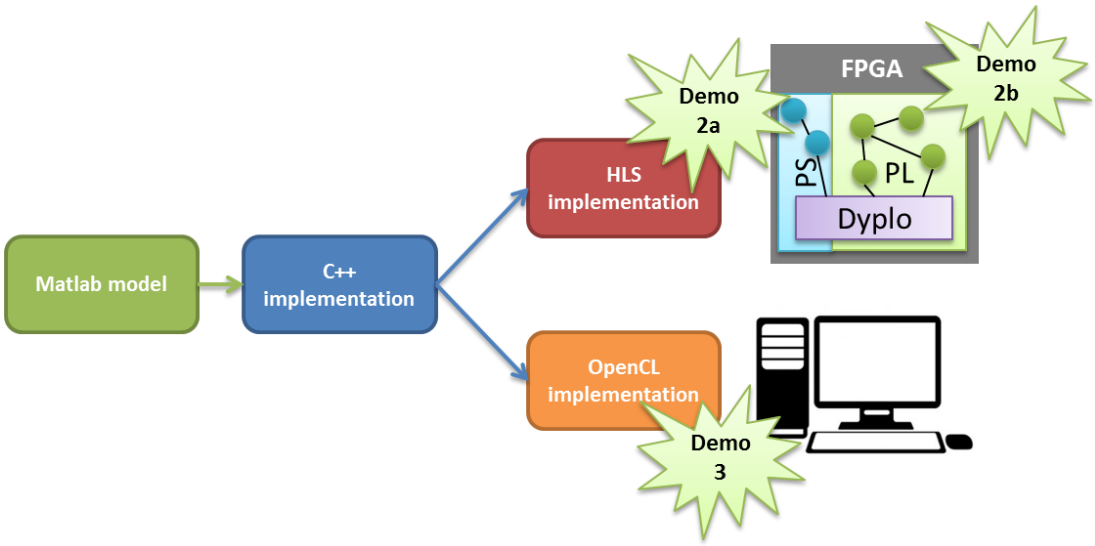


Figure 3: Use case structure for Demo 2 and 3, a FPGA and PC based (CPU + GPU) implementation were made, starting from a common C++ implementation

2.2.2 Requirements

We have studied tools and techniques for medical image streaming, processing and displaying for 1) portability across PC platforms 2) to enable portability towards FPGA's and 3) to reduce the development time for FPGA's. We have deviated from our requirements stated in D1.1 in the sense that we did not address the storage of images as was part of functional requirement 1. All other functional and non-functional recruitments were met or partially met with some small deviations as described below.

Functional requirements:

Requirement 1: Functions required

This requirement was mostly met, we chose to not implement storage for PC based nodes and instead focus more on (embedded) image processing nodes which support image streaming, processing and displaying. Demo 1 incorporates image streaming and processing; Demo 2 incorporates image streaming, processing and displaying; Demo 3 incorporates image processing and displaying.

Requirement 2: Supported functional processing blocks

This requirement was met, but motion compensation was replaced by noise reduction which exposes similar image processing challenges. Demo 1 incorporates 2D interpolation and image algebra; Demo 2 and 3 incorporate resolution up- and down- scaling, spatial and temporal filtering, noise reduction, grayscale conversion and image algebra. The results of these components were verified with a reference implementation and visual inspection.

Requirement 3: Change pipeline configuration at runtime

This requirement was labeled as nice to have and was partially met in a use case (Deliverable 4.4) where we explored the applicability of Dyplo for run time re-routing and re-configuration of image processing on a SoC Zync FPGA.

Requirement 4: Adaptability for performance

This requirement was labeled as nice to have and was partially met by Demo 2a where we show the difference between an embedded system with a Zync SoC with a software only application and a software application with hardware acceleration. The hardware acceleration requires more development time but drastically reduces the

latency and jitter. The measurements on the execution time of the SW only application and the application with parts accelerated in HW have been presented in D5.2 Table 3. A comparison for Jitter is presented below in Table 4 where we show a comparison of jitter for the software only application on the ARM PS and the software application with parts accelerated on the FPGA PL for 100 runs of both applications. The amount of jitter on the software side is significant and can on average be improved by at least 4x by moving to an implementation with the nodes accelerated on the PL. When moving to a hardware only implementation we will be able to reduce the jitter even further, since the PL implementation can be made hard real time.The latency of the application is also reduced by accelerating parts in the hardware first of all due to a speed up of the processing time. Second, by moving to a full hardware implementation we can reduce the total latency even further. Overall using an embedded real-time platform based on an FPGA will by nature reduce the latency and jitter. Also run time re-routing and re-configuration with Dyplo might be exploited to switch to lower image quality algorithms when higher performance is desired or vice versa.

Processing step	SW jitter [ms]			HW jitter [ms]			improvement [ratio]		
	min	max	avg	min	max	avg	min	max	avg
P1	0.00	47.94	4.48	0.00	5.64	1.04	1.00	8.51	4.33
P2	0.00	14.43	4.86	0.00	5.53	0.87	2.00	2.61	5.58
P3	0.04	719.81	85.95	0.00	0.09	0.02	inf.	7739.91	5289.27

Table 4: Comparison of jitter for software only and the software application with parts accelerated on the FPGA PL

Non-functional requirements

Requirement 1: Adaptability to new architectures

This requirement is partially met. We have defined a tool flow to manually go from an algorithm in Matlab to a C++ implementation and then from there on use the same code base to make 1) an implementation with OpenCL for portability to CPU and GPU platforms as demonstrated with Demo 3; 2) an implementation with Vivado HLS to an FPGA platform as demonstrated in Demo 2. Hence the same code base was used to build two implementations for three different architectures. For the OpenCL implementation we could effortlessly switch between a CPU and GPU platform (Demo 3), porting to the FPGA platform still involved much manual effort to reach good performance with the HLS tools (Demo 2).

Requirement 2: Economical and reliable platforms

This requirement is met. With Demo 1 we demonstrate that we were able to integrate a processing step, designed with HLS tooling, in our legacy FPGA based image processing pipeline. With Demo 2 and 3 we demonstrate that we can port a PC based algorithm to an FPGA platform. With these techniques it will become possible to merge image processing from multiple nodes in a single node thus reducing the complexity and costs of the system and improving the reliability. The processing step integrated in Demo 1 runs at full performance, Demo 2 and 3 almost meet the desired performance and we see good opportunities for improvement; the current implementation is based on a naïve port of the original Matlab algorithm, we only performed platform specific optimizations such as data movement and scheduling, algorithmic optimizations will improve the performance by simplifying and reducing the arithmetic operations, also there is still room to increase the clock frequency of the FPGA for Demo 2.

Baseline & Challenges

With Demo 1 we were able to meet the desired frame rate with low processing latency and jitter as reported in Deliverable 5.2 section 2.2.2 thus demonstrating the feasibility of FPGA based image processing with HLS tooling. For Demo 2 and 3 we used a somewhat smaller frame size because that better matches the available clinical recordings. Demo 2a currently does not meet the desired frame rate as reported in D5.2 section 2.3.2; however we expect that by moving to a hardware only implementation (as in Demo 2b) we can reach the desired frame rate, as was further explained in D5.2 section 2.3.2. With Demo 2a we also demonstrate that a hardware

implementation is beneficial for the reduction of jitter as discussed with Functional Requirement 4. Demo 2b almost meets the desired frame rate as reported in D5.2 section 2.4.6 and we see good opportunities for improvement by increasing the clock frequency of the FPGA. Demo 3 almost meets the desired frame rate as explained in D5.2 section 2.5.2, also as explained above for Non-Functional Requirement 2 we expect to reach a higher frame rate by performing algorithmic optimizations. The overall system latency can be reduced by combining multiple nodes, by moving from PC nodes to embedded nodes such as FPGA's and by using stream based processing instead off frame based processing; with frame based processing every node first needs to buffer a full frame before it can start processing, meaning that even very fast processing nodes add one frame-time latency to the total system latency. These three solutions were part of our research questions and have been addressed with our demonstrators. Not every Demo meets all requirements, this is acceptable because we have built different demonstrators to address different challenges. Overall we can conclude that we have met our desired image pipeline performance values.

Table 5: Baseline, desired imagine pipeline performance values and results for each demonstrator with '-'below, '=' equal to and '+' above requirements

Parameter	Baseline values	Desired values	Achieved values			
			Demo 1	Demo 2a	Demo 2b	Demo 3
Image size	1024x1024 at 16 bit	1024x1024 at 16 bit	1024x1024 at 16 bit	960x960 at 16 bit	960x960 at 16 bit	960x960 at 16 bit
Frame rate	30 frames/s	60 frames/s	++	--	-/=	-/=
Latency	200 ms	100 ms	++	++	++	+
Maximum Jitter	20 ms	10 ms	++	++	++	=/+

2.2.3 Objectives

Objective-1 – Enabling Massive Data Rate Processing

In relation to this objective we have developed custom accelerators for FPGA's with High Level Synthesis design tools. This was done both for an existing imaging pipeline operating at very high data rates (Demo 1) and for new architectures as a proof of concept (Demo 2). We started out with an evaluation of two currently commercially available HLS tools, in order to select the right tool for our needs. These results were presented in D3.2. Also we presented Vivado HLS experiences and optimization guidelines in D3.4. Next to that we have developed a tool flow from a Matlab Algorithm through Microsoft Visual Studio to HLS or OpenCL in order to make the tools more accessible to software programmers, to make the Algorithm more portable and to be able to use the same test framework throughout the whole development process. This tool flow was presented in D4.1. With this tool flow we have developed a Fixed-point Analyzer and Scaler Tool to aid us in floating point to fixed point conversion which is a desired step when moving to HLS accelerators. This tool was presented in D4.3 and results were presented in D4.2. Together with TUE we have developed modeling and analysis techniques to address design time questions related to throughput, latency and (buffer) resource optimization for custom accelerator networks, this work was presented in D4.2 and D5.2 and used in Demo 2b.

Image or video processing algorithm development is mainly done in a frame based manner which allows random access of the frame and parallelization techniques such as tiling. When moving to FPGA accelerators we cannot buffer a full frame before we start processing, due to amongst others memory bandwidth, power and latency requirements. Therefore the algorithm has to be implemented in a stream based manner where we wish to process pixels as soon as they come in and as quickly as possible pass the result on to the next processing step (accelerator). This involves intensive hand optimizations such as using line buffers and data re-ordering instead of random memory access. This work was done manually for Demo 2a and 2b. We have made a first effort to

automate this work or to hide the stream based processing from the programmer. In D3.7 we presented a simulation of a platform with several rVex cores targeted to a Zync FPGA to enable streaming processing programmed through OpenCL on FPGA's. More work is required in this area, but this is a nice start. Ideally we would like to decouple the arithmetic algorithm operations, the scheduling of these operations and the dataflow.

Application specific parallelization has been addressed with Demo 2 and 3 where we started off with the same C implementation, but different parallelization strategies were used for the HLS implementation and the OpenCL implementation, also many different parallelization strategies for the HLS accelerators were studied and compared, the results have been presented in D3.4.

Objective-3 – Composability and Cross-Domain Applicability

We have demonstrated software portability with Demo 2 and 3 where we started the development from a common code base for our algorithm towards thee different platforms used for the demonstrators. With HLS tooling we were able to implement the algorithm on an FPGA. With OpenCL we could implement the algorithm on both a CPU and a GPU (no platform specific optimizations). However there still is a gap between the OpenCL description and an implementation on the FPGA. With the streaming rVex platform simulation we have demonstrated a way to close the gap and both Xilinx and Intel have recently released tools to develop FPGAs with OpenCL, currently these tools only address the data center market where the host and FPGA communicate through PCI-E, but Xilinx has also announced to be working on a version of the tooling for Zync SoC type of devices with the ARM as a host. These kind of tools enable us to more easily switch between platforms and merge functionality from multiple nodes thus creating composability in our systems and reducing the complexity.

With objective 1 we have already discussed the creation of custom accelerators with HLS tooling. In order to use these kind of tools for complex algorithms we need to divide the algorithm in many small processing steps or nodes. These nodes are used as accelerators on the FPGA PL but are programmed and controlled with parameters from the PS. This exposes two problems 1) communication between the PS and PL, interfacing the acceleration fabric, is labor intensive and error prone 2) we need to connect the different nodes together, preferable in a flexible and composable way. The first problem can be address by using tools such as SDSoC from Xilinx which offers a complete environment for both the software and the hardware development and the communication between the PS and PL is automated in the tool. We address the second problem by using a network on chip which is part of the Dyplo middle ware solution. Dyplo is a development environment, a software API and an FPGA IP with network on chip and offers a software interface to FPGA accelerators. We have previously discussed this product in D1.5, D4.4 and D4.6 and it is used in Demo 2. The network on chip helps us to easily put the many different accelerator nodes together; the nodes can even be re-routed at run time thus offering high composability. Another useful feature offered by Dyplo is partial re-configuration of the FPGA where most of the FPGA infrastructure is static and only a designated area on the FPGA is programmable. This partial area can at run-time be programmed with a partial bit-stream. This way we were able to quickly test small changes to accelerators, building a partial bit-stream takes much less time than building the full bit-stream for the FPGA. Run-time re-routing and partial re-configuration might also be used to change the configuration of our image processing pipeline as was discussed in a use case in D4.4.

Objective-4 – Robustness to Variability

For real-time image and video processing in the healthcare domain it is crucial to guarantee system performance. With this objective we aimed to reduce jitter in the performance of our systems, reduce the latency and be able to give guarantees on the system design. As already discussed with the Baseline & Challenges we might reduce jitter and latency by moving from PC based platforms to embedded real-time platforms and by combining multiple nodes into one. Next to that we have developed models and analysis techniques together with the TUE to predict the throughput and guarantee deadlock free behaviour for complex networks of FPGA accelerators (already discussed with objective 1), which is an important step in the development of reliable embedded real-time platforms.

2.2.4 Research questions

We have presented our research questions in the introduction to this section. The research question were address with our Demonstrators, the results were addressed with the discussion of our requirements, baseline & challenges and objectives.

2.2.5 Validation of functionality

We have used different levels of validation:

- Demonstrator 1 was completely validated on all levels of the V-model including integration in an iXr system
- Demonstrator 2 and 3 served as feasibility study and were validated as such

2.2.6 Verification of robustness and reliability

We have used different levels of verification:

- Demonstrator 1 was completely verified based on component and system requirements
- Demonstrator 2 and 3 served as feasibility study and were verified on low level input and output specifications

2.2.7 Validation of performance metrics

The validation of performance metrics is presented in the Baseline & Challenges section of this document.

2.2.8 Lessons learned

In general we see good opportunities to reduce classical FPGA development time by using High Level Synthesis tooling and network on chip like solutions thus increasing the engineering efficiency and at the same time making designs better testable, increasing the composability and making designs easier to maintain. In the future we wish to abstract even further from FPGA designs and enable portability from OpenCL and other heterogeneous languages, such tools are currently entering the high performance computing market and we see good opportunities for these tools for embedded devices (SoC) as well.

Maturity of tools

The Vivado High Level Synthesis was used to develop our custom accelerators. This tool offers a fast way to build a demonstrator on Xilinx FPGAs. The use of the C/C++ test bench for simulation results in quick development cycles and can easily be extended to a software tool flow with for example Microsoft Visual Studio. We have used many different versions of this tool from early 2014 up until 2017. In 2014 the tool was still a bit buggy, but this has greatly improved, up to the point that it now has become a mature reliable technique. Vivado HLS still has its limitations; we see good applicability for image processing algorithms and single or multi-dimensional algorithms in general, however control logic seems more difficult to implement. Next to that a deep understanding of FPGAs is still mandatory to truly optimize the implementation, which was necessary to meet our requirements.

Development effort

Being able to use HLS tooling with C++ input already greatly reduced the development effort for the FPGA platform. We estimate a 4x speedup in development time for our kind of implementation, Xilinx reports up to a 15x speedup for less complex implementations. Next to that we could much more easily test the design, simulate the implementation and test the final implementation on the target all with the same test framework. We estimate that we gained more than a 4x speedup with HLS opposed to manual VHDL when making small algorithmic changes to the implementation.

Portability

AS described with objective 3 we aimed to be portable from a common C++ implementation to both FPGAs, with HLS tooling, and to CPUs and GPUs with OpenCL. Ideally we would like to develop directly in OpenCL and be portable to CPU, GPU and FPGA platforms from the same implementation. Both Xilinx and Intel have recently released tools to develop FPGAs with OpenCL, currently these tools only address the data center market where the host and FPGA communicate through PCI-E, but Xilinx has also very recently released a version of the tooling for Zync SoC type of devices with the ARM as a host. OpenCL tooling for FPGAs might also help to abstract further from the FPGA and overcome the currently needed hardware knowledge for HLS.

Network on Chip and Partial Reconfiguration

Using SoC kind of devices such as the Zync FPGA require the developer to work on both the software (PS) and hardware (PL) implementation as well as the communication (driver and bus) between the two. We have used Dyplo, a middleware solution from Topic Embedded Products, to reduce the complexity for the developer. This tool abstracts the PL for the user and offers a software experience for both the PS and PL development. It does this by generating a network on chip (the Dyplo IP) and implementing a run-time software library to use the accelerators in the software environment. Next to that it uses a tool flow (Dyplo Development Environment) to generate the FPGA implementation. This tool can be used to build FPGA accelerators from software, where it uses Vivado HLS to build the accelerators form a software description, and it builds the full FPGA implementation. Additionally Dyplo abstracts the partial reconfiguration tools from Xilinx, which allows the user to at run-time reconfigure accelerators in the network. This flow has proven to be very useful for us during development. Typically much time is spend on rework and incremental design for FPGAs because the complete bit stream of the FPGA has to be re-build for every small change. With partial reconfiguration and the Dyplo network we can easily change only a small part of the FPGA implementation, thus saving much time during development and providing flexibility for testing different configurations.

2.2.9 Conclusion

Our research questions, objectives and baseline & challenges were addressed with three different demonstrators each isolating several of our challenges. We have proven that our overall intended approach is feasible and can be of value to our system. Parts of the outcomes related to FPGA tooling are currently being used for development. We have started follow-up studies to further address the gap between OpenCL and FPGA development in order to take benefit of the abstraction offered by higher level languages.

2.3 UEF: Breast Cancer Diagnostics

During the ALMARVI project, UEF has focused on the creation of breast cancer analysis software that consists of several components that are integrated into our IBCAS environment on MATLAB. All components have been developed while keeping the project objectives in mind and aimed to meet them in their full potential.

2.3.1 Introduction

IBCAS (Intelligent Breast Cancer Analysis System) is a Matlab-based demo system for conducting image analysis experiments on histopathological breast cancer images. With this system, we have experimented image enhancement, feature extraction, segmentation, and classification methods for medical imagery. In addition, we have implemented an easy-to-use stand-alone C-based application of the system for demonstrating and concretizing how to utilize the research results in real-world medical analyzes without the need to perform any complex choices in selecting feature extraction, segmentation, or classification methods, i.e., “optimal” settings are automatically selected by the application. Moreover, we have also implemented an OpenCL-based version of the system that utilizes GPUs and its development is currently in its finalization phase. Furthermore, we have converted selected algorithms of our Matlab-based version of the IBCAS system into C/C++ for cooperation purposes aiming at running them on TUDelft’s rVEX platform, which is a reconfigurable and extensible Very-Long Instruction Word (VLIW) processor that is part of the overall “Liquid Architectures” research theme within the Quantum Engineering Lab at TUDelft, The Netherlands. The rVEX processor architecture is based on the VEX ISA. Figure 4 illustrates our Matlab-based version of the IBCAS system in action.

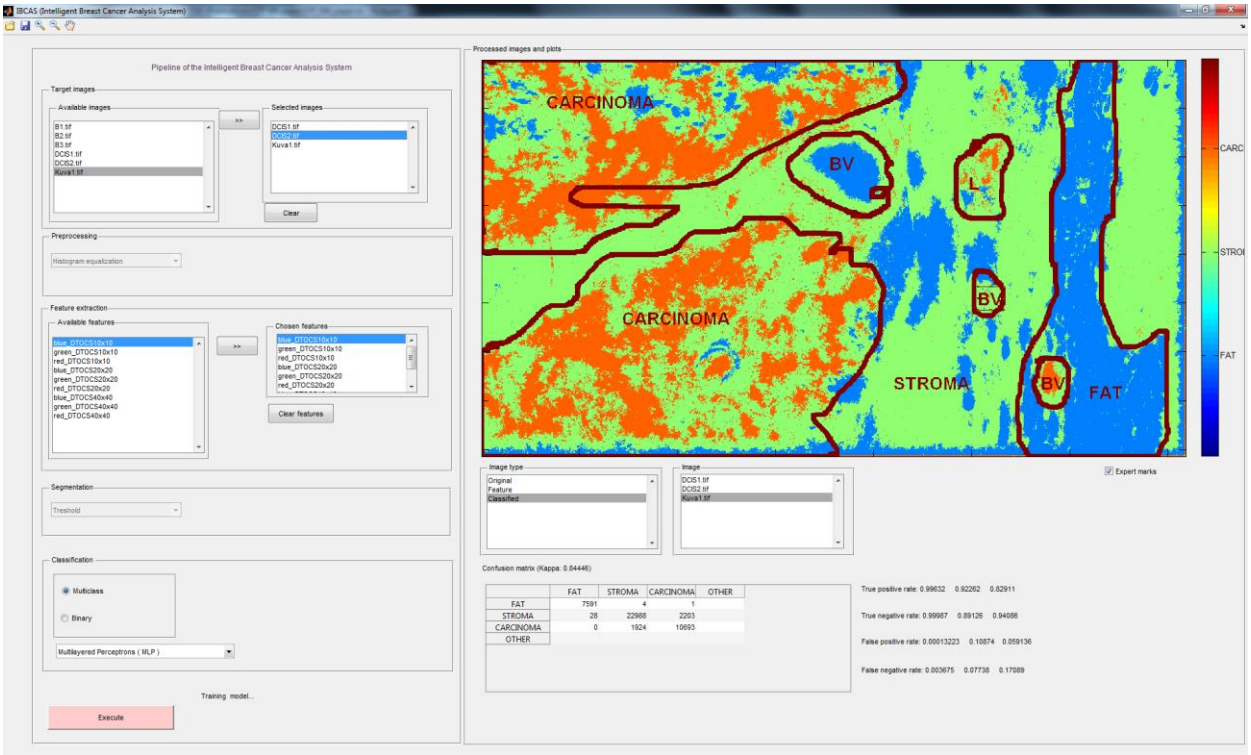


Figure 4: Matlab-based version of the IBCAS system in action.

Figure 5 illustrates our stand-alone C-based application in action.

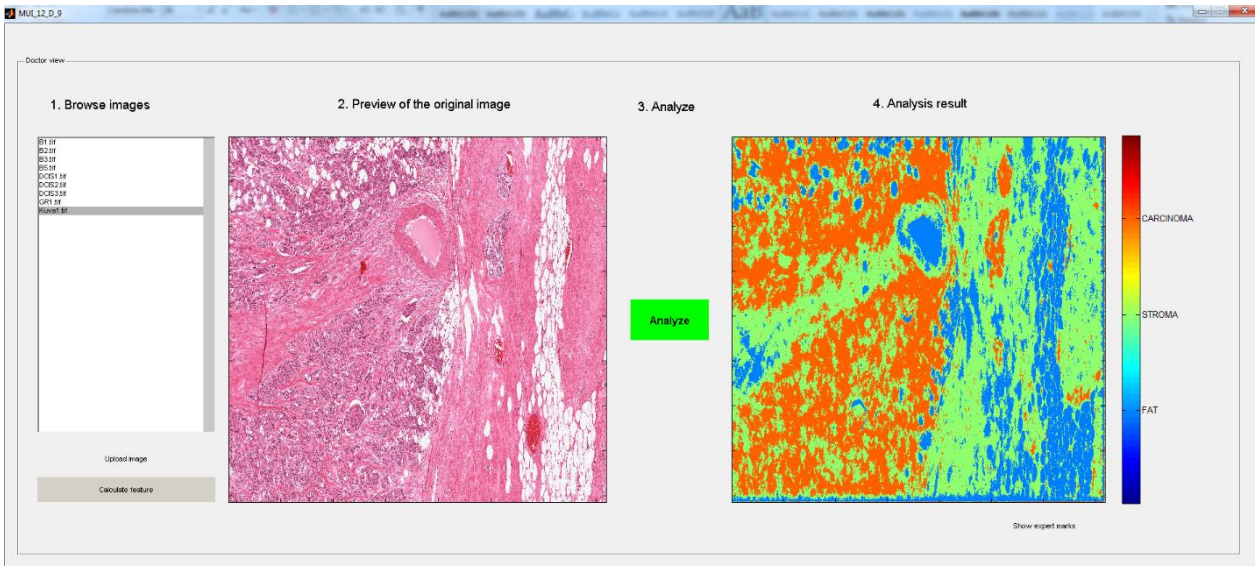


Figure 5: Stand-alone C-based application in action.

Figure 6 summarizes different SW components of the IBCAS system.

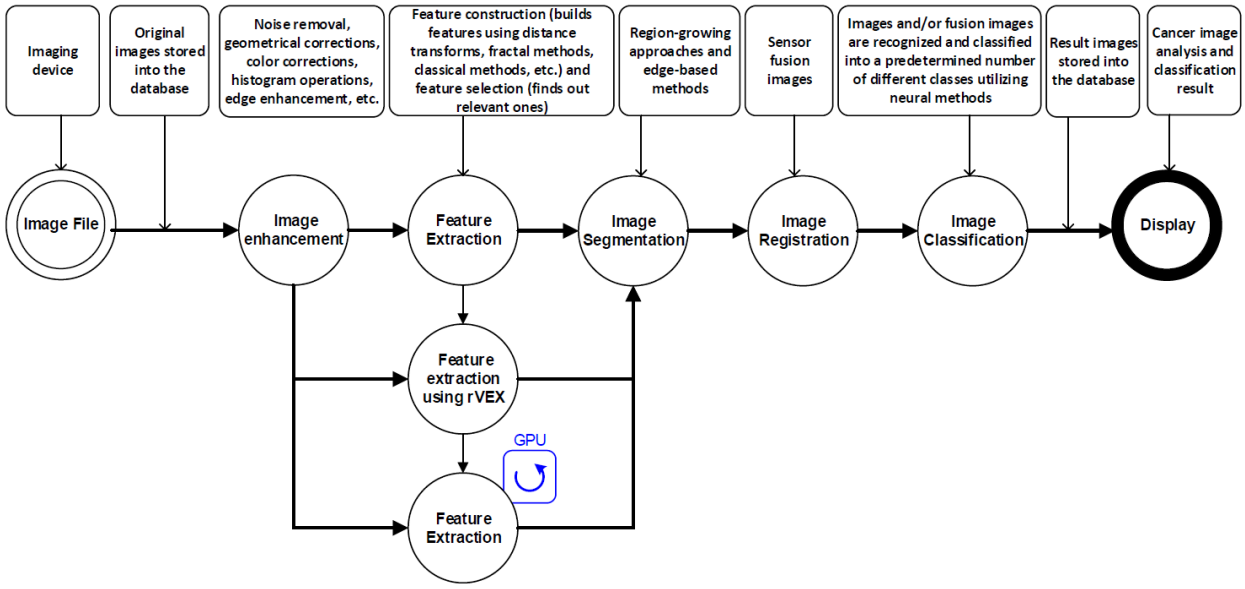


Figure 6: SW components of the IBCAS system.

We have used only off-the-shelf HW, i.e., typical PCs and laptops for running both Matlab-based versions of the IBCAS system and stand-alone C-based application as well as an efficient GPU-based laptop with large memory size for running OpenCL-based version of the IBCAS system. Moreover, our C/C++-based versions of the selected algorithms of IBCAS are currently being tested in TUDelft's custom HW based rVEX platform, which is a reconfigurable and extensible VLIW processor architecture that is already integrated into ALMARVI platform. Since our OpenCL-based version is currently in its finalization phase, it has not yet been integrated into ALMARVI platform, but the integration work will be ready before the end of the ALMARVI project.

2.3.2 Adaptive Bottom-Hat Filtering

The addressed problem is nuclei extraction of microscopic breast cancer images. The grayscale images consist of bright dots and dark regions, where bright areas are background and dark regions correspond to the nuclei that

are under investigation. Bottom-Hat Filtering are basic methods of mathematical morphology for small target detection. The Bottom-Hat Filtering is defined dually as the difference between the closing and the input image. Let *T* denote transformation operator, *f: E -> R* be a grayscale image, and *b(x)* be a grayscale structuring element. Then the Bottom-Hat Filtering is defined as (Equation 1):

$$T_w(f) = f - f \bullet b, \quad (1)$$

where • denotes the morphological closing operation.

Their usage for the nuclei extraction is justified due to the characteristics of the objects to be detected. However, the basic functionality as such is not sufficient enough to detect nuclei effectively which has led to the creation of derived adaptive filters. The goal of image thresholding is to allow certain pixel values to pass through the transformation. Performing the thresholding adaptively results in taking the variations in illumination into account. The adaptivity of our method lies in the way the adaptive thresholding is used in the structuring element.

The structuring element values are generated by first dividing an image into windows of certain size after which the calculation is processed inside each window. The values that define the structuring element are average grayscale value, deviation, and the difference between maximum and minimum grayscale values. We have studied the idea of color channel manipulation in our previous work and utilizing this idea we combined the two approaches. To enhance the histopathologic image, the RGB color channel manipulation is applied according to our formula (Equation 2): [UEF1]

$$I_{out} = R - (G + B) / 2 \quad (2)$$

where R,G,B are color channels of the input image.

After the color channel manipulation step, the adaptive Bottom-Hat Filtering is applied. The resulting image (see Figure 7) shows the final result of nuclei extraction. To compare the resulting images of filtering with (see Figure 7 left) and without (see Figure 7 right) the color channel manipulation both images are presented. The left image, the result of the adaptive Bottom-Hat Filtering together with color channel manipulation, presents clear contrast between the nuclear area and other areas. In the right image, some undesirable tissue structures and cell components such as fat and edges of stroma are still present as black areas whereas in the left image only the nuclei are highlighted as white and other areas grey.

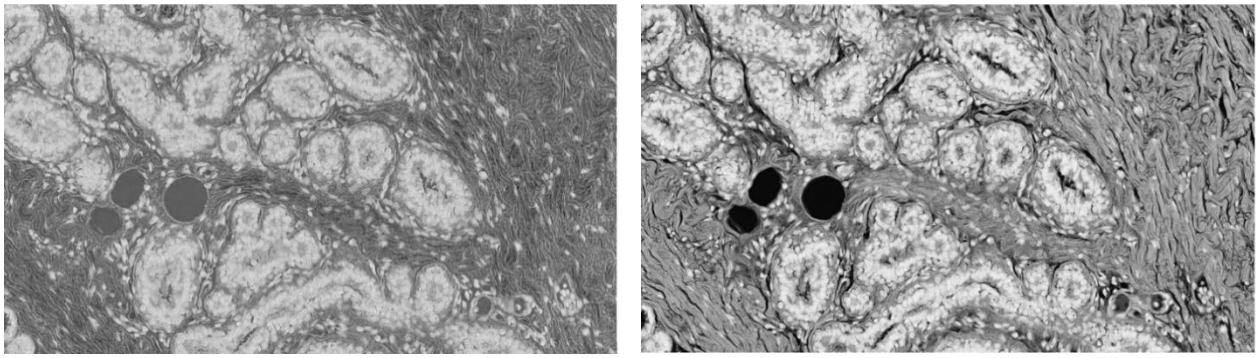


Figure 7: Resulting images of adaptive Bottom-Hat Filtering with (left) and without (right) color channel manipulation.

The objectives of the ALMARVI project were met as the used algorithm is very light in calculation, which results in the fast and effortless real-time processing of images. The algorithm is challenging to parallelize, because of the structuring element based computation.

Figure 8 illustrates three different resolutions of the original image of size 2872 x 1628 pixels (a) that is first scaled down to 1436 x 814 pixels (b), and finally scaled down to 718 x 407 pixels (c). The three images appear almost visually the same when used with these resolutions. However, there are noticeable differences in the details of the images. This can be seen when zooming in the certain parts of the images as showed in Figure 8. Moreover, when utilizing the lowest resolution image, the calculation time reduces to a fraction compared to the original one when using DTOCS-based feature extraction methods (excluding Alpha-DTOCS).

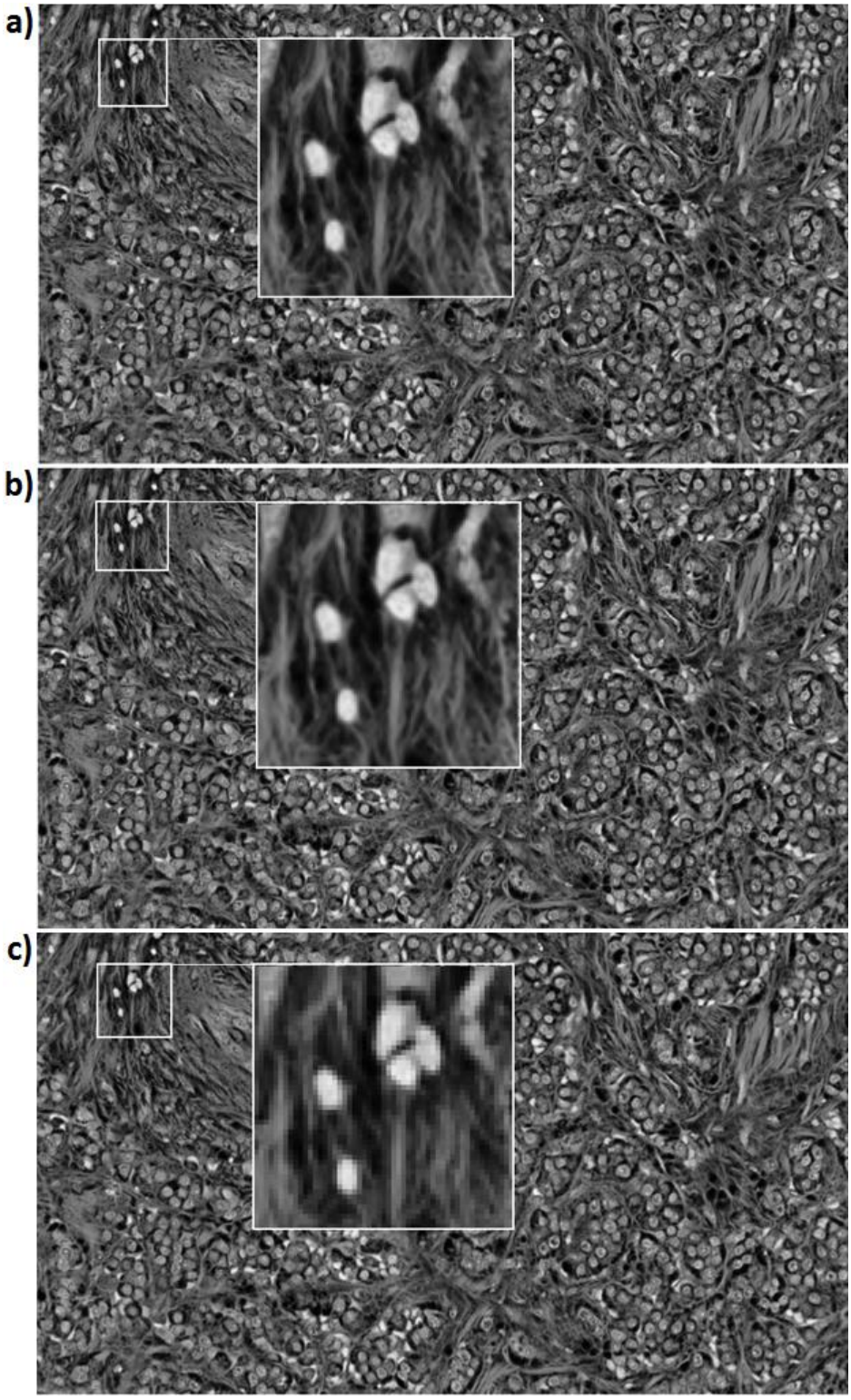


Figure 8: a) Original image (2872 x 1628 pixels). b) Resized image (1436 x 814 pixels). c) Further resized image (718 x 407 pixels).

2.3.3 Hyperchromatic Area Detection

HC-method (hyperchromatic) detects hyperchromatic areas from whole slide images (WSIs). HC-method separates hyperchromatic nuclei from, e.g., stroma (see Figure 9). HC-method can be used as a first step in detecting mitotic figures and other signs of tissue abnormalities that may be a sign of neoplasm. Hyperchromatic nucleus can be an

indication of over-active nuclear function, which may be an early indication of cancerous cell changes or a response to damage or other stimulus.

Hyperchromatic area detection algorithm is coded with Matlab. The input of the algorithm is an image. The algorithm handles the whole image data; it reads the image data in jp2-format and makes the hyperchromatic area detection utilizing a blue image, defined as follows:

$$I_{out} = ((100 * B)./(1+R+G)) .* (256./(1+B+R+G)) \quad (3)$$

where R,G,B are color channels of the input image. The output of the algorithm are hyperchromatic pixels detected in the input image.

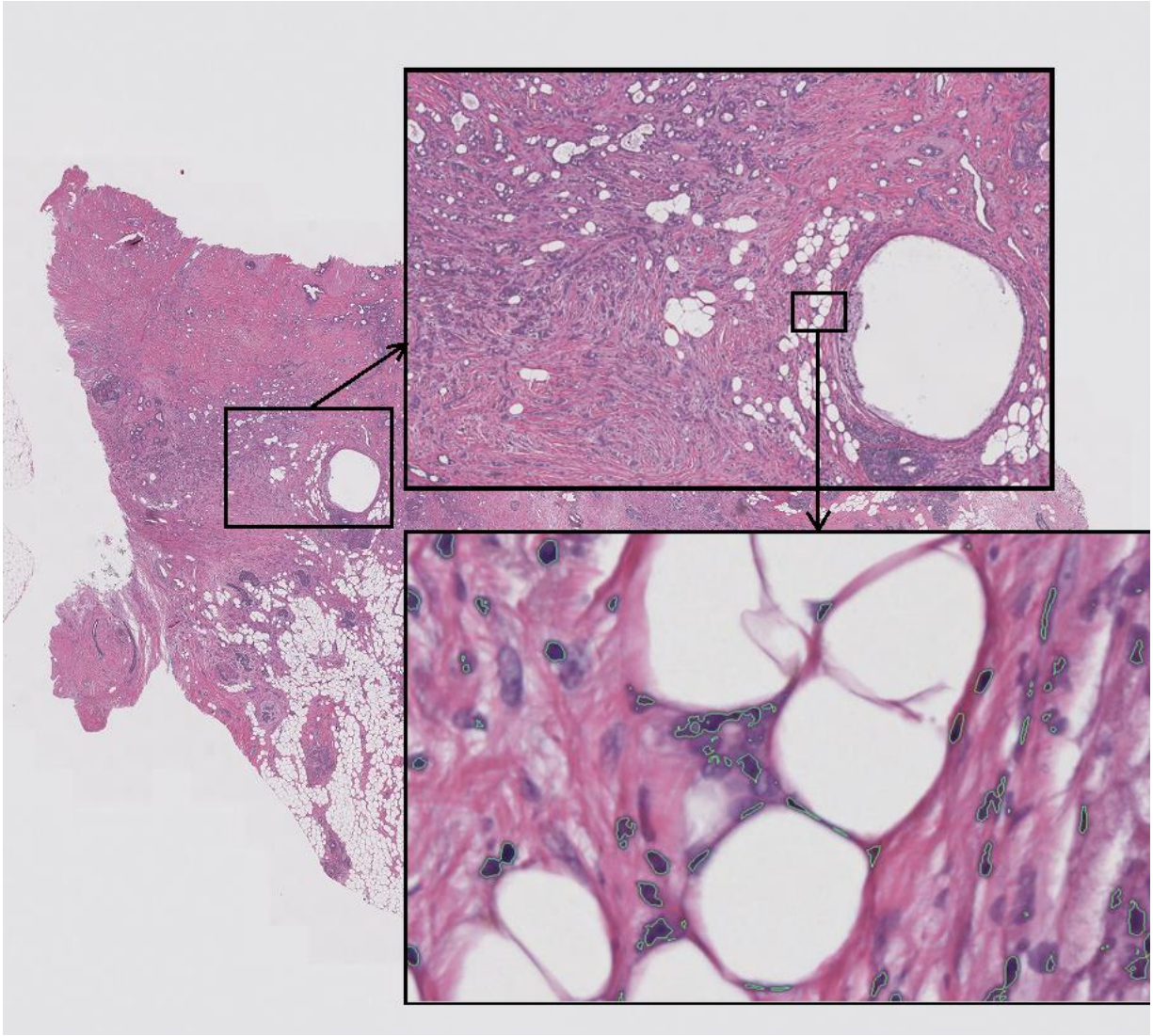


Figure 9: Hyperchromatic areas are detected from the WSI and marked with green color for visualization purposes.

The algorithm was tested with several jp2 compressed WSI input images in Matlab and also with IBCAS. Hyperchromatic calculation is simple and very scalable, which is essential since the input images are sizeable. Hyperchromatic area detection was tested with images of five different sizes (more instances per one size) and compared to Bottom-Hat Filtering (see Figure 10). The results show a clear need for parallelization when utilizing large sized images. The whole algorithm, comprising the color channel manipulation and the tresholding, is parallelizable.

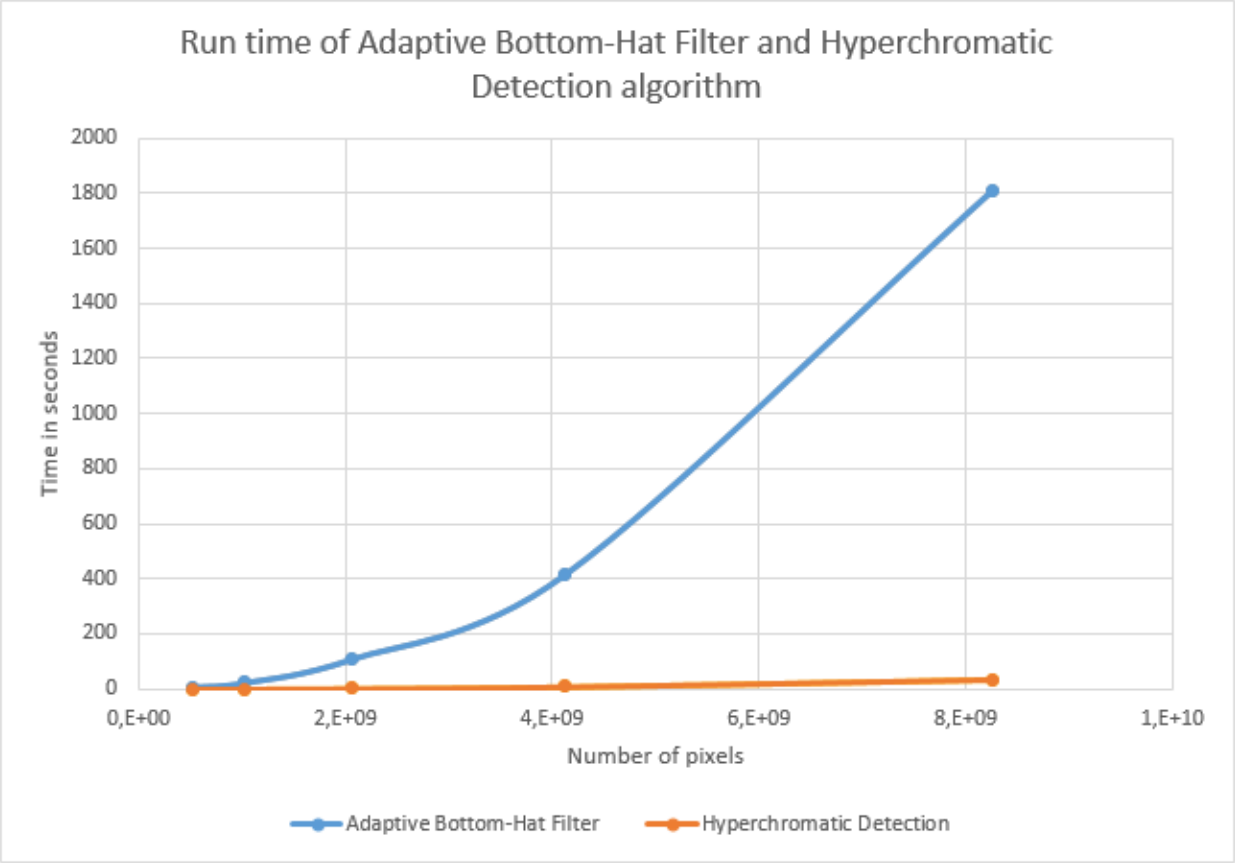


Figure 10: Comparison of adaptive Bottom-Hat Filtering and Hyperchromatic Area Detection.

Figure 11 illustrates three different resolutions of the original image of size 2872 x 1628 pixels (a) that is first scaled down to 1436 x 814 pixels (b), and finally scaled down to 718 x 407 pixels (c). The three images appear almost visually the same when used with these resolutions as well as it shows noticeable differences in the details of the images when zooming in as was the case also with Figure 8. Thus, same conclusions apply here as with the case of Figure 8.

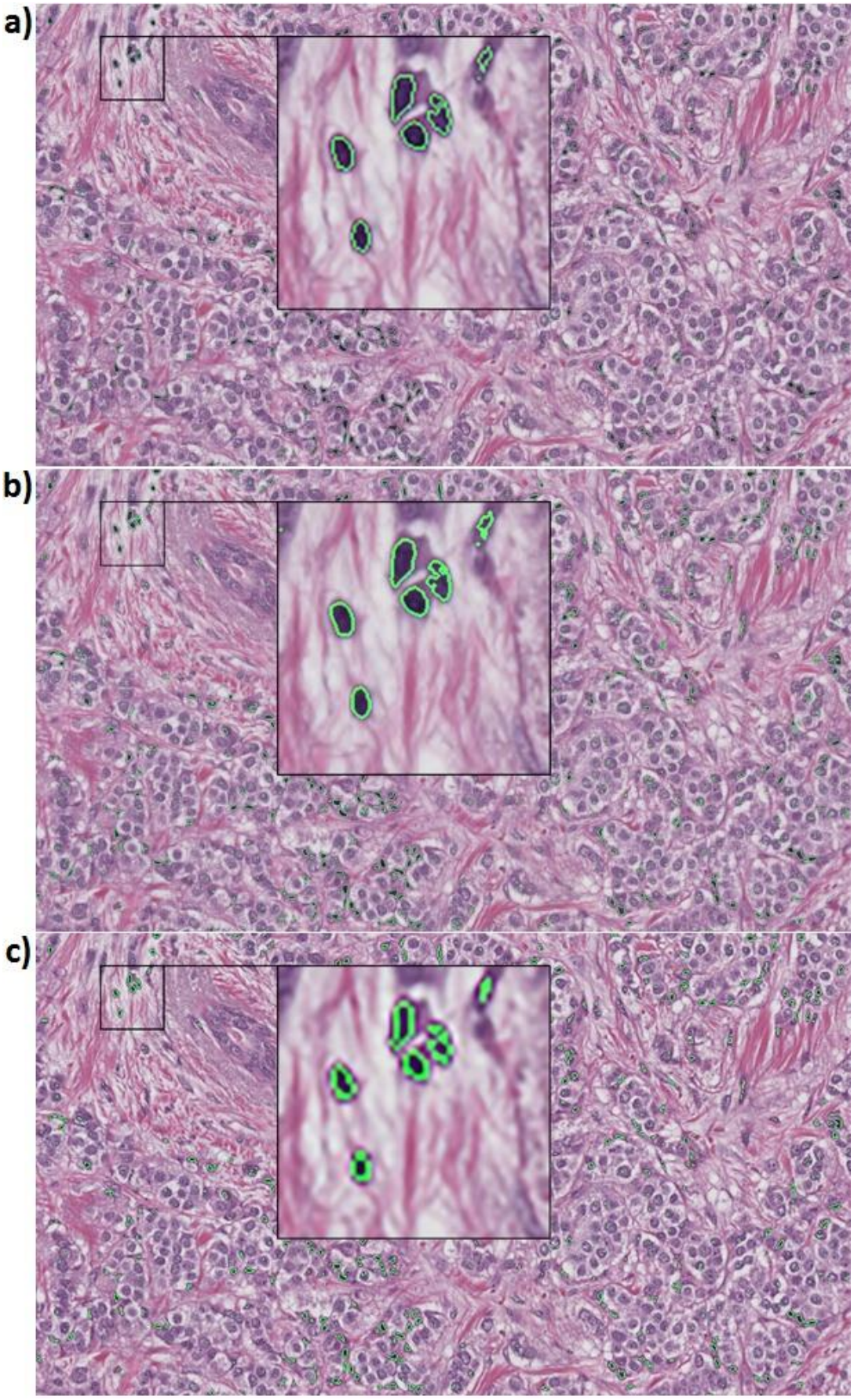


Figure 11: a) Original image (2872 x 1628 pixels). b) Resized image (1436 x 814 pixels). c) Further resized image (718 x 407 pixels).

2.3.4 Tissue and Nuclei Classification

DTOCS (Distance Transform on Curved Space) is used in breast cancer analysis for image segmentation/classification purposes in which different areas of an image, for example, stroma, carcinoma, fat, blood vessel, lobules, etc. are classified by utilizing SOM and MLP neural networks (see Figure 11). Figure 12 illustrates our tissue classification problem with expert markings.

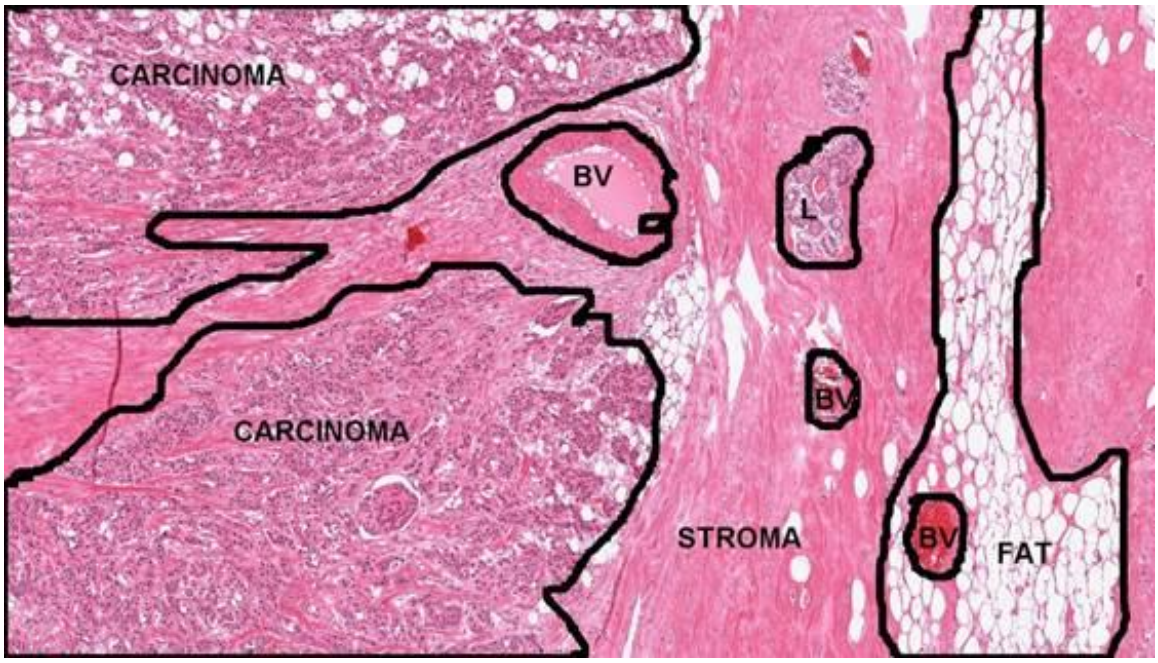


Figure 12: Tissue classification problem with expert markings.

The Distance Transform on Curved Space (DTOCS) is a method that calculates minimal distances on curved surfaces. The minimal distance paths obtained using it are geodesic paths on the curved surface. Parallel local operations affect each pixel independently, without taking into account the new values obtained for the neighboring pixels in the same parallel scan. The method is based on the idea of having a parallel computer with one processing element for each pixel. Some specialized architectures, like pyramid machines, can be used to efficiently implement parallel transforms, but advanced image processing today is typically done on sequential computers. SLT (Sequential Local Transform) algorithms also transform each pixel using its neighborhood, but as the pixels are processed sequentially rather than in parallel, the new values are used as soon as they become available. The parallel and sequential approaches are shown to produce mathematically equivalent distance maps. [UEF2, UEF3]

The sequential implementation of the DTOCS is based on the modifications of the SLT algorithm by Rosenfeld and Pfaltz. In the DTOCS, any pixels can be reference pixels, from which distances are calculated. To define the calculation area, a binary image $\mathcal{F}$ is needed for the transformation in addition to the original gray-level image . The calculation area X in $\mathcal{F}$ is initialized to max (the maximal representative number of memory) and the complement area X^c to 0. The zero-valued pixels indicate the source points for the distance calculations, that is, the feature pixels, when distances are calculate from the nearest feature, or the background pixels, when distances are calculated form the background into the object. Neither the calculation area, nor the reference pixel set needs to be connected. The original gray-level image is only used for calculating the local distance values between pixels, and remains intact through the transformation. The binary image, which originally defines the calculation area, is transformed into a gray-level image containing distance values. [UEF2, UEF3]

The DTOCS is calculated using the distance propagation mechanism. First the image used as input to Equation 4 is initialized so that reference pixels get value 0 and pixels in the calculation area get value max (the maximal

representative number of memory). In the DTOCS, any pixels can be selected as reference pixels, that is, distances can be calculated from single points on the surface, or from areas, and the calculation area can be connected or disconnected. The propagated local distance values contain a horizontal component for the shift in the image plane, and a vertical component defined by the gray-level difference between the pixels. The difference corresponds to the change in altitude between the pixels. [UEF2, UEF3]

$$\text{DTOCS: } d(p_i, \mathrm{p_{i-1}}) = |G(p_i) - G(p_{i-1})| + 1 \tag{4}$$

One possible speedup is to replace our sliding window based DTOCS by our Alpha-DTOCS. Alpha-DTOCS calculation gives approximately the same result in much shorter time compared with traditional DTOCS calculation with sliding window approach using, for example, 20x20 sized window. Earlier we utilized smaller images of size 1436 x 814 pixels, which were possible to calculate also with traditional DTOCS with sliding window approach within few hours and also then our Alpha-DTOCS helped us to speed-up the calculation from typical 4 hours per image to approximately 4 seconds per image, thus making the system almost real-time. However, since we are now utilizing WSI images of size up-to 200 000 x 100 000 pixels, it is no longer anywhere near practical to use the traditional DTOCS calculation with sliding window approach. Instead, with our Alpha-DTOCS approach, we can get approximately the same results within much shorter time.

The speedup of the Alpha-DTOCS with respect to the DTOCS can be seen in Figure 13. The test was realized on images with five different sizes. Further parallelization of the Alpha-DTOCS is desirable. We have preliminary ideas and its implementation is part of our future research work.

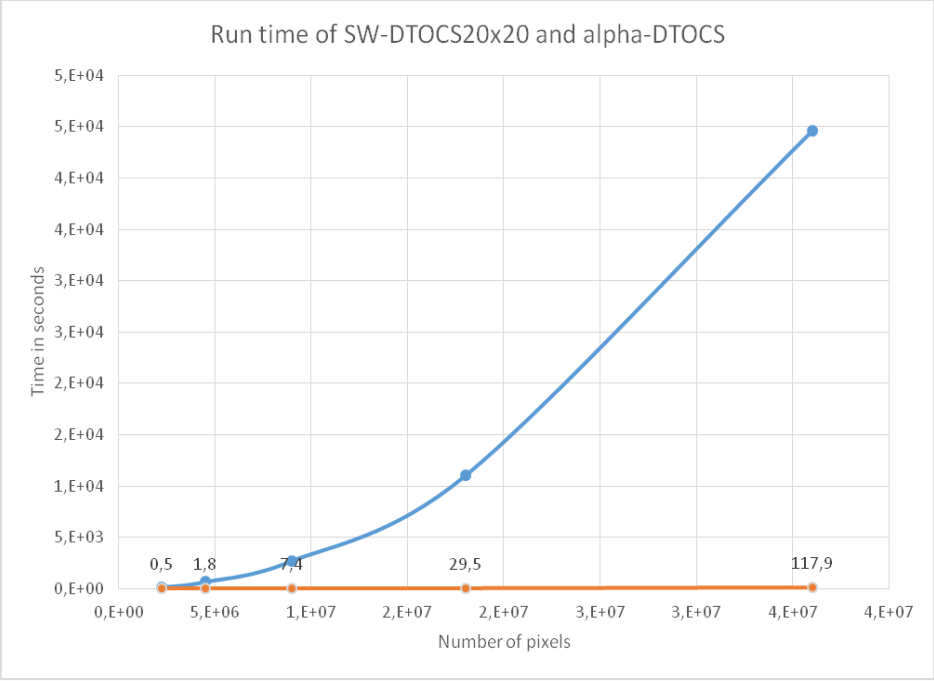


Figure 13: Hyperchromatic area classification with DTOCS20x20 (window size) (blue) and Alpha-DTOCS (orange) runtime comparison on 5 classes of differently sized images.

Our IBCAS demo system incorporates now the following features: Tissue and Nuclei classification. Tissue Classification utilizes DTOCS and Bottom-Hat Filtering feature extractors, and artificial neural networks for classification. Nuclei classification utilizes DTOCS, Bottom-Hat Filtering, and Hyperchromatic area detection feature extractors, and artificial neural networks for classification. The user can choose which classification will be used to analyse the image in IBCAS.

2.3.5 Requirements

Functional requirement 1:

- **Description:** Image segmentator must be able to handle images of different magnitudes and convert them from false color images to gray-level images.
- **Priority:** High
- **Type:** SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 100% success. Methods are capable of handling any type of images of any magnitude.

Functional requirement 2:

- **Description:** Feature extractor must be able to calculate multiple features inside each image area, thus forming feature vectors.
- **Priority:** High
- **Type:** SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 50% success. Partially implemented.

Functional requirement 3:

- **Description:** Image classifier must be able to recognize both cancer and non-cancer areas inside previously segmented image areas. Classifier must be able to utilize fusion data, i.e., feature vectors obtained using several images with different magnitudes.
- **Priority:** High
- **Type:** HW + SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 75% success. Partially implemented, because fusion data was not utilized.

Functional requirement 4:

- **Description:** Image displayer must be capable of visualizing cancer images with different magnitudes with high enough accuracy. This involves the use of different screen resolutions in different hardware platforms.
- **Priority:** High
- **Type:** HW + SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 0% success. During our research work in ALMARVI, we found out that this requirement is not relevant for our research work.

Non-functional requirement 1:

- **Description:** The Intelligent Cancer Diagnostics Application must run on two different hardware platforms:
  1) A standard PC equipped with multiple cores in CPU/GPU as well as a large amount of external and main memory for efficiency reasons.
  2) A tablet (e.g. iPAD) for usability and mobility reasons: however, all computationally intensive tasks will be performed using an efficient Linux workstation that sends results (e.g., via a WLAN connection) to the tablet.
- **Priority:** High
- **Type:** HW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 50% success. Due to lack of resources, the tablet part of this requirement was not implemented.

Non-functional requirement 2:

- **Description:** The Intelligent Cancer Diagnostics Application must run on two different software platforms:
  1) Implemented using Matlab: It will be mainly used in the first HW platform for efficiency reasons.

2) Implemented using Java: It will be used both in the first and second HW platforms.

- **Priority:** High
- **Type:** SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 50% success. Java part was not implemented, since we planned to run our algorithms on other partners HW platforms.

Non-functional requirement 3:

- **Description:** The Intelligent Cancer Diagnostics Application must be capable of real-time operation on both hardware platforms in order to make the application utilizable.
- **Priority:** High
- **Type:** HW + SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 50% success. We are still waiting for the results of TUDelft's rVEX platform and we are also finalizing of two algorithms for (near) real-time operation.

Non-functional requirement 4:

- **Description:** The Intelligent Cancer Diagnostics Application must utilize multi-core computing in order to achieve real-time operation.
- **Priority:** High
- **Type:** HW + SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 30% success. Our Matlab algorithms have been converted into C++ and have been given to be tested onto TUDelft's rVEX platform. We are still waiting for results.

Non-functional requirement 5:

- **Description:** The Intelligent Cancer Diagnostics Application must utilize GPU computing in order to achieve real-time operation.
- **Priority:** Medium
- **Type:** HW + SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 60% success. Our OpenCL implementation is in its finalization phase.

Non-functional requirement 6:

- **Description:** The Intelligent Cancer Diagnostics Application must run on Windows environment.
- **Priority:** High
- **Type:** SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 100% success.

Non-functional requirement 7:

- **Description:** The Intelligent Cancer Diagnostics Application must run on Linux environment.
- **Priority:** High
- **Type:** SW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 100% success.

Non-functional requirement 8:

- **Description:** The first HW platform must have a local database (an external hard drive or internal hard drive of a standard PC) for storing both original and result images.
- **Priority:** High
- **Type:** HW
- **Verification procedure:** Field test / Demonstrator
- **Results:** 100% success.

2.3.6 Objectives

The following table lists the requirements in terms of the baseline and desired values for the UEF breast cancer diagnosis use case.

Table 6. Requirements in terms of the baseline and desired values.

Parameter	Baseline	Desired values
Image size	1436x814 RGB 24-bit	200.000x100.000 RGB 24-bit
Processing speed (results)	4s (Alpha-DTOCS) / 1-3h (SW-DTOCS)	1s (Alpha-DTOCS)
CPU parallelization support	No	Yes
GPU parallelization support	No	Yes
OpenCL support	No	Yes
Support for multiple mobile platforms	No	Yes
Internet browser based user interface	No	Yes
Off-the-Shelf components	Yes / No (Nokia TTA & TuD rVEX)	Yes / Yes
Portability to Nokia TTA and TuD rVEX	No (Nokia TTA) / No (TuD rVEX)	Yes / Yes

Objective-1 – Enabling Massive Data Rate Processing

We have designed and implemented application specific algorithms for biomedical image analysis bearing in mind efficient computation times. Moreover, OpenMP and OpenCL were experimented within the algorithm development to give us the needed speedups in image processing. Furthermore, we also achieved huge speed-ups in image processing times by replacing our sliding window based DTOCS by our novel Alpha-DTOCS. Indeed, earlier we utilized smaller images of size 1436 x 814 pixels, which were possible to calculate also with traditional DTOCS with sliding window approach within few hours and also then our Alpha-DTOCS helped us to speed-up the calculation from typical 4 hours per image to approximately 4 seconds per image, thus making the system almost real-time. However, since we are now utilizing WSI images of size up-to 200 000 x 100 000 pixels, it is no longer anywhere near practical to use the traditional DTOCS calculation with sliding window approach. Instead, with our novel Alpha-DTOCS approach, we can get approximately the same results within much shorter time. We achieved calculation speed-ups from typical 50 days per image to 20 hours per image when utilizing large WSI images.

Objective-3 – Composability and Cross-Domain Applicability

The development of UEF's OpenCL-based version of the system that utilizes GPUs is currently in its finalization phase. We used an efficient GPU-based laptop with large memory size for running OpenCL-based version of the IBCAS system. The goal is to finalize our OpenCL implementation before the EU Commission final review. Results are planned to be demonstrated during the review in demonstrator section.

Objective-4 – Robustness to Variability

UEF was able to accomplish this objective by reducing image resolution to enable faster processing time requirements. Images were processed with different scales yielding similar results compared to the original (non-scaled) image. Our methods, however, do not gain significant benefit from reducing image resolution.

2.3.7 Research questions

There are fundamental areas of the domain that are not fully understood and should be explored by ALMARVI. For example, there are limitations in implementing parallelization for our feature extraction methods. The bottleneck in our image analysis pipeline is the feature extraction methods that are computationally quite heavy. Unfortunately, the parallelization is very difficult to implement because the distance transforms are global operations. Only possibility to use parallelization in the distance transforms is to use the jump flooding algorithm [UEF4], which is a GPU-based algorithm.

Segmentation is a fundamental problem and step in computer vision. In medical image analysis, segmentation is a crucial step, e.g., for image understanding and for making accurate diagnosis. The obtained medical images are prone to noise, blurring, and other decremental effects. If these effects are not handled, it will pose problems in analyzing the image. Some images may have a smooth border between two tissue types, which makes the segmentation process more challenging. Some state-of-the-art methods, such as Geodesic Active Contour (GAC) models [UEF5] using moving fronts propagation or level-set method [UEF6] in medical image segmentation, are a popular and widely used in the biomedical imaging field. They are robust methods against noisy images and are capable of handling images with smooth edges or regions, i.e., where the boundaries between two tissue types are fine and smooth. The problem with active contour methods is that they are expensive to compute because after each iteration, the contours have to be reinitialized. There exists methods for doing the segmentation without reinitializing, but this is still a heavy computational operation [UEF7]. Our goal was to use DTOCS based methods [UEF2] in such a way that they are able to handle multi-scale and multi-region segmentation efficiently in off-line and on-line computation. Gray-scale histology imagery are mainly considered, but color images (3D) will also be considered in histology image segmentation.

Breast cancer tissue consists of cells that grow uncontrollably. The structures formed by these cells and their morphology define the tumor type as well as the spreadness and aggressiveness of the cancer. To extract cancerous areas of an image, an accurate feature extraction method, such as distance transform [UEF2], needs to be implemented. When referring to cancerous features, we point out a symptom or a group of symptoms of a disease such as changes in tissue texture and architecture. As cancerous nucleus are darker, larger, and vary in shape when compared to healthy nucleus [UEF8]. Our objective was to automate the extraction of these features and combine them with earlier findings.

Different nuclei shape features in breast tissue images will be considered in our future research work. The *differences* and *similarities* in the shape features that are detectable by the *human eye* and/or *machine vision* application were studied. Humans are better in semantic retrieval and using intuition in complex situations. Machine vision is strong in performing tedious tasks, e.g., situations where an image is systematically divided to (e.g., hundreds of) smaller objects and each of those objects are then systematically analyzed. Tumors of the breast, like tumors in general, are lesions of that are results of abnormal growth. Cancer cells gain the ability to grow and divide in an uncontrolled manner. Nuclei of dividing cells have unique visible features: e.g., condensed chromosomes and disappearing nucleoli [UEF9]. These features may be utilized in a machine vision diagnosis approach. UEF build a machine vision system with the approach of utilizing the unique visible features of the nuclei with the emphasis of shape features. One of our future research work will be in recognizing cells in different cell cycle phases, e.g., dividing cells or stabile cells, using machine vision methods. Recognizing a high level of dividing cells can be a sign of abnormal tissue growth.

2.3.8 Validation of functionality

Our IBCAS demonstrator was validated on all levels of the V-model. The used MATLAB code was written by UEF and at every stage throughout our design process, we made sure that the design matches the results of the MATLAB code. Our demonstrator was continued from the previous EU project (HIGH-PROFILE) from which we have already validated it.

2.3.9 Verification of robustness and reliability

The image processing algorithms were tested on a variety of image types and sizes, which were acquired from our collaborators from the Biomedical technology institute of University of Tampere. All image processing algorithms yield acceptable and robust results and these were compared to what we know about the theory of the algorithms. How well the algorithms performed in a classification task, we tested the methods using different neural network models. The validation of the neural networks models were done using cross-validation and confusion matrix metrics.

2.3.10Conclusion and Lessons learned

Histopathological image analysis continues to be a very challenging task. The small variations in images (tissue, cells, and coloring technique) poses problems for accurate segmentation, feature extraction, and classification. These small variations makes it difficult for classifying malignant cancerous tissues from the image. The methods we were researching in ALMARVI project were not trivially parallelizable, because these methods require multiple iterations and also each iteration is dependent on the results from the previous iterations. However, despite of these challenging problems and obstacles, we managed to get promising results with lot of potential for future research work.

3. Security

3.1 Introduction

In the security domain, ALMARVI has developed five main demonstrators: large area video surveillance, road traffic surveillance, smart surveillance, logical analysis of multimodal camera data and the protection of walnut tree harvest against birds. This chapter describes the evaluation of the demonstrators. Table 7 below shows the relation of the demonstrators to the ALMARVI objectives.

Coverage of ALMARVI Objectives		Security / Surveillance and Monitoring				
		Large Area Surveillance	Road Traffic Surveillance	Smart Surveillance	Analysis of multimodal camera data	Walnut Tree Harvest Protection
1	Acceleration fabrics	■	■		■	■
	Design tools	■			■	■
	Application specific parallelization	■	■		■	■
	Quality - performance trade-off					
2	Low power cores	■	■			■
	SW quality - power trade-off	■				
	HW quality - power trade-off					
	Algorithm resilience for power efficiency	■				
3	Software portability	■	■	■	■	■
	Interfaces for heterogeneous acceleration fabrics	■	■	■		■
4	Guarantee system performance	■	■			■
	Guarantee power consumption	■	■			■

Table 7: Objectives for the Security demonstrators

3.2 ASELSAN/OZYEGIN: Large Area Video Surveillance

3.2.1 Introduction

In large area video surveillance applications with many cameras, we apply an optical flow algorithm to all camera streams. If optical flow flags no motion, the respective stream is not dispatched to a server for further processing nor stored on disk. If enough motion is detected, then the respective stream is dispatched to servers for further processing in order to detect certain objects and is stored on disk. Also, the respective stream undergoes more precise optical flow analysis until there is again not enough motion. The vectors produced from optical flow are not only used to switch between these two modes but also are used by the specific detection and recognition algorithms run on the servers. Hence, optimized implementation (GPU and FPGA based) of optical flow in terms of usage of hardware resources and power, while being able to support high throughput and resolution (for the second mode after motion is detected), was critical in this demonstrator.

In addition to the above "Optical Flow Demonstrator", ASELSAN and OZYEGIN also worked on a "Fusion Demonstrator" with GPU and CPU implementations. In addition, ASELSAN worked on a "Segmentation Demonstrator" as well as a "Registration Demonstrator" on an embedded platform with built-in GPU. These algorithms (segmentation and registration) are to be used for the detailed image analysis algorithms run on the server farms once the optical flow platform dispatches the frame for further processing. This report will only detail the optical flow demonstrator, which showed that with FPGA we can achieve lower power consumption and smaller form factor for a wide range of throughputs and resolutions (requiring up to 75 GFLOPS). Although, for the fusion demonstrator, the same has been demonstrated (with performances up to 172 Mpix/s). For segmentation and registration, ASELSAN showed that an embedded platform with built-in GPU is able to meet our performance per Watt per dollar objectives. For example, for segmentation, 5 fps at 640x480 pix/f was achieved on a COM Express embedded computer (VXG101 of Connect Tech).

In our optical flow and fusion demonstrators, we did not want to target a single FPGA platform or a single throughput-resolution. On the other hand, optimizing an FPGA implementation requires fixing the target performance and the hardware platform. Coming up with an optimized implementation on a new platform with different throughput and resolution, even when a full implementation is carried out on a different platform with or same throughput-resolution, is extremely labour intensive compared to a new CPU or GPU implementation. Thus, OZYEGIN created flows and tools to streamline hardware/software design and verification, through which we speed up FPGA design up to 10-fold. A summary of the specifics of this process is presented in subsections titled "Research Questions" and "Lessons Learned".

We chose Anisotropic Huber-L [ASEL1] Optical Flow algorithm and started with their freely available implementation as our version zero. ASELSAN later fine-tuned the algorithm on GPU. With the help of flows and tools created at OZYEGIN, the algorithm was implemented on 3 different FPGAs, namely, Xilinx Spartan 6 (on Linera's FMU-S6 board), Altera Cyclone IV (on Terasic's De2i 150 board), Altera Arria 10 (on Nallatech's 385A board) at 11 different performance points (see below) in terms of not only throughput and resolution but also in terms of precision (i.e., number of iterations). The compute horse power we were able to obtain from these 3 boards was up to, respectively, 300 MFLOPS, 900 MFLOPS, 75 GFLOPS.

- **Altera PCIe implementations (initiation interval = 10 cycles):**
  - Cyclone IV, 1 iter, 20 fps, 576x768 pix/f, 30 MHz
  - Cyclone IV, 2 iter, 20 fps, 576x768 pix/f, 30 MHz
  - Cyclone IV, 3 iter, 20 fps, 576x768 pix/f, 30 MHz
  - Arria 10, 1 iter, 19.5 fps, 640x480 pix/f, 60 MHz
  - Arria 10, 2 iter, 19.5 fps, 640x480 pix/f, 60 MHz
  - Arria 10, 25 iter, 19.5 fps, 640x480 pix/f, 60 MHz
  - Arria 10, 50 iter, 19.5 fps, 640x480 pix/f, 60 MHz
  - Arria 10, 50x10 iter, 1.95 fps, 640x480 pix/f, 60 MHz
  - *Arria 10, 50x10 iter, 3-level pyramid, 3.75 fps, 640x480 pix/f, 150 MHz*
- **Xilinx USB3.0 implementations (initiation interval = 10 cycles):**
  - Spartan-6, 1 iter, 5 fps, 576x768 pix/f, 15 MHz
  - Spartan-6, 2 iter, 5 fps, 576x768 pix/f, 15 MHz

3.2.2 Requirements

Functional requirement: Achieved

Description: Anisotropic Huber-L [ASEL1] Optical Flow up to 500 iterations and 3-level pyramid per frame

Priority: Must have

Type: HW + SW

Verification procedure: Verified against MATLAB + Verified in RTL simulation + Demonstration

Non-functional requirement 1: Achieved

Description: Less than 10 W per 1 Mpix/s at low accuracy (3 iterations)

Priority: Must have

Type: HW + SW

Verification procedure: Power measurements through a smart plug

Non-functional requirement 2: Achieved

Description: Host/FPGA interface supports both PCIe and USB

Priority: Must have

Type: HW + SW

Verification procedure: Software Test + Demonstration

Non-functional requirement 3: Achieved

Description: Quick portability across different FPGAs (both Altera and Xilinx)

Priority: Must have

Type: HW + SW

Verification procedure: Verified in RTL simulation + Demonstration

Non-functional requirement 4: Achieved

Description: 1 Mpix/s throughput in a single unit at high accuracy (500 iterations + 3-level pyramid)

Priority: Nice to have

Type: HW + SW

Verification procedure: Verified against MATLAB + Verified in RTL simulation + Demonstration

3.2.3 Objectives

Objective-1 – Enabling Massive Data Rate Processing

Thanks to MAFURES, our internal High-Level Synthesis (HLS) tool, and Vivado HLS, we were able to create 11 versions of our FPGA implementation. We targeted 3 different FPGAs, namely, Xilinx Spartan 6, Altera Cyclone IV, Altera Arria 10. Out of these, Altera Arria 10 is a very high-end 20-nm FPGA with around 15 billion transistors, whose production quantities were available in 2016. It has a hypothetical peak performance of 1366 GFLOPS. We were able to achieve 75 GFLOPS plus several giga integer operations per second. Once we speed up the clock frequency of the Warp unit from 150 to 300 MHz (the frequency met by other blocks), we will be able to achieve 150 GFLOPS. The hypothetical peak performance of 1366 GFLOPS assumes all floating-point multipliers of the FPGA are utilized and are running at 900 MHz. However, in this application we are constrained by the Block RAMs of the FPGA, and we use 500 of the 1500 floating-point multipliers. We achieved over 5 fps at 576x384 pix/f on an Nvidia GTX 780 GPU with single (32-bit) precision as reported in our SAMOS 2015 paper. To be exact 1.17 Mpix/s was achieved, which meets the goal set in our non-functional requirement 4 above. On the Arria 10 FPGA, we achieved 1.15 Mpix/s.

Objective-2 – Achieving Low-Power Consumption

We were able to show through this project that we are able to achieve lower power consumption with an FPGA solution as compared to GPU for a given performance (pix/s or equivalently fps x resolution). We below have two tables for comparison. The first compares a high-end FPGA with high-end GPU. The second table compares mid-range FPGA and GPU.

Number of Iterations in Optical Flow at 12.5 fps	Nvidia GeForce GTX 980 Ti (GPU) Power Consumption with server-class host	Altera Arria 10/1150GX (FPGA) Power Consumption with the same server-class host
Idle	105 W	105 W
1	+ 70 W	+ 25 W
2	+ 71 W	+ 25 W
25	+ 80 W	+ 27 W
50	+ 90 W	+ 30 W

Table 8: Power Consumption Comparison of the high-end GPU GPU and FPGA implementations

Although not identical, these are top ranking chips of their product category in the years they were introduced, which are only one year apart. The FPGA is one year (2016 vs 2015) and one generation (20 nm vs 28 nm) newer. These are both very high-end chips. 980 Ti is the second highest performance GPU from Nvidia that was available in quantity after Titan X until a few months ago. Arria 10, on the other hand, is the most high-end FPGA family from Altera until Stratix 10 ships. Although the two chips are comparable in terms of being one of the most high-end products of their category, there is quite a big price differential between the two ($700 vs $5000). That is mainly due to the fact that Nvidia's volume of GTX product line for gaming at home is very high, and hence the prices are low. On the other hand, if we take Quadro K6000 product line, which is a professional product line with a different grade of technical support and much longer uptimes, the price jumps to almost $4000, although it falls short of the performance of GTX 980 Ti. Therefore, we conclude that it is almost an apples to apples comparison when these two chips implement the same algorithm at the same video processing throughput, which is 768x576 pixels/frame at 12.5 fps while applying our optical flow algorithm of choice with 50 iterations and 1 pyramid level.

We made the power consumption measurements using: *Edimax Wi-Fi Smart Plug with Energy Management (SP-2101W)* which is available in many online stores, including Amazon, and is around $35. Both in the case of GPU and FPGA, we measured power consumption when neither GPU nor FPGA based optical flow runs. We call that "Power Consumption at System Idle", and it measures at 105W as stated on the second row of Table 8. When we run the GPU or FPGA, we measure again and take the difference. Table 8 reports those differences in the cells that start with "+".

As for our mid-range comparison, we made power measurements with a laptop GPU that was high-end 3-4 years ago, namely, Radeon 7870M. Note that 7870M was over $700 when it came out in 2012. However, the Cyclone IV/GX150 board from Terasic preserves its price. This is again due to the volume of the respective products. The below table lists the power consumption of the 7870M GPU and Cyclone IV FPGA at 6 fps of 768x576 pixel frames. While, in the case of the previous table, the host of GPU and FPGA are the same, the GPU and FPGA below have different hosts. The FPGA board falls into the embedded category. The GPU falls into the mobile category (i.e., for laptops) and is one notch up compared to the FPGA platform which we have benchmarked.

Number of Iterations in Optical Flow at 6 fps	AMD Radeon HD 7870M (GPU) Power Consumption with laptop host	Altera Cyclone IV/GX150 (FPGA) Power Consumption with an Intel Atom host
Idle	38 W	16 W
1	+ 35.6 W	+ 1.4 W
2	+ 36.5 W	+ 1.5 W
3	+ 37.7 W	+ 1.5 W

Table 9: Power Consumption Comparison of the mid-range GPU and FPGA implementations

FPGAs can beat, as shown above, GPUs in power consumption, as they, by nature, better utilize their logic. However, that requires the host software to be pipelined if that is where the video stream comes from and where

the output streams go to. If the host software is not pipelined, i.e., not multi-threaded, then the FPGA may idle and its power consumption metrics may not fare well compared to GPU and multi-core CPU at the extent it is expected. By the way, in our case, we do not have power consumption comparisons to a multi-core CPU implementation of the same algorithm, because the optical flow algorithm we implemented is a highly precise algorithm. When it is run in our high accuracy settings, it requires 75 GFLOPS, and it is not possible to achieve such throughput on a multicore CPU.

Objective-3 – Composability and Cross-Domain Applicability

Our GPU implementation of optical flow uses OpenCL language. Not only Nvidia (our chosen GPU platform) supports OpenCL but also pretty much all other GPU vendors. It can even be ported with relatively reasonable effort to Altera FPGAs through Altera's OpenCL to FPGA synthesis tool. ASELSAN's experience before ALMARVI showed that this approach offers savings in power consumption compared to GPU when targeting modest performance goals. However, when ambitious performance targets are the case (as in our high accuracy mode), Altera's OpenCL path is not a viable solution. That is why we took the path of FPGA implementation through HLS.

Our FPGA implementation approach has a flexible framework with a choice of USB or PCIe host interface. It can also work on Windows and Linux. The software interface is based on a clean API and hence can be integrated into other platforms when needed. Moreover, it supports both Xilinx and Altera FPGAs and all FPGA families within each vendor.

Objective-4 – Robustness to Variability

We spec our FPGA logic based on the worst-case operating conditions. Hence, in terms of operating temperature range and ripple in the supply voltage, our FPGA solution is pretty robust. On the GPU side, if Quadro product line is used (directly from Nvidia), then robustness to operating conditions would be greater. Also, if the system is specced out with a large margin for the required pixel throughput, it will endure changes in fps. With slight RTL change, the system can even endure resolution changes.

General Objectives

Overall, we have used our optical flow design as a test case to determine the requirements to achieve quick architectural exploration and short FPGA design cycles for video processing applications and to come up with a framework and tools to meet those requirements.

3.2.4 Research questions

Creating real-time low-power video processing designs on heterogeneous platforms that are cost effective requires significant architectural exploration especially on the hardware side, which requires estimation tools and short design cycles (hence sophisticated design and verification tools). Due to such needs, our research efforts have revolved around the following headings:

- HLS for high-throughput floating-point FPGA designs: Traditional HLS (including Vivado HLS) puts resource sharing multiplexers within the same cycle as the function units. This lowers the clock frequency we can achieve. Our own HLS tool is called MAFURES and stands for Multiplexer Aware FUnction and Register Scheduler. MAFURES places resource sharing multiplexers in their own cycles and can even reserve multiple cycles for them. MAFURES already overperforms Vivado HLS, however, there is more room for improvement in the case of MAFURES, which requires more research.
- Area-Power-Throughput Estimator: When designing on an FPGA, it is very important to set a target or predict whether or not a performance target can be achieved. However, at the end of the project, the targets may not be achieved unless there is a good prediction method. For this, an estimator tool that estimates power, area, and throughput for a given set of FPGA resources is needed. Without an estimator, we would have to go all the way from RTL development to synthesis for too many architectures.
- Cycle Accurate Flow Simulator: Cycle accurate flow simulator separates flow and computation. This method is very useful in debugging. It facilitates the identification and elimination of all flow related errors early on.

3.2.5 Validation of functionality

The MATLAB code written by Werlberger el al. [ASEL1] for their optical flow algorithm is our golden reference. At every stage throughout our design process, we make sure the design matches the results of the MATLAB code. There are some small expected differences due to the floating point units used on FPGA. The design has been also validated visually in real-time on the FPGA in addition to the simulation strategy explained above.

3.2.6 Verification of robustness and reliability

The Arria 10 FPGA card used has a maximum temperature operating range of 0-100 Celsius. It is designed to support all temperatures in terms of the strength of the design. The delay of the algorithm, the constant generation of power consumption, and the computation of the correct values every time it is run have been experimentally verified.

3.2.7 Validation of performance metrics

The power and performance validation of the running optical flow algorithm on the FPGA was experimentally taken. There is no significant difference between the calculated performance values and the reality.

3.2.8 Lessons learned

Because of the significant size of our FPGA designs, almost all the problems that can be encountered in an FPGA design and implementation process have been encountered. Seeking solutions to these problems has given the team considerable experience.

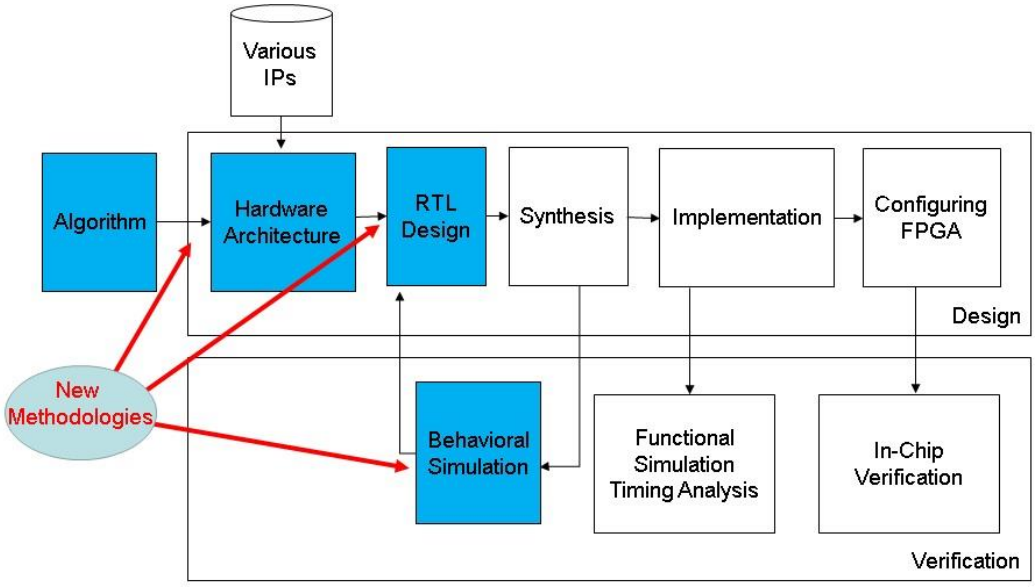


Figure 14: FPGA Design Flow

Figure 14 shows the FPGA design process. Serious problems were encountered in the places shown as new methodologies above. Therefore, we have discovered that new design methods should be developed in the relevant stages.

Especially multi-cycle floating point units may not be utilized all the time; turning these units off during periods when they are not active reduces power consumption. Another important factor in power consumption is the use of DRAM. During the early phases of our development work, DRAM bandwidth was not efficiently utilized in most cases. We had to optimize our DRAM traffic to lower power consumption.

3.2.9 Conclusion

The optical flow design that has been discussed above has a significant size. It has a pipelined datapath, which we designed using MAFURES HLS and the other tools mentioned above. Optical flow also instantiates a downscaler, upscaler, and warp unit, all of which were designed using MAFURES. Especially, Warp was quite complicated and MAFURES helped a lot. MAFURES was also employed for the Fusion design, which was also complicated. In conclusion, we have put together a set of processes and tools through which we can develop more demonstrators relatively easily.

3.3 CAMEA: Road Traffic Surveillance

3.3.1 Introduction

This demonstrator includes real-time object detection in HD video using a low cost and low power compact system. Such embedded system will be used for road traffic surveillance applications as counting of vehicles, license plate detection (for further recognition) and many others. The core of the system – the object detector – is based on the WaldBoost algorithm and is trained for cars' license plates. The results of our development within ALMARVI project and also important parameters of our demonstrator are, in more detail, described in deliverable D5.4 and later in this document.

Figure 15:CAMEA demonstrator - Camera based on Xilinx Zynq

We met our objectives (defined in D1.1) by designing and manufacturing an embedded camera based on Xilinx Zynq SoC which is introduced and in detail described in D1.2 System Architecture. This demonstrator is configured to detect licence plates and is using classifier trained in cooperation with BUT. The object detection is fully HW accelerated in FPGA and is performed by the IP core of Waldboost based detector (developed by BUT) and described in D2.5. Results of the detection are processed by ARM cores of Xilinx Zynq (drawing rectangles as results of the detection) which are also setting up and driving the CMOS sensor (Python 2k CMOS from On Semiconductor). CMOS data output is attached directly to the FPGA. The camera is also equipped with a H.264 hardware encoder from Fujistu. Resultant camera output enhanced by detection results is compressed and streamed in MPEG-TS over the Wi-fi and Ethernet network and can be displayed by most of the media players with H.264 decoding support.

BUT in collaboration with CAMEA proposed and described the object detection algorithms suitable for the ALMARVI project in deliverable D2.5. These are based on WaldBoost approach with features used for detection vehicles' license plates. BUT also trained a Waldboost detector classifier based on the image datasets collected and provided by CAMEA.

3.3.2 Requirements

Functional Requirements

Surveillance and monitoring image/video processing

Description: Capturing scene and image preprocessing.
Priority: Must have.
Type: HW/SW
Verification procedure: Measuring output frame rate of image/video stream and comparing images (after color transformation) with real scene seen by human eye

Results achieved: This requirement was fulfilled by designing and manufacturing the CAMEA demonstrator - Xilinx Zynq based camera with attached CMOS sensor. The FPGA is receiving the image from the CMOS sensor and is also post-processing results and buffering images into RAM memory. The camera is able to compress output video by H.264 encoder and stream it over a network such that visual results were validated by a standard video player on a PC.

Surveillance and monitoring image/video processing

Description: Capturing HDR images (or video stream)
Priority: Nice to have.
Type: HW/SW
Verification procedure: Comparison of results of detection and analysis methods carried out on HDR vs. LDR images
Results achieved: We implemented only simple HDR extension into our final demonstrator. The camera is capturing a sequence of differently exposed images, but instead of merging them into a HDR image, the detection is performed on every image in sequence. This is simulating HDR input very well while avoiding artifacts that can occur during merging process (called as "ghosts") when the scene contains object movement. This approach did not require any change in object detector IP what we found very useful as we have found out that training the classifier for HDR images is much more difficult than for standard LDR images. The achieved improvements in detection accuracy are summarized later in subsection 3.3.6.

Object detection

Description: Localization of objects (e.g. faces or cars) in images captured by camera
Priority: Must have.
Type: SW
Verification procedure: Statistical accuracy will be validated on test dataset
Results achieved: We measured the accuracy of the IP core against the IP core's reference C implementation. The image datasets used for validation and measuring of accuracy were captured in real traffic under different weather and lightning conditions. The validation is further described in 3.3.5 and the ROC curve of detector is shown in Figure 16.

Non-functional Requirements

Surveillance and monitoring image/video processing

Description: The camera will capture images with defined frame rate (CCD driven by a FPGA part of Xilinx Zynq) and it will perform an image transformation to a desired color space
Priority: Must have.
Type: HW/SW
Verification procedure: Same as in functional requirements
Results achieved: The camera is equipped by a CMOS sensor. The FPGA part of Zynq contains hardware interface for CMOS which consists from LVDS differential pairs. Input data from the CMOS are dispatched to decoding unit which is extracting video data and signaling from raw data stream. This unit has a grayscale output for the object detector.

Surveillance and monitoring image/video processing

Description: The camera will compose HDR images from multiple exposures (this can improve detection results)
Priority: Nice to have.
Type: HW/SW
Verification procedure: Same as in functional requirements
Results achieved: We did not implement the composition of HDR images, instead of this solution we chose to acquire alternating sequence of differently exposed images. This is simulating HDR input very well while avoiding artifacts that can occur during merging process (called as "ghosts"), if the scene contains object movement. The achieved improvement in detection accuracy are summarized later in subsection 3.3.6.

Object detection

Description: The system will be built on WaldBoost algorithm

Priority: Must have.
Type: SW
Verification procedure: Same as in functional requirement
Results achieved: We used WaldBoost algorithm based IP core developed by BUT.

3.3.3 Objectives

OBJECTIVE 1 - Enabling Massive Data Rate Processing

To fulfil the first objective, the object detector IP was ported to Xilinx Zynq architecture. The new architecture provides better performance results (Table 10), allows much higher resolution of input image and also allows higher scanning windows against the previous implementation. We also improved the image downscaling mechanism in terms of a better sampling algorithm and also maximum number of image scales. The IP core is very easily configurable and even allows to send downscaled image as an input for another instance of detector which expands the capabilities of further downscaling and detecting even bigger objects (up to the size of a whole image).

Parameter	Baseline Xilinx Spartan6	Desired Xilinx Zynq
Image size	640x480 at 6 bit	1280x720 at 8 bit
Scanning window	21px	up to 28px
Frame rate	95 fps	70 fps
Filter scaling	bilinear 5/6	Lanczos 5/6 (precise)
FPGA design	at 100 MHz	at 200 MHz

Table 10: Baseline Spartan6 (current) and desired Zynq implementation of the object detector

Our FPGA object detector is able to process very high frame rate due to smart design and several key improvements tailored to FPGA architecture, as a fine task parallelization and effective pipelining. For massive data rate processing, it is very important to focus on stream processing and minimize data accesses "outside" the FPGA, for example to DDR memory. This is the main bottleneck of every FPGA. For this reason we designed a very complex pipeline, which is able to process several features and also scanning windows in parallel.

The overall performance can be further improved (up to 4-times), at the expense of a bit worse accuracy, by skipping every second position of scanning window in horizontal and vertical direction.

One of the key improvements was performed in multi-scale detection. One of the key improvements was performed in multi-scale detection. We are performing image downscaling on-the-fly to the same line buffer and then we are saving a lot of bandwidth to the DDR memory.

OBJECTIVE 2 - Achieving Low Power Consumption

We have satisfied the second objective which was focused on camera power consumption. Whole camera demonstrator consumes 8W at full load which includes CMOS at 60 FPS (~1.5W), LP detection on the same speed, H.264 encoding (~1W) and streaming of the video over network.

The main power saving was achieved by porting to the Zynq platform itself which is based on much better factory process (45nm -> 28nm). The implementation is more demanding to FPGA resources, as can be observed in Table 1 of deliverable D2.9, but, thanks to better factory process, we reduced the power demands by 31%.

Another power saving was achieved by PCB redesign and by transferring the Zynq with detector IP core inside the camera.

The power consumption can be further reduced in exchange for lower precision of detection (which is not very recommended) or by limiting the size of an object which the detector can found. In that case, the core can be smaller and less demanding on FPGA resources because there won't be the need of full input image downscaling.

OBJECTIVE 3 – Composability, Flexibility, and Cross-Domain Applicability

The detector IP core is written fully in VHDL which allows relatively easy applicability not only for other Xilinx FPGAs but also for FPGAs of other vendors.

Our detector is complied with AXI standards and uses AXI4-Stream standard for image input interface and for transferring the detection results. The configuration interface is AXI-Lite compatible. This ensures easy composability with other computational and data transferring blocks like DMA, image processing units etc. not only in Xilinx designs.

Our object detector is easily reconfigurable and the detection class can be changed by a microcode. It is also possible to update the microcode on-the-fly without reprogramming the FPGA. The microcode is a result of classifier training process. It is possible to perform training of classifier for detection of different classes of objects. The only limitation is the object size which is dependent on scanning windows size set during synthesis process.

The core is configurable also through set of registers which specifies the input image resolution, number of image scales and also the classifier final thresholdthat affects the numbers of true and false positive detections.

OBJECTIVE 4 – Robustness to Variability

The object detector IP core has a constant performance for calculation of 350M of weak classifier per second (weak classifier = one LRD feature). As we are not limited by an OS, the computational power and latency is constant. Camera’s design also guarantees constant power consumption.

3.3.4 Research questions

Our research questions are focused on object detection task. We have proven that using the hardware accelerated Waldboost algorithm is advantageous for traffic applications. The main benefit is in speed of LP detection which is exceeding performance of our standard PC implementation (Core i5) almost 10-times while consuming just small fraction of power. The Waldboost algorithm also proved to be very precise, as shown in Figure 16.

Initially, we chose the Waldboost due to it's easy paralelization. The state-of-the-art algorithms and IP cores are mostly based on Viola-Jones Adaboost with Haar features. These features are demanding on resources and needs to be calculated over integral image which makes the memory subsystem very demanding and complicated. All these negatives makes the Viola-Jones algorithm very non-effective for implementation on the FPGA.

We have also implemented LRD (Local Rank Differences) features instead of traditional LBP. The main benefit is the size of classifier alpha tables that needs to held just 17 values instead of 256 for LBP. The using of LRD then saves a great amount of BlockRam resources, but on the other hands LRD features are slightly worse from the point of detection accuracy.

The detector is trained and then strongly dependent on license plate orientation. The current classifier covers only license plates with horizontal orientation (±5°). It is not a problem for road traffic surveillance where we can adjust the camera position towards passing cars. However, there are a few potential applications where the rotate invariance would be necessary (mobile cameras for police etc.). For this case, we have made several experiments with Random Forrest algorithm that can detect more than one class of the objects. These individual classes then could mean differently rotated license plates.

3.3.5 Validation of functionality

We have validated the functionality of the IP core versus the IP core's reference C and Matlab implementation. Results of these applications were compared with simulation output of Vivado VHDL simulation. The images used for validation were captured in real traffic under different weather and lightning conditions.

We have also performed a camera test in our lab on a special testing device called "barrel" where the licence plates are attached to barrel's surface simulating approach of vehicle by rotation. We are currently planning the environmental tests to validate other aspects of our camera demonstrator such as camera software and hardware reliability, temperature range etc.

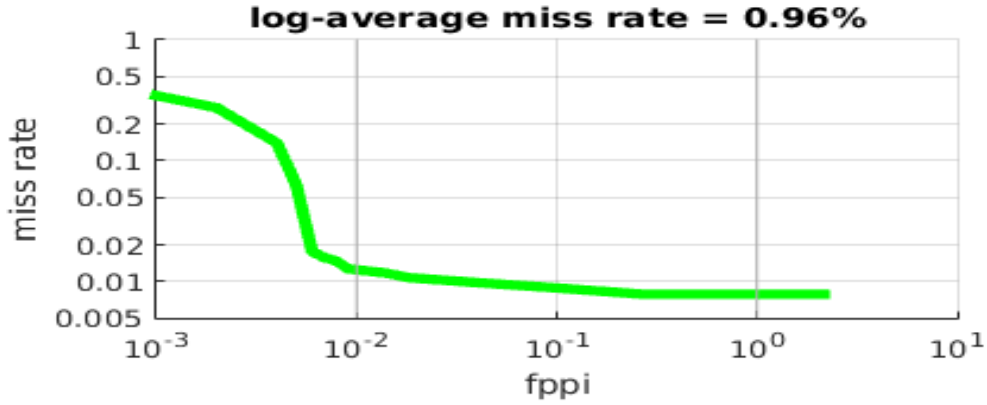


Figure 16: ROC curve of the license plate detector

3.3.6 Verification of robustness and reliability

We have evaluated the detector in real light condition to test the detector robustness in HDR mode. These images were captured on a location where the CAMEA has the long-term bad detection accuracy due to unsuitable mutual position of the camera and the sun. We performed a tests over real images captured on location on the highway. Table 2 shows a rapid increase of detection accuracy in the HDR mode. The overall accuracy has increased and especially in the problematic time period it can be seen that the standard camera mode was not able to detect any license plate.

Time	No. of images	single exp. time Error (%)	HDR - 3 expositions Error (%)
Night (1:00 - 3:00)	200	8,5	5
Day (11:00-13:00)	200	6,5	3,5
Problematic time 18.00-19.00 (Sun behind camera) – cloudy	200	7	2,5
Problematic time 18.00-19.00 (Sun behind camera) – sunny	200	100	39,5

Table 2: License plate detection accuracy with or without HDR feature in different daylight conditions

3.3.7 Validation of performance metrics

The expected performance goal was real-time video object detection at a resolution of 1280x720. Theminimal frame-rate commonly used in the current system is 20 FPS. Moreover, for HDR mode we need twice or three times more input images - 40 and 60 FPS respectively. According to the specifications, the IP core of the Waldboost detector is able to process up to 66 FPS so we should have potential performance reserve. This reserve can be even higher (approx. 100FPS) in case we know the approximate size of a license plate and then we can avoid full-downscaling of an input image.

We have tested the real performance on a "barrel" test device (mentioned above) simulating real traffic. The IP core was accompanied by hardware performance counters which served for measuring the number of clock periods needed for processing the whole image. They confirmed the performance from IP core specification - the core finished processing in advance before the next image arrived from the CMOS even in the worst case (several license plates in one image).

The expected detection latency is derived from the performance of the detector and its internal architecture. The detector is able to detect more than 66 FPS. Taking this worst-case number and required target resolution of 1280x720 pixels, it is exactly 47520 lines of image per second. While the detector needs up to 28 lines of input images to start the detection (depends on scan window size, could be smaller), the average delay of detection should be at worst the same as a time of receiving those 28 lines from the camera (0,590ms) plus execution of the classifier itself (~30us@200MHz) which means latency of 0.620ms. This time meets the <1ms latency requirement set out in D1.4.

3.3.8 Lessons learned

The most efficient way of reducing power consumption of an image processing system is avoiding regular industrial/personal computers with connected cameras and going for an embedded solution. Using a FPGA we are able to build the all-in-one solution including an image sensor and all the controlling logic. Such a low power system has ability of autonomous operation including accelerated/parallelized image processing e.g. object detection. This is the key for a further application of cameras for traffic monitoring e.g. battery/solar powered sites on roads or vehicles equipped with multicamera systems for mobile surveillance.

Using HDR image capturing (simple multi-exposition or more advanced composition) has many advantages when lightning or weather conditions get worse. This is not rare case and sometimes strong sunlight or rain makes the system blind unpredictably. Not with HDR on board that shows the processing algorithms more details. This definitely allows better results in real scenarios.

3.3.9 Conclusion

We managed to successfully demonstrate real time embedded object detection for traffic monitoring scenarios. As the Waldboost-based detector algorithm has been ported for the Xilinx Zynq, the system showed its performance and power efficiency at the same time. The all-in-one design with all components integrated (CMOS sensor, Zynq chip, peripherials) help to reduction of the overall power consumption as well. Moreover, the system core (object detection) is portable and thus it can be transferred to wide variety types of FPGAs with minor modifications. Finally, full-HDR feature or Random forest algorithm for more object's classes are potential enhancements for the next generation of the camera that improves its features even more.

3.4 Hurja: Smart Surveillance (HSS)

3.4.1 Introduction

The use case for this demonstrator is a Virtual Gateway described in D1.1 where the objective was to identify and count persons who are entering or leaving an area or gateway.

Some changes to the original use case have been made, such as:

- No dedicated HSS mobile application was made
- Facial identification is not used, identification relies on HOG (Histogram of Oriented Gradients) features which are captured from the video frame using background subtraction and searching the resulting foreground layer for chunks of moving objects. See D2.2 and D2.4 for further details of the algorithm.

3.4.2 Requirements

The server collects data from the detections including but not limited to the timestamp, location and the running count of detections from multiple units with cameras. Some cropped images will also be saved for external usage (such as showing the detection on GUI). No video or full size frames are stored from the camera streams.

3.4.3 Objectives

Parameter	Baseline	Desired values
Image size for processing	800 x 600	800 x 600
Frame rate	15fps	24fps
Video capture processing speed	200ms	200ms
OpenCL support	Yes, via OpenCV	Yes, via OpenCV
Support for multiple mobile applications	No	Yes
Of the shelf components	Yes	Yes

Table 11: Baseline (current) and desired performance values

Objective-1 – Enabling Massive Data Rate Processing

The implemented algorithm pre-processes the images early on to discard frames where there likely aren't any pedestrians to detect. Such frames include frames without any motion or frames where the motion happens in a small area. The images will be sent to the server for further analysis and are cropped from the frame in the area where the motion was observed. The cropped image is also converted to grayscale and compressed using JPG compression to further reduce the overall bandwidth usage. In testing, direct feed video was analysed at 15fps, which is sufficient to detect all possible objects that pass by the camera. Compressing the frame and using external server to further analyse the image speeds up the data processing on a Raspberry Pi platform(RPi).

JPEG compression

One part of the person identification tool is the module for data transfer, where individual frames from the video feed are compressed to minimize the bandwidth and storage requirements. Here, the JPEG approach was chosen, which provides required functionality.

The frames are compressed using JPEG compression, attaining up to 28 % lower file size for a single image, when the JPEG quality parameter is set to 30. The compression is done on the RPi while encoding the image to base64 string. The resulting string is on average 25 % shorter than the original uncompressed image string. Bandwidth saved with the JPEG compression can be up to 400MB/hour.

The bandwidth required for the algorithm is reduced by 28% allowing more data to be processed in shorter time. Images with smaller file sizes are also less demanding to process than uncompressed files. [D5.4]

Objective-2 – Achieving Low-Power Consumption

The power requirement for the unit is roughly between990mA and1600mA. The value was calculated by summing up the individual power requirements of the components used in the module. The two cameras required 440mA to 600mA and storage drive requires 100mA to 300mA depending on the usage. Using a Raspberry Pi3 as its core,which roughly draws 500mA. According to theRaspberry Pi Foundation[6] Raspberry Pi cannot draw more power than 1 Amp. Using the abovementioned JPEG compression allows less data to be sent which reduces the required processingon the server[D5.4].

Objective-3 – Composability and Cross-Domain Applicability

The described system can accommodate off the shelf basic components like the Raspberry Pi and simple webcams. The algorithm operates independently and re-ports back to a server, from where the results can be fetched using API. Using the JPEG compression can be used in wide variety of cases where it is not important to show the resulting image but to use it for further processing [D5.4].

Objective-4 – Robustness to Variability

The implemented algorithm is robust against most challenges that might arise during the use case for this demonstrator. The algorithm accounts for the differences between different cameras by using histogram equalization [D2.2]. The algorithm uses the background detection to track moving objects in the foreground. The background subtraction learns the background over time and can adapt to changes in lighting and other conditions, such as moving the camera. JPEG compression reduces the number of random patterns of the image as a side effect, making it robust to variability [D5.4].

3.4.4 Research questions

No research questions were set at the beginning but during the project it came apparent that one research question could've been "how far you can we go with off-the shelf components and libraries to enable pedestrian detection and tracking". The answer to that is that with some difficulties and hurdles, it's possible to build a lightweight and easy to set up system with low power requirements with the right components and libraries.

From day one we have been using the Raspberry Pi (RPi) microcomputer and most the problems came with its rather low processing power which could just barely handle real time video stream from two cameras. After it was updated to revision 2 of the RPi that had more processing power for the same power cost, it was possible to do extra pre-processing on the RPi itself and free resources from the server.

3.4.5 Validation of functionality

The algorithm was tested for the first time live in a sports event in February 2016. The algorithm was used two times during the event, first time when people were entering the event area and the second time when they were

leaving. For detailed description see D2.9. After the event the algorithm has been used in office space for testing purposes.

3.4.6 Verification of robustness and reliability

The unit is enclosed in a weatherproof casing with the International Protection Marking (IPM) of P67 which is reliable against heavy rain and dust proof. The algorithm is robust against variability by using histogram equalization and JPEG equalization to reduce the amount of random noise in the captured images. See D5.4 for further details.

The unit can run forever if plugged into a reliable power source or alternatively, the unit can use its own 16 000 mAh battery which can last up to 23 hours in normal use. The HSS unit can be plugged into the external power bank / battery and the battery into a wall outlet. With this setup and in case of a power outage the HSS unit can continue processing the video feeds.

3.4.7 Validation of performance metrics

The performance of the algorithm using the RPi2 and two cameras was on average 15 fps with the original image resolution of 1920 x 1080 pixels. The images were downscaled to 800 x 600 pixels right after capture. Results obtained from testing the algorithm with two active live video streams where multiple pedestrians were detected walking with varying light conditions for 10 minutes. The average fps was then calculated from the results.

3.4.8 Lessons learned

When working on HSS system development we learned that it is possible to utilize off-the-shelf components and open source software libraries to build a system for tracking and counting pedestrians with multiple camera modules. The components were selected focusing to the portability and low power requirements, which was a learning experience on its own. We also learned that some of the used open source algorithms were combined in a way that had been used before in different purposes such as background subtraction with hog detection.

3.4.9 Conclusion

We managed to build a pedestrian detection and counting system with off-the-shelf components. The algorithm running on the Raspberry Pi runs OpenCV based algorithm to determine if there's anything interesting (i.e. large moving object) in the image. We came to conclusion that by using a pre-trained classifier running on an external server to combine all the detection results, it's possible to count multiple unique pedestrians across multiple camera systems by running the same basic detection algorithm.

3.5 VTT: Logical Analysis of Multimodal Camera Data

3.5.1 Introduction

The ALMARVI contribution of VTT addresses the security/surveillance and monitoring demonstrator with an application use case that focuses on logical analysis in a multi-camera surveillance system. The logical analysis entails the usage of camera data to extract machine level data, and analyses this data to support the surveillance system users in the scope of the ALMARVI platform.

3.5.2 Requirements

The demonstrator focuses on multi-camera object recognition and tracking in cameras and between cameras to get 3D context awareness of the person's motions. The demonstrator utilizes many camera nodes that are connected to a central processing unit, and therefore presents distributed and parallel processing. The demonstrator presents algorithms such as automatic calibration to get sense of 3D space, and person recognition and tracking. The challenge is to get the 3D sense and object recognition tracking accurate enough and tackling performance challenge of integration of many video streams.

Deviation from original plan: scrapping of infrared and depth cameras, and drawing 3D/depth information from normal video feed (no need for specific depth cameras). This new approach is more in focus with the overall ALMARVI project scope (video data and its algorithm). Moreover, the new system is more easily configurable to existing camera systems, and requires no extra devices to be acquired. The base approach of the use case of combining many camera sources, many different video data types, and utilizing video depth information remains the same.

3.5.3 Objectives

Baseline & Challenges

• Tackling the shortcomings of single type camera approaches with different types of camera information

• The usage of depth sensors in a surveillance target area to produce accurate 3D position information of object for surveillance system users

• The information from one source is not always adequate. Combining the data from different cameras and sensors will produce supporting data for analysis and reasoning.

Parameter	Baseline	Desired values
Frame resolution	640 x 480	800 x 600 and more
Frame rate (advanced object detection & tracking)	2 - 4,5 frames/s	4 - 12 frames/s
Latency	400 ms	200 ms
OpenCL support	No	Yes
Advanced multi-camera tracking	No	Yes
Off the shelf components	Yes	Yes

Table 12: Baseline (current) and desired performance values

Objective-1 – Enabling Massive Data Rate Processing

Multi-camera object tracking entails a multi-component algorithm for identifying persons, and a methodology to compute camera calibration parameters. These contribute to ALMARVI goals accordingly:

ALMARVI Goal 1: Multi-camera object tracking contributes to ALMARVI goal 1 - Enabling Massive Data Rate Processing by utilizing a heterogeneous and scalable execution platform in the surveillance context, and supporting an application specific architecture for the video processing. The video processing algorithms also support application specific parallelization.

Objective-3 – Composability and Cross-Domain Applicability

Multi-camera object tracking entails a multi-component algorithm for identifying persons, and a methodology to compute camera calibration parameters. These contribute to ALMARVI goals accordingly:

And ALMARVI Goal 3: Multi-camera object tracking contributes to ALMARVI goal 3. Composability, Flexibility, and Cross-Domain Applicability by demonstrating interoperability and software portability, so that it can be adapted to heterogeneous execution platforms and products by providing proper interfaces and possessing suitable system architecture. The developed software can also be ported for various heterogeneous execution platforms.

3.5.4 Research questions

• Tackling the shortcomings of single type camera approaches with different types of camera information

• The usage of depth sensors in a surveillance target area to produce accurate 3D position information of object for surveillance system users

• The information from one source is not always adequate. Combining the data from different cameras and sensors will produce supporting data for analysis and reasoning.

3.5.5 Validation of functionality

Validating the functionality of 3D multi-camera object recognition and tracking in cameras and between cameras is done by:

1) Initial system calibration based on computing the 3-d transformation parameters of a chess-board target template as seen with at least two cameras at a time and find the mutual transformation matrices between the two cameras as a difference transformation.

2) Possible iteration of the transformation between cameras based on a list of object point correspondences in synchronised two camera images. Points correspondences could be indicated manually, but it is also possible to use automatic detection of correspondences, e.g. by observing the images produced by a point-like flash light moving in the 3-d scene and seen by at least two cameras simultaneously.

The final stage uses the resulting transformation matrices between cameras for computing the sparse geometric interpolation grids between the corresponding camera images for all depth ranges. Two-way interpolation is done based on the forward transformation (say from left to right image) and the reverse transformation (say from right to left image).

Performance is not critical for doing the calibration calculations. However, the improvement of accuracy and the possibility of using it for checking transformation accuracy as with phase 2 is essential. Also, phase 2 was developed as a tool to experiment with possible automated calibration methodology (work in progress). The separation of phase 3, i.e. pre-computation of geometry transformations and using only interpolation with real-

time depth estimation has proved to be an essential architectural improvement for the simplicity, performance and parallelization prospects for the real-time depth calculation task.

3.5.6 Verification of robustness and reliability

Validating the reliability of multi-camera object recognition and tracking in cameras and between cameras is done by: The first phase is todetect upper bodies and faces from it using HAAR detectors. A probabilistic colour histogram (PCH) is calculated for the upper bodies. PCH provides a statistical distribution of the colours appearing on the subject by utilizing a k-nearest neighbour classifier (KNN). Another method used is local binary patterns (LBP) that transforms the image capture of a face into a one dimensional feature vector. Both PCH and LBP are stored into a database for later use. A web based visualization utility was developed for demonstration and result analyzation purposes. The utility fetches the results from the database and user is able to select an observation for more detailed inspection. The user is then able to fetch the best matches for the person observed based on PCH, face LBP or both.

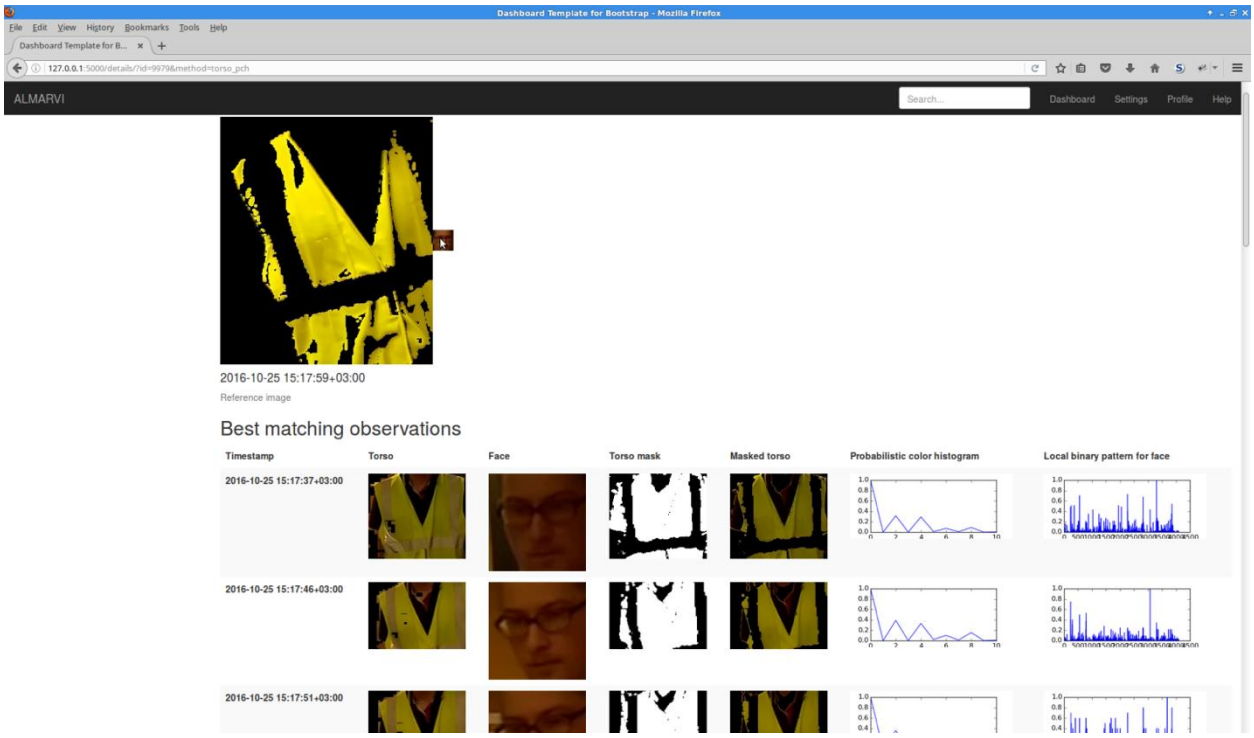


Figure 17. Fetching the results for an observation and visualization of the nearest matches.

The robustness of the system was evaluated with the approach of accuracy. PCH based on kNN classifier was used to calculate average euclidean distances between the torso PCHs isolated from the same test videos as above. The PCHs of the torsos were averaged from the whole test video and the distances were calculated from the averaged histograms. The results are presented in the following table. The simulation tracking test with two videos was also done with kNN classifier. KNN classifier was trained with training data. After this the training data was used to evaluate the kNN classifier itself to see how well it adapts to the colour recognition task. **The accuracy results for kNN was 0.934.**

3.5.7 Validation of performance metrics

Evaluation of 3-d depth estimation methods and algorithms based on passive scene observation by two or more calibrated cameras. The target application is the identification of depth and movement of persons and other moving targets in surveillance-like contexts. The adopted methodology is divided into two tasks, i.e. 1) on-line measurement of dynamic depth data, 2) off-line set-up of cameras so that depth information can be collected.

Real time depth measurement consists of the following stages, i.e.

1) identification of changing regions within each camera images (based on pixel-wise quartile-based filtering)

2) deducing depth for changing pixels from each camera by comparing similarity of pixels between a pixel from one camera and the interpolated corresponding pixels in their image produced by another camera per depth range as produced by a precomputed lower resolution interpolation grid per each pair of cameras

Phase 1 can be made fully parallel for any subregions of the image. The algorithm relies on statistical similarity of pixels in majority of positions between conscutive frames. E.g. with an Intel Xeon 3,2GHz processor one frame can be processed in 1...2 seconds after the initialisation phase by a single core using HD frames of size1920x1080 pixels. Assuming 12 cores to be used, of the order of 25 ... 50 half-size frames of size 960x540 can be estimated to be processed at best in real time (50 frames per second) or nearly real time (25 frames per second).

Phase 2 can be run in about 90 seconds for a typical HD input image (size 1920x1080) on the processor and single core mentioned above. Also this part is feasible for highly parallel realisation. In principle, depth correlation could be run in parallel for each pixel position and each depth range. For instance, if there are 250 candidate depth ranges, the work load could be divided into about 500 million initial interpolation and colour comparison task per the pixels of the input image and the depth candidates could the directly be collected as a 3-d array of the same size for phase 3.

3.5.8 Conclusion

The demonstrator utilizes many camera nodes that are connected to a central processing unit, and presents algorithms such as automatic calibration to get sense of 3D space, and person recognition and tracking. The demonstrator is up and running, a 3D sense is created, and persons are identified and tracked from camera to another.

- The system is able to extract machine level data, and analyse this data to support the surveillance system users in the scope of the ALMARVI platform.
- The system draws 3D/depth information from normal video feed.
- The system is configurable to existing camera systems, and requires no extra devices to be acquired.
- The system utilizes different types of camera information from many cameras
- The 3D sense and object recognition tracking accuracy is satisfactory
- Performance challenge of integration of many video streams with heavy algorithm calculations remains to a degree, but the developed algorithms are made to be parallelizable which provides performance boost.

3.6 UTIA: Protection of Walnut Tree Harvest Against Birds

3.6.1 Introduction

The UTIA demonstrator focuses on implementation of the execution platform with hardware acceleration of image/video processing algorithms as well as on design methodology used for implementation of algorithms for the platform. Special attention is focused on portability of results between FPGA boards, low power and high performance requirements posed on video processing solutions. We are demonstrating capabilities of our platform by implementing three hardware accelerated algorithms. They can be demonstrated individually or implemented together if ported to bigger FPGA device. The first algorithm is a simple Sobel filter based motion detection, the second is background subtraction derived from OpenCV and the third is object detection derived from the OpenCV implementation as well.

All three algorithms are suitable to be used in the ALMARVI demonstrator Walnut Harvest Protection against birds. This document will focus on object detector implementation while the details for first two are given in ALMARVI deliverable D5.4.

As a starting point, ALMARVI Python Camera Platform (APCP) [UTIA1] was imported to the Xilinx SDSoC tool and the object detection hardware accelerator was implemented on top of it. The APCP platform was shared in ALMARVI among partners and thus each system implemented on top of it will be compatible with our accelerators at the Vivado tool level once they are exported back out of the SDSoC tool.

Description of object detection demonstrator

The ability of APCP platform to be used for detection of birds for protection of walnut tree harvest will be demonstrated. The intended configuration of the demonstrator is shown in Figure 18. The camera hardware is a smart camera prototype consisting of:

- Trenz TE0701-5 Carrier board,
- TE0715-3-30 FPGA SoM with XC7Z030 FPGA,
- FMC IMAGEON Video In/Out Extension board, and
- Python 1300 camera sensor with fixed focus optics

The camera faces a tree to be protected or place nearby (or underneath) where birds are usually present and can be detected. The bottom part of the figure shows all intended functions which our use case may need for its operation. In this document a description to the Object Detector block is given.

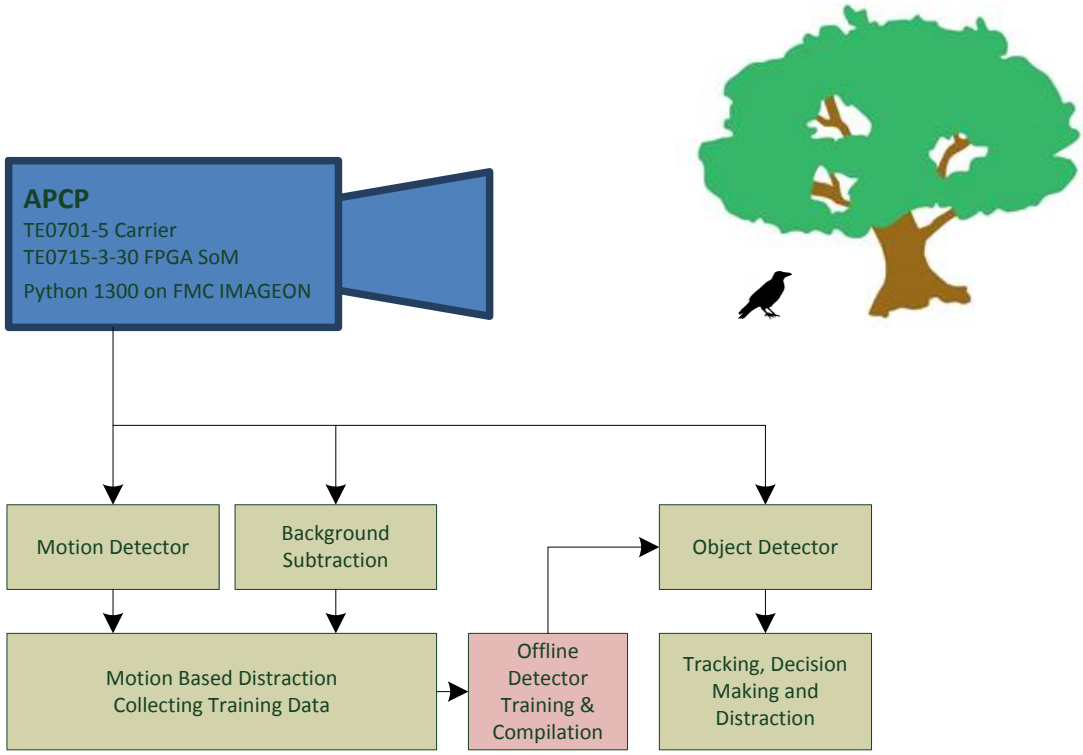


Figure 18: Walnut harvest protection use case architecture

The APCP hardware setup can be found in Figure 19. The setup shown in the figure was used for measurement of the total power consumed by the hardware.

The hardware accelerated object detection shown in Figure 20 was implemented using the Xilinx SDSoC Tool. The process is divided into six streaming operations chained together:

1. Video frame is read from memory to stream and converted to monochromatic 1 byte per pixel by the block **MonoConversion.**
2. Monochromatic frame is equalized by the **Histogram Equalization** operation. The implementation of the equalization from Xilinx HLS Video Library [UTIA2] was used. The library provides synthesizable versions of selected OpenCV 2.4.5 functions. The core was wrapped with interfaces for the Xilinx SDSoC tool for integration to video processing stream.
3. **Subsampling** block was implemented by us to provide simple alternative to OpenCV operation resize which is too generic, hardware demanding and it is not fully pipelined. Operation was designed directly in SDSoC tool. The purpose of the operation in the chain is to match size of detected objects in given frame with object detector window size.
4. **Gaussian Blur** block is another core from Xilinx HLS Video Library wrapped by SDSoC interfaces. It is used to smooth the subsampling output. The Gaussian kernel of dimension 5x5 was used.
5. **Integralf** operation computes integral image and squared integral image for fast evaluation of features in successive cascade detector. The source of **Integral** operation from Xilinx HLS Video Library was modified to support floating point numbers instead of doubles. This modification is possible because we have input of maximal size 1280x1024 of byte pixels. Thus maximal value of sum output is 1280x1024x255 which fits to integer range and its square is in range of 32 bit floating point.
6. **Object detection** core uses integral image to find candidate objects by evaluation of cascade classifier. The classifier must have identical structure as OpenCV 2.4.5 cascade classifier. The cascade in XML format must be first converted to ROM memories used by Vivado HLS tool for implementation of the detector.

For that purpose we have developed utility which automatically performs conversion. Moreover, each stage of the cascade is generated also as a macro which can be used for fully pipelined implementation on the expense of more FPGA resources being used (this implementation we refer as fast stages). The influence of this possibility on performance and power will be investigated later in this document.

7. Operation **Paint BBox** is software function from OpenCV Library which is used by ARM core to mark detected candidates in analysed frame for demonstration purposes.

Figure 19: Used demonstrator hardware: Python1300 sensor with optics attached to FMC IMAGEON extension board (red PCB), power measurement, Ethernet connectivity, 2x HDMI output for demo outputs, TE0715-3-30 SoM (under black passive cooler).

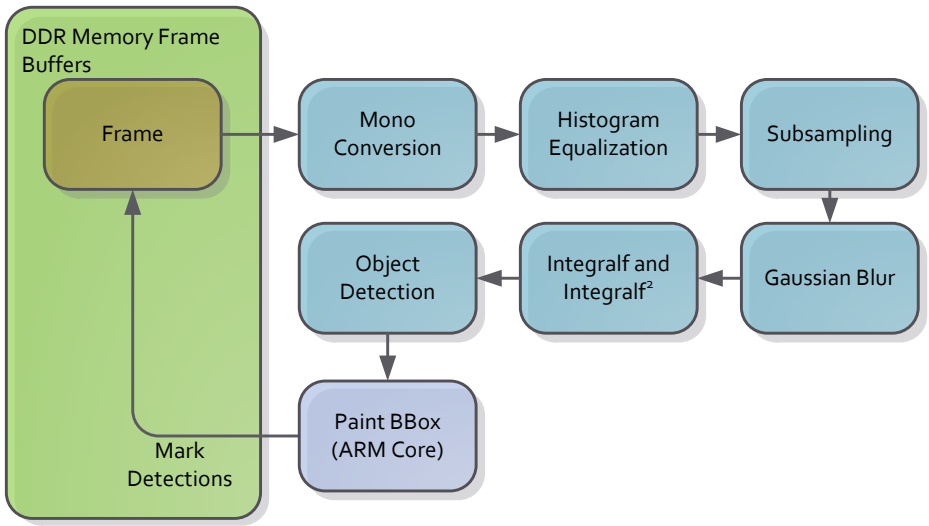


Figure 20: Frame processing stream for object detection

As it has been stated, during compile time the designer can decide to implement classifier stages as fully pipelined hard wired macro or as loop evaluating features in that stage sequentially in independent iterations. In our implementation of the cascade the fully pipelined stages must precede all remaining iterative stages. The number of fast stages to be used is given as compile time constant. For the evaluation of the demonstrator,the face cascade provided in OpenCV 2.4.5 and simple bird cascade was used. The results are summarised in Table 13, Table 14 and Table 15.

In the first table, the performance and power consumption of two detectors are presented. Values are measured for two sizes of detection window. The total power consumed by board is independent of both window size and cascade parameters. We estimate that for 6,48 W total power the consumption of the FPGA is less than 5W.

From the second table can be seen that by adding one stage of classifier improve performance of face detector 1,5x on the expense of 1,7x more resources used. For the bird detector case, one fast stage reaches 3,2x performance on expense of 1,4x more resources used. Adding one more fast stage improves performance insignificantly but consumes 2,5x more resources than baseline implementation. It is clear, that in both cases one fast stage is the most effective solution if the performance is the most important factor.

	Cascade parameters		Implementation	WindowSize 143x143	Window Size 26x26	Maximal Total Power
Detector	# of Stages	# of Features in Cascade	# of Fully Pipelined Stages	FPS	FPS	[W]
Face	22	2135	1	15	-	6,48W
Bird	19	309	2	71	15	6,48 W

Table 13: Hardware accelerated object detector performance and power comparison

Bird detector	Detector Implementation		
Fast Stages	0	1	2
FF	18002	25737	45071
LUT	16259	23850	46568
BRAM	157	157	157
Performance and power			
Frames per second	21,6	70	71
Energy per pixel [nJ]	222,5	70,5	69,6

Table 14: Bird detector resources and performance.

Face Detector	Detector Implementation	
Resources	Iterative	1 fast stage
FF	18020	30784
LUT	16356	28632
BRAM	220	220
Performance and power		
Frames per second	11	16
Energy per pixel [nJ]	441	307

Table 15: Face detector resources and performance.

3.6.2 Requirements

Original use case requirements were summarized in ALMARVI deliverable D1.1. Following list is providing results reached in ALMARVI project by UTIA and their discussion. The rectangle with original D1.1 requirement is leading each individual item in the list:

Functional requirement 1

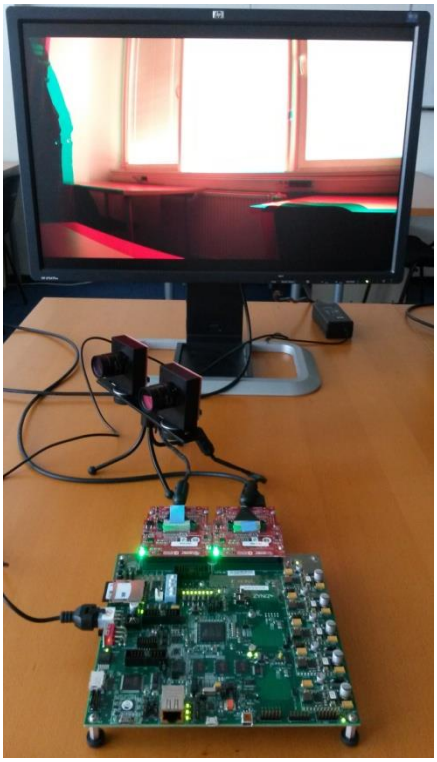
Original D1.1 Requirement
Description: Color video sensor 1920x1080p60 directly connected to FPGA pins with secured portability to at least two FPGA boards and FPGA families (Xilinx Kintex and Zynq). **Priority:** High **Type:** HW **Verification procedure:** Demonstration of color video sensor driving the Video sensor input block on Xilinx Kintex and Zynq FPGA on KC705 board and ZC702 board.

We have implemented number of camera sensor platforms (baseline Vivado designs with support for Camera sensor and video capture chain containing VDMA). Each platform uses at least one HDMI output and HDMI input or camera sensor input. They also differ in used FPGA and development board. The platforms however share support for FMC IMAGEON extension board (HDMI and Camera interfaces) and basic video capture stream with VDMA core in the middle. It allows building hardware accelerators processing frames of captured video independently of video input and output synchronization.

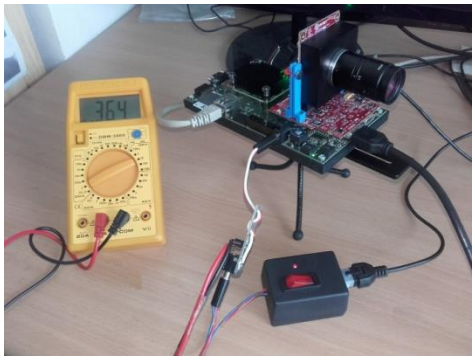
List of supported camera sensors:

- FMC IMAGEON extension board, Vita2000 sensor (1920x1080p60), Supported also stereo for ZC702 and KC705
- FMC IMAGEON extension board, Python 1300 sensor (1280x1024p60)
- FMC IMAGEON extension board, Toshiba Full HD colour video sensor with fixed resolution (1920x1080p60).

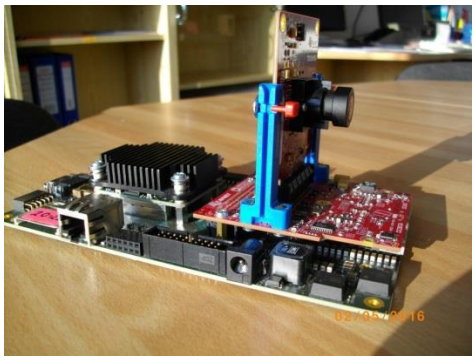
The detailed analysis of each platform and used camera sensors is given in ALMARVI deliverable D3.5, chapter 7. The demo version of the platforms with detailed documentation can be found on UTIA web pages [UTIA6].



a) Stereo Vita 2000 Camera Sensors (synchronous capture), ZC702 Development board



b) ALMARVI Python Camera Platform (APCP), Power measurement setup



c) Toshiba Full HD Camera, TE0701-5 Carrier with TE0715-3-30 SoM

Figure 21: Hardware used for camera platform alternatives in ALMARVI

Functional requirement 2

Original D1.1 Requirement
Description: Implementation of video stream block instances (Color Filter Array Interpolation; RGB to/from YCrCb Color-Space Converter; blocks Chroma Resampler; Video DMA; Video Timing Controller; HDMI and DVI Input/output) supporting the 1920x1080p60 video with the 148,5 MHz video clock frequency on at least 2 FPGA boards and 2 FPGA families (Xilinx Kintex and Zynq FPGA). **Priority:** Must have **Type: HW and SW** **Verification procedure:** Demonstration of complete color video chain for Xilinx Kintex and Zynq FPGA on KC705 board and ZC702 board.

All required cores were successfully implemented at required speed. All demo platforms provided by UTIA are using them. The list of platforms is summarized in ALMARVI deliverable D3.5, chapter 7. Their demo versions are downloadable from UTIA web pages [UTIA6].

For Kintex support we have implemented low resolution video chain with web server based control GUI as one of early demonstrators and Full HD stereo Vita 2000 camera sensor support, although not synchronized. In the next development the support for Kintex was not used anymore because Kintex device family is not supported in Xilinx SDSoC tool.

Carrier Board	FPGA Module	FPGA device	HW Acceleration support		Description
			SOSoC	EdkDSP	
TE0701-5	TE0720-03-1QF TE0720-03-1CF TE0720-03-2IF	XA7Z020-1CLG484Q XC7Z020-1CLG484C XC7Z020-2CLG484	Yes	Yes	Automotive grade Commercial grade Industrial grade
TE0701-5	TE0715-3-30-1	XC7Z030-1SBG485I	Yes	Yes	Industrial grade
ZC702	-	XC7Z020-1CLG484C	Yes	Yes	Xilinx development board
KC705	-	XC7K325T-2FFG900C	No	Yes	Discontinued support in ALMARVI, No SDSoC possible

Table 16: Summary of supported FPGA devices in UTIA platforms (baseline designs with HDMI/Camera In and HDMI out chain)

Functional requirement 3

Original D1.1 Requirement
Description: Detection and marking of the moving birds in the 1920x1080p60 in the video stream with the rate of at least 10 FPS with the power consumption of the FPGA max 4.2 W. **Priority:** Must have **Type:** SW and HW **Verification procedure:** Measurement on the demonstrator.

Our implementation of object detector originates from OpenCV. Its performance depends on scale of image for which is object detected, on detection window size and on complexity of cascade found in training process.

We have implemented hardware accelerated cascade detectors for face and bird detection. For the face detector, pre-trained cascade from OpenCV library was used. The bird detector was trained on simple images of tree and birds. In that case, full flow from construction of training data sets to automatic generation of hardware accelerator parameters and structures was developed.

Both detectors are using window size 20x20. When bigger objects must be detected, the scale of input image must be changed to size in which object matches window size. We have implemented core which can subsample the input image to desired scale. We have also implemented utility which transforms cascade parameters in OpenCV XML format to parameters of the accelerator implemented in Vivado HLS. The utility also provides fully pipelined macros as an alternative implementation of individual detection cascade stages. After that, hardware accelerator can be compile time configured to balance performance and resources for each detection cascade case by employing more or less pipelined 'fast' stages.

Power measurement of the FPGA device alone was not possible for our hardware platforms since all power lines are not accessible on surface layers of PCB. For that reason we measure total board power. It consists of FPGA power as well as power for DCDC converters including losses and peripheral devices on carrier board. As the power consumption of the carrier board thepower consumption without FPGA SoM snapped in can be used. Measured results are presented in Table 13, Table 14 and Table 15.

Functional requirement 4

Original D1.1 Requirement
Description: Data fusion of the video processing and the audio processing.
Priority: Must have
Type: HW and SW
Verification procedure: Functional test on the demonstrator.

For the detection of birds we have used more usual surveillance solution based solely on image information. For that reason the data fusion of audio and video was not implemented.

Functional requirement 5

Original D1.1 Requirement
Description: Direct access to the video stream data from EdkDSP accelerators
Priority: Must have
Type: HW and accelerator firmware SW.
Verification procedure: Functional tests on the demonstrator.

For integration of EdkDSP to our hardware platform we have used global shared DDR memory to which are mapped all devices in the system including EdkDSP, SDSoC hardware accelerators, Zynq ARM cores and VDMA unit. Thus all of them can directly access frame data stored in DDR memory by the VDMA unit. User is responsible for access synchronization and CPU cache management.

Functional requirement 6

Original D1.1 Requirement
Description: C compiler for the EdkDSP microcontroller PicoBlaze6.
Priority: Must have
Type: SW
Verification procedure: Functional tests on the demonstrator.

UTIA provides C compiler, API and examples for its EdkDSP platform.

Functional requirement 7

Original D1.1 Requirement
Description: 1 Gb Ethernet connectivity, with DDR3 based file system and www browser GUI.
Priority: Must have
Type: SW and HW
Verification procedure: Functional tests on the demonstrator

Each supported development board: ZC702, KC705 and TE0808 carrier is equipped with 1GB Ethernet interface and can support Linux with web server GUI.

Non-functional requirement 1

Original D1.1 Functional requirement
Description: Autonomous function in standard day light conditions with possibility of remote control of device status via Ethernet, including the remote switch off of the device by the operator. **Priority:** Must have **Type:** Given by conditions specific for the application **Verification procedure:** Functional tests on the demonstrator

Our implementations of image processing are providing all basic controls to set camera exposure, analog and digital gains of sensors and other parametrization. For simplicity, in our demos we implement serial terminal based user interface. We have also shown implementation of web based GUI in our early demonstrator prototype [UTIA5]. The same technology can be used also in the case of Zynq based systems. For each of our development boards we provide Linux support with 1GB Ethernet.

Non-functional requirement 2

Original D1.1 Functional requirement
Description: Rate of the false alarms max 1 per day. **Priority:** Must have **Type:** Given by conditions specific for the application, to minimize the disturbance by false alarms. **Verification procedure:** Functional tests on the demonstrator.

We have implemented complete top down flow for hardware accelerated object detection. It starts by collecting onsite training data. It can use for that purpose simple motion detection algorithm. After that the collected data can be used for training of the detector. The tools already provided in OpenCV library can be used. After that, UTIA provides utility which automatically generates object detector parameters and macros for fast stages synthesizable by Vivado HLS tool. Implementation of this complete detector training and development flow provides strong support to reach detector designs with best possible reliability.

Non-functional requirement 3

Original D1.1 Functional requirement
Description: Documentation for the integration of the video processing blocks and the EdkDSP accelerator blocks in form of netlist IP modules and SW libraries under the Vivado 2013.4 system design flow and the SDK 2013.4 SW design flow into another video processing or DSP processing scenarios and another prototyping boards with Xilinx FPGAs. **Priority:** Must have **Type:** Design flow, HW and SW documentation. **Verification procedure:** Documentation tested by an external application engineer on the identical board and HW and tested also on different FPGA board with Xilinx FPGA.

In the development process, UTIA have provided number of downloadable technology demonstration packages [UTIA6] for all boards used in ALMARVI. They include hardware accelerators prepared for bird detection use case as well as more generic EdkDSP accelerators. Each downloadable package also contains detailed documentation which is necessary to reproduce our results in other labs worldwide.

3.6.3 Objectives

Based on our pre-ALMARVI results were proposed ALMARVI goals. It was expected that new FPGA technology, development tools and ALMARVI developed architecture will be able to meet them. The original, expected and reached values are summarised in Table 17.

Parameter	Baseline	Desired values	ALMARVI Results
Image size	1024x768 RGB 888 bit	1920x1080 RGB 888	Yes
Video frame rate primary	60 frames/s	60 frames/s	Yes
Rate for moving object detection and labeling	5 frames/s	10 frames/s	Yes, but dependent on scale of detector window
FPGA power consumption < 5W	Yes	Yes	Yes, 6.48 W Totalpower means FPGA <5W
FPGA Power per Gflop (peak performance)	1 W/GFLOP/s	0.3 W/GFLOP/s	Birds: 70 nJ/pix Faces: 307 nJ/pix (faces)*
FPGA Power per Gflop (average performance)	2 W/GFLOP/s	0.6 W/GFLOP/s	EdkDSP Yes
Vivado 2013.4 design tool chain	No	Yes	Yes, supported up to Vivado 2015.4
Compatibility based on Xilinx Video I/O IPs	No	Yes	Yes
Portability to different Xilinx boards	No	Yes	Yes, see requirements
Support for High Level Synthesis (C to HW)	No	Yes	Yes, using Xilinx SDSoC tool
Compatibility with Ultra Scale FPGA (20nm)	No	Yes	Yes, preliminary tests passed
Of the shelf components	No (video sensor)	Yes	Yes, Python 1300 used amongst partners in ALMARVI
Internet browser based user interface	No	Yes	Supported
Gigabit Ethernet point to point support	No	Yes	Yes
FPGA technology	45nm	28nm, (UltraScale 20 nm)	Yes

Table 17: ALMARVI results compared to original goals

*We had to stick to measure energy per one processed pixel since the custom FPGA designs contain mixture of integer, fixed point and reduced precision floating point operations thus it is impossible to take them as equal to standard floating-point operations. Moreover, we are using library of stream processing operations from Vivado HLS for which might not be always possible to count operations (cores can be precompiled in library, can have variable execution length etc.)

Objective-1 – Enabling Massive Data Rate Processing

To enable massive data rate processing we have used:

- Device from Xilinx Zynq7000 family. It combines ARM cores with FPGA fabric for implementation of hardware accelerators.
- The design tool flow based on Xilinx Vivado 2015.4 and Xilinx SDSoC 2015.4 tools.
- Petalinux 2015.4 OS support.
- Prefer algorithms which can be fully pipelined and have other features favourable for efficient FPGA implementations.
- Direct streaming pixel data between hardware accelerators without using VDMA and frame buffers in DDR memory

We have shown in our demonstrator that processing of frames at full 1920x1080 size and 150MHz clock speed is achievable for all used video processing cores. Even the object detector core can reach full speed, but on the expense of huge resources used. The background subtraction core can reach full speed as well but the data bandwidth to feed pixel models in and out of DDR memory is too narrow for full HD computation.

Objective-2 – Achieving Low-Power Consumption

Chosen generation of FPGA devices together with optimal implementation of hardware accelerators allowed us to reach low power execution. The power measurement results can be found in Table 13, Table 14 and Table 15.

Objective-3 – Composability and Cross-Domain Applicability

The objective was reached by sticking to the tool chain based on Xilinx Vivado and SDSoC tools and UTIA tools for EdkDSP.The tools can be used to integrate the complex designs composed of heterogeneous cores implemented in C/C++ and glued to the system by SDSoC tool automatically. The baseline platform implemented in Vivado for whole group of hardware boards allows porting these high level functions easily between different hardware boars.

For the walnut harvest Protection use case we had to compose system of hand written cores for: color to gray conversion, subsampling and cascade classifier. They were attached to Xilinx Video Library cores for: histogram equalization and Gaussian blur filter. In the end, also library function for integral image was used after its modification to floating point arithmetic. For our design we have shown the possibility to export the result back from SDSoC tool to Vivado project. After that the EdkDSP accelerators can be integrated to the whole platform as well.

Objective-4 – Robustness to Variability

The basic methods for detection of birds are based on: motion detection – in this case the bird itself is not detected thus every motion in monitored area is assumed to be a bird, object detection – this method runs detector which detects birds based on features computed for image. The features and detector structure are first found by machine learning process where positive and negative samples must be provided for offline detector training.

In order to provide robustness of solution for our use case we have implemented two methods of motion detection, the first one compares edges of two consecutive frames and the second uses background subtraction for detection of moving object. The motion detection is complemented with third implemented algorithm for object detection. In order to reach robustness of the bird detection in various conditions we have implemented all three solutions which can be switched or reconfigured to FPGA fabric depending on real world conditions.

General Objectives

In this use case demonstrator we have shown the possibility to take complex image or video processing algorithms and implement their embedded hardware accelerated versions. This way many other embedded solutions with high performance and low power per processed pixel can be implemented. The provided solutions are

interconnecting together embedded ARM cores, dedicated hardware accelerators, stream processing, DMA units and other custom accelerators as EdkDSP. Resulting solution reaches efficiency and low power by exploiting the power of the heterogeneous architecture.

3.6.4 Research questions

In the ALMARVI deliverable D1.1 there is provided list of research questions. In this section we try to provide our solutions to them and we also add newly emerging ones. All questions are summarized in following table:

D1.1 Research Question	Resolution
Design of reference test-cases.	Multiple reference designs were implemented in order to get experience in used tools and technology. The individual test cases are documented and as a demos downloadable from UTIA web pages. They include image processing accelerators by SDSoC tool, web based GUI, HDMI video in/out pass-through designs, EdkDSP platform implementation for different Zynq devices with benchmarking applications, ALMARVI Python Camera Platform, etc.
Modelling of the reference “golden” algorithms	We have used golden reference models in C/C++. It is the best solution for the SDSoC tool which can directly run this model on ARM core in Zynq device. Later Vivado HLS can compile hardware accelerator out of it almost without changes in sources. We have used sources from OpenCV library.
Mapping into local memories and data flow of the EdkDSP accelerators.	Our demos show that we have solved connection of the Microblaze CPU with EdkDSP accelerators to the Zynq AXI bus system.
Verification of system composed from the central CPU and multiple accelerators.	As individual golden models in C/C++ (from OpenCV) run directly on target platform on ARM CPUs. The output of their combination can also be verified by running them on ARM. After that, they can be all converted to hardware accelerators to ARM and checked against each other.
We plan to use existing tools from Xilinx: System Generator for the ISE design flow and Xilinx System Generator for the DSP optimized for the Vivado design flow. The methodology for use of this Simulation/Verification/ Automated HDL code-generation needs to be developed in the context of ALMARVI project.	We have used complete design flow for Xilinx 2015.4 version of the tools including Vivado, Petalinux, SDK, SDSoC and EdkDSP compiler, API, and debug analyzer.
Future Research Questions	
In our approach we assume that the object detector is configured by parameters in ROM memory. However, it is also possible to store parameters in RAM and to load and update cascade parameters online. We leave methods how to support it in SDSoC tool to further research.	
We use the camera sensor to capture video with constant exposure, gains, focus, and iris settings. How to adjust these values with respect to external conditions is also topic for future research.	
Yet another challenge will be potential update of the cores for the latest Ultrascale and Ultrascale+ devices including latest versions of the SDSoC tool.	

3.6.5 Validation of functionality

In the development process of the synthesizable cascade classifier we have used OpenCV implementation in C++ sources as golden model to each individual processing steps in Figure 20. The cores are providing identical results to testing set of still images (subset of images used for training of the detector).

3.6.6 Verification of robustness and reliability

The robustness and reliability of the implemented cascade detector is identical to its OpenCV golden model. The detailed description of the algorithm and its objective assessment is provided in original work of Viola-Jones and Lienhart and Jochen-Maydt [UTIA3,UTIA4]. The implemented accelerator was tested against OpenCV golden model and it has identical results on the same images provided as input.

3.6.7 Validation of performance metrics

Our performance metrics can be found in Table 17 in the first 6 rows. By implementation of the detector example application we were able to confirm power consumption and frame processing speed limits. Cores for motion detection and background subtraction were also evaluated and the results are already included in ALMARVI D5.4. All image processing cores meet requested performance limits.

We have also implemented platform including EdkDSP accelerators for different Zynq family FPGAs. The performance of single (8xSIMD) EdkDSP floating point accelerator was measured [UTIA7]. It reaches 0.61 W/GFLOP/s. In extreme cases where communication is not the limiting factor this number can reach as low as 0.3 W/GFLOP/s. All tested demo platforms with EdkDSP accelerators are downloadable from UTIA web pages [UTIA6] including detailed documentation.

3.6.8 Lessons learned

The SDSoC tool provides unprecedented possibility to create hardware accelerators written in “C/C++” language and to automatically generate complete dataflow network around them. The implementation of complex image or video processing for FPGA, however, can be still complex and demanding task. Thus the availability of image/video processing library with HLS synthesizable functions is a key to rapid development. The support for Open Source OpenCV library was tested by our team. We have found that the integration of system composed of Xilinx Video library cores (HLS synthesizable OpenCV cores), hand coded cores and SDSoC tool provides efficient design flow with possibility to debug in software, test in hardware and to incrementally extend the hardware accelerated part of the design.

On the other hand the SDSoC tool introduces limitations on accelerator interfaces and global memory accesses. In our development we have also bumped into some errors in Vivado HLS itself (for example: if the child class of original template class is derived and in source code there is one instance of that child, all HLS pragmas, which are located in constructor of parent class are followed properly. But when the second instance of that class is created, all pragmas seem to be ignored by HLS for that instance).

We have also learned a possibility to take stream processing cores from SDSoC and to make from them pure Vivado HLS based with input/output stream. This allows using them also in FPGAs outside Zynq family. However, we cannot recommend to compose larger systems of such cores since it is almost impossible to foresee all buffering and handshake back pressure effects to integrate them all reliably to one system.

In the development of the object detection algorithm we have found that the different cascade classifiers lead to different optimal number of fully pipelined stages in terms of resources used per FPS speed improvement achieved. We have implemented different combinations of stages for bird and face detector cases and the results can be found in Table 14 and Table 15 respectively.

3.6.9 Conclusion

The object detection based on OpenCV implementation of cascade classifier was trained and used for construction of hardware accelerator capable to speed up the detection process on the prototype smart camera platform. The optimal configuration of the detector in terms of its parallelization, pipelining and resource consumption were found. Individual measurements for two individual cases of cascade were analysed. The first is capable to detect faces and the second is trained on bird image against tree.

It was shown that used hardware and design methodology can benefit largely from:

- Video In/Out platform designed in Vivado tool abstracting frame processing from video I/O hardware
- Possibility to exploit SDSoC tool capability to use Vivado HLS for implementation of a new stream processing IP cores and to snap them into data streaming chain or to generate automatically DMAs to feed them by data
- Petalinux support for the platform
- Availability of software OpenCV library which allows exploiting ofresults of open source community around computer vision.
- Xilinx HLS Video library which implements a subset of OpenCV as Vivado HLS synthesizable functions.
- Capability of SDSoC tool to export result back to standalone Vivado project enables in addition:
    - to involve other Vivado developers to extend the platform by their own cores without licensing SDSoC tool,
    - or to include UTIA EdkDSP acceleration cores in addition to cores developed in SDSoC which is also shown in ALMARVI deliverable D3.6

4. Mobile

4.1 Introduction

We have applied the ALMARVI design methodologies for selected use cases of mobile handset domain and demonstrated customized architectures for the domain using tools designed in ALMARVI. Further, the demonstrator has shown that the developed toolset and methodologies can be leveraged across multiple domains.

Objective was to demonstrate application of ALMARVI tools to cross-domain and scalable execution platform implementation for multimedia and radio processing that can serve different product categories in performance, power, development time, and cost constraints ranging from Radio and Imaging use cases to ultra-low power wearable computers. ALMARVI approach supports a broad selection of heterogeneous acceleration fabrics such as CPUs, DSPs, GPUs, FPGA, but in addition application specific programmable co-processors etc. to provide a good trade-off between the performance/throughput, energy efficiency and reuse via programmability. All targets should be supported from the same C/C++/OpenCL application source code. The selected algorithm benchmarks were used to set the minimum performance requirements for the implemented solutions

Customized processors provide a middle ground between fixed function accelerators and generic programmable cores. They bring benefits of hardware tailoring to programmable designs, while adding new advantages such as reduced implementation verification effort. The hardware of customized processor is optimized for executing a predefined set of applications, while allowing the very same design being used to run other, close enough routines by switching the executed software in the instruction memory. The degree of processor hardware tailoring is dictated by the use case and the targeted product.

In any case, the processor customization process is highly demanding and error-prone, with high non-recurring engineering costs. Moreover, as the design process of customized processors is usually iterative in nature, porting the required software program codes to new processor variations needs either assembly language rewrites or retargeting the compiler. One approach to simplifying the processor customization process is to compose the processor from a set of component libraries and other verified building blocks, thereby reducing the required verification effort. The software porting problem can be alleviated with automatically retargeted software development kits.

TTA-Based Co-Design Environment (TCE) is a processor design and programming toolset which is based on a processor template that supports different styles of parallelism efficiently. TCE enables rapid design of cores ranging from tiny scalar microcontrollers to multicore vector machines with a resource oriented design methodology that emphasizes reuse of components.

- The novelty of mobile handset demonstrator lies in extensive usage of TCE toolset for design and implementation of customized processing solutions for the selected application use cases. During the project the TCE toolset was extended in ALMARVI WP3 and WP4 to meet the requirements of the designers.
- D5.6 has demonstrated the design and implementation of application specific customized parallel processors for the selected use cases. The software is co-designed with the hardware; to efficiently utilize the parallel hardware, it is written in a language that exposes parallelism to the compiler and the runtime.
- The demonstrator focuses on exploring the performance-power trade-offs. During the demonstrator development, multiple variations of both the software and the architecture were produced. Each of the variation presents different performance-power ratios.
- The produced parallel vector machine used in the demonstrator is of state of the art in the category of programmable high performance low power computation. This is achieved by using a static exposed datapath architecture model as a basis for the design.
- In the demonstrator, an established programming standard, OpenCL was used. This was motivated by the ability to partially develop and verify the software outside the processor design flow. The portability aspect is especially useful in this setting where the same software is compiled to hundreds of different processor variations during the design phase. For the implemented processors of the project compiler uses the LLVM project as a backbone and pocl to provide OpenCL support.

- The produced demonstrators are tailored towards integration to heterogeneous SoCs with multiple different devices and a central general purpose processor, all with different ISAs.
- System performance is guaranteed by setting the minimum performance boundaries for the application test cases. The performance of co-optimized hardware and software design is measured automatically by integration server.
- As the system performance is guaranteed by automated continuous integration the power consumption can be guaranteed by synthesizing the design on a selected silicon node and estimating the power performance of the custom processor from post-layout simulations. In addition small scale ASIC prototype of the design have been manufactured.

Table 18: Objectives of the mobile demonstrator

	Coverage of Almarvi Objectives	Mobile	
		Segmentation and LTE receiver	Image and video enhancement
1	Acceleration fabrics		
	Design tools		
	Application specific parallelization		
	Quality - performance trade-off		
2	Low power cores		
	SW quality - power trade-off		
	HW quality - power trade-off		
	Algorithm resilience for power efficiency		
3	Software portability		
	Interfaces for heterogeneous acceleration fabrics		
4	Guarantee system performance		
	Guarantee power consumption		

4.2 NOKIA: Image Segmentation and LTE receiver

4.2.1 Introduction

4G LTE is a standard of high-speed, low latency, data for wide-area cellular communications, and builds upon the technologies developed by 3GPP project. The most demanding and compute intensive algorithms in modern radio receiver relate to signal detection and demodulation. MIMO technique employs multiple transmitter and receiver antennas for transmitting multiple parallel data streams. As the system employs M x N different paths for the signal higher diversity, more reliable communications and higher throughput is achieved. The expense to be paid is the higher receiver complexity with exponential complexity increase to the MIMO and modulation order. LTE provides 10 device categories up-to 452.2 Mbit/s downlink data rate with 4 MIMO layers, and up to 102 Mbit/s uplink data rate with 2 MIMO layers in a single 20 MHz LTE carrier. For 5G systems researchers have suggested massive MIMO approaches, meaning tens of parallel antennas and parallel data streams. State-of-the-art implementations even for 2x2 MIMO receivers employ either custom HW or vector DSPs for signal processing. For implementer, it means RTL level coding or assembly level coding with compiler intrinsic extensions. High-level programming languages or standard OpenCL based programming models are not supported, meaning that there is no portability of implementation, yet alone scalability.

In the second co-processor design, we targeted audio signal processing in a wearable, always-on type of a device. The processor is implementing audio signal processing algorithms such as IIR bi-quads, linear filters, spectral analysis and adaptive filters. The main use case for the audio processing is isolation of the headphone user from environment with active noise cancellation. For realistic virtual reality immersion, the audio plays very critical role as sound-scene will inform the user to which direction to look. The input sample rate of such systems is quite modest compared to wideband radio transceivers, but the use-case requires extremely low energy consumption as well as very short processing latency of below (1/48000) seconds. The main target was to implement the low latency algorithms, such as active noise cancellation, in programmable TTA co-processor with significantly lower latency and with lesser power consumption than with traditional CPU.

The FFT as well as image segmentation use cases were implemented as OpenCL source codes. However, Nokia internal customer finally dropped these requirements and clear focuses were to focus on MIMO Detection from the high-throughput scenario perspective and to audio signal processing from ultra-low power wearable perspective.

4.2.2 Requirements

We had 6 separate requirements, Fourier Transform, MMSE MIMO Detector, LORD MIMO Detector, Audio Signal Processing, Image Segmentation, MEP silicon verification ASICs.

Fast Fourier Transform

Description: FFT co-processor for LTE: (64, 128, 256, 512, 1024, 2048). Co-processor power target < 500 mW.

Priority: Must have -> Nice to have

Type: HW + SW

Verification procedure: Benchmarking and Demonstration. Power consumption is estimated from pre-layout netlist simulation with switching activity.

MMSE MIMO Detector

Description: MMSE MIMO detector for LTE. Co-processor power target < 500 mW.

Priority: Must have

Type: HW + SW

Verification procedure: Benchmarking and Demonstration. Co-processor power consumption is estimated from pre-layout netlist simulation with switching activity.

Layered Orthogonal Lattice (LORD) MIMO Detector

Description: LORD MIMO detector for LTE. Co-processor power target < 500 mW.

Priority: Must have

Type: HW + SW

Verification procedure: Benchmarking and Demonstration. Power consumption is estimated from pre-layout netlist simulation with switching activity.

Audio Signal Processing

Description: Cascade of six second-order IIR sections implementing an equalizer. Co-processor power target < 1mW.

Priority: Must have

Type: HW + SW

Verification procedure: Benchmarking and Demonstration. Power consumption is estimated from pre-layout netlist simulation with switching activity.

Image segmentation 1080p

Description: 1080p video segmentation. Co-processor power consumption target is 1W.

Priority: Nice to have

Type: HW + SW

Verification procedure: Demonstration. Power consumption is estimated from pre-layout netlist simulation with switching activity.

MEP silicon verification ASICs

Description: Minimum Energy Point (MEP) silicon verification. Reduced instruction set functionality ASIC implemented for some of the requirements of chapter 4.1.2. to verify MEP operation. The instruction reduction is done due to the limited on-chip resources of the small-scale ASIC.

Priority: Must have

Type: HW

Verification procedure: Demonstration. Power consumption and execution time are measured from the ASIC.

4.2.3 Objectives

Table 19: Objective of the Nokia Demonstrator

Benchmark	Baseline	Desired values
2x2 MMSE MIMO Demodulation		
Rate	14.4M subcarriers/s	14.4M subcarriers/s
Implementation	HW accelerator	OpenCL + TTA co-processor
Power Budget	500mW	500mW
2x2 LORD MIMO Demodulation		
Rate	14.4M subcarriers/s	14.4M subcarriers/s
Implementation	HW accelerator	OpenCL + TTA co-processor
Power Budget	500mW	500mW
Audio signal processing		
Signal Rate	48000 kHz	48000 kHz
Implementation	C + ARM M4	C + TTA co-Processor
Power Budget	3 mW	1 mW

Objective-1 – Enabling Massive Data Rate Processing

LTE MIMO DETECTION: Table 20 shows the results of these performance benchmarks. The performance requirements of LTE Category 4(2x2 MIMO, 64-QAM modulation, 150 Mbit/s) can be achieved with a single core when using the MMSE algorithm or with three cores when using the LORD algorithm which has better detection performance, and a four-core cluster can exceed the performance requirements of LTE r11(4x4 MIMO, QAM-64-modulation, 600 Mbits/s) when using the MMSE algorithm.

AUDIO SIGNAL PROCESSING: The data rate in audio signal processing is quite small, just 4 input channels 24bit/48kHz sample rate each. However, the difficulty of Active Noise Cancellation lies in ultra-low latency requirement. Lower the latency of the feedback path, the better quality and wider bandwidth response the ANC will have. The actual feedback for the algorithm of choice, non-linear filtered-x LMS, was calculated in fraction of a sample rate (5us) using the parallel processing capabilities of the designed TTA co-processor. However, the algorithmic performance was dominated by the aggregate decimation and interpolation group delay (42us) and even more by the group delay of the used power amplifier and the loudspeaker (150us). Our conclusion was that the overall performance was not at all impacted by digital path design, but the power amplifier and loudspeaker element selection.

Table 20: Performance numbers of LORD and MMSE algorithms running on the processor on different modes and different core counts.

Algorithm	Ntx	Nrx	Modulation	Single-core	Dual-Core	Quad-Core
Lord	2	2	QPSK	122.3 Mbps	226.0 Mbps	445.3 Mbps
Lord	2	2	16-QAM	125.1 Mbps	241.0 Mbps	471.5 Mbps
Lord	2	2	64-QAM	74.4 Mbps	145.7 Mbps	286.3 Mbps
Lord	2	4	QPSK	84.9 Mbps	161.1 Mbps	314.3 Mbps
Lord	2	4	16-QAM	103.4 Mbps	199.6 Mbps	391.0 Mbps
Lord	2	4	64-QAM	69.3 Mbps	135.9 Mbps	267.1 Mbps
MMSE	2	2	QPSK	217.9 Mbps	380.1 Mbps	703.1 Mbps
MMSE	2	2	16-QAM	426.7 Mbps	746.4 Mbps	1382.0 Mbps
MMSE	2	2	64-QAM	548.6 Mbps	976.2 Mbps	1849.7 Mbps
MMSE	2	2	256-QAM	570.9 Mbps	1042.9 Mbps	1979.2 Mbps
MMSE	4	4	QPSK	60.1 Mbps	117.2 Mbps	229.0 Mbps
MMSE	4	4	16-QAM	119.3 Mbps	232.4 Mbps	454.1 Mbps
MMSE	4	4	64-QAM	171.5 Mbps	334.7 Mbps	639.3 Mbps
MMSE	4	4	256-QAM	209.5 Mbps	409.3 Mbps	784.3 Mbps

Objective-2 – Achieving Low-Power Consumption

LTE MIMO DETECTION: The four-core ASIC configuration of the LTE processor was synthesized and placed and routed with Synopsys tools using a 28nm Fully Depleted Silicon On Insulator (FDSOI) process technology. Clock gating and multi-threshold voltage optimizations were enabled in synthesis. Operating conditions were set to 1V supply voltage and 25◦C temperature. The routed design achieves a clock rate of 968 MHz and has a cell area of 2.47mm2 at utilization of 71%. A layout image is shown in Figure 22. Power estimation was performed in IC Compiler based on switching activity extracted from RTL simulations. Each kernel execution exhibits a warm up period during which the instruction caches are filled; this period was omitted from the extracted switching activity in-order to obtain power figures representative of prolonged execution. Table 21shows the simulated power usage and power-efficiency of a four-core version of the chip.

AUDIO SIGNAL PROCESSING: Low power consumption has been measured from post-layout simulations of the architected processor design. The measurements revealed that leveraging the sub threshold voltage design with 0.4V operating voltage the processor was running with just 300µW, clearly under our requirement of 1mW.

Table 21: Post layout power usage numbers of LORD and MMSE algorithms running on the processor on different modes.

Algorithm	Ntx	Nrx	Modulation	Power (mW)	E/bit (pJ)
LORD	2	2	64-QAM	295.3	905.7
LORD	2	4	64-QAM	269.3	1008.2
MMSE	2	2	64-QAM	226.2	122.3
MMSE	2	2	256-QAM	240.7	121.6
MMSE	4	4	64-QAM	245.1	367.0

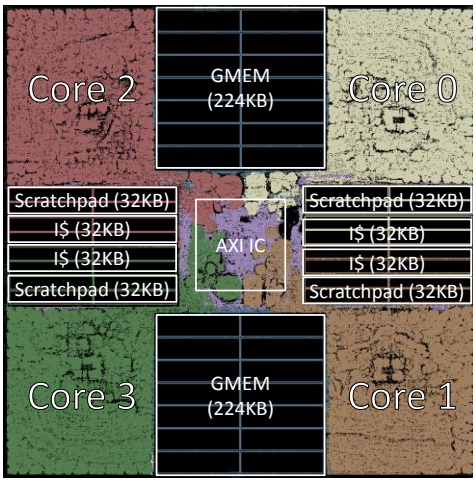


Figure 22: Quad-core ASIC layout

Objective-3 – Composability and Cross-Domain Applicability

From algorithm design and final software implementation perspective it is beneficial that we have single source of algorithm code which executed on all platforms. Our implementation with OpenCL as the selected implementation language uses the TCE and pocl frameworks to achieve the portability of very generic algorithm code to application specific processing architecture. Using application specific customization to accelerate critical parts of the processing we can same time satisfy the power consumption requirements and additionally still have the original algorithm design source code very close to the original. Cross-Domain applicability of the frameworks we have demonstrated with implementing use-cases from several application domains and processing requirements from two different ends of the spectrum.

Objective-4 – Robustness to Variability

Targeting a programmable, yet customizable, processor allows user do customization the processor architecture design and later software changes to deployed products already in the market. Design is also robust to run time variability as in run time based on service level or based on environmental conditions more suitable algorithm can be selected to guarantee user satisfaction. As an example, MMSE algorithm would be used when high throughput is required close to serving base station, while much more complex LORD algorithm would be used in cell edge with lower data throughput but guaranteeing signal reception.

4.2.4 Research questions

We had defined three main research questions:

1. How close to the power-performance of fixed function hardware accelerators is it possible to get with bare bone exposed data path parallel processors while retaining high-level programming language re-programmability for resource reuse, algorithm updates, and on- the-field bug fixing.
2. Is it possible to reduce required the verification effort required for a typical completely new customized parallel processor hardware implementation under 3 man months using automated testing?
3. The performance of a programmable accelerator is highly dependent on the quality of the software compiler and the compiler is restricted by the expressiveness of the input language; how much can a parallel programming language help in reaching performance portability across varying style of programmable accelerators in contrast to the traditional sequential language such as C

Our observations:

1. The actual receiver implementation details are often manufacturer specific and unpublished. However, in ESSIRC, 2010 Studer et al published results on approximately 770 mW power consumption with energy efficiency of 1016 pJ/bit using a manufacturing process which is at least 4x less power efficient than the process we were using. But these numbers show that the proposed programmable solution can reach the performance class of pure fixed-function hardware-based solutions.

2. Setting up a proper continuous integration both for the SW part of the design as well as for the HW part of the design is extremely crucial for the project schedule. Only automated continuous integration of SW and HW can catch both the SW errors as well as slippages in performance targets (such as processing budgets). Using TCE tools it is very rapid to get the first design iterations done and measured, but as always, the devil is in the details. Targeting FPGAs with the synthesized processor is very rapid and continuous integration with HW in the loop makes design iterations very rapid and easily verifiable. Depending on the design complexity the parallel processor can be designed well within 3 months. The FPGA design in our case was however only the architectural starting point. The IC design process, routing, synthesis, verification, can easily take another 3-6 months.
3. Having a portable parallel language description of the algorithm makes or breaks the performance portability, and even then, in some cases, lack or availability of target specific extensions can have huge impact on the performance. OpenCL is very good starting point as the performance portable source code.

4.2.5 Validation of functionality

The approach TCE takes to verifying the designs is shown in Figure 23. It uses a layered top-down approach where each level in the implementation abstraction hierarchy is compared against the previous one. At the first level, the designer, who uses a portable high-level language program as an application description, can implement and verify the software functionality using a desktop environment and the work station's CPU. Printouts to standard output can be used to produce the initial "golden reference" data.

Each layer of implementation abstraction can be compared to the golden reference data by including a standard output instruction in the processor architecture, used to produce the output at the different levels. In RTL simulation, the function unit implementation uses the VHDL or Verilog file printout APIs, and at the FPGA prototyping stage it may write to a JTAG-UART console. Further verification can be done by comparing bus traces which contain values in each processor transport bus at each instruction cycle.

Finally, the core is integrated to a system level model. TCE provides facilities to help producing project files and other integration files to different FPGA flows. System level simulation is supported via bindings for SystemC. The bindings allow plugging TTA instruction accurate core simulation models to larger SystemC models to enable cycle accurate performance modelling and functional verification simulations.

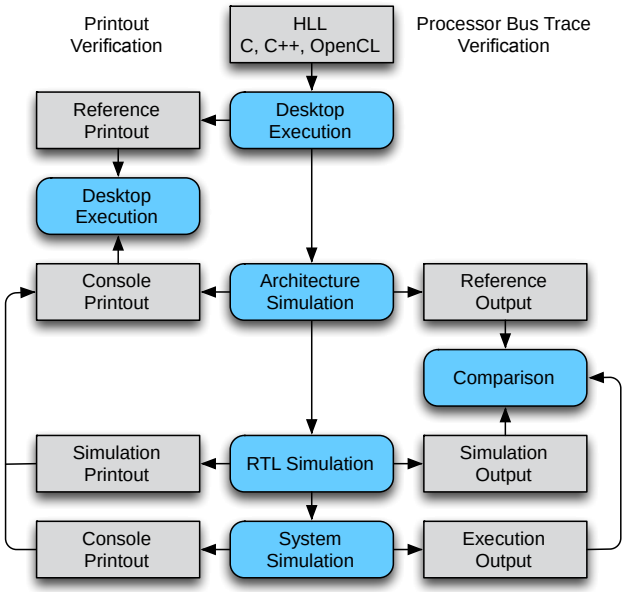


Figure 23: Architecture verification flow

4.2.6 Verification of robustness and reliability

In the mobile demonstrator, both for the development of the TCE tools, pocl OpenCL implementation as well as development of the demonstration use cases, test-driven development and automated testing methods are leveraged. As the reliability test cases are typically long, they are executed as a part of nightly build and results are reported to the developers.

4.2.7 Validation of performance metrics

In the mobile demonstrator, both for the development of the TCE tools, pocl OpenCL implementation and the development of the demonstration use cases, a test-driven development and automated testing and verification methods are leveraged. We have defined a large set of test cases and each commit to the version control triggers execution of automated test suites to verify both the functionality and the performance. The execution time and power usage are compared to the minimum allowed values and reported to the developers.

For the LTE receiver test case a pre-defined test vector for the LTE MIMO demodulation are passed to the TTA device for calculations. The kernel is launched using OpenCL API which contains primitive profiling features in the standard API. The kernel computation time is measured using the profiling features.

For the Audio signal processing test case, there is a known soft real time restriction that is derived from the sampling rate of the processed audio signal. Thus, it is important that the implemented test case can operate in real-time; otherwise the quality of the output will suffer dramatically. The goals were verified using the FPGA prototype, where the possible quality degradation will be audible as clicks and other additional noises.

In addition, small-scale test chips will were manufactured to increase the believability of the TCE design flow for larger processor designs and to get true measurements leveraging ultra-low power design methods. The test chips were not useful for the real application, but were tailored for power measurement and verification purposes.

4.2.8 Lessons learned

As always, the devil is in the details. It is very rapid to do the initial designs for programmable co-processors; however, it takes quite a long time to mature the designs and go through the design iterations. The “final” iteration thus the layout and synthesis of IC in our case still revealed architectural bottlenecks and even bugs, which were unobserved in the FPGA design. One must reserve at least a same amount of time to IC design as for architecture and functional verification on FPGA.

4.2.9 Conclusion

We introduced TCE and pocl frameworks to the ALMARVI design community and have successfully demonstrated the benefits of application specific parallelization.

We have demonstrated the benefits and capabilities of targeting all heterogeneous acceleration fabrics: CPUs, DSPs, GPUs and FPGAs from single source code. During the project, we proposed and implemented baseline of using different computation frameworks such as TTA and rVex under common acceleration API through pocl.

Application specific processors provide a middle ground between fixed function accelerators and generic programmable cores. They bring benefits of hardware tailoring to programmable designs, while adding new advantages such as reduced implementation verification effort. The hardware of customized processor is optimized for executing a predefined set of applications, while allowing the very same design being used to run other, close enough routines by switching the executed software in the instruction memory. The degree of processor hardware tailoring is dictated by the use case and the targeted product.

We have successfully demonstrated the application of the ALMARVI toolset to implement application specific cores to different ends of computation complexity spectrum. From one end we are talking about low latency but extreme throughput image and radio communications and from the other very minimal power budget audio computations. The toolsets used in this demonstrator can be applied across domains.

Quality trade off in selection of the algorithm in runtime has been addressed by using different algorithms to the same purpose, such as MMSE and LORD in the case of LTE. In similar fashion in image processing demonstrator the noise level can be predicted based on ISO of a captured frame and it is a design time job to find optimal

parameters for different noise levels. The de-nosing strength is then the parameter that can be adjusted in run-time based on the noise level and user preferences.

To tackle system performance and guaranteed power consumption using a component library based processor design with automated RTL generation was helpful in reducing implementation and verification effort.

We used a layered top-down approach where each level in the implementation abstraction hierarchy is compared against the previous one. Each layer of implementation abstraction was compared to the golden reference data by including a standard output instruction in the processor architecture, used to produce the output at the different levels. In RTL simulation, the function unit implementation used the VHDL or Verilog file printout APIs, and at the FPGA prototyping stage it wrote to a JTAG-UART console. Finally, verification can include clock cycle count measurement stage as well as power measurement stage from the final target platform.

From the performance perspective, we can state that we have met the original requirements we set for the use cases. In addition, we have verified that we can use open source frameworks developed in ALMARVI to produce high quality designs.

4.3 VISIDON: Image and video Enhancement

4.3.1 Introduction

This demonstrator focuses on implementation of image enhancement algorithms for standard of the-shelves mobile platform. The main purpose of the demonstrator is to find performance and power consumption differences between various heterogeneous processing units available on the platform, namely ARM CPU, ARM NEON, and GPU. The non-local means image de-noising software implemented and optimized for different processing units is used for experimenting processing speed and power consumption to asses ALMARVI objectives 1 and 2. In addition design and software portability related to ALMARVI objective 3 is considered and robustness to variability (ALMARVI objective 4) is discussed.

4.3.2 Requirements

ALMARVI deliverable D1.1 (Use Cases, Requirements and System Specifications) defines the use-case for this demonstrator and it also defines the functional and non-functional requirements for this demonstrator. These are summarized as below.

Functional requirements:

Functional requirement 1 – Reduce luminance noise level of input image frame

This requirement was achieved by implementing the Non-local means image de-noising algorithm and applying it to Y-channel of the YUV input image. This reduces the luminance noise. Requirement was verified by processing multiple noisy test images with the algorithm and compared the resulting images to the inputs.

Description: Reduce luminance noise level of input image frame.

Priority: Must have.

Type: SW

Verification procedure: Simulation / field test

Functional requirement 2 - Reduce chrominance noise level of input image frame

This requirement was achieved by implementing the Non-local means image de-noising algorithm and applying it to UV-channel of the YUV input image. This reduces the chrominance noise. Requirement was verified by processing multiple noisy test images with the algorithm and compared the resulting images to the inputs. Alternatively average filter was considered since chrominance channel details are not as important as luminance channel, and simple averaging is fast to perform and reduces noise from chrominance channel.

Description: Reduce chrominance noise level of input image frame.

Priority: Must have.

Type: SW

Verification procedure: Simulation / field test

Functional requirement 3 -Enhance edges of input image frame

This requirement was partly achieved. Non-local means algorithm does not apply edge enhancement and separate sharpening or another edge enhancement algorithm should be used. For this reason, another algorithm designed for image super-resolution was used. This algorithm enhances the edges and high frequency details of the images. Requirement was verified by testing algorithm with multiple images captured with a mobile phone

Description: Enhance edges of input image frame.

Priority: Nice to have.

Type: SW

Verification procedure: Simulation / field test

Functional requirement 4 - Avoid blur and detail loss of input image

This requirement was partly achieved. Non-local means de-noising is effective against noise and keep image details reasonably well. Based on algorithm parameters, user can adjust how much noise is reduced and how well original image details can be preserved. Requirement was verified by processing multiple test images with the algorithm and compared the resulting images to the inputs.

Description: Avoid blur and detail loss of input image.

Priority: Nice to have.

Type: SW

Verification procedure: Simulation / field test

Functional requirement 5 - Increase resolution of the full frame or in regions of interest

This requirement was achieved by applying super-resolution algorithm for input images. This algorithm enhances the effective resolution of the original image and can be applied either for the whole image or region of interest. Requirement was verified by testing algorithm with multiple images captured with a mobile phone.

Description: Increase resolution of the full frame or in regions of interest.

Priority: Nice to have.

Type: SW

Verification procedure: Simulation / field test

Non-functional requirements

Non-functional requirement 1 - 1080p - 2060p image at 30 fps is processed in real time

This requirement was achieved by implementing non-local means de-noising algorithm for mobile GPU using OpenCV. Requirement was verified by measuring the processing speed on development platform.

Description: 1080p - 2060p image at 30 fps is processed in real time.

Priority: Must have.

Type: SW + HW

Verification procedure: Field test / demonstration

Non-functional requirement 2 - 13MP image is processed in under 700ms

This requirement was achieved by implementing non-local means de-noising algorithm for mobile GPU using OpenCV. Requirement was verified by measuring the processing speed on development platform.

Description: 13MP image is processed in under 700ms

Priority: Must have.

Type: SW + HW

Verification procedure: Field test / demonstration

Non-functional requirement 3 - Real-time processing (preview) consumes power less than 200mW with 1080p images and under 500mW with 2060p images

This requirement was not fully achieved. Power consumption of modern mobile SoC platforms is clearly higher than expected. Requirement was verified by measuring the power consumption on development platform.

Description: Real-time processing (preview) consumes power less than 200mW with 1080p images and under 500mW with 2060p images.

Priority: Nice to have.

Type: SW + HW

Verification procedure: Field test / demonstration

4.3.3 Objectives

Table 22 below summarizes the objectives set for this demonstrator. All 'must-have' objectives were achieved and detailed analysis of power consumption requirement ('nice to have') was made. The following sections discuss these as defined in the ALMARVI project objectives.

Table 22: Baseline (current) and desired performance values

Parameter	Baseline	Desired values
Image size	1280x720 YUV 8-bit 8MP (capture photo YUV 8-bit)	3840x2160 YUV 8-bit 16MP (capture photo YUV 8-bit)
Frame rate (preview)	15 frames/s	30 frames/s
Processing speed (capture)	2500 ms	700 ms
OpenCL support	No	Yes
Support for multiple mobile platforms	No	Yes
Of the shelf components	Yes	Yes
Power consumption	700mW @ 15fps	200mW @ 30fps

Objective-1 – Enabling Massive Data Rate Processing

The objective for this demonstrator was to enable processing a multi-megapixel (13-16MP) image in less than 700ms and enable 30fps processing with HD resolution video input. The objective was achieved by implementing the non-local means de-noising algorithm for mobile GPU. The performance was measured on the latest Qualcomm Snapdragon development board (Open-Q 820) and the results are summarized in the Table 23.

Table 23: Processing time measurements for Objective 1 with Open-Q 820 Snapdragon

Image Resolution	CPU 1 Core	CPU 4 Cores	CPU 1 Core NEON	CPU 4 Cores NEON	GPU	Remarks
2MP: 1920x1080	403ms	174ms	238ms	99ms	**11ms**	NLM parameters are set to fast mode, only luminance noise is filtered
8MP: 3840x2160	1615ms	690ms	986ms	402ms	**32ms**	NLM parameters are set to fast mode, only luminance noise is filtered
16MP: 5312x2988	4641ms	1953ms	2790ms	1170ms	**615ms**	NLM parameters are set to high quality mode and also chrominance channel is filtered

From the Table 23 it can be seen that only GPU implementation could achieve the objectives. Algorithm parameters were set to meet prior objectives on target platform. In preview mode (real-time requirement) faster but not as high-quality setup was used. The non-local mean parameters for real-time mode were (please check ALMARVI deliverable 'D2.9 – Final report on low power scalable image/video algorithms' for more details of different algorithm parameters).

Fast mode (real-time de-noising):

Patch size: 8x8 pixels

Search window: 9x9 pixels with 3 pixels step in each direction (totally 9 neighbors considered)

Sliding window step: 8 pixels

Hiqh quality mode:

Patch size: 8x8 pixels

Search window: 11x11 pixels with 1 pixels step in each direction (totally 121 neighbors considered)

Sliding window step: 7 pixels

Objective-2 – Achieving Low-Power Consumption

The objective was to achieve 200mW power consumption at 30 fps frame rate with FullHD (1920x1080) input frames. Power consumption was measured as total system power when performing algorithm on Qualcomm Snapdragon 820 development board. Also the baseline system power measured without algorithm running was measured in order to try to find out the actual power consumption of the algorithm. The Table 24 summarizes the measured power consumption on target platform using GPU version of the algorithm implementation. Figure 24 shows current [mA] measurement for 30fps FullHD (1920x1080) stream, and for comparison Figure 25 shows similar measurements for CPU+NEON version of the algorithm. It should be noted that only GPU implementation could achieve the target frame rate (30fps) as measured for Objective 1. The fastest CPU implementation could only achieve 10fps.

Table 24: Power consumption measurements for Objective 2 with Open-Q 820 Snapdragon @30FPS

Image Resolution	Current measurement [mA]	Power Consumption [mW]	Remarks
1920x1080	91mA	1092mW	NLM parameters are set to fast mode, only luminance noise is filtered
3840x2160	140mA	1680mW	NLM parameters are set to fast mode, only luminance noise is filtered

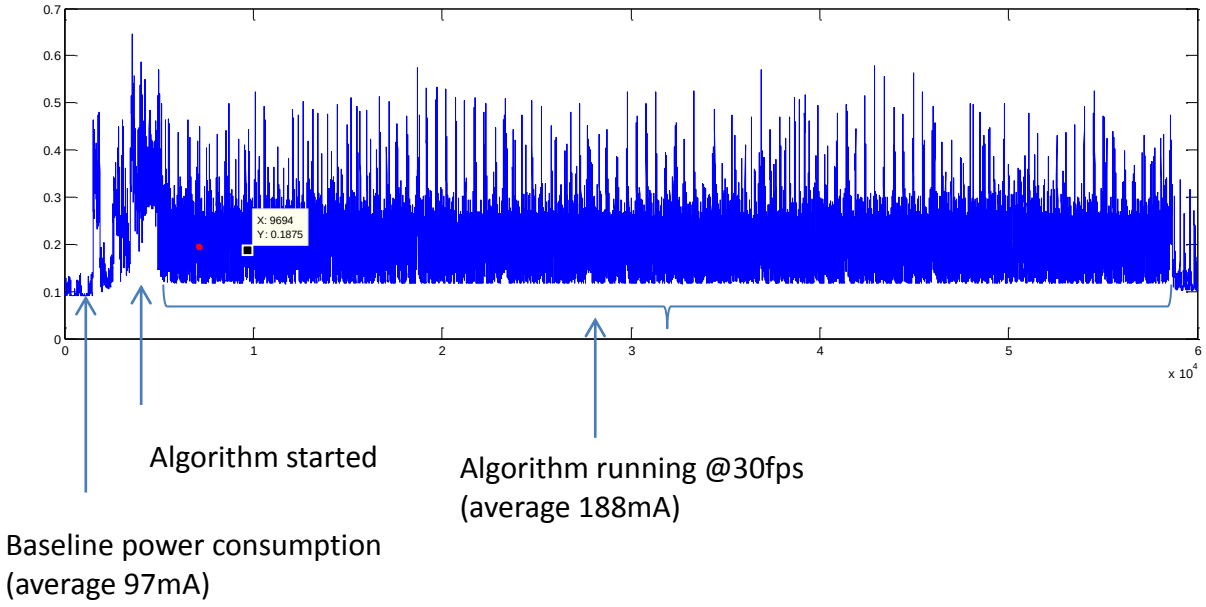


Figure 24: Power consumption measurement (mA) when running noise reduction for FullHD (1920x1080) size frames at 30 frames per second with GPU (OpenCL) implementation.

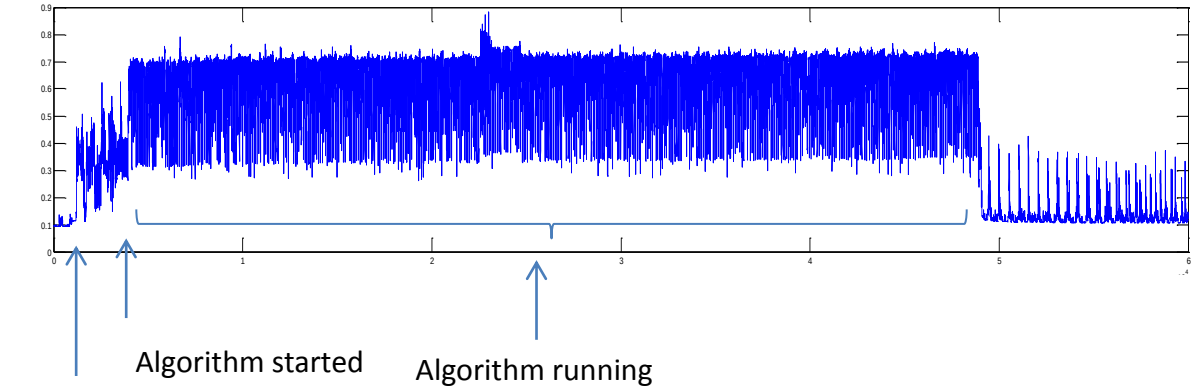


Figure 25: Power consumption measurement (mA) when running noise reduction for FullHD (1920x1080) size frames at 10 frames per second with CPU (4 thread + ARM NEON) implementation.

The target objective was 200mW with 30 frames per second. The development board uses 12V power supply, and with easy POWER = VOLTAGE x CURRENT computation it can be calculated that the power consumption is 12V x 0.091A = 1.092W = 1092mW. This is clearly higher than expected. For comparison, the fastest CPU implementation using only 10FPS consumes 12V x 0.433A = 5.196W = 5196mW.

Because target objective is much less than the minimum achieved power consumption of 1092mW, we wanted to experiment what would be the minimum power consumption of target hardware when GPU is activated with the OpenCL framework. For this experiment, we only load FullHD size images from CPU to GPU and read it back from GPU to CPU at 30 frames per second rate. Figure 26 shows the result of this experiment. It can be calculated that the power consumption of such process is 12V x 0.045A = 0.54W = 540mW. Based on this observation it is worthwhile to say that it is quite impossible to achieve as low as 200mW power consumption with Qualcomm Snapdragon 820 when GPU is used to process FullHD size images at 30 frames per second.

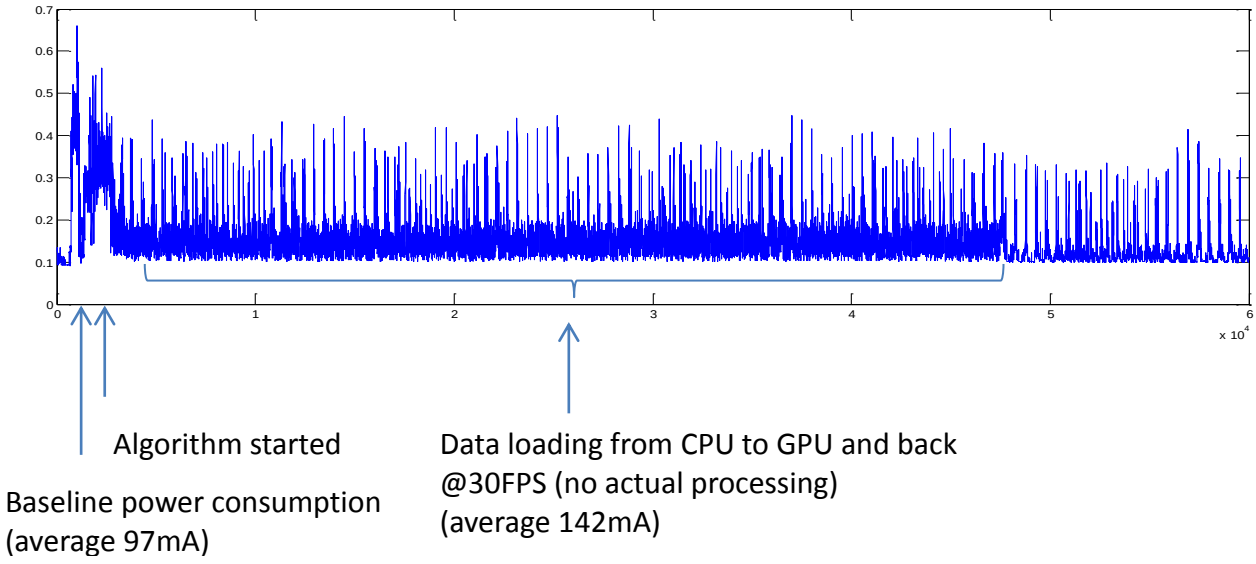


Figure 26: Power consumption measurement (mA) when loading FullHD (1920x1080) size frames at 30 frames per second from CPU to GPU and reading back. No any actual processing for input is made, but only memory loading.

In addition to GPU framework power consumption testing we wanted to verify what would be power consumption when CPU is only used and very simple buffer copying (memcpy) is done at 30 fps rate. Figure 27 shows the result of this experiment. It can be calculated that the power consumption of such process is 12V x 0.039A = 0.468W = 468mW. It is therefore clear that all memory intensive computation, such as processing large images at video frame rate would consume power relatively much compared to target objective (200mW), and this objective is not possible to achieve with the current hardware.

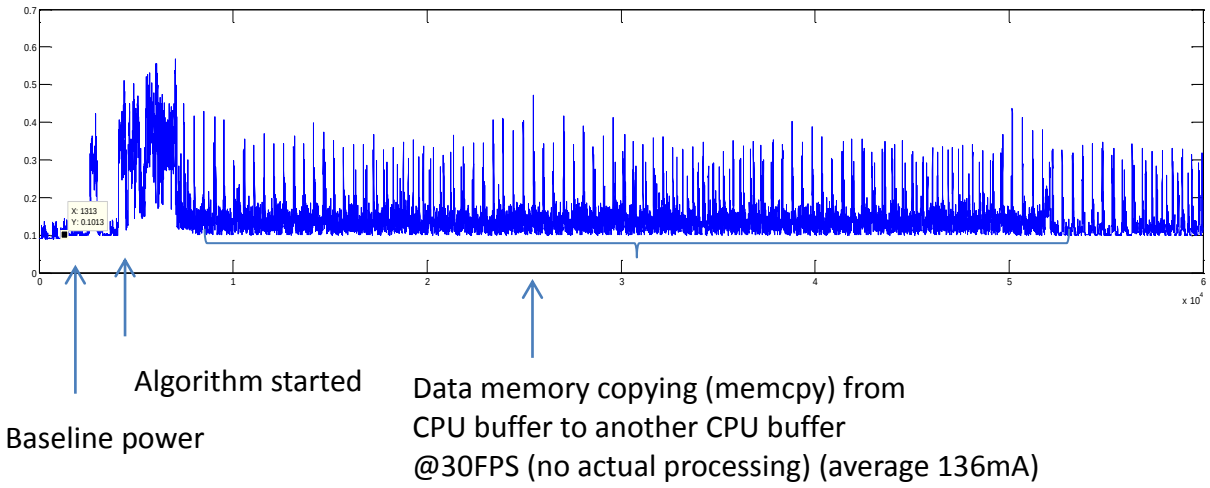


Figure 27: Power consumption measurement (mA) when copying FullHD (1920x1080) size frames at 30 frames per second from CPU buffer to another CPU buffer (memcpy). No any actual processing for input is made, but only memory loading.

Objective-3 – Composability and Cross-Domain Applicability

There are a number of different mobile platforms and the one selected in this demonstrator represents the high-end class device. From the software point of view it would be beneficial to have single software that is able to run on various different mobile platforms. To achieve this all the software is written in standard APIs and programming languages. In this kind of software product and use-case it is not required to fully avoid engineering when integrating solution to new platform. Thus it can be chosen which version of the implementation (standard C/C++, ARM NEON, or OpenCL) can be used with an another target platform, and what run-time parameters will lead to good enough performance. The API itself is the same for the de-noising function and the underlying implementation is selected based on target hardware.

During ALMARVI project there has been significant development in mobile platforms. Even though this mobile demonstrator has only considered Qualcomm Snapdragon platforms there have already been three different generations of the hardware of which have been used to evaluate and develop this demonstrator. The Table 25 summarizes the main differences between these platforms. To evaluate Objective 3 we have used the exact same code base and executed in on different hardware. Only minimal changes were required to code to enable to run this implementation on all these platforms so it can be considered to very portable. Also, the performance gain achieved with latest GPU accelerators was directly recognized and the same implementation could run twice as fast on Adreno 530 GPU compared to Adreno 330 GPU.

Table 25: The main difference between Snapdragon 800, 810 and 820 series development boards used in this demonstrator.

	Snapdragon 800	Snapdragon 810	Snapdragon 820
CPU	Quad-core Krait 400@2.3GHz	Octa-core (4x ARM Cortex A57 + 4x ARM Cortex A53) @2.0GHz	Quad-core Kyro@2.2GHz
CPU Bit Architecture	32-bit	64-bit	64-bit
GPU, and API support	Adreno 330, OpenGL ES 3.0, OpenCL 1.1 Embedded Profile	Adreno 430, OpenGL ES 3.1, OpenCL 1.2 Full Profile	Adreno 530, OpenGL ES 3.2, OpenCL 2.0 Full Profile
Memory Type and Speed	Dual-channel LPDDR3, 800MHz	Dual-channel LPDDR4, 1600MHz	Dual-channel LPDDR4, 1866MHz
Process Technology	28nm	20nm	14nm

Objective-4 – Robustness to Variability

It is also desired that algorithm performance is predictable and robust enough. Processing time of our NLMS is linearly dependent of the used image size and the algorithm parameters. Thus it can be well predicted during the design time. On the other hand, mobile handsets are complex systems that are simultaneously running multiple services which may affect available processing resources. This cannot be predicted during the design time and the algorithm must be able to tolerate availability of changing shared resources. In addition, mobile battery based devices may contain specific power manager solutions that limit resources when the battery is running low. This is also something that cannot be predicted but needs to be tolerated. In practice, one could adjust run-time parameters based on the processing latency to adapt on varying computing resources to limit computation per frame. In addition to changes of available computing resources, the algorithm should be able to handle different kind of noise levels in input images. In practice, the noise level can be predicted based on ISO of a captured frame and it is a design time job to find optimal parameters for different noise levels. The de-nosing strength is the parameter that can be adjusted based on the noise level and user preferences.

General Objectives

General objectives of this demonstrator were to study and show how heterogeneous mobile platforms can perform typical image processing tasks and how efficient underlying hardware components are in terms of processing speed and power consumption. The non-local means de-noising algorithm was selected to this demonstrator because it contains many fundamental steps that are common to many other image processing tasks, such as block matching and weighting. The goal was to understand better what kinds of implementations can be found efficiently for these kinds of image processing task.

In mobile imaging, typical requirements are processing speed and power consumption (Objectives 1 and 2). These were systematically measured with target platforms. Also differences between different implementations (mobile GPU, multi-core CPU, ARM NEON ASM) were systematically analysed during the project, and based on the target objectives set in the early phase of the project, it was found that efficient mobile GPU implementation can achieve these objectives in terms of processing speed requirements. The power consumption was found to be more than was desired with the target hardware.

In addition to measurable objectives, such as speed and power consumption, this demonstrator considered solution portability, product lifecycle and maintenance. Mobile platform development is very fast, and during the ALMARVI project, there were several generations of the hardware available. It was found that the software implemented with the standard APIs is very straightforward to run on different platforms. This is a good signal when developing imaging solutions for fragmented hardware base as it is in the mobile domain. On the other hand, only Qualcomm platforms were considered in this demonstrator, and in reality one should pay attention to evaluate with other mobile platforms as well (such as Mediatek).

4.3.4 Research questions

As stated in the deliverable D1.1 one of the main research question was to understand the current status of OpenCL tools, support, and API compatibilities in common of-the-selves mobile platforms. In the early phase of the project, OpenCL was in the beginning to become standard tool for utilizing heterogeneous hardware for general computation on mobile platforms. During the project, OpenCL support has increased from the Embedded 1.1 Profile to the full 2.0 profile, and basically offers very flexible tools for general purpose computation and is useful for various image processing tasks as well.

Another research question was to find out how to efficiently use the mobile hardware in terms of power consumption. In practice the goal was to understand how different computing units that are available on mobile platforms can perform on similar image processing tasks. The same algorithm was implemented for different computing units (CPU, GPU, SIMD CPU) and it was measured how these differs in processing speed and power consumption. It was find out that mobile GPU was the most powerful in terms of power consumption and processing performance with the selected image processing algorithm.

4.3.5 Validation of functionality

Functionality of the demonstrator was validated with real-world image data with simulation tests and with a separate Android application which was developed for the target platform. Requirements set by deliverable D1.1 were checked one by one, and real hardware was used to run algorithm for image data. Quality requirements were visually assessed and measurable characteristics (power consumption and speed) were systematically measured.

4.3.6 Verification of robustness and reliability

Algorithm quality and performance was analysed with large set of input images in order to verify how it works with different kinds of input images. Demonstrator was also run on different mobile platforms for verifying its performance on varying hardware.

4.3.7 Validation of performance metrics

Performance metrics were validated with the target hardware using the standard processing time and power consumption measurement mechanism. It should be noted that several measurement were made and average results were reported since it is was found that single executions can slightly vary when executed on mobile platform.

4.3.8 Lessons learned

The most important findings that were made with this demonstrator were:

- Mobile GPU is more efficient in terms of power consumption for certain kind of image processing task compared to multi-core mobile CPU.
- OpenCL API is useful for mobile imaging and can be supported by the modern mobile platforms.
- ARM NEON SIMD is relatively efficient in terms of processing speed, but when used in multi-core scenario it can consume more power compared to mobile GPU.
- Power consumption of the COTS SoC platform (Snapdragon 820 in this demonstrator) is higher than expected when memory intensive algorithms are run (access to image pixels).
- Standard APIs (ANSI C/C++, OpenCL, ARM NEON) are very useful when implementing software solutions for different mobile platforms. The same implementation can be run on different generation of the devices.

4.3.9 Conclusion

This demonstrator considered mobile image enhancement use-case and it was studied how to efficiently use heterogeneous mobile platform for image de-noising. The demonstrator was implemented on Qualcomm Snapdragon platform running Android OS. The performance of the demonstrator was analysed based on the objectives set in the early phase of the project. Based on quantitative and qualitative measurements it was found that the objectives were achieved as summarized in Table 26.

Table 26: Summary of the results achieved in this demonstrator.

Parameter	Baseline	Desired values	Requirement	ALMARVI result
Image size	1280x720 YUV 8-bit 8MP	3840x2160 YUV 8-bit 16MP	Must have	100% achieved (supports large images)
Frame rate (preview)	15 frames/s	30 frames/s	Must have	100% achieved. GPU implementation could achieve @30 FPS when using 'fast' run-time processing parameters
Processing speed (capture)	2500 ms	700 ms	Must have	GPU implementation could achieve objective when using 'high quality' run-time processing parameters
OpenCL support	No	Yes	Must have	100% achieved (platform supports OpenCL and implementation was verified)
Support for multiple mobile platforms	No	Yes	Nice to have	100% achieved (standard API support multiple platforms, and implementation was verified with different devices)
Of the shelf components	Yes	Yes	Must have	100% achieved (implementation can be run on standard COTS mobile devices)
Power consumption	700mW @ 15fps	200mW @ 30fps	Nice to have	50% achieved. Based on extensive analysis, 200mW requirement is not possible with the current off-the-selves mobile hardware. However, mobile GPU can achieve 3,75x reduction (with 3x speed-up) compared to CPU and power consumption reduction of GPU implementation is over 14 times compared to CPU version.

5. Evaluation

This section evaluates the results of the various demonstrators and shows how they collectively address the four objectives of the ALMARVI project. Table 27 shows the collective coverage of the ALMARVI objectives in all three application domains by all demonstrators in the project. The table shows that all project objectives and sub-objectives are addressed by at least one demonstrator in the project. Most notably, the "low power" sub-objective of "Algorithm resilience for power efficiency" is covered by only one demonstrator: Large Area Surveillance (Aselsan). This is done by investigating the power consumption of various execution platforms as the number of iterations for the surveillance algorithm is modified to achieve the desired functionality with a given power budget. This also indicates that resilience for power efficiency is a difficult objective to achieve in various applications.

Another notable objective to mention in the "cross domain" sub-objective of "Software portability", which is covered by all the demonstrators in the project. This is not surprising since this represents one of the cornerstone targets for the ALMARVI project. All partners investigated the potential of porting their software to various HW platforms to ensure the higher performance, lower power consumption or higher cost efficiency of the platforms, Partners experimented with the potential of portable languages, such OpenCL, to help them ensure easy portability between the platforms.

Table 28: Objectives coverage for all ALMARVI demonstrators

	Coverage of Almarvi Objectives	Healthcare		Security / Surv. & Monitoring					Mobile	
		Interventional X-Ray (PHILIPS)	Breast Cancer Diagnostics (UEF)	Large Area Surveillance (Aselsan)	Road Traffic Surveillance (CAMEA)	Smart Surveillance (HURJA)	Analysis of multimodal camera data (VTT)	Walnut Tree Harvest Protection (UTIA)	Segmentation and LTE receiver (NOKIA)	Image & video enhance. (VISIDON)
1	Acceleration fabrics	+	+	+	+		+	+	+	+
	Design tools	+		+			+	+	+	
	Application specific parallelization	+	+	+	+		+	+	+	+
	Quality - performance trade-off								+	+
2	Low power cores		+	+	+			+	+	
	SW quality - power trade-off		+	+						
	HW quality - power trade-off								+	+
	Algorithm resilience for power efficiency			+						
3	Software portability	+	+	+	+	+	+	+	+	+
	Interfaces for heterogeneous acceleration fabrics	+		+	+	+		+	+	+
4	Guarantee system performance	+		+	+			+	+	+
	Guarantee power consumption			+	+			+	+	+

In terms of partners, the table shows that smaller industrial partners (such as HURJA) cover the least number of objectives in the project. This is understandable due to the limited scope of their demonstrators. The bigger partners, on the other hand, like NOKIA and Aselsan cover the largest number of objectives in the table, due to the elaborate investigations they perform on their demonstrators. One exception is Philips, which covers most objectives, except those related to low power. This can be explained by the limited need for low power for the Philips use case.

6. Lessons learned

In this chapter we will address the general lessons learned related to developing the ALMARVI Demonstrators. The combined lessons learned from the ALMARVI partners can be generalized to four different categories: Tools, Portability, Power and Integration.

6.1 Tools

Many partners have used FPGA platforms for the implementation of their ALMARVI demonstrators. Both Xilinx and Intel (Altera) FPGAs were used. We have used different tool flows, but we all have in common that we want to abstract from the low level FPGA implementation. In order to enable flexibility and portability, to make the design more testable and to reduce development time in general. Simulink HDL coder, Xilinx Vivado HLS, Xilinx SDSoC with OpenCV libraries, Xilinx EdkDSP and Topic Embedded Systems Dyplo were evaluated and applied in several demonstrators. OZYEGIN developed the MAFURES HLS tool which was used for the ASELSAN demonstrator. The TTA-based Co-Design Environment addresses the development of customized processors (FPGA or ASIC) in general. Several other tools have been used and developed to aid the design process. Elaborate experiences with several of the tools are given below.

6.1.1 Vivado High Level Synthesis tools

Vivado HLS supports coding in C, C++ and SystemC and facilitates the implementation of this software description into hardware. Outputs of the tool can be VHDL, Verilog and a Packaged IP ready to be implemented in a hardware design. Vivado HLS supports a C/C++ test bench which is used both for C simulation and RTL simulation (co-simulation), thus allowing developers to use the same test environment for the development of the algorithm on the generated hardware and as such saving a lot of time and warranting correctness of the implementation through the flow. Another advantage is that at least the initial development can be done by software developers. Figure 28 shows the Vivado HLS Design Flow.

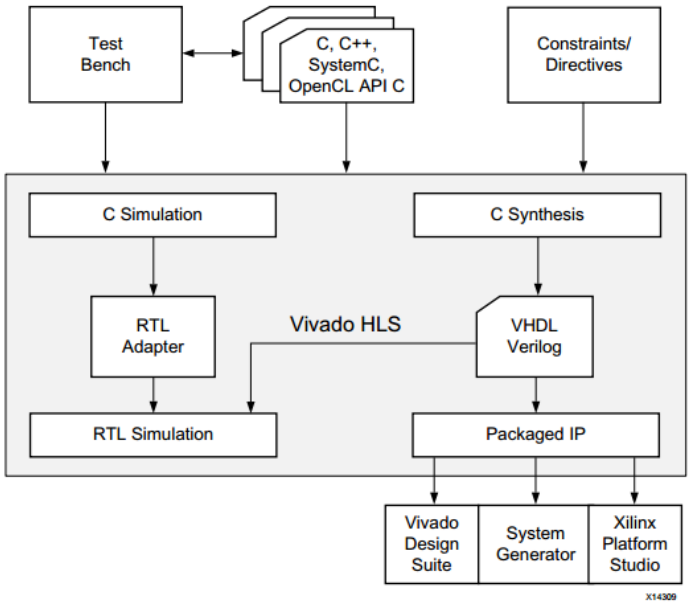


Figure 28: Vivado HLS Design Flow from Xilinx

Implementation Experience

- Floating point to fixed point

The original implementation of the algorithm was in floating point, the floating point variables were manually converted to fixed point variables while maintaining sufficient precision in the computations. Vivado HLS has C++ libraries to define bit precise variables. All variables were defined using these bit precise types.

- Memory allocation and pointers

Due to the nature of the hardware it is not possible to use dynamic memory allocation, and pointers can only be used when the size of a memory reference is known at compile time.

- Fixed size arrays

Also because the hardware cannot be changed dynamically, all arrays must be to be configured to hold the maximum required image size.

- Memory architecture

A first look at for example a bi-linear interpolation algorithm shows that the bottleneck of the algorithm is in the reading of 4 input pixels for each computation, assuming that a computation can be performed in one single clock cycle. These read operations will have to be done in parallel in order to be able to pipeline the algorithm. This optimization was obtained easily by applying 2D array partitioning pragma's in Vivado HLS. The corresponding memories were created manually using the Vivado toolflow outside this HLS module.

- Parallelization

Vivado HLS has pipeline pragmas which we used to obtain the optimum pipelining. Even though the memory was optimized to read 4 input pixels in a single cycle it was still not possible to fully pipeline the interpolation due to handling boundary conditions on the edges. We found that handling the boundary conditions after reading the memory resulted in a better pipelined design and optimal performance. A pre-check on the boundary conditions causes a branch in the program flow which cannot be pipelined.

- Line buffers

Our original algorithm was implemented as frame based processing with 'random' access to images. In the FPGA we need to implement stream based processing. This conversion needs to be performed manually and requires much hardware knowledge. Line buffers were implemented with Vivado HLS streams to buffer history data, for example for a convolution operation.

- Interface

Vivado HLS allows to manually define interfaces in SystemC, however we preferred to use the standard available interfaces. The standard control interface includes, start, ready, done and idle signals, the done output of the module becomes active only when the full block has finished. Start should be activated when a new input is available. This control interface was connected to our image processing pipeline interface. Although the interface is designed to support continuous streaming images (image overlap in the processing part), we use the interface as required in the imaging pipeline to handle single image processing only. Note that the imaging pipeline is continuous streaming but the processing stage in the pipeline are single image only.

Memory interfaces were implemented as single ported ROM for the Vivado HLS block, but in fact are dual ported RAM such that we can write and read the memory through the AXI interface with the processor.

The Video in- and output are implemented as streams. In C++ it is possible to use a stream template class, which is modeled as an infinite depth FIFO; streams provide a safe way to work with streaming data applications.

- Control logic

Next to the Vivado HLS interface control logic we wanted to implement control logic to be able to process (and debug) intermediate output results. Control logic is implemented to handle the end of image state. The implementation supported by Vivado HLS was very unreliable. On the other hand, modelling user control logic in C has negative effects on the performance. Expanding the video data bus with 2 extra bits gave the required result without any performance degradation. The so called start-stop bits are split off in the image processing stage before calculations start and added after the calculations and measurement visualization are done, this way we could add extra control logic to the interface.

Conclusion

This tool offers a fast way to build a demonstrator on Xilinx FPGAs. The use of the C/C++ test bench for simulation results in quick development cycles and can easily be extended to a software tool flow with for example Microsoft Visual Studio. We have used many different versions of this tool from early 2014 up until 2017. In 2014 the tool

was still a bit buggy, but this has greatly improved, up to the point that it now has become a mature reliable technique. Vivado HLS still has its limitations; we see good applicability for image processing algorithms and single or multi-dimensional algorithms in general, however control logic seems more difficult to implement. Next to that a deep understanding of FPGAs is still mandatory to truly optimize the implementation, which was necessary to meet our requirements, especially for streaming data applications.

Being able to use HLS tooling with C++ input already greatly reduced the development effort for the FPGA platform. We estimate a 4x speedup in development time for our kind of implementation, Xilinx reports up to a 15x speedup for less complex implementations. Next to that we could much more easily test the design, simulate the implementation and test the final implementation on the target all with the same test framework. We estimate that we gained more than a 4x speedup with HLS opposed to manual VHDL when making small algorithmic changes to the implementation.

It would be beneficial to abstract even further from the 'low level' optimizations through better pragma's and analysis by the tool or by supporting languages which are more suitable to describe parallelism.

6.1.2 OpenCV

Xilinx provides support for image and video processing using open source OpenCV library on Zynq Devices. The OpenCV support can be divided into three different levels which are summarized in the following paragraphs.

Software OpenCV support for Zynq ARM

Xilinx Vivado tool is providing possibility to implement software applications using cross compiled software OpenCV library. The OpenCV hello world example application is provided under Xilinx Vivado SDK and described in [OCV1]. The OpenCV library can be found as third party dynamic library precompiled for Zynq Linux target. The library allows to implement initial image/video processing based either on files read from file system and results written back as files or as a solution working with real image/video frames inside embedded platform without hardware acceleration.

OpenCV support in Vivado HLS

The Vivado HLS tool comes with its own implementation of OpenCV library functions. It is a subset of the OpenCV library which was simplified and rewritten to be synthesizable for FPGA. For that reason, the list of supported image formats and number of arguments may differ from original software OpenCV library. A detailed list of supported operations and their arguments can be found in Xilinx Wiki pages [OCV3]. The Xilinx application note [OCV2] provides guidelines which designer should follow to take original OpenCV applications and rewrite/redesign them for hardware accelerated ones using Vivado HLS.

OpenCV support in SDSoC

The Xilinx SDSoC tool provides possibility to implement algorithms in C/C++ and turn them into hardware accelerators automatically. The tool provides also solution for data communication and synchronization and thus it is more convenient for rapid development of hardware accelerated image/video processing algorithms in FPGAs.

The SDSoC tool internally uses Vivado HLS for implementation of hardware accelerator. It puts its own limitations on data interfaces in order to be able to integrate automatically the core into whole system. Thus exploiting the OpenCV library available in Vivado HLS is possible as well under the assumption that SDSoC interface limitations are followed. Few examples are available at Xilinx wiki [OCV4]. We found that they are from original Vivado HLS application note and their documentation is not finished yet as well as the example package is still under heavy development.

The SDSoC tool provides also debug capabilities based on the fact, that the accelerator source code is provided in C/C++ and thus it can be compiled to ARM core of Zynq. Vivado HLS OpenCV library functions can be debugged only if their body is fully implemented in C/C++ and is not using internal components from libraries. Thus debugging OpenCV application in SDSoC cannot be always guaranteed. For that reason, the rewriting simple operations which cannot be debugged in HLS OpenCV library must be always considered in order to speed up debug phases of the project.

The example OpenCV function used in SDSoC environment is given in

Figure ***29***. The function is implemented in two versions, one for the software execution on ARM (PS), the second for compilation to HW accelerator (PL) using Vivado HLS. It may be possible to simulate the second part in software as well. This possibility depends on how the used functions from the OpenCV HLS library are written.

```
#ifndef __SDSVHLS__
#include "image_cores.h"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
using namespace cv;

void image_filters(     int hdmi_in[1080*1920], int hdmi_out[1080*1920])
{

  const char coef[3][3] =  {{-1,-2,-1},
                            { 0, 0, 0},
                            { 1, 2, 1}};

  Mat src(1080, 1920, CV_8UC4, (int *)hdmi_in);
  Mat dst(1080, 1920, CV_8UC4, (int *)hdmi_out);
  Mat kernel(3, 3, CV_8SC1, (char *)coef);

  Point anchor = Point( -1, -1 );
  double delta = 0;
  int ddepth = -1;

  filter2D(src, dst, ddepth , kernel, anchor, delta, BORDER_DEFAULT );
}
#else /* is __SDSVHLS__ defined */
#include "image_cores.h"
#include <hls_video.h>

void image_filters(     int hdmi_in[1080*1920], int hdmi_out[1080*1920])
{
#pragma HLS DATAFLOW

  const char coef[3][3] =  {{-1,-2,-1},
                            { 0, 0, 0},
                            { 1, 2, 1}};

  hls::Mat<1080,1920,HLS_8UC3> src;
  hls::Mat<1080,1920,HLS_8UC3> dst;
  hls::Window<3,3,char> kernel;
  hls::AXIM2Mat<1920,int, 1080, 1920, HLS_8UC3>(hdmi_in, src);

  for(int j=0; j<3; j++) {
    for(int i=0; i<3; i++) {
      kernel.val[j][i] = coef[j][i];
    }
  }

  hls::Point_<int> anchor = hls::Point_<int>( -1, -1 );
  double delta = 0;
  int ddepth = -1;

  hls::Filter2D(src, dst, kernel, anchor);
  hls::Mat2AXIM<1920,int, 1080, 1920, HLS_8UC3>(dst, hdmi_out);
}
#endif
```

Figure 29: Example OpenCV function in SDSoC. The original OpenCV for ARM and rewritten version for Vivado HLS are shown (compilation to hardware accelerator). The segment at the top is used for SDSoC software execution, it links to the dynamic OpenCV libraries cross compiled for Zync ARM Linux. The bottom segment is used for Vivado HLS which is compiling the hardware accelerator. It interfaces the HLS OpenCV Library. AXIM2Mat and Mat2AXIM functions are passing the data between SDSoC and OpenCV worlds.

Conclusions

For the image/video pre-processing stages it is possible to use stream processing OpenCV functions to generate hardware accelerators. Their implementation is compatible with OpenCV library and lead do direct acceleration of development process even if there are at this moment numerous problems with documentation and debugging.

6.1.3 TTA-based Co-Design Environment (TCE)

TCE is a mature toolset for designing and programming customized processors based on the Transport Triggered Architecture. However, as it is an academic tool with an open source community version that hasn't yet gone through a commercialization process, its end-user usability has not been polished to the maximum. This is due to academic priorities being in producing publications; good quality papers are hard to write about tool polishing engineering work.

This sometimes causes additional work when modifying the processor designs that could be reduced rather easily by focusing a little more on the end user usability aspects. Some of the found issues are as follows:

* The link between the implementation and architecture is a weak one. Whenever the architecture in the Processor Designer (ProDe) tool is changed, its implementation must be manually recreated in the Hardware Database Editor.

* When creating new affected-by relationships for operations in Operation Set Editor (OSEd), it takes a lot of time to find each pair of dependencies. It could be made easier somehow by allowing a bunch of operations depending on another bunch.

* The mapping from Proxim to the source code lines should be improved. Currently one has to inspect the disassembly window for the link to the source code. This could be easily improved to add proper source code debugging capabilities.

* ProDe has all the info needed to compute the instruction width. It should be displayed in the user interface as it's often a very interesting design aspect.

* HDBEditor requires listing the opcodes even though nowadays there is a mandatory alphabetical ordering from the OSAL operation names to the opcode numbers.

* The mapping between function unit ports in the architecture model and the implementation ports is unclear. This is because HDB doesn't include the port name in the database as part of the architecture; strictly put it isn't, but could be in case "architecture" definition includes also the assembly text where the port names are needed. Currently generic names such as p1 and p2 are used and it's sometimes hard to figure out which ports are meant.

* The implementation description files (IDF) tend to contain absolute paths to implementation files, which makes them "unportable" as the same absolute paths are not found in other computers where they might be opened for further editing.

The tools should favor relative paths whenever possible, or there should be another "package processor design" tool that prunes the paths and creates an archive of all the needed files.

Outside the few usability issues, the toolset is of high quality and even with the latest experimental new features there are rarely major show stopped bugs found.

6.2 Portability

We have seen different solutions in achieving portability in the ALMARVI project. Portability from one platform to another might be achieved by using open portable languages such as OpenCL, many partners have used this language successfully for a range of different platforms and implementations. The open source pocl was used, and further developed for ALMARVI, to support OpenCL for the ALMARVI platforms.

The step from OpenCL to FPGA platforms (through HLS) is still in development but both Xilinx and Intel are making progress in this direction. Alternative solutions to achieve portability towards FPGA platforms are for example High Level Synthesis, which can be used to translate OpenCL kernels (C/C++) to FPGA accelerators or more general a C or C++ algorithm to an FPGA accelerator. Or we can implement OpenCL programmable soft cores such as the rVex on an FPGA.

6.3 Power

When aiming for lower power consumption for our implementations we see that the biggest gain can be to port the application to (more) embedded lower-power platforms. However it is key to find the right platform for the right implementation. For example moving an almost fully sequential algorithm to a highly parallel multicore might not achieved the desired power reduction. Also for example fixed or floating point arithmetic should be taken into

account related to the platform. A floating point GPU implementation should not be compared to a floating point FPGA implementation, but to a fixed point FPGA implementation instead. Most of us identified memory access as the biggest power drain of the implementation. This can for example be addressed by moving from a frame based data architecture to a stream based data architecture to omit external memory accesses. In the following sections we describe several considerations regarding power measurements and results on power consumption.

6.3.1 Power measurements

The preferred approach to obtain power/energy consumption based platform efficiency conclusions is to measure

1. Instantaneous power consumption over complete execution time per frame
2. Precise total execution time per frame

or

1. Total consumed energy per frame
2. Precise total execution time per frame

for a large number of frames and average over the frames. Maximum and minimum power/energy consumptions per frame over a large number of frames are also very useful statics from both optimization and system design points of view.

For example, an instantaneous power consumption graph that extends over multiple frames provides great insight to the algorithm/code running on a specific processor. By comparing such graphs obtained for multiple processors, one can reach useful conclusions regarding the efficiency of varying implementations of an algorithm running on a given set of processors. And finally, by pushing optimizations on all platforms to commonly accepted/known limits, one can conclude that a specific processor is the best platform for a given algorithm in terms of (GFLOPS/Watt) or (GFLO/(sec * Watt)), which is quite similar to (processed pixels / Joule).

However, processors are not the only source of power/energy consumption. When considering complete systems, power efficiency should include

1. Processor power usage
2. RAM power usage
3. Auxiliary power usage (ports, fans, etc.)

For certain algorithms, the memory sub-system can contribute substantially to the overall power/energy usage in return for improved performance. For example, a high speed DRAM with a large interface is guaranteed to improve performance for an algorithm that requires heavy random memory accesses with light compute intensity per access. However, such a performance improvement will come at increased power/energy consumption. It is also important to keep in mind that cache architecture of the processor and specifics of the DRAM used in the system can affect overall power consumption.

Finally, before making power/energy consumption based optimality conclusions it is important to keep in mind that the specific algorithm implementation under consideration is also important. An implementation that does not efficiently utilize memory caching mechanism readily available on a given processor is guaranteed to consume more power and have a longer overall execution time per frame. Hence, all power/performance measurements should be conducted on optimized implementations.

As useful as it may look, sub-system power/energy measurement based approach has a major flaw. Precise power measurements of processor and memory sub-system in isolation are hard and complicated, especially for COTS systems which were not designed with such purposes in mind. Depending on the specifics of the system such measurements may not even be possible without special hardware support.

Based on all these observations, in the context of ALMARVI project we opted to go with overall system power measurements. When comparing performances of different implementations running on different processors or acceleration fabrics, we have two possible scenarios. If the co-processor or special purpose acceleration fabric is located in the same system as the main processor they are compared against, the overall power/energy consumption comparisons are straight forward, as the expected auxiliary power consumptions are expected to be the same or very close. Furthermore, increased auxiliary power consumption in the case of co-processor or acceleration fabric use is most likely due to the activation of this additional platform and as a result should be accounted in the power measurement.

If the co-processor or acceleration fabric is being compared against a processor residing on a different system (different COTS card, mobile or desktop computer) than the execution conditions and system properties are detailed as much as possible and any unexpected power/energy consumption observations are properly correlated with varying system properties.

6.3.2 Power consumption

UTIA experience with power consumption of Full HD video processing on ZYNQ is summarized and briefly quantized in this section.

The reference system is described in Figure 30. It is 28nm ZYNQ system with ARM Cortex A9 processor operating on 666 MHz located on TE0720-02-2IF module with 1GByte of DDR3. The carrier board TE701-05 provides support for one FMC card. Toshiba color video sensor is connected via this FMC card and provides color input with Full HD resolution 1920x1080p60. Output of the processing system is connected to the Full HD monitor via HDMI interface on the FMC card. Edge detection algorithm is accelerated in the SDSoC 2015.4 and compiled to the programmable logic part of the ZYNQ together with data movers. The accelerator can be controlled in the runtime by selection of the number of micro-lines to be processed in each Full HD video frame.

UTIA platform provides also the 8xSIMD floating point vector accelerator EdkDSP operating at 120 MHz connected to the 100 MHz MicroBlaze processor. MicroBlaze has program and data in the DDR3 memory shared with the ARM processor. MicroBlaze application code communicates with the EdkDSP accelerator. EdkDSP accelerator performs synchronous SIMD vector atomic operations with data located in the 120 MHz side of dual-ported SRAM memories in the PL part of ZYNQ. MicroBlaze can communicate data to and from the other, 100 MHz side of the dualported memories of the accelerator. The schedule of atomic operations can be dynamically changed by change of the finite state machine (FSM) of the accelerator implemented in 120MHz PicoBlaze processor. Its firmware program can be compiled in UTIA EdkDSP C compiler and changed in the runtime. The demo is working with the floating point FIR filer and the adaptive floating point LMS filter. Filters can be executed on the same EdkDSP accelerator by change of the firmware. See Figure 30.

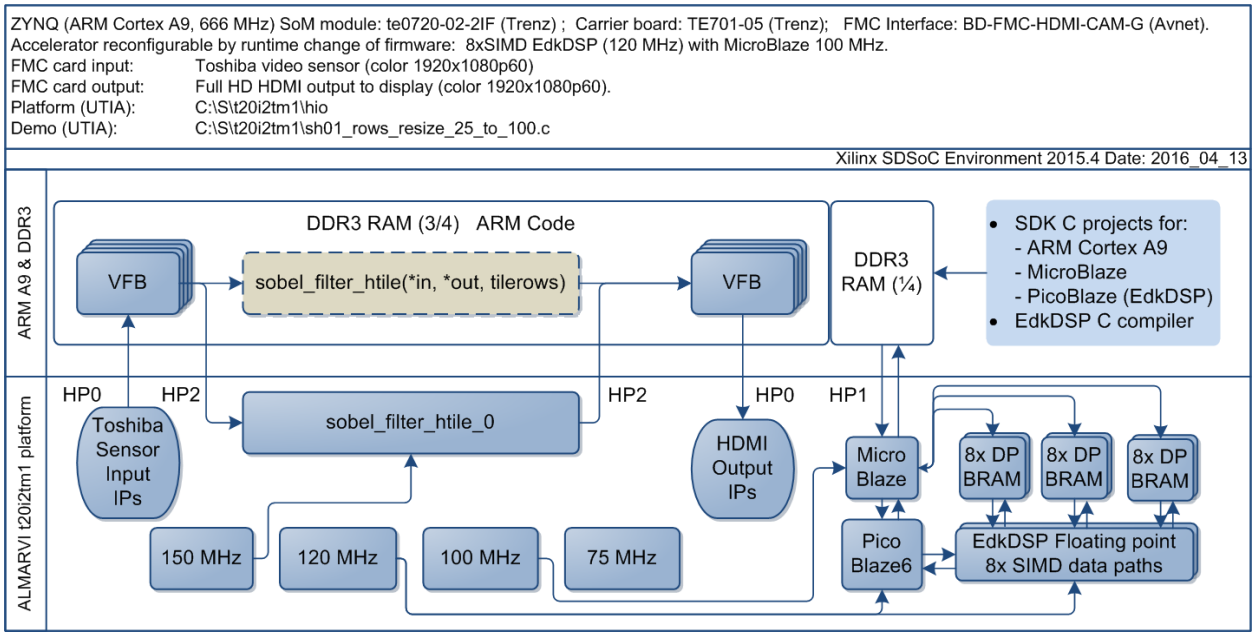


Figure 30: Reference ALMARVI platform with Toshiba color video sensor, edge detection HW accelerator generated by the SDSoC 2015.4 flow and the dynamically reprogrammable 8xSIMD EdkDSP accelerator performing FIR filter and adaptive LMS filter computation in parallel to the edge detection on Full HD video signal.

Power consumption is measured for the complete system presented in Figure 30. It is measured by measurement of the current in the 12V power supply, before all DC2DC regulators. Power is displayed for ARM SW computation in C function sobel_filter_htile() in Figure 31 and for the accelerated HW implementation in Figure 32 in Watts.

The HW accelerator IP core is programmed to process from 270 to 1080 microlines of each frame. This results in changes of the video frame rate which can be achieved by the system.

Case of the vertical processed area with the size from 270 to 810 microlines:

The video processing frame rate is constant (59 FPS). This is limitation enforced by the maximal video frame rate 60 FPS defined by the output video frame buffer DMA HW engine. See Figure 32.

Case of the vertical processed area with the size from 810 to 1080 (Full HD) microlines:

The video processing frame rate is decreasing linearly from 59 FPS to 41,6 FPS. This is due to the time needed to process the edge detection in the selected area. See Figure 32.

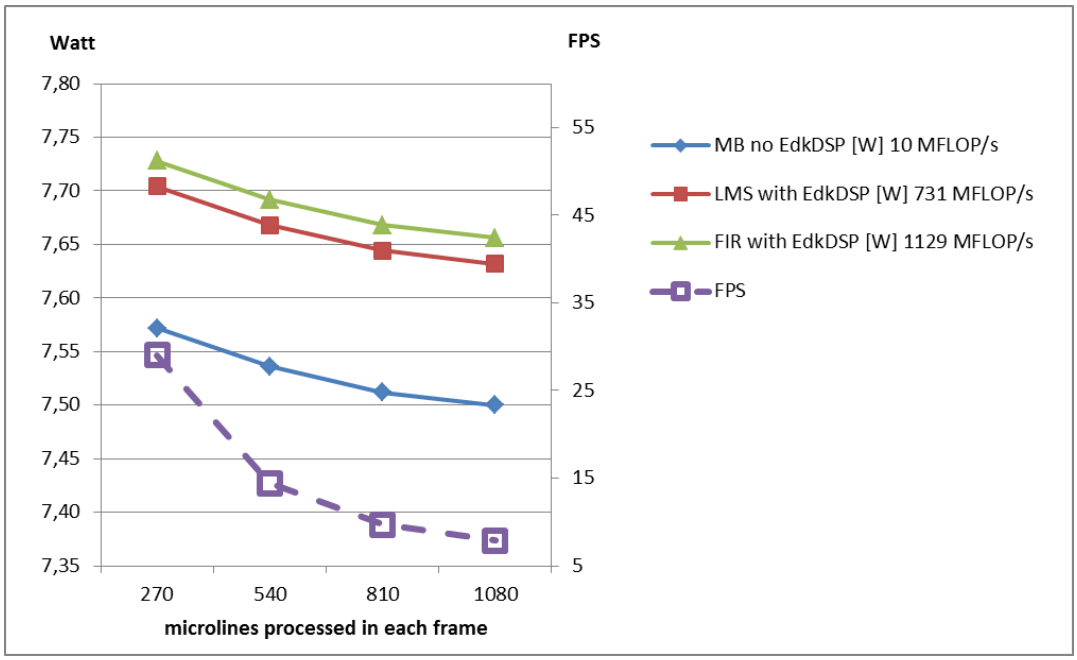


Figure 31: ARM SW computation - power consumption and video frame rate for changing number of processed microlines. Power consumption in case of EdkDSP floating point processing of FIR and adaptive LMS.

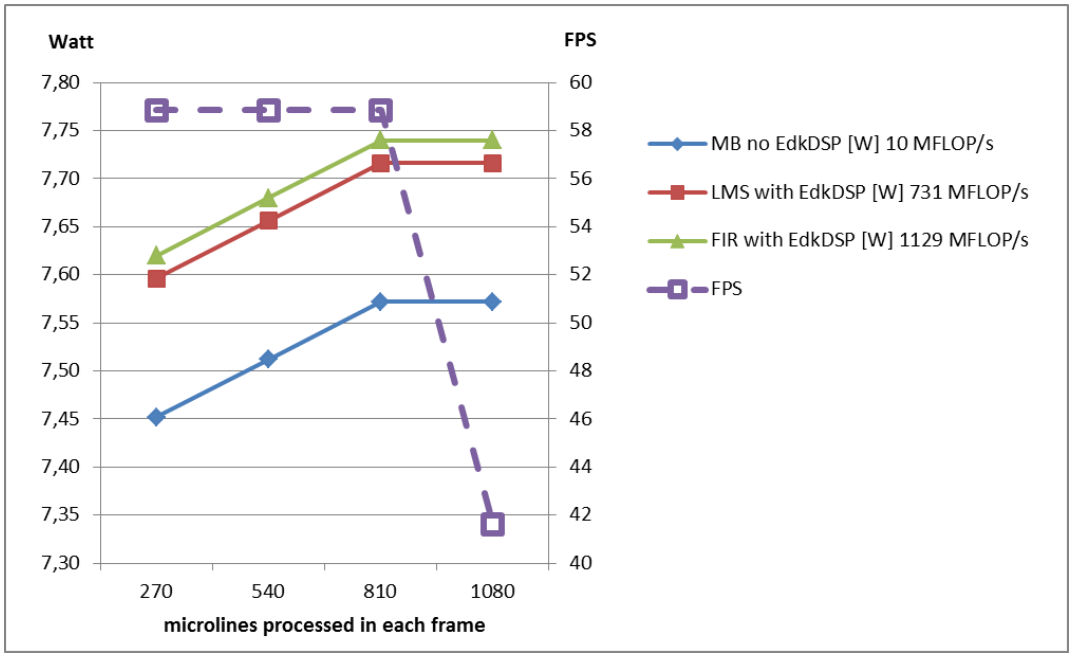


Figure 32: Accelerated HW computation - power consumption and video frame rate for changing number of processed microlines. Power consumption in case of EdkDSP floating point processing of FIR and adaptive LMS.

Figure 31 and Figure 32 are demonstrating only relatively small changes of power consumption of the system with the increasing size of processed area from 270 microlines to 810 microlines. Less data is processed with the constant frame rate (59 FPS). This linear increase of power consumption is stopped for the processed area from

810 microlines to the maximum of 1080 microlines. The increased data transfers per frame are compensated by reduced processing frame rate from 59 FPS to 41,6 FPS. This results in practically constant power consumption. See Figure 32.

Figure 31 and Figure 32 are documenting the increase of the total power consumption in case of parallel computing of LMS filter or the FIR filter on the EdkDSP reprogrammable accelerator or in the MicroBlaze processor in software without the use of the accelerator, without any considerable increase of the total power consumption.

Conclusions

We can conclude that the increase of power consumption due to HW accelerators is relatively small comparing to the total system power consumption measured on real system processing video data from Full HD video sensor.

The processing frame rate for the Full HD has been increased from 7,9 FPS in case of ARM SW to 41,6 FPS in case of the corresponding HW accelerator generated by the Xilinx SDSoC 2015.4 design flow for the Almarvi ZYNQ platform.

This increase of performance is in parallel with the execution of adaptive LMS filter or FIR filter on the reconfigurable floating-point 8xSIMD EdkDSP accelerator:

- The 100 MHz MicroBlaze SW (without EdkDSP accelerator) delivers only 10 MFLOP/s.
- The 120 MHz (8xSIMD) EdkDSP accelerator delivers:
    - 731 MFLOP/s in case of the adaptive LMS filter
    - 1129 MFLOP/s in case of the floating point FIR filter

Increase of power consumption coming from the computation performed in the (8xSIMD) EdkDSP accelerator is +150mW in case of LMS and it is +170mW in case of FIR filter.

Hardware accelerators provide significant reduction of energy per processed frame and energy per processed floating point operation.

The total power of the 28nm ZYNQ system with ARM Cortex A9 processor operating on 666 MHz located on TE0720-02-2IF module with 1GByte of DDR3 stay in the range close to 8W for the Full HD, HW-accelerated video processing of data from the Toshiba color video sensor performed in parallel with the reconfigurable floating point 8xSIMD accelerator computation of adaptive filters.

6.4 Integration

In this section we describe some lessons learned during integration of multiple cameras and hardware components.

6.4.1 Synchronization of camera streams

UTIA team considers interfacing one or two camera inputs for video processing. For the case of two cameras connected into one processing platform, the synchronization of frames must be done. Two possible scenarios were considered:

- locally connected camera sensors, and
- remote camera sensors.

The detailed discussion can be found in D2.6 Multi-Node Camera Data Logical Analysis.

In this section we focus on low-level camera interfacing starting from image capture in video sensor and ending as video data stream inside the FPGA fabric where it can be processed. This component must be implemented as an input to the video processing system at each of two scenarios above. Used hardware configuration is one or two ON Semi Vita2000 CMOS camera sensors (global shutter, 2.3 Megapixel, 60Hz frame rate) with identical optics and FMC IMAGEON extension board (both by AVNET) populated on Xilinx ZC702 development kit. Full HD video signal 1920x1080p60 configuration was implemented in two incremental steps.

Step 1, Parallel cameras & two independent data streams

Figure 33: Two independent video streams

Step 2, Synchronized streams

For two local camera sensors connected using two FMC IMAGEON extension boards, per pixel synchronization of input video data streams can be implemented. For that purpose, the CMOS sensor control core was modified to provide identical trigger and clock signals for both chips to control their shutter and data readout. The camera setup is the same as in the independent case, see Figure 34 left hand side. The difference is the internal synchronization of video signals and video streams from both sensors as shown in ChipScope waveform, see Figure 34 right hand side. The waveform is divided into four groups; first two are fully synchronous video signals from sensors (video signal is defined as video data precisely locked with horizontal and vertical timing). The second pair represents video data streams (video data itself with stripped out synchronization). Only Start of the Frame (SoF) and End of the Line (EoL) flags are transmitted with pixel data. In the figure can be seen, that each pair has identical cycle exact waveform indicating correct synchronization. The problems met in the implementation of both designs were summarized in Table 29.

Table 29: Problems found in camera interfacing designs and their solution

Lessons Learned	Description	Solutions and Workarounds
Flexible cables	Cables are causing irregular problems in initialization of data transmissions	Fixed camera mount, cables firmly attached
Light & Flares	Flares and high ambient light increase probability that Vita2000 CMOS sensor fails initialize properly	Manual and automatic detection of failed initialization and re-initialization option
Failing CMOS Sensor Synchronization (for stereo)	It was observed, that after CMOS initialization, even if all control to sensors is identical the sync operation may fail	Levels of synchronization checks and automatic measures to correct it
Failing stereo video stream synchronization when passing through different clock regions	Even if the video data are already separated from timing and passed to processing stages, the crossing of clock domains using FIFO buffers introduces non deterministic behavior and the system loses synchronization (even if the clock domain on both FIFO ends are identical)	One clock domain for both streams must be implemented or pixels from both sensors must be merged to one data word before conversion to stream.

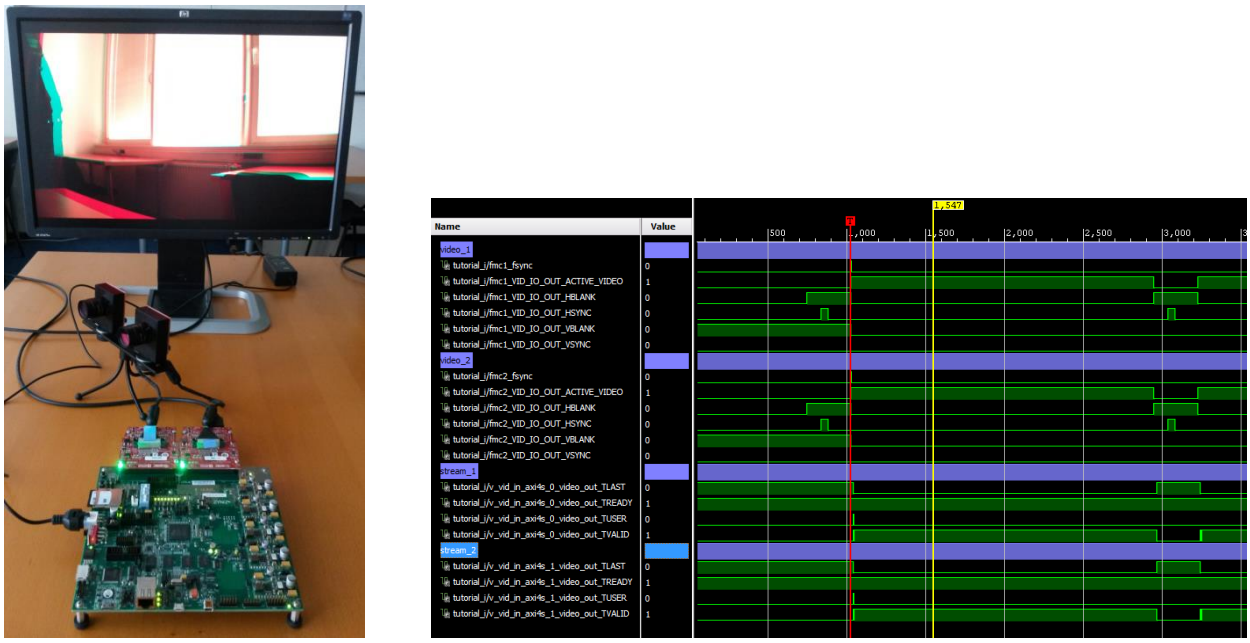


Figure 34: Fully synchronized stereo camera demo (left) and video signal and data stream waveform (right)

Conclusions

The experimental setup interfacing two cameras in perfect sync was implemented from the off the shelf components. We have identified and solved issues with hardware and synchronization, see Table 29. The stereo camera demo which creates red-cyan anaglyph was presented at the 1st review.

6.4.2 Hardware connectors

CAMEA (jointly cooperating with BUT FIT) has built a camera based on Xilinx Zynq SoC. Zynq is a FPGA with dual-core ARM Cortex-A9 processor on chip. The camera is equipped with 5Mpix CMOS Python sensor manufactured by ON Semiconductor as shown in Figure 35.

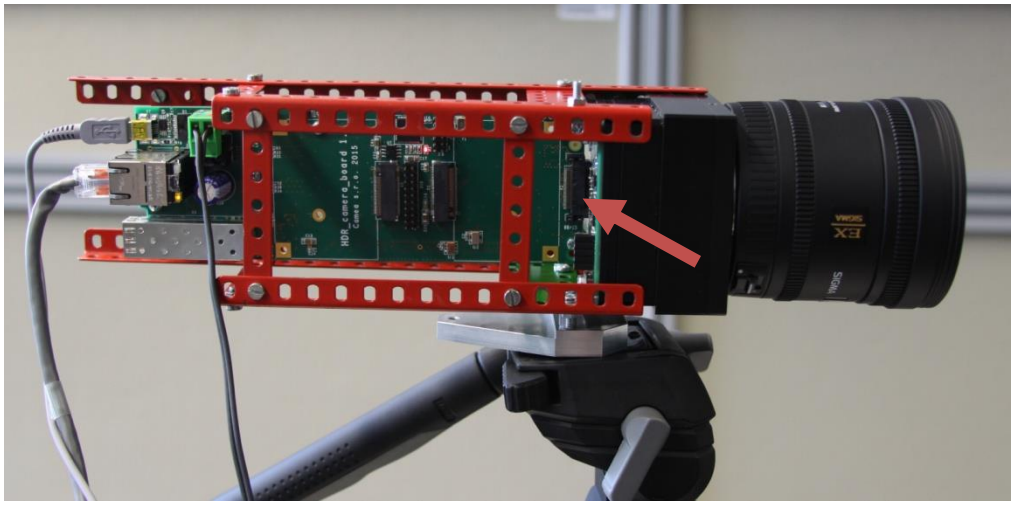


Figure 35: Camera prototype

We have chosen the Samtech angled connector for fitting the PCB with the CMOS sensor to the camera baseboard. We assumed very good mechanical properties along with high frequency interconnection (up to 720MHz).

Several weeks after using the camera we experienced problem with couple data pins on connector, they seemed to be disconnected under some circumstances like touching the objective mount or moving the camera itself etc. We tried to tighten the enclosure, but it was getting the problems worse and worse day by day.

Finally, we discovered that the second row of pins to connector (hidden behind the first row) is not soldered properly. This assumption was confirmed thanks to a specialized company for soldering the BGA packages and connectors and the unsoldered pins appeared under X-ray images. Unfortunately, they were not able to fix the soldering of the connector properly. It turned out we have a defective series of connectors. The second row of pins was much higher than the first one and thus they are unable to reach the PCB and to be soldered (see Figure 36).

After warranty claim at official supplier, we have got the same defective series of connectors again. After another claim, they promised to resolve problem directly with manufacturer, but the manufacturer sent the defective series of connectors again. Subsequently, they officially put them off the market. Unfortunately, there is no alternative from another supplier, the only solution is performing of a redesign of both PCBs.

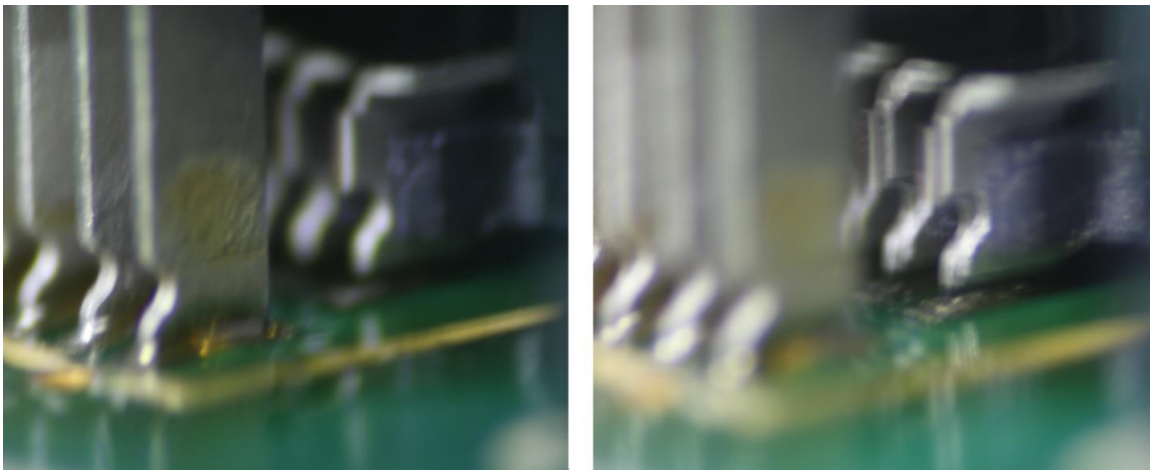


Figure 36: Detail of pins captured by a special micro camera

We temporarily solved the issue with another type of connectors with the same pin pitch and connected the boards with Flex cables. The connectors on CMOS PCB are not designed to be angled by 45° so they "lie down" on surrounding components, but it's more than sufficient for further development (see Figure 37).

Anyway, this experience is valuable at the end as it convinced us to use flex cables on other boards and also on new version of camera. They are not just mechanically favorable due to mechanical separation of the boards, but they also handle well high signal frequencies and the data transfers are absolutely reliable even at 720MHz data rate.

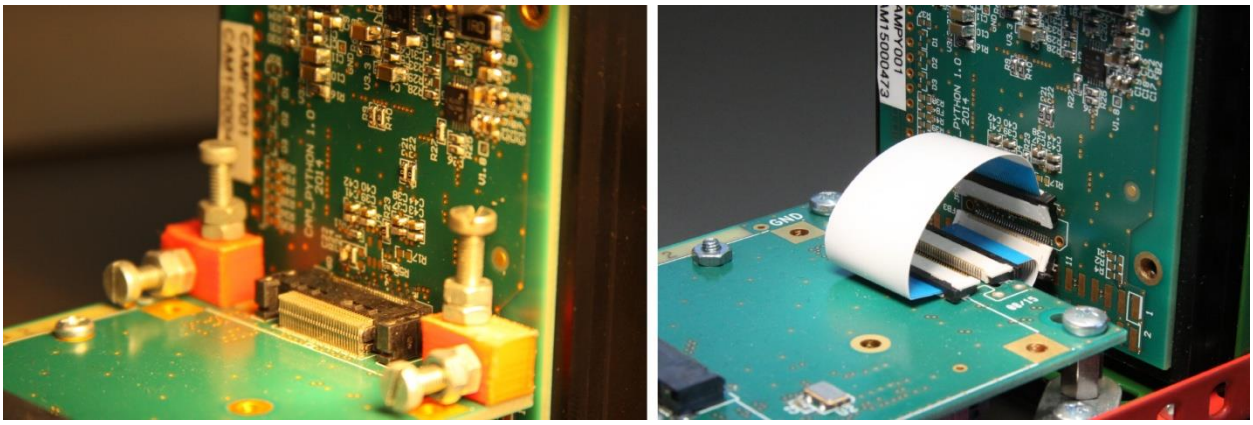


Figure 37: Connectors before (left) and after the fixing (right)

7. Conclusions

This deliverable discussed how the different partners of the ALMARVI project worked to fulfil the overall objective of the ALMARVI project, which is to provide solutions to enable massive date rate image/video processing at low power budgets under variability conditions. The partners achieved this with the integrated and rigorous evaluation of all their use cases for performance and power consumption. In addition, an analysis and evaluation is given on the ALMARVI demonstrators in the different application domains against the overall project objectives.

The various ALMARVI use cases from the three application domains (healthcare, security/ surveillance/ monitoring, and mobile) are considered. To demonstrate and validate the project developments and results, the ALMARVI concepts will be evaluated using demonstrators from the three different application domains. For each domain, the common researched concepts for algorithms, design tools, system software stack, and many-core execution platform are described. The deliverable discussed that the various use cases used many, if not all, the system stack components shown in Figure 38 below, in each of the application domains. At the hardware level, partners compared off-the-shelf hardware solutions to ALMARVI-specific processors, such as the rVEX, the TTA or the edkDSP. At the system abstraction level, partners experimented with portable solutions, such as OpenCL, and compared it to their solution specific abstraction. At the algorithm level, some partners developed their own algorithms to satisfy the needs of their ALMARVI use case.

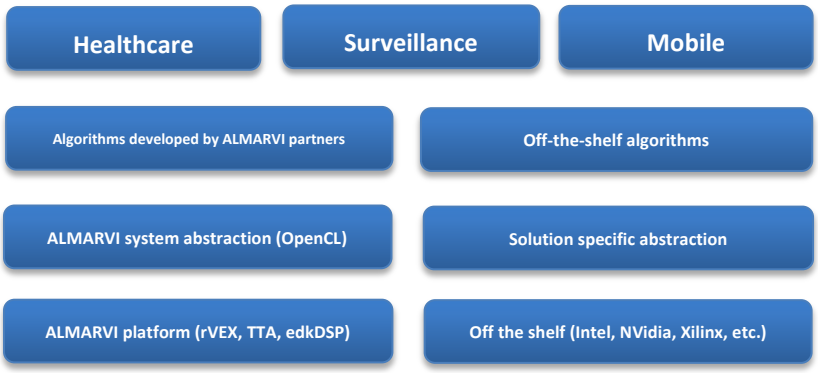


Figure 38: The ALMARVI system stack components in each of the application domains

We also showed how the various demonstrators achieve their target requirements as defined in detail in deliverable D1.1 at the beginning of the project. The investigations of the various partners in the ALMARVI project target the specific objectives of the ALMARVI project: 1. Massive data rate, 2. Low power, 3. Composability, 4. Robustness. Each partner targets specific aspects of these objectives as discussed in this deliverable. The deliverable showed that all project objectives and sub-objectives are addressed by at least one demonstrator in the project.

Most notably, the “cross domain” sub-objective of “Software portability” is covered by all the demonstrators in the project. This is not surprising since this represents one of the cornerstone targets for the ALMARVI project. All partners investigated the potential of porting their software to various HW platforms to ensure the higher performance, lower power consumption or higher cost efficiency of the platforms, Partners experimented with the potential of portable languages, such OpenCL, to help them ensure easy portability between the platforms.

Another notable objective to mention is the “low power” sub-objective of “Algorithm resilience for power efficiency”, which is covered by only one demonstrator: Large Area Surveillance (Aselsan). This is done by investigating the power consumption of various execution platforms as the number of iterations for the surveillance algorithm is modified to achieve the desired functionality with a given power budget. This also indicates that resilience for power efficiency is a difficult objective to achieve in various applications.

In terms of partners, the document showed that smaller industrial partners (such as HURJA) cover the least number of objectives in the project. This understandable due to the limited scope of their demonstrators. The bigger partners, on the other hand, like NOKIA and Aselsan cover the largest number of objectives, due to the elaborate investigations they performed on their demonstrators. One exception is Philips, which covers most objectives, except those related to low power. This can be explained by the limited need for low power for the Philips use case.

8. References

[UEF1] Tiia Ikonen, Keijo Haataja, Pekka Toivanen, Teemu Tolonen, and Jorma Isola: Nuclei Malignancy Analysis Based on an Adaptive Bottom-Hat Filter. *Proceedings of the IEEE 16th International Conference on Intelligent Systems Design and Applications (ISDA'2016),* Porto, Portugal, December 14–16, 2016.

[UEF2] Pekka Toivanen: New Geodesic Distance Transforms for Gray-Scale Images. Pattern Recognition Letters, 17, pp. 437-450, 1996.

[UEF3] Leena Ikonen: Distance Transforms on Gray-Level Surfaces. Ph.D. Thesis, Lappeenranta University of Technology, 2006.

[UEF4] Rong G. and Tan T.: Jump flooding in GPU with applications to Voronoi diagram and distance transform. *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games,* 109-116 pp., 2006.

[UEF5] Caselles Vincent, Kimmel Ron and Sapiro Guillermo: *Geodesic Active Contours, International Journal of Computer Vision,* 22(1), 61-79, 1997.

[UEF6] Osher S.J and Sethian J.A: Fronts Propagation With Curvature Dependent speed: Algorithms Based on Hamilton-Jacobi Formulations. *Journal of Computational Physics,* 79, 12-49, 1988.

[UEF7] Chunming Li, Chenyang Xu, Changfeng Gui and Mardtin D. Fox: Level-set Evolution Without Re-inilization: A New Variational Formulation.*Computer Vision and Pattern Recognition,* 1, 430-436, 2005.

[UEF8] Thiran Jean-Philippe and Macq BenoÎt: Morphological Feature Extraction for the Classification of Digital Images of Cancerous Tissues. *IEEE Transactions on Biomedical Engineering,* Vol. 43, No. 10, pp. 1011-1020, 1996.

[UEF9] Cooper G.M. and Hausman R.E.: *The Cell: A Molecular Approach.* Sinauer Associates Inc., U.S., 2009.

[ASEL1] Werlberger el al., Graz University of Technology (published in BMVC 2009)

[UTIA1] ALMARVI Python Camera Platform, http://sp.utia.cz/index.php?ids=results&id=apcp

[UTIA2] Xilinx HLS Video Library, http://www.wiki.xilinx.com/HLS+Video+Library

[UTIA3] Paul Viola and Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE CVPR, 2001. The paper is available online at http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_CVPR2001.pdf

[UTIA4] Rainer Lienhart and Jochen Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP 2002, Vol. 1, pp. 900-903, Sep. 2002. This paper, as well as the extended technical report, can be retrieved at http://www.multimedia-computing.de/mediawiki//images/5/52/MRL-TR-May02-revised-Dec02.pdf

[UTIA5] UTIA Web GUI demo for KC705, http://sp.utia.cz/results/Utia_EdkDSP_Vivado_2013_4_KC705/Utia_EdkDSP_Vivado_2013_4_KC705.pdf

[UTIA6] Downloadable UTIA demos, http://sp.utia.cz/index.php?ids=results

[UTIA7] SDK 2015.4 Projects for Evaluation of HW Accelerated Video Processing with Python 1300 Sensor and (8xSIMD) EdkDSP Accelerator on TE0720-03-2IF Module and TE0701-06 Carrier, Jiří Kadlec, Zdeněk Pohl, Lukáš Kohout, http://sp.utia.cz/results/t20i2pm4/t20i2pm4_2015_4_te0701_06.pdf, UTIA, 2017

[OCV1] SDK OpenCV example, http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/SDK_Doc/tasks/sdk_t_opencv_template.htm

[OCV2] S. Neuendorffer, T. Li, and D. Wang, "Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries", XAPP1167, http://www.xilinx.com/support/documentation/application_notes/xapp1167.pdf

[OCV3] HLS Video Library, http://www.wiki.xilinx.com/HLS+Video+Library

[OCV4] SDSoC designs and platforms, online, http://www.wiki.xilinx.com/SDSoC+designs+and+platforms